

Universitatea “Politehnica” din Bucureşti  
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

*Aplicație pentru terminalele mobile bazată pe QR-code-uri și  
geolocație utilizând baze de date distribuite*

## **Proiect de diplomă**

prezentat ca cerință parțială pentru obținerea titlului de

*Inginer* în domeniul *Calculatoare și Tehnologia  
Informației*

programul de studii de licență *Ingineria  
Informației (CTI – INF)*

Conducător științific

Absolvent

*As. Ing. Valentin PUPEZESCU Raluca - Cristina LĂCĂTUȘU*

*Anul  
2013*

Universitatea "Politehnica" din Bucureşti  
Facultatea de Electronică, Telecomunicații și Tehnologia Informației  
**Departamentul** Electronică Aplicată și Ingineria Informației

Aprobat Director de Departament :

Prof. Dr. Ing. Sever Pașca

**TEMA PROIECTULUI DE DIPLOMĂ**

a studentului Lăcătușu C. Raluca - Cristina

**1. Titlul temei: Aplicație pentru terminalele mobile bazată pe QR-code-uri și geolocație utilizând baze de date distribuite**

**2. Contribuția practică, originală a studentului va consta în:**

Dezvoltarea unei aplicații pentru terminale mobile inteligente – smartphone – folosind concepte de programare obiect – orientată, tehnologii de markup și baze de date distribuite.

Cu ajutorul aplicației se pot genera și citi coduri inteligente de tip QR, pe baza cărora se pot obține informații într-un mod rapid folosind servicii web.

Prin intermediul facilității de localizare – geolocație – aplicația poate afișa utilizatorului final informații mult mai precise, personalizate.

Utilizarea bazelor de date distribuite permite un flux de date mai mare, astfel că utilizatorul final poate obține informații în timp real, fără a afecta performanțele generale ale întregului sistem.

**3. Realizarea practică/ proiectul rămâne în proprietatea: UPB**

**4. Proprietatea intelectuală asupra proiectului aparține: UPB**

**5. Locul de desfășurare a activității: UPB, ETTI**

**6. Data eliberării temei: 1 oct. 2012.**

**CONDUCĂTOR LUCRARE:**

S.L. Dr. Ing. Pupezescu Valentin

**STUDENT:**

Lăcătușu Raluca - Cristina

## **Declarație de onestitate academică**

Prin prezenta declar că lucrarea cu titlul “*Aplicație pentru terminalele mobile bazată pe QR-code-uri și geolocație utilizând baze de date distribuite*”, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității “Politehnica” din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Calculatoare și Tehnologia Informației*, programul de studii *Ingineria Informației (CTI – INF)* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 1.07.2013

Absolvent Raluca - Cristina LĂCĂTUȘU





# Cuprins

Introducere.....	9
Capitolul 1: Baze de date.....	11
1.1 MySQL.....	11
1.1.1 Introducere.....	11
1.1.2 Scurt istoric.....	11
1.1.3 Noțiuni generale.....	12
1.1.4 Tipuri de date.....	14
1.2 NoSQL.....	19
1.2.1 MongoDB.....	20
1.2.1.1 Introducere.....	20
1.2.1.2 Scurt istoric.....	21
1.2.1.3 Conceptele MongoDB.....	22
1.2.1.4 Anatomia unui document din MongoDB.....	22
1.2.1.5 BSON - formatul de date de schimb în MongoDB.....	22
1.2.1.6 Tipuri de date.....	23
1.2.1.7 Operații folosite de MongoDB pentru lucrul cu consola.....	25
Capitolul 2: PHP.....	29
2.1 Introducere.....	29
2.2 Scurt istoric.....	29
2.3 Sintaxa PHP.....	30
2.4 OOP.....	30
2.5 Integrarea PHP-ului cu bazele de date.....	32
Capitolul 3: Tehnologii pentru User Interface.....	35
3.1 HTML/XHTML.....	35
3.1.1 Introducere.....	35
3.1.2 Structura unei pagini HTML.....	35
3.1.3 Tag-uri, elemente și atrbute.....	35
3.2 CSS.....	37
3.2.1 Introducere.....	37
3.2.2 Selectori în CSS.....	38
3.2.3 Proprietăți CSS.....	38
3.3 JavaScript.....	41
3.4 jQuery Mobile.....	42
3.5 Coduri QR.....	46
Capitolul 4: Proiectarea aplicației.....	47
4.1 Prezentare generală a interfeței.....	47
4.1.1 Secțiunile Authentication, Registration, Reset Password.....	47
4.1.2 Secțiunea Coffee Shop.....	48
4.1.3 Secțiunea Settings.....	49
4.1.4 Secțiunea Menu.....	50
4.1.5 Secțiunea Basket.....	51
4.1.6 Secțiunea Locations.....	52
4.1.7 Review Order.....	53

4.1.8 Order confirmed.....	53
4.1.9 Secțiunea Favourite.....	54
4.1.10 Secțiunea Activity.....	54
4.1.11 Secțiunea Friends.....	55
4.2 Structura bazei de date MySQL.....	56
4.3 Teste de performanță.....	57
4.3.1 Testarea aplicației la nivel de Client.....	57
4.3.2 Testarea bazelor de date.....	57
Concluzii.....	63
Bibliografie.....	65
Anexe.....	67

## **Lista acronimelor**

SDK - Software Development Kit (Platformă de dezvoltare software)  
NFC - Near Field Communication (Comunicare în câmp apropiat)  
SQL - Structured Query Language (Limbaj de interogare structurat)  
SaaS - Software As A Service  
PHP - PHP: Hypertext Preprocessor (Limbaj preprocesare hypertext)  
HTML - HyperText Markup  
XHTML - eXtensible HyperText Markup Language  
Language (Limbaj de marcaje hypertext)  
CSS - Cascading Style Sheet  
LAMP - Linux Apache MySQL PHP  
ISAM - Indexed Sequential Access Method  
mSQL - miniSQL  
SGBD - Sistem de Gestiune a Bazelor de Date  
VARCHAR - Variable Length Character String (Sir de caractere cu lungime variabilă)  
BLOB - Binary Large OBject  
XML - eXtensible Markup Language  
JSON - JavaScript Object Notation  
RDBMS - Relational DataBase Management System (Sistem de management al bazelor de date)  
BSON - Binary Script Object Notation  
UTF-8 - Universal Character Set Transformation Format - 8 bit  
DB - DataBase  
PDF - Portable Document Format  
GIF - Graphics Interchange Format  
JPG - Joint Photographic Group  
PNG - Portable Network Graphics  
CGI - Common Gateway Interface  
POO/OOP - Object Oriented Programming (Programare Obiect Orientată)  
PHP/FI - Personal Home Page/Forms Interpreter  
PDO - PHP Data Objects  
DML - Data Manipulation Language  
DDL - Data Definition Language  
LOBS - Large OBjectS  
MySQLi - MySQL Improved  
JSP - Java Server Pages  
ASP - Active Server Pages  
RGB - Red Green Blue (Roșu Verde Albastru)  
URL - Uniform Resource Locator  
DOM - Document Object Model  
AJAX - Asynchronous JavaScript And XML  
UI - User Interface (Interfață cu utilizatorul)  
IIS - Internet Information Services  
JPA - Java Persistence API  
SLA - Service-Level Agreement



# Introducere

Telefoanele mobile de tip smartphone au cunoscut o creștere exponențială în ultimii ani, acest lucru datorându-se în primul rând aplicațiilor care ușurează activitățile zilnice. Din ce în ce mai mulți utilizatori aleg terminale mobile inteligente în detrimentul calculatoarelor clasice (desktop/notebook) pentru citirea corespondenței electronice, accesarea rapidă a informațiilor, accesarea rețelelor de socializare etc. În 2012 au fost descărcate aproximativ 46 miliarde de aplicații, dintre care 89% (aproximativ 40 miliarde) au fost contra cost. Pentru 2013 se preconizează un număr de 82 de miliarde de aplicații downloadate, ceea ce înseamnă ca numărul total va ajunge la 165 miliarde de aplicații. Până în 2017, piața aplicațiilor mobile va crește până la 63,5 miliarde de dolari.

Piața aplicațiilor mobile este în continuă expansiune datorită platformelor de dezvoltare puse la dispoziție de către producătorii sistemelor de operare pentru telefoane smartphone. Pionierii terminalelor mobile sunt Apple Inc. prin iPhone. Odată cu apariția primului iPhone, dezvoltatorii au avut ocazia să vândă aplicațiile proprii prin intermediul unui magazin virtual - AppStore, câștigând aproximativ 1\$/ aplicație. La scurt timp după introducerea magazinului AppStore au apărut Google Play, Ovi Store, Windows Phone Store etc.

Aplicațiile pentru smartphone se pot încadra în două mari categorii: aplicații native (dependente de platforma pe care rulează) și aplicații web (aplicații care rulează în browser, pe orice terminal mobil). Aplicațiile native pot folosi toate facilitățile SDK-ului sistemului de operare și au un design uniform; în schimb actualizările se produc foarte greu (pentru fiecare versiune de aplicație este nevoie de aprobare din partea administratorului platformei).

Aplicațiile web pot rula în orice browser web, pot fi întreținute și actualizate relativ repede. Dezavantajul aplicațiilor web este că nu pot accesa toate facilitățile oferite de către SDK și nu există un model definit de design.

Proiectul prezintă realizarea unei aplicații pentru iPhone, care facilitează comenziile de produse prin folosirea codurilor QR. Această aplicație poate fi folosită în toate locațiile de tip restaurant, cafenea, ceainărie. Avantajul folosirii unei aplicații de acest tip este că efectuarea și preluarea comenziilor cu ajutorul telefoanelor inteligente poate ajuta atât consumatorul, cât și vânzătorul. Toate comenziile efectuate vor fi stocate într-o bază de date, ceea ce facilitează generarea de rapoarte de vânzare, statistici de produse etc.

Realizarea acestei aplicații permite utilizatorului să efectueze diverse acțiuni, cum ar fi comandarea unor produse, scanarea unor QR-code-uri pentru a beneficia de oferte, localizarea pe hartă, direct de pe terminalele mobile, într-un timp foarte scurt. De asemenea, aplicația oferă detalii despre locațiile cafenelelor din lanțul „Coffee Shop”, precum și localizarea lor pe hartă, relativ la poziția utilizatorului.

Prima etapă a acestui proces presupune crearea unui cont, prin completarea unor date personale într-un formular, iar apoi logarea utilizatorului. În aplicație putem să adăugăm prieteni, invitându-i prin introducerea adresei de e-mail, putem vedea istoricul acțiunilor realizate de utilizator, dar și ofertele speciale.

Cu ajutorul aplicației putem scana un QR-code pentru a beneficia de o reducere pentru o eventuală comandă. Codul scanat poate fi folosit o singură dată, păstrându-se un istoric al utilizării

voucherelor. Dacă se scanează un alt cod QR, care nu este generat de Coffee Shop, utilizatorul va primi un mesaj de eroare. După efectuarea comenzi, se va genera un cod QR care conține codul comenzi memorate în baza de date. Prin scanarea codului cu ajutorul unei alte aplicații, se pot afla informații despre utilizatorul care a efectuat comanda, detaliile comenzi, precum și eventualele coduri de reducere folosite.

Codurile QR au devenit din ce în ce mai utilizate în ultimii ani, datorită faptului că pot conține până la 4296 de caractere alfanumerice și 7089 de caractere numerice. Dacă aceste coduri sunt scanate cu o aplicație specială, ele pot oferi referințe foarte rapide despre diverse locuri sau produse. Fiindcă suntem în perioada în care telefoanele mobile domină industria produselor, au apărut din ce în ce mai multe aplicații care ne ușurează traiul, una dintre aceste aplicații fiind scanner-ul de coduri QR. În cazul de față, prin folosirea codurilor QR sau a tehnologiei NFC, se va fluidiza procesul de comandă, evitând astfel aglomerația la casă, în special la orele de vârf.

Un alt concept care apare destul de des în ultimul timp este acela al bazelor de date distribuite. Un obiectiv major al acestui sistem de baze de date este de a oferi usorință cu care utilizatorii din locații diferite pot accesa baza de date. Un alt obiectiv este autonomia locală, care înseamnă capacitatea de a administra o bază de date locală și funcționarea acesteia independent atunci când conexiunile la alte noduri au eșuat.

Scopul acestui proiect este de a realiza o comparație între cele 2 tipuri de baze de date: relaționale (MySQL) și distribuite (MongoDB). Aplicația rulează pe ambele sisteme de baze de date, acest lucru realizându-se schimbând valoarea unei variabile (tipul conexiunii - Mongo sau MySQL). În ultima parte a proiectului voi prezenta câteva teste de performanță pentru ambele sisteme de baze de date, pentru a evidenția avantajele și dezavantajele folosirii fiecărui sistem de baze de date.

Tehnologiile și framework-uri folosite pentru dezvoltarea acestui proiect sunt:

- MySQL - v. 5.5.25
- Mongo - v. 2.4.5
- PHP - v. 5.4.4
- PhoneGap - . 2.5.0
- HTML/XHTML - v. 4.0/v. 1.0
- CSS - v. 2.0
- JavaScript/jQuery Mobile - v. 1.9

# Capitolul 1: Baze de date

Bazele de date au devenit o componentă esențială a vieții de zi cu zi în societatea modernă. În fiecare zi desfășurăm activități care implică interacțiunea cu o bază de date, de exemplu, rezervarea biletelor online, interogarea soldului cardului.

O bază de date este o colecție de date corelate din punct de vedere logic care este destinată unui anumit grup de utilizatori. O bază de date trebuie să asigure abstractizarea, integrarea, integritatea, securitatea, partajarea și independența datelor.

O bază de date este o colecție de date centralizate, creată și menținută computerizat, în scopul prelucrării datelor necesare aplicației. Prelucrarea datelor se referă la operațiile de introducere, ștergere, actualizare și interogare a datelor.

## 1.1 MySQL

### 1.1.1 Introducere

MySQL, lansat în 1995, a devenit cel mai popular sistem de baze de date open source cu peste 100 de milioane de copii ale software-ului de-a lungul istoriei sale. Datorită vitezei superioare, fiabilității și ușurinței de utilizare, MySQL a devenit alegerea preferată pentru Web, Web 2.0, SaaS deoarece elimină problemele majore asociate cu downtime-ul, întreținere și administrative pentru aplicații moderne, on-line.

MySQL este o parte importantă din LAMP (Linux, Apache, MySQL, PHP/Perl/Python). Tot mai multe companii folosesc LAMP ca o alternativă la pachetele software scumpe, datorită costurilor mai mici și datorită libertății platformei.

### 1.1.2 Scurt istoric

MySQL a fost fondat și dezvoltat în Suedia, de către doi suedezi și un finlandez: David Axmark, Allan Larsson și Michael “Monty” Widenius, care au lucrat împreună încă din anii '80. Aceasta poveste merge înapoi în 1979, când inventatorul MySQL, Michael Widenius a dezvoltat un instrument de baze de date *in-house* numit UNIREG pentru gestiunea bazelor de date. UNIREG este o interfață teleimprimator care utilizează o conexiune low-level pentru depozitare ISAM cu indexare. De atunci, UNIREG a fost rescris folosind mai multe limbi de programare și a fost extins pentru a gestiona baze de date mari. Acesta este încă disponibil în prezent, dar este în mare parte înlocuit de MySQL.

Compania suedeză TcX a început dezvoltarea de aplicații web-based în 1994, folosind UNIREG. Din păcate, acest sistem de gestiune a datelor nu a îndeplinit cerințele aplicației web, iar TcX a început să caute alternative.

TcX a examinat SQL și mSQL. mSQL(miniSQL - Highes Technologies.) este un SGBD (Sistem de Gestiuare a Bazelor de Date) ieftin care era open-source. La acea vreme, mSQL era la versiunea 1.x și avea mult mai puține caracteristici decât versiunea disponibilă în prezent. mSQL nu suportă indecsi. Performanțele mSQL au fost multe mai mici în comparație cu cele ale UNIREG-ului și îi lipseau anumite caracteristici obligatorii pentru TcX.

Monty l-a contactat pe David Hughes, autorul mSQL, pentru a vedea dacă Hughes este interesat în combinarea mSQL cu UNIREG pentru a putea indexa datele în mSQL. Hughes deja începuse să lucreze la mSQL 2. Compania a testat unele servere comerciale, dar toate s-au dovedit a fi prea lente pentru tabele de mari dimensiuni ale firmei.

Monty a început să programeze un server nou, Interfața de programare era proiectată în mod explicit pentru pentru a fi similară celei folosite de mSQL, deoarece pentru mSQL erau disponibile numeroase instrumente gratuite, iar prin utilizarea unei interfețe similare aceleași instrumente puteau fi folosite pentru MySQL, cu un efort de portare minim.

Până în mai 1995, TcX a avut o bază de date care a satisfăcut nevoile sale interne: MySQL 3.11. Un partener de afaceri, David Axmark de la Detron HB, a început presarea TcX pentru a lansa MySQL pe Internet. De asemenea, David lucra la documentație și la a determina MySQL să construiască folosind utilitarul GNU configure. MySQL 3.11.1 a fost dat lumii întregi în 1996, sub forma de distribuție binară pentru Linux și Solaris, în prezent, MySQL funcționează pe mult mai multe platforme și este disponibil atât în formă binară, cât și sursă.

MySQL nu este un proiect Open Source, deoarece este necesară o licență în anumite condiții. Totuși, MySQL se bucură de o amplă popularitate în comunitatea Open Source, deoarece termenii de licență nu sunt foarte restrictivi, (în esență, MySQL este în general gratuit, dacă nu doriți să obțineți profit prin vânzarea sistemului sau a unor servicii care necesită utilizarea acestuia).

În ceea ce privește numele MySQL, Monty spunea “nu este foarte clar de unde provine denumirea MySQL. Multe dintre librăriile create de firma TcX aveau prefixul ‘My’ pentru mai bine de 10 ani. Cu toate acestea, fiica mea se numește ‘My’. Deci, originea numelui de MySQL rămâne un mister.”

Acum câțiva ani, TcX a fost absorbită de MySQL AB. Această schimbare a permis un mai bun control asupra dezvoltării și suportului pentru MySQL. MySQL AB, o companie suedeză condusă de fondatorii bazelor de date MySQL, deține dreptul de autor pentru MySQL. De la lansarea pe piață a acestui SGBD, se estimează că MySQL rulează pe circa 4 milioane de servere.[1]

În 2008, compania MySQL AB a fost cumpărată de Sun Microsystems pentru 1 miliard de dolari.

### **1.1.3 Notiuni generale**

Un server de baze de date (în cazul nostru MySQL) este un program care poate stoca o cantitate mare de informații într-un format organizat, care este ușor accesibil prin limbaje de programare cum ar fi PHP.

MySQL poate fi folosit atât pe platforme Windows, cât și Unix. MySQL este un SGBD sigur deoarece accesul la bazele de date se face prin autentificarea utilizatorului.

MySQL nu este un sistem de baze de date perfect, dar este suficient de flexibil pentru a lucra bine în medii foarte solicitante, cum ar fi aplicații web. De exemplu, MySQL poate fi configurat să ruleze bine pe o gamă largă de hardware și suportă o varietate de tipuri de date. Cu toate acestea, caracteristica cea mai neobișnuită și importantă este arhitectura „storage-engine”, al cărei design separă procesarea interogărilor de alte operații făcute de MySQL.

MySQL este folosit împreună cu limbajele de programare JAVA, PHP pentru a construi aplicații. Pentru a administra bazele de date MySQL se poate folosi modul linie de comandă sau o interfață grafică: MySQL Administrator sau MySQL Query Browser.

SQL din numele de MySQL este acronimul “Structured Query Language”. Acest limbaj este bazat pe limba engleză și este, de asemenea, utilizat în alte baze de date, cum ar fi Oracle și Microsoft SQL Server.

Serverul de MySQL este un manager al sistemului de baze de date. Este foarte rapid, fiabil și ușor de actualizat. El se ocupă de toate instrucțiunile bazei de date. De exemplu, dacă se dorește să se creeze o nouă bază de date, se trimit un mesaj către MySQL de forma „creează o nouă bază de date și numește-o „bazaDeDate””. Serverul de MySQL creează apoi un subdirector în directorul de date, numește acest subdirector „bazaDeDate” și introduce fișierele în formatul solicitat de subdirector. În același mod, pentru a adăuga date în baza de date, se trimit un mesaj la serverul de MySQL, oferindu-i datele necesare și locația în care se vor adăuga datele.

Înainte de a putea transmite instrucțiunile către serverul de MySQL, acesta trebuie să fie pornit și să fie în stare de aşteptare. De obicei, se setează serverul să pornească atunci când se pornește calculatorul și continuă să funcționeze tot timpul. Aceste sunt setările uzuale de configurare pentru o aplicație web. Cu toate acestea, nu este necesar ca serverul să pornească o dată cu calculatorul. Dacă este nevoie, acesta poate fi pornit manual ori de câte ori este nevoie ca o bază de date să fie accesată. Cât timp serverul MySQL este pornit, el ascultă continuu mesajele direcționate către el.[2]

MySQL este un sistem de gestiune de baze de date (SGBD). Serverul de MySQL poate administra mai multe baze de date în același timp. De fapt, mulți utilizatori ar putea avea baze de date diferite, dar gestionate de un singur server MySQL. O bază de date poate exista chiar dacă aceasta este goală. Datele dintr-o bază de date sunt stocate într-una sau mai multe tabele. Mai întâi se creează baza de date și tabelele pentru a adăuga orice intrare în MySQL. Când este creată, baza de date este goală. Apoi se adaugă tabelele în baza de date. Tabelele sunt structurate în rânduri și coloane. Fiecare rând reprezintă o entitate în baza de date. Fiecare coloană conține un element de informare despre entitate, cum ar fi numele unui client, numele unei cărți. Locul în care un anumit rând și coloană se intersectează, se numește câmp.

Tabelele în baza de date pot fi relaționate. Adesea, un rând într-un tabel este legat de mai multe rânduri dintr-un alt tabel. De exemplu, este posibil să avem o bază de date care conține date despre cărțile pe care le deținem. Ar putea exista o tabelă pentru cărți și o tabelă cu autori. Un rând în tabela autorilor poate conține informații despre autorul mai multor cărți din tabela cărților. Când tabelele sunt relaționate, trebuie să existe o coloană într-una din tabele pentru a putea corela datele din cele 2 tabele. Numai după ce s-a creat structura bazei de date se pot adăuga intrări.

Un bun design al bazei de date este crucial pentru performanța bazei de date. Dacă o bază de date este proiectată prost poate îngreuna execuția interogărilor.

Când se creează un tabel în MySQL, trebuie să specificăm tipul de date pentru fiecare coloană. Coloanele pot accepta parametrii optionali pentru a personaliza modul în care datele sunt stocate sau afișate în tabelă. Există două tipuri de parametri pentru afișarea datelor: M și D .

Parametrul M este folosit pentru a specifica dimensiunea de afișare (adică numărul maxim de caractere) pentru a fi utilizată de valorile din coloană. În cele mai multe cazuri, acest lucru va limita gama de valori care pot fi specificate în coloană. M este un număr întreg ce poate lua valori între 1

și 255. Pentru tipurile numerice (de ex. INT), acest parametru nu restricționează intervalul de valori care pot fi stocate. În schimb, poate cauza spații (sau zerouri în cazul unei coloane ZEROFILL) care urmează să fie adăugate la valoare, astfel încât aceasta să ajungă la lățimea afișajului dorit. În plus, stocarea unei valori mai mari decât limita impusă poate cauza probleme.[3]

Parametrul D permite specificarea numărului de zecimale care pot fi stocate pentru o valoare scrisă în virgulă mobilă. Acest parametru poate fi setat la un maxim de 30, dar se recomandă ca valoarea lui D să fie egală cu M-2.

#### 1.1.4 Tipuri de date

MySQL suportă o mare varietate de tipuri de date, iar alegerea corectă a tipului de date pentru datele care vor fi stocate este crucial pentru obținerea rezultatelor eficiente.

##### Tipuri de șiruri de caractere

MySQL suportă o gamă largă de tipuri de date String. Aceste tipuri de date s-au schimbat foarte mult în versiunile 4.1 și 5.0, ceea ce face complicată folosirea lor. Începând cu versiunea 4.1, fiecare coloană de tip String poate avea propriul set de caractere și propriul set de reguli de sortare. Acest lucru poate afecta performanțele bazei de date.

##### VARCHAR și tipuri de CHAR

Cele două tipuri de date de tip String VARCHAR și CHAR stochează caractere. Din nefericire, este greu de explicat exact modul în care sunt stocate aceste valori în memorie. Termenul de VARCHAR este un nume combinat: *VARi-able length CHARacter string*.

Acest tip de date este foarte util, deoarece astfel MySQL permite planificarea dimensiunii unei baze de date și efectuarea căutării mult mai ușoare. Dezavantajul este că, dacă s-ar încerca vreodată stocarea unei valori mai mare decât este posibil, aceasta va fi trunchiată la lungimea stabilită inițial.

