MINISTERUL EDUCAȚIEI, CERCETĂRII, TINERETULUI ȘI SPORTULUI

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Salanta Andrei

M-shop virtual bazat pe proximitate și geolocalizare

LUCRARE DE LICENȚĂ

Absolvent: **Marius Andrei SALANȚA**

Coordonator ştiinţific: **s.l. ing. Cosmina IVAN**

2013

MINISTERUL EDUCAȚIEI, CERCETĂRII, TINERETULUI ȘI SPORTULUI

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

VIZAT,

DECAN,

Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,

Prof. dr. ing. Rodica POTOLEA

Absolvent: **Marius Andrei SALANȚA**

M-shop virtual bazat pe proximitate și geolocalizare

1. **Enunţul temei:** *Aplicaţia implementată permite localizarea utilizatorului şi determinarea magazinelor din proximitatea acestuia, afişarea produselor de la magazinele favorite,o lista de cumpărături , un meniu personalizabil precum si determinarea celui mai scurt drum de la poziţia utilizatorului la cel mai apropiat magazin.*

2. **Conţinutul lucrării:** *Introducere, Obiectivele proiectului, Studiu Bibliografic, Analiză și fundamentare teoretică, Proiectare de detaliu şi implementare, Testare şi validare, Manual de instalare și utilizare, Concluzii.*

3. **Locul documentării:** Internet, Universitatea Tehnică din Cluj, catedra de Calculatoare

4. **Consultanţi:** s.l. ing. Cosmina IVAN

5. **Data emiterii temei:** 1 februarie 2013

6. **Data predării:**12 septembrie 2013

Absolvent: _____

Coordonator ştiinţific: _____

Cuprins

Capitolul 1. Introducere

1.1.Contextul proiectului

Aparatele de telefonie mobilă au cunoscut în ultima perioadă una dintre cele mai mari evoluții ale aparatelor electronice. Istoria telefonului mobil începe în anul 1910, când inventatorul suedez Lars Magnus Ericsson pune în practică conceptul de telefonie mobilă, instalându-și un astfel de dispozitiv în mașina sa, iar prin intermediul unei antene reușește să se conecteze la rețeaua de telefonie fixă națională în timp ce se deplasa prin țară[1].

În anul 1973 Martin Cooper de la compania Motorola, efectuează cu telefonul Motorola DynaTAC prima convorbire cu Joel Engel de la compania AT&T BELL Labs. Acest dispozitiv era un sistem analogic sși face parte din Generația 1 a tehnologiilor telefoanelor mobile. Pentru Generația 2 se foloseau sisteme mobile digitale, aceasta fiind urmată de Generația 2 ½ care folosea o tehnologie bazată pe sisteme radio digitale bazate pe pachete. A treia generație a apărut în anul 2001 în Japonia , fiind introdusă de compania NTT DoCoMo, care folosea un sistem cu lungime de bandă msare, care creștea viteza de transmisie, capacitatea rețelei și calitatea serviciilor oferite. A patra generație de telefoane mobile a apărat în anul 2012 fiind numita "4G", în care transmisiile sunt executsate în modul "streaming media", prin care informația multimedia este transmisă promp și cu fidelitate[2].

Tehnologia telefoniei mobile se află astăzi în plină evoluție. Aceasta constă atât în creșterea numărului de abonați și de posesori de telefoane, cât și în creșterea numărului de servicii și opțiuni realizabile cu ajutorul telefonului, bazate pe Internet și GPS, cum ar fi:ghidaj pentru pietoni (de ex. Găsirea unei rute pentru a ajunge la o destinație dorită), informații locale (de ex. Localizarea celui mai apropiat hotel), sistem de plata, modalitate de acces securizat la informații sensibile ale utilizatorului.

Un telefon inteligent (din engl., „smartphone") este un telefon mobil multifuncțional, care dispune de o tastatură reală sau virtuală și care oferă funcționalități de agenda, calendar, navigare, e-mail, calculator, aparat foto sau aparat de filmat. Primul smartphone a fost realizat de compania americana IBM și a fost numit *Simon*[3].

Una dintre cele mai folosite strategiile de marketing folosite de firmele care produc telefoane inteligente este cea orientată spre client. Astfel procesul de proiectare a unui telefon inteligent presupune un studiu de piata intensiv, precum și imbunătățirea caracteristicilor telefoanelor mobile. Astfel de-a lungul timpului modalitatea de prezentare vizuală a telefoanelor s-a schimbat drastic.Principalele caracteristici ale noii generații de telefoane mobile sunt:

- Dimensiunea redusă: primele telefoane avea dimensiuni de ordinul zecilor de centimetri, în prezenta lungimea și lățimea sunt mai mici decât 10 centrimetri , iar grosimea este de ordinul milimetrilor.
- Comoditatea: designul telefoanelor mobile moderne sunt ergonomice și destinate a fi folosite cu o singură mână.
- Localizarea: cererea de informații bazate pe localizare a crescut în ultima perioadă, iar telefoanele de generatia a treia au introdus acestă capabilitate.

[1]Paul Goransson, Raymond Greenlaw, *Secure roaming în 802.11 networks. Newnes, 31 mai 2007, pg. 16*
[2]http://ro.wikipedia.org/wiki/Istoria_telefonului_mobil#cite_note-1 , ultima accesare 1 august 2013
[3]http://en.wikipedia.org/wiki/Smartphone , ultima accesare 1 august 2013

- Securitatea: aplicațiile pot folosi date de logare, fiind necesarăasigurarea securității aplicațiilor prin transmiterea datelor prin protocoale securizate sau prin cripatarea datelor înainte de transmitere.
- Personalizarea: Întrucat timpul alocat de clienți utilizării telefonului este din ce în ce mai mare, personalizarea conținutului este importantă pentru fidelizarea utilizatorilor.

Majoritatea telefoanelor inteligente prezente pe piața fac parte din generatia 3G. În conținuare vor fi prezentate principalele tipuri de servicii oferite de sistemele 3G:

- servicii de transmisie vocală: transfer voce, conferințe, poștă vocală
- servicii de mesagerie simplă: serviciu de mesaje scurte
- servicii de internet de mare viteză: acces rapid la rețelele locale și internet/intranet, cumpărături online, jocuri interactive, video-streaming, mesaje audio
- servicii multimedia interactive: teleprezența, videoconferințe

Aplicațiile desktop sunt incă cele mai utilizate aplicații din domeniu dar există o tendință reală înspre migrarea tuturor serviviilor spre internet, spre dispozitivele mobile și spre sistemele de tip 'cloud'.

1.2.Domeniul temei

În contextul dezvoltării tehnologice, adesea apar noi oportunități și noi cereri din partea utilizatorului. Tehnologia face posibil ca utilizatorii să acceseze resurse într-un mod ușor și comod. Proiectul prezentat îndeplineste cererea utilizatorului și oferă ocazia de a găsi ușor, rapid și confortabil, cele mai bune produseale distribuitorilor și producătorilor de alimente sănatoase, a unui meniu zilnic care poate fi personalizat, a unei hărti unde se găsesc magazinele din proximitatea utilizatorului, precum și a unei liste de cumpărături care se poate autopopula.

Stilul de viață din ce în ce mai sedentar, obligă oamenii să mănânce mai sanatos. În multe din țările europene, precum și de pe conținentele americane, s-au lansat campanii naționale în ceea ce privește alimentația sănătoasă, precum și un stil de viață sănătos. Acest proiect dorește a veni în sprijinul utilizatorilor săi, prin a oferi informații localizate și personalizate pentru alimentația lor.

Conform studiul Health At A Glance Europe 2012[4], realizat de OECD (Organizaţia Economică pentru Cooperare şi Dezvoltare) şi Comisia Europeană aproape 17 % din populația Europei, se confrunta cu obezitatea. Deși Romania este țara cu cea mai mică rată a obezității din Europa (8%), un număr din ce în ce mai mare de oameni se îndreapta spre diete vegetariene și crude. Produsele prezentate vor fi marcate cu indicatori vizuali pentru a atrage atenția asupra numărului de calorii.

În funcție de poziția geografică a utilizatorului, se va prezenta o listă de magazine din proximitatea acestuia și pentru fiecare magazin, se vor afișa cele mai bune produse.

Prin oferirea unui meniu pentru fiecare zi, și ingredientele necesare pentru prepararea acestuia, a magazinului unde se pot găsi, precum și adăugarea ingredientelor automată în lista de cumpărături, poate stimula utilizator să își formeze o dietă mai sănătoasa prin folosirea aceastei aplicații.

[4]Organizaţia Economică pentru Cooperare şi Dezvoltare, Comisia Europeana, http://www.oecd.org/els/health-systems/HealthAtAGlanceEurope2012.pdf ultima accesare 1 august 2013

Un astfel de sistem, pe lângă faptul ca ar putea oferi un stil de viata mai sănătos, ar putea reduce timpul petrecut în magazin pentru căutarea alimentelor și ar putea reduce tentația utilizatorului, când se afla în magazin, de a cumpara alimente nesănătoase.

În Statele Unite ale Americii există deja câteva proiecte în curs de dezvoltare care oferă aceași gama de servicii,unul din cele mai mari fiind Zipongo[5], care și-a început activitatea în 2010 și are în prezent un număr impresionant de utilizatori, atât pe site cât și pe aplicațiile mobile.

În România nu există o implmentare a unui astfel de sistem. Fiind un proiect cu caracter practic și de actualitate, se pretează a fi folosit atât de utilizatorii simpli cât și de companii, care vor ca angajații lor să aibă un stil de viață sănătos, pentru o productivitate mai mare. Deși există campanii media care susțin ideea unei vieți sănătoase, un astfel de sistem practic, nu este implementat, nu numai în România, cât nici în multe din țările dezvoltate din lume.

O altă utilitate pe care ar putea sa o aibă un astfel de sistem ar fi crearea și implementarea de soluții adaptabile fiecărui lanț de magazine sau fiecărui producător, pentru promovarea produselor proprii. În acest caz sistemul ar căpata un caracter publicitar și de promovare. Mai mult, se poate implementa un sistem de cumpărare online, astfel, timpul petrecut în magazin devine mult mai mic. Totodată o legătură cu o firmă de catering, ar putea ajuta utilizatorul și mai mult.

1.3.Conținutul lucrării

Primul capitol cuprinde introducerea cititorului în contextual dezvoltării proiectului, în ceea ce privește atât partea tehnologică, cât și partea practică. Este prezentată istoria telefonului mobil și domeniul temei, un stil de viață sănătos.

În cel de-al doilea capitol, sunt prezentate obiectivele proiectului si specificațiile acestuia, prin enumerarea cerințelor funcționale și non-funcționale.

Capitolul al treilea prezintă studiul biblografic efectuat pentru proiectarea sistemului și trei aplicații similare sistemului.

În capitolul patru sunt prezentate conceptele teoretice folosite atât în aplicația server cât și în aplicația client, precum și avantajele folosirii acestora. În a doua parte a acestui capitol, este prezenatată analiza sistemului, prin diagramele de cazurilor de utilizare, schema bloc a sistemului și arhitectura sa conceptuală.

În capitolul cinci este prezentată proiectarea de detaliu a sistemului fiind exemplificate structura și diagramele de clase ale aplicațiilor client și server. În subcapitolul „Implementare”, sunt prezentate clasele și componentele celor două aplicații și interfața utilizator a aplicației client.

În capitolul șase sunt prezentate metodele de testare și validare a întregului sistem.

Capitolul șapte cuprinde specificațiile de instalare și manualul de utilizare a aplicației client.

În capitolul opt, sunt prezentate concluziile implementării acestui proiect, precum și posibilele dezvoltări ulterioare ale acestuia.

În anexele unu, doi si trei, sunt prezentate listele care cuprind numele tuturor figurilor din această lucrare, acronimele folosite si referințele bibliografice.

[5] http://techcrunch.com/2012/07/25/zipongo-seed-round/ ultima accesare 1 august 2013

Capitolul 2. Obiectivele proiectului

2.1.Obiective generale

Se dorește proiectarea și realizarea unei aplicații mobile destinată dispozitivelor mobile Apple care să ofere o soluție rapidă și ușor de folosit utilizatorilor care doresc să aibă un stil de viață sănătos.

Caracterul practic și de actualitate al proiectului, precum și utilitatea acestuia necesită un sistem, robust, rapid, fiabil și care să funcționeze și în mod offline. Din ce în ce mai multe persoane sunt interesate de acest domeniu, așadar principalele caracteristici pe care le oferă acest sistem, sunt: listarea magazinelor din proximitatea utilizatorului, afișarea celor mai bune produse a acestor magazine, un program care oferă un meniu zilnic și care poate fi personalizat, precum și o listă de cumpărături care oferă toate caracteristicile unui liste de cumpărături, și în plus se poate popula automat cu ingredientele necesare acestui meniu.

Acest sistem trebuie să ofere o interfață cu utilizatorul prietenoasă și intuitivă, o viteză mare, caracteristici ale produselor categorizabile. Bazat pe aceste obiective, întreg sistemul este distribuit în patru module, două proprii: aplicația server, aplicația client și două auxiliare: sistemul Urban Airship și sistemul APNs.

Astfel, proiectarea și implementarea celor două sisteme proprii, precum și integrarea lor cu celelate două sisteme auxiliare, reprezintă un alt obiectiv al acestui proiect. Aceste module integrate vor asigura funcționarea sistemului în conformitate cu cerințele. În conținuare vor fi prezentate succint cele patru module:

- **Aplicația server**: este un sistem care pe baza locației utilizatorului, calculează care sunt cele mai apropiate magazine și în functie de rezultat returnează cele mai bune produse ale acestora. Totodată, meniul zilnic oferit utilizatorului, precum și preferințele acestuia vor fi păstrate într-o bază de date, care poate fi utilizată ca și o sursă comună de informații, în cazul extinderii sistemului cu mai multe aplicații client.
- **Aplicația client**: reprezintă, printr-o interfața prietenoasă, o platformă pentru afișarea datelor locale, cât și pentru sincronizarea preferințelor utilizatorului cu serverul. Este o aplicație client a sitemului de notificări APNs. Poate una dintre cele mai importante cerințe ale aplicație client, este determinarea pozitiei utilizatorului și transmiterea acesteia la server, prin serviciile web implementate de sistemul server.
- Sistemul**Urban Airship**:este un sistem ajutător care asigura transmiterea notificărilor de la server la aplicația client, prin intermediul sistemului APNs. Este o soluție convenabilă, întrucât implmentarea unui sistem de transmiterea a notificărilor la serviciul APNs este consumatoare de timp.
- Sistemul**APNs**: este un serviciu oferit de compania Apple, care înaintează notificările primite de la sistemul Urban Airship către toate dispozitivele înregistrate. Acest sistem are capabilitatea de a retransmite notificările, în cazul în care aceastea nu au fost primite.

2.2.Specificația proiectului

Caracteristicile și cerințele proiectului vor fi specificate în cele ce urmează prin gruparea lor în **cerințe funcționale** și **non-funcționale**.

2.2.1.Cerințe funcționale

Cerințele funcționale definesc funcțiile pe care trebuie să le îndeplinească sistemul, precum și comportamentul acestuia în diferite situații. Cerințele de comportament a proiectului sunt descrise în diagramele de utilizare. Cerințele funcționale descriu intrările pe care trebuie să le accepte sistemul, ieșirile produse de acesta, datele folosite de sistem (precum și modul lor de stocare), calculele ce trebuie efectuate de sistem și sincronizarea datelor și a serviciilor.

După cum s-a specificat și în obiectivele proiectului, aplicația este destinată utilizatorilor care vor sa aibă un stil de viață mai sănătos și care are ca și cerințe funcționale principale, localizarea utilizatorului, prezentarea unei liste de magazine pe baza locației, listarea celor mai bune produse ale producătorilor și distribuitorilor, un meniu zilnic personalizabil, prezentarea unei hărți cu magazinele din proximitatea utilizatorului precum și o listă de cumpărături, care permite adăugarea automată a ingredientelor necesare meniului.

Ţinând cont de împarţirea întregului sistem în patru module: aplicaţia server, aplicaţia client, sistemul Urban Airship şi serviciul APNs, în conţinuare vor fi detaliate cerinţele funcţionale ale fiecărui subsistem.

Aplicaţia client este cea mai importantă şi cea mai semnificativă componetă a întregului sistem, fiind singura cu care utilizatorul interacţionează direct. Această componentă este o aplicaţie destinată dispozitivelor mobile Apple, fiind o aplicaţie nativă, care funcţionează doar pe acest tip de dispozitive.Există, astfel, o serie de cerinţe funcţionale pe care aceasta aplicaţie trebuie să le indeplinească.

Întrucât aplicaţia client, comunică cu aplicaţia server prin Internet şi există metode care utilizează date sensibile al utilizatorului, cominicarea dintre aceste două module trebuie să fie securizată, astfel se va folosi protocolul Https. Pentru a evita supraîncarcarea serverului cu cereri neautorizate, fiecare cerere de date iniţiată de aplicaţia client trebuie semnată. Semnatura se va face utilizand o cheie secretă, cunoscută atât de aplicaţia server cât şi de aplicaţia client, care, împreună cu data curentă şi url-ul cererii, va fi criptata utilizând algoritmul SHA-HMAC.

Principalele cerințe ale **aplicaţiei client** sunt enumerate în cele ce urmează, grupate după principalele funcții care le indeplinesc:

- Lista de produse:
  - Afişarea celor mai bune produse ale distribuitorilor în funcţie de magazinele selectate
  - Gruparea produselor în funcție de categoriile alimentelor
  - Gruparea produselor în funcție de magazine
  - Adăugarea sau ștergerea produselor din lista de cumpărături
- Lista de magazine
  - Localizarea utilizatorului
  - Afişarea unei liste de magazine din proximitatea utilizatorului
  - Selectarea/deselectarea magazinelor dorite
  - Afişarea magazinelor pe hartă
  - Calcularea celei mai scurte rute pâna la un magazin selectat
- Lista de cumpărături
  - Afișarea listei de cumpărături
  - Ştergerea sau marcarea ca şi cumpărate a alimentelor din lista de cumpărături
- Lista cu elementele meniului
  - Afişarea unui meniu zilnic, cu posibilitatea de personalizare a acestuia

    - Adăugarea automată a ingredientelor din meniu în lista de cumpărături
- Autentificarea utilizatorului
- Persistenţa datelor

Pentru a îndeplini cerinţele enumerate mai sus, **aplicația client**, pe lângă operaţiile interne, trebuie să comunice cu aplicaţia server, cu sistemul Urban Airship şi sistemul APNs. Aplicaţia client interacţionează cu aplicaţia server în două moduri: cerere de date şi trimitere de date. Cererea de date este utilizată pentru obţinerea listelor de magazine, produse şi pentru meniul zilei. Trimiterea de date se foloseşte pentru transmiterea magazinelor selectate, a alimentelor adăugate în lista de cumpărături şi pentru ştergerea intrărilor din lista de cumpărături. Comunicare cu sistemul Urban Airship se face pentru a înregistra fiecare aparat de telefonie în baza de date, printr-un identificator unic al dispozitivului, pentru a putea face posibilă transmiterea notificărilor de la aplicaţia server la dispozitivele mobile. În ceea ce priveşte comunicarea cu serviciul APNs, şi aici se trimite o cerere de înregistrare cu un identificator unic, generat automat de sistemul de operare. Acest identificator este calculat pe baza dispozitivului şi a aplicaţie.

Aplicaţia server, are rolul de a prelua cererile aplicaţiei client, şi de a furniza date prelucrate din baza de date. Astfel, aplicaţia server are următoarele cerințe:

- Autentificarea utilizatorului în sistem
- Verificarea veridicităţii cererilor adresate de alte sisteme
- Transmiterea listei de magazine, celei mai apropiate de locaţia utilizatorului pe o rază data
- Modificarea în baza de date a preferintelor utilizatorului
- Adăugarea / ştergerea ingredientelor din lista de cumpărături
- Transmiterea listei de produse şi ingrediente
- Transmiterea meniului zilnic
- Personalizarea prin adăugarea sau ştergerea elementelor în/din meniu
- Trimitere notificărilor când se introduc produse noi în baza de date
- Găzduirea unui baze de date

Pentru a îndeplini aceste cerinţe, precum şi cerinţele globale ale sitemului, aplicaţia server comunică cu aplicaţia client şi cu sistemul UA. Comunicarea cu aplicaţia client se rezumă la a primi cereri de la aceasta şi de a modifica sau oferi date din baza de date. Comunicare cu sitemul UA, se face prin trimiterea unor cereri de transmiterea a notificărilor, care mai apoi vor fi trimise către serviciul APNs şi mai apoi spre dispozitivele mobile.

Sistemul Urban Airship, este un sistem terţ, care crează notificări în urma cererilor primite de la aplicaţia server şi le trimite mai departe la serviciul APNs. În ceea ce priveşte comunicare cu aplicaţia client, acest sistem primeste ca şi date de intrare, un identificator unic al fiecarui dispozitiv mobil, pe baza căruia serviciul APNs ştie către ce aparate să inainteze notificările. Astfel, principalele caracteristici ale acestui sistem sunt:

- Înregistrarea identificatorilor unici ai dispozitivelor mobile
- Primirea cererilor de transmitere de notificari de la aplicaţia server
- Transmiterea notificărilor către serviciul APNs

Sistemul APNs, este un serviciu oferit de compania Apple, care înaintează notificările spre dispozitivele mobile care au instalată aplicaţia client. Un avantaj al acestui sistem, este că înregistrează dispozitivele care au primit notificarea, iar celor care din diverse motive nu au putut primit notificarea (lipsă internet, lipsă semnal, dispozitiv închis), le va trimite din nou în

momentul în care este sesizată disponibilitatea dispozitivului. Acest sistem primeste cererile de inaintarea a notificărilor de la sistemul UA.

În concluzie, funcţionarea corectă a fiecărui sistem în parte, împreună cu procesul de comunicare dintre ele, determină funcţionarea corectă a întregului sistem şi indeplinirea tutoror cerinţelor funcţionale.

2.2.2.Cerinţe non-funcţionale

Cerinţele non-funcţionale sunt cerinţe care specifică criterii generale de utilizare şi testare a unui sistem. Aceste cerinţe completează cerinţele funcţionale ale proiectului.

Astfel am identificat următoarele cerinţe non-funcţionale:

- **Accesibilitate:** Cererile utilizatorilor trebuie să fie îndeplinite în orice moment. În caz contrar ,indisponibilitatea sistemului va duce la pierderea clienţilor.
- **Disponibilitate:** După descărcarea datelor de pe server acestea sunt stocate local pe dispozitive fiind persistente, astfel încât, în cazul în care serverul nu funcţionează pentru o perioadă de timp, informaţiile pot fi vizionate de utilizatori. Acest lucrueste valabil şi în cazul în care utilizatorul se află într-o zonă fara acoperire de semnal.
- **Fiabilitate**: Fiabilitatea este reprezentată de probabilitatea ca un sistem software să funcţioneze fără erori pentru o perioadă de timp data, într-un mediu dat. Întrucât pe piaţa produselor software mai există aplicaţii asemănătoare, lipsa fiabilităţii va duce la dispariţia produsului de pe piaţa, lucru care ar avea consecinţe nefaste pentru proprietarul aplicaţiei.
- **Utilizare uşoară**: Aplicaţia fiind destinata unui număr mare de utilizatori, factorul uman reprezintăun rol important, astfel interfaţa grafică este proiectată să respecte standardele specifice utilizatorilor de dispozitive Apple. Ecranele şi elementele grafice utilizate sunt intuitive şi predictibile. Pentru încărcarea datelor din aplicație se foloseste metoda de încărcare leneşă, astfel încat utilizatorul nu trebuie să aştepte încărcarea altor informaţii decât cele pe care le vede. Având în vedere că aplicaţia comunică cu un alt sistem software (server care oferă date), utilizabilitatea uşoara se reflectă în uşurinţa de comunicare a sistemului şi de adaptare a acestuia la mediul software.
- **Concurenţa**: Sistemul (în special partea de server) trebuie să fie flexibil şi să se adapteze unui număr mare de utilizatori
- **Utilizarea eficientă a resurselor**: Tinând cont de resursele hardware limitate ale dispozitivelor mobile, aplicaţia trebuie să utilizeze optim resursele de memorie pentru a evita funcţionarea înceată sau chiar închiderea automată a aplicaţiei.
- **Securitatea**: Deoarece aplicația client trimite date personale la aplicația server este nevoie de una sau mai multe soluții de securitate, care să facă procesul de comunicare sigur.

Capitolul 3. Studiu bibliografic

În articolul 1, este prezentată istoria telefonului mobil, în principiu apariția acestuia și aportul adus inventatorul său, Lars Magnus Ericsson.

În referințele 2 și 3, sunt prezentate evoluția dispozitivelor telefonice mobile, punându-se accentul pe contextul și evoluția telefoanelor mobile inteligente (din engl. Smartphone).

În studiul 4 este prezentat un articol despre starea sănătății europenilor, studiul realizat de OECD (Organizația Economică de Cooperare și Dezvoltare).

În articolul 5, este prezentată ideea principală a suitei de aplicații Zipongo, aplicație similară proiectului de licență.

În referințele 6,7,8,9 și 10, sunt prezentate alte aplicații similare.

În articolele din 11 și 12, sunt prezentate ideile și principiile teoretice ale tehnologiei Cocoa Touch, iar în articolul 13 sunt prezentate particularități ale acestei colectii de clase.

În referința 14, este prezentat mediul de dezvoltare XCode.

În articolul 15, sunt prezentate atât ideile de bază ale framework-ului Core Data cât și particularitățile acestuia.

În articolul 16, sunt prezentate diferențele dintre Core Data și o baza de date.

În referințele 17 și 19 este prezentat serviciul de notificări oferit de compania Apple, iar în articolul 18 sunt prezentate principiile de funcționare ale notificărilor, precum și detalii despre acestea.

În articolele 20 și 21 sunt prezentate informații generice despre limbajul de programare PHP, iar în referințele 22,24 și 25 sunt prezentate informații despre framework-ul Zend.

În referințele 26 și 27 este prezentat mediul de dezvoltare NetBeans.

În articolele 28, 29 și 30 sunt prezentate standardul JSON și platforma pentru baze de date MySql.

În articolul 31 este prezentat sablonul de proiectare Modal View Controller, iar în 32 și 33 sunt prezentate conceptele de protocoale și sablonul arhitectural de delegare.

În articolul 34 este prezentată distributia versiunilor sistemului de operare iOS.

3.1.Programe similare

3.1.1.Zipongo

Zipongo este un proiect cu dezvoltarea începută în Silicon Valey, care urmăreşte să permită o alimentaţie sănătoasă prin asocierea unei platforme concepute după câteva studii comportamentale, cu reduceri la alimentele prezente pe site[6]. Acest proiect, are legături formate cu principalele lanţuri de magazine din Statele Unite ale Americii[7] şi oferă o „piaţă”

[6] http://www.forbes.com/sites/davidshaywitz/2013/02/21/rock-healths-new-class-daring-to-be-useful/ , ultima accesare 28 iunie 2013

[7] http://venturebeat.com/2013/02/20/rock-health-startups-whipping-healthcare-industry-into-shape/, ultima accesare 28 iunie 2013

virtuală care oferă utilizatorilor reţete, reduceri la alimente şi meniuri sănătoase, bazate pe nevoile utilizatorului, pe stilul său de viaţa şi pe preferinţele acestuia[8].