Varchar stochează șiruri de caractere de lungime variabilă și este cel mai comun tip de date de tip String. Varchar utilizează 1 sau 2 octeți în plus pentru a stoca lungimea valorii dorite. 1 octet este folosit pentru stocare dacă lungimea maximă a coloanei este de 255 de octeți sau mai puțin și 2 octeți dacă lungimea este mai mare. Presupunând ca folosim setul de caractere Latin1, pentru un VARCHAR(10) se vor utiliza până la 11 octeți pentru stocare. Pentru un VARCHAR(1000) se pot utiliza până la 1002 octeți, pentru că este nevoie de 2 octeți pentru stocarea informațiilor privind lungimea.[4]

VARCHAR contribuie la performanțele bazei de date, deoarece economisește spațiu. Cu toate acestea, deoarece rândurile sunt de lungime variabilă, lungimea lor poate crește când sunt updateate, acest lucru ducând la o muncă suplimentară. Dacă lungimea unui rând crește și nu mai are loc în locația inițială, comportamentul este de stocare motor-dependentă. De exemplu, MyISAM poate fragmenta rândul, iar InnoDB ar putea fi nevoit să împartă pagina pentru ca rândul să aibă loc.

Este recomandată utilizarea VARCHAR atunci când lungimea maximă a coloanei este mult mai mare decât lungimea medie. În versiunea 5.0. sau în versiunile mai noi, MySQL păstrează la sfârșit spații când se stochează date. În versiunea 4.1 sau mai vechi, nu se adaugă spații.

Tipul de date CHAR este de lungime fixă. MySQL alocă întotdeauna spațiu suficient pentru numărul specificat de caractere. Când se stochează o valoare de tip CHAR, MySQL elimină toate spațiile finale. Valorile sunt indentate cu spații necesare pentru comparații.

CHAR este util dacă se dorește stocarea unor siruri de caractere foarte scurte, sau când se dorește stocarea unor date de aproximativ aceeași lungime. De exemplu, CHAR este o alegere bună pentru stocarea valorilor MD5 pentru parolele utilizatorilor, care sunt întotdeauna de aceeași lungime. CHAR reprezintă o alegere mai bună decât VARCHAR pentru date care se schimbă adesea, deoarece un rând de lungime fixă nu este predispus la fragmentare. Pentru coloane foarte scurte, CHAR este de asemenea mai eficient decât VARCHAR: CHAR(1) proiectat să stocheze doar valorile Y sau N va folosi doar un octet, iar VARCHAR(1) ar folosi 2 octeți.

Modul în care datele sunt stocate depinde de motoarele de stocare, și nu toate motoarele de stocare manipulează datele de lungime fixă sau de lungime variabilă în același mod. Memoria motorului de stocare utilizează rânduri de lungime fixă, acest lucru determinând alocarea spațiului maxim posibil pentru fiecare valoare chiar dacă lungimea unui câmp este mai mică. [5]

Tipuri de date	Octeți ocupăți	Exemple
CHAR(n)	n ( $\leq 255$ )	CHAR(5): "Hello" ocupa 5 octeți CHAR(57): "New York" ocupa 57 octeți
VARCHAR(n)	până la n ( $\leq 65535$ )	VARCHAR(100): "Buna" ocupa 4 octeți plus 1 octet extra VARCHAR(9): "Buna ziua" ocupă 9 octeți plus 1 octet extra

*Tabelul 1.1 Tipuri de date siruri de caractere în MySQL - Sursa [5]*

### Tipuri de date TEXT

Diferențele între TEXT și VARCHAR sunt mici:

- înainte de versiunea 5.0.3, MySQL a scos spațiile dinainte și după pentru câmpurile VARCHAR și astfel, pot ajunge doar până la 255 octeți în lungime
- câmpurile TEXT nu pot avea valori implice
- MySQL indexează doar primele n caractere dintr-un câmp TEXT (n este specificat când se creează indexul)

Toate aceste lucruri conduc la concluzia că VARCHAR este mai bun și mai rapid dacă este nevoie de căutare în întregul conținut al unui câmp. Dacă nu este necesară căutarea a unor caractere având lungimea mai mare decât un anumit număr de caractere, atunci este recomandată folosirea tipului de date TEXT.[6]

Tipuri de date	Octeți ocupați
TINYTEXT(n)	până la n ( $\leq 255$ )
TEXT(n)	până la n ( $\leq 65535$ )
MEDIUMTEXT(n)	până la n ( $\leq 16777215$ )
LONGTEXT(n)	până la n ( $\leq 4294967295$ )

*Tabelul 1.2 Tipuri de date TEXT în MySQL - Sursa [5]*

### Tipuri de date BINARY și VARBINARY

Tipuri asemănătoare VARCHAR și CHAR sunt BINARY și VARBINARY, care stochează siruri de caractere binare. Sirurile binare sunt foarte asemănătoare cu sirurile convenționale, dar ele păstrează octeți în loc de caractere. Indentarea este, de asemenea, diferită: MySQL adaugă la valorile binare „0”(octet zero) în loc de spații.

Acste tipuri sunt folositoare atunci când se dorește stocarea datelor de tip binar și când este nevoie să se compare valorile respective de către MySQL. MySQL compară octet cu octet sirurile. Ca urmare, compararea binară poate fi mult mai simplă decât compararea caracterelor, ceea ce înseamnă ca se realizează mult mai repede.

Tipuri de date	Octeți utilizați	Exemple
BINARY(n) sau BYTE(n)	n ( $\leq 255$ )	Ca și la CHAR, dar conține date binare
VARBINARY (n)	până la n ( $\leq 65535$ )	Ca și la VARCHAR, dar conține date binare

*Tabelul 1.3 Tipuri de date binare în MySQL - Sursa [5]*

### Tipul de date BLOB

Termenul BLOB este acronimul pentru Binary Large OBject, și, prin urmare este utilizat pentru date binare care depășesc 65636 octeți în lungime. Diferența majoră între BLOB și BINARY este că BLOB-urile nu pot avea valori implicate.

Tipuri de date	Octeți ocupați
TINYBLOB (n)	până la n ( $\leq 255$ )
BLOB (n)	până la n ( $\leq 65535$ )
MEDIUMBLOB (n)	până la n ( $\leq 16777215$ )
LONGBLOB (n)	până la n ( $\leq 4294967295$ )

*Tabelul 1.4 Tipuri de date BLOB în MySQL - Sursa [5]*

## Tipuri de date numerice

MySQL acceptă diverse tipuri de date numerice, de la un singur octet până la dublă precizie în virgulă mobilă. Deși datele de tip numeric pot ocupa cel mult 8 octeți, este de preferat să se aleagă cel mai mic tip de date care va ocupa valoarea cea mai mare pe care o poate lua variabila respectivă. Acest lucru va ajuta la menținerea bazei de date mici, rapidă și ușor de accesat.

Tabelul 1.5 arată tipurile de date numerice suportate de MySQL și intervalele de valori pe care le poate conține.

Tipul de date	Octeți utilizați	Valoare minimă (cu/fără semn)	Valoarea maximă (cu/fără semn)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT sau INTEGER	4	0	4294967295
BIGINT	8	-9.22337E+18	9.22339E+18
		0	1.84467E+19
FLOAT	4	-3.40E+38	3.40E+38
		-	-
DOUBLE sau REAL	8	-1.7976931348623157E+308	1.7976931348623157E+308

*Tabelul 1.5 Tipuri de date numerice în MySQL - Sursa [5]*

## Tipuri de date DATE și TIME

Tipuri de date	Format
DATETIME	‘0000-00-00 00:00:00’
DATE	‘0000-00-00’
TIMESTAMP	‘0000-00-00 00:00:00’
TIME	‘00:00:00’
YEAR	0000 (Doar anii 0000 s9 1901 - 2155)

*Tabelul 1.6 Tipuri de date DATE și TIME în MySQL - Sursa [5]*

Aceste tipuri de date se referă la dată și timp. Tipurile DATETIME și TIMESTAMP sunt afișate în același mod. Principala diferență este ca TIMESTAMP are o gamă îngustă de valori (1970 - 2037), pe când DATETIME poate conține aproape orice dată dorită. Cu toate acestea, TIMESTAMP este folositoare pentru că MySQL știe să completeze acest câmp singur. Dacă nu se specifică valoarea câmpului TIMESTAMP, acesta este completat automat.[7]

## Tipul de date AUTO\_INCREMENT

Uneori este nevoie să fii sigur că ai intrări unice în baza de date. Acest lucru se poate face verificând cu multă atenție datele inserate în baza de date, însă pot exista mici greșeli de neatenție, rezultând astfel una sau mai multe intrări identice în baza de date. Soluția optimă este de a utiliza o coloană suplimentară doar pentru a putea diferenția intrările. O coloană de tip AUTO\_INCREMENT stabileste valoarea intrării în baza de date astfel: adaugă o unitate față de valoarea aceluiasi camp al intrării anterioare. Practic valoarea pentru o coloană setată cu AUTO\_INCREMENT este autoincrementată la fiecare inserare în baza de date a unui rând nou.[8]

Ex: Adăugarea unei coloane cu autoincrementare

*ALTER TABLE nume ADD id INT UNSIGNED NOT NULL AUTO\_INCREMENT KEY;*

Operația de ALTER funcționează pe un tabel existent, și poate adăuga, modifica sau șterge coloane. Exemplul nostru adaugă o coloană numită „id” cu următoarele caracteristici:

- INT UNSIGNED - coloana este de tip INTEGER și poate stoca peste 4 miliarde de intrări în baza de date;
- NOT NULL - asigură faptul că fiecare coloană va avea o valoare. Mulți programatori folosesc NULL pentru anumite câmpuri care nu au valoare, dar acest lucru poate duce la intrări identice, ceea ce anulează scopul coloanei „id”. Astfel, vom elimina posibilitatea de a exista valori NULL pentru acest câmp;
- AUTO\_INCREMENT - face ca MySQL să stabilească o valoare unică pentru această coloană pentru fiecare rând existent. Nu avem control asupra acestei valori. Conțea că faptul că această valoare va fi unică.

- KEY - o coloană cu autoincrementare este utilă ca și cheie. Putem căuta anumite rânduri bazându-ne pe valoarea aceste coloane. Fiecare intrare va avea un identificator unic, un număr care va începe de la valoarea 1 și se va incrementa pe măsură ce se vor stoca intrări în baza de date.[9]

## Chei primare

Intrările într-o tabelă necesită existența unei chei primare, unice. Acest lucru este folositor atunci când se vor combina date din mai multe tabele. Mai devreme am exemplificat tipul de date AUTO\_INCREMENT, care poate deveni o cheie primară.[10]

*ALTER TABLE classics ADD id INT(13) PRIMARY KEY;*

Pentru a crea cheia primară atunci când este creată tabelă, putem utiliza comanda din exemplul de mai jos.

```
CREATE TABLE studenti ( nume VARCHAR(128),
prenume VARCHAR(128),
facultate VARCHAR(16),
an SMALLINT,
id INT(13),
INDEX(nume(20)),
INDEX(prenume(20)),
INDEX(facultate(4)),
INDEX(an),
PRIMARY KEY (id)) ENGINE MyISAM;
```

## 1.2 NoSQL

O bază de date de tip NoSQL prevede un mecanism diferit de stocare și interogare a datelor față de bazele de date tradiționale relaționale. Acest mecanism este folosit pentru a obține scalarea orizontală și disponibilitate crescută. Unii utilizatori se referă la acest sistem de baze de date prin „Not only SQL” pentru a evidenția că unele sisteme din familia NoSQL permit interogări de tip SQL.

Sistemele de baze de date de acest tip sunt foarte bine optimizate în cazul interogărilor. Viteza cu care lucrează sistemele NoSQL este mult mai mare față de sistemele clasice SQL, ceea ce asigură o mai bună scalabilitate și performanță.

Sistemele de gestionare NoSQL sunt utile atunci când se lucrează cu o cantitate mare de date și când nu este nevoie de un sistem relațional. Datele pot fi structurate, dar NoSQL este utilizat atunci când cel mai important lucru este abilitatea de a stoca și returna o mare cantitate de date, iar relația dintre elemente nu este atât de importantă. Un exemplu de utilizare ar fi atunci când se dorește stocarea a milioane de perechi cheie-valoare într-un tablou asociativ sau atunci când este nevoie de stocarea a miliarde de intrări. Această organizare este folosită pentru analiza în timp real a unor liste care cresc continuu, cum ar fi mesajele Twitter.

Bazele de date NoSQL sunt tot mai folosite în aplicații web 2.0, în rețele de socializare, unde datele sunt de cele mai multe ori generate de utilizatori. Din cauza diversității utilizatorilor este dificil să maparea conținutului utilizatorilor într-o bază de date relațională, pentru că schema trebuie să fie cât mai flexibilă. Cu cât aplicațiile web sau rețelele de socializare devin mai populare, cu atât crește cantitatea de date care trebuie stocată în baza de date. Dacă se folosește un sistem de baze de date relațional, stocarea informațiilor devine dificilă și putem întâmpina probleme. De aceea web-site-urile populare - Facebook, Twitter folosesc un sistem de baze de date de tip NoSQL.

Bazele de date NoSQL reprezintă o trecere către baze de date superioare ce vor integra flexibilitatea și performanțele lor actuale cu modelul relațional. Odată cu apariția bazelor de date NoSQL, dezvoltatorii au oportunitatea de a beneficia de mai multă agilitate în modelul de date abordat. De asemenea aceste baze de date constituie modelul optim pentru aplicațiile web. De aceea cunoașterea caracteristicilor lor este foarte importantă, în special înainte de a migra la o astfel de soluție.

Există 4 tipuri de baze de date NoSQL:

- Key - value (cheie - valoare) - Informația este stocată în pereche - cheie - valoare. Poate fi privit ca un dicționar cu o cheie unică, valoarea putând fi orice. Redis, Dynamite, Voldemort sunt baze de date care au această structură.
- Document - based (orientat pe documente) - Datele sunt stocate și organizate sub forma unei colecții de documente (XML sau JSON). Documentele sunt flexibile, fiecare document poate avea orice număr de câmpuri. Apache CouchDB și MongoDB sunt cele mai populare baze de date de acest tip.
- Column - based (orientat pe coloane) - Acest tip de baze de date stochează informația în tabele, similar cu RDBMS. Informația este stocată în coloane, și nu sub formă de rânduri. Cassandra, Hypertable Hbase sunt cele mai utilizate baze de date de acest tip.
- Graph - oriented (orientat pe grafuri) - Pentru acest tip de baze de date este folosită teoria grafurilor pentru stocarea datelor și interogarea bazelor de date. Suportă stocarea datelor foarte complexe. Neo4j este un exemplu de baze de date de acest tip.

## 1.2.1 MongoDB

### 1.2.1.1 Introducere

MongoDB (numele provine de la „humongous” - enorm) este un sistem de baze de date open-source orientat pe documente, dezvoltat de compania 10gen. Acest sistem de baze de date face parte din familia NoSQL. Spre deosebire de sistemele de baze de date clasice relaționale care stochează informația în tabele, MongoDB stochează datele sub forma documentelor de tip JSON, având o schемă dinamică. Mongo numește acest format de stocare BSON.

Acest sistem de stocare face ca înregistrarea datelor să fie mai ușoară, dar și mai rapidă. 10gen a început dezvoltarea acestui sistem de baze de date în octombrie 2007. MongoDB este utilizat de MTV Networks, Craigslist, Foursquare și UIDAI Aadhaar. MongoDB este cel mai popular sistem de baze de date din familia NoSQL.

MongoDB este ușor de învățat. Există totuși similarități între conceptele bazelor de date MongoDB și conceptele bazelor de date de tip relațional. Dezvoltatorii care au utilizat RDBMS și migrează către MongoDB pot întâmpina diverse probleme de adaptabilitate.

Implementează ideea unui design flexibil al bazei de date. Nu trebuie definită structura bazei de date înainte de a stoca informațiile, ceea ce este folositor atunci când trebuie să stocăm date nestructurate. Acest sistem este foarte scalabil. Are foarte multe opțiuni pentru a menține performanțe optime, în timp ce dimensiunea și traficul datelor cresc, fără efectua schimbări în structura aplicației.

MongoDB este gratis și poate fi downloadat și utilizat fără a plăti. Are o documentație foarte bine pusă la punct, iar utilizatorii acestui sistem de baze de date contribuie la dezvoltarea lui în permanență.

### 1.2.1.2 Scurt istoric

MongoDB a fost creat de către fondatorii DoubleClick, proiect care a fost cumpărat de către Google pentru suma de 3.1 miliarde de dolari. După vânzarea acestui proiect, echipa a început să lucreze la numeroase proiecte, dar au întâmpinat probleme de scalabilitate.

În 2007 au fondat compania 10gen și au început să lucreze la o platformă pentru cloud, asemănătoare cu Google App Engine. Platforma celor de la 10gen era dezvoltată server-side, folosind JavaScript. Aceasta platformă a fost numită inițial „ed”, acest nume fiind compus din inițiala numelor celor 2 asociați - Elliot și Dwight, iar baza de date a fost numită „p”. În vara anului 2008 au decis să schimbe numele acestor două proiecte, Sistemul de baze de date a fost redenumit „Mongo”, iar platforma pentru cloud a primit numele „Babble”.

După un an de zile au renunțat la platforma Babble și au transformat Mongo într-un proiect open-source. După acest pas, sistemul de baze de date a început să devină popular printre utilizatori. În ultimul an, MongoDB s-a dezvoltat foarte mult, deoarece numărul de utilizatori a crescut exponential. Fiind un sistem de baze de date open-source, utilizatorii au customizat sistemul, creând proiecte proprii: Casbah, Morphia, MongoMapper, Mongoose, CandyGram, MongoKit, Mongoid, Ming, MongoEngine, Pymongo-Bongo, ActiveMongo, morph și MongoRecord. Utilizatorii integreză acest sistem de baze de date în numeroase proiecte deja existente: Drupal, Doctrine, Django, ActiveRecord, Lighttd sau NGINX.

Cine utilizează MongoDB?

- **Craiglist** - este unul dintre cele mai populare web-site-uri de anunțuri gratuite. Folosește MongoDB pentru a stoca miliarde de înregistrări. Pentru această aplicație web s-a folosit inițial MySQL. Migrarea către MongoDB le-a permis schimbarea design-ului pentru a obține o mai bună scalabilitate.
- **Foursquare** - este o rețea de socializare bazată pe locație. Stocă informații despre locațiile pe hartă a punctelor de interes (restaurante, cafenele, etc.) și informații despre utilizatorii care vizitează aceste locații. Utilizează MongoDB pentru stocarea datelor.
- **CERN** - Renumitul laborator de fizica particulelor localizat în Geneva utilizează MongoDB pentru agregarea datelor necesare pentru experimentul „Large Hadron Collider”. Rezultatele

interrogărilor, efectuate pe cantități masive de date sunt stocate cu ajutorul MongoDB pentru o eventuală utilizare în viitor.

### 1.2.1.3 Conceptele MongoDB

Un server MongoDB găzduiește mai multe baze de date. Bazele de date pot fi comparate cu niște containere și sunt independente unele de altele. O bază de date MongoDB conține una sau mai multe colecții. O colecție reprezintă un set de documente. Acest concept este analog din punct de vedere logic cu structurarea bazelor de date relaționale. Spre deosebire de tabelele din bazele de date relaționale, în MongoDB nu trebuie să definim structura datelor ce urmează să fie stocate în prealabil. Un document stocat într-o colecție este o unitate de date.

În MongoDB, obiectul principal se numește document. Documentele nu au o schemă predefinită, cum este în cazul tabelelor din bazele de date relaționale. Un document este asemănător cu un tablou multidimensional. Într-o matrice putem avea un set de chei care mapează valorile. Valorile pot fi ele însese un alt tablou. Documentul MongoDB este o matrice JSON.[11]

Un document conține un set de câmpuri sau perechi de tip cheie-valoare. Cheile sunt siruri de caractere, iar valorile pot fi de mai multe tipuri: siruri de caractere, numere întregi, numere reale, tipuri de date timestamp, etc. Se poate stoca până și un întreg document ca valoare a unui câmp într-un alt document.

### 1.2.1.4 Anatomia unui document din MongoDB

Exemplul de mai jos stochează date despre un utilizator într-o aplicație web:

```
{  
  "_id": ObjectId("4db31fa0ba3aba54146d851a")  
  "username": "joegunchy"  
  "email": "joe@mysite.org"  
  "age": 26  
  "is_admin": true  
  "created": "Sun Apr 24 2011 01:52:58 GMT+0700 (BDST)"  
}
```

Acest document are șase câmpuri. Organizarea datelor este identică cu sintaxa JSON sau JavaScript Object Notation. Valoarea primului câmp, `_id`, este generată într-un mod automat. MongoDB generează automat un Objectid pentru fiecare document creat într-o colecție și asignează valoarea acestuia câmpului `_id` al documentului. Acest id este unic, ceea ce înseamnă că oricare două documente aflate în aceeași colecție nu vor avea aceleași valori pentru câmpul `_id`, acest lucru semanând cu conceptul de cheie primară din bazele de date relaționale. Următoarele două câmpuri, `username` și `email` conțin date de tipul string (sir de caractere), câmpul `age` conține o valoare de tip integer, `is_admin` conține date de tip boolean. Ultimul câmp stochează date de tipul JavaScript Date Time, reprezentat ca un sir de caractere.

### 1.2.1.5 BSON - formatul de date de schimb în MongoDB

Am observat mai sus că structura unui document imită un obiect JSON. Când stocăm un document în baza de date, acesta este serializat într-un format special binar codificat, cunoscut sub numele de

BSON, prescurtarea pentru JSON binar. Avantajul formatului BSON este faptul este mult mai eficient decât formatele convenționale, cum ar fi XML și JSON. De asemenea, BSON suportă toate tipurile de date suportate de JSON(șiruri de caractere, numere întregi, numere reale, Boolean, vectori, obiecte, NULL), plus unele tipuri de date speciale, cum ar fi expresii regulate, object ID, dată, dată binară și cod. Limbajele de programare cum ar fi PHP, Python, Java au biblioteci care gestionează conversia de structuri de date specifice limbajului. Acest lucru permite limbajului să comunice cu MongoDB și să manipuleze datele cu ușurință.

### 1.2.1.6 Tipuri de date

MongoDB suportă o gamă largă de tipuri de date pentru valorile din documente. Documentele din MongoDB au un format asemănător cu formatul JSON. Acest format este ușor de înțeles și de parsat. Pe de altă parte, capacitatele formatului JSON sunt limitate, deoarece tipurile de date sunt: null, boolean, numerice, șiruri de caractere, vectori și obiecte.

Chiar dacă aceste tipuri de date sunt suficiente pentru majoritatea aplicațiilor, există anumite tipuri de date care lipsesc și care în anumite aplicații nu pot lipsi. De exemplu, JSON nu suportă data ca și tip de dată. Nu există mai multe tipuri de date numerice, astfel că nu se poate diferenția un număr întreg de un număr real. Totuși, MongoDB suportă alte tipuri de date esențiale de tipul cheie - valoare. [12]

Tipurile de date suportate de acest format:

*null* - acest tip poate fi folosit pentru a reprezenta o valoare nulă, dar poate fi folosit și pentru o valoare care nu există

{“x” : null}

*boolean* - poate fi folosit pentru valori de tipul *true* sau *false* (adevărat sau fals)

{“x” : true}

*32-bit integer, 64-bit integer* - aceste formate nu pot fi reprezentate în consolă. JavaScript suportă doar numere reprezentate pe 64 de octeți în virgulă mobilă. Așadar, aceste tipuri vor fi convertite în 64 de octeți.

*64-bit floating point number* - numere reprezentate pe 64 de octeți în virgulă mobilă - toate numerele vor fi de acest tip

{“x” : 3.14}, {“x” : 3}

*string* - orice șir de caractere de tip UTF-8 poate fi reprezentat de *string*

{“x” : “foobar”}

*symbol* - acest tip nu poate fi reprezentat în consolă. Dacă din baza de date se va primi un simbol, acesta va fi convertit într-un string.

*object id* - un id obiect este un ID pentru documente reprezentat pe 12 octeți

{“x” : ObjectId()}

*date* - datele sunt stocate sub forma milisecundelor de la începutul epocii.

```
{“x” : new Date()}
```

*expresii regulate* - documentele pot conține expresii regulate, utilizând sintaxa JavaScript

```
{“x” : /foobar/i}
```

*cod* - documentele pot conține cod JavaScript

```
{“x” : function() /* ... */ }
```

*date binare* - datele binare sunt siruri de octeți arbitrare. Nu pot fi manipulate utilizând consola.

*valoare maximă* - BSON conține un tip special de date care reprezintă cea mai mare valoare posibilă. Consola nu poate reprezenta acest tip.

*valoare minimă* - BSON conține un tip special de date care reprezintă cea mai mică valoare posibilă. Consola nu poate reprezenta acest tip.

*undefined* - această valoare poate fi folosită în documente. JavaScript are un tip distinct pentru *null* și *undefined*.

```
{“x” : undefined}
```

*array* - tablou - seturile de date sau liste de date pot fi reprezentate utilizând acest tip de date

```
{“x” : [“a”, “b”, “c”]}
```

*embedded document* - documentele pot conține alte documente

```
{“x” : {“foo” : “bar”}}
```

## Numere

JavaScript are un tip de date numit „number” (număr). Din cauză că MongoDB are 3 tipuri de numere (întregi pe 4 octeți, întregi pe 8 octeți și numere reale pe 8 octeți), consola trebuie să găsească o modalitate de a elobi limitările impuse de JavaScript. Prin definiție, orice număr scris în consolă este tratat ca și un număr *double* de către MongoDB. Acest lucru înseamnă că dacă din baza de date este întors un întreg reprezentat pe 4 octeți și apoi va fi transformat într-un număr reprezentat în virgulă mobilă.