Printre avantajele acestui sistem, se numără asociarea cu cele mai imporante lanţuri de magazine din Statele Unite ale Americii, personalizarea meniurilor şi a produselor în funcţie de preferinţele utilizatorului şi oferta de promoţii pentru alimentele cumpărate prin intermediul acestui sistem, format din site, şi aplicaţii mobile pentru cele trei mari platforme: iOS, Android şi BlackBerry[9].

Ca şi dezavantaj, sau posibilă implementare ulterioră, este lipsa unei hărţi pe care se să afişeze poziţia utilizatorului şi poziţiile celor mai apropiate magazine.

Sistemul Zipongo, fiind una dintre cele mai complexe aplicații similare a avut un rol important în stabilirea principalelor functionalități ale sistemului.

3.1.2.Fooducate

Fooducate este o platformă cu aplicaţii pentru web, iOS şi Android, formată dintr-un colectiv de deieteticieni şi programatori, care oferă informaţii despre cele mai multe mărci de alimente din principalele lanţuri de magazine din Statele Unite ale Americii.

Cea mai importantă funcţionalitate a acestei platforme este scanarea prin intermediul aplicaţiilor mobile a codurilor de bară de pe fiecare produs, şi afişarea automată a ingredientelor şi compararea lor cu standardele presupuse pentru o dieta sănătoasă[10]. Un alt avanjat al acestei aplicaţii este acordarea de calificative alimentelor şi oferirea unor alternative mai sănătoase.

Printre dezavantajele acestei aplicații se numără, lispa unei liste de cumpărături pre programabile sau oferta unor reţete.

Avantajele acestei aplicații ar putea fi preluate si integrate și în sistemul propus, pentru a adăuga noi funcționalități și pentru a mări categoria de utilizatori vizați.

3.1.3.Foodily

Acestă aplicaţie se bazează pe principiul unei reţele de socializare, care are ca şi scop oferirea de reţete sănătoase. Reţelele de socializare au avut o creştere vertiginoasă în ultim timp, deci abordarea prin acesta tehnică a unei platforme care ofera utilizatorului posibilitatea de a avea un mod de viaţă mai sănătos, pare o idee bună. Acest sistem oferă utilizatorului mijloace pentru a schimba reţete cu prietenii săi şi o secţiune de întrebări răspunsuri despre modul de proparare a reţetelor.

Avantajele aduse de prezența rețelelor de socializare ar putea fi implementat și în sistemul propus pentru a obține benificiile de markenting aduse de aceasta.

[8] http://techcrunch.com/2012/07/25/zipongo-seed-round/ ultima accesare 28 iunie 2013
[9] http://www.zipongo.com/whats-zipongo ultima accesare 28 iunie 2013
[10] http://www.fooducate.com/about ultima accesare 29 iunie 2013

Capitolul 4. Analiză și fundamentare teoretică

4.1.Fundamentare teoretică

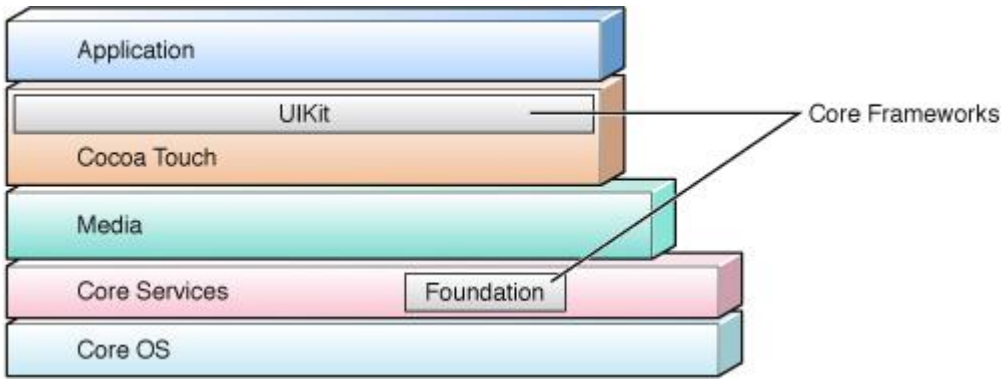
Acest capitol reprezintă o sinteză teoretică a tehnologiilor folosite, atât la nivel de structură a **aplicațiilor client** și **server**, cât și la nivelul de comunicare al componentelor, precum și modelele teoretice de proiectare folosite.Astfel în acest capitol vor fi prezentate tehnologiile alese atât pentru aplicația client, cât și pentru aplicația server.

4.1.1.Tehnologii și resurse utilizate

4.1.1.1. Cocoa Touch/Objective-C și mediul de dezvoltare integrat XCode

Cocoa Touch este tehnologia care permite dezvoltarea aplicațiilor pentru dispozitivele mobile Apple, fiind formată dintr-o colecție de framework-uri. Această tehnologie este, din multe puncte de vedere, simulară tehnologie Cocoa, specifică Machintosh-ului, dar aduce în plus și pune accent pe recunoașterea gesturilor și optimizări[11]. Cea mai mare parte din aceasta colecție de framework-uri este implementată în Objective-C, un limbaj de programare orientat pe obiecte, care este compilat să ruleze la viteze foarte mari , pentru a asigura o bună experiență a utilizatorului la interacțiunea cu aplicația. Objective-C este un superset al limbajului C, astfel încât se pot introduce clase scrise în C sau C++[12].

În conținuare este ilustrată și explicată diagrama arhitecturală a acestei tehnologii, prin etajarea principalelor nivele din care este formată.



Figură 1.Arhitectura platformei Cocoa Touch

În conținuare este explicat fiecare nivel al arhitecturii:

- **Core OS**: Acest nivel conține programul kernel, fișierele de sistem, infrastructura de rețea, infrastructura de securitate, gestionarea energiei precum și un număr de drivere pentru dispozitive. Tot aici se găseste libraria libSystem, care conține interfețele de programare ale aplicațiilor pentru multe servicii.
- **Core Services**: Framework-urile din acest nivel oferă servicii precum manipularea șirurilor de caractere, managementul colecțiilor, rețea sau preferințe. Totodată, resursele hardware ale dispozitivelor pot fi accesate de aici. În acest nivel sunt oferite

[11]https://developer.apple.com/technologies/ios/cocoa-touch.html ultima accesare 1 iulie 2013
[12]http://developer.apple.com/library/ios/documentation/general/conceptual/DevPedia-CocoaCore/Cocoa.html ultima accesare 2 iulie 2013

abstractizări ale tipurilor de date comune, cum ar fi șiruri de caractere sau colecții de obiecte. Tot aici este definit framework-ul CoreData , care reprezintă o soluție pentru managementul unui graf de obiecte, precum și persistenta lor.

- **Media**: Acest nivel depinde de nivelul inferior CoreServices și oferă servicii grafice și multimedia.
- **Cocoa Touch**: Frameworkurile din acest nivel ajută în mod direct la dezvoltarea aplicațiilor în iOS(sistemul de operare de pe dispozitivele Apple).
- **UIKit**: Acest framework conține toate elementele pe care o aplicație le afișează utilizatorului și definește structura pentru comportamentul aplicațiilor, incluzând manipularea evenimentelor utilizator.
- **Foundation**: Acest framework definește comportamentul obiectelor, stabilește mecanismele de managment ale acestora, precum și obiecte pentru tipurile de date primitive[13].

Objective-C este un limbaj de programare simplu, proiectat pentru programarea pe obiecte,fiind definit ca un set de extensii aduse limbajului standard C. În cele ce urmeaza vor fi prezentate câteva particularități ale limbajului Obiecte C: după cum sugerează și numele programării orietata pe obiecte, programele dezvoltate în Objective-C sunt contruite în jurul obiectelor. Un obiect asociază datele pe care le înglobează cu un set particular de operații care folosesc sau schimbă aceste date prin transmiterea de mesaje. Objective-C oferă un tip de date care poate identifica o variabilă obiect, fără a specifica clasa particulară a acestui obiect. Operațiile descrise mai sus sunt numite *metode și* datele pe care le manipulează sunt numite *variabile instanța* (acestea mai pot fi referite ca și *ivar* sau*variabilemembru*). Așadar, un obiect înglobează o structură de date și un grup de proceduri, într-o entitate de sine stătătoare. Ca și cum o funcție C protejează variabilele locale (ascunandu-le de restul programului), un obiect ascunde atât variabilele instanță cât și implementarea metodelor. În objective-C este definit un identificator de obiect ca și un tip de date distinct, cu numele *id*. Acest tip de date este un tip de date general, care poate referi orice obiect indiferent de clasa lui și poate fi folosit pentru instanțe ale unei clase sau chiar pentru clasa. Sintaxa declarării unui astfel de obiect arată în felul următor :*id unObiect*.

Mediul de dezvoltare Xcode este o unealtă care face posibilă dezvoltarea aplicațiilor folosind kiturile de dezvoltare softare ale tehnologiilor Cocoa și Cocoa Touch. Este o aplicație care permite dezvoltarea unui proiect de la concepere până la publicare. Principalele caracteristici ale acestui mediu de dezvoltare sunt[14]:

- Crearea și managementul proiectelor, incluzând specificarea platformelor, specificarea dispozitivelor, dependențelor și configurarea fișierelor executabile.
- Scrierea codului sursă într-un editor care permite aucompletarea elementelor de sintaxa, colorarea sintaxei sau indentarea automată.
- Depanarea proiectului local, în simulator, sau la distanță, pe dispozitivele mobile, cu ajutorul unui depanator vizual.
- Compilarea codului sursă cu ajutorul a unuia din cele două compilatoare disponibilie : GCC (GNU C compiler) sau LLVM-GCC (Low Level Virtual Machine Compiler).

Xcode poate genera fisiere executabile, coduri sursă puntând fi scrise în C, C++, Objective-C sau Objective C++.

[13]https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html#//apple_ref/doc/uid/TP40002974-CH3-SW16 ultima accesare 5 iulie 2013

[14]https://developer.apple.com/library/mac/documentation/ToolsLanguages/Conceptual/Xcode_User_Guide/000-About_Xcode/about.html ultima accesare 6 iulie 2013

4.1.1.2.Framework-ul CoreData

Core Data este un framework care oferă soluții automate și generalizate unor task-uri comune asociate cu ciclul de viață a obiectelor precum și managmentul grafului de obiecte. Totodată Core Data permite persistența obiectelor. Permite organizarea datelor după modelul relațional entitate-atribute și oferă oportunitarea serializării datelor în tipuri de stocare în formatele XML, Sqlite și date binare. Datele pot fi manipulate folosind obiecte de nivel înalt repezentate de entități și relațiile dintre aceastea. Core Data interacționează direct cu SQLite, fiind astfel un wrapper SQL, ajutând programatorul să nu fie nevoit să scrie cod SQL.

Core Data folosește pentru descrierea datelor un model de date de nivel înalt, format din entități și relațiile dintre ele , precum și de interogări a datelor pentru returnarea unor entități respectând anumite criterii. Astfel entitățile pot fi folosite la nivel pur obiectual, programatorul nefiind nevoit să insiste aupra detaliile de stocare și interogare.

Arhitectura de bază

În majoritatea aplicațiilor , accesul la un document care conține date, presupune crearea unei metode de a deschide fișierul, o referință la document, o metodă de citire a datelor pornind de la rădacina documentului. Pentru scriere și salvare, este nevoie de asemenea de o metodă descriere , o metodă de salvare și o metodă de închidere a documentului. Totodată trebuie creat și un model pentru operația de refacere a conținutului, în cazul în care se dorește acest lucru. Programatorul este responsabil pentru crearea acestor metode.

Utilizând framework-ul CoreData, cea mai mare parte a acestor funcționalități este deja implementată. Această implementare este oferită de un obiect (NSManagedObjectContext), referit în cele ce urmeaza ca și *context*. Contextul are rolul de a intermedia între cererile făcute de utilizator (respective obiectele din aplicație) și mediul de stocare extern (colecție de obiecte persistente)[15].

Contextul de gestiune și obiectele gestionate

Contextul poate fi interpretat și ca o zona tampon, capabilă să facă operații de interogare și salvare asupra mediului de stocare persistent. Când se interoghează mediul de stocare pentru date, se crează copii temporare ale entităților din baza de date, unde acestea, grupate, formează un graf de obiecte (sau o colecție de grafuri de obiecte, când nu există relații între entități). Aici se oferă posibilitatea modificării obiectelor. Pentru ca aceste modificări să persiste, treabuie apelată metoda de salvare a datelor, altfel modificarile sunt doar temporare și se vor pierde odată cu ștergere obiectelor temporare, mai sus mentionate, din memorie.

Obiectele model sunt cunoscute ca și obiecte gestionate (NSManagedObject)[generic]. Entitățile sunt mapate în obiecte care moștenesc aceste obiecte model generice. Toate obiectele gestionate trebuie să fie înregistrate contextului, deoarece acesta realizează operația de salvare. Adăugarea obiectelor sau ștergerea unor obiecte este facută de asemenea de obiectul context. Contextul de gestiune urmărește modificările făcute obiectelor, aici fiind inclusă și adăugarea și ștergerea, și pune la dispozitie și operațiile de anulare (undo) și refacere a modificărilor. Când se realizează schimbări în structura mediului de stocare (relatiile dintre entități), integritatea grafului de obiecte este verificată și menținută de contextul de gestiune.

[15]http://developer.apple.com/library/mac/documentation/cocoa/Conceptual/CoreData/Articles/cdBasics.html#//apple_ref/doc/uid/TP40001650-TP1 ultima accesare 7 iulie 2013

Când se dorește salvarea obiectelor temporare în mediul de stocare, contextul de stare verifică dacă obiectele gestionate sunt valide.în caz afirmativ, obiectele sunt salvate în mediul de stocare persistent și se crează înregistrări unde se adăugă obiectele nou create, șterse sau modificate.În caz negativ, se ridică o excepție și se comunică starea obiectelor invalide.

Există situații în care este nevoie de mai multe obiecte de gestiune a contextului. În acest caz este foarte importantă soluția de management al obiectelor de gestiune a contextului pentru a se evita stările de inconsistență în timpul salvări, deoarece fiecare context operează pe obiectele sale de gestiune independent. Soluții pentru tratarea acestor situații pot fi utilizarea tranzacțiilor, firelor de execuție convergente spre firul de execuție principal pe baza timpului de creare, folosirea notificărilor (modelul observatorului).

Cererile de interogare

Pentru extragerea datelor din contextul de gestiune, se utilizează cereri de interogare, referite în conținuare ca și *interogări*. O interogare este un obiect (NSFetchRequest), în cadrul căruia se specifică detaliile obiectelor care se vor a fi extrase din mediul de stocare. O cerere de interogare poate să aibă trei atribute, din care unul este obligatoriu, și anume numele entității. Cererea poate să mai conțină și un predicat care condiționează extragerea obiectelor, precum și un șir de descriptori după care se face ordonarea rezultatului.

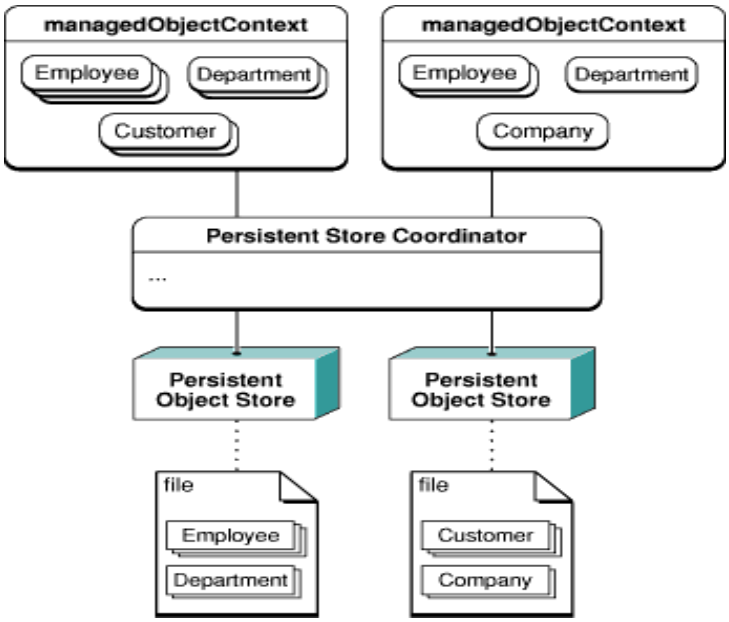
Cererea de interogare este preluată de contextul de gestionare, care returnează un șir de obiecte care se potrivesc cu cererea de interogare.(în cazul în care nu se găsește nici un obiect care să satisfacă condițiile din interogare se returnează un șir care nu conține nici un obiect.) Din moment ce toate obiectele administrate trebuie să fie înregistrate cu un context de gestiune, odată cu returnarea acestora, ele sunt înregistrare în contextul care a preluat cererea de interogare. De remarcat este faptul ca un context de gestionare poate sa conțina o singură instanța a aceluiați obiect model. dacă un obiect există deja în contex, acesta este returnat în rezultat.

Core Data este orientat pe cerere, așadar nu se crează mai multe obiecte decât este nevoie. Graful de obiecte nu conține toate înregistrarile din mediul de stocare. În cazul în care există relații între obiecte, o cerere de interogare nu determină contextul să returneze și obiectele cu care obiectele din rezultat au o legatură. Însă când se face referință la o relație, obiectul la care pointează referința este returnat automat. Când un obiect nu mai este folosit, acesta este dealocat (acest lucru nu înseamnă și ștergerea lui din graful de obiecte.)

Coordonatorul mediilor de stocare persistente

Colecția de obiecte care mediază între obiectele din aplicație și mediul de stocare extern, poate fi imaginat ca și fiind organizat într-o structură de tip stivă. În parte superioară a stivei se află obiectele context de gestionare, iar în partea inferioară a stivei se găsesc mediile de stocare persistente.Între contexte și mediile de stocare se găsește coordonatorul mediilor de stoacare (NSPersistentStoreCoordinator).

In principiu acest coordonator definește o stivă și are rolul unei fațade pentru obiectele context, astfel încât un grup de medii de stocare să pară ca și un singur mediu agregat. Astfel un obiect context poate crea un singur graf de obiecte, incorporând datele din mai multe fișiere. În următoareaeste prezentă o astfel de stivă.



Figură 2. Stivă Core Data[16]

Medii de stocare persistente

Un mediu de stocare este asociat unui singur fișier și este responsabil pentru maparea datelor din mediul de stocare și obiectele corespunzătoare din aplicație, create de obiectele context de gestionare. Singura interacțiune a programatorului cu mediile de stocare persistente ar trebui să fie când acesta specifică locația unui nou fișier extern care ar trebui atașat aplicației.Arhitectura aplicației este independentă de mediul de stocare folosit.

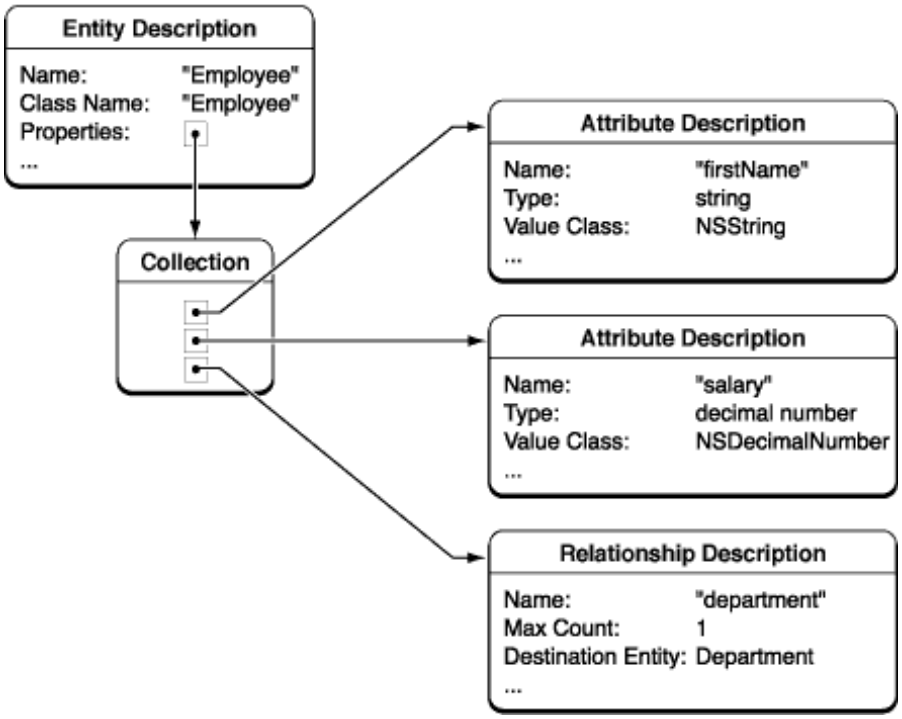
Obiectele gestionate și modelul obiectelor gestionate

Pentru a putea gestiona eficient graful de obiecte și pentru a putea oferi persistența, Core Data are nevoie de o descriere cât mai detaliată a obiectelor cu care operează. Un model al obiectelor gestionate este schema în care se descriu obiectele gestionate, sau entitățile.

Modelul este alcătuit dintr-o colecție de entități și structura acestora (metadate despre entitate, incluzând numele entității, numele clasei care reprezintă entitatea în aplicație, atributele și relațiile entității). La rândul lor atributele și relațiile sunt reprezentate de obiecte de descriere a atributelor și relațiilor.

Obiectele gestionate trebuie să fie instanțe ale NSManagedObject sau subclase ale acestuia. NSManagedObject este un obiect abstract care poate să reprezinte orice entitate, care folosește un mediu de stocare intern și privat pentru a ține evidența proprietăților sale și implementează metodele necesare comportamentului unui obiect gestionat. Un obiect gestionat păstrează o referință la descrierea entității a cărei instanța este. Face referire la descrierea entității pentru a primi metadatele despre sine, incluzând numele entității pe care o reprezintă și informații despre atribute și relații. De asemenea este posibilă subclasarea NSManagedObject pentru a adăuga funcționalități suplimentare.

[16]https://developer.apple.com/library/ios/documentation/DataManagement/Devpedia-CoreData/coreDataStack.html#//apple_ref/doc/uid/TP40010398-CH25-SW1 ultima accesare 18 iulie 2013



Figură 3.Exemplu de entitate și relații[17]

Diferențe dintre CoreData și SQL

O bază de date este reprezentată de un sistem de stocare persistent al datelor, organizată în table, cu rânduri și coloane, care are ca și principal țel păstrarea datelor pe disk. CoreData este un graf de obiecte de gestiune cu caracteristici de ciclu de viața, căutare și persistență[18].

Core Data	**Baza de date (SQLite)**
Funcția primară este aceea de a stoca și oferi date.	Funcția primară este managementul grafului de obiecte.
Operează pe date stocate pe disk	Operează pe date stocate în memorie.
Pot fi tranzacționale, utilizatori multipli	Non-tranzacționale, utilizator unic
Poate șterge tabele și edita date fără a fi încărcate în memorie	Operează doar în memorie
Salvarea pe disk a datelor poate dura mult	Salvarea datelor este rapidă
Crearea unui număr mare de înregistrari poate fi înceată	Creare unui număr mare de obiecte se face în memorie și durează puțin (deși salvarea poate dura mai mult)
Oferă constrângeri precum chei unice	Constrângerile sunt lăsate în seama logicii de bussiness a aplicației
Nu oferă popularea obiectelor automată	Oferă popularea obiectelor automată
Sintaxa Sql personalizată	Nu suportă sintaxa Sql personalizate
Oferă posibilitatea selectării anumitor câmpuri ale unei înregistrări	Nu oferă posibilitatea selectării anumitor câmpuri ale unei înregistrări
Nu suportă migrare automată	Suportă migrare automată
Cod complex	Cod simplu

[17] https://developer.apple.com/library/mac/documentation/cocoa/Conceptual/CoreData/Articles/cdBasics.html#//apple_ref/doc/uid/TP40001650-TP1 accesare 18iulie 2013

[18] http://www.cocoawithlove.com/2010/02/differences-between-core-data-and.html utlima accesare 20 august 2013

Căutare (în funcție de nivelul programatorului)	Căutare optimizată

4.1.1.3.Push Notifications şi APNs

Ţinând cont de faptul că aplicaţiile specifice dispozitivelor Apple nu pot rula în background (decat în anumite situaţii, în care acest proiect nu se încadrează), notificările sunt o soluţie pentru a informa utilizatorul că aplicaţia are informaţii care se doresc a fi prezentate. Informaţiile pot fi reprezentate de mesaje, evenimente calendaristice sau introducerea de date noi pe un server la distanţă. Notificările pot avea trei forme (sau orice combinaţie între cele 3): alerte vizuale care afişează un mesaj scurt, insigne sau sunete. Când utilizatorul este notificat că aplicaţia are un mesaj, eveniment sau alte date de afişat, aceştia pot opta să deschidă aplicaţia şi să vadă detaliile sau să ignore notificarea, caz în care aplicaţia nu se va deschide[19].

Pe sistemul de operare iOS, doar o singură aplicaţie poate fi activă în acelaşi moment de timp. Această problemă este rezolvată prin folosirea notificărilor. Multe aplicaţii rulează într-un mediu bazat pe timp, sau sunt interconectate cu alte sisteme în care pot apărea evenimente de care utilizatorii ar putea fi interesaţi şi aplicaţia nu este în prim plan. Notificările oferă posibilitatea aplicaţiilor să atenţioneze utilizatorii că aceste evenimente au avut loc.

O **notificare**[20] este un mesaj scurt pe care un server il transmite sistemului de operare al unui dispozititv.La rândul său sistemul de operare trimite o notificare către utilizatorul aplicaţiei că există date care trebuie descărcate. Dacă utilizatorul acceptă această caracteristică şi aplicaăia este înregistrată în mod corespunzător, atunci notificarea este transmisa sistemului de operare şi mai apoi aplicaţiei. APNS(Apple Push Notification Service) este tehnologia care oferă această funcţionalitate.

Notificarea constă într-o sarcină care conţine o listă de atribute care conţine proprietăţi definite de APNs şi care specifică cum trebuie informat utilizatorul.Din motive de performanţă sarcina este mică. Limita maximă pe care o notificare poate să o transmită este de 256 de bytes. Deşi se pot defini proprietăţi de către programator, acest lucru nu se recomandă şi nici transportul datelor prin sistemul de notificări, întucât livrarea notificărilor nu este garantată.APNS păstrează ultima notificare pe care o primeşte de la un furnizor destinată unei aplicaţii de pe un dispozitiv. Aşadar, pentru un dispozitiv care ar fi trebui să primească o notificare, dar nu a avut acoperire de semnal sau acces la internet, APNS-ul transmite notificarea către dispozitiv în momentul în care acesta are conexiune la internet. Orice dispozitiv iOS cu versiunea mai mare decat 4.0 poate primi notificări atât prin conexiuni celulare sau Wi-Fi. Conexiunea Wi-Fi poate fi folosită doar în cazul în care nu există conexiune celulară sau dacă dispozitivul este de tip iPod.

O notificare poate conţine una sau mai multe proprietăţi care specifică următoarele acţiuni:

- Un mesaj de alertă afişat utilizatorului
- Un număr care va reprezenta insigna icoanei aplicaţiei
- Numele unui fişier sunet

Sarcina unei aplicaţii poate arăta astfel:

[19]https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html ultima accesare 8 iulie 2013

[20]https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/WhatAreRemoteNotif.html#//apple_ref/doc/uid/TP40008194-CH102-SW1 utlima accesare 8 iulie 2013

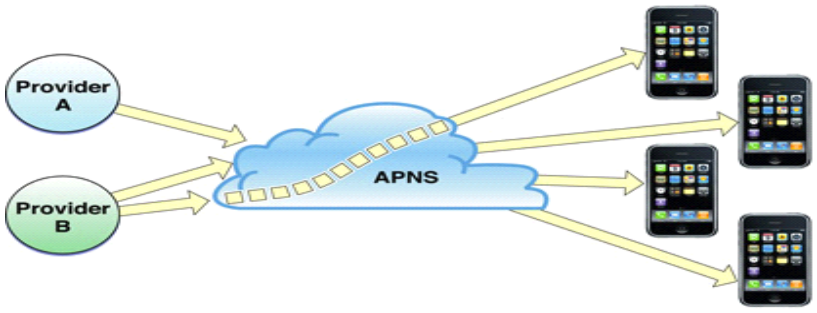
```
{
"apps": {
        "alert" : "Got new data"
        "badge" : 7
        "sound" : "chime.aiff"
        }
}
```

Apple Push Notofication Service[21] (**APNs**) este piesa centrală care face posibilă utilizarea notificărilor, fiind un sistem robust și eficient de propagare a informaţiilor spre dispozitive cum sunt iPhone, iPad sau iPod. Fiecare dispozitiv crează o legatură criptată pe bază de date de identificare unice cu serviciul și primește notificări prin acesta conexiune persistentă. Dacă o notificare ajunge la aplicaţia client când aceasta nu rulează, dispozitivul trimite un mesaj de alertă către utilizator.În afară de a fi un serviciu de transport simplu și eficient APNS-ul oferă și servici de stocare și înaintare.

APNS transportă și rutează o notificare de la un furnizor la un anumit dispozitiv. O notificare este este un mesaj scurt care conţine două părţi importante: un jeton de autentificare și sarcina notificării. Jetonul de autentificare este analog unui număr de telefon și care ajuta APNS să localizeze dispozitivul pe care este instalată aplicaţia client. Totodată jetonul de autentificare este folosit în sistemul de rutare a notificării. Sarcina notificării este un Json cu proprietăţi predefinite, care specifică cum ar trebui atenţionat utilizatorul aplicaţiei.

Fluxul de date pentru o notificare este unidirectional, de la furnizor spre aplicaţia client. Furnizorul crează un pachet de notificare care include jetonul de autentificare și sarcina notificării. Apoi furnizorul trimite notificarea spre APNS, care la rândul lui trimite notificarea către dispozitivul corespunzător.Când un furnizor se autentifica la APNS. Acesta trimite serviciului date de indentificare a aplicaţiei pentru care se doreste trimiterea notificărilor. Aceste date cuprind identificatorul aplicaţiei și identificatorul dispozitivului mobil.

În următoarea figură este prezentat fluxul de date pentru o aplicaţie.



Figură 4.Fluxul date de la mai mulţi furnizori la mai multe dispozitive

Pentru a comunica cu un furnizor și un dispozitiv, APNS trebuie să menţină puncte de intrare deschise către acestea. Pentru a asigura securitatea acestor puncte de intrare, trebuie verificat accesul către acestea. Astfel APNS-ul defineşte două nivele de încredere pentru

[21] https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html#//apple_ref/doc/uid/TP40008194-CH100-SW9 utlima accesare 7 iulie 2013

furnizori , dispozitivie și comunicarea dintre acestea. Astfel cele două nivele de încredere sunt acordate pentru conexiune și pentru jetonul de autentificare.

La nivelul de conexiune, Apns-ul trebuie să stabilească cu certitudine că furnizorul care cere trimiterea notificărilor este autorizat și că Apple a fost de acord să trimită aceste notificări. Pe partea comunicării cu dispoztivele, APNS-ul trebuie să valideze conexiunea cu dispozitivul ca fiind legitimă. După stabilirea nivelului de încredere, APNS-ul trebuie să trimită notificările doar dispozitivelor precizate.

La nivelul jetonul de autentificare, se asigură acurateţea rutării mesajelor cu ajutorul jetonul.Jetonul este un identificator unic al unui dispozitiv cu care APNS se conectează. Dispozitivul partajează jetonul cu furnizorul său, astfel încât în orice proces de comunicare, jetoul însoţeşte fiecare notificare.

4.1.1.4.Php, framework-ul Zend și mediul de dezvoltare NetBeans

PHP

Php, acronim pentru HyperText Preprocessor , este un limbaj de programare folosit pentru a produce pagini web dinamice, fiind folosit pe scară largă pentru a dezvolta pagini și aplicații web. Se foloseste în principal inglobal în cod HTML, dar începand cu versiunea 4.3.0 se poate folosi și în mod 'linie de comandă', permițând astfel crearea de aplicații independente. Este unul din cele mai importante și folosite limbaje de programare web server-side.

Php este ușor de utilizat, fiind un limbaj de programare structurat, sintaxa limbajului fiind o combinatie între C, Pearl și Java[22]. Una din cele importante facilități ale limbajului este înglobarea acestui cu majoritatea bazelor de date relationare (MySql, Oracle, MSSqlServer PostgreSql sau DB2). Php poate rula pe majoritatea sistemelor de operare (Unix, Linux, Windows sau MacOSX) și poate interacționa cu majoritatea serverelor web.Codul Php este interpretat de serverul web și generează un cod HTML care este afișat utilizatorului prin aplicația client.Principalele tipurile de date folosite de Php sunt următoarele: date boolean, întregi, float, șiruri de caractere, șiruri, obiecte și null.

Ceea ce face ca PHP să difere de JavaScript, pe partea clientului, este că codul său este executat pe server, generând HTML care este apoi trimis către client. Clientul va primi rezultatele rulării acelui script, fără a putea cunoaște codul-sursă ce stă la bază. Serverul web poate fi configurat în așa fel încât toate paginile să fie executate cu php, astfel, nici o informație nu va mai putea fi vizibilă în codul sursă al paginiii web. Php este convenabil atât pentru programatorii începatori, cât și pentru programatorii avansați[23].

Scripturile Php sunt utilizate în trei situații, descrise în cele ce urmează:

- Scripting de partea serverului,acesta fiind unul din cel mai tradițional și de bază domeniu al utilizării PHP. Pentru funcționarea pe partea serverului este nevoie de trei module:
  - analizatorul PHP (în calitate de CGI, sau modul pentru server)
  - un server web
  - un navigator web.

  Pe serverul web trebuie instalat PHP, iar ieșirile programului PHP pot fi accesate cu navigatorul web, vizualizând pagina PHP prin server.

[22] http://www.php.net/manual/ro/preface.php, ultima accesare 18 iunie 2013
[23] http://www.php.net/manual/en/intro-whatis.php , ultima accesare 18 iunie 2013

- Scripting în linia de comandă. Există posibilitatea ca un script PHP să ruleze fără un server și navigator web. Este nevoie doar de analizatorul PHP pentru a-l utiliza în acest mod.
- Scrierea aplicațiilor de birou.

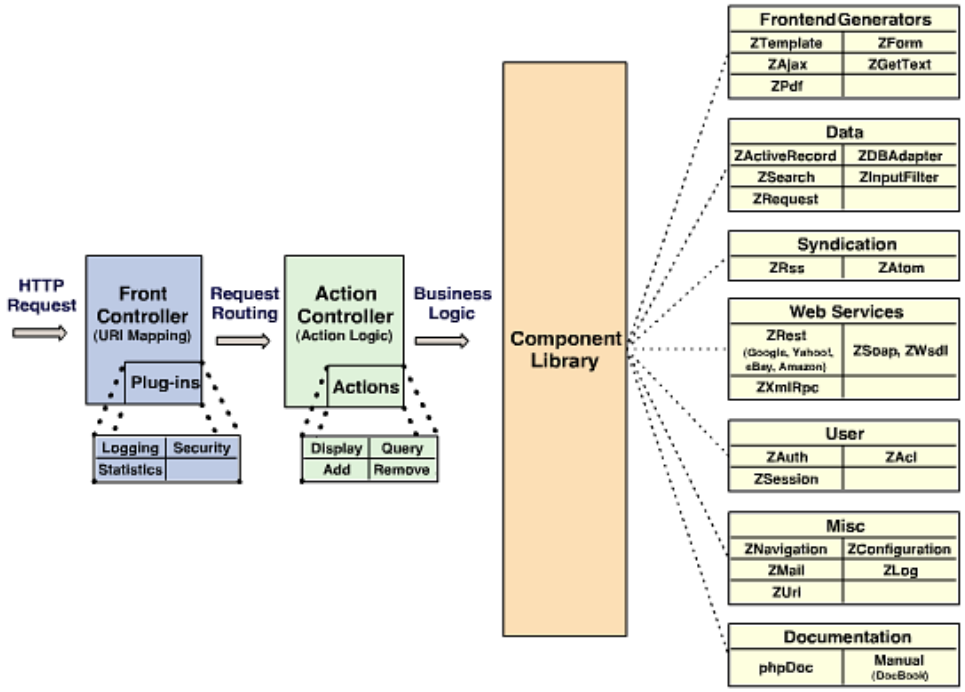
Așadar, utilizand PHP se oferă libertatea alegerii sistemului de operare și a serverului web.Mai mult, exită posibilitatea de a utiliza programarea procedurală sau programarea orientată pe obiecte (POO), sau o combinație a acestora[24].

Framework-ul Zend 2

Framework-ul **Zend** este o colecție de librării, implementată în PHP 5, care oferă o abordare obiectuală aplicațiilor web.Structura acestui framework este unică, prin faptul că proiectarea acestuia a fost făcută într-un fel în care orice componentă are dependințe minimale fața de oricare altă componentă. Această decuplare a componentelor oferă posibilitatea programatorului să folosească doar părțile necesare.

Chiar dacă se pot utiliza separat, componentele framework-ului Zend 2 care sunt înglobate în libraria standard, au o arhitectura care urmărește principiul de programare Model-Vedere-Controler. Totodată oferă o abstractizare a bazei de date foarte simplu de înțeles și implementat, validări sau filtrări. În acest framework se oferă și metode de autentificare sau autorizație, toate acestea sub o interfață orientată obiect.

În următoarea figură este prezentată pe larg arhitectura framework-ului Zend.



Figură 5.Arhitectura framework-ului Zend[25]

Un alt avantaj care propune utilizarea acestui framework, chiar și în proiecte de dimensiuni mari, este posibilitatea integrării cu ușurință a altor servicii web oferite de companii mari din domeniul IT, cum ar Google sau Microsoft.[26]

[24] http://www.php.net/manual/en/intro-whatcândo.php, utima accesare 20 iunie 2013
[25] http://shiflett.org/blog/2005/dec/zend-framework-webcast ultima accesare 10 augist 2013
[26] http://framework.zend.com/manual/2.0/en/ref/overview.html ultima accesare 10 august 2013

Așadar principalele caracteristici[27] care au recomandat acest framework pentru a fi folosit, sunt:

- Toate componentele sunt în totalitate orientate obiect
- Arhitectura decuplată ale componentelor
- Interdependențele minimale dintre module
- Implementare ușoară folosind sablonul arhitectural Model-Vedere-Controler
- Suport pentru mai multe tipuri de baze de date, printre care e inclusă și MySql, baza de date folosită de aplicația server
- Sistem de caching care suportă mai multe tipuri de implementare, printre care se numără memoria sau fișiere de sistem.

Mediul de dezvoltare integrat NetBeans

Mediul de dezvoltare integrat NetBeans, este o aplicație dezvoltată în special pentru Java, dar se pretează a fi folosită și prentru scierea aplicațiilor PHP. Este scris în Java și poate să funcționeze pe orice mașină care rulează sisteme de operare precum Windows, OS X, Linux, Solaris sau orice platformă care suporta JVM[28].

Pentru dezvoltarea aplicațiilor PHP, principalele caractistici[29] ale acesui sistem sunt următoarele:

- Evidențierea cuvintelor din sintaxă
- Completarea automată a codului
- Evidențierea erorilor de sintaxă
- Analiza semnatică a parametrilor funcțiilor
- Analiza variabilelor și evidențirea celor nefolosite
- Depanator de cod PHP
- Suport pentru framework-ul Zend care este folosit pentru scrierea aplicației server
- Oferă posibilități de testare precum PHPUnit sau Selenium

Având în vedere aceste caracteristici, este posibilă implmentarea aplicației client al proiectului prezentat prin folosirea acestui mediu de dezvoltare integrat.

4.1.1.5.JSON și MySql

JSON

JSON, acronim pentru JavaScritp Object Notation, este un standard deschis conceput pentru schimbul de date, fiind ușor de citit și scris de oameni și ușor de parsat și generat pentru mașini. Deriva din JavaScript, dar este independent de platforma și limbaj de programare. Este folosit pentru a serializa și transmite date structurate prin intermediul unei

[27] http://en.wikipedia.org/wiki/Zend_Framework ultima accesare 10 august 2013
[28] http://netbeans.org/features/platform/features.html ultima accesare 12 august 2013
[29] http://en.wikipedia.org/wiki/NetBeans, ultima accesare 12 august 2013

rețele, în special în modelul de comuncare server-client, fiind o alternativă tot mai des folosită pentru XML[30].

Json-ul este construit pe 2 structuri:

- O colecție de perechi nume : valoare. În diverse limbaje de programare acestea sunt implementate ca și obiecte, înregistrari, structuri, dicționare, hash table, liste sau șiruri asociative. În limbajul objective-C se foloseste obiectul *NSDictionary* sau *NSMutableDictionary*.
- O listă ordonată de valori. Cele mai multe limbaje de programare implementează acesta listă ca și un sir, vector , listă sau secvență. În objective-C se folosește obiectul *NSArray* sau *NSMutableArray*.

Tipurile de date suportate de JSon sunt numerele , șirurile de caractere, valorile booleane, șiruri, obiecte (colecții de perechi cheie: valoare neordonate) și null. Analog schemei de validare XML, există și o schema de validare Json.

MySql

Mysql este un sistem de gestiune a bazelor de date relațional, gratuit, open-source, fiind o componentă cheie a stivei LAMP (Linux, Apache, MySql, PHP). MySql este scris în C și C++ și poate rula pe multe sisteme de operare. Multe limbaje de programa au incluse librării pentru accesul la baza de data MySQL.

Deși este folosit foarte des împreună cu limbajul de programarePHP, cu MySQL se pot construi aplicații în orice limbaj. Există multe scheme API disponibile pentru MySQL ce permit scrierea aplicațiilor în numeroase limbaje de programare pentru accesarea bazelor de date MySQL, cum are fi: C, C++, C#, Java, Perl, PHP, Python, FreeBasic, etc., fiecare dintre acestea folosind un tip spefic API. O interfață de tip ODBC denumită MyODBC permite altor limbaje de programare ce folosesc această interfață, să interacționeze cu bazele de date MySQL cum ar fi ASP sau Visual Basic. În sprijinul acestor limbaje de programare, unele companii produc componente de tip COM/COM+ sau .NET (pentru Windows) prin intermediul cărora respetivele limbaje să poată folosi acest SGBD mult mai ușor decât prin intermediul sistemului ODBC. Aceste componente pot fi gratuite (ca de exemplu MyVBQL) sau comerciale.

Ca și PHP, MySQL este componentă integrată a platformelor LAMP sau WAMP (Linux/Windows-Apache-MySQL-PHP/Perl/Python). Popularitatea sa ca aplicație web este strâns legată de cea a PHP-ului care este adesea combinat cu MySQL și denumit Duo-ul Dinamic. În multe cărți de specialitate este precizat faptul ca MySQL este mult mai ușor de invățat și folosit decât multe din aplicațiile de gestiune a bazelor de date, ca exemplu comanda de ieșire fiind una simplă și evisentă: „exit” sau „quit”.

Pentru a administra bazele de date MySQL se poate folosi modul linie de comandă sau, prin descărcare de pe internet, o interfață grafică: MySQL Administrator și MySQL Query Browser. Un alt instrument de management al acestor baze de date este aplicația gratuită, scrisă în PHP, phpMyAdmin.

MySQL poate fi rulat pe multe dintre platformele software existente: AIX, FreeBSD, GNU/Linux, Mac OS X, NetBSD, Solaris, SunOS, Windows 9x/NT/2000/XP/Vista.[31]

Principalele caracteristici[32] pentru care a fost ales sistem sunt prezentate mai jos:

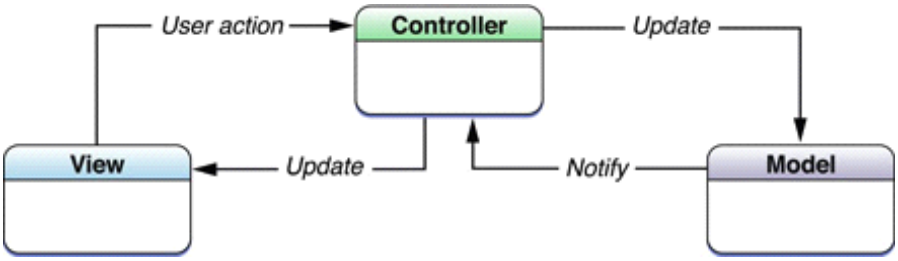
[30] http://www.json.org/ ultima accesare 10 august 2013
[31] http://en.wikipedia.org/wiki/MySQL ultima accesare 11 august 2013
[32] http://ist.berkeley.edu/services/ds/db/mysql/basic ultima ccesare 11 august 2013

- O instanță MySql dedicata care rulează pe o bază de date partajată
- Spațiu de utilizare de 5 GB
- 28MB de memorie RAM dedicată instanței
- Rate de acces la baza de date de 40 de cereri/secundă pentru interogări simple, cum ar fi selecturile sau inserturile
- Rutine de mentenanță care se efectuează automat

4.1.2.Concepte teoretice de proiectare

4.1.2.1.MVC (Modal View Controller)

Șablonul de proiectare MVC asignează obiectelor din aplicație unul din următoarele trei roluri: model, vedere sau controler, precum și modurile de comunicare dintre acestea. Fiecare din cele trei tipuri de obiecte este separat de celelalte prin limite abstracte și procesul de comunicare se face prin aceste limitări abstracte. MVC este un șablon central al unei aplicații Cocoa. Există multe avantaje în folosirea acestui sablon : multe din obiecte tind să fie reutilizabile și interfețele lor vor fi mai bine definite, extensibilitate. Multe dintre tehnologiile și arhitecturile Cocoa sunt bazate pe MVC și au nevoie ca obiectele din aplicație să joace unul din cele 3 roluri. În următoarea figură sunt prezentate grafic cele trei roluri și modelul de comunicare dintre acestea[33].



Figură 6.Șablonul ahitectural Model - Vedere - Controler

Obiectele model încapsulează date specifice unei aplicații și definesc logica care manipulează și proceseara datele. Un obiect model poate avea relatii de tipul unu-la-unu, unu-la mulți sau mulți-la-unu cu alte obiecte. O mare parte din datele unei aplicații sunt date persistente, care sunt încărcate și salvate în fișiere sau baze de date. În mod ideal, obiectele model nu ar trebui să aibă nici o legatură cu obiectele de tip vedere (care reprezintă vizual datele din model).Comunicarea: acțiunile utilizatorilor în interfața cu utilizatorul pot crea, șterge sau modifica datele. Aceste acțiuni sunt transmise obiectului controler care modifică modelul, iar după ce modelul s-a modificat acesta transmite un mesaj controlerului care conține logica ulterioară.

Obiectele vedere sunt obiectele dintr-o aplicație pe care utilizatorul le poate vedea. Un astfel de obiect are funcționalități de desenare și poate răspunde la acțiunile utilizatorilor. Rolul principal al acestor obiecte este de a aplica datele din modelul aplicației și să permită editarea acestor date. Cu toate acestea, obiectele vedere trebuie să fie decuplate de obiectele model. Comunicarea se face doar cu obiectul controler pe care îl notifică când utilizatorul realizează acțiuni și de la care primește date când modelul se schimbă.

[33]https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/MVC.html ultima accesare 10 august 2013

Obiectele controler se comportă ca și un intermediar între unul mai mai multe obiecte vedere și unul sau mai multe obiecte model. Aceste obiecte crează obiectele model și vedere și asigură un proces de comunicare între ele. Comunicarea : un controler interpretează acțiunile utilizatorilor din obiectele vedere și transmite datele obiectelor model. După ce modelul s-a modificat, noile date sunt preluate de controller și transmise obiectele vedere pentru a fi afișate.

4.1.2.2.Protocoale de comunicare

Protocoalele declară metode care pot fi implementate de orice clasă din aplicație. Cele mai importante trei utilități ale protocoalelor sunt următoarele:

- declară metode pe care alte obiecte pot să le implementeze
- declară interfața pentru un obiect în timp ce ascunde implementarea lui
- captează asemănări între clasele care nu sunt legate ierarhic

Interfețele claselor și a categoriilor declară metode care sunt asociate cu o clasă praticulară, în principal metode pe care clase le implementează. Protocoalele formale și informale, declară metode care sunt independente de o anumită clasă, dar pe care orice clasă poate să le implementeze. Un protocol este o inșiruire de declarații de metode, independente de orice clasăă. Protocoalele oferă posibilitatea declarării metodelor independente de ierarhia claselor, astfel încât acestea pot fi folosite într-un mod diferit față de clase și categorii. Lista metodelor declarate de un protocol poate fi scrisă oriunde în aplicație (în orice clasă), fără ca identitatea clasei să fie de interes pentru obiectele care implementează protocolul.

Protocoalele joacă un rol important în proiectarea aplicațiilor orientate pe obiecte, mai ales când proiectul este dezvoltat de mai mulți programatori și sunt folosite obiecte din alte proiecte. Cocoa folosește protocoalele pentru a facilita procesul de comunicare prin mesaje.Metodele unui protocol nu trebuie implementate de clasa ca îl implementează , doar în cazul în care acestea sunt marcate ca obligatorii (nu se găsesc în blocul de declarații precedat de directiva @optional)[34].

În aplicația client protocoalele vor fi folosite la nivel de implementare, utilizate atât pentru popularea listelor, cât și pentru implementarea acțiunilor necesare la selectarea unui rând din tabel. O altă utilizare a protocoalelor este folosită pentru determinarea acțiunilor necesare la schimbarea stării aplicației.

4.1.2.3.Delegare

Delegarea este un șablon arhitectural simplu, dar puternic în care un obiect dintr-un program acționează în numele sau în concordanta cu un alt obiect. Obiectul care deleagă păstrează o referință la obiectul delegat și la momentul oportun trimite un mesaj spre acesta. Mesajul transmis informează delegatul ca un eveniment pe care obiectul care l-a delegat urmează să trateze sau a trat evenimentul respectiv. Delegatul poate să răspundă acestui mesaj prin actualizarea stării lui sau a altor obiect din aplicație sau poate să răspundă cu o valoare care afecteaza tratarea evenimentului. Avantajul principal al delegării este constituit de ușurința cu care poate fi personalizat comportamentul mai multor obiecte în funcție de un obiect central[35].

[34]https://developer.apple.com/library/ios/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/WorkingwithProtocols/WorkingwithProtocols.html utlima accesare 12 august 2013

[35]https://developer.apple.com/library/ios/documentation/general/conceptual/DevPedia-CocoaCore/Delegation.html ultima accesare 14 august 2013

Obiectul care crează un obiect delegat este de obicei un obiect dintr-un framework, iar delegatul este un obiect personalizat care de obicei joacă rolul unui controler. Într-un sistem în care se face managmentul memoriei automat, obiectul care deleagă alt obiect păstrează o referință "slabă" la acesta, iar în cazul în care managementul memoriei se face manual, referința către obiectul delegat trebuie să fie una "puternică".

Delegatul celor mai multe clase în framework-urile Cocoa Touch, este înregistrat automat ca și observator la notificările postate de obiectul care deleaga. Astfel delegatul trebuie să implementeze doar o metodă de notificare declarată de framwork-ul clasei pentru a putea primi mesajul de notificare.

Șablonul arhitectural sursă de date este aproape identic cu delegarea. Diferența constă în relația cu obiectul delegat. În loc să fie cedat controlul asupra interfeței cu utilizatorul, sursei de date îi este cedat controlul asupra datelor. Obiectul care deleagă păstrează o referință către obiectul delegat și ocazional cere datele pe care are nevoie să le afișeze. Ca și în cazul delegării, un obiect sursa de date trebuie sa adopte un protocol și să implementeze un număr minim de metode obligatorii definite de protocol.

4.1.2.4.Singleton

O clasă care implementeză șablonul arhitectural *singleton* returnează aceași instanță a obiectului de fiecare dată când aplicația il cere. O clasă permite crearea oricâtor instanțe, pe când o clasă singleton returnează aceeași instanță de fiecare dată. Un obiect singleton reprezintă un punct de acces global la resursele clasei.

Instanța globală a obiectului singleton se obține prin metoda de factorizare. Clasa incărca intârziat instanța prima dată când este cerută și astfel se asigură că nu se va mai crea o alta instanță. Totodată un obiect singleton se asigură că instanța nu poate fi copiată sau distrusă.

Acest șablon arhitectural se folosește în cazul în care se dorește coordonarea centralizată a unui sistem. O altă utilizare frecventă a acestora este reprezentată de înlocuirea variabilelor globale.(astfel nu se încarcă spațiul numelor globale și este permisă inițializarea întârziată).