## Data

În JavaScript, obiectul Date este utilizat pentru tipul de data MongoDB. Pentru a crea un nou obiect de tip *Date* folosim *new Date()*. Apelând constructorul pe post de funcție este returnat o reprezentare a datei, nu obiectul *Date* în sine.

## Arrays - tablouri

Valorile conținute de vectori pot fi de mai multe tipuri - siruri de caractere, numere, etc. Într-un array poate fi stocat un alt array. Un avantaj pe care îl are un vector în MongoDB este că acest sistem de baze de date știe să interpreteze și să înțeleagă structura lui.

## Embedded documents

Documentele incorporate sunt documente MongoDB întregi, care sunt folosite ca și valoare pentru o cheie într-un alt document. Ele pot fi folosite pentru a organiza datele într-un mod mult mai natural.

### \_id și ObjectIds

Fiecare document stocat în MongoDB trebuie să aibă o cheie „\_id”. Valoarea acestei chei poate fi de orice tip, dar predefinit este *ObjectId*. Într-o singură colecție, fiecare document trebuie să aibă o valoare unică pentru „\_id”, acest lucru asigurând că fiecare document poate fi identificat în mod unic.

*ObjectId* este tipul implicit pentru câmpul „\_id”. Este mult mai ușor să folosim acest tip de cheie, în detrimentul cheilor tradiționale primare, deoarece nu se mai consumă timp pentru sincronizarea tuturor cheilor autoincrementabile între mai multe servere.

### 1.2.1.7 Operații folosite de MongoDB pentru lucrul cu consola

Cele 4 operații clasice cu ajutorul cărora manipulăm și vizualizăm date în consolă sunt: crearea, citirea, actualizarea și ștergerea.

#### Crearea documentelor

Funcția *insert* adaugă un document la o colecție. De exemplu, dacă am dori să stocăm un articol, mai întâi se creează o variabilă locală, numită *articol*. Această variabilă va fi un obiect de tip JavaScript, care va reprezenta documentul. Câmpurile acestui document vor fi: „titlu”, „conținut” și „data”.

```
articol = {"titlu": "Articol",
           "continut": "Acesta este articolul din ziar,"
           "data": new Date()}

{"titlu": "Articol",
 "continut": "Acesta este articolul din ziar.",
 "date": "Sat Jun 12 2013 11:23:21 GMT -0500 (EST)"}
```

Acest obiect este un document valid MongoDB, deci poate fi stocat în colecția „ziare”, folosind metoda *insert*:

```
db.ziare.insert(articol)
```

Articolul a fost salvat în baza de date. Putem să o vizualizăm folosind metoda *find*:

```

db.ziare.find()
{
    "_id" : ObjectId("4b23c3ca7525f35f94b60a2d"),
    "title" : "Articol",
    "content" : "Acesta este articolul din ziar.",
    "date" : "Sat Jun 12 2013 11:23:21 GMT - 0500 (EST)"
}

```

Câmpul „\_id” a fost adăugat în mod automat, iar ordinea celorlalte perechi cheie-valoare a fost păstrată.

## Citirea documentelor

Funcția *find* returnează toate documentele dintr-o colecție. Dacă dorim să vizualizăm un singur document dintr-o colecție putem să folosim funcția *findOne*:

```

db.ziare.findOne()
{
    "_id" : ObjectId("4b23c3ca7525f35f94b60a2d"),
    "title" : "Articol",
    "content" : "Acesta este articolul din ziar.",
    "date" : "Sat Jun 12 2013 11:23:21 GMT - 0500 (EST)"
}

```

## Actualizarea documentelor

Dacă dorim să modificăm articolul de mai sus, putem folosi funcția *update*. Această funcție primește (cel puțin) doi parametrii: primul este criteriul pentru căutarea documentului pe care dorim să îl modificăm, iar al doilea este noul document. Dacă dorim să adăugăm comentarii la articolul creat mai sus, creem un vector cu comentarii:

```
articol.comentarii = [ ]
```

Apelăm funcția *update*, modificând titlul:

```
db.ziare.update({titlu : "Articol"}, articol)
```

În momentul de față, noul document conține și câmpul „comentarii”. Dacă vom apela funcția *find*, vom avea următorul rezultat:

```

db.ziare.find()
{
    "_id" : ObjectId("4b23c3ca7525f35f94b60a2d"),
    "title" : "Articol",
    "content" : "Acesta este articolul din ziar.",
    "date" : "Sat Jun 12 2013 11:23:21 GMT - 0500 (EST)"
    "comentarii" : [ ]
}

```

## Ștergerea documentelor

Funcția *delete* șterge permanent din baza de date. Dacă această funcție este apelată fără parametrii, se vor șterge toate documentele din colecție. Putem să ștergem documentele pe baza unor criterii:

```
db.ziare.remove({titlu : "Articol"})
```



## Capitolul 2: PHP

### 2.1 Introducere

„PHP este un limbaj de programare. Acest nume provine din limba engleză și este un acronim recursiv: **Php**: Hypertext Preprocessor. În principiu, acest limbaj de programare este folosit pentru crearea paginilor/aplicațiilor web dinamice, dar, începând cu versiunea 4.3.0, poate fi folosit și în modul „linie de comandă”. Este unul din cele mai importante limbaje de programare web open-source și server-side. PHP însemna inițial *Personal Home Page*.<sup>1</sup>”

PHP rulează pe toate sistemele de operare importante, de la variante UNIX, incluzând Linux, FreeBSD, Ubuntu, Debian și Solaris, până la Windows și MacOSX. Poate fi folosit pe servere Apache, Microsoft IIS și Netscape/iPlanet.

Limbajul este unul flexibil. Poate genera o mulțime de documente: PDF, GIF, JPEG și PNG, dar și filme Flash. Una din caracteristicile importante ale PHP-ului este posibilitatea de a comunica cu bazele de date. PHP poate fi integrat cu foarte multe tipuri de baze de date: MySQL, PostgreSQL, Oracle, Sybase, MS-SQL, DB2. Chiar și sistemele de baze de date mai recente sunt suportate de PHP, cum ar fi MongoDB.

PHP deține o librărie cu cod PHP pentru a ajuta programatorul să efectueze sarcini obișnuite, cum ar fi conectarea la o bază de date, corectarea erorilor, etc.

„PHP este un limbaj server-side. Un limbaj server-side este similar cu JavaScript și permite integrarea unor mici programe (script-uri) în codul HTML al unei pagini/aplicații web. Când se execută, aceste programe oferă un control mai mare față de HTML, asupra a ceea ce va apărea în fereastra browser-ului. Diferența esențială dintre JavaScript și PHP este stadiul de încărcare a paginii web în care aceste programe sunt executate.[13]

Codul scris în limbaje client-side, cum ar fi JavaScript, este citit și executat de către browser după ce pagina web este descărcată de pe server. În schimb, codul scris în limbajele server-side (PHP), este executat de către serverul web, înainte de a trimite pagina web browser-ului. În timp ce limbajele client-side oferă control asupra modului în care o pagină web se comportă atunci când este afișată în browser, limbajele server-side permit generarea paginilor personalizate înainte ca acestea să fie trimise către browser. După ce serverul web a executat codul PHP incorporat în pagina HTML, rezultatul ia locul codului PHP în pagină.” [14]

### 2.2 Scurt istoric

Rasmus Lerdorf a creat PHP în 1994, dar varianta actuală este foarte diferită de cea inițială. Lerdorf a scris o serie de scripturi Perl Common Gateway Interface (CGI), pe care le utiliza la administrarea propriului web-site. Aceste scripturi erau utilizate pentru a afișa diferite informații, dar și pentru a înregistra traficul pe pagina sa personală. El a rescris aceste script-uri în C pe motive de performanță, modificându-le astfel încât să poată lucra cu formulare web, dar și pentru a se putea conecta la baze de date. A redenumit acest proiect „Personal Home Page/Forms Interpreter” sau PHP/FI. PHP/FI putea fi utilizat pentru a construi aplicații web simple, dinamice. Ulterior, când a lansat prima versiune, a adăugat și posibilitatea de a integra sintaxa HTML. Sintaxa PHP era similară cu sintaxa Perl.

Zeev Suraski și Gutmans Andi au rescris limbajul în 1997 și astfel, au lansat PHP 3, schimbând numele limbajului în acronimul recursiv PHP: Hypertext Preprocessor. După o perioadă de testare, au lansat versiunea oficială în iunie 1998. Suraski și Gutmans au început să modifice nucleul PHP, producând motorul Zend în 1999. Au fondat Zend Technologies, având sediul în Ramat Gen, în Israel.

În august 2008, PHP 4 a fost lansat de către Zend. În iulie 2004, PHP 5 a fost lansat. Această versiune a adus noi îmbunătățiri, suportând programarea obiect orientată.

## 2.3 Sintaxa PHP

Sintaxa PHP este foarte asemănătoare cu sintaxa JavaScript, C, C++, C#, Objective-C, Java, Perl. Numele claselor și funcțiilor definite de utilizator, precum și unele cuvinte cheie (echo, while, class) sunt nu țin cont de majusculă (case-insensitive). Numele variabilelor sunt sensibile la majusculă (case-sensitive).

Un script PHP constă dintr-o serie de comenzi, instrucțiuni sau declarații. Fiecare linie de comandă este o instrucțiune adresată server-ului web, pe care acesta trebuie să o execute pentru a putea trece la următoarea instrucțiune. Fiecare comandă, instrucțiune sau declarație este termintă cu simbolul „;”. O instrucțiune complexă folosește accoladele pentru a marca un bloc de cod.

Simbolul „\$” este folosit în multe limbaje de programare, sub diverse forme. În PHP, acest simbol trebuie scris înaintea tuturor variabilelor. Acest lucru este necesar pentru a face mai rapidă parsarea codului.

Spre deosebire de alte limbaje de programare, cum ar fi Python, care sunt stricte în privința identării, PHP nu ține cont de acest lucru. Totuși, identarea codului este recomandată pentru a-l înțelege mai bine și pentru a găsi mai ușor eventualele greșeli de sintaxă.

Funcțiile sunt utilizate pentru a separa secțiuni de cod care realizează o anumită prelucrare a codului. Funcțiile sunt de obicei realizate în cazul în care aceeași operație se va executa de mai multe ori în același program. Prin folosirea funcțiilor vom economisi timp și linii de cod. Dacă nu vom transforma un anumit bloc de cod într-o funcție, iar acel bloc de cod este folosit de mai multe ori în program, și vom decide să modificăm o instrucțiune, va fi destul de greu pentru programator să modifice blocul de cod oriunde apare. Transformarea blocurilor de cod identice în funcții surtează codul sursă, îl face mai ușor de citit, dar și mai rapid.

Dacă avem un program cu foarte multe linii de cod, este posibil să atribuim același nume de variabilă de mai multe ori. Folosind PHP, avem posibilitatea de a decide în ce scop putem declara și accesă o variabilă. De exemplu, putem spune ca variabila \$temp poate fi utilizată doar în interiorul unei funcții. În PHP putem declara și variabile locale.

## 2.4 OOP

Programarea orientată pe obiecte facilitează design-ul programelor. „POO este unul dintre cei mai importanți pași făcuți în evoluția limbajelor de programare spre o abstractizare în implementarea programelor. A apărut din necesitatea exprimării problemei într-un mod natural ființei umane.” [15]

POO înțelege legătura fundamentală între date și codul care funcționează cu aceste date și permite proiectarea și implementarea programelor în jurul acestei legături. „De exemplu, un sistem pentru

evidență unor utilizatori ai unei platforme web păstrează date despre foarte multe persoane. Într-un limbaj de programare procedurală, fiecare persoană reprezintă o structură de date și are atașată o listă de funcții care interacționează cu structura respectivă de date (crearea de noi persoane etc.). Într-un limbaj de programare orientat pe obiecte, fiecare utilizator reprezintă un obiect - o structură de date care conține și cod.

Un obiect de tip „persoană” conține numele utilizatorului și parola aferentă, dar și un mecanism de recunoaștere a mesajelor postate de utilizator. Un obiect de tip mesaj știe de către ce thread/discuție aparține. Un obiect de tip discuție este o colecție de obiecte de tip mesaj.

Obiectul, ca uniune de cod și date, este unitatea modulară pentru dezvoltarea aplicațiilor și reutilizarea codului. Axându-ne pe exemplul de mai devreme, avem nevoie de aceleași informații pentru fiecare utilizator. Când proiectăm aplicația, definim câmpurile și funcțiile pentru fiecare utilizator. În termeni de programare obiect-orientată, descriem clasa *utilizator*.

Un *obiect* este o instanță a unei clase. Obiectele și clasele seamănă cu valorile și tipurile de date. Există doar un tip de date integer, dar mai multe numere întregi posibile. În mod similar, programul definește o clasă *utilizator*, dar pot fi creați mai mulți utilizatori.

Datele asociate unui obiect se numesc *proprietăți*. Funcțiile asociate unui obiect se numesc *proprietăți*. Când se definește o clasă, sunt definite și proprietățile sale, dar și metodele sale.

Depanarea și întreținerea programului sunt mult mai ușor de realizat dacă se folosește incapsularea. O clasă are anumite metode care sunt folosite de obiecte, iar codul exterior acestei clase nu are acces direct la structura datelor clasei.

În programarea obiect orientată întâlnim noțiunea de *moștenire*. Aceasta este un mod de a defini o nouă clasă, spunând că este o clasă existentă, dar cu anumite proprietăți și metode noi sau modificate. Vechea clasă se numește *superclasă* (sau părinte sau clasă de bază), iar noua clasă se numește *subclasă* (sau clasă derivată). Moștenirea este o formă de reutilizare a codului de bază. Astfel, codul de bază este reutilizat în loc de a fi recopiat în noua clasă. Orice modificări facute în clasa de bază sunt automat și în clasa derivată.

Pentru a crea un obiect al unei clase existente putem folosi:

```
$object = new Class;
```

Presupunând că s-a definit clasa *Persoana*, vom crea un obiect de tip *Persoana*:

```
$persoana = new Persoana;
```

O dată creat obiectul, putem folosi operatorul „->” pentru a accesa metodele și proprietățile obiectului:

```
$object->proprietate  
$object->metoda([arg, ... ])
```

Metodele se comportă ca și nițte funcții (specifice obiectului în cauză); ele pot primi argumente și pot returna valori. În definiția unei clase putem specifica ce metode și ce proprietăți sunt publice, dar și care dintre ele sunt accesibile doar în interiorul clasei, folosind modificatori de acces *public* și *privat*. Putem utiliza acești modificatori pentru incapsulare.

Pentru a proiecta și dezvolta aplicații în PHP, într-o manieră orientată pe obiecte, vom folosi cuvântul cheie *class*. Definirea unei clase presupune definirea numelui, proprietățile și metodele acesteia. Numele clasei este case-insensitive și trebuie să respecte regulile PHP-ului pentru identificatori.

O metodă este o funcție definită în interiorul unei clase. Chiar dacă PHP nu impune nicio restricție asupra definirii funcțiilor, cele mai multe metode acționează numai asupra datelor din obiectul în care se află metoda. Numele metodelor care încep cu două simboluri *underscore* (\_) sunt specifice PHP-lui și sunt folosite pentru serializarea obiectelor.

În cadrul unei metode, variabila *\$this* conține o referință la obiectul în care a fost definită metoda. Metodele utilizează această variabilă pentru a accesa proprietățile obiectului curent și pentru a apela alte metode pentru acel obiect. Declararea proprietăților sunt opționale.

Folosind modificatori de acces, putem schimba vizibilitatea proprietăților. Proprietățile care sunt accesibile din afara domeniului de aplicare a obiectului ar trebui să fie declarate publice. Proprietățile unei instanțe care pot fi accesate doar de metodele din aceeași clasă trebuie să fie declarate folosind modificatorul de acces *private*. În cele din urmă, proprietățile declarate ca și *protected* pot fi accesate doar de metodele clasei obiectului și de metodele claselor care moștenesc clasa respectivă.”[16]

## 2.5 Integrarea PHP-ului cu bazele de date

PHP suportă peste 20 de sisteme de baze de date. Bazele de date relaționale, cum ar fi MySQL, PosrgreSQL și Oracle sunt cele mai folosite pentru aplicațiile web dinamice. Există două moduri de a accesa bazele de date cu ajutorul PHP-ului. Prima este utilizarea unei extensii ale bazei de date, iar cealaltă este utilizarea unui driver numit PDO. PDO(PHP Data Objects) poate fi folosit ca un nivel de abstractizare pentru conexiunea dintre programele PHP și diverse baze de date. PDO definește o interfață simplă și consistentă între diverse baze de date.

Dacă utilizăm o extensie specifică unei baze de date, codul este strâns legat de baza de date utilizată. De exemplu, dacă utilizăm extensia MySQL-ului, numele funcțiilor, parametrii și modul de manipulare al erorilor sunt complet diferite de cele ale altor extensii de baze de date. Dacă dorim migrarea de la MySQL către PostgreSQL, atunci codul va trebui modificat aproape în totalitate. Pe de altă parte, PDO, nu accesează funcții specifice extensiilor sistemelor de baze de date, astfel că migrarea de la un sistem de baze de date la altul se face foarte ușor.

Portabilitatea abstractizării librăriei PDO are un dezavantaj: codul este puțin mai lent decât codul care utilizează extensiile specifice bazelor de date.

PHP comunică cu bazele de date relaționale, cum ar fi MySQL și Oracle folosind Structured Query Language (SQL). Sintaxa SQL este împărțită în două părți. Prima, limbajul de manipulare a datelor sau DML(Data Manipulation Language) este utilizată pentru a prelua și modifica datele într-o bază de date existentă. LMD este compact și este format din doar 4 acțiuni: SELECT, INSERT, UPDATE și DELETE. Setul de comenzi SQL utilizat pentru a crea și modifica structurile bazei de date este cunoscut sub numele de Data Definition Language sau DDL.

Extensia PDO definește o interfață pentru accesarea datelor prin intermediul PHP-ului. Fiecare driver al unui sistem de baze de date care implementează interfața PDO poate impune reguli specifice bazelor de date specifice cum ar fi expresiile regulate.

PDO are următoarele caracteristici unice:

- este o extensie nativă C
- are avantajul de a folosi noile caracteristici specifice PHP 5
- folosește buffer-ul de citire a datelor din setul de rezultate
- oferă caracteristici comune bazelor de date
- este încă în măsură să acceseze funcții specifice bazelor de date
- poate utiliza tehnici bazate pe tranzacții
- poate interacționa cu LOBS (Large Objects) în baza de date
- poate implementa cursoare

## Interfața MySQLi Object

Platforma cea mai folosită pentru interacțiunea bazelor de date cu PHP-ul este cea a MySQL-ului. PHP are un număr diferit de interfețe prin care poate comunica cu MySQL-ul, cea mai folosită fiind interfața obiect-orientată MySQLi.

Această interfață orientată pe obiecte este construită în PHP, având un standard de instalare configurabil (extensia MySQLi trebuie activată pentru a putea fi utilizată în PHP). Tot ce trebuie să facem pentru a începe utilizarea acestei extensii este să instanțiem clasa:

```
$db = new mysqli(host, user, password, databaseName);
```

O dată ce am instanțiat clasa în variabila `$db`, putem folosi metodele pentru obiecte, pentru a realiza operații asupra datelor.

```
$db = new mysqli("localhost", "raluca", "1q2w3e9i8u7y", "biblioteca");
$sql = "INSERT INTO carti (autorID, titlu, ISBN, anPublicatie, disponibilitate) VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1)";
if($db->query($sql)) {
    echo "Datele cartii au fost introduse.";
}
else {
    echo "Inserarea datelor nu s-a putut face. Încercați mai târziu." ;
}
$db->close();
```

Prima dată am instanțiat clasa MySQLi în variabila `$db`. Apoi am construit o comandă SQL și am memorat-o în variabila `$sql`. Am apelat metoda de interogare a clasei și în același timp am testat valoarea returnată pentru a determina dacă interogarea s-a realizat cu succes (TRUE). La sfârșit am apelat metoda `close` pentru clasă pentru a distrugere clasa din memorie.

## MongoDB

„Bazele de date de tip NoSQL sunt în creștere de popularitate, deoarece lucrează altfel decât structura tipică de SQL. Bazele de date de tip NoSQL sunt de asemenea foarte populare pentru aplicațiile pentru terminalele mobile.

```
$mongo = new Mongo();
$db = $mongo->library;
$authors = $db->authors;
$author = array('authorid' => 1, 'name' => "J.R.R. Tolkien");
$authors->insert($author);
$author = array('authorid' => 2, 'name' => "Alex Haley");
$authors->insert($author);
$author = array('authorid' => 3, 'name' => "Tom Clancy");
$authors->save($author);
```

Prima linie de cod din exemplul de mai sus descrie crearea unei noi conexiuni la motorul bazei de date Mongo și a unei interfețe obiect. Următoarea linie arată conectarea la librăria colecției, și dacă nu există colecția, atunci Mongo o va crea. Vom crea o interfață obiect cu conexiunea la baza de date `$db` și vom crea un document în care vom stoca datele despre autor. În următorii pași putem vedea cum sunt adăugate datele în documentul *autori* în două moduri diferite.

Primele două instrucțiuni folosesc metoda `insert()`, iar a treia instrucțiune folosește metoda `save()`. Singura diferență între aceste două metode este că metoda `save()` va actualiza o valoare în cazul în care aceasta este deja în documentul respectiv și are o cheie existentă `_id`.”[17]

# **Capitolul 3. Tehnologii pentru User Interface**

## **3.1 HTML/XHTML**

HyperText Markup Language (HTML) este un limbaj de marcare utilizat pentru crearea paginilor web ce pot fi afişate într-un browser (sau navigator). HTML este un format text proiectat pentru a putea fi citit și editat de utilizatori folosind un editor de text simplu. HTML se poate genera direct, utilizând tehnologii de codare din partea serverului cum ar fi PHP, JSP sau ASP [18]. Atunci când este realizat un web-site, cel mai important lucru este crearea legăturilor.

### **3.1.1 Introducere**

La începutul anilor 90 început dezvoltarea unui nou limbaj de programare/marcare. Limbajul a fost creat pentru a oferi o modalitate pentru utilizatori de a afișa diverse informații în browserele web. Ultima versiune majoră pentru HTML a apărut în februarie 2011. În ianuarie 2000 s-au adăugat unele reguli mai stricte pentru HTML 4.01, astfel apărând XHTML (Extensible Hypertext Markup Language).

HTML și XHTML definesc un set de reguli pentru marcarea documentelor, dar și pentru modul structurării informațiilor.

### **3.1.2 Structura unei pagini HTML**

Când browser-ul primește conținut, acesta încearcă să îl proceseze pentru a verifica dacă este un document HTML. Chiar dacă unele elemente sau tag-uri lipsesc, sau dacă nu este bine structurat, browser-ul poate randa conținutul. Chiar dacă un document este valid (respectă regulile HTML/XHTML), el trebuie să conțină un număr de elemente structurate cu un conținut adecvat. Un document, pentru a fi valid, trebuie să conțină următoarele informații:

- declarația tipului de document
- tag-ul `<html>`
- în tag-ul `<html>`, tag-ul `<head>`
- în tag-ul `<head>` trebuie să existe elementul `<title>`, adrese URL necesare, script-uri folosite și elementele meta
- după închiderea tag-ului `</head>`, se deschide tag-ul `<body>`, care va conține toate informațiile vizibile utilizatorilor
- în cadrul tag-ului `<body>` putem să definim elemente bloc, putem să structurăm informația după bunul plac.

### **3.1.3 Tag-uri, elemente și attribute**

HTML/XHTML au definite o serie de elemente, fiecare dintre ele încadrându-se într-un anumit domeniu semantic, având un nume derivat sau împrumutat din limba engleză. Elementele definesc structura documentului și pune bazele pentru prezentarea și manipularea acestuia.

Fiecare element de referință dintr-un document este încadrat între două tag-uri - token-uri între paranteze unghiuilare (< >), conținând numele elementului folosit. Tag-urile de deschidere încep întotdeauna cu simbolul „<”, urmat de numele elementului, de referințele atributelor sau valori asociate elementului.

Atributele sunt folosite pentru a furniza informații suplimentare despre elementul pentru care sunt definite. Toate attributele sunt alcătuite din două părți: un nume și o valoare.

```
<a href="http://www.Google.com" target="_blank">
```

- numele este proprietate elementului pe care dorim să o setăm. În exemplul de mai sus, *href* este numele proprietății pe care am definit-o.
- valoarea este setată pentru atributul definit. În exemplul de mai sus, adresa URL este valoarea pentru atributul *href*.

Atributele cele mai utilizate în paginile web sunt cele din categoria *Core*: class, id, title, style. Atributul *id* este folosit pentru a asocia un identificator unic unui element într-o pagină web. Putem asocia acestui *id* diferite evenimente de tip JavaScript sau proprietăți de tip CSS. Sintaxa pentru a asocia un *id* este *id = "string"*.

Atributul *class* este folosit pentru a arăta cărei clase de elemente aparține. Dacă avem mai multe elemente de tip *<p>* care vor primi aceeași proprietăți CSS, atunci putem să atribuim o clasă pentru a fi mai ușor de stilizat. Pentru a ataşa o clasă unui element vom folosi următoarea sintaxă: *class = "numeClasa"*. Un element HTML poate apartine mai multor clase, numele lor fiind despărțite prin spațiu la definirea lor.