4.2.Analiză

4.2.1.Cazuri de utilizare

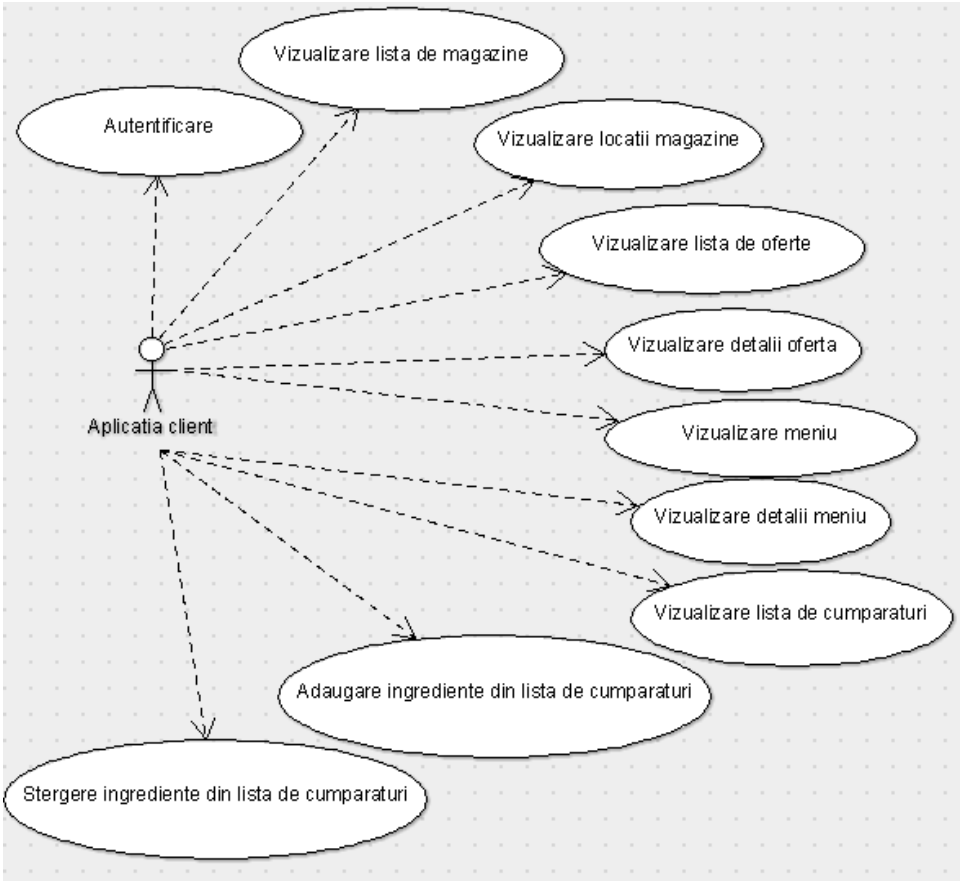
. Cazurile de utilizare reprezintă o descriere a sistemului din punct de vedere funcțional.Modelul cazurilor de utilizare reprezintă ansamblul tuturor cazurilor de utilizare și utilizatorii acestuia (actorii). Diagramele cazurilor de utilizare descriu comportamentul dorit al sistemului și oferă o imagine de ansamblu asupra modului în care actorii pot folosi sistemul. Cazurile de utilizare reprezintă o descriere a sistemului din punct de vedere funcțional.

După cum s-a menționat la specificațiile proiectului aplicația este împărțită în patru componente, din care două sunt proprii și două auxiliare. În continuare se vor prezenta cazurile de utilizare, specifice pentru fiecare componentă.

Aplicația Server

În figura următoarea este prezentată diagrama de cazuri de utilizare pentru aplicația server. Actorul principal este aplicația client, care la acțiunile utilizatorului final, realizează acțiunile descrise mai jos. După cum se poate observa în aceasta diagramă a cazurilor de

utilizare se disting zece cazuri de utilizare, care corespund cu cele zece servicii furnizate de componenta server a sistemului.



Figură 7.Diagrama cazurilor de utilizare pentru componenta server

- **Scută descriere a scenariului:**
  Sunt prezentate interacţiunile dintre aplicaţia client si aplicaţia server.
- **Actorul principal:**
  Actorul principal in acest scenariu de utilizare este aplicaţia client.
- **Precondiţii:**
  Nu există.
- **Scenariu de succes:**
  1. *autentificare*: serviciu necesar pentru autentificarea în sistem a utilizatorului
  2. final care este folosit pentru a face diferența dintre utilizatori diferiți
  3. *vizualizarea liste de magazine*: serviciu necesar pentru trimiterea listei de magazine în functie de poziția utilizatorului
  4. *vizualizare locații magazine*: serviciu necesar pentru returnarea locațiilor magazinelor
  5. *vizualizare listă de produse*: serviciu necesar pentru afișarea în aplicația client a listei de produse de la magazinele selectate de utilizator
  6. *vizualizare detalii produs*: serviciu necesar pentru afișarea către utilizator a detaliilor produselor
  7. *vizualizare meniu*: serviciu care returnează o listă de meniuri pentru următoarele șapte zile
  8. *vizualizare detalii meniu*: serviciu necesar pentru afișarea detaliilor specifice pentru fiecare meniu

9. *vizualizare listă de cumpărături*: serviciu necesar pentru afișarea în aplicația client a ingredientelor din lista de cumpăraturi a utilizatorului, serviciu care împreună cu adăugarea și ștergerea intrărilor din lista de cumpărături, formează o cerința funcțională amintită la subcapitolul cerințe funcționale.
10. *adăugare ingrediente în lista de cumpărături*: serviciu care permite adăugarea înregistrarilor în lista de cumpărături
11. *ștergerea ingredientelor din lista de cumpărături*: serviciu care permite ștergerea ingredientelor din lista de cumpărături

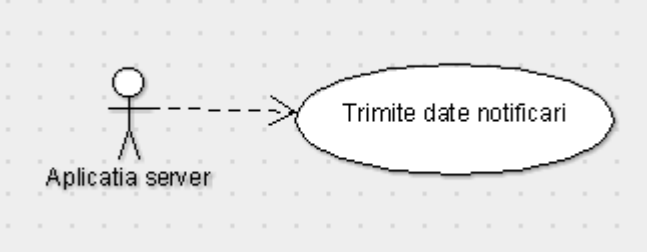
- **Scenariu de eşec:**
  Nu există.

Sistemul Urban Airship

Sitemul Urban airhip are trei cazuri de utilizare cu actori diferiți.

Pe de o parte, actorul este aplicația server, care în momentul introducerii unui nouprodus trimite datele necesare trimiterii notificării către sistemul UA. Cazul de utilizare aferent este descris viziual în următoarea figură.

- **Scută descriere a scenariului:**
  Interacţiunea dintre aplicaţia server şi sistemul Urban Airship.
- **Actorul principal:**
  Actorul principal este aplicaţia server.
- **Precondiţii:**
  Nu există precondiţii ale acestui caz de utilizare.
- **Scenariu de succes:**
  1. Aplicaţia server *trimite datele* necesare trimiterii *notificării* la sistemul UA.
- **Scenariu de eşec:**
  1. Sistemul UA nu validează datele necesare trimiterii notificării.



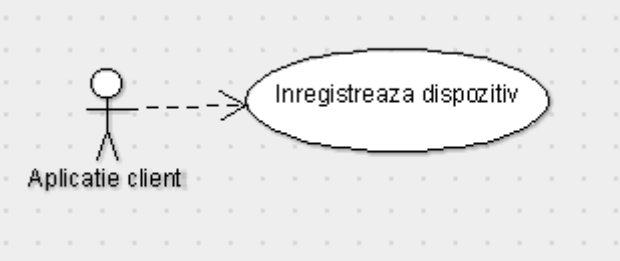
Figură 8. Caz de utilizare: Urban Airhip - aplicația server

Pe de altă parte, actorul este aplicația client care înregistrează fiecare dispozitiv pe care este instalată aplicația în baza de date a sistemului Urban Airhip pentru ca acesta să poată să trimită spre apns datele necesare trimiterii notificărilor. În următoareafigură se poate vedea acest caz de utilizare.

- **Scută descriere a scenariului:**
  Interacţiunea dintre aplicaţia client şi sistemul Urban Airship.
- **Actorul principal:**
  Actorul principal este aplicaţia client.
- **Precondiţii:**
  Nu există precondiţii ale acestui caz de utilizare.
- **Scenariu de succes:**

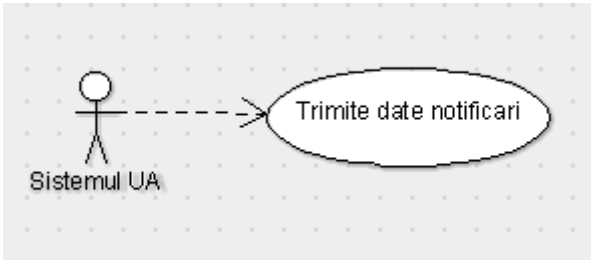
1. Aplicaţia client se *înregistrează cu succes dispozitul mobil*.

- **Scenariu de eşec:**
  1. Sistemul UA nu validează informaţiile dispozitivului mobil.



Figură 9. Caz de utilizare: Urban Airhip - aplicația client

În ultimul rând, actorul principal este aplicația urban airhip, care înaintează datele primite de la server spre serviciul APNs, pentru a fi transmise către dispozitive. Acest caz de utilizare este prezentat în următoarea figură.



Figură 10. Caz de utilizare: Urban Airship – APNs

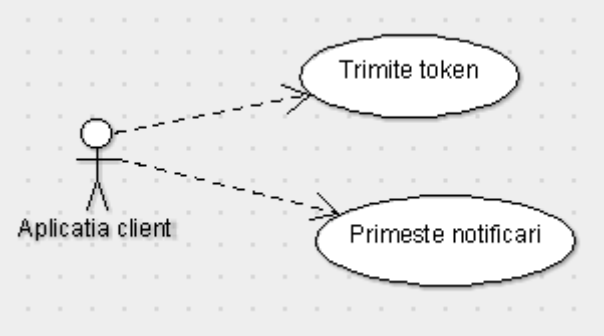
- **Scută descriere a scenariului:**
  Interacţiunea dintre aplicaţia sistemul Urban Airship serviciul APNs.
- **Actorul principal:**
  Actorul principal este sistemul UA.
- **Precondiţii:**
  Nu există precondiţii ale acestui caz de utilizare.
- **Scenariu de succes:**
  1. Sistemul *UA trimite cu succes date notificării* catre APNS.
- **Scenariu de eşec:**
  2. Sistemul APNs nu este accesibil.
  3. Sistemul APNs nu validează datele.

Serviciul APNs

Serviciul APNs, oferit de Apple, înaintează datele primite de la sistemul Urban Airship către toate dispozitivele care au instalată aplicaţia client şi primeşte de la acesta un identificator unic, numit token, care ajută la regăsirea fiecărui dispozitiv. Aceste două cazuri de utilizare sunt prezentate în următoarea figură.

- **Scută descriere a scenariului:**
  Interacţiunea dintre aplicaţia client şi serviciul APNs.
- **Actorul principal:**
  Actorul principal este aplicaţia client.

- **Precondiţii:**
  Nu există precondiţii ale acestui caz de utilizare.
- **Scenariu de succes:**
  1. Aplicaţia client *trimite* cu succes *token-ul*.
  2. Aplicaţia client *primeşte notificări*.
- **Scenariu de eşec:**
  1. APNs nu validează informaţiile identificatorului unic.
  2. Aplicaţia client nu poate primi notificări.

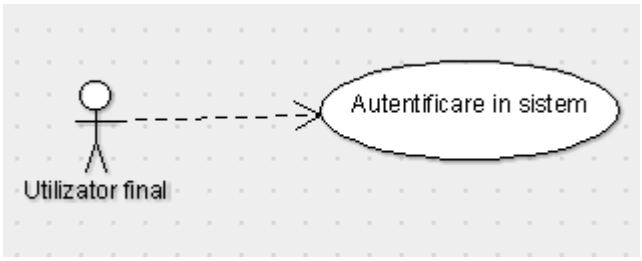


Figură 11. Caz de utilizare: Aplicația sistem - APNs

Aplicația client

Pentru aplicația client diagrama cazurilor de utilizare este mai complexă, prin urmare aceasta a fost împărțită în cinci părți, prezentate în cele ce urmează. Actorul acestor cazuri de utilizare este reprezentat de utilizatorul final, fiind vorba de utilizatorul aplicației client.

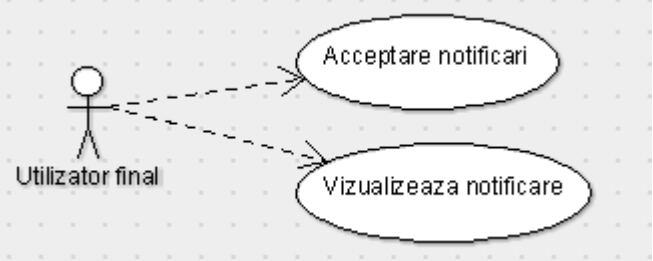
Autentificare utilizatorului este un caz de utilizare care implementează una din cerințele funcționale. Autentificarea este necesară pentru a putea menține integritatea datelor specifice fiecărui utilizator. Acesta este prezentată vizual în următoarea figură.



Figură 12. Caz de utilizare: utilizator final – autentificare

- **Scută descriere a scenariului:**
  Scenariul descrie procesul de autentificare a utilizatorului in sistem.
- **Actorul principal:**
  Actorul principal este utilizatoul aplicaţiei client.
- **Precondiţii:**
  Nu există precondiţii ale acestui caz de utilizare.
- **Scenariu de succes:**
  1. Procesul de *autentificare* se realizează cu succes.
- **Scenariu de eşec:**
  1. Aplicaţia client nu este disponibilă.
  2. Datele de autentificare nu sunt corecte.

În următoarea figură sunt prezentate două cazuri de utilizare, în legătură cu notificările. Pentru a putea primi şi vizualiza notificări, utilzatorul trebuie să accepte primirea acestora. În caz contrar, notificările nu vor fi trimise la dispozitiv, până la modificarea stării pentru acceptarea notificărilor.

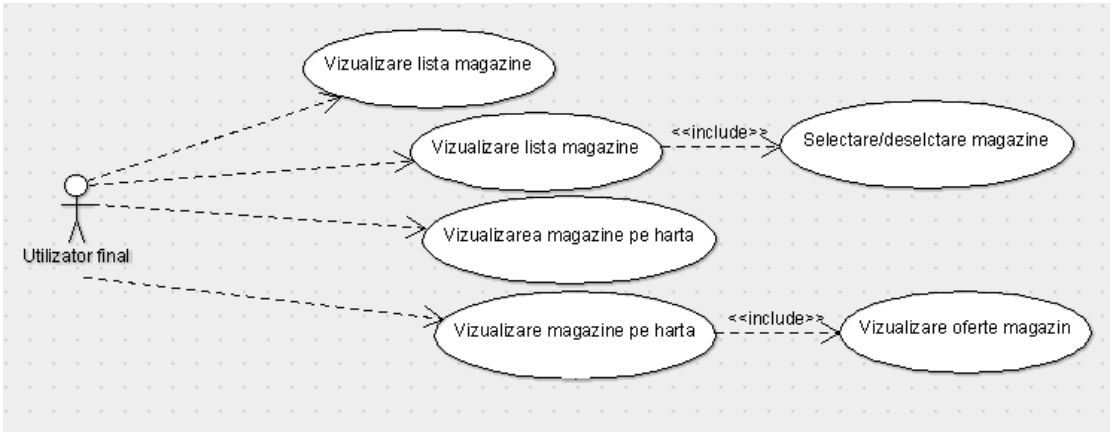


Figură 13. Caz de utilizare: utilizator final – notificari

- **Scută descriere a scenariului:**
  Interacţiunea dintre utilizatorul final şi aplicaţia client.
- **Actorul principal:**
  Actorul principal este utilizatorul final.
- **Precondiţii:**
  Nu există precondiţii ale acestui caz de utilizare.
- **Scenariu de succes:**
  1. Utilizatorul *final acceptă primirea notificărilor*.
  2. Utilizatorul final *vizualizează notificarea primită*.
- **Scenariu de eşec:**
  1. Utilizatorul final nu acceptă primirea notificărilor.

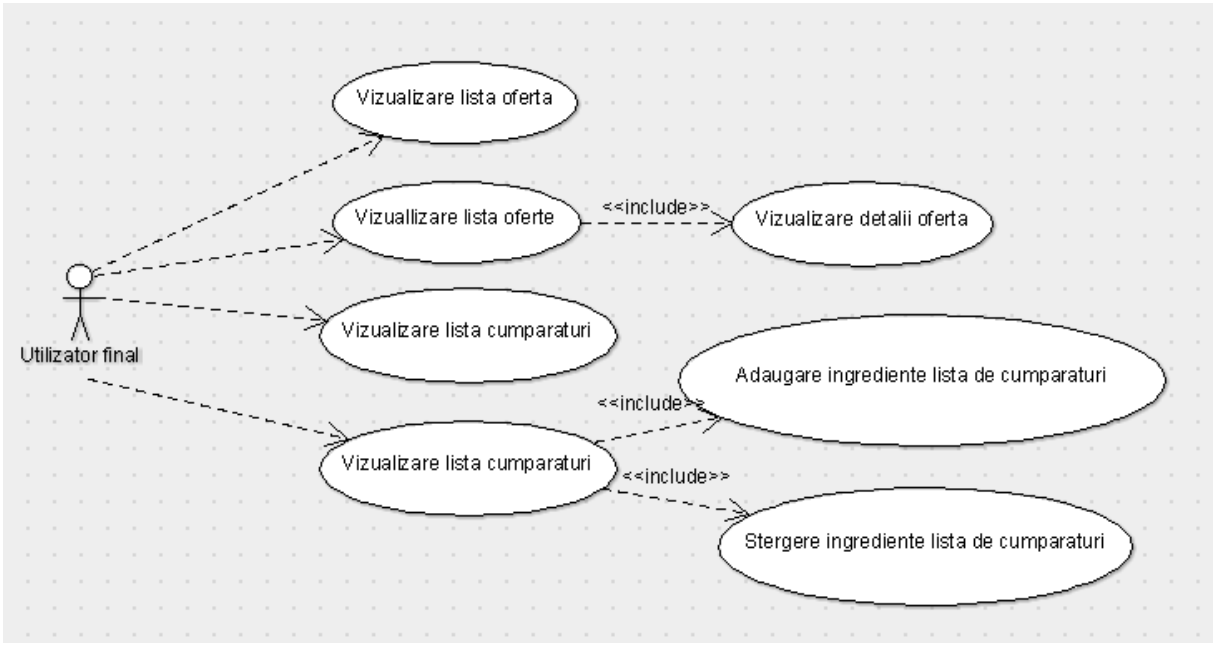
În următoarea diagrama de cazuri de utilizare sunt prezentate cele patru acțiuni pe care le poate face un utilizator, în legatură cu lista de magazine.

- **Scută descriere a scenariului:**
  Interacţiunea dintre utilizatorul final şi aplicaţia client.
- **Actorul principal:**
  Actorul principal este utilizatorul final.
- **Precondiţii:**
  1. Utilizatorul *vizualizează lista de magazine*.
  2. Utilizatorul *vizualizează magazinele pe hartă*.
- **Scenariu de succes:**
  1. *Vizualizare listă magazine*: utilizatorul poate vizualiza lista de magazine primită de la aplicația server
  2. *Selectare/ deselectare magazine*: reprezintă acțiunile pe care le poate face un utilizator pe lista de magazine. Aceste acțiuni sunt dependente de vizualizarea listei de magazine.
  3. *Vizualizare magazine pe hartă*: utilizatorul poate vizualiza pe o hartă magazinele cele mai apropiate de locația sa curentă
  4. *Vizualizare produse magazin*: Utilizatorul poate vizualiza produsele fiecărui magazin individual, acțiune pe care o poate face de pe ecranul cu harta.
- **Scenariu de eşec:**
  Nu există.



Figură 14. Cazuri de utilizare: utilizator final – magazine

În următoare figură sunt prezentate, cazurile de utilizare ale actorului în legatură cu vizualizarea produselor și a listei de cumpărături, precum și a acțiunilor posibile pe lista de cumpărături.



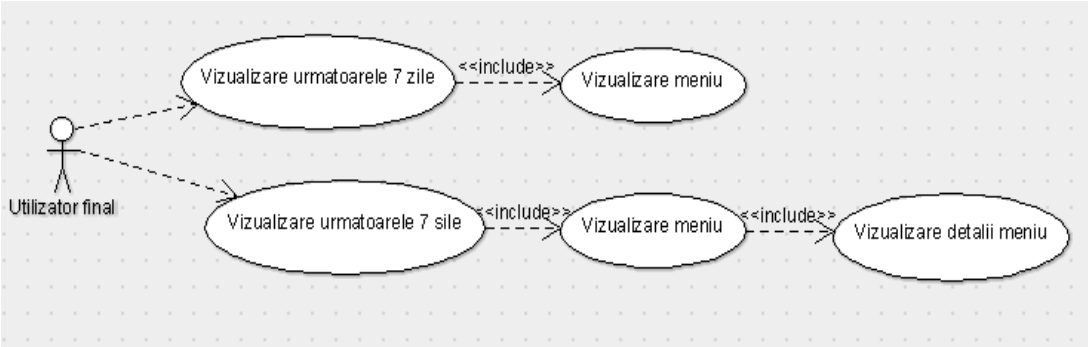
Figură 15. Cazuri de utilizare: utilizator final - produse și lista de cumpărături

- **Scută descriere a scenariului:**
  Interacţiunea dintre utilizatorul final şi aplicaţia client.
- **Actorul principal:**
  Actorul principal este utilizatorul final.
- **Precondiţii:**
  1. Utilizatorul *vizualizează lista de produse*
  2. Utilizatorul *vizualizează lista de cumpărături*.
- **Scenariu de succes:**
  1. *Vizualizare listă produse:* utilizatorul final are posibilitatea de a vedea produsele de la magazinele sectate
  2. *Vizualizare detalii produs*: fiecare produs are detalii suplimentare care nu sunt vizibile în listă. La selectarea unui produs din lista de produse se vor afișa detaliile acesteia

3. *Vizualizare listă de cumpărături*: utilizatorului aplicației mobile poate vizualiza lista de cumpărături, care conține ingredientele adăugate de acesta.
4. *Adăugare ingrediente în lista de cumpărături*: utilizatorul are posibilitatea de a adăuga ingrediente în lista de cumpărături
5. *Ştergere ingrediente din lista de cumpărături*: utilizatorul poate șterge ingredientele adăugate în lista de cumpărături după achiziționarea acestora.

- **Scenariu de eşec:**
  Nu există.

În figura de mai jos sunt prezentate cazurile de utilizare care definesc acțiunile pe care poate să le facă utilizatorul pe meniurile propuse. Fiecare meniu este specific unei zile, astfel inainte de vizualizarea propiu zisă a meniului, utilizatorul trebuie săselecteze ziua pentru care dorește să vadă meniul.



Figură 16. Cazuri de utilizare: utilizator final – meniu

- **Scută descriere a scenariului:**
  Interacţiunea dintre utilizatorul final şi aplicaţia client.
- **Actorul principal:**
  Actorul principal este utilizatorul final.
- **Precondiţii:**
  1. Utilizatorul *vizualizează următoarele şapte zile.*
- **Scenariu de succes:**
  1. *Vizualizare meniu*: acestăacțiune poate fi făcuta de utilizatorul final după vizualizarea listei următoarelor șapte zile
  2. *Vizualizarea detalii meniu*: acestăacțiune este dependentă de vizualizarea listei care cuprinde următoarele șapte zile, precum și a meniului aferent zilei selectate
- **Scenariu de eşec:**
  Nu există.

4.2.2.Schema bloc a sistemului

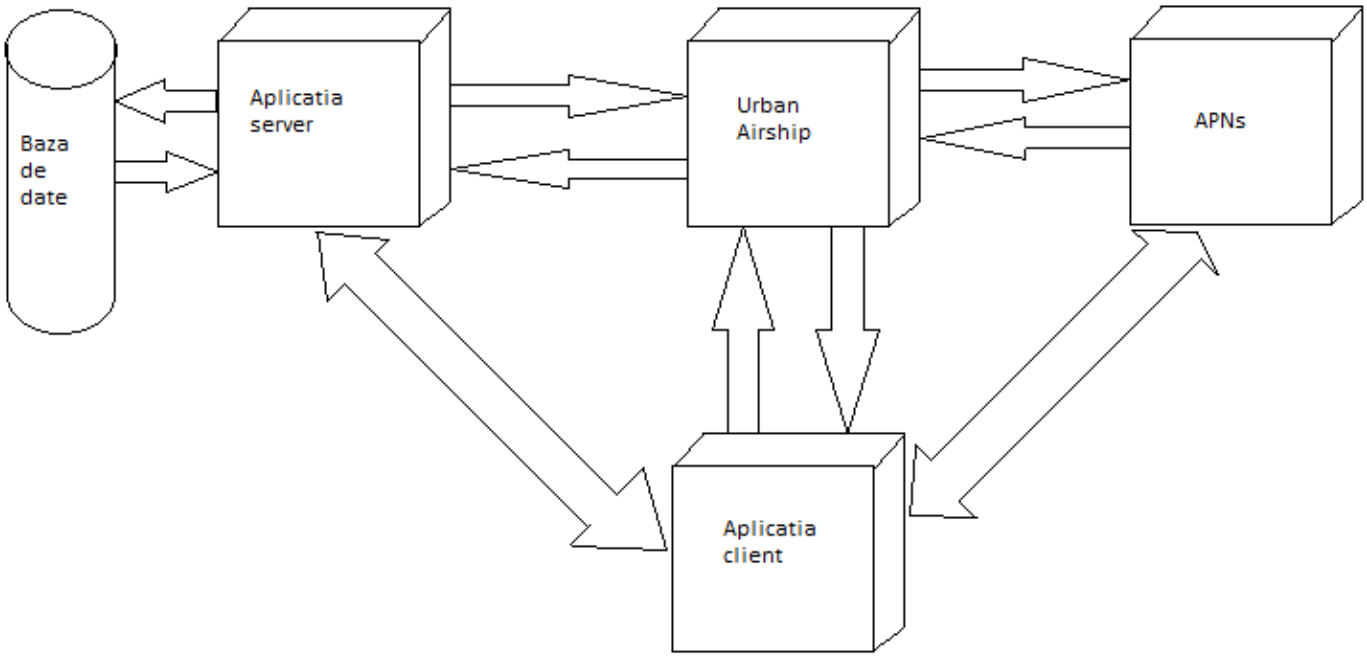
Diagrama bloc este o reprezentare a unui sistem, în care fiecare componentă este reprezentată de un bloc, iar legăturile dintre blocuri sunt reprezentate vizual prin săgeti uni sau bidirecționale.

În următoarea figură este prezentată schema bloc a sistemului, care conține cele patru componente: două proprii (aplicația server și aplicația client) și două ajutatoare (sistemul Urban Airhip și serviciul APNs).

Aplicația server este un sistem care răspunde cererilor adresate de aplicația client, și comunică cu sistemul Urban Airship pentru a face posibilă transmiterea informațiilor.

Comunicarea cu baza de date este bidirecționala, realizându-se atât operații de citire cât și operații de scriere. În ceeea ce privește comunicarea cu sistemul Urban Airship, aici se trimit date legate de mesajul notificărilor.

Aplicația client este singura componentă care interacționeaza direct cu utilizatorul final. Aceasta comunică cu toate celelalte componente în vederea îndeplinirii tuturor cerințelor funcționale ale sistemului. Comunicarea cu sistemul Urban Airhip este minimală, singura interacțiune fiind înregistrarea dispozitivului. Interacțiunea cu serviciul APNs este bidirecțională. De la aplicația client la APNs se realizează doar un singur proces de comunicare, la instalarea aplicației, când se trimite identificatorul dispozitivului. Comunicarea cu aplicația server se face bidirecțional, adresându-se cereri de date, și se așteaptă primirea acestora.