Atributul *title* este folosit pentru a denumi un element HTML. Sintaxa este: *title = "string"*. Comportamentul acestui atribut este diferit în funcție de elementul căruia îi este atribuit. De cele mai multe ori titlul apare sub forma unui tooltip sau apare atunci când se încarcă elementul. Nu toate elementele necesită atributul *title*.

Atributul *style* permite specificarea proprietăților *inline* de CSS. Sintaxa este

*<p style="font-family:Arial, color:black">Text</p>*. Este de evitat acest atribut, deoarece este marcat ca și *deprecated* în XHTML 1.0, acest lucru însemnând că nu va mai putea fi folosit în versiunile ulterioare ale HTML-ului și XHTML-ului.[19]

Cele mai utilizate categorii de *tag-uri* în HTML sunt:

**Listele** - această categorie conține 3 elemente: liste neordonate (în care elementele vor apărea cu bullet-uri), liste ordonate (elementele vor apărea având numere sau litere înainte) și liste cu definiții (permisă specificarea termenului, iar apoi definiția acestuia)

Dacă dorim afișarea unei liste neordonate, vom folosi tag-ul *<ul>*. Fiecare informație din listă va fi încadrată între tag-urile *<li>, </li>*. Pentru liste ordonate vom folosi tag-ul *<ol>*, iar pentru liste definite vom folosi *<dl>*. Listele definite vor avea ca elemente tag-urile *<dt>* și *<dd>*. Pentru a defini termenul vom folosi tag-ul *<dt>*, iar pentru a scrie definiția termenului vom folosi tag-ul *<dd>*.[20]

**Elemente grupate** - Pentru a structura mai ușor informația sau pentru a crea secțiuni în document putem să utilizăm tag-urile *<div>* și *<span>*. Aceste elemente nu afectează modul în care o pagină

web este afișată, dar sunt folosite pentru a ataşa proprietăți cu ajutorul CSS-ului. Tag-ul `<div>` este utilizat pentru a grupa blocuri de alte elemente, secțiuni, iar tag-ul `<span>` este utilizat pentru a grupa elemente *inline* (în linie). `<span>` se va folosi pentru a grupa elemente în interiorul tag-ului `<p>`.

**Paragrafele** - Sunt utilizate pentru a marca un text mai lung. Acest element are câteva proprietăți predefinite de CSS pentru identare.

**Link-uri** - Un hyperlink este un cuvânt, sau un grup de cuvinte, sau o imagine, care atunci când este acționat evenimentul de *click* va trimite utilizatorul către altă pagină sau document sau poate downloada fișiere. Când cursorul este peste un link, atunci el va avea forma unei „mâini”. Atributul `href` este foarte important pentru acest element, deoarece el va indica destinația utilizatorului la acțiunea de click. Sintaxa este: `<a href = "http://google.com" target = "_blank">Click aici </a>`. Un alt atribut folosit împreună cu acest element este `target`. Acest atribut specifică unde să deschidă documentul linkuit.

**Headings** - Motoarele de căutare folosesc aceste tag-uri pentru a indexa structura și conținutul unei pagini web. H1 este cel mai mare heading, iar H6 este cel mai mic.

**Imagini** - În documente HTML imaginile sunt introduse cu ajutorul tag-ului `<img>`. Pentru a fi afișată o imagine într-o pagină web trebuie să completăm câmpul `src`(source - sursă). Acest atribut va lua valoarea URL-ului unde se află imaginea pe care vrem să o afișăm.

**Tabele** - Un tabel este format din rânduri (`<tr>`), iar fiecare când este format din celule (`<td>`). Un tabel se definește cu ajutorul tag-ului `<table>`.

**Elemente de formular** - Formularele HTML trimit datele către un server. Un formular de acest tip poate să conțină câmpuri pentru text, checkbox-uri, butoane radio, buton pentru submit.

## 3.2 CSS

### 3.2.1 Introducere

CSS - Cascading Style Sheets - este un standard folosit pentru a stiliza și formata elementele dintr-o pagină web. Acest lucru se întâmplă deoarece CSS se află într-o strânsă legătură cu DOM-ul (Document Object Model). Folosind CSS putem modela ușor și rapid orice element. Stilizarea unui element se poate face în interiorul paginii web adăugând atributul `style`, dar se poate face și prin gruparea tuturor stilurilor într-un singur fișier cu extensia `.css`. Gruparea acestor informații reduce cantitatea de cod care poate fi duplicate pe pagini, poate fi mai ușor de menținut. Adăugarea unui fișier cu stiluri într-o pagină web se realizează:

```
<link rel = "stylesheet" type = "text/css" href = "style.css">
```

Pentru a identifica elementele din pagina web, ne folosim de id-uri, clase și tipurile elementelor. Atunci când utilizăm `id-uri` folosim sintaxa: `#id { font-style:italic; }`, iar în cazul claselor folosim `.clasa { font-style:italic; }`. Pentru identificarea id-urilor/claselor utilizăm simbolurile „#”, respectiv „.”.

Regula CSS este o instrucțiune, sau un grup de instrucțiuni care transmite browser-ului cum să randeze/stilizeze un anumit element din pagină. Fiecare instrucțiune în CSS începe cu *selectorul*,

acesta fiind elementul asupra căruia se va aplica regula CSS. Toate proprietățile care trebuie asignate elementului se vor afla între simbolurile {} care vor fi scrise după selector. Instrucțiunea se va termina cu simbolul „;”. Acoladele sunt utilizate pentru a separa multiple instrucțiuni de tip CSS.

### 3.2.2 Selectori în CSS

Selectorul *tip* - acest selector acționează asupra tuturor elementelor de același tip. De exemplu dacă avem *p* {color:black}, atunci toate elementele de tip p (paragraf) vor avea culoarea fontului neagră.

Selectorul de tip *descendent* - acest selector permite aplicarea stilurilor asupra unor elemente conținute de alte elemente. De exemplu, următoarea linie de cod va afecta toate link-urile conținute de elemente de tip paragraf: *p a* {color:red}.

Selectorul de tip *copil* - acest tip de selector seamănă cu selectorul de mai sus, de tip *descendent*, dar este mult mai strict atunci când se aplică regulile. Va selecta doar elementele care sunt copiii direcții ai unor elemente.

Selectorul de tip „*înrudit*” - acest selector se aplică pentru toate elementele care se află pe același nivel cu cel selectat și care sunt în imediata apropiere. Ex: *p + a* {color:red}. Aceste selector va colora toate paragrafele în roșu, doar dacă sunt urmate în cod de elemente de tip link.

Selectorul *id* - id-ul poate fi folosit o singură dată într-un document web. În CSS ne putem referi direct la elementul dorit, dacă acesta are setat un id.

Selectorul *class* - mai multe elemente dintr-o pagină web pot avea aceeași clasă, iar pentru a seta proprietăți pentru acest grup de elemente folosim selectorul de tip clasă.

Selectorul *atribut* - multe elemente din HTML suportă attribute, astfel că putem să selectăm elemente pe baza acestor proprietăți. Ex.: [type = “submit”] {width:100px;}

Selectorul *universal* - pentru a stabili un anumit stil pentru toate elementele din pagină, vom folosi acest selector - \*. [21]

### 3.2.3 Proprietăți CSS

Stiluri pentru background

- *background-color* - definește culoarea fundalului unui element.
- *background-image* - specifică imaginea care va fi utilizată pe post de fundal.
- *background-repeat* - în mod normal, imaginea de pe fundal este repetată atât orizontal, cât și vertical. Dacă dorim ca o imagine să fie repetată doar orizontal, sau doar vertical, vom folosi aceasă proprietate.
- *background-position* - vom folosi această proprietate dacă dorim ca o imagine de fundal să aibă o poziție anume în pagina web.

## Fonturi

- font-family - fontul unui text poate fi stabilit prin aceasta proprietate. Putem alege un anumit font, dar de asemenea, putem alege o familie de fonturi.
- font-style - această proprietate pentru a transforma textul în text *italic*.
- font-size - dimensiunea unui font se stabilă prin această proprietate. Mărimea fontului poate fi relativă sau absolută. Mărimea absolută stabilăște o dimensiune fixă, iar mărimea relativă stabilăște dimensiunea fontului în funcție de elementele vecine și permite utilizatorului să o schimbe în browser. Dacă nu este specificată această proprietate, ea va lua o valoare predefinită - 16px.
- font-weight - se folosește pentru a îngroși textul.

## Stiluri pentru text

- color - este folosită pentru a seta culoarea unui text. Această proprietate poate primi una din următoarele tipuri de valori: o valoarea hexazecimală (#fff000), o valoare de tip RGB (*rgb(255,0,0)*) sau numele unei culori (*red*).
- text-decoration - se folosește pentru a elimina sau adăuga elemente decorative într-un text, cum ar fi linia pentru subliniat care apare la link-uri.
- spacing
- text-align - este folosită pentru a alinia orizontal textul. Poate primi următoarele valori: *center, left, right, justify*.
- text-transformation - se va folosi atunci când dorim să specificăm dacă un text conține numai majuscule sau minuscule.
- text-indent - este folosită pentru a indenta prima linie dintr-un text.

## Pozitionarea elementelor

- absolute - un element pozitionat absolut va fi pozitionat relativ față de primul părinte. Părintele nu trebuie să aibă setată proprietatea de pozitionare statică.
- relative - pozitionarea relativă a unui element.
- fixed - un element care va avea această proprietate va avea o pozitionare fixă în pagina web.

## Pseudo-clase

- :first-letter - se va aplica un stil pentru prima literă a textului conținut de element.
- :first-child - se va aplica un anumit stil pentru primul copil al elementului.
- :link - se folosește pentru a adăuga un anumit stil pentru link-urile care nu au fost vizitate.
- :visited - se folosește pentru a adăuga un anumit stil pentru link-urile care au fost vizitate.

- :hover - se folosește pentru a adăuga un anumit stil pentru elemente atunci când mouse-ul este deasupra lor.
- :active - se folosește pentru a adăuga un anumit stil pentru link-uri, atunci când sunt accesate.
- :focus

## Stiluri pentru liste

- list-style-type - dacă dorim să avem alt simbol decât *bullet-ul* elementelor dintr-o listă neordonată, vom folosi această proprietate.
- list-style-image - putem seta ca și *bullet* o imagine customizată.

## Pseudoelemente

- :before - poate fi folosit pentru a insera conținut înainte de elementul selectat.
- :after - poate fi folosit pentru a insera conținut după elementul selectat.

## Margini și identare

- margin - setează distanța elementului față de elementele vecine. Această proprietate nu are culoare, ea fiind transparentă. Valoarea dată va fi în pixeli.
- padding - indentează textul față de chenarul imaginari al elementului. Această proprietate este afectată de culoarea background-ului, dacă aceasta este setată.

## Chenar

- border - putem seta dimensiunea unui chenar, tipul și culoarea.
- border-color - se setează doar culoarea unui chenar.
- border-style - se setează stilul chenarului.
- border-width - se setează dimensiunea chenarului.

## Dimensiunile unui element

- width - stabilește lățimea unui element.
- height - stabilește înălțimea unui element.
- min-width - setează o lățime minimă a unui element
- min-height - setează o înălțime minimă a unui element
- max-width - setează o lățime maximă a unui element
- max-height - setează o înălțime maximă a unui element

### 3.3 JavaScript

În prezent, există mai multe limbaje de programare, dar JavaScript este de departe cel mai comun limbaj de programare utilizat pentru crearea paginilor web. JavaScript aduce o funcționalitate dinamică a site-urilor web.

JavaScript a apărut pentru prima dată în browser-ul web Netscape Navigator în 1995. Deseori, JavaScript este confundat cu limbajul Java. Numirea a fost doar un truc de marketing pentru a aduce un beneficiu noului limbaj de programare, beneficiind de popularitatea pe care limbajul Java o avea deja.

JavaScript a crescut și mai mult în popularitate atunci când elementele HTML au căpătat definiție mai formală, fiind structurate în DOM(Document Object Model). Datorită faptului că JavaScript și PHP au o sintaxă ce se bazează pe cea a limbajului C, ele seamănă foarte mult. JavaScript oferă dezvoltatorilor web un limbaj de programare care poate fi utilizat pentru a executa sarcini precum:

- citirea elementelor din documente și scrierea unor noi elemente în pagina web;
- manipularea textului;
- efectuarea unor calcule matematice;
- atașarea unor evenimente elementelor din pagina web;
- determinarea dimensiunilor ecranului utilizatorului, determinarea versiunii sau rezoluția browser-ului;
- efectuarea unor acțiuni bazate pe condiții, cum ar fi alertarea utilizatorului în cazul în care acesta introduce informații greșite într-un formular.

În limbajele de programare orientate pe obiecte, obiectele din viața reală sunt reprezentate/modelate, folosind un set de obiecte care formează un model obiect. Fiecare metodă descrie o acțiune care poate fi facută asupra unui obiect.

JavaScript poate fi încorporat într-o pagină sau plasat într-un fișier script extern. Script-urile se vor adăuga în interiorul tag-urilor `<script>`, `</script>`. Fișierele de tip JavaScript se vor adăuga astfel:

```
<script type="text/javascript" src="http://someserver.com/script.js"> </script>
```

Variabilele în JavaScript trebuie să respecte următoarele reguli:

- numele variabilelor pot conține doar literele a-z, A-Z, simbolul „\$” și simbolul „\_”;
- primul caracter al numelui variabilei poate fi doar literă sau simbolul \$;
- numele sunt *case-sensitive*;
- nu există o limită stabilită pentru lungimea unei variabile.

Când o variabilă este declarată într-o funcție, ea poate fi accesată doar în interiorul acelei funcții. După apelarea funcției, variabila se distrugă. Aceste variabile se numesc locale. Dacă o variabilă locală este declarată folosind cuvântul cheie `var` în interiorul funcției, ea va ocupa memorie atât

timp cât funcția rulează. Dacă o variabilă este declarată în afara unei funcții, ea poate fi accesată de orice metodă/funcție din pagina web. Durata de viață a acestor variabile începe atunci când sunt declarate și se termină atunci când pagina este închisă. Variabilele locale ocupă mai puțină memorie și resurse decât cele globale, deoarece au nevoie de memorie doar în momentul în care funcția se execută.

Șirurile de caractere pot exista în JavaScript dacă sunt cuprinse între ghilimele simple sau duble. Vectorii sunt similari cu cei din PHP, astfel că ei pot conține atât valori, cât și alți vectori.

Pentru a accesa diferite obiecte din DOM, JavaScript separă obiectele, proprietățile și metodele folosind simbolul „.”.

La fel ca și în cazul PHP-ului, JavaScript oferă acces la funcții și la obiecte. Utilizarea și sintaxa sunt, de asemenea, destul de asemănătoare cu cele din PHP. Deși avem acces la zecile de funcții predefinite, ne putem crea și funcțiile proprii. [22]

O funcție se definește astfel:

```
function nume_functie([parametru [...]])  
{ instrucțiuni}
```

Definirea unei funcții începe cu cuvântul *function*, urmat de numele acesteia pe care programatorul îl atribuie. Numele trebuie să înceapă cu o literă sau cu simbolul „\_”. Parantezele sunt obligatorii chiar dacă nu există parametrii. Dacă avem mai mulți parametrii, aceștia vor fi separați prin virgulă. În JavaScript există o convenție privind numele funcțiilor: dacă numele este format din mai multe cuvinte, fiecare literă de început al cuvântului va fi o majusculă, exceptie făcând prima literă a numelui funcției. Funcțiile sunt folosite pentru a efectua anumite operații asupra datelor.

### 3.4 jQuery Mobile

În ultimul timp partea web s-a mutat din ce în ce mai mult spre terminalele mobile, astfel că programatorii au trebuit să se adapteze. Deși web-ul nu s-a schimbat, atunci când un web-site pentru terminalele mobile este dezvoltat, trebuie să fim atenți deoarece există foarte multe tipuri de telefoane/tablete pe care trebuie să testăm aplicația. Cea mai mare diferență este dimensiunea ecranului.

Termenul de *aplicatie web* are mai multe sinonime sau concepte, cum ar fi: aplicație mobilă, widget-uri, aplicație hibridă, aplicație HTML5, etc. În particular, o aplicație web diferă de tipicele website-uri pentru terminalele mobile. Deși aplicația web are o implementare mult mai complexă, interfața cu utilizatorul este mai elaborată, ea este creată folosind tehnologii web precum HTML, CSS, JavaScript, AJAX. Aplicațiile web folosesc de asemenea și caracteristici HTML5, precum accesul offline sau geo-locatăția.

O aplicație web poate fi implementată în mai multe moduri:

- accesând un browser;
- instalată sub forma unei aplicații web full-screen;
- instalată prin intermediul unui pachet oficial pus la dispoziție de diversi dezvoltatori, uneori fiind numite widget-uri;

- ca o aplicație web integrată într-o aplicație nativă, acest proces fiind cunoscut sub numele de hibrid.

O aplicație web generează noi provocări pentru web designeri, dar și pentru programatorii web, deoarece aceasta necesită încărcarea ecranelor în loc de încărcarea paginilor, menținerea navegației cu două sensuri între ecrane, dar și crearea unor butoane special pentru dispozitivele cu ecran tactil.

A apărut necesitatea de a crea aplicații web având o complexitate mai mare decât a unui website. Acesta a fost motivul pentru care a apărut jQuery Mobile: pentru a ajuta dezvoltatorii, dar și designerii la crearea unor experiențe multiplatforme și ușor de customizat. Acest framework a primit sprijinul oficial din partea unor companii importante din acest domeniu:

- Adobe
- Mozilla Corporation
- HP Palm
- BlackBerry/RIM
- Nokia
- DeviceAtlas și dotMobi

Pentru a înțelege acest framework, este să aflăm câteva informații despre el:

- jQuery Mobile nu este o alternativă jQuery pentru browserele mobile;
- pentru a folosi jQuery Mobile, programatorul trebuie să includă framework-ul tipic jQuery;
- jQuery Mobile nu este un SDK pentru aplicații mobile;
- poți crea o aplicație pentru terminalele mobile utilizând jQuery Mobile, dar programatorul trebuie să mai instaleze anumite pachete pentru a compila aplicația precum o aplicație nativă;
- jQuery Mobile nu este un framework JavaScript;
- în afară de câteva cazuri izolate, rularea jQuery Mobile nu necesită cod JavaScript;
- jQuery Mobile nu este o soluție pentru toate tipurile de aplicații mobile.

jQuery Mobile este un framework care oferă experiența unei aplicații mobile pentru dispozitive mobile, tablete și este folosit mai ales pentru crearea interfețelor tactile, cross-platform, utilizând HTML5. Platforma folosește core-ul librăriei JavaScript, jQuery, folosește CSS3 pentru stilizarea elementelor.

jQuery Mobile a apărut în August 2010 ca și un framework modern, inclusiv multe pattern-uri. Este o soluție fiabilă pentru dezvoltarea aplicațiilor multi-platform. Cele mai importante caracteristici ale acestui framework sunt:

- este cross-platform, cross-device și cross-browser

- este optimizat UI(User Interface) pentru dispozitivele cu ecran tactil
- design-ul poate fi customizat, în funcție de cerințele utilizatorului
- folosește semantică ușor de înțeles precum HTML5
- pentru a încărca un conținut dinamic se va folosi AJAX
- este construit pe baza core-ului jQuery
- este îmbunătățit progresiv.[23]

Îmbunătățirea progresivă este o tehnică simplă, dar foarte puternică, folosită în web design și definește unele straturi de compatibilitate, permitând astfel, ca orice utilizator să acceseze conținutul de bază, serviciile și funcționarea unei aplicații web.

O aplicație dezvoltată cu ajutorul framework-ului jQuery Mobile funcționează pe aproape orice browser modern. Totuși, se recomandă testarea unei aplicații mobile web pe diferite platforme, utilizând:

- telefoane mobile, tablete
- emulatoare și simulatoare pentru terminalele mobile

Emulatorul este un software care traduce codul compilat de la o arhitectură originală la platforma pe care rulează. Pentru partea de terminale mobile, emulatorul este o aplicație desktop care emulează hardware-ul dispozitivului mobil și sistemul de operare, permitându-ne să testăm sau să depanăm aplicația.

jQuery Mobile are un set mare de componente UI (User Interface), pe care le putem utiliza în dezvoltarea aplicației noastre. Totuși, putem adăuga/crea propriile noastre componente folosind HTML și CSS. Putem împărți componentele acestui framework în următoarele grupuri:

- componente pentru toolbar
- componente pentru formatare
- butoane
- liste
- componente pentru formulare

**Toolbar** - este o zonă optională într-o aplicație web și definește antetul(header) și/sau subsolul (footer). Antetul este aproape mereu existent într-o aplicație web mobilă. În antet regăsim de obicei titlul și/sau butoane de navigație. Se definește astfel:

```
<div data-role="header">
    <h1>Page's title</h1>
</div>
```

Subsolul(footer) este o zonă similară localizată în josul aplicației web. Poate include informații despre copyright sau o serie de butoane. Se definește astfel:

```
<div data-role="footer"> </div>
```

Fiecare toolbar poate fi poziționat în patru moduri: modul inline, modul standard fix, modul fix full-screen și modul fix adevărat. Modul *inline* este cel predefinit pentru fiecare toolbar. Acest lucru înseamnă că se va mișca o dată cu ecranul principal al aplicației. Dacă vom avea conținutul unei pagini mai mare decât înălțimea ecranului, atunci footer-ul va rămâne ascuns în mod implicit și va apărea după defilarea paginii în jos, în timp ce header-ul/antetul va fi vizibil cât timp sensul de parcursere al paginii este în sus.

**Formatarea** - un lucru esențial pe care trebuie să-l știm este că orice cod HTML trebuie încadrat între tag-urile `<div data-role="content">...</div>`. Fiecare temă predefinită are incluse stiluri cu margini, fonturi, culori custom. jQuery Mobile oferă unele şablonane pentru a defini conținutul care urmează să fie afișat sub formă de coloane, numite *layout grids*. Aceste grid-uri acționează precum un tabel. Această metodă de layout folosește clase CSS pentru a defini zonele grid-urilor și coloane. Numărul grid-urilor poate fi de la 2 la 5. Fiecare coloană este invizibilă, folosește lățimea 100% și nu are padding-uri sau margini.

**Butoane** - putem folosi orice element pentru a face legături între pagini sau pentru a lega conținutul de link-uri externe. Totuși, un element predefinit nu este ușor de randat pentru dispozitivele tactile. Un buton este o componentă UI(User Interface), este o zonă cu text sau o icoană pe care se poate atașa acțiunea de *click*. Un buton poate fi creat în moduri diferite:

- folosind un element de tip buton;
- folosind un element de tip *input* care este randat ca și *buton*, definind `type="button"`, `type="submit"`, `type="reset"` sau `type="image"`;
- folosind `data-role="button"` pentru orice element.

Un buton creat cu jQuery Mobile este randat cu o etichetă centrală, colțuri rotunde și umbre, depinzând de compatibilitatea browser-ului cu CSS3. Putem renunța la umbre sau la colțuri rotunde setând valori de tip Boolean pentru aceste proprietăți:

```
<a href="#" data-role="button" data-shadow="false" data-corners="false">Help</a>
```

În mod normal, fiecare pictogramă este randată la stânga textului butonului. Putem schimba poziția butonului folosind atributul `data-iconpos`, care suportă valorile *right*, *left*.

**Liste** - În jQuery Mobile, o listă este un element HTML - *ol* sau *ul*. Rolul este definit astfel `data-role="listview"`. O listă în jQuery Mobile este randată pentru dispozitivele cu ecran tactil, iar fiecare element al listei va avea aceeași lățime ca și ecranul. Totuși, uneori apare necesitatea existenței unor liste combinate cu alte elemente HTML. În aceste cazuri vom folosi listele *inline*. Pentru definirea lor vom folosi următoarea sintaxă:

```
<ol data-role="listview" data-inset="true">
    <!-- item rows -->
</ol>
```

Adăugând interacțiuni tactile pentru listele create putem crește complexitatea aplicației. Dacă un element al unei liste conține un element, acesta se va transforma într-un rând interactiv pentru utilizator. Utilizarea listelor interactive este una dintre cele mai bune metode pentru a crea legăturii într-o pagină, deoarece acestea sunt optimizate pentru ecranul tactil.

**Componentele formularelor** - jQuery Mobile suportă formulare web standard, folosind manipulare AJAX. Formularele sunt compatibile cu toate dispozitivele mobile, inclusiv cu cele cu ecran tactil. Datele unui formular vor fi trimise către server după ce s-a apăsat un buton căruia i-a fost asociată o acțiune de *submit*. Sintaxa pentru a defini un formular este:

```
<form action="send.php" action="get">...</form> [24]
```

### 3.5 Coduri QR

Un QR cod este un simbol bidimensional. Au fost inventate în 1994, în Japonia, de către Denso, una din cele mai importante companii din grupul Toyota. Inițial au fost folosite pentru a urmări vasele navale. Un cod QR acționează ca un link încorporat în lumea reală, integrându-l cu lumea virtuală. În prezent ele sunt utilizate pentru campaniile de publicitate, conectându-le astfel la website-urile companiilor, la paginile de concurs sau la meniuri on-line.