Figură 17. Schema bloc a sistemului

Sistemul Urban Airship comunică cu aplicația server, primind de la aceasta date despre notificări, date pe care le transmite mai departe la serviul APNs. Comunicare cu aplicația client se realizează o singură dată, la momentul instalării aplicație client pe dispozitivele mobile, când sunt primite informații despre identificarea fiecărui dispozitiv.

Serviciul APNs primește de la aplicația client identificatorul fiecărui dispozitiv pe care se instalează aplicația, iar în momentul în care primește cereri de trimitere de notificări de la sistemul Urban Airship, le transmite către toate dispozitivele înregistrate

4.2.3.Arhitectura conceptuală

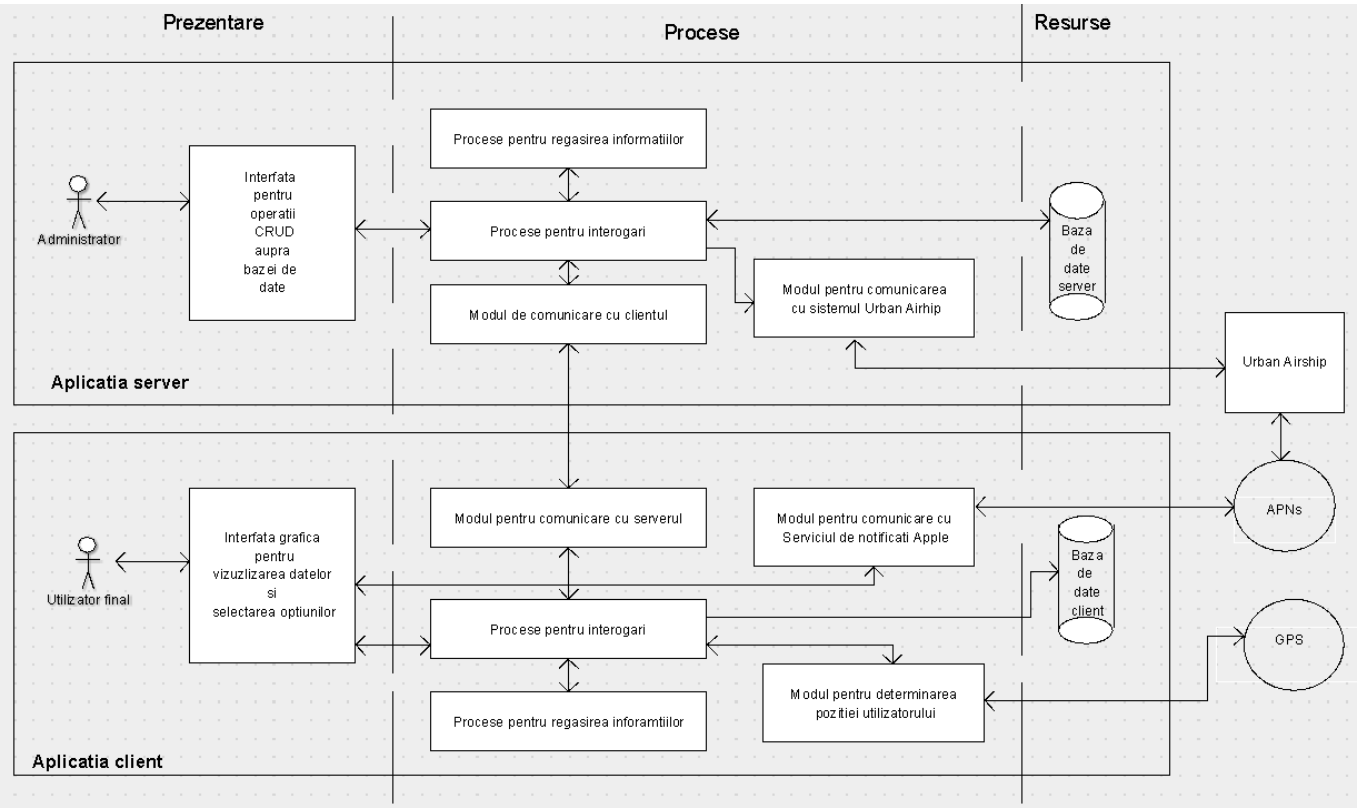
Principalele componente care alcătuiesc întreg sistemul, precum și comunicarea dintre ele sunt prezentate în următoarea figura. Având în vedere, structura sistemuluiîmpărțită pe trei nivele: prezentare, procese și resurse, se vor prezenta principalele componente ale aplicației server și ale aplicației client.

La nivel prezentațional, aplicația server are o interfață pentru operații CRUD asupra bazei de date, reprezentată de interfața grafică a pachetului Xamp, pentru MySql. Aplicația

client, are o interfață grafică care permite vizualizarea datelor, precum și selectarea de către utilizatorului final, a anumitor opțiuni.

La nivel de procese, ambele aplicații au un manager pentru regăsirea informațiilor, și un manager pentru efectuarea interogărilor. Pe aplicația server există un modul de comunicare cu aplicația client, iar pe aplicația client, există un modul asemănator, care îndeplinește funcția de comunicare cu serverul. Aplicația server, conține și un modul pentru comunicare cu sistemul Urban Airship, iar aplicația client conține un modul cu care comunică cu serviciul APNs. Tot pe partea aplicației client, modulul pentru determinatrea poziției utilizatorului, comunică prin intermediul funcțiilor sistem, cu un GSP, care transmite locația curentă a utilizatorului, iar pe baza acestei locații, sunt afișate în conținuare alte informații cerute de utilizatorul final.

La nivelul resurselor, ambele aplicații, conțin o bază de date, care oferă persistența datelor. Astfel, aplicația client, poate funcționa în mod offline, afișându-se rezultatele bazate pe ultima locație cunoscută a utilizatorului.



Figură 18.Arhitectura conceptuală

În concluzie, toate aceste componente, precum și procesele de comunicare dintre acestea, formează întreg sistemul și îndeplinesc toate cerințele funcționale și non funcționale ale sistemului.

Capitolul 5. Proiectare de detaliu și implementare

5.1.Proiectare de detaliu

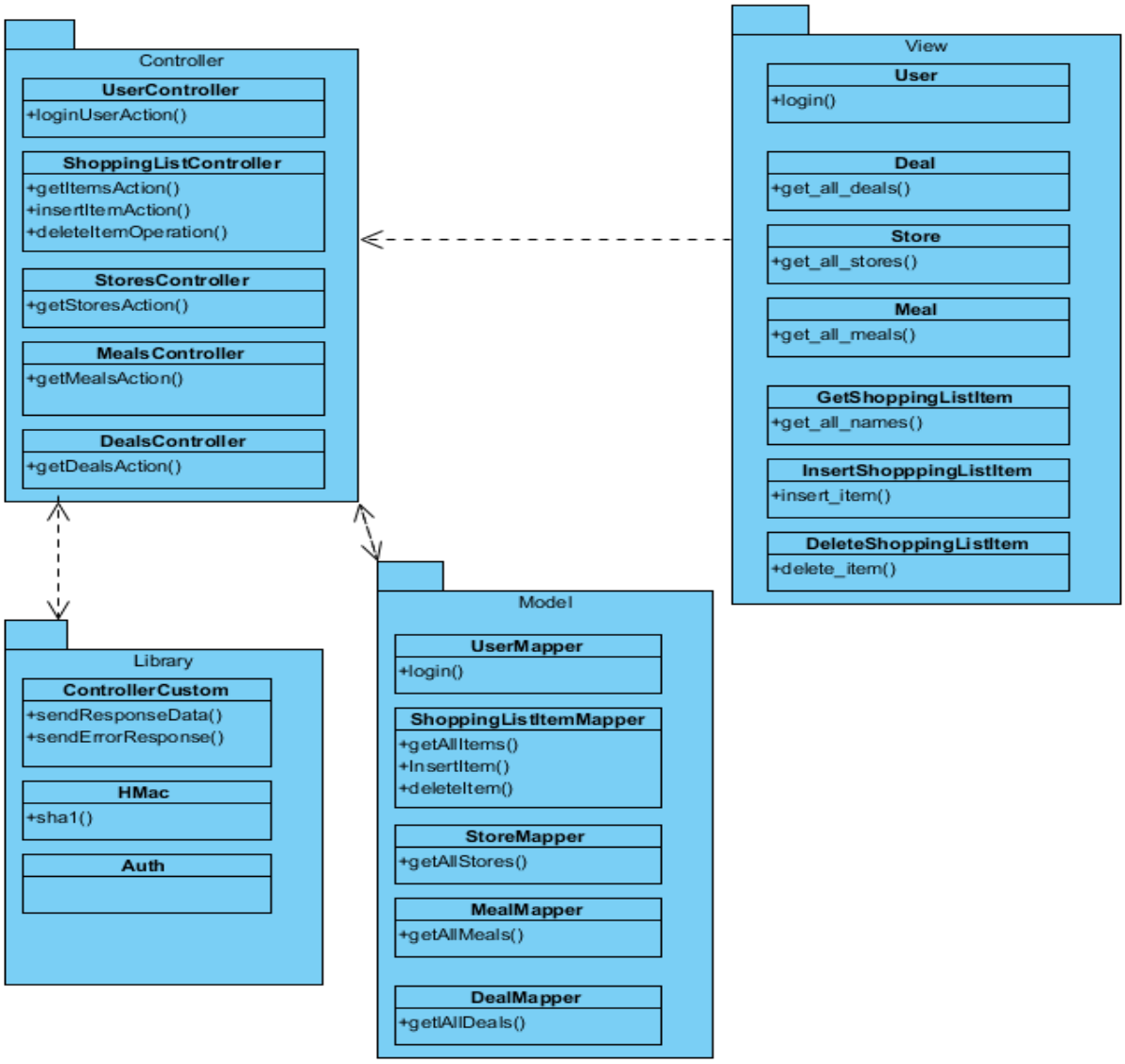
În acest capitol se vor prezenta în detaliu structurile celor două componente principale ale sistemului, respectiv aplicația client și aplicația server, precum și modalitățile de comunicare între acestea și interfetele cu cele 2 sisteme ajutătoare. Pe lângă toate acestea se mai adăugă și componenta reprezentând baza de date, descrisă și ea în cele ce urmează.

5.1.1.Structura

În cele ce urmează sunt descrise și ilustrate structurile celor două componente ale sistemului, precum și bazele de date aferente lor.

Aplicația server

Aplicația server este compusa din patru pachete și anume: Model, View , Controller și Library, replezentând compomenetele șablonului arhitectural MVC, precum și o un pachet ajutător. În următoare figură sunt prezentate cele patru pachete, precum și legăturile dintre ele.



Figură 19. Diagrama de pachete - aplicație server

După cum se poate observa fiecare din cele patru pachete îndeplineste un anumit rol, prezentat în cele ce urmează:

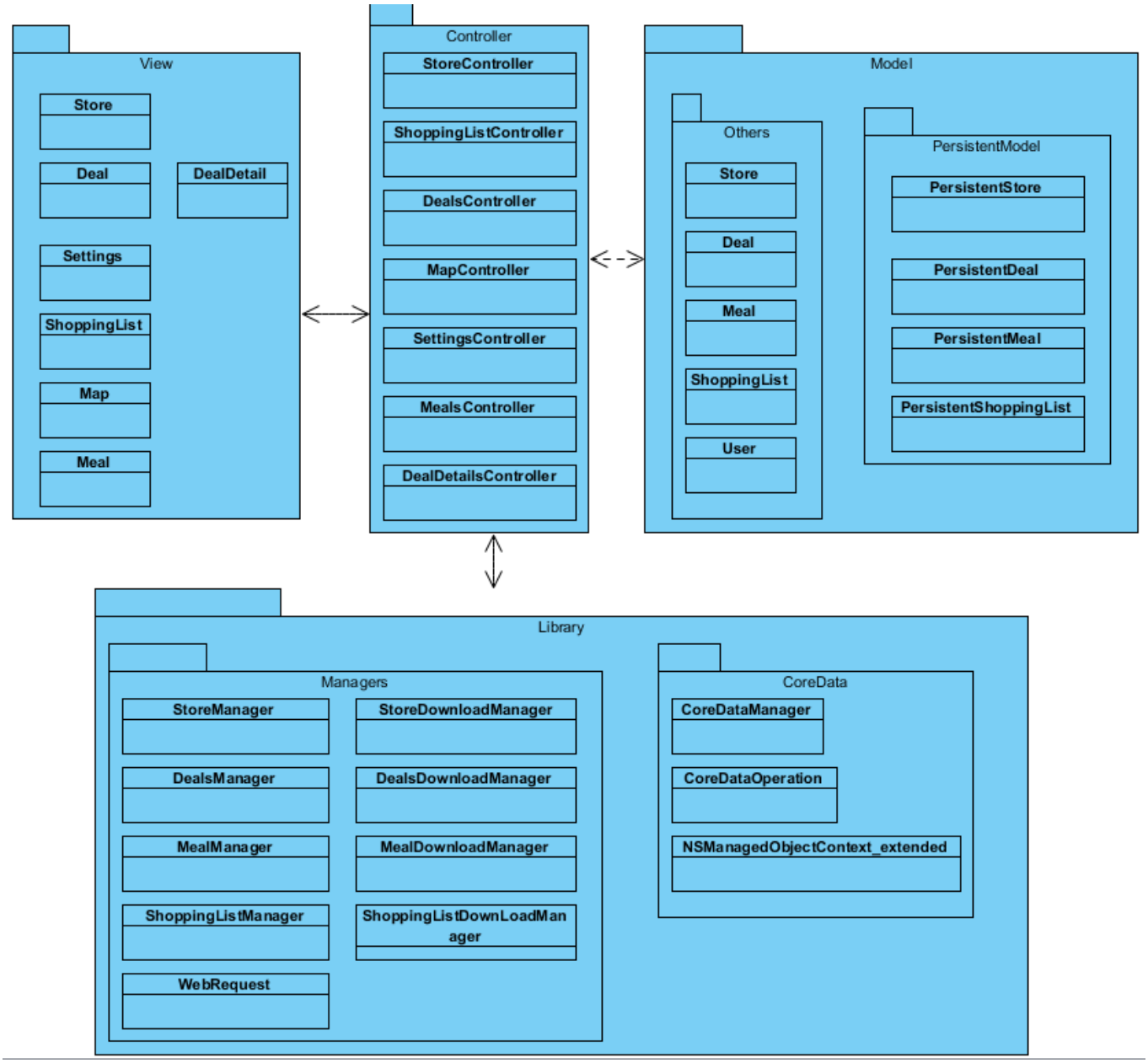
- **Controller** : acest pachet conține cinci clase specifice fiecărui model și fiecărei vederi, făcând legătura dintre acestea. Fiecare clasă îndeplineste cel puțin o acțiune venită de la una din clasele din pachetul View, transmite mesajul către modulul corespunzator și apoi, pe baza răspunsului oferit de model, trimite către utilizatorul final răspunsul așteptat. În acest caz, utilizatorul final este aplicația client.
- **Model:** În acest pachet sunt cuprinse cinci clase care reprezintă modelele aplicație, obiecte mapate peste tabelele din baza de date, și un pachet numit Mapper, care conține cinci clase de mapare, care instanțiază câte un obiect model și efectuează operații pe baza lui.
- **View:** acest pachet conține șapte clase, fiecare putând reprezenta o vedere (în acest caz o vedere este un obiect virtual care înaintează cererea venită la de aplicația client, și apelează acțiunea necesară din controlerul aferent) .
- **Library**: în acest pachet sunt incluse trei clase ajutătoare, care realizeară autentificarea, calculează semnătura și formatează și trimite răspunsurile cererilor.

Aplicația client

La fel ca și aplicația server, aplicația client este formată din patru mari pachete. Conform șablonului arhitectural Model View Controler, avem trei pachete cu aceleasi nume și un pachet Library care conține mai multe pachete. În continuare vor fi prezentate cele patru pachete:

- **Model** : acest pachet conține două subpachete:
  - **PersistentModel**: acest pachet înglobează modelele persistente din aplicație, mapate peste entitățile din modelul general al aplicație. Clasele acestui pachet moștenesc din NSManagedObjectModel.
  - **Others**: conține aceleași obiecte ca și pachetul anterior, singura diferența fiind că ele sunt obiecte generice, utilizate în restul aplicației, pentru a putea fi modificare temporar, fără a fi salvate în baza de date locală. Aceasta abordare face utilizarea obiectelor model posibilă pe mai multe fire de execuție. Acest pachet conține și modelul utilizatorului.
- **Controller**: în acest pachet există pentru fiecare ecran din aplicație câte un controler. Aceste controlere interacționează cu obiecte formate din clasele din pachetul Library, subpachetul Managers, și cu seturi de modele, pentru a afișa utilizatorului informațiile necesare. Fiecare controller are o referință la vederea corespunzătoare.
- **View**: în acest pachet sunt cuprinse vederile folosite în aplicație, și reprezintă interfața utilizator a aplicație client. Fișierele din acest pachet sunt de tip .xib și reprezintă o serie de elemente grafice organizate sub forma unui Xml.
- **Library**: acest pachet două pachete principale:
  - **Managers** : în acest pachet sunt păstrate obiectele singleton care se ocupă de managmentul datelor și de procesul de comunicare între server și client.
  - **CoreData**: conține o suită de clase care asigură persistența datelor.

În următoare figură este prezentată diagramade pachete a aplicației client.

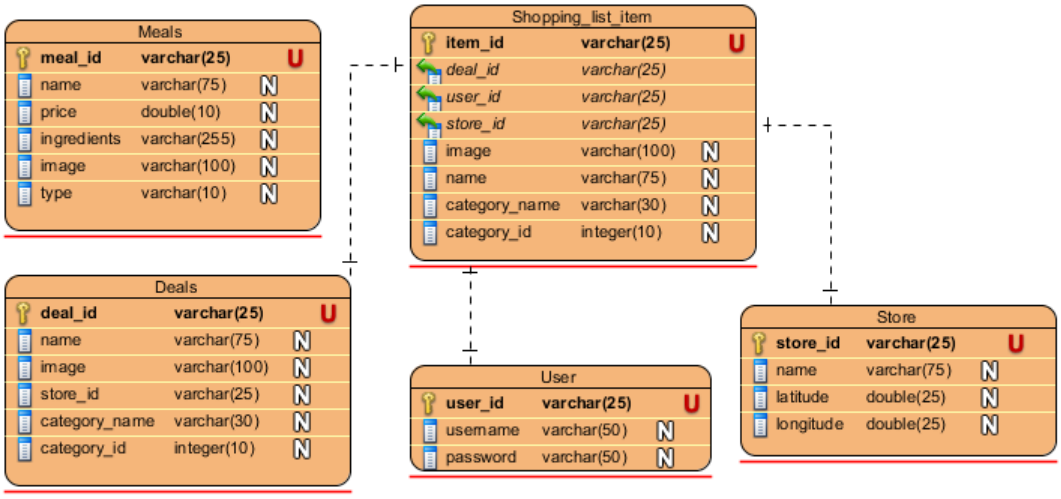


Figură 20.Diagramă de pachete - aplicația client

Baza de date server

Baza de date este formată din cinci tabele, structurate astfel încât să modeleze cât mai bine cerințele întregului sistem și să poată să răspundă cât mai bine la interogările principale la care va fi supusă.

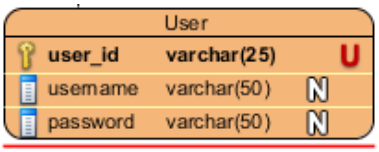
Tabelele bazei de date sunt: *User*, *Deals*, *Meals*,*Shopping_list_item* și *Store*. Figura de mai jos prezintă cele cinci tabele.



Figură 21.Diagrama bazei de date - aplicația server

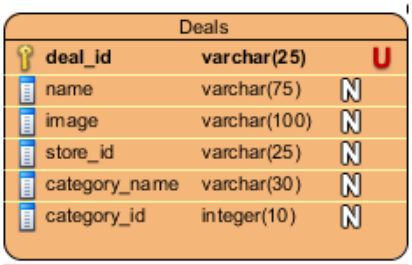
Tabelele reprezintă entitățile din modelul entitate-relație (ER) pe baza căruia a fost modelată lumea reală a sistemului.În continuare sunt prezentate pe rând toate tabelele bazei de date, insistându-se asupra atributelor și a tipurilor de date folosite, precum și asupra utilizabilității tabelului în sitem.

Tabla *User* repezintă ca și entitate un utilizator din lumea reală a sistemului, care dorește să folosească acest sistem. Atributele acestei entități sunt numele de utilizator, parola și identificatorul unic. În acest context, tabela *User*are ca și cheie primară câmpul *user_id*, de tip varchar, iar câmpurile *username* și *password*, mapează numele utilizatoului și parola acestuia, ambele câmpuri fiind de tip varchar. În următoare figură este prezentată structura tabelului.



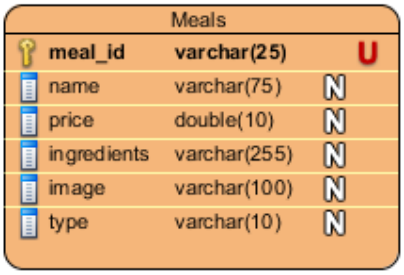
Figură 22. Tabel utilizator

Tabla *Deals* reprezintă maparea unui produs ale unui magazin sau a unui producător din lumea reală și are ca și atribuite: identificator, nume, pret, imagine, identificatorul magazinului, numele categoriei și identificatorul acesteia. Cheia primară a acestui tabel este *deal_id* și este de tip varchar. Numele,prețul și imaginea, sunt mapate pe câmpurile de tip varchar: *name,price,image*. Câmpul imagine conține un url către o imagine statică salvată pe server.Identificatorul magazinului este mapat pe câmpul de tip varchar, *store_id*, iar numele și identificatorul categoriei din care face parte produsul sunt mapate pe câmpurile de tip varchar si integer *category_name*, respectiv *category_id*. În următoare figură este prezentată structura tabelului.



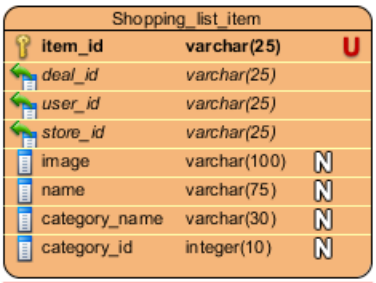
Figură 23.Tabel produse

Tabela *Meals* reprezintă maparea entității meniu din lumea reală. Atributele acestei entități sunt: identificator, nume, ingrediente, imagine și tip. Identificatorul și numele se mapează pe câmpurile *meal_id* și *name*, ambele de tip varchar. Câmpul varchar *ingredientes* reprezintă o înșiruire de ingrediente separate de caracterul virgulă, imaginea este mapată pe câmpul *image*, iar tip-ul elementului de meniu, corespunde câmpului *type* de tip varchar și reprezintă ce fel de tip de mâncare este acest element (vegetarian, paleo, mediteranean). În următoare figură este prezentată structura tabelului.



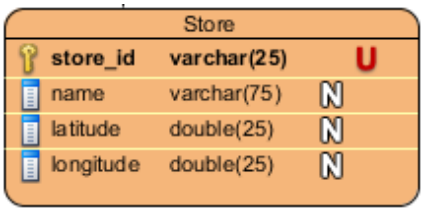
Figură 24.Tabel element meniu

Tabla *Shopping_list_item* reprezintă entitatea unei intrări din baza de date. Acesta entitate este identificată unic de câmpul *item_id* de tip varchar. Un alt atribut al acestei entități este identificatorul utilizatorului, corespondent câmpului varchar *user_id*. Oferta este identificată prin câmpul *deal_id*, de tip varchar. Celelate atribute ale acestei entități sunt : identificatorul produsului, numele și imaginea atașata acestuia, numele și identificatorul categoriei, precum și identificatorul magazinului. Aceste atribute sunt mapate pe câmpurile de tip varchar sau integer: *deal_id*, *name*, *image*, *category_name*, *category_id* și *store_id*. în următoarea figură se poate observa structura acestui tabel.



Figură 25. Tabel elemente lista de cumpărături

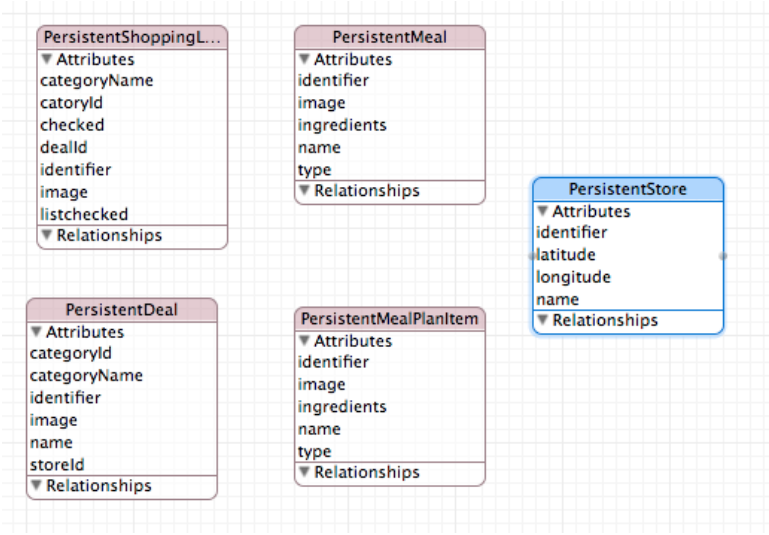
Tabela *Store* reprezintă un magazin fizic, și se indentifică prin următoarele atribute: identificartor, nume, latitudine și longitudine. Cheia primară a acestui tabel este *store_id* și reprezintă identificatorul unic al magazinului, fiind de tipul varchar. Numele se mapează pe câmpul, cu tipul varchar, *name*, iar latitudinea și longitudinea se mapează pe câmpurile de tip numere cu virgulă*latitude* și *longitude*, reprezentând poziția fiecărui magazin. În următoare figură este prezentată structura tabelului.



Figură 26.Tabel magazin

Baza de date client

Baza de date a aplicației client este într-un fel similară bazei de date din aplicația sever. În continuare vor fi prezentate entitățile Core Data care alcătuiesc modelul general al soluției de persistență a aplicație client. În următoare figură se pot observa entitățile și atributele acestora.



Figură 27.Diagrama bazei de date - aplicația client

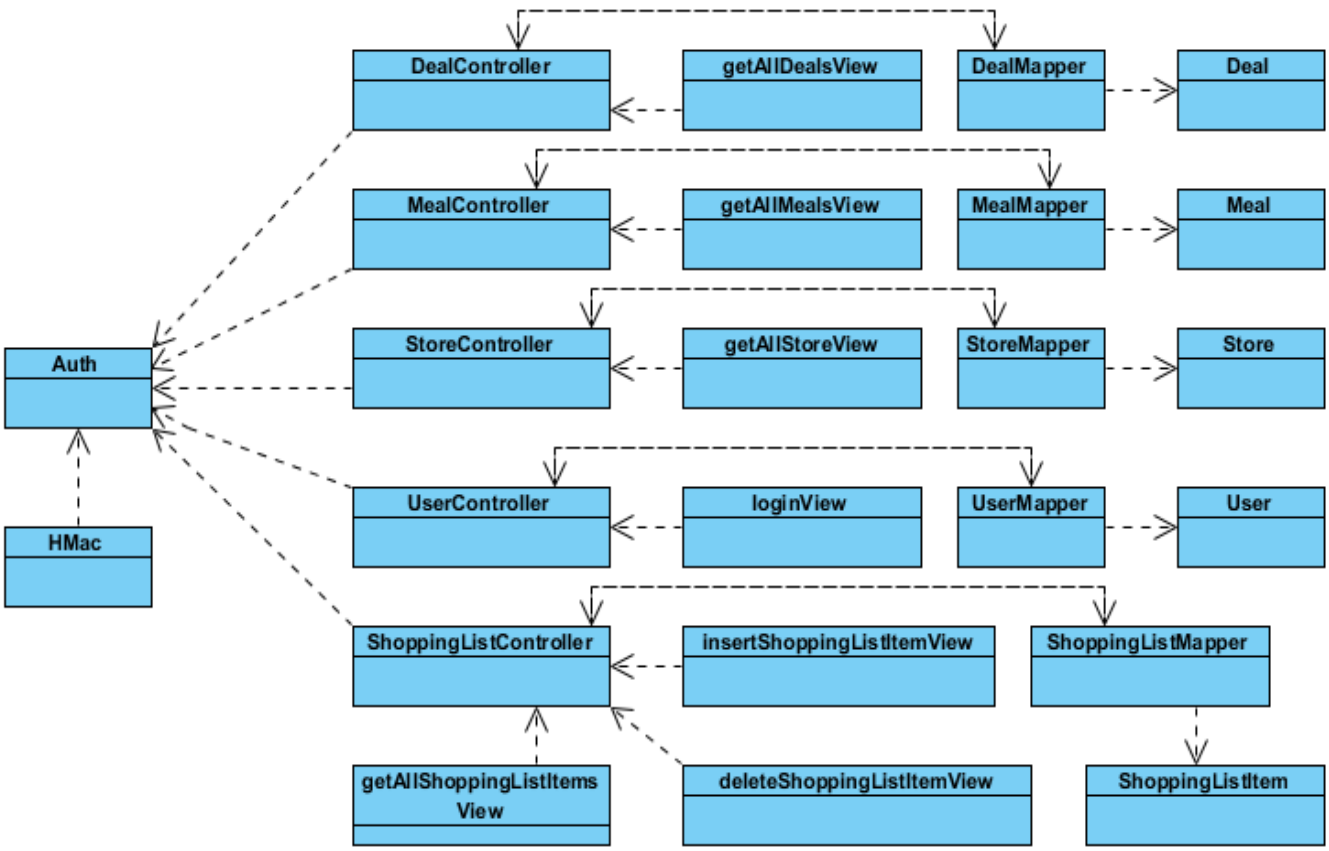
5.1.2.Diagrame de clase

În această secțiune vor fi prezentate o serie de diagrame de clase pentru principalele și cele mai importante module ale sistemului.

Aplicația server

Aplicația server, care este proiectată pentru a oferi date aplicației client, are o serie de cerințe care trebuie implementate: autentificarea utilizatorului, returnarea listelor de produse, meniuri și magazine, precum și a listei de cumpărături. Totodată trebuie implementate și metode pentru adăugarea și ștergerea înregistrărilor din baza de date.

În următoare figură este prezentată diagrama de clase a aplicației server.



Figură 28. Diagrama de clase - aplicația server

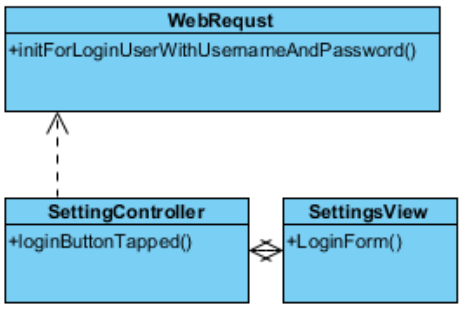
Pentru a îndeplini cerințele funcționale specificate în Capitolul 2, aplicația client utilizează o serie de clase care comunică între ele. Clasa *Auth* , folosind algoritmnul *HMac* din clasa cu acelasi nume, are funcția de a verifica dacă semnatura trimisă de alicația utilizator este validă. Clasa vedere *getAllDealsView* primeste cererea de returnarea a produselor și o transmite clasei DealController, care mai departe inaintează cererea la Clasa *DealMapper*, care folosind clasa model *Deal*, execută interogarea și întoarce răspunsul la clasa controler. Analog, în cazul cererii pentru obiectele din meniu, clasa *getAllMealsView* preia cererea de la aplicația client, o inaintează spre clasa *MealController*, care pe baza clasei *MealMapper* și a clasei model *Meal*, primeste răspunsul și il transmite aplicației client. Pentru a primi lista de magazine, aplicația client trimite o cerere către clasa *getAllStoresView*, care o transmite clasei *StoreController*. De aici cererea ajunge la clasa *StoreMapper*, care împreună cu clasa model Store, gaseste lista de magazine dintr-o raza fixa fata de pozitia utilizatorului și o returneaza clasei controlor. Pentru autentificarea utilizatorului, clasa vedere *loginView*, preia cererea de la aplicația client, o transmite controlerului, repreyentata de clasa *UserControler*. Datele de autentificare sunt transmise clasei *UserMapper*, care pe baza tabelului definit de clasa model *User*, returneaza identificatorului utilizatorului, în cazul în care acesta există în baza de date. în ceea ce privește achizitionarea datelor pentru lista de cumpărături, vederea *getAllShoppingListItemsView*, preia cererea de la aplicația client, o transmite clasei controler, *ShoppingListController*, care înaintează cererea sper clasa *ShoppingListMapper*, care împreună cu clasa model *ShoppinListItem*, determina lista de rezultate care trebuie intoarse. Pentru inserarea sau ștergerea unei înregistrari din baza de date, din tabelul listei de cumpărături, clasele vedere *insertShoppingListItemView*, respectiv *deleteShoppingListItemView* , trimit cererea primita mai departe la clasa controler, *ShoppingListController*, care o înaintează clasei *ShoppingListMapper*. Aici se construieste interogarea și după executia ei, rezultatul este intors controlerului, care raspunde aplicației

client. Asadar, utilizarea acestor clase, precum și procesul de comunicare dintre ele, fac posibila indeplinira cerințelor funcționale.

Aplicația client

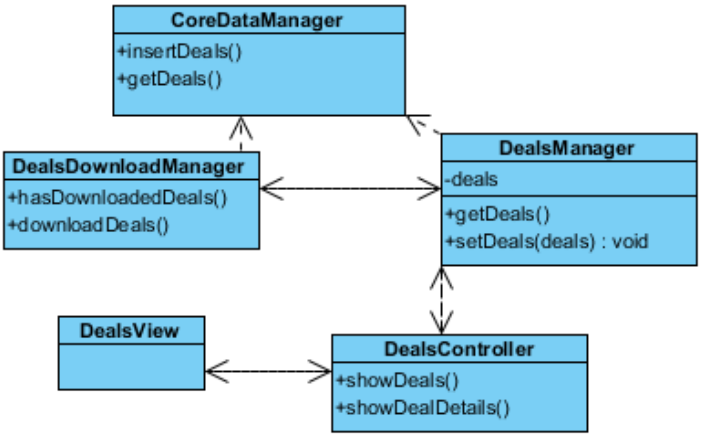
În aplicația clientului proiectată pentru dispozitive mobile, există o serie de cerințe funcționale care trebuie implementate: autentificare, vizualizarea magazinelor, a produsele și a detaliilor acestora, a meniului, precum și vizualizarea listei de cumpărături și adăugarea și ștergerea elementelor din aceasta. În acest sens sunt prezentate mai jos, cu ajutorul diagramelor, clasele ce ajută la realizarea acestor cerințe precum și relațiile dintre acestea.

Înainte de a realiza orice fel de operație, utilizatorul aplicației trebuie să se autentifice în sistem.Pentru autentificare, utilizatorul completează câmpurile din clasa *SettingsView*, iar apoi acțiunea de autentificare este transmisă clasei *SettingsController*. Deoarece acțiunea de login conditionează orice altă acțiune, implementarea cererii de logare se face în controler. Aici se instanțiază clasa *WebRequest*, cu constructorul pentru logarea utilizatorului. După ce operația s-a incheiat cu succes, utiliyatorul este directionat spre ecranul cu produse. Clase care fac aceast operație posibila sunt: *SettingsController* și vederea *Settings*. În următoare figură sunt prezentate vizual aceste clase.



Figură 29.Diagramă de clase : client - login

Vizualizarea produselor este un proces ce implică descărcarea acestora și afișarea în vederea *DealsView*.Clasele care comunică pentru realizarea acestei cerințe funcționale sunt reprezentate în următoare figură, la fel și comunicarea dintre acestea.

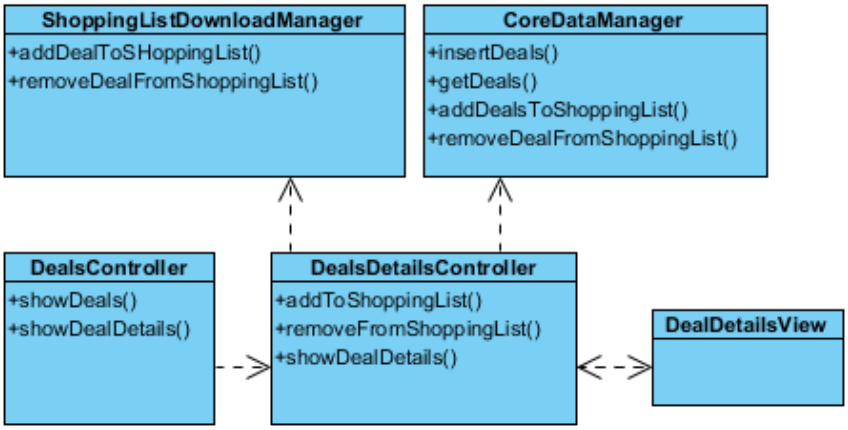


Figură 30.Diagramă de clase : client – produse

Pentru a vizualiza produsele, o serie de clase comunică și fac posibilă această acțiune. La deschiderea aplicației, dacă utilizatorul este logat, clasa *DealsController* trimite o cerere de regăsire a produselor clasi *DealsManager*. Acestă clasă, dacă în sesiunea curentă, produsele au fost descărcate, le cere clasei *CoreDataManager*, iar dacă nu, cere clasei DealsDownloadManager să le descarce de pe server. După ce acestea au fost descărcate, ele

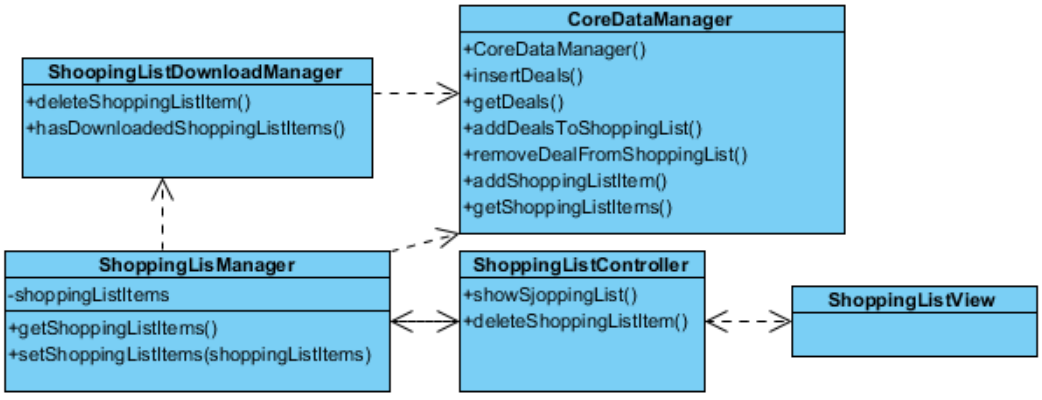
sunt salvate în baza de date locală, și un mesaj de confirmare este trimis clasei *DealsManager*, care înaintează acest mesaj mai departe, spre clasa *DealsController*, împreună cu șirul produselor.

Vizualizarea detaliilor unui produs se face în clasa vedere *DealDetailsView*. La fiecare apariție pe ecranul dispozitivului a clasei ce reprezintă interfața utilizator pentru afișarea detaliilor produselor, instanța clasei *DealsDetailsController*, afișează datele primite de la clasa *DealsController*. Există două acțiuni care se pot face în aceasta clasă: adăugarea și ștergerea unui produs din lista de cumpărături. Acest lucru se poate realiza prin comunicarea clasei controler cu managerul de download a listei de cumpărături și cu clasa *CoreDataManager* care asigura persistența datelor.Pentru vizualizarea detaliilor produselor, s-au folosit clasele descrise în figura următoare.



Figură 31.Diagramă de clase - detalii produse

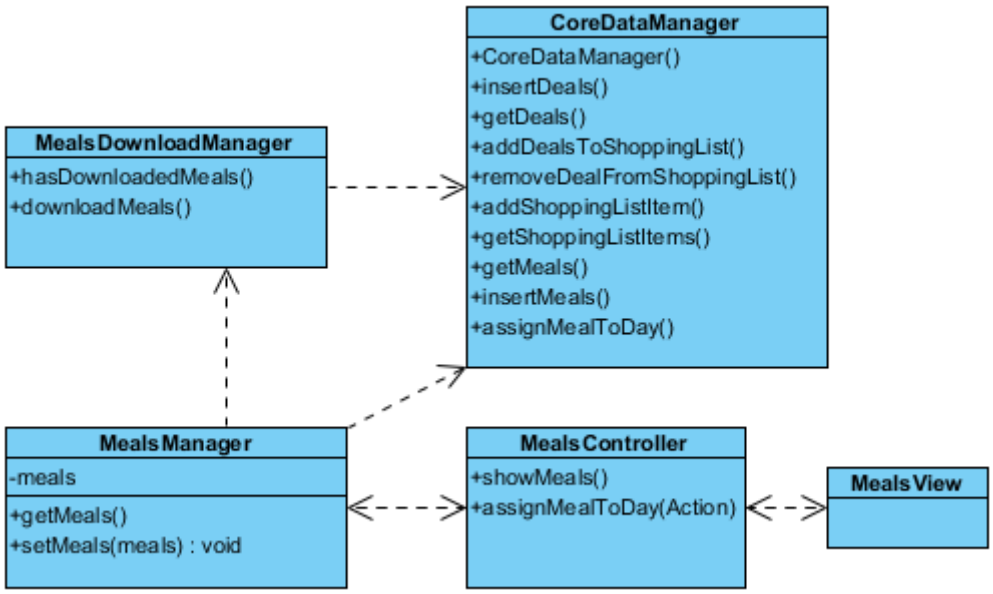
Pentru implementarea cerinței funcționale care descrie afișarea listei de cumpărături, mai multe clase comunică între ele.Clasa *ShoppingListController* cere datele de la clasa *ShoppingListManager*, care în cazul în care ele au fost descărcate în sesiunea curentă le cere clasei *CoreDataManager*. În caz contrar, datele sunt descărcate de clasa *ShoppingListDownloadManager* și apoi înaintate spre clasa controler, care le afișează utilizatorului utilizând clasa *ShoppigListView*. Tot aici se poate șterge o intrare din lista de cumpărături. Clase sunt reprezentate grafic în următoarea figură.



Figură 32. Diagramă de clase - lista de cumpărături

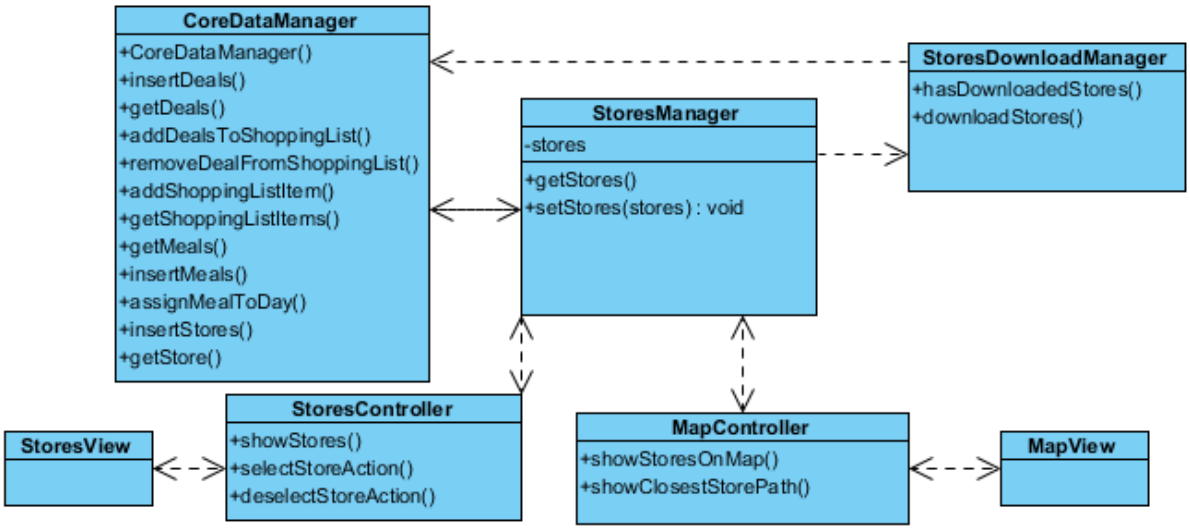
Afișarea meselor din meniu se face în clasa *MealsView* prin intermediul clasei *MealsController*. Dacă datele au fost deja descărcate în sesiunea curentă de utilizare, ele sunt cerute clasei *CoreDataManager*. În caz contrar, clasa *MealsManager* cere clase *MealsDownloadManager* să le descarce, iar după ce operațiunea s-a terminat, acestea sunt afișate utilizatorului. Aici, utilizatorul are posibilitatea să asigneze mesele prezentate unei

zile.Astfel, se îndeplinește cerința funcțională despre personalizarea meniului. În următoare figură sunt reprezentate grafic aceste clase și relațiile dintre ele.



Figură 33. Diagramă de clase - afișarea meniului

În final, afișarea magazinelor în lista de magazine, dar și pe hartă, împreună cu cel mai scurt drum dintr poziția utilizatorului și cel mai apropiat magazin, sunt repreznetate în următoarea figură. Clasele care fac posibilă îndeplinirea acestei cerințe funcționale, precum și selectarea și deselectarea magazinelor preferate, sunt afișate în clasa *StoreView* pentru lista de magazine și în clasa *MapView* pentru reprezentarea magazinelor pe harta. Clasa *StoresDownloadManager* se ocupă cu descărcarea listei de magazine de pe server, iar clasa CoreDataManager, salvează aceasta lista pentru a oferi persistența. Clasa *StoresManager*, inițiază, dacă este cazul, și păstrează lista de magazine.



Figură 34. Diagrama de clase - afișarea magazinelor în lista și pe harta

5.2.Implementare

5.2.1.Clase și componente

În continuare va fi prezentată organizarea fiecărei componente a sistemului, împreună cu cele mai importante aspecte legate de implementarea claselor şi comunicarea dintre acestea.

Aplicația server

Aplicația server are rolul de a furniza informații, precum și de a face modificări pe baza de date, în funcție de cererile aplicației client.

Prima cerință funcțională a aplicației server este verificarea autenticității cererilor pentru îndeplinirea cerinței de securitate. Această cerință este implementată folosind clasele *Auth* și *Hmac*. Clasa *Hmac*, conține un algoritmul de criptare pa baza unei chei și a unui mesaj, descris în cele ce urmează.

```
public static funcțion sha1($key, $message) {
                    $blocksize = 64;
                    $opad = str_repeat(chr(0x5c), $blocksize);
                    $ipad = str_repeat(chr(0x36), $blocksize);
                    if (strlen($key) < $blocksize) {
                            $key = $key . str_repeat(chr(0), ($blocksize - strlen($key)));
                    }
                    $hmac = sha1(($key ^ $opad) . sha1(($key ^ $ipad) . $message, true), true);
                    return base64_encode($hmac);
}
```

Cheia folosită este cunoscută doar în aplicația server și aplicația în aplicația client. Astfel, orice altă aplicație ca utilizator încearcă să folosească motodele publice din aplicația server, va primi un răspuns de eroare. În clasa Auth, se verifică dacă există headerele necesare, în caz afirmativ se verifică veridificitatea autentificării. Mesajul este format prin concatenarea uri-ului cu constata client și data la care a fost inițiată cererea. Dacă cererea este semnată corespunzator, atunci se merge mai departe, la calcularea și returnarea răspunsului. Aceste verificari sunt reprezentate în urmatorul bloc de cod.

```
$message = $_request->getMethod() . $request_uri . '/' . $header_client . $header_date;
// check if the request has been signed correctly
if ($header_auth != DK_Crypt_Hmac::sha1($api_key, $message)) {
   $response->sendErrorResponse(400,
        'InvalidSignature', 'Authentication failed because the provided signature is not valid');
}
```

În ceea ce privește implementarea celorlalte cerințe funcționale, aceasta este facută cu o combinație de clase din cele trei pachete: *Model*, *Controler* și *Vedere*. Pentru returnarea listei de magazine din proximitatea utilizatorului aplicației mobile, cererea adresată de aceasta este este primită de clasa get-all-stores, și transmisa unui instanțe a clasei *StoreController* prin metoda numita *getAllStoresAction*(). Aici se crează o instanța a clasei *StoreMapper*, se extrag parametri post din cerere și inițializează variabilele latitudine și longitudine(utilizând valorile din câmpurile *lat* și*long* din parametrii post) și se apelează metoda *getAllStores* cu aceste

variabile. După calcularea rezultatului în variabila response, aceasta este transmisă aplicației client.În cele ce urmează este prezentat codul acestei metode din clasa *StoreController*.

```
public function getAllStoresAction() {
	$storeMapper = new Application_Model_StoreMapper();
	$post = $this->getRequest();
	$lat = $post->getParam('lat');
	$long = $post->getParam('long');
	$result = $storeMapper->getAllStores($lat,$long);
	$this->sendResponseData(200,$result);
}
```

În clasa *StoreMapper*, se instanțiază clasa *Store*, și se interoghează tabelul aferent, pentru a obține lista cu toate magazinele din baza de date. După acest pas, fiecare înregistrare care reprezintă un magazin fizic, este procesată prin calcularea distanței de la magazin, la locația utilizatorului și transmisă prin parametri. Dacă acestă distanță este mai mică decât o constată prestabilită, care este setată la valoarea 1 și reprezintă raza de 1 kilometru în care magazinele trebuie să se afle, atunci înregistrarea magazinului este adăugata într-un sir, care la final va fi returnat apelantului. Calcularea distanței dintre două pozitii reprezentate prin latitudine și longitudine, este prezentată în urmatorul bloc de cod.

```
private funcțion distance($lat1, $lng1, $lat2, $lng2, $miles = FALSE) {
		$pi80 = M_PI / 180;
		$lat1 *= $pi80;
		$lng1 *= $pi80;
		$lat2 *= $pi80;
		$lng2 *= $pi80;

		$r = 6372.797; // mean radius of Earth în km
		$dlat = $lat2 - $lat1;
		$dlng = $lng2 - $lng1;
		$a = sin($dlat / 2) * sin($dlat / 2) +
					cos($lat1) * cos($lat2) * sin($dlng / 2) * sin($dlng / 2);
		$c = 2 * atan2(sqrt($a), sqrt(1 - $a));
		$km = $r * $c;
		return ($miles ? ($km * 0.621371192) : $km);
}
```

Procesele pentru returnarea produselor, a listei și a obiectelor din meniu sunt asemănătoare, cu excepția calculării distanței. Condițiile sunt include direct în interogarea bazei de date. Pentru returnarea listei de cumpărături pentru un utilizator, se extrage din headerele primite odată cu cererea, din câmpul *user_id*, identificatorul unuic al utilizatorului și se contruiește interogarea de selecție cu condiția ca identificatorul utizatorului din înregistrările din baza de date să fie egal cu identificatorul transmis prin cerere.În următorul bloc de cod , se poate vedea interogarea pentru returnea listei de cumpărături.

```
public funcțion getAllShoppingListItems($headers) {
	foreach ($headers as $name => $value) {
	  if ($name == 'user_id') {
	    $select = $this->_dbTable->select()->from('ShoppingListItem')
				->where('user_id = ?', $value)
				  ->query()
```

```
                                    ->fetchAll();
            }
        }
        return $select;
}
```

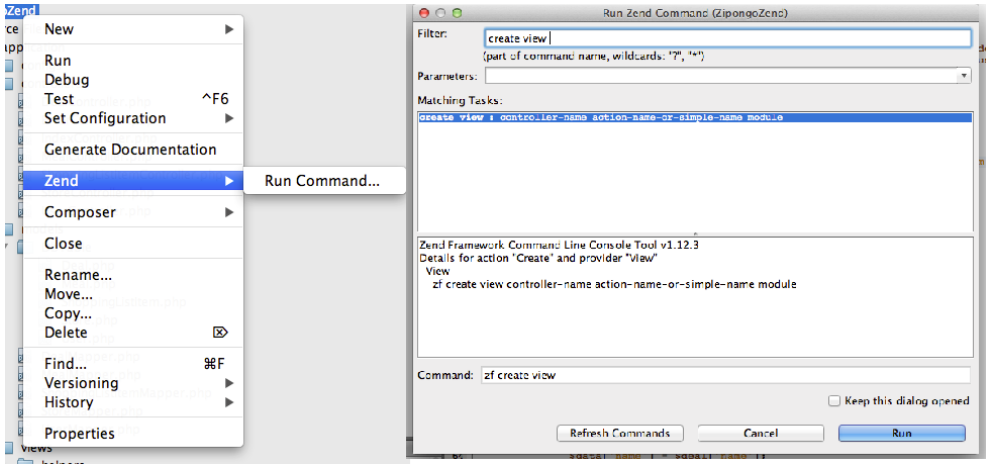
O altă cerință de funcționare este inserarea și ștergerea din tabelul ShoppingListItems a înregistrărilor, în urma acțiunilor utilizatorului. Pentru inserarea în tabel, din headerele primite împreună cu cererea de inserție, se extrage identificatorul utilizatorului, apoi pe baza identificatorului produsului se interoghează baza de date pentru a extrage toate produsele cu acel identificator. Din informațiile primei înregistrări găsite este construită o noua variabilă, căreia i se mai adăuga identificatorului utilizatorului și identificatorul produsului. Acesta nouă varibilă conține în acest moment toate datele necesare pentru o forma o înregistrare în tabelul listei de cumpărături. Asadar, se poate insera noua înregistrare în baza de date.

Pentru ștergerea unei înregistrări din tabelul *ShoppingListItem*, se extrage identificatorul unic al utilizatorului și împreună cu identificatorul produsului,se crează o interogare de ștergere și se apelează. După ștergere se transmite un mesaj de succes.