Codurile sunt din ce în ce mai populare. Le putem găsi pe etichetele produselor, pe panouri publicitare, pe clădiri, invitând turiștii sau trecătorii să le scaneze și să descopere informații codificate. De mici dimensiuni, codul poate fi ascuns sau integrat atractiv în ziare, reviste, magazine, etc.

Codurile QR pot conține orice caracter alfanumeric, dar și link-uri către anumite website-uri, astfel ajutând utilizatorul să afle informații despre un anumit loc sau obiect (o practică cunoscută sub numele de „etichetare mobilă”).

Datele pot fi traduse într-un cod QR prin orice generator de coduri QR, dintre care multe sunt disponibile gratuit online. Utilizatorii introduc datele pe care le doresc a fi traduse, generatorul produce un cod, care apoi poate fi afișat pe cale electronică sau în format tipărit.

Software-urile de decodare folosesc camera telefoanelor mobile interpretează codurile, care reprezintă un set de informații compactate. După ce software-ul este deschis, utilizatorul îndreaptă camera către cod și îl scanează. Software-ul interpretează codul, iar telefonul mobil va afișa un text sau va cere permisiunea de a lansa browser-ul pentru a deschide o pagină web.

Simbolurile bidimensionale conțin, în general, mai multă cantitate de date în comparație cu simbolurile liniare (aprox. de 100 de ori mai multă informație) și prin urmare, necesită mult mai mult timp și procesul de prelucrare este mult mai complex.

# Capitolul 4. Proiectarea aplicației

## 4.1 Prezentare generală a interfeței

În cele ce urmează voi prezenta toate ecranele aplicației împreună cu funcționalitățile și specificațiile fiecărui. Codul aplicației se poate găsi în Anexa 2.

### 4.1.1 Secțiunile Authentication, Registration, Reset Password

Primul ecran al aplicației este secțiunea de autentificare, având două câmpuri pentru introducerea username-ului și a parolei, după cum se vede în figura 4.1. Dacă logarea se face cu succes, ecranul de logare va dispărea, în locul lui apărând ecranul principal al aplicației (Fig.4.2). Pentru realizarea operației de logare, în cazul MySQL am efectuat o interogare în baza de date de tip *SELECT...WHERE...AND*, printr-o funcție având ca parametrii username-ul și parola. Rezultatul returnat de această funcție este *id-ul* care va fi stocat în baza de date locală. Realizez stocarea *id-ului* utilizatorului pentru a preîntâmpina autentificările repetitive atunci când utilizatorul închide și redeschide aplicația sau inchide instanța aplicației din memoria virtuală. Procesul este asemănător pentru MongoDB, diferă doar instrucțiunea de selectare în baza de date și formatul *id-ului* returnat.

Tot în acest ecran, în caz că utilizatorul nu are cont, poate apăsa butonul „+” pentru a se înregistra. Astfel, utilizatorul trebuie să completeze un formular cu datele personale, să își aleagă un username și o parolă, pe care să le folosească pentru a beneficia de aplicație. Dacă utilizatorul completează întregul formular corect, datele sunt trimise către server, acesta procesează informațiile și trimit un mesaj de confirmare a înregistrării contului. Pentru MySQL, datele sunt trimise către serviciul „registerUser” sub formă de parametrii, urmând să fie inserate în tabela „users” prin operația de *INSERT*. Dacă operația nu se realizează cu succes, utilizatorul va primi o notificare. Pentru realizarea acestei operații în cazul bazei de date Mongo, am folosit funcția *insert*, având ca parametru un obiect care conține datele introduse de utilizator.

De asemenea, dacă utilizatorul și-a uitat parola, poate apăsa butonul de „Forgot your password”. Prin introducerea adresei de e-mail, utilizatorul poate solicita schimbarea parolei. Dacă adresa nu este validă (nu există niciun user cu adresa de e-mail respectivă), utilizatorul aplicației este avertizat că adresa nu există în baza de date. Altfel, parola este resetată și este scrisă într-un fișier text în folder-ul aplicației. La nivel de baze de date și PHP, această acțiune se realizează apelând o funcție *resetPassword* care are ca parametru adresa de e-mail a utilizatorului. Prima operație care se efectuează este *SELECT...WHERE*, aplicând ca și filtru de căutare e-mailul utilizatorului. După ce intrarea se găsește în baza de date, se efectuează o operație de *UPDATE* cu o nouă parolă. În cazul bazelor de date Mongo am folosit funcția *find*, având ca parametru valoarea e-mailului. După această interogare folosesc funcția *update*, având ca parametri *id-ul* utilizatorului și noua parolă.

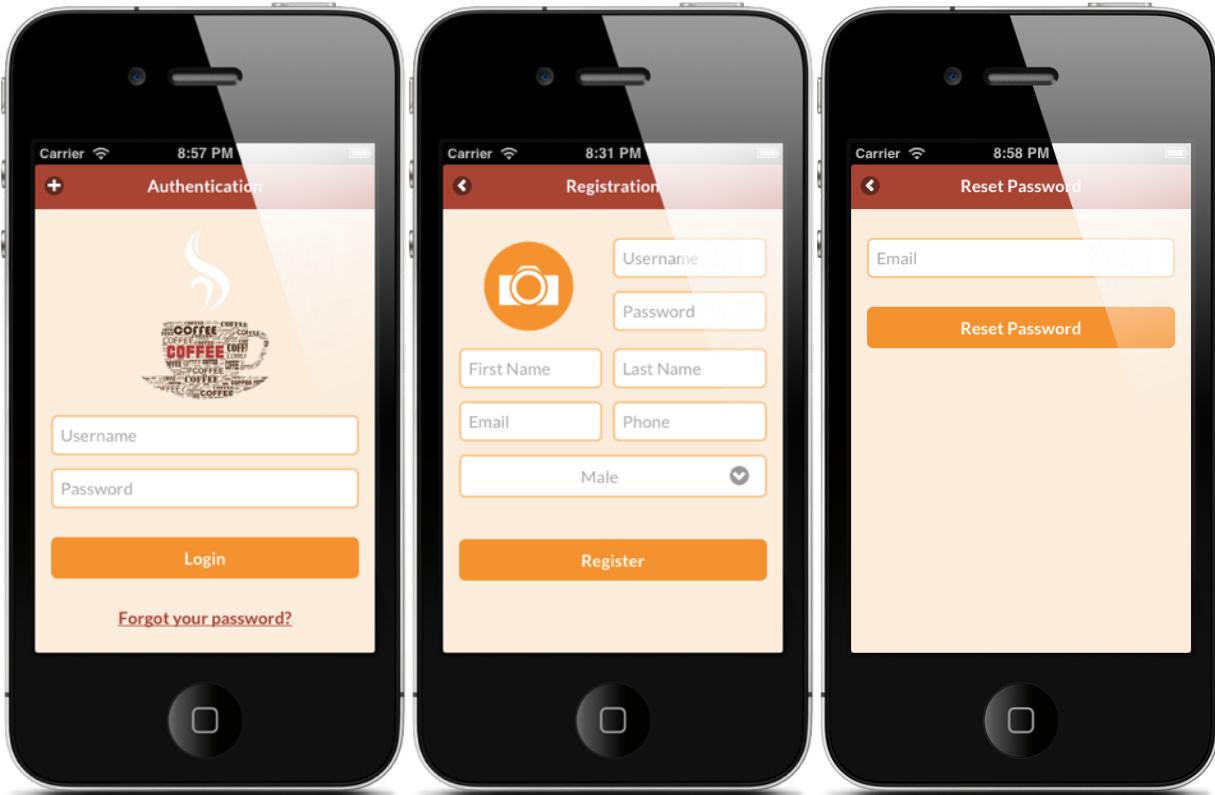


Fig. 4.1 Secțiunile *Authentication/Registration/Reset Password*

#### 4.1.2 Secțiunea Coffee Shop

După ce utilizatorul s-a autentificat, se va încărca ecranul principal (fig. 4.2), care conține 3 butoane: „Menu”, „Locations”, „Special Offers” și 5 butoane care se regăsesc în josul ecranului: „Basket”, „Favourite”, „Scan”, „Activity” și „Friends”. De asemenea, în partea dreapta-sus a aplicației există un buton pentru a accesa detaliile contului.

Cele 3 butoane din centrul ecranului sunt utilizate pentru vizualizarea:

- produselor din meniu
- locațiilor
- ofertelor speciale

Pentru afișarea ofertelor speciale, în MySQL, realizez o interogare în tabela „offer”, de tip *SELECT \**. În MongoDB, utilizez funcția *find* pentru a găsi ofertele speciale.

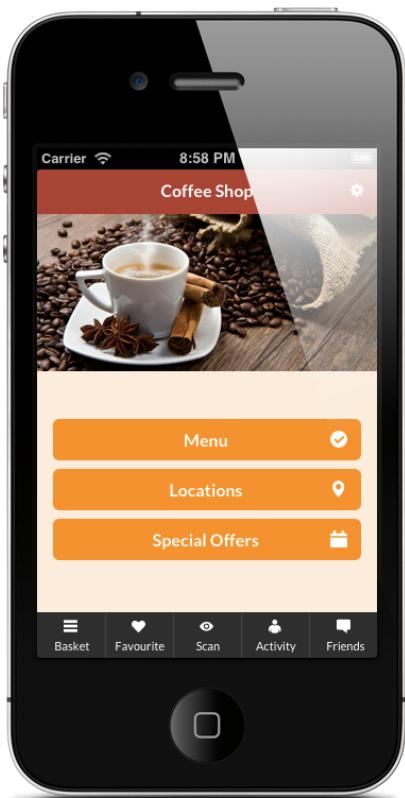


Fig. 4.2 Secțiunea *Coffee Shop*

#### 4.1.3 Secțiunea Settings

Această secțiune (fig. 4.3) cuprinde 2 tab-uri: „Profile” și „Vouchers”.

Tab-ul „Profile” prezintă datele introduse de utilizator în procesul de înregistrare a contului, dar, în același timp, permite modificarea acestora. Tot în acest ecran putem observa și un buton de „Logout”. Dacă utilizatorul dorește modificarea datelor personale, va schimba informațiile din câmpurile afișate și va apăsa butonul de „Update”. După această acțiune, ecranul de home, „Coffee Shop” (fig. 4.2) apare. Dacă se dorește delogarea utilizatorului, se va apăsa butonul de „Logout”, iar ecranul din fig. 4.1 va apărea.

Pentru modificarea informațiilor de profil, datele sunt transmise ca parametrii funcției *doUpdateProfile*. Se realizează operația *UPDATE* pentru tabela „users”, dar și operația de *INSERT* în tabela „activity”. Dacă utilizatorul schimbă parola, acesta este direcționat către pagina de Autentificare.

În tab-ul „Vouchers” utilizatorul poate verifica toate voucherele pe care le-a scanat cu ajutorul butonului „Scan” și pe care nu le-a folosit la efectuarea unei comenzi. Voucherele sunt afișate sub forma unui cod unic, parțial ascuns, dar și valoarea discount-ului. Pentru afișarea voucherelor efectuați o interogare în tabela „voucher” de tip *SELECT...JOIN*. Deoarece Mongo nu este o bază de date relatională, operația de *JOIN* nu este posibilă, iar pentru realizarea unui echivalent se selectează toate voucherelor din colecția „users\_voucher” care aparțin utilizatorului cu *id-ul* respectiv. Pentru fiecare document, se caută în tabela „vouchers” codul voucherului scanat și discount-ul oferit.

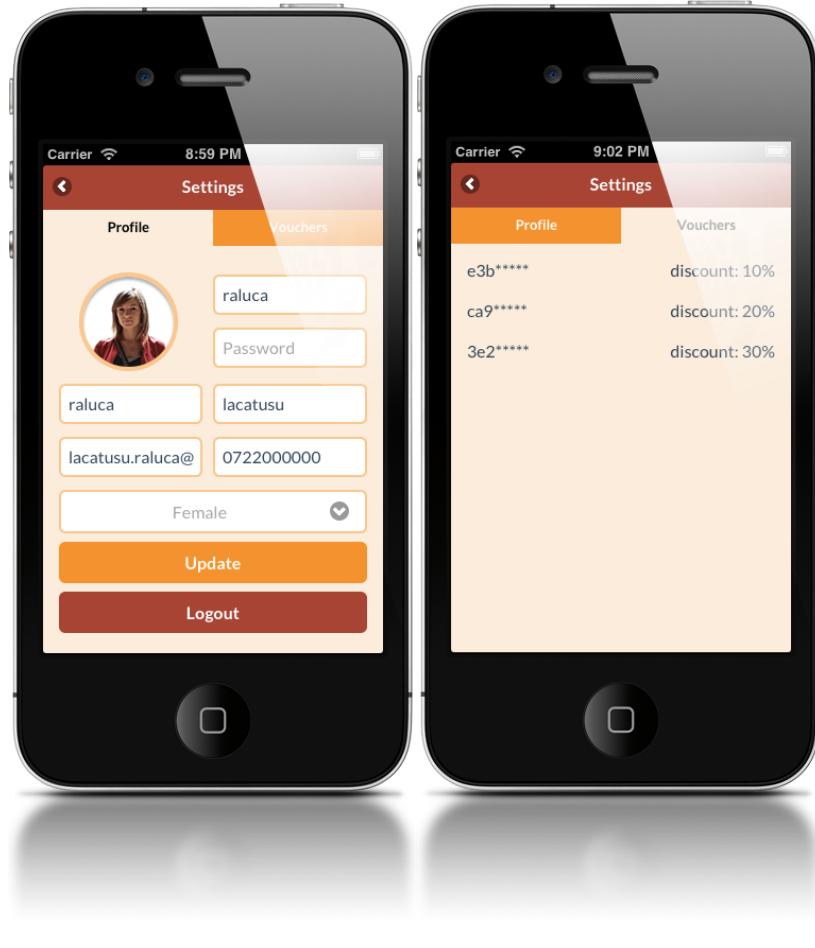


Fig. 4.3 Secțiunea *Settings* (*Profile* și *Vouchers*)

#### 4.1.4 Secțiunea **Menu**

În acest ecran(fig. 4.4) se ajunge apăsând din secțiunea „Coffee Shop”(fig. 4.2) butonul „Menu”. Aici putem regăsi o listă cu toate produsele disponibile în cafenea. Fiecare produs are titlu, imagine, preț, dar și două butoane cu ajutorul cărora îl putem adăuga la lista de favorite sau în coșul de cumpărături. De asemenea, tot din acest ecran putem să accesăm detaliiile utilizatorului prin apăsarea butonului din dreapta-sus.

Putem căuta anumite produse din meniu completând campul de „Filter items...”. Din acest ecran ne putem întoarce la ecranul principal apăsând butonul „<”. În dreapta sus găsim butonul de „Settings”.

Dacă apăsăm butonul „+” produsul este adăugat în coșul de cumpărături (*Basket*) și vom primi o alertă „Added to Basket”. Dacă apăsăm butonul de adăugare în lista de favorite, culoarea butonului se va schimba din roșu în portocaliu, semnalizând astfel acțiunea.

Pentru afișarea produselor din meniu folosesc, în cazul MySQL, operația *SELECT...JOIN*. Utilizez această interogare pentru a afișa toate produsele din meniu, dar și pentru a evidenția produsele adăugate la lista de favorite de către utilizatorul logat. În cazul MongoDB selectez toate produsele din meniu, parcurg lista de produse și verific dacă în colecția „*favourites*” există intrări pentru produsul respectiv și utilizatorul curent. Dacă găsesc o intrare în colecția „*favourites*”, atunci la obiectul de tip produs adaug proprietatea de *favourites* = 1.



Fig. 4.4 Secțiunea *Menu*

Fig. 4.5 Secțiunea *Basket*

#### 4.1.5 Secțiunea *Basket*

Această secțiune (fig. 4.6) reprezintă coșul de cumpărături al utilizatorului. Produsele din listă au fost adăugate folosind ecranul „Menu”, reprezentat în fig. 4.4. În stânga numelui unui produs este afișată cantitatea comandată. Cantitatea poate fi modificată folosind butoanele „-” și „+”. Sub numele produsului comandat este afișat și prețul. Putem căuta un produs anume completând câmpul de „Filter items...”.

Pentru afișarea produselor conținute în coș de către utilizatorul logat, efectuez o operație de *SELECT...JOIN* pentru tabelele „menu” și „basket”. Dacă utilizatorul nu are niciun produs adăugat în coș, atunci se adaugă în vectorul *items* produsul respectiv. Pentru baza de date MongoDB folosesc funcția *find* pentru a căuta în colecția „basket” produsele salvate de utilizatorul logat. Parcureg obiectul returnat de operația de intergoare și dacă produsul care urmează a fi adăugat în coș există deja, se incrementează cantitatea.

După lista de produse putem observa un buton conținând prețul total al comenzii. Valoarea acestui buton se modifică automat la actualizarea coșului. După apăsarea acestui buton ecranul aplicației se va schimba în ecranul „Review Order”.

Dacă nu avem niciun produs adăugat în coș, atunci vom primi o alertă cu mesajul „Basket is empty”. În ecranul acestei secțiuni regăsim butoanele de *Back* și *Settings*. Dacă nu, este adăugat ca un nou document.

#### 4.1.6 Secțiunea Locations

O facilitate importantă oferită de această aplicație este integrarea unei hărți Google. Harta apare în tabul „Map” din secțiunea „Locations” (fig. 4.5). Pe această hartă sunt reprezentate locațiile exacte ale cafenelelor, oferind posibilitatea utilizatorului de a alege cea mai apropiată cafea față de poziția utilizatorului.

Pe această hartă sunt afișate pinpoint-uri de culoare portocalie, reprezentând locațiile cafenelelor. Harta oferă o imagine de ansamblu, având în centru locația utilizatorului, reprezentată de un pinpoint roșu. Există posibilitatea de a mări sau micșora harta, de a modifica ecranul hărții. Harta prezintă străzi, folosirea unor hărți cu imagini satelit solicitând o cantitate mare de informații, deci o vitează mai mică de randare.

Dacă se apasă pe un pinpoint portocaliu, este afișat un pop-up cu informații despre locația respectivă: numele cafenelei, adresa, numărul de telefon și adresa de e-mail. Acest pop-up se poate închide apăsând butonul „x”. Deoarece integrarea hărții necesită o conexiune la internet, în lipsa acesteia va apărea eroarea „Could not connect to server”. În tabul „List” regăsim o listă cu toate cafenelele din rețeaua „Coffee Shop”. Fiecare cafea prezintă următoarele detalii: nume, adresă, telefon, adresa de e-mail, dar și distanța de la poziția utilizatorului până la cafeneaua respectivă. Distanța este calculată cu următoarea formulă:

```
6371 * acos(cos(radians(latitudinea_locatiei_utilizatorului)) *  
cos(radians(latitudinea_locatiei_cafenelei)) * cos(radians(longitudinea_locatiei_utilizatorului) -  
radians(longitudinea_locatiei_cafenelei)) + sin(radians(latitudinea_locatiei_utilizatorului)) *  
sin(radians(latitudinea_locatiei_cafenelei)))
```

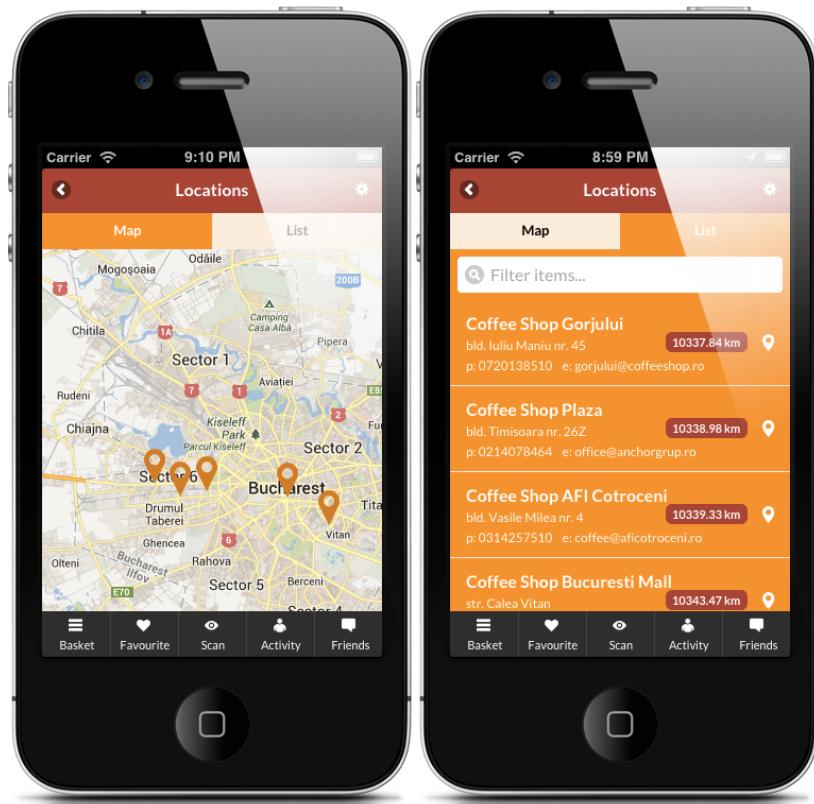


Fig. 4.5 Secțiunea Locations(Map și List)

#### 4.1.7 Review Order

În acest ecran (fig. 4.7) se ajunge după apăsarea butonului cu prețul total al comenzi. Dacă utilizatorul nu are scanate vouchere, atunci pe ecran va apărea un mesaj care explică următorul pas în procesul de efectuare a comenzi, dar și prețul total al comenzi.

Dacă utilizatorul are scanate vouchere, atunci va apărea o listă cu discount-urile aferente. Dacă se dorește aplicarea unui discount se va selecta un voucher, iar valoarea comenzi se va modifica. Nu se poate selecta aplicarea mai multor discount-uri. După apăsarea butonului cu valoarea comenzi, aplicația ne va trimite în ecranul „Order Confirmed”.

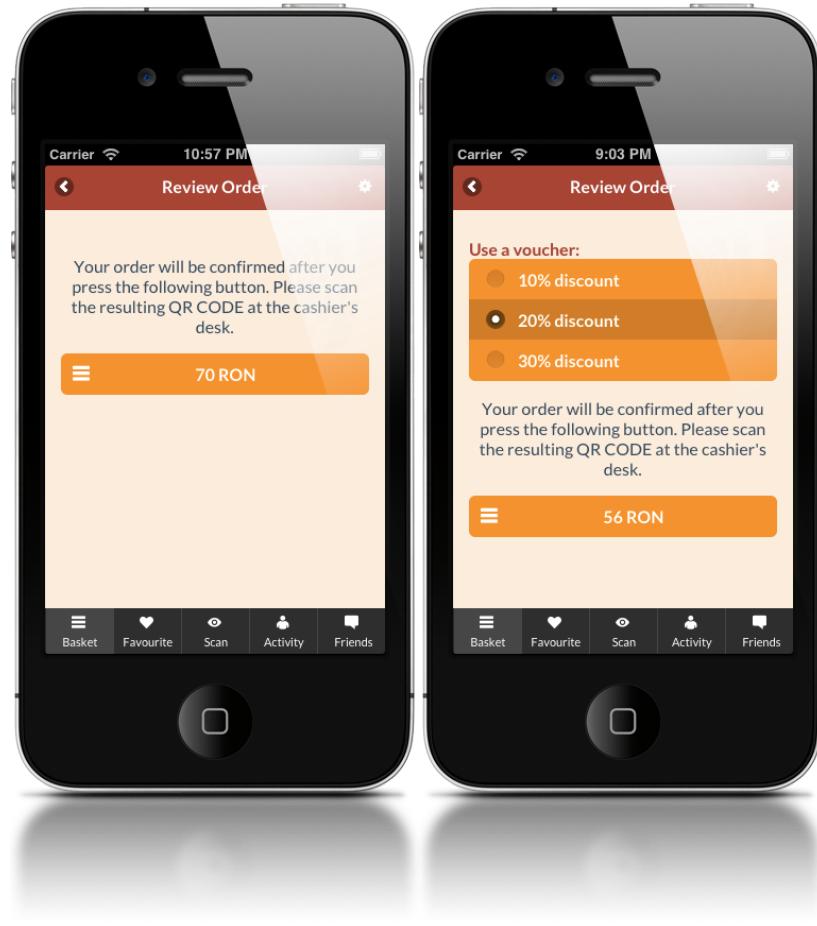


Fig. 4.7 Review Order

#### 4.1.8 Order Confirmed

În acest ecran (fig. 4.8) se va ajunge după apăsarea butonului cu valoarea totală a comenzi. Se va genera un QR-code care conține codul comenzi, memorat în baza de date. Cu aplicația de administrare se va scana acest cod, extrăgându-se din baza de date detaliile comenzi. Această căutare în baza de date se va face după codul comenzi care se obține decodând QR-code-ul. În ecranul acestei secțiuni regăsim butoanele de *Back* și *Settings*.