În conținuare sunt ilustrate codurile sursă pentru ștergerea și inserarea unui înregistrări în tabelul *ShoppingListItems*.

```
public funcție deleteCode($deal_id, $headers) {
        foreach ($headers as $name => $value) {
            if ($name == 'user_id') {
                $user_id = $value;
                $delete = $this->_dbTable->delete(array('deal_id = ?' => $deal_id,
                                                'user_id = ?' => $user_id));
            }
        }
        return  $delete;
    }
public funcție insertShoppingListItem($deal_id, $headers) {
        foreach ($headers as $name => $value) {
            if ($name == 'user_id') {
                $user_id = $value;
            }
        }
            $select = $this->_dbTable->select()->from('ShoppingListItem')
                                    ->where('user_id = ?' , $user_id)
                                    ->where('deal_id = ?' , $deal_id)->query()->fetchAll();
        for ($i = 0; $i < 1; $i++) {
            $deal = $select[$i];
            $data = array ();
            foreach ($deal as $key => $value) {
                $data[$key] = $value;
            }
            $data['user_id'] = $user_id;
            $this->_dbTable->insert($data);
            }
}
```

Pentru crearea unui nou punct de acces (a unei vederi) se apelează o comandă *Zend*, prin apasarea click-dreapta pe proiect, selectarea opțiunii Zend, apoi selectarea opțiunii *RunCommand*. În fereastra aparută se introduce numele comenzii și numele controlerului de care sa fie legată acestă actiune. În următoarea figură sunt reprezentați vizual acesti pași.



Figură 35.NetBeans - creare vedere

Pentru trimiterea notificărilor când se inserează un nou câmp în baza de date în tabelul *Meals*, se folosește următorul bloc de cod. Trebuie setate cheile aplicației și cheia secretă primite în urma creări unei aplicații pe site-ul *UrbanAirship*. Pentru a specifica informațiile care trebuie conținute în notificare, se crează un șir de perechi nume-valoare, cu câmpurile *badge* , *alert* și *sound* unde se specifică valorile reprezentând numărul care să apară pe icoana aplicației, mesajul și numele sunetului care să atenționeze utilizatorului că o nouă notificare a sosit.

```
define('APPKEY','TAa1LJmPRo6mL-Bi1132FA');
	define('PUSHSECRET', 'kW6xSvhASumqtEXjDu0S-Q'); // Master Secret
	define('PUSHURL', 'https://go.urbanairship.com/api/push/broadcast/');

	$contents = array();
	$contents['badge'] = "+1";
	$contents['alert'] = "Howdy, doody";
	$contents['sound'] = "cow";
	$push = array("aps" => $contents);

	$json = json_encode($push);

	$session = curl_init(PUSHURL);
	curl_setopt($session, CURLOPT_USERPWD, APPKEY . ':' . PUSHSECRET);
	curl_setopt($session, CURLOPT_POST, True);
	curl_setopt($session, CURLOPT_POSTFIELDS, $json);
	curl_setopt($session, CURLOPT_HEADER, False);
	curl_setopt($session, CURLOPT_RETURNTRANSFER, True);
	curl_setopt($session, CURLOPT_HTTPHEADER, array('Content-Type:application/json'));
	$content = curl_exec($session);
	echo $content; // just for testing what was sent

	// Check if any error occured
	$response = curl_getinfo($session);
	curl_close($session);
```

Aplicația client

Aplicația client, este cel mai important modul al acestui sistem, fiind singurul cu care utilizatorul interacționează în mod direct. Aplicația este organizată pe principiul șablonului

arhitectural *Model – Vedere – Controler*. În cele ce urmează vor fi prezentate toate componentele acestui sistem, precum și clasele ajutatoare. în urma implementării acestor clase, aplicația client îndeplineste toate cerințele funcționale și non-funcționale enumerate în Capitolul 2.

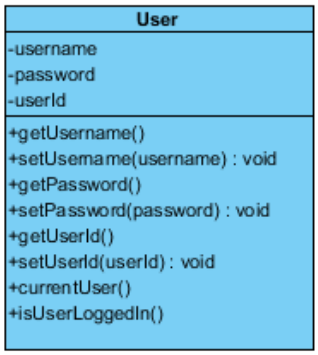
În pachetul Model există câte o pereche de clase model, construite după structura model persistent – model deserializat, pentru fiecare model necesar. Aceste clase sunt: *PersistenDeal, PersistentMeal, PersistentShoppingListItem, PersistentStore, Deal, Meal, ShoppingListItem, Store*. Tot în acest pachet se găsește și clasa User care păstrează datele pentru un utilzator logat. Clasele cu prefixul *Persistent*, sunt clase care moștenesc din clasa *NSManagedObject*, reprezentând entitățile contruite în modelul *CoreData*. Aceste clase oferă persistența datelor, fiind obiectele returnate de interogările *CoreData*.Pentru a insera obiecte în baza de date locală, se poate folosi o listă de mapare a proprietăților entităților *CoreData* cu atributele obiectelor din răspunsul cererii de la server. Dacă nu este specificată o asemenea listă, maparea se va face folosind egalitatea numelor proprietăților și a atributelor. Lista de mapare trebuie specificată într-un fișier de tip .plist(property list).

Pentru a asigura îndeplinirea cerințelor nonfuncționale legate de rapidate și concurență, se folosește un sistem cu mai multe fire de execuție pentru interogările de inserarea, ștergere , modificare și selectare a obiectelor model. Întrucât obiectele care moștenesc din *NSManagedObject*, nu se pot folosi în siguranță pe mai multe fire de execuție, este nevoie de deserializarea acestora în obiecte care pot fi folosite în siguranță pe mai multe fire de execuție. Acest lucru se realizeazț prin adăugarea unei categorii la Clasa *NSManagedObject* care realizeazî acest lucru. În următorul bloc de cod se poate observa implementarea deserializării obiectelor model *CoreData* în obiecte sigure pentru utilizarea pe mai multe fire de execuție. Aceasta clasă face acest lucru automat, dacă există clase cu numele corespunzator. În caz contrar, returnează o instanța a unui dictionar.

```
- (id)atributesToObject {
            NSEntityDescription *entity = [self entity];
            NSDictionary *allAtributes = [entity attributesByName];
           NSArray *allAtributesNames = [allAtributes allKeys];
            NSString *className = [entity name];
            NSString *serializedObjectClassName = nil;
   if ([className rangeOfString:persistentObjectKeyword].location != NSNotFound){
      serializedObjectClassName= [className stringByReplacingOccurrencesOfString:persistentObjectKeyword
                                                withString:emptyString];
   }
   if ([className rangeOfString:persistentObjectKeywordCapitalized].location != NSNotFound) {
 serializedObjectClassName = [className
stringByReplacingOccurrencesOfString:persistentObjectKeywordCapitalized
withString:emptyString];
   }
   if (serializedObjectClassName) {
      Class serializedClass = NSClassFromString(serializedObjectClassName);
      id serializedObject = [[serializedClass alloc] init];
      //if class exists
      if (serializedObject) {
         for (NSString *atributeName în allAtributesNames) {
            id atributeValue = [self valueForKey:atributeName];
            NSString *firstLetter = [atributeName substringToIndex:1];
            NSString *capitalAtributeName = [NSString stringWithFormat:@"%@%@",[firstLetter
capitalizedString],[atributeName substringFromIndex:1]];
            NSString *selectorName = [[NSString alloc]
initWithFormat:@"%@%@:",setterPrefix,capitalAtributeName];
```

```
            // if the serialized object has the managed object property
            if ([serializedObject respondsToSelector:NSSelectorFromString(selectorName)]) {
                SuppressPerformSelectorLeakWarning(
                    [serializedObject performSelector:NSSelectorFromString(selectorName)
                                        withObject:atributeValue];
                );
            }
        }
    }
    return serializedObject;
}
// if no class following the rule was found return a NsDictionary
NSMutableDictionary *retVal = [[NSMutableDictionary alloc] init];
for (NSString *atributeName în allAtributesNames) {
    id atributeValue = [self valueForKey:atributeName];
    [retVal setSafeObject:atributeValue forKey:atributeName];
}
return [NSDictionary dictionaryWithDictionary:retVal];
}
```

Clasa *User* definește atributele unui utilizator, asigurând persistența acestora între mai multe sesiuni ale aplicație, prin salvarea lor în contextul aplicație, prin clasa *NSUserDefaults*. Atributele acestei clase sunt *username*, *password* și *userId*, care reprezintă, numele utilizatorului, parola contului și identificatorul unic al acestuia, identificator primit în răspunsul de la serviciul web care realizează operația de autentificare. Pe lângă aceste atribute, acesta clasa conține și o metodă cu numele isUserLoggedIn pentru verificarea faptului ca utilizator este logat. În următoare figură este prezentată interfața acestei clase, reprezentând proprietățile și metodele ei.

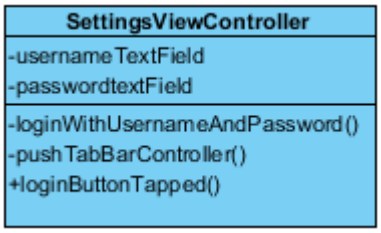


Figură 36.Interfața clasa User

În pachetul *Controller* sunt plasate clasele care realizează legătura dintre clasele vedere, cele cu care interacționează utilizatorului, și clasele model, precum și clasele care se ocupă de managmentul datelor. Fiecare ecran al aplicației este asociat câte unui controler. Astfel în aplicație se găsesc următoarele controlere: *DealsViewController, MealsView Controller, MapViewController, ShoppingListViewController, StoreController, Settings Controller, DealDetailsViewController.* Implementările acestor controlere, pentru clasele care afișeaza liste de elemente sunt similare. Prin urmare, se vor prezenta în continuare clasele pentru controlorele asociate ecranelor pentru produse, detaliile produsului, lista de cumpărături, hartă și autentificare.

Controlerul cu numele *SettingsViewController*, reprezintă clasa care se ocupă de autentificarea utilizatorului. Această clasă are referințe la obiectele de interfață grafică din vederea asociată: câmpurile de text pentru numele utilizatorului și parola acestuia, precum și

la implementarea acțiunii butonului de autentificare. După ce utilizatorul introduce datele de autentificare și apasă butonul de autentificare, aceasta clase instanțiazp clasa *WebRequest* (clasă care contruiește cererile pentru comunicarea cu serviciile web) cu constructorul de autentificare, și pornește cererea de autentificare. După ce procesul de autentificare s-a terminat există două posibile continuări. Dacă autentificarea a fost cu succes se prezintă un controler cu mai multe ecrane. dacă autentificarea a eșuat, utilizatorului îi este prezentată o alertă prin care este informat ca datele de autentificare nu sunt corecte. În continuare este prezentată această clasă, împreună cu atributele și metodele ei.



Figură 37.Interfața clasa SettingsViewController

Clasa controler *DealsViewController* se ocupă de preluarea acțiunilor utilizatorului și comunicarea dintre clasele model și vedere asociate lui.Totodată pentru îndeplinirea cerințelor funcționale, această clasă comunică și cu clasele de management a datelor, *DealsManager* și *DealsDownloadManager*. La apariția ecranului care afișează lista de produse, se verifică dacă lista a fost descărcată în sesiunea curentă. În caz afirmativ, datele sunt interogate din baza de date locală cu ajutorul clasei *DealsManager* și se afișează utilizatorului într-un tabel, o instanță a unei clase numite *UITableView*.

Pentru afișarea datelor în listă, aceasta clasă implementează protocoalele *UITableViewDataSource*, pentru crearea rândurilor din tabelși*UITableViewDelegate* pentru identificarea acțiunilor care se fac pe rândurile tabelului. În acest caz este nevoie doar de metoda apelată când utilizatorului apasă pe un rând din tabel. În cazul în care lista nu a fost descărcată în sesiunea curentă se transmite un mesaj de descărcare clasei *DealsDownloadManager*, care după ce procesul de descărcare se termină, înaintează datele spre *DealsManager*, unde cu ajutor clasei *CoreDataManager*, acestea sunt salvate local, apoi controlerul este notificat ca există date pentru a fi afișate.

Sursa de date pentru tabel se păstreaza într-un șir de obiecte *Deal*. Numărul de obiecte din acest șir determină numărul de rânduri din tabel. Fiecare rând din tabel este reprezentat de o celulă, o instanță a clasei *UITabelViewCell*. Pentru a nu supra încarca memoria aceste celule sunt refolosite. Acest proces implica crearea a unui număr minim de celule. Numărul minim de celelu necesar unui tabel este egal cu câtul împarțirii înălțimii tabelului la înălțimea celulei. Un tabel poate reutiliza oricâte tipuri de celule. Fiecare tip de celulă este identificat printr-un șir de caractere unic, reprezentat de proprietatea *cellIdentifier* a celulelor. În continuarea este prezentată implementarea metodei pentru crearea sau reutilizarea celulelor.

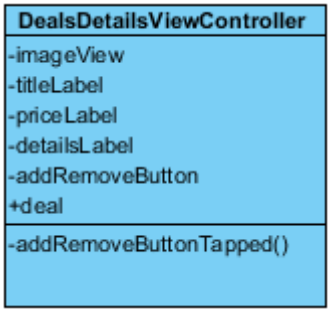
```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *cellIdentifier = @"dealsCellIdentifier";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellIdentifier];
    if (!cell) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle reuseIdentifier:cellIdentifier];
    }
    Deal *deal = [self.dataSource objectAtIndex:indexPath.row];
    if (deal.inShoppingList) {
        cell.textLabel.font = [UIFont boldSystemFontOfSize:21];
        cell.detailTextLabel.font = [UIFont boldSystemFontOfSize:18];
    } else {
```

```
        cell.textLabel.font = [UIFont systemFontOfSize:21];
        cell.detailTextLabel.font = [UIFont systemFontOfSize:18];
    }
    cell.textLabel.text = deal.name;
    cell.detailTextLabel.text = [[StoresManager sharedInstance] storeNameForStoreId:deal.storeId];
    return cell;
}
```

Când utilizatorul apasă pe un rând din tabel, se apelează *metoda –(void)tableView: didSelectRowAtIndexPath:* cu parametrii *tableView* (tabelul) și *indexPath* (un obiect care are două proprietăti care sunt folosite în această situație, *section* și *row*). Pe baza secțiunii și a rândului, se determină modelul asociat rândului și se transmite controlerului de navigație comanda de a crea și arăta vederea pentru ecranul de detalii ale produsului.

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    Deal *deal = [self.dataSource objectAtIndex:indexPath.row];
    DealDetailsViewController *ddvc = [[DealDetailsViewController alloc]
                        initWithNibName:@"DealDetailsViewController"bundle:nil];
    ddvc.deal = deal;
    [self.navigationController pushViewController:ddvc animated:YES];
}
```

În momentul în care utilizatorul apasă pe un rând din tabel, este prezentat ecranul pentru vizualizarea detaliilor produsului. Acest ecran este compus din controlerul *DealsDetailsViewController* și vederea asociată acestuia. Pe lângă elementele de afișarea a detaliilor, acestă clasă conține un buton, pentru adăugarea sau ștergerea produsului din lista de cumpărături. În funcție de statea obiectului Deal transmis de instanța clasei *DealsView* Controller, textul de pe buton este diferit și funcțiile pe care le îndeplinește aceasta sunt diferite. În cazul în care produsul nu este în lista de cumpărături a utilizatorului, butonul are textul „*Adăuga în lista*„ și la apasarea lui, produsul este marcat ca fiind în lista de cumpărături și se apelează metoda din clasa *MealDownloadController*, pentru a introduce un produ în lista de cumpărături și în baza de date de pe *server*. În cazul în care produsul este deja în lista de cumpărături, butonul are textul „*Șterge din lista*„ și la apăsarea acestuia produsul va fi șters din lista de cumpărături atât local cât și de pe server. Astfel, se realizează sincronizarea dintre aplicația server și aplicația client. În următoarea imagine, este reprezentată interfața acestei clase, cu proprietățile și metodele ei.



Figură 38.Interfața clasa DealsDetailsViewController

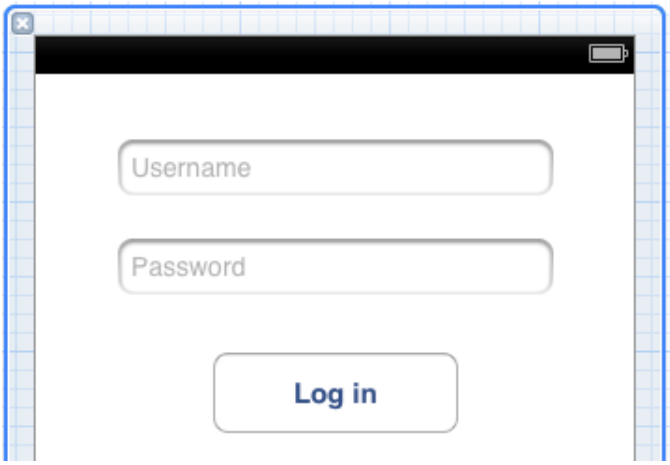
Controlerul *MapViewController*, împreună cu clasa vedere aferentă și sursa de date, realizează cerința funcțională pentru afișarea magazinelor pe hartă și găsirea celui mai scurt drum de la locația utilizatorului la cel mai apropiat magazin. Pentru calcularea și afișarea drumului, precum și a magazinelor pe hartă, s-a folosit framework-ul „*GoogleMaps.sdk*„ ,

dezvoltat de compania Google, și care are o licență care permite utilizarea lui în orice proiect. Pentru găsirea celui mai apropiate magazin de poziția utilizatorului s-a folosit următorul algoritm. Se parcurge șirul de magazine, se calculează dinstanța dintre poziția utilizatorului și pozitia magazinului. Dacă această poziție este mai mică decât cea mai mică distanță gasită până la iterația curentă, se înlocuiește distanța minimă cu distanța recent calculată. Dacă distanșa calculată nu este mai mică decât cea mai mică distanță memorată, se trece la următorul magazin. Variabila care conține cea mai mică distanță este inițializată cu cel mai mare întreg de 64 de biți. Acest algoritm este prezentat în următorul bloc de cod.

```
- (Store *)closestStoreFromCurrentLocation {
    Store *result = nil;
    float minDistance = INT64_MAX;
    CLLocation *currentLocation = [[CLLocation alloc] initWithLatitude:46.755122 longitude:23.586144];
    for (Store *s în [StoresManager sharedInstance].stores) {
        CLLocation *location = [[CLLocation alloc] initWithLatitude:[s.latitude floatValue] longitude:[s.longitude
floatValue]];
        if ([currentLocation distanceFromLocation:location] < minDistance) {
            minDistance = [currentLocation distanceFromLocation:location];
            result = s;
        }
    }
    return result;
}
```

Pentru afișarea listei de cumpărături, controlerul *ShoppingListViewController*, comunică cu clasele model și vedere asociate, precum și cu clase ajutătoare din pachetul *Library, ShoppingListManager și ShoppingListDownloadManager.* La apariția ecranului se verifică dacă lista de cumpărături a fost descărcată în sesiunea curentă. Dacă da, atunci datele necesare sunt cerute de la *ShoppingListManager* și afișate utilizatorului printr-un tabel. Spre deosebire de celelate ecrane, acesta folosește un tabel cu două secțiuni. Una pentru elementele deja cumpărate și una pentru elementele necumpărate. Așadar, în timpul vizitei în magazin, utilizatorul poate să țină evidența cumpărături foarte usor. Așadar utilizatorul poate efectua acțiunea de marcare ca și cumpărate a elementelor din lista, precum și ștergerea acestora. Ștergerea se efectuează prin glisarea degetului, de la dreapta spre stânga, pe un rând din tabel. În urma acestei acțiuni, apare un buton de delete, iar prin apăsarea acestuia se va șterge elementul atât local cât și din baza de date de pe server.

În ceea ce privește vederile asociate fiecarui controller, ele sunt generate prin utilitarul *Interface Builder*, incluse în aplicația *Xcode*. Fișierele generate au extensia .xib și au în spate cod organizat după standard-ul XML. În următoarea figură este prezentată vederea ecranului de login.



Figură 39.Interface builer - vedere pentru ecranul de autentificare

Pachetul *Library* conține clase ajutătoare, care se ocupă în principiu cu managmentul datelor. Acest pachet conține o serie de clase manager asociate fiecărui controler din aplicație, precum și o suită de clase care realizează operații de inserare, modificare și ștergere din graful de obiecte generat de *CoreData* din mediul de stocare persistent al aplicației client. Pe lângă aceste clase, există și o clasa *WebRequest*, care prin mai multi constructori specifici fiecărei metode ale serviciilor web implementate în aplicația server, contruiește câte un obiect cerere care realizează comunicare cu serverul.

În continuare este prezentată această clasă, fiind detaliat constructorul pentru operația de autentificare și modul de efectuare a cererii din clasa *SettingsViewController*, la apăsarea butonului de autentificare. Formarea obiectului cerere este alcătuită din câțiva pasi: asignarea url-ului, asignarea headerelor, asignarea tipului de metodă și asignarea valorilor POST. Metoda *setCommonHeaders*, este o metodă care este folosită de toti constructorii și care seteaza headere comune tuturor cererilor, cum ar fi identificatorul utilizatorului.

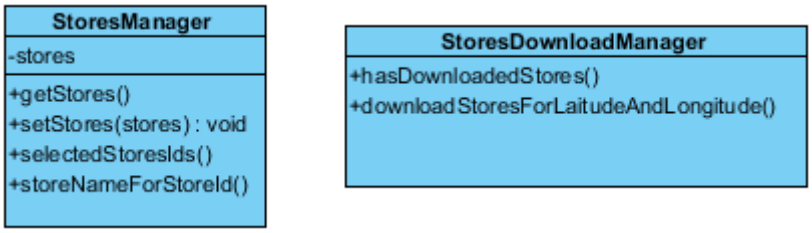
```
- (id)initForLoginWithEmail:(NSString *)email andPassword:(NSString *)password {
    self = [super init];
    if (self) {
        NSString *query = [NSString stringWithFormat:@"%@",loginUri];
        NSURL *url = [NSURL URLWithString:[NSString stringWithFormat:@"%@%@",userBase,query]];
                        self.request = [[ASIFormDataRequest alloc] initWithURL:url];
                        NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
                        NSLocale *enUSPOSIXLocale = [[NSLocale alloc]
initWithLocaleIdentifier:@"en_US_POSIX"];
                        [formatter setLocale:enUSPOSIXLocale];
                        [formatter setTimeZone:[NSTimeZone timeZoneWithAbbreviation:@"GMT"]];
                        [formatter setDateFormat:@"EEE, dd MMM yyyy HH:mm:ss ZZZ"];
                        NSString *timestamp = [formatter stringFromDate:[NSDate date]];
                        [self.request addRequestHeader:@"Date" value:timestamp];
                        [self.request addRequestHeader:@"X-ZIPONGO-Auth" value:[self signatureForQuery:query
andTimeStamp:timestamp andMethod:@"POST"]];
                        [self setCommonHeaders];
                        self.request.validatesSecureCertificate = NO;
                        self.request.requestMethod = @"POST";
        [self.request addPostValue:email forKey:@"email"];
        [self.request addPostValue:password forKey:@"password"];
    }

    return self;
}
```

În continuare, este prezentat modul în care se generează un astfel de obiect, precum și modul de operare pentru cele două cazuri de terminare posibile ale cererii. Această clasă suportă folosirea blocurilor. Blocurile sunt bucăti de cod, care pot fi scrise în linie, dar care se apelează asincron. De asemenea, obiectul cerere nu blochează threadul principal cât timp așteaptă răspunsul de la server. După cum se poate observa în următorul bloc de cod, există două cazuri posibile de terminare a cererii. Când aceasta se termină cu succes, în funcție de răspunsul returnat de aplicația server, se poate diferenția dacă utilizatorul există sau nu în baza de date . Dacă răspunsul este gol, atunci se arată o alertă care înștiințează utilizatorul că datele sale de autentificare nu sunt corecte. dacă este returnat cel putin un obiect în răspuns, atunci se extrage și setează identificatorul utilizatorului și se afișeaza următorul ecran. Dacă cererea nu ajunge la server, sau nu se primește un răspuns în timp util, aceasta este considerată eșuată și se arată utilizatorului o altă alertă care specifică acest lucru.