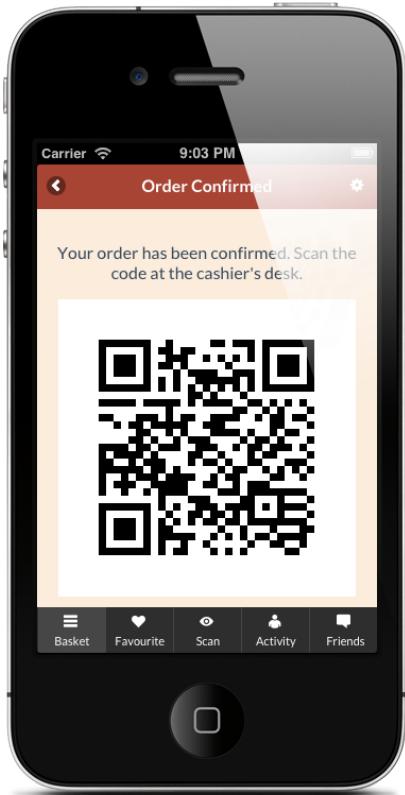


Fig. 4.8 Order Confirmed

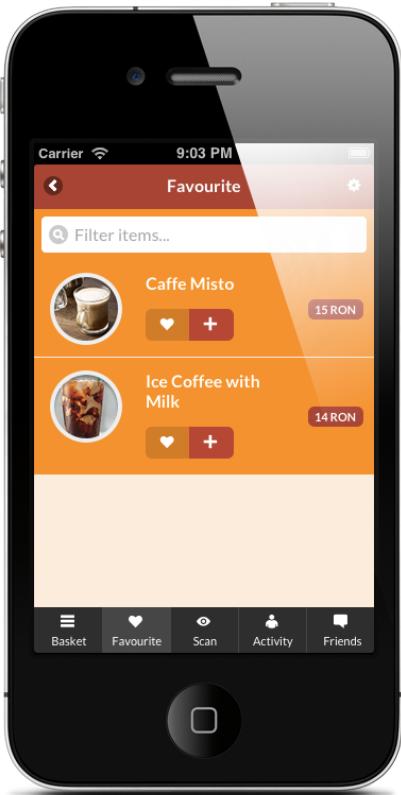


Fig. 4.9 Secțiunea Favourite

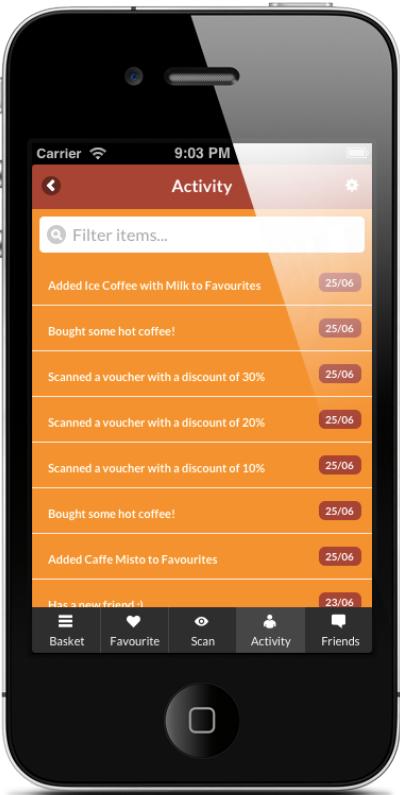


Fig.4.10 Secțiunea Activity

#### 4.1.9 Secțiunea Favourite

Acest ecran (fig. 4.9) este identic cu ecranul secțiunii „Menu”, singura diferență fiind că produsele afișate sunt cele adăugate de utilizator în această secțiune. Un produs poate fi eliminat din această listă prin apăsarea butonului de favorite. Dacă nu avem niciun produs adăugat va apărea următorul mesaj: „Sorry, no available items”. Detaliile produselor care apar sunt numele produsului și prețul acestora. În ecranul acestei secțiuni regăsim butoanele de *Back* și *Settings*. Putem căuta un produs anume completând câmpul de „Filter items...”.

Pentru baza de date de tip MongoDB, efectuez o interogare în colecția *favourites*, având ca filtru id-ul utilizatorul logat. Pentru fiecare document caut în tabela „menu” detalii despre produsul respectiv.

#### 4.1.10 Secțiunea Activity

În această secțiune (fig. 4.10) putem vedea istoricul tuturor acțiunilor efectuate de utilizator. În dreptul fiecărei acțiuni este afișată data în care s-a efectuat. Dacă dorim filtrarea sau căutarea unor acțiuni se poate completa câmpul de „Filter items...”. În ecranul acestei secțiuni regăsim butoanele de *Back* și *Settings*.

Pentru baza de date de tip MySQL, efectuez o interogare în tabela *history*, folosind operația de *SELECT...WHERE...ORDER BY*, având ca filtru id-ul utilizatorului logat. Datele vor fi ordonate

descrescător în funcție de valoarea câmpului „date”. Dacă se găsesc intrări care îndeplinesc condiția, se parcurg și se adaugă în vectorul *items*.

În cazul MongoDB, folosesc funcția *find* pentru a căuta în colecția *history*, având ca parametru id-ul utilizatorului logat.

#### 4.1.11 Secțiunea Friends

Acest ecran (fig. 4.11) conține lista de prieteni ai utilizatorului logat. Dacă se dă click pe un prieten din listă, ecranul se va schimba într-un ecran în care vom vedea activitatea/istoricul prietenului respectiv. Un prieten se poate adăuga apăsând butonul „+”. Va apărea un pop-up în care trebuie să se adauge adresa de e-mail pentru a putea găsi utilizatorul. După apăsarea butonului „Invite” se va trimite o cerere către utilizatorul invitat. Pe ecran va apărea un mesaj de alertă prin care se confirmă invitația. Utilizatorul invitat primi un mesaj „Friend Request” cu două variante de răspuns: „Accept” și „Reject”.

Dacă nu există niciun utilizator cu adresa de e-mail introdusă vom primi un mesaj „Wrong email address”. Dacă utilizatorul nu are niciun prieten invitat va apărea mesajul de alertă „You have no friends. Try to invite some”.



Fig. 4.11 Secțiunea Friends

## 4.2 Structura bazei de date MySQL

Tabele și tipurile datelor stocate, relațiile dintre tabele sunt exemplificate în anexa 1. Tabela „users” este relaționată cu tabelele „basket\_order” („id” - „user”, ambele câmpuri fiind de tip *int*), „basket” („id” - „user”, ambele câmpuri fiind de tip *int*), „friends” („id” - „request”/„accept”, câmpurile fiind de tip *int*), „favourites” („id” - „user”, ambele câmpuri fiind de tip *int*), „users\_voucher” („id” - „user”, ambele câmpuri fiind de tip *int*), „history” („id” - „user”, ambele câmpuri fiind de tip *int*).

Poza realizată de utilizator la crearea contului este trimisă codată *BASE64* către server, apoi este salvată în câmpul *photo*, în tabela „users” și este de tip *LONGLOB*, lucru care a îngreunat operațiile de interogare a tabelei.

Tabela „basket” este relaționată și cu tabela „menu” prin câmpurile product/id, ambele de tip *int*. De asemenea, această tabelă este relaționată și cu tabela „basket\_order” prin câmpurile product/id, ambele de tip *int*. Tabela „users\_voucher” este relaționată și cu tabela „voucher” prin câmpurile voucher/id, ambele de tip *int*.

Schema bazei de date Mongo nu a putut fi reprezentată sub forma unei diagrame, dar relațiile dintre tabele sunt similare.

## 4.3 Teste de performanță

### 4.3.1 Testarea aplicației la nivel de Client

Am testat aplicația atât pe simulatorul pus la dispoziție de programul Xcode(v. 4.6.3), cât și pe un iPhone 4S. Testarea folosind simulatorul a fost făcută pentru a verifica funcționalitatea aplicației, dar și pentru a detecta eventualele erori. Erorile le-am putut identifica folosind debugger-ul și logarea în consolă a evenimentelor produse. Totuși, un dezavantaj major al simulatorului este acela că nu sunt disponibile informații despre poziționarea geografică, lucru care m-a împiedicat să testeze funcționalitatea hărții și a serviciilor de localizare. Un alt dezavantaj este că nu pot folosi camera pentru a scana codurile QR și detecta posibilele erori. De asemenea, nu este disponibilă testarea aplicației folosind mai multe tipuri de conexiuni la internet.

La testarea aplicației direct pe iPhone am întâlnit erori legate de serviciile de detectare a poziției pe hartă și a distanțelor până la cea mai apropiată locație.

### 4.3.2 Testarea bazelor de date

Pentru testarea bazelor de date și a eficienței interogărilor am creat script-uri care încarcă colecțiile/tabelele cu 10000, 100000, 1000000 de intrări, precum și script-uri care realizează operații de selectare a tuturor intrărilor dintr-o colecție/tabelă, operații de selectare a intrărilor care îndeplinesc una sau mai multe condiții, operații de combinare a intrărilor din două colecții/tabele, operații de actualizare a intrărilor, operații de stergere și de adăugare. Testarea automată a fost realizată folosind programul Apache Benchmark(v. 2.3) cu următoarele specificații:

- pentru 10000 intrări am folosit 10 sesiuni concurente, 100 interogări (fig. 4.12 - 4.19)
- pentru 100000 intrări am folosit 2 sesiuni concurente, 4 interogări (fig. 4.20 - 4.27)
- pentru 1000000 intrări am folosit 1 sesiune, 2 interogări (fig. 4.28 - 4.35)

Testele au fost realizate pe un computer Apple Macintosh cu procesor Intel Core 2 Duo, 2.53 GHz, 8 GB RAM. Pentru evitarea erorilor de tip *Connection Timed Out* am mărit timpul de așteptare al răspunsului din partea server-ului.

#### Teste de performanță pentru 10000 intrări

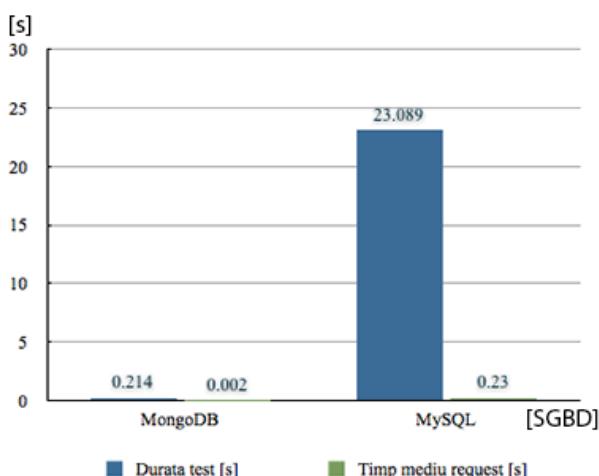


Fig. 4.12 Operația *SELECT \**

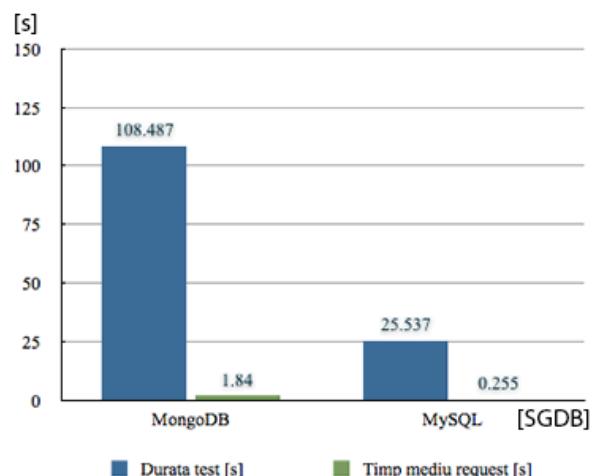


Fig. 4.13 Operația *SELECT...JOIN*

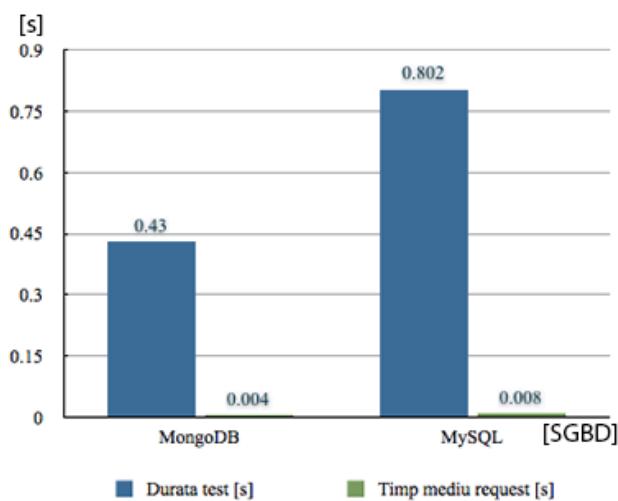


Fig. 4.14 Operația *SELECT...WHERE*

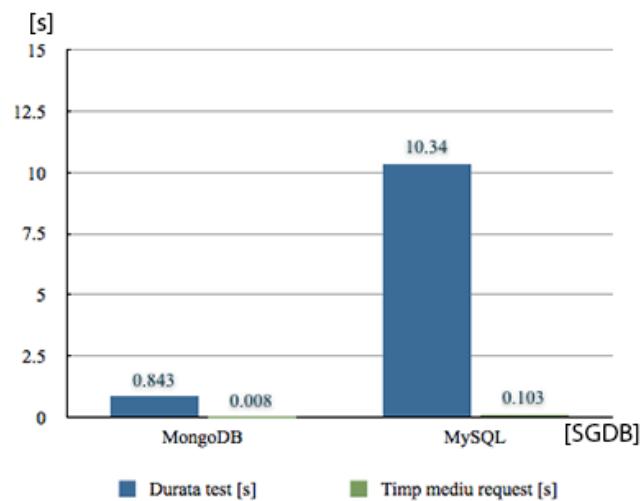


Fig. 4.15 Operația *SELECT...WHERE...AND*

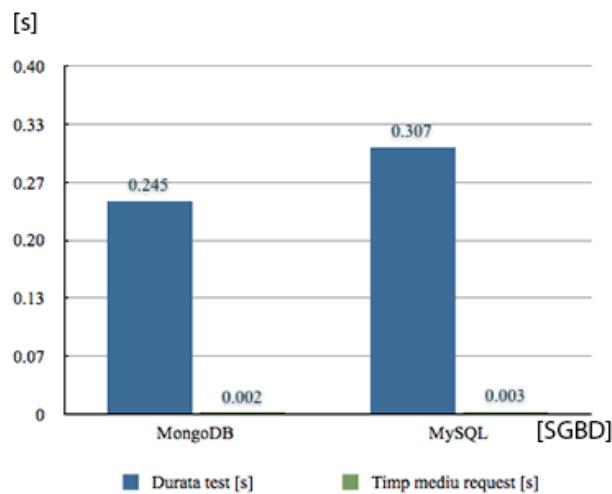


Fig. 4.16 Operația *UPDATE*

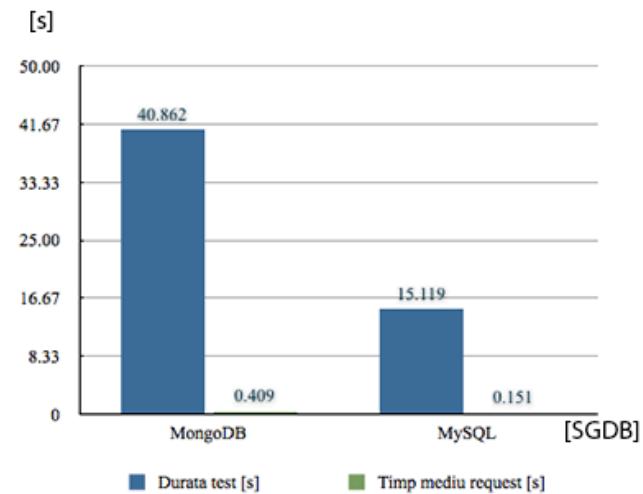


Fig. 4.17 Operația *UPDATE NULL*

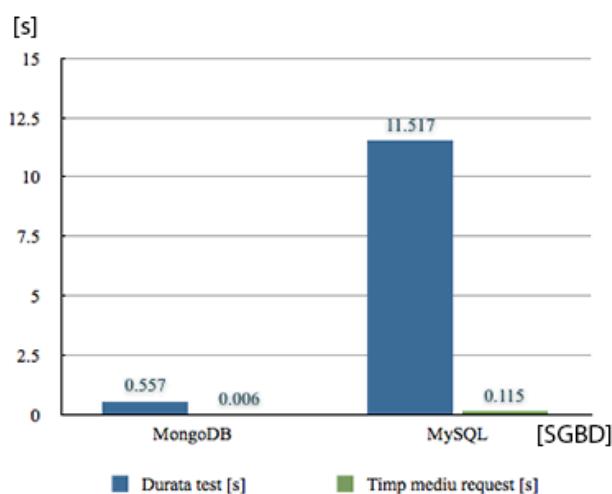


Fig. 4.18 Operația *DELETE*

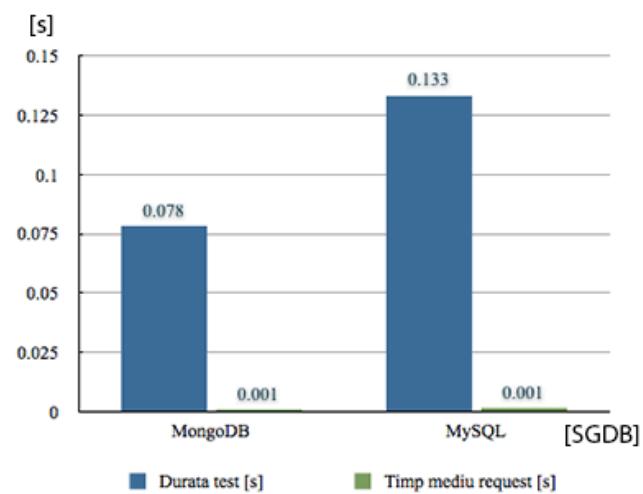


Fig. 4.19 Operația *INSERT*

## Teste de performanță pentru 100000 intrări

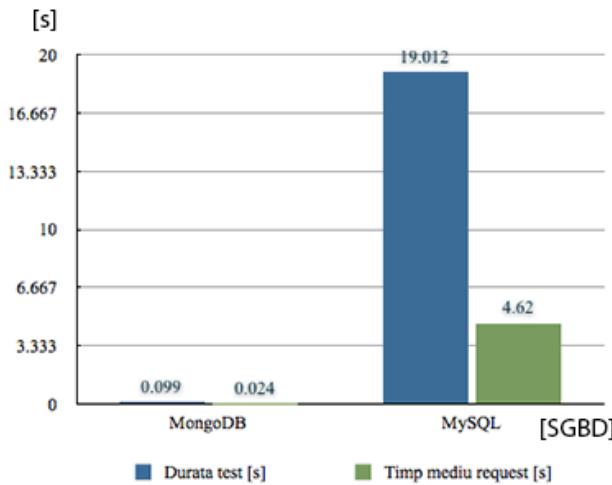


Fig. 4.20 Operația *SELECT \**

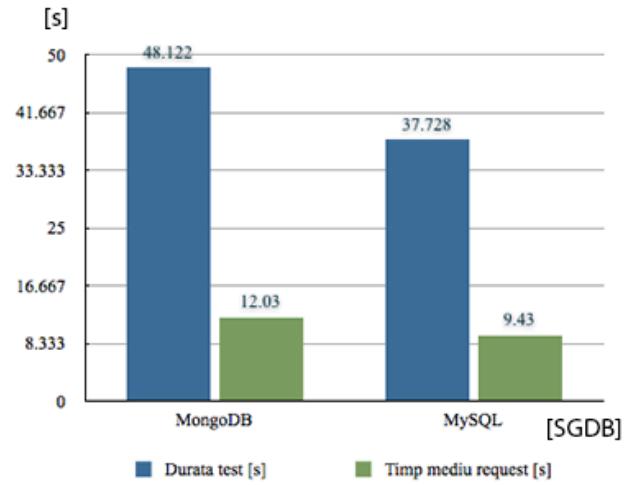


Fig. 4.21 Operația *SELECT...JOIN*

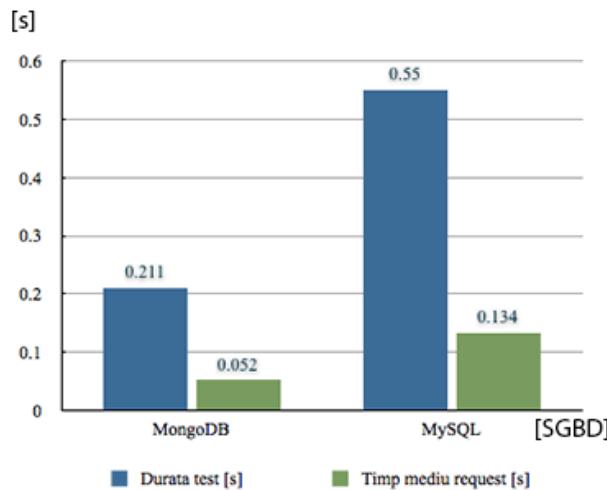


Fig. 4.22 Operația *SELECT...WHERE*

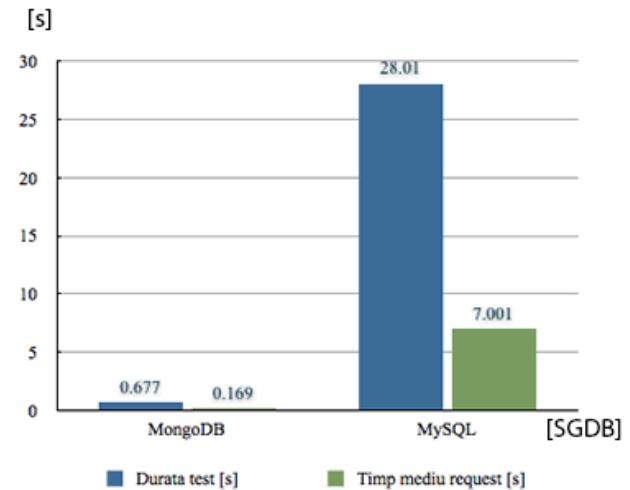


Fig. 4.23 Operația *SELECT...WHERE...AND*

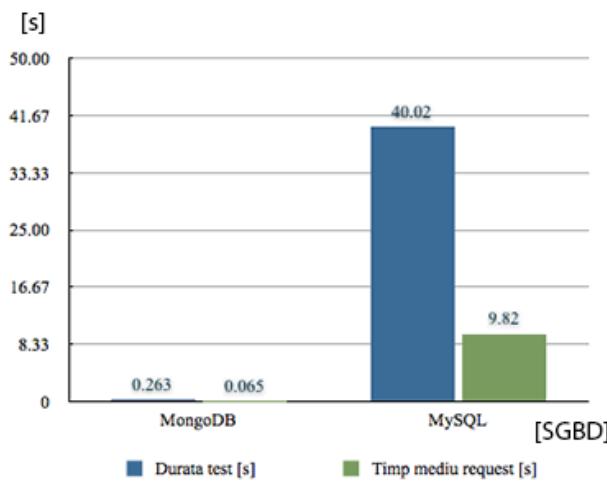


Fig. 4.24 Operația *UPDATE*

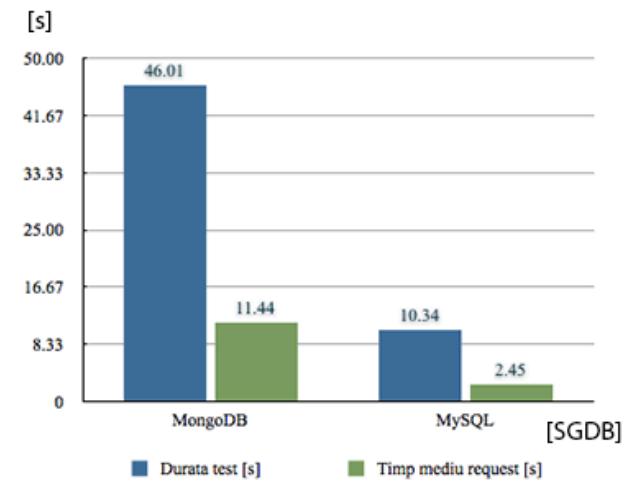


Fig. 4.25 Operația *UPDATE NULL*

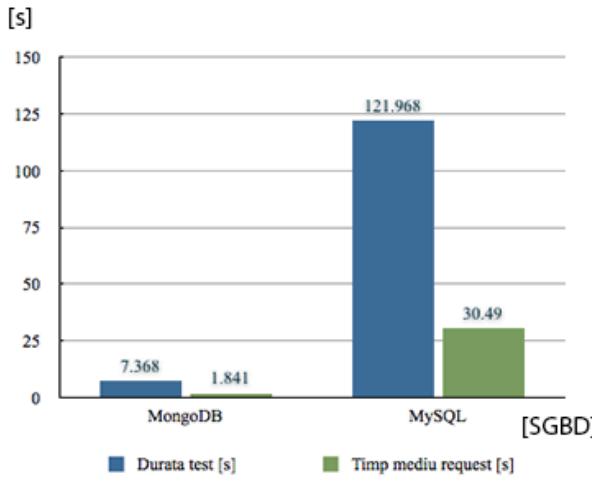


Fig. 4.26 Operația *DELETE*

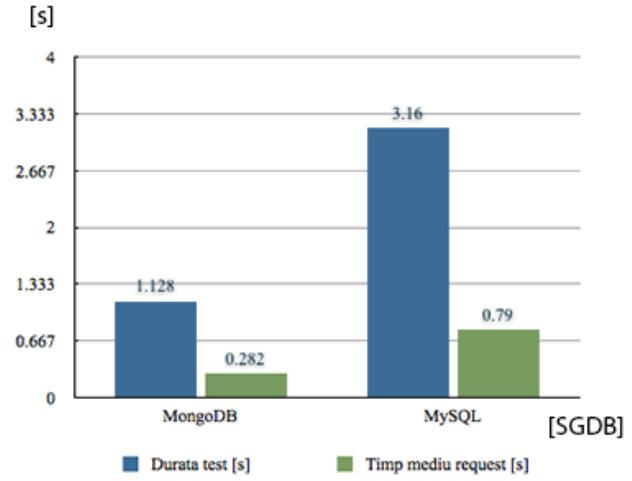


Fig. 4.27 Operația *INSERT*

#### Teste de performanță pentru 1000000 intrări

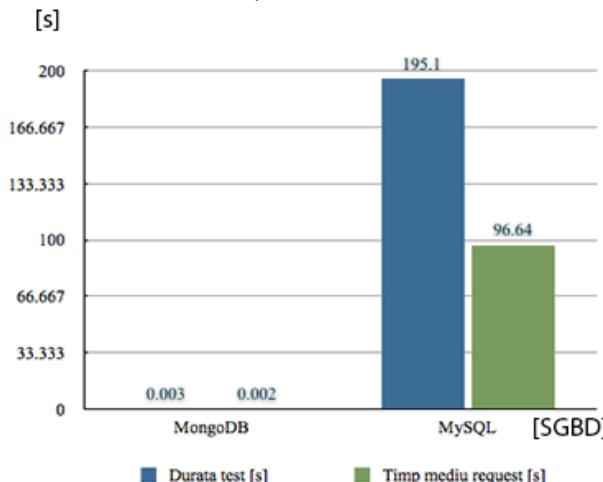


Fig. 4.28 Operația *SELECT \**

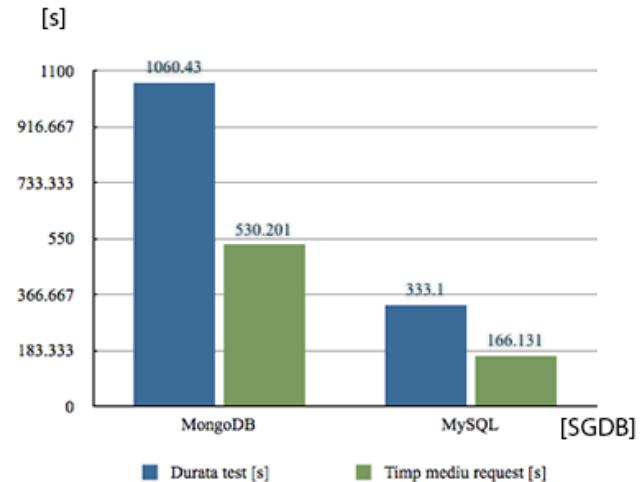


Fig. 4.29 Operația *SELECT...JOIN*

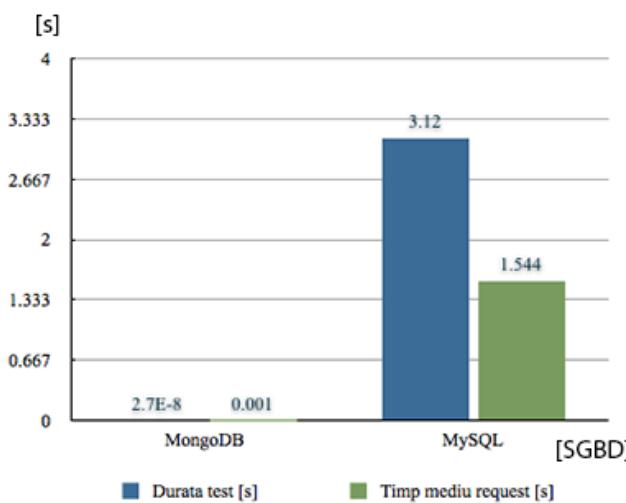


Fig. 4.30 Operația *SELECT...WHERE*

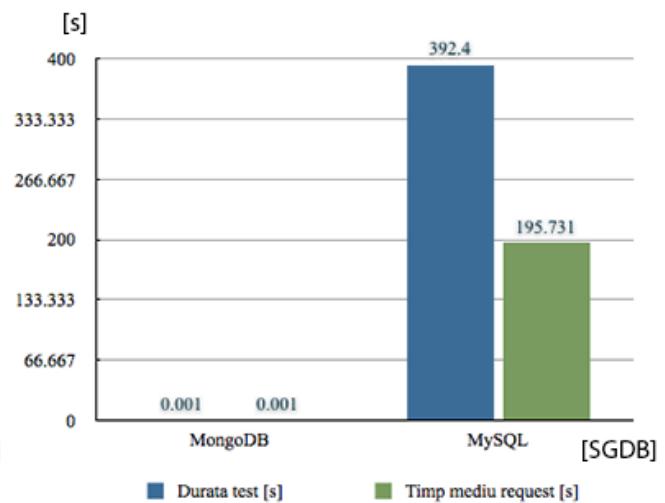


Fig. 4.31 Operația *SELECT...WHERE...AND*

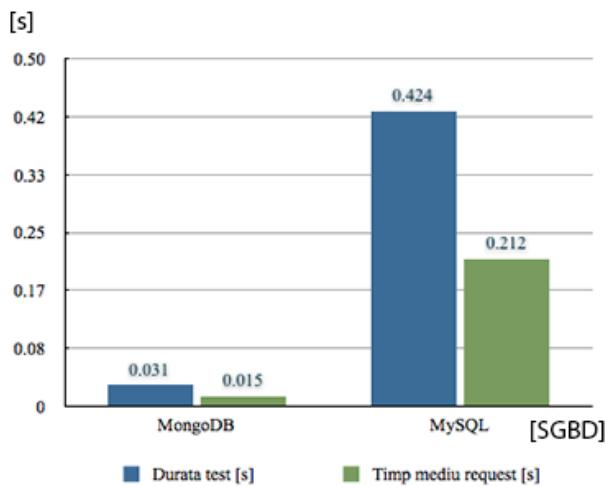


Fig. 4.32 Operația *UPDATE*

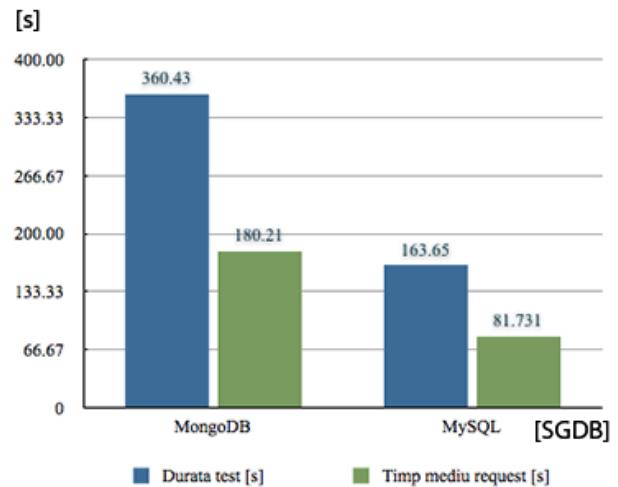


Fig. 4.33 Operația *UPDATE NULL*

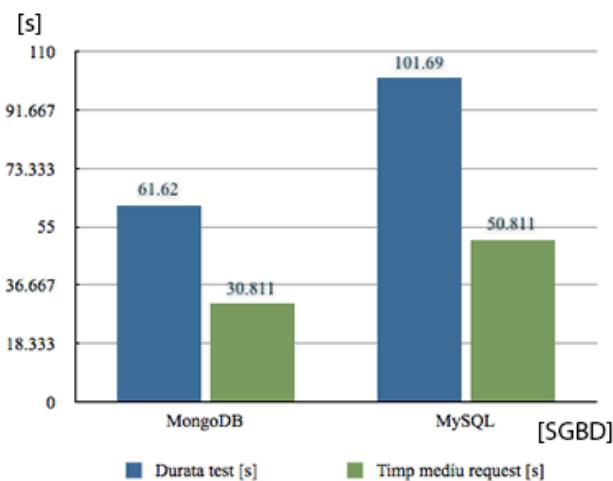


Fig. 4.34 Operația *DELETE*

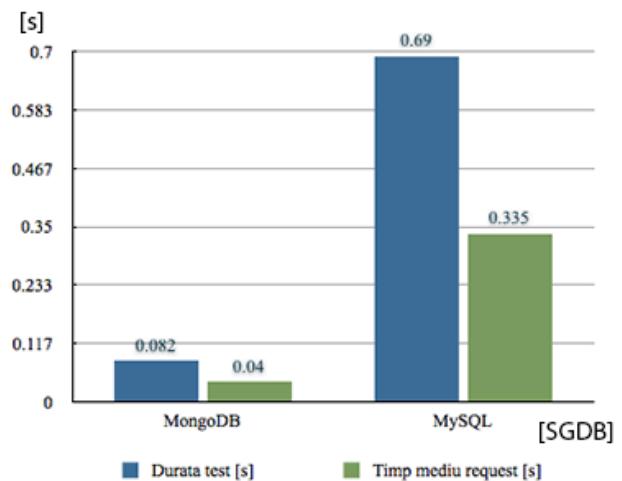


Fig. 4.35 Operația *INSERT*



## Concluzii

Aplicațiile care folosesc un framework de tip PhoneGap, Sencha, Appcelerator Titanium, LiveCode etc. sunt mult mai ușor de implementat atât din punct de vedere al interfeței cu utilizatorul, cât și al compatibilității cu diferitele sisteme de operare pentru telefoanele inteligente și tablete.

Serviciile web apelate de către o aplicație de acest tip pot fi schimbatе foarte ușor, nefiind nevoie de un update al acesteia în magazinul de aplicații. Prin folosirea tehnologiilor HTML, CSS, JavaScript se pot realiza noi interfețe, animații, funcționalități față de cele puse la dispoziție de SDK-ul specific platformei.

Multe companii, servicii au deja o versiune mobilă pentru website-uri, astfel că realizarea unei aplicații folosind o platformă de tip PhoneGap este foarte rapidă, presupunând doar configurarea platformei de dezvoltare și copierea codului HTML/CSS/JavaScript.

Aplicațiile construite pe platforma PhoneGap permit accesarea diferitelor funcționalități puse la dispoziție de SDK prin crearea unor plugin-uri ce pot fi apelate prin cod JavaScript (ex: cititorul de coduri QR). Majoritatea platformelor de dezvoltare (SDK) oferă suport doar pentru baze de date de tip SQL Lite printr-un driver incorporat. Prin folosirea unei platforme de tip PhoneGap putem crea servicii care să se conecteze la orice tip de bază de date, utilizând un driver dedicat pentru Apache, Lighttpd, IIS, IBM WebSphere Application Server etc. De asemenea, avem acces și la o bază internă SQL Lite prin plugin-ul PhoneGap.

Un dezvoltator poate crea cu ușurință o aplicație pe care să o publice pe toate platformele mobile. fără mari eforturi; codul de interfață/servicii web este același, diferă doar mediul de dezvoltare (Xcode pentru iOS, Eclipse pentru Android și BlackBerry, Visual Studio pentru Windows Phone). Comunitățile de dezvoltatori HTML/CSS/JavaScript sunt mai numeroase și oferă suport mult mai rapid decât cele de Objective-C/Java Android etc. (spre exemplu, pe stackoverflow.com din 10 răspunsuri legate de cod, doar unul ține de Objective C, restul fiind de HTML/CSS).

Prin folosirea aplicației Coffee Shop, utilizatorul poate să își adauge singur produsele pe care dorește să le comande. Acest lucru ajută utilizatorul să facă o alegere corectă fără să fie influențat de angajații companiei.

Prin localizarea magazinelor Coffee Shop, utilizatorul rămâne mereu informat asupra locațiilor și promoțiilor curente, putând vizualiza în orice moment dat cea mai apropiată cafenea și ofertele speciale. Prin folosirea aplicației se elimină nevoia de a tipări meniul, ofertele, nota de plată pe suport durabil - hârtie.

Folosirea codurilor QR pentru comunicarea unei comenzi duce la timpi mult mai mici de procesare a comenzilor, astfel eliminându-se cozile de așteptare și nevoia de un număr mare de casieri la orele de vârf.

Bazele de date relationale (în mare parte MySQL) au devenit partea de bază într-o organizație comercială, programatorii și administratorii bazelor de date sunt în permanentă monitorizare pentru optimizarea, back-up-ul și timpul de acces al informației. Încercările recente de a transforma baza de date în obiecte (Hibernate, JPA, iBATIS) au din ce în ce mai mulți adepti, însă performanțele întâmpină anumite limitări, precum probleme de tranzacții, probleme de cache-ing, probleme de transformare etc.

Companiile mici și medii încă folosesc baze de date de tip MySQL din cauza programelor necesare desfășurării activităților zilnice care nu sunt configurabile pentru a lucra cu baze de date NoSQL. Companiile foarte mari folosesc sisteme de baze de date dedicate (Oracle, IBM DB2) datorită suportului enterprise al SLA-ului și a scalabilității. Bazele de tip NoSQL sunt folosite de către entuziaști, companii care au înțeles adevărata lor putere și funcționalitate. Aceste companii folosesc în mare parte soluții open-source și pun la dispoziția utilizatorului API-uri pentru a interacționa cu produsele lor (Craiglist, Foursquare, CERN).

Sintaxa PHP (dar nu numai) pentru a interacționa cu baze de date Mongo este mult mai intuitivă pentru un dezvoltator web (având la bază cod JavaScript; un document Mongo este un obiect JSON), față de cea a MySQL.

Fiind o bază de date fără schemă, putem introduce într-o colecție Mongo documente cu proprietăți diferite.

Ștergerea unei baze de date de tip Mongo presupune eliminarea fizică de pe disk, pe când la ștergerea din MySQL spațiul nu se eliberează automat - fișierul *ibdata1* rămâne la aceeași dimensiune, fiind nevoie de operații suplimentare (drop, truncate, ștergere fizică, import) pentru a elibera spațiul pe disk.

După cum se observă din graficele de performanță, operațiile de *insert* sunt mult mai rapide în Mongo. Acest lucru se datorează faptului că MySQL scrie rezultatul într-un fișier *jurnal* și *log* și apoi întoarce un răspuns script-ului server-side, pe când Mongo scrie documentul direct pe disk sau pe o colecție de disk-uri(replică) și întoarce răspunsul către scriptu-ul server-side.

MySQL folosește tehnici moderne, cum ar fi utilizarea mai multor fire de execuție, atunci când mai multe interogări trebuie să fie procesate în același timp. Astfel, pentru a crește performanțele bazei de date, sunt utilizate mai multe multe fire de execuție pentru a lucra în paralel. Dacă avem o tabelă cu multe intrări și avem nevoie să efectuăm un număr mare de interogări, MySQL procesează aceste query-uri simultan. Conceptul intern de MongoDB este complet diferit de cel al MySQL-ului. Interrogările în MongoDB nu se efectuează simultan. Acest lucru înseamnă că toate interogările sunt trimise către un server MongoDB în coada de așteptare. Server-ul procesează interogările pe rând, dar se pot utiliza replici ale bazelor de date pentru a realiza query-uri simultan.

Chiar dacă MongoDB realizează interogări pe tabele mult mai repede, acest lucru nu înseamnă că acest sistem de baze de date este cel mai fiabil. Lipsa indicilor din MongoDB poate avea un impact asupra produsului software.

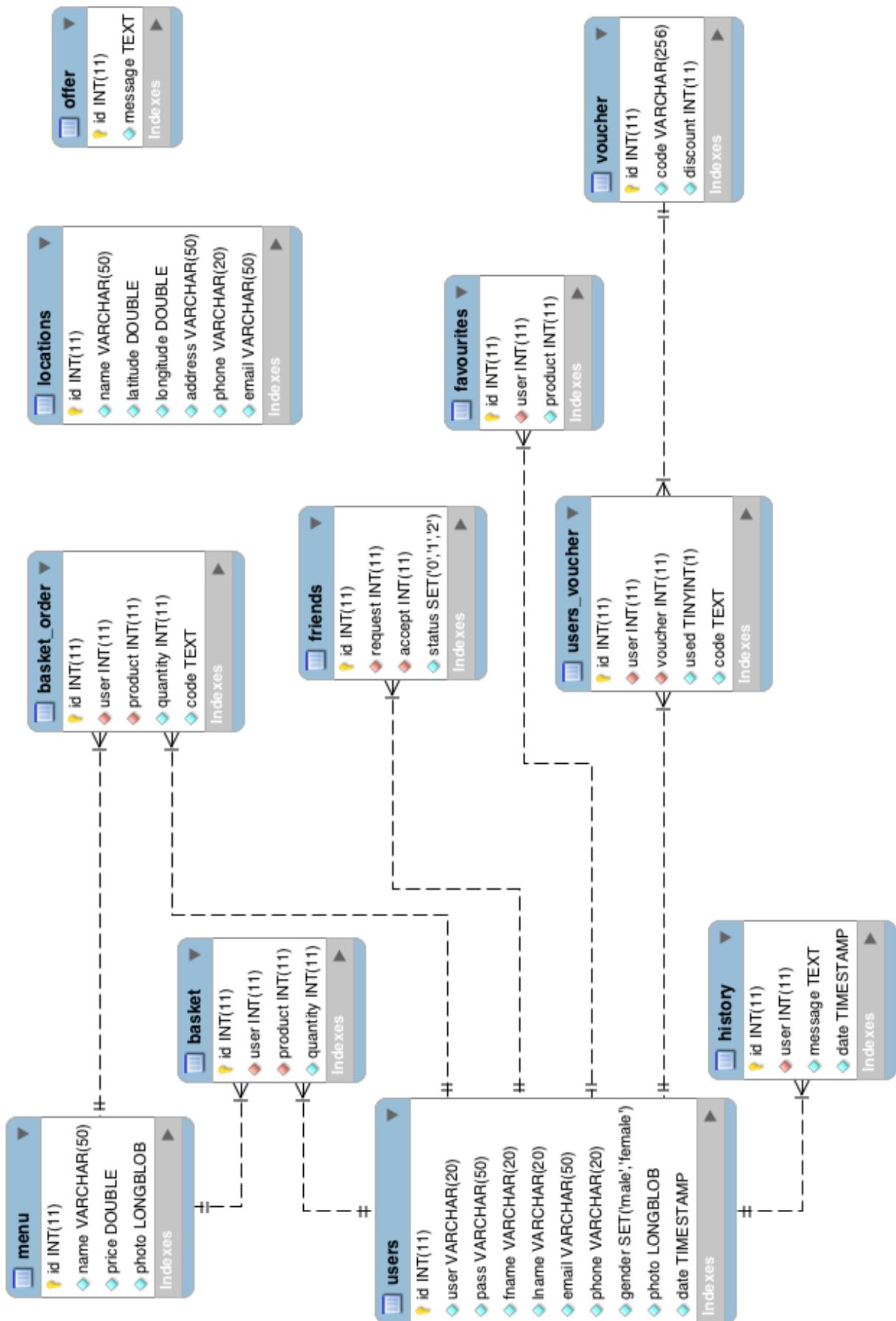
După cum putem vedea din graficele de mai sus, MySQL rulează anumite interogări (*JOIN*) mult mai repede decât MongoDB. Acest lucru se datorează faptului că MongoDB nu poate efectua operația *JOIN* server-side. Toate operațiile de acest tip sunt efectuate în MongoDB client-side. Acest lucru înseamnă un timp mai mare de execuție față de MySQL.

## Bibliografie

- [1] - <https://en.wikipedia.org/wiki/MySQL> - accesat la data de 15.05.2013
- [2],[3],[4],[5],[6],[7],[8],[9],[10] - NIXON R., Learning PHP, MySQL, JavaScript, and CSS, Second Edition, O'Reilly Media, Inc., United States of America, August 2012, ISBN: 978-1-449-31926-7
- [11] - FRANCIA S., MongoDB and PHP, First edition, O'Reilly Media, Inc., United States of America, January 2012, ISBN: 978-1-449-31436-1
- [12] - CHODOROW K., DIRELF M., MongoDB: The Definitive Guide, First Edition, O'Reilly Media, Inc., United States of America, September 2010, ISBN: 978-1-449-38156-1
- [13] - <http://ro.wikipedia.org/wiki/PHP> - accesat la data 16.05.2013
- [14] - YANK K., PHP & MySQL: Novice to Ninja, Fifth Edition, SitePoint Pty. Ltd., United States of America, 2012, ISBN: 978-0-9872478-1-0
- [15] - <http://ro.wikipedia.org/wiki/POO> - accesat la data 16.05.2013
- [16], [17] - TATROE K., MACINTYRE P., LERDORF R., Programming PHP, Third Edition, O'Reilly Media, Inc., United States of America, February 2013, ISBN: 978-1-449-39277-2
- [18] - [http://ro.wikipedia.org/wiki/HyperText\\_Markup\\_Language](http://ro.wikipedia.org/wiki/HyperText_Markup_Language) - accesat la data de 17.05.2013
- [19],[20],[21],[22] - DUCKETT J., Beginning HTML, XHTML, CSS, and JavaScript, Wiley Publishing, Inc., Indianapolis, Indiana, 2010, ISBN: 978-0-470-54070-1
- [23],[24] - FIRTMAN M., jQuery Mobile: Up and Running, O'Reilly Media, Inc., United States of America, February 2012, ISBN: 978-1-449-39765-4



## Anexa 1 (schema bazei de date MySQL)



## Anexa 2 (codul sursă)

### *index.html*

```
<!DOCTYPE html>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <meta name="format-detection" content="telephone=no" />
        <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1, minimum-scale=1, width=device-width, height=460, target-densitydpi=device-dpi" />
        <link rel="stylesheet" type="text/css" href="css/jquery.mobile.flatui.css" />
        <link rel="stylesheet" href="css/style.css" />
        <title>Coffee Shop</title>
    </head>
    <body onload="init()">
        <div id="launcherPage" data-role="page">
        </div>
        <div id="loginPage" data-role="page">
            <div data-role="header">
                <h1>Authentication</h1>
                <a id="register" data-icon="plus" class="ui-btn-left" data-iconpos="notext" href="#addUser">Add User</a>
            </div>
            <div data-role="content">
                <center>
                    
                </center>
                <form id="loginForm">
                    <div data-role="fieldcontain" class="ui-hide-label">
                        <label for="username">Username:</label>
                        <input type="text" name="username" id="username" value="" placeholder="Username" />
                    </div>
                    <div data-role="fieldcontain" class="ui-hide-label">
                        <label for="password">Password:</label>
                        <input type="password" name="password" id="password" value="" placeholder="Password" />
                    </div>
                    <input type="submit" value="Login" id="submitButton" data-theme="b">
                </form>
                <div style="text-align:center;">
                    <a href="#forgotPassword">Forgot your password?</a>
                </div>
            </div>
        <div id="addUser" data-role="page">
            <div data-role="header">
                <h1>Registration</h1>
                <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext" data-rel="back">Back</a>
            </div>
            <div data-role="content">
                <form id="registerForm">
                    <div class="ui-grid-a">
                        <div class="ui-block-a">
                            <div style="margin-left:-14px;padding-top:14px;text-align:center;">
                                
                            </div>
                        </div>
                        <div class="ui-block-b">
                            <div data-role="fieldcontain" class="ui-hide-label" data-type="horizontal">
                                <label for="rusername">Username:</label>
                                <input type="text" name="rusername" id="rusername" value="" placeholder="Username" />
                            </div>
                            <div data-role="fieldcontain" class="ui-hide-label">
                                <label for="rpassword">Password:</label>
```

```

        <input type="password" name="rpassword" id="rpassword" value="" placeholder="Password" />
            </div>
        </div>
    </div>

    <div class="ui-grid-a">
        <div class="ui-block-a" style="padding-right:10px;">
            <div data-role="fieldcontain" class="ui-hide-label">
                <label for="rfname">First Name:</label>
                <input type="text" name="rfname" id="rfname" value="" placeholder="First Name" />
            </div>
        </div>
        <div class="ui-block-b">
            <div data-role="fieldcontain" class="ui-hide-label">
                <label for="rlname">Last Name:</label>
                <input type="text" name="rlname" id="rlname" value="" placeholder="Last Name" />
            </div>
        </div>
    </div>

    <div class="ui-grid-a">
        <div class="ui-block-a" style="padding-right:10px;">
            <div data-role="fieldcontain" class="ui-hide-label">
                <label for="remail">Email:</label>
                <input type="email" name="remail" id="remail" value="" placeholder="Email" />
            </div>
        </div>
        <div class="ui-block-b">
            <div data-role="fieldcontain" class="ui-hide-label">
                <label for="rphone">Phone:</label>
                <input type="tel" name="rphone" id="rphone" value="" placeholder="Phone" />
            </div>
        </div>
    </div>
    <div data-role="fieldcontain" class="ui-hide-label gender-select">
        <select name="rgender" id="rgender" data-theme="a">
            <option value="male">Male</option>
            <option value="female">Female</option>
        </select>
    </div>
    <div data-role="fieldcontain" class="ui-hide-label" style="height:0 !important;">
        <textarea id="rpicture" style="display:none;"></textarea>
    </div>
    <input type="submit" value="Register" id="registerButton" data-theme="b"/>
</form>
</div>
</div>
<div id="forgotPassword" data-role="page">
    <div data-role="header">
        <h1>Reset Password</h1>
        <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext" data-rel="back">Back</a>
    </div>
    <div data-role="content">
        <form id="resetForm">
            <div data-role="fieldcontain" class="ui-hide-label">
                <label for="femail">Email:</label>
                <input type="email" name="femail" id="femail" value="" placeholder="Email" />
            </div>
            <input type="submit" value="Reset Password" id="forgotButton" data-theme="b"/>
        </form>
    </div>
</div>
<div id="mainPage" data-role="page">
    <div data-role="header" data-position="fixed" data-tap-toggle="false">
        <h1>Coffee Shop</h1>