```
- (IBAction)loginTapped:(id)sender {
   WebRequest *request = [[WebRequest alloc] initForLoginWithEmail:self.emailTextField.text
andPassword:self.passwordTextField.text];
   __weak ASIHTTPRequest *req = request.request;
   [req setCompletionBlock:^{
      SBJsonParser *parser = [[SBJsonParser alloc] init];
      NSArray *response = [parser objectWithString:req.responseString];
      if (response && response.count > 0) {
         [[User currentUser] loginUserWithName:self.emailTextField.text andIdentifier:[[response
objectAtIndex:0] objectForKey:@"user_id"]];
         [self proceed];
      } else {
         UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Eroare"
                                                         message:@"Acest utilzator nu există în baza de date"
                                                        delegate:nil
                                               cancelButtonTitle:@"OK"
                                               otherButtonTitles:nil];
         [alert show];
      }
   }];
   [req setFailedBlock:^{
      UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Eroare"
                                                      message:@"Procesul de login nu s-a terminat cu succes"
                                                     delegate:nil
                                            cancelButtonTitle:@"OK"
                                            otherButtonTitles:nil];
      [alert show];
   }];
   [req startAsynchronous];
}
```

Tot în pachetul *Library*, sunt găsite clasele care realizează managementul datelor.În continuare se va prezenta clasa *StoresManager* și clasa *StoresDownloadManager*.Ambele clase sunt construtie după șablonul arhitectural Singleton, deorece este necesară doar o singură instanță a acestora dea lungul execuției aplicației. Clasa *StoreDownloadManaged* conține două metode: *hasDownloadedStores* și *downloadStoresForLatitudeAndLongitude*. Prima metodă returnează false dacă în sesiunea curentă nu s-a descărcat lista de magazine, true în caz contrar. A doua metoda primeste ca parametrii locația curentă a utilizatorului prin latitudine și longitudine, instanțiază un obiect *WebRequest* , pornește cererea de date către aplicația server și asteaptă răspunsul de la aceasta. Apoi, în functie de răspunsul primit, decodifică lista de magazine într-un șir de dicționare și o trimite instanței clasei

StoresManager. Clasa *StoresManager*, prin metoda *setStores*: trimite mai departe lista de magazine către clasa *CoreDataManager* care o inserează în mediul de stocare persistent din aplicaţiea client. Prin metoda *getStores*, cere datele din baza de date şi le stochează în proprietatea stores. Mehoda *selectedStoredIds*, returnează şirul de identificatori a magazinelor selectate de utilizator, iar metoda *storeNameForStoreId*, returnează numele magazinul care are ca şi identificator , parametru primit. Celelalte clase manager sunt, la fel ca şi cele două prezentate mai sus, interfeţe pentru operaţii Crud asupra bazei de date şi pentru descărcarea sau trimiterea datelor de la sau la aplicaţia server. În următoarea figură sunt prezentate interfeţele acestor două clase.



Figură 40.Interfaţa claselor StoresManager şi StoresDownLoadManager

Clasele *CoreDataManager* şi *CoreDataOperation*, sunt de asemenea incluse în pachetul *Library*. Clasa *CoreDataManager* este o clasă care conţine atât metode generice cât şi metode personalizate care se ocupă de operaţiile Crud asupra bazei de date ale aplicaţiei client. Conţine o proprietate queue, de tip *NSOperaţionQueue*, care se ocupă de managmentul operaţiilor. Clasa *CoreDataOperation* este o subclasă a clasei *NSOperation*, care conţine prin definitia ei, un mecanism de efectuare a unor operaţii pe fire de executie secundare şi care ajută la efectuarea unor operaţii asincron, astfel încât să nu se blocheze firul de execuţie principal şi utilizatorul să fie nevoie să aştepte terminarea operaţiilor.

Clasa *LocationManager*, de asemenea prezentă în pachetul *Library*, are rolul de a determina poziţia utilizatorului. La apelul metodei start, se cere sistemului de operare poziţia curentă a utilizatorului şi se salvează în mediul de stocare *NSUserDefaults*, de unde este apoi luată de clasele care au nevoie de poziţia utilizatorului pentru a o comunica serverului.

Clasa *AppDelegate*, este o clasă care implementeaza protocolul *UIApplicationDelegate*, care conţine metodele necesare obţinerii ciclului de viaţă a aplicaţiei, precum şi starea sa actuală. În această clasă, se face atât înregistrarea cu serviciul Apple pentru notificări cât şi comunicarea jetonului dispozitivului către sistemul *UrbanAirship*.

În concluzie, prin implementarea acestor clase, s-au indeplinit toate cerinţele funcţionale şi non-funcţionale ale aplicaţiei client.

5.2.2.Interfaţa utilizator

O componentă importantă a întreg sistemului este interfaţa cu utilizatorul, deoarece acesta este cea care intră în contact direct cu acesta. O aplicaţie trebuie să aibă o interfaţă cât mai plăcută, simplă şi uşor de folosit, având în vedere faptul că utilizatorul programului este un om simplu, care nu au cunoştinţe avansate în domeniu. După cum a fost amintit şi în capitolele anterioare interfaţa cu utilizatorul se realizează cu ajutorul utilitarul InterfaceBuilder, care permite introducerea a numeroase elemente de grafică precum: butoane, label-uri, tabele, etc.

Dintre toate componentele sistemului aplicaţia client, este singura care intreacţionează direct cu utilizatorul. Aşadar, se vor prezenta toate ecranele aplicaţiei, începând cu ecranul pentru autentificare. Aceasta conţine două câmpuri de text şi un buton de autentificarea. La introducerea eronată a credenţialelor sau în cazul în care cererea de autentificarea eşuează, pe

acest ecran pot apare alerte care atenționează utilizatorul despre ceea ce se întâmplă. În următoarea figură este prezentat ecranul de autentificare, împreună cu alertele posibile.

Figură 41. Ecranul de autentificare

În continuare sunt prezentate ecranele care conțin lista de magazine și harta. Pe ecranul cu lista de magazine, utilizatorul poate naviga în direcții verticale prin lista și poate selecta sau deselecta magazinele prin apăsarea pe rândul respectiv. Un magazin selectat este reprezentat prin textul îngrosat al numelui acestuia. Pe ecranul cu harta, utilizatorul poate vizualiza magazinele și drumul cel mai scurt până la cel mai apropiat magazin și poate efectuate operații de mărire și micșorare pe hartă. Cele două ecrane, parte a interfeței utilizator pot fi vizulizate în următoarea figură.

Figură 42. Ecran hartă și lista de magazine

Lista produselor este prezentată în ecranul din stânga din următoarea imagine. Acțiunile pe care poate să le realizeze utilizatorul sunt de navigare verticala și prin apăsarea unui rând, se va afișa animat ecranul de detalii a produsului. Acesta conține o imagine reprezentativă pentru produs, precum și detalii legate de preț sau descriere. Acest ecran conține și un buton, care în cazul în care produsul este deja adăugat în lista de cumpărături, o șterge din aceasta. În caz contrar, o adăuga. Acest ecran este reprezentat vizual în centru stânga imaginii de mai jos. În a treia imagine este prezenatată lista de cumpărături a utilizatorului. Acțiunile pe care aceasta poate să le facă sunt ștergerea unei înregistrări din listă, sau marcharea înregistrărilor ca și cumpărare sau necumpărate. Înregistrarile cumpărate și cele care urmeaza a fi cumpărate, sunt separate în două secțiuni. În ultima imagine este prezentata opțiunea de ștergere a unei înregistrări din lista.

Figură 43.Ecrane produse, detalii produse și lista de cumpărături

În următoarea figură este prezentat ecranul meniului. Aici utilizatorul poate să își planifice meniurile, prin apăsarea butonului planifică. Aceasta acțiune determină apariția unui meniu de selecție a unei date. Prin selectarea unei daea, meniul este asignat acelei date și va fi șters din lista principală.

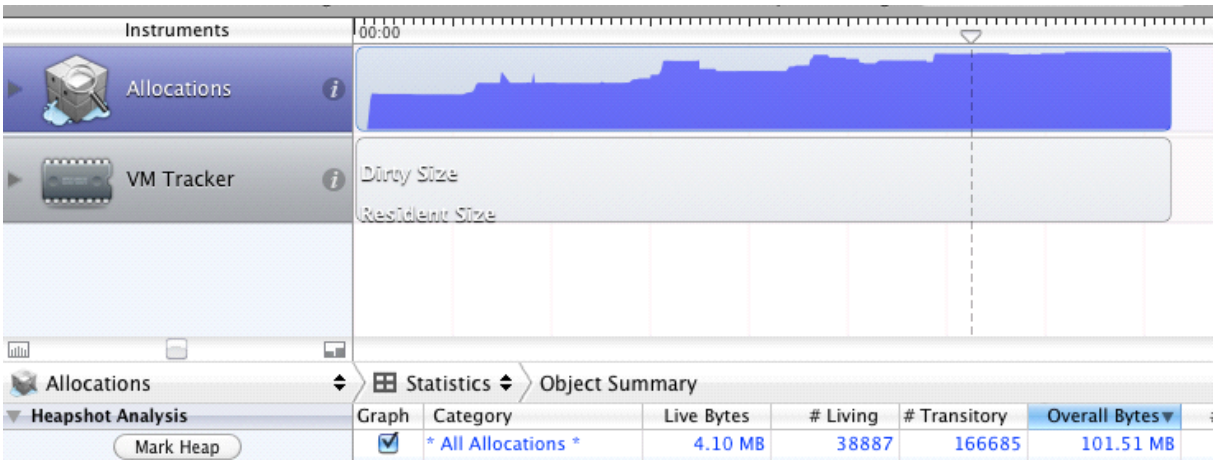
Figură 44.Ecrane meniu

Capitolul 6. Testare și validare

Testarea software este un proces de validare și verificare a unui produs software în vederea respectării cerințelor funcționale și non-funcționale care au ghidat proiectarea și implementarea lui. Astfel, sistemul trebuie să ruleze și să aibă un comportament corespunzator așteptărilor. Scopul principal al testării unui produs software este detecția erorilor, izolarea și corectarea defectelor care au cauzat erorile. Testarea poate demonstra că produsul funcționează sau nu în anumite condiții.

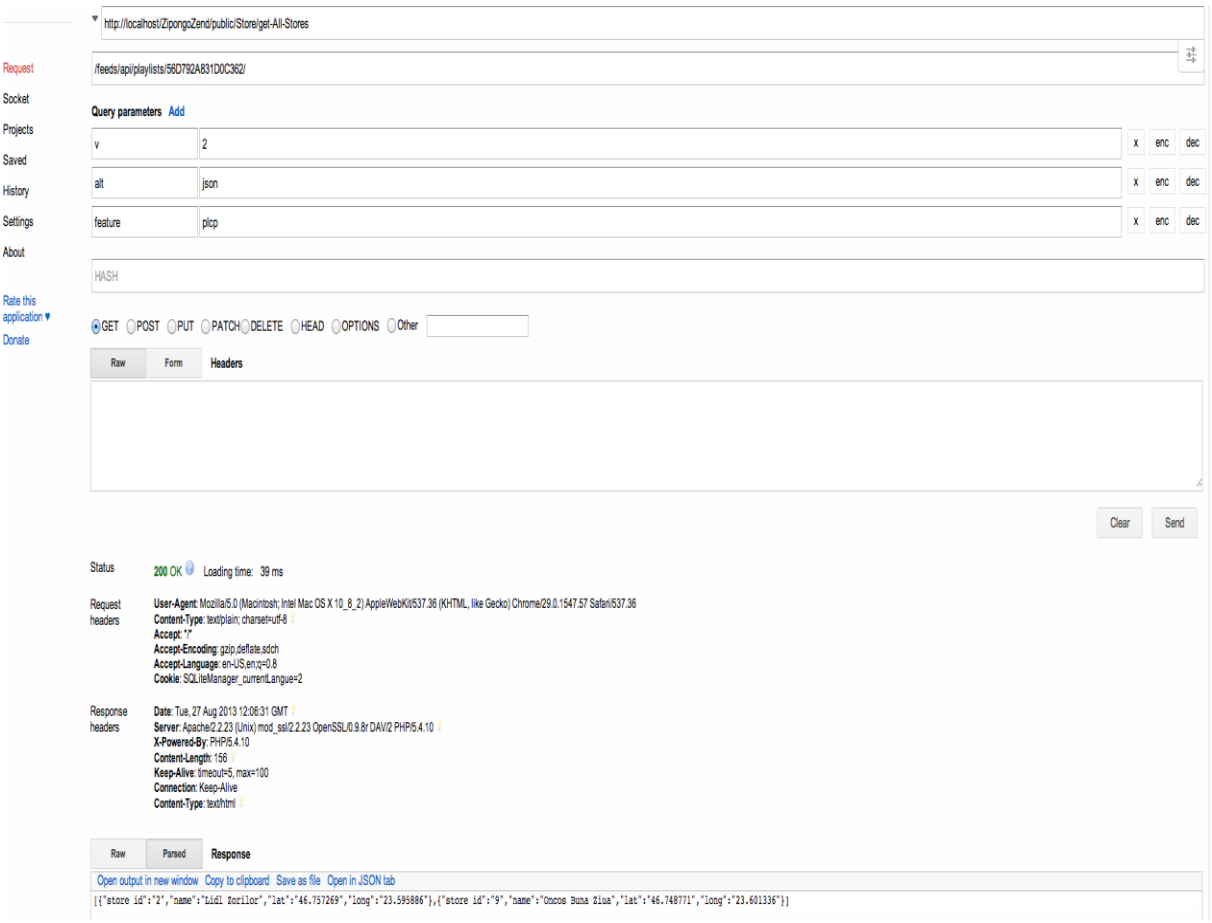
În cadrul procesului de testare s-a urmărit ca cele două componente ale sistemului să funcționeze corect, îndeplinind cerințele funcționale ale sistemului.Pentru depistarea și corectarea erorilor am folosit modul de debug al IDE-ului utilizat, precum și un serviciu extern (pentru testarea aplicației server).

Componenta client aplicației a fost testata manual, și cu ajutorul unui utilitar numit, *Instruments*, utilitar care masoară performanata aplicației. Am folosit acest utilitar pentru testarea consumului de memorie al aplicației. Deoarece dispozitivele mobile dispun de memorie limitată, este importantă testareautilizării acesteia. În următoarea figurăeste prezentat rezultatul acestui test. Performanțase încadrează în jumătatea inferioară a intervalului admis (2-50MB). Această performanță a fost obținută prin reutilizarea tuturor componentelor, păstrandu-se în memorie doar datele absolut necesare. Se poate observa ca au fost alocați pe durata monitorizarii (1 minut) 101.58 MB, dar la momententul de timp în care s-a folosit cea mai multă memorie, valoarea acesteia a fost de 4.10 MB.



Figură 45.Utilitarul Instruments - monitorizarea utilizarii memoriei

Componenta server a fost testată atât prin apelarea metodelor din aplicația client, cât și folosind extensia cu numele RestClient oferită de navigatorul web, Mozzila Firefox care afișează răspunsul primit de la metoda apelata, precum și codul de răspuns al acesteia si headerele trimise si primite. În următoarea figură se poate observa apelul metodei *get-all-stores* și răspunsul primit de la aplicația server.



Figură 46. Rest client - apelul și răspunsul metodei get-all-stores

Testarea comunicării dintre cele patru componente ale aplicației s-a facut folosind următorul scenariu de test:

- S-a descărcatia client pe un dispozitiv mobil iPhone 4S
- S-a efectuat operație de login de două ori:
    - prima dată cu credențiale invalide: s-a observat alerta care indică acest lucru
    - a doua dată cu credentiale valide: s-a observat apariția ecranului magazinelor.
- După apariția ecranului cu magazine, s-a asteptat descărcarea acestora și s-au realizat acțiuni de selectie a magazinelor.
- S-a afișat ecranul hărtii și s-au putut observa magazinele pe harta, precum și cel mai scurt drum dintre poziția utilizatorului și pozitia celui mai apropiat magazin, drum trasat cu o linie groasă de culoare albastră
- La afișarea ecranului cu produse s-a apăsat pe unul dintre rândurile tabelului și ecranul de detalii a produsului a fost afișat
- S-a realizat acțiunea de adăugare în lista de cumpărături
- La revenirea pe ecranul produselor, produsul selectat are numele îngrosat
- La afișarea ecranului listei de cumpărături, se poate observa produsul adăugat
- Prin apăsarea pe rândul produsului,aceasta trece în categoria obiectelor cumpărate
- Prin apăsarea pe noul rând al produsului, aceasta trece înapoi în categoria obiectelor care trebuie cumpărate

- Prin glisarea degetului de la dreapta la stânga, se poate observa apariăia unui buton de ștergere rosu.
- Prin apăsarea butonului de ștergere, oferta dispare din lista de cumpărături
- Pentru verificarea sinconizării dintre aplicația client și aplicația server, după introducerea și ștergerea unui element din lista de cumpărături, s-a repornit aplicația și s-a putut observa, prezența sau absența produsului în lista de cumpărături
- La afișarea ecranului meniului, s-a putut observa lista obiectelor din meniu.
- La apăsarea butonului „Planifica„ s-a observat apariția unui meniu cu următoarele șapte date
- La selecția unei date, elementul de meniu a dispărut din lista elementelor neplanificate și a apărut în lista elementelor planificate.
- Pentru testarea notificărilor, s-a introdus o nouă înregistrare în baza de date, iar la scurt timp după, pe dispozitivul mobil s-a primit o notificare.

Capitolul 7. Manual de instalare și utilizare

7.1.Cerințe de resurse

În ceea ce privește cerințe de resurse, sistemul este dezvoltat pentru a acoperi o gama cât mai variată de dispozitive mobile.

Resursele software minimale sunt reprezentate de sistemul de operare de pe dispozitivele Apple la versiunea 5.0, dat fiind faptul ca un studiu[36] arată că 98 % din dispozitivele mobile de pe piață folosesc versiunea 5.0 sau mai mare a sistemului de operare iOS.

În legătură cu cerințele hardware, singura limitare pentru instalarea aplicației este spațiul de pe disc necesar, care trebuie sa fie mai mare de 8 MB.

7.2.Specificații de instalare

Instalarea aplicației client este foarte ușoară, nefiind nevoie de nici un fel de configurări. Aplicația se descărca de pe magazinul online AppStore. De obicei aplicația este instalată pe primul slot disponibil, începând cu a doua pagină, de pe "desktop"-ul dispozitivului.După descărcarea aplicației client, aceasta este gata de utilizare și se lansează prin apăsarea pe icoana corespunzătoare.

7.3.Manual de utilizare

După instalarea aplicației, apare ecranul de autentificarea. Acțiunile câte trebuie să le faca utilizatorul pentru autentificarea în sistem sunt următoarele:

- Introducerea numelui de utilizator
- Introducerea parolei
- Apăsarea butonului „Autentificare„

Pe ecranul care afișează produsului utilizatorul poate să facă următoarele acțiuni:

- Navigare verticală prin ținerea și mișcarea degetului pe ecran
- Selectarea unui rând din tabelul prezentat pentru afișarea ecranului care conține detaliile produsului

Pe ecranul care prezinta detaliile produsului, utilizatorul poate efectua următoarele acțiuni:

- Adăugarea produsului în lista de cumpărături
- Ștergerea produsului din lista de cumpărături

Pe ecranul care afișează lista magazinelor, utilizatorul are opțiunea de a face următoarele acțiuni:

- Navigare verticală
- Selectarea unui magazin ca fiind favorit
- Deselectarea unui magazin din lista de magazine favorite

[36]http://www.technobuffalo.com/2013/06/21/ios-6-distribution-over-90-percent/ ultima accesare 16 august 2013

Pe ecranul care afișeaza harta care cuprinde magazinele şi drum cel mai scurt dintre pozitia utilizatorului şi a celui mai apropiat magazin, utilizatorul poate efectua următoarele acțiuni:

- Mărire sau micșorare vedere pe hartă
- Mutarea vederii curente în orice direcție

Ecranul listei de cumpărături oferă utilizatorului posibilitatea de a efectua următoarle acțiuni:

- Navigare verticală prin lista de cumpărături
- Selectarea unei înregistrări din listă
- Deselectarea unei înregistrări din listă
- Ștergerea unei înregistrări din listă

Pe ecranul de meniu, utilizatorul are următoarele optiuni:

- Navigarea verticală prin lista de mese neasignate
- Navigarea verticală prin lista de mese asignate
- Asignarea unui mese la o zi
- Ștergerea unei mese dintr-o zi

Capitolul 8. Concluzii

8.1.Realizări

Un proiect este rezultatul unui set de activități propuse la începutul acestuia şi a celor apărute pe parcurs, propuneri care trebuie realizate pentru finalizarea proiecutlui. Procesul de realizare a sistemului a presupus urmarea anumitor paşi: documentarea asupra subiectului care urmează a fi abordat, identificarea cerinţelor funcţionale şi non-funcţionale, alegerea unui limbaj de programare pentru îndeplinirea cerinţelor, alegerea celor mai potrivite tehnologii suportate de limbajul de programare ales, analiza sistemului, proiectarea acestuia, implementarea propriu zisă, testarea şi documentarea proiectului.

Dezvoltarea aplicaţiei client, în limbajul de programare Objective-Cnu mi-a pus probleme având o oarecare experienţa în acest limbaj, iar programarea obiectuală îmi era familiară. Folosirea framework-ului Zend a fost o provocare la începutul dezvoltării aplicaţiei server, însă perioada de acomodare a fost relativ scurtă. Folosireauneltor specifice fiecărui limbaj de programare, a uşurat considerabil efortul de programare. La nivelul studiului biliografic, am învăţat lucruri noi în ceea ce priveşte unele tehnologii Cocoa, precum şi framework-ul Zend şi metodele de comunicare într-un sistem client-server.

Implementarea sistemului folosind tehnologiile prezentate în capitolul 4, precum şi deciziile de proiectare, oferă sistemuluireutilizabilitate şi modularitate. Astfel modifcările sau adăugările ulterioare nu necesită un efort de programare mare.

Serviciul web permite conectarea şi a altor aplicaţii client. Fiind un serviciu web de tip REST comunicare dintre client şi server este foarte simplă, transmiterea datelor în format Json sau Xml fiind foarte eficientă şi independentă de platforma aplicaţiei client. Asftel se pot realiza aplicaţii mobile Android sau Symbian.

În final, s-a realizat un sistem cu două componente, folosind şi două componente ajutătoare, care împreună comunică şi realizează specificatiile inițialedescrise în cerinţele proiectului.

8.2.Dezvoltări ulterioare

Obiectivele prezentate în capitolul 2 au fost îndeplinite, dar există şi o serie de noi caracteristici ale aplicaţiei care pot fi integrate, pentru a oferi o experienţă câtmai placută utilizatorului.

În cele ce urmează vor fi prezentate dezvoltările ulterioare care pot fi aduse atât aplicaţiei client cât şi aplicaţiei server.

Astfel **aplicaţia client** poate fi îmbunătăţită prin adăugarea următoarelor caracterisitici:

- adăugarea posibilităţii de căutare în diferitele liste folosite
- adăugarea unei opţiuni de căutare în baza de date a aplicaţiei server
- adăugarea paginării pentru liste, acestu lucru ar face cererile pentru date mai rapide precum şi procesarea lor locală mai rapidă
- adăugarea unui sistem pentru inserareamanuală a unei locaţii , în cazul în care nu se poate găsi locaţia utilizatorului
- îmbunătăţirea listei de cumpărături prin adăugarea unor noi stări ale elementelor

- adăugarea unor opţiuni de publicare a conţinutului pe diferite reţele de socializare

Aplicația server ar putea fi îmbunătăţită prin următoarelor caracterisitici:

- adăugarea unei pagini pentru efectuarea operaţiilor Crud de către un administrator
- adăugarea unei pagini pentru vizualizarea magazinelor
- adăugarea unei optiuni de editare a a înregistrărilor din baza de date
- crearea unui sistem propriu de transmitere a notificărilor spre APNS, având în vedere posibilele întârzierile provocate de sistemul Urban Airhip
- securizarea datelor
- crearea unui site care să utilizeze aceleaşi resurse
- micşorarea mărimii răspunsului trimis către aplicaţia client, prin introducea paginării
- optimizarea cererilor către baza de date
- memorarea discounturilor cumpărate de utilizator prin aplicaţie şi creerea unui algoritm pentru a oferi acestuia discounturi la produsele preferate

Anexa 1. Listă de figuri

Anexa 2. Glosar de termeni si acronime

MVC: șablonul arhitectural model-view-controller

GPS : Global Positioning System: sistem de poziționare pe glob

API : Application Programming Interface: interfață pentru programarea aplicațiilor

APNs : Apple notification center service : serviciu oferite de compania Apple pentru trimiterea notificarilor

UA : Urban Airhip : sistem ce faciliteaza tranmiterea notificarilor spre APNs

CD : Core Data

JSON : Javascript Object Notation

PHP : HyperText Preprocessor

Zend : framework pentru Php cu module decuplate

JVM : Java Virtual Machine

GCC : GNU C compiler

LLVM-GCC : Low Level Virtual Machine Compiler

Anexa 3. Referințe bibliografice

1	Goransson, Paul , Raymond Greenlaw, *Secure roaming în 802.11 networks. Newnes, 31 mai 2007, pg. 16*
2	Organizaţia Economică pentru Cooperare şi Dezvoltare, Comisia Europeana, *Health at a Glance:Europe 2012*, http://www.oecd.org/els/health-systems/HealthAtAGlanceEurope2012.pdf, ultima accesare 1 august 2013
3	Rip, Empson,*With $1M+ From Guitar Hero Co-founder & Others, Zipongo Is Building The Mint.com For Healthy Living*, http://techcrunch.com/2012/07/25/zipongo-seed-round/, ultima accesare 1 august 2013
4	Shaywitz, David,*Rock Health's New Class: Daring To Be Useful*, http://www.forbes.com/sites/davidshaywitz/2013/02/21/rock-healths-new-class-daring-to-be-useful/ , ultima accesare 28 iunie 2013
5	Apple Inc. , *Cocoa Touch frameworks*, https://developer.apple.com/technologies/ios/cocoa-touch.html, ultima accesare 1 iulie 2013
6	Apple Inc.,*Cocoa Core Competencies, Cocoa Touch*, http://developer.apple.com/library/ios/documentation/general/conceptual/DevPedia-CocoaCore/Cocoa.html, ultima accesare 2 iulie 2013
7	Apple Inc., *Cocoa Fundamentals Guide, What is Cocoa*, https://developer.apple.com /library/mac/documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html#//apple_ref/doc/uid/TP40002974-CH3-SW16, ultima accesare 5 iulie 2013
8	Apple Inc., *Xcode User Guide, Xcode*, https://developer.apple.com/library/mac/ documentation/ToolsLanguages/Conceptual/Xcode_User_Guide/000-About_Xcode/about.html ultima accesare 6 iulie 2013
9	Apple Inc., *Core Data Programming Guide, Core Data*, https://developer.apple.com/ library/mac/documentation/cocoa/Conceptual/CoreData/Articles/cdBasics.html#//apple_ref%20/doc/uid/TP40001650-TP1, ultima accesare 7 iulie 2013
10	Matt, Gallagher, *The differences between Core Data and a Database*, http://www.cocoawithlove.com/2010/02/differences-between-core-data-and.html, ultima accesare 20 august 2013
11	Apple Inc., *Local and Push Notification Programming Guide*, https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/ RemoteNotificationsPG/Chapters/ApplePushService.html ultima accesare 8 iulie 2013
12	Apple Inc., *Local and Push Notification Programming Guide, Remote Notifications*, https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/ RemoteNotificationsPG/ Chapters/WhatAreRemoteNotif.html#//apple_ref/doc/uid/TP40008194-CH102-SW1 ultima accesare 8 iulie 2013
13	Apple Inc., *Local and Push Notification Programming Guide, Apple Push Service*, https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/ Remote NotificationsPG/Chapters/ApplePushService.html#// apple_ref/doc/uid/ TP40008194-CH100-SW9 ultima accesare 7 iulie 2013
14	The PHP Group, *Prefață*, http://www.php.net/manual/ro/preface.php, ultima accesare 18 iunie 2013
15	The PHP Group, *What is PHP?*,http://www.php.net/manual/en/intro-whatis.php, ultima accesare 18 iunie 2013
16	The PHP Group, *What can PHP do?*,http://www.php.net/manual/en/intro-

	whatcândo.php, utima accesare 20 iunie 2013
17	Chris, Shiftlett, *Zend Framework Webcast*, http://shiflett.org/blog/2005/dec/zend-framework-webcast ultima accesare 10 augist 2013
18	Zend Technologies Ltd., *Introduction to Zend Framework 2*,http://framework.zend.com/manual/2.0/en/ref/overview.html ultima accesare 10 august 2013
19	Oracle Corporation and/or its affiliates, *NetBeans Platform Features*, http://netbeans.org/features/platform/features.html ultima accesare 12 august 2013
20	*Introducing JSON*, http://www.json.org/ ultima accesare 10 august 2013
21	The Regents of the University of California, *MySQL Basic*, http://ist.berkeley.edu/services/ds/db/mysql/basic ultima accesare 11 august 2013
22	Apple Inc., *Cocoa Core Competencies, Model View Controller*, https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/MVC.html ultima accesare 10 august 2013
23	Apple Inc., *Programming with Objective-C, Working with protocols*, https://developer.apple.com/library/ios/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/Workingwith Protocols/WorkingwithProtocols.html utlima accesare 12 august 2013
24	Apple Inc,. *Cocoa Core Competencies, Delegation*, https://developer.apple.com/library/ios/documentation/general/conceptual/DevPedia-CocoaCore/Delegation.html ultima accesare 14 august 2013
25	Brandon Russell, *Apple Charts iOS Distribution Figures; iOS 6 Over 90 Percent*, http://www.technobuffalo.com/2013/06/21/ios-6-distribution-over-90-percent/ ultima accesare 16 august 2013
26	Salomie, Ioan, *Cursuri: Tehnici de programare*, Sisteme distribuite (2011).
27	Dînşoreanu, Mihaela, *Cursuri: Inginerie software* (2011)