```

```

        <a data-icon="flat-settings" class="ui-btn-right" data-iconpos="notext"
href="#settingsPage">Settings</a>
    </div>
    
    <div data-role="content" class="menuBtn">
        <a href="#" data-role="button" data-icon="flat-checkround" data-theme="b"
data-iconpos="right">Menu</a>
        <a href="#" data-role="button" data-icon="flat-location" data-theme="b"
data-iconpos="right">Locations</a>
        <a href="#" data-role="button" data-icon="flat-calendar" data-theme="b"
data-iconpos="right">Special Offers</a>
    </div>
    <div data-role="footer" data-position="fixed" class="black-footer" data-tap-
toggle="false">
        <div data-role="navbar" data-iconpos="top">
            <ul>
                <li><a href="#" data-role="button" data-icon="flat-menu" data-
theme="a" data-inline="true">Basket</a></li>
                <li><a href="#" data-role="button" data-icon="flat-heart" data-
theme="a" data-inline="true">Favourite</a></li>
                <li><a href="#" data-role="button" data-icon="flat-eye" data-
theme="a" data-inline="true">Scan</a></li>
                <li><a href="#" data-role="button" data-icon="flat-man" data-
theme="a" data-inline="true">Activity</a></li>
                <li><a href="#" data-role="button" data-icon="flat-bubble" data-
theme="a" data-inline="true">Friends</a></li>
            </ul>
        </div>
    </div>
    <div id="menuPage" data-role="page">
        <div data-role="header" data-position="fixed" data-tap-toggle="false">
            <h1>Menu</h1>
            <a data-icon="flat-settings" class="ui-btn-right" data-iconpos="notext"
href="#settingsPage">Settings</a>
            <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext"
href="#">Back</a>
        </div>
        <div data-role="content">
            <ul data-role="listview" data-icon="flat-plus" data-theme="b" data-count-theme="a"
data-filter="true" data-filter-theme="b" id="menuItems">
            </ul>
        </div>
        <div data-role="footer" data-position="fixed" class="black-footer" data-tap-
toggle="false">
            <div data-role="navbar" data-iconpos="top">
                <ul>
                    <li><a href="#" data-role="button" data-icon="flat-menu" data-
theme="a" data-inline="true">Basket</a></li>
                    <li><a href="#" data-role="button" data-icon="flat-heart" data-
theme="a" data-inline="true">Favourite</a></li>
                    <li><a href="#" data-role="button" data-icon="flat-eye" data-
theme="a" data-inline="true">Scan</a></li>
                    <li><a href="#" data-role="button" data-icon="flat-man" data-
theme="a" data-inline="true">Activity</a></li>
                    <li><a href="#" data-role="button" data-icon="flat-bubble" data-
theme="a" data-inline="true">Friends</a></li>
                </ul>
            </div>
        </div>
        <div id="locationsPage" data-role="page">
            <div data-role="header" data-position="fixed" data-tap-toggle="false">
                <h1>Locations</h1>
                <a data-icon="flat-settings" class="ui-btn-right" data-iconpos="notext"
href="#settingsPage">Settings</a>
                <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext"
href="#">Back</a>
            </div>
            <div data-role="navbar" class="bodybg">
                <ul>
                    <li><a href="#" id="showMap">Map</a></li>
                    <li><a href="#" id="showList">List</a></li>
                </ul>
            </div>
        </div>
    </div>

```

```

        </ul>
    </div>
</div>
<div data-role="content" id="map-canvas"></div>
<div data-role="content" id="map-list">
    <ul data-role="listview" data-filter="true" data-theme="b" data-count-theme="a"
data-filter-theme="b">
        </ul>
    </div>
    <div data-role="footer" data-position="fixed" class="black-footer" data-tap-
toggle="false">
        <div data-role="navbar" data-iconpos="top">
            <ul>
                <li><a href="#basketPage" data-role="button" data-icon="flat-menu" data-
theme="a" data-inline="true">Basket</a></li>
                <li><a href="#favouritePage" data-role="button" data-icon="flat-heart" data-
theme="a" data-inline="true">Favourite</a></li>
                <li><a href="#scanPage" data-role="button" data-icon="flat-eye" data-
theme="a" data-inline="true">Scan</a></li>
                <li><a href="#activityPage" data-role="button" data-icon="flat-man" data-
theme="a" data-inline="true">Activity</a></li>
                <li><a href="#friendsPage" data-role="button" data-icon="flat-bubble" data-
theme="a" data-inline="true">Friends</a></li>
            </ul>
        </div>
    </div>
</div>
<div id="offersPage" data-role="page">
    <div data-role="header" data-position="fixed" data-tap-toggle="false">
        <h1>Special Offers</h1>
        <a data-icon="flat-settings" class="ui-btn-right" data-iconpos="notext"
href="#settingsPage">Settings</a>
        <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext"
href="#mainPage">Back</a>
    </div>
    <div data-role="content" style="text-align:center;">
        
        <div style="width: 90%; margin: 0 auto;" id="offersText">
        </div>
    </div>
    <div data-role="footer" data-position="fixed" class="black-footer" data-tap-
toggle="false">
        <div data-role="navbar" data-iconpos="top">
            <ul>
                <li><a href="#basketPage" data-role="button" data-icon="flat-menu" data-
theme="a" data-inline="true">Basket</a></li>
                <li><a href="#favouritePage" data-role="button" data-icon="flat-heart" data-
theme="a" data-inline="true">Favourite</a></li>
                <li><a href="#scanPage" data-role="button" data-icon="flat-eye" data-
theme="a" data-inline="true">Scan</a></li>
                <li><a href="#activityPage" data-role="button" data-icon="flat-man" data-
theme="a" data-inline="true">Activity</a></li>
                <li><a href="#friendsPage" data-role="button" data-icon="flat-bubble" data-
theme="a" data-inline="true">Friends</a></li>
            </ul>
        </div>
    </div>
</div>
<div id="basketPage" data-role="page">
    <div data-role="header" data-position="fixed" data-tap-toggle="false">
        <h1>Basket</h1>
        <a data-icon="flat-settings" class="ui-btn-right" data-iconpos="notext"
href="#settingsPage">Settings</a>
        <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext"
href="#mainPage">Back</a>
    </div>
    <div data-role="content">
        <ul data-role="listview" data-icon="flat-plus" data-theme="b" data-count-theme="b"
data-filter="true" data-filter-theme="b" id="basketItems">
        </ul>
        <a href="#reviewPage" data-role="button" data-icon="flat-menu" data-theme="a"
id="shop"></a>
    </div>

```

```

        </div>
        <div data-role="footer" data-position="fixed" class="black-footer" data-tap-
toggle="false">
            <div data-role="navbar" data-iconpos="top">
                <ul>
                    <li><a href="#basketPage" data-role="button" data-icon="flat-menu" data-
theme="a" data-inline="true" class="ui-btn-active ui-state-persist">Basket</a></li>
                    <li><a href="#favouritePage" data-role="button" data-icon="flat-heart" data-
theme="a" data-inline="true">Favourite</a></li>
                    <li><a href="#scanPage" data-role="button" data-icon="flat-eye" data-
theme="a" data-inline="true">Scan</a></li>
                    <li><a href="#activityPage" data-role="button" data-icon="flat-man" data-
theme="a" data-inline="true">Activity</a></li>
                    <li><a href="#friendsPage" data-role="button" data-icon="flat-bubble" data-
theme="a" data-inline="true">Friends</a></li>
                </ul>
            </div>
        </div>
        <div id="reviewPage" data-role="page">
            <div data-role="header" data-position="fixed" data-tap-toggle="false">
                <h1>Review Order</h1>
                <a data-icon="flat-settings" class="ui-btn-right" data-iconpos="notext" href="#settingsPage">Settings</a>
                <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext" data-
rel="back">Back</a>
            </div>
            <div data-role="content">
                <div id="reviewVouchers" data-role="fieldcontain"></div>
                <center>
                    <p style="text-align:center;">Your order will be confirmed after you press the
following button. Please scan the resulting QR CODE at the cashier's desk.</p>
                </center>
                <a href="#" data-role="button" data-icon="flat-menu" data-theme="b" id="finishOrder"></a>
            </div>
            <div data-role="footer" data-position="fixed" class="black-footer" data-tap-
toggle="false">
                <div data-role="navbar" data-iconpos="top">
                    <ul>
                        <li><a href="#basketPage" data-role="button" data-icon="flat-menu" data-
theme="a" data-inline="true" class="ui-btn-active ui-state-persist">Basket</a></li>
                        <li><a href="#favouritePage" data-role="button" data-icon="flat-heart" data-
theme="a" data-inline="true">Favourite</a></li>
                        <li><a href="#scanPage" data-role="button" data-icon="flat-eye" data-
theme="a" data-inline="true">Scan</a></li>
                        <li><a href="#activityPage" data-role="button" data-icon="flat-man" data-
theme="a" data-inline="true">Activity</a></li>
                        <li><a href="#friendsPage" data-role="button" data-icon="flat-bubble" data-
theme="a" data-inline="true">Friends</a></li>
                    </ul>
                </div>
            </div>
            <div id="orderPage" data-role="page">
                <div data-role="header" data-position="fixed" data-tap-toggle="false">
                    <h1>Order Confirmed</h1>
                    <a data-icon="flat-settings" class="ui-btn-right" data-iconpos="notext" href="#settingsPage">Settings</a>
                    <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext" href="#mainPage">Back</a>
                </div>
                <div data-role="content">
                    <center>
                        <div id="qrOrder"></div>
                    </center>
                </div>
                <div data-role="footer" data-position="fixed" class="black-footer" data-tap-
toggle="false">
                    <div data-role="navbar" data-iconpos="top">
                        <ul>

```

```

        <li><a href="#basketPage" data-role="button" data-icon="flat-menu" data-
theme="a" data-inline="true" class="ui-btn-active ui-state-persist">Basket</a></li>
        <li><a href="#favouritePage" data-role="button" data-icon="flat-heart" data-
theme="a" data-inline="true">Favourite</a></li>
        <li><a href="#scanPage" data-role="button" data-icon="flat-eye" data-
theme="a" data-inline="true">Scan</a></li>
        <li><a href="#activityPage" data-role="button" data-icon="flat-man" data-
theme="a" data-inline="true">Activity</a></li>
        <li><a href="#friendsPage" data-role="button" data-icon="flat-bubble" data-
theme="a" data-inline="true">Friends</a></li>
    </ul>
</div>
</div>
<div id="favouritePage" data-role="page">
    <div data-role="header" data-position="fixed" data-tap-toggle="false">
        <h1>Favourite</h1>
        <a data-icon="flat-settings" class="ui-btn-right" data-iconpos="notext"
href="#settingsPage">Settings</a>
        <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext"
href="#mainPage">Back</a>
    </div>
    <div data-role="content">
        <ul data-role="listview" data-icon="flat-plus" data-theme="b" data-count-theme="a"
data-filter="true" data-filter-theme="b" id="favItems">
    </ul>
</div>
    <div data-role="footer" data-position="fixed" class="black-footer" data-tap-
toggle="false">
        <div data-role="navbar" data-iconpos="top">
            <ul>
                <li><a href="#basketPage" data-role="button" data-icon="flat-menu" data-
theme="a" data-inline="true">Basket</a></li>
                <li><a href="#favouritePage" data-role="button" data-icon="flat-heart" data-
theme="a" data-inline="true" class="ui-btn-active ui-state-persist">Favourite</a></li>
                <li><a href="#scanPage" data-role="button" data-icon="flat-eye" data-
theme="a" data-inline="true">Scan</a></li>
                <li><a href="#activityPage" data-role="button" data-icon="flat-man" data-
theme="a" data-inline="true">Activity</a></li>
                <li><a href="#friendsPage" data-role="button" data-icon="flat-bubble" data-
theme="a" data-inline="true">Friends</a></li>
            </ul>
        </div>
    </div>
</div>
<div id="activityPage" data-role="page">
    <div data-role="header" data-position="fixed" data-tap-toggle="false">
        <h1>Activity</h1>
        <a data-icon="flat-settings" class="ui-btn-right" data-iconpos="notext"
href="#settingsPage">Settings</a>
        <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext"
href="#mainPage">Back</a>
    </div>
    <div data-role="content">
        <ul data-role="listview" data-icon="flat-plus" data-theme="b" data-count-theme="a"
data-filter="true" data-filter-theme="b" id="activityItems">
    </ul>
</div>
    <div data-role="footer" data-position="fixed" class="black-footer" data-tap-
toggle="false">
        <div data-role="navbar" data-iconpos="top">
            <ul>
                <li><a href="#basketPage" data-role="button" data-icon="flat-menu" data-
theme="a" data-inline="true">Basket</a></li>
                <li><a href="#favouritePage" data-role="button" data-icon="flat-heart" data-
theme="a" data-inline="true">Favourite</a></li>
                <li><a href="#scanPage" data-role="button" data-icon="flat-eye" data-
theme="a" data-inline="true">Scan</a></li>
                <li><a href="#activityPage" data-role="button" data-icon="flat-man" data-
theme="a" data-inline="true" class="ui-btn-active ui-state-persist">Activity</a></li>
                <li><a href="#friendsPage" data-role="button" data-icon="flat-bubble" data-
theme="a" data-inline="true">Friends</a></li>
            </ul>
        </div>
    </div>
</div>

```

```

        </ul>
    </div>
</div>
<div id="scanPage" data-role="page">
    <div data-role="header" data-position="fixed" data-tap-toggle="false">
        <h1>Scan</h1>
        <a data-icon="flat-settings" class="ui-btn-right" data-iconpos="notext"
        href="#settingsPage">Settings</a>
        <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext"
        href="#mainPage">Back</a>
    </div>
    <div data-role="content">
        <div id="scanMsg">
        </div>
    </div>
    <div data-role="footer" data-position="fixed" class="black-footer" data-tap-
    toggle="false">
        <div data-role="navbar" data-iconpos="top">
            <ul>
                <li><a href="#basketPage" data-role="button" data-icon="flat-menu" data-
                theme="a" data-inline="true">Basket</a></li>
                <li><a href="#favouritePage" data-role="button" data-icon="flat-heart" data-
                theme="a" data-inline="true">Favourite</a></li>
                <li><a href="#scanPage" data-role="button" data-icon="flat-eye" data-
                theme="a" data-inline="true" class="ui-btn-active ui-state-persist">Scan</a></li>
                <li><a href="#activityPage" data-role="button" data-icon="flat-man" data-
                theme="a" data-inline="true">Activity</a></li>
                <li><a href="#friendsPage" data-role="button" data-icon="flat-bubble" data-
                theme="a" data-inline="true">Friends</a></li>
            </ul>
        </div>
    </div>
</div>
<div id="settingsPage" data-role="page">
    <div data-role="header" data-position="fixed" data-tap-toggle="false">
        <h1>Settings</h1>
        <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext"
        href="#mainPage">Back</a>
    <div data-role="navbar" class="bodybg2">
        <ul>
            <li><a href="#" id="showProfile">Profile</a></li>
            <li><a href="#" id="showVouchers">Vouchers</a></li>
        </ul>
    </div>
</div>
<div data-role="content">
    <div id="contentProfile">
        <form id="updateProfile">
            <div class="ui-grid-a">
                <div class="ui-block-a">
                    <div style="padding-top:10px;text-align:center;padding-right: 13px;">
id="userPicture">
                    </div>
                </div>
                <div class="ui-block-b">
                    <div data-role="fieldcontain" class="ui-hide-label" data-
type="horizontal">
                        <label for="username">Username:</label>
                        <input type="text" name="username" id="username"
placeholder="Username" />
                    </div>
                    <div data-role="fieldcontain" class="ui-hide-label">
                        <label for="upassword">Password:</label>
                        <input type="password" name="upassword" id="upassword"
placeholder="Password" />
                    </div>
                </div>
            </div>
            <div class="ui-grid-a">
                <div class="ui-block-a" style="padding-right:10px;">
                    <div data-role="fieldcontain" class="ui-hide-label">

```

```

        <label for="ufname">First Name:</label>
        <input type="text" name="ufname" id="ufname" placeholder="First
Name" />
    </div>
</div>
<div class="ui-block-b">
    <div data-role="fieldcontain" class="ui-hide-label">
        <label for="ulname">Last Name:</label>
        <input type="text" name="ulname" id="ulname" placeholder="Last
Name" />
    </div>
</div>
<div class="ui-grid-a">
    <div class="ui-block-a" style="padding-right:10px;">
        <div data-role="fieldcontain" class="ui-hide-label">
            <label for="uemail">Email:</label>
            <input type="email" name="uemail" id="uemail"
placeholder="Email" />
        </div>
    </div>
    <div class="ui-block-b">
        <div data-role="fieldcontain" class="ui-hide-label">
            <label for="uphone">Phone:</label>
            <input type="tel" name="uphone" id="uphone"
placeholder="Phone" />
        </div>
    </div>
</div>
<div data-role="fieldcontain" class="ui-hide-label gender-select">
    <select name="ugender" id="ugender" data-theme="a">
        <option value="male">Male</option>
        <option value="female">Female</option>
    </select>
</div>
<input type="submit" value="Update" id="updateProfileButton" data-theme="b">
    <a href="#" data-role="button" data-theme="a" id="logoutButton">Logout</a>
</form>
</div>
<div id="contentVouchers">
</div>
</div>
</div>
<div id="friendsPage" data-role="page">
    <div data-role="header" data-position="fixed" data-tap-toggle="false">
        <h1>Friends</h1>
        <a data-icon="plus" class="ui-btn-right" data-iconpos="notext" href="#"
id="addFriend">AddFriend</a>
        <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext"
href="#mainPage">Back</a>
    </div>
    <div data-role="content">
        <ul data-role="listview" data-icon="arrow-r" data-theme="b" data-count-theme="a"
data-filter="true" data-filter-theme="b" id="friendsItems">
        </ul>
    </div>
    <div data-role="footer" data-position="fixed" class="black-footer" data-tap-
toggle="false">
        <div data-role="navbar" data-iconpos="top">
            <ul>
                <li><a href="#basketPage" data-role="button" data-icon="flat-menu" data-
theme="a" data-inline="true">Basket</a></li>
                <li><a href="#favouritePage" data-role="button" data-icon="flat-heart" data-
theme="a" data-inline="true">Favourite</a></li>
                <li><a href="#scanPage" data-role="button" data-icon="flat-eye" data-
theme="a" data-inline="true">Scan</a></li>
                <li><a href="#activityPage" data-role="button" data-icon="flat-man" data-
theme="a" data-inline="true">Activity</a></li>
                <li><a href="#friendsPage" data-role="button" data-icon="flat-bubble" data-
theme="a" data-inline="true" class="ui-btn-active ui-state-persist">Friends</a></li>
            </ul>
        </div>
    </div>

```

```

        </div>
    </div>
<div id="friendActivityPage" data-role="page">
    <div data-role="header" data-position="fixed" data-tap-toggle="false">
        <h1>Activity</h1>
        <a data-icon="flat-settings" class="ui-btn-right" data-iconpos="notext" href="#settingsPage">Settings</a>
            <a data-icon="arrow-l" class="ui-btn-left" data-iconpos="notext" data-rel="back">Back</a>
        </div>
        <div data-role="content">
            <ul data-role="listview" data-icon="flat-plus" data-theme="b" data-count-theme="a" data-filter="true" data-filter-theme="b" id="friendActivityItems">
                </ul>
            </div>
            <div data-role="footer" data-position="fixed" class="black-footer" data-tap-toggle="false">
                <div data-role="navbar" data-iconpos="top">
                    <ul>
                        <li><a href="#basketPage" data-role="button" data-icon="flat-menu" data-theme="a" data-inline="true">Basket</a></li>
                        <li><a href="#favouritePage" data-role="button" data-icon="flat-heart" data-theme="a" data-inline="true">Favourite</a></li>
                        <li><a href="#scanPage" data-role="button" data-icon="flat-eye" data-theme="a" data-inline="true">Scan</a></li>
                        <li><a href="#activityPage" data-role="button" data-icon="flat-man" data-theme="a" data-inline="true">Activity</a></li>
                        <li><a href="#friendsPage" data-role="button" data-icon="flat-bubble" data-theme="a" data-inline="true" class="ui-btn-active ui-state-persist">Friends</a></li>
                    </ul>
                </div>
            </div>
        </div>
        <script type="text/javascript" src="cordova-2.5.0.js"></script>
        <script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=true"></script>
</body>
</html>
init.js
```

```

var debug = true;
var host = "http://10.10.10.47/coffee/";
function doConsole(message) {
    if(debug) console.log(message);
}
function toTitleCase(str) {
    return str.replace(/\w\S*/g, function(txt){return txt.charAt(0).toUpperCase() +
txt.substr(1).toLowerCase();});
}
function init() {
    document.addEventListener("deviceready", deviceReady, true);
    delete init;
}
function checkLogin() {
    if(!window.localStorage.getItem("userID")) {
        $.mobile.changePage("#loginPage");
    } else {
        $.mobile.changePage("#mainPage");
    };
}
function login() {
    $("#loginPage").on("pageshow", function(){
        $("#loginForm").on("submit", function() {
            var user = $("#loginForm #username").val();
            var pass = $("#loginForm #password").val();
            if(!user || !pass) {
                navigator.notification.alert("Enter username and password", function() {});
            }
        });
    });
}
```

```

        $("#" + loginForm + " #submitButton").parent().removeClass('ui-btn-active');
    } else {
        $.post(host+"dologin.php", {username:user, password:pass}, function(msg) {
            switch(msg) {
                case -1:
                    navigator.notification.alert("Could not connect to database", function()
                });
                    $("#" + submitButton).parent().removeClass('ui-btn-active');
                    break;
                case 0:
                    navigator.notification.alert("Wrong username or password", function()
                );
                    $("#" + submitButton).parent().removeClass('ui-btn-active');
                    break;
                default:
                    window.localStorage.setItem("userID", msg);
                    document.getElementById('loginForm').reset();
                    $.mobile.changePage("#mainPage");
                    break;
            };
        }, "json").error(function(){
            navigator.notification.alert("Could not connect to server", function() {});
            $("#" + submitButton).parent().removeClass('ui-btn-active');
        });
    };
    return false;
});
});

$("#loginPage").on("pagehide", function() {
    $("#" + loginForm).unbind("submit");
});

function onSuccessPhoto(imageData) {
    $("#rpicture").text(imageData);
}

function onFailPhoto(message) {
    navigator.notification.alert("Could not use camera", function() {});
}

function register() {
    $("#takepic").on("click", function() {
        navigator.camera.getPicture(onSuccessPhoto, onFailPhoto,
            {quality: 20, destinationType: Camera.DestinationType.DATA_URL});
    });
    $("#registerForm").on("submit", function() {
        var rusername = $("#registerForm #rusername").val();
        var rpassword = $("#registerForm #rpassword").val();
        var rfname = $("#registerForm #rfname").val();
        var rlname = $("#registerForm #rlname").val();
        var remail = $("#registerForm #remail").val();
        var rphone = $("#registerForm #rphone").val();
        var rgender = $("#registerForm #rgender").val();
        var rpicture = $("#registerForm #rpicture").val();
        if(!rusername || !rpassword || !rfname || !rlname || !remail || !rphone || !rgender || !rpicture) {
            navigator.notification.alert("Please complete all registration fields", function() {});
            $("#registerForm #registerButton").parent().removeClass('ui-btn-active');
        } else {
            $.post(host+"doregister.php", {username: rusername, password: rpassword, fname: rfname,
                lname: rlname, email: remail, phone: rphone, gender: rgender, picture:rpicture}, function(msg) {
                switch(msg) {
                    case -1:
                        navigator.notification.alert("Could not connect to database", function()
                });
                    $("#registerButton").parent().removeClass('ui-btn-active');
                    break;
                case 0:
                    break;
                default:
                    document.getElementById('registerForm').reset();
                    $.mobile.changePage("#loginPage");
                    break;
            });
        }
    });
}

```

```

        }, "json").error(function() {
            navigator.notification.alert("Could not connect to server", function() {});
            $("#registerButton").parent().removeClass('ui-btn-active');
        });
    };
    return false;
});
$("#addUser").on("pageshow", function() {
});
$("#addUser").on("pagehide", function() {
});
}
function forgotPassword() {
    $("#resetForm").on("submit", function(){
        var email = $("#resetForm #femail").val();
        if(!email) {
            navigator.notification.alert("Enter email address", function() {});
            $("#resetForm #forgotButton").parent().removeClass('ui-btn-active');
        } else {
            $.post(host+"doreset.php", {mail:email}, function(msg) {
                switch(msg) {
                    case -1:
                        navigator.notification.alert("Could not connect to database", function());
                    case 0:
                        navigator.notification.alert("Wrong email address", function() {});
                        $("#forgotButton").parent().removeClass('ui-btn-active');
                        break;
                    case 1:
                        navigator.notification.alert("A new password has been sent to "+email,
function() {});
                        document.getElementById('resetForm').reset();
                        $.mobile.changePage("#loginPage");
                        break;
                    default:
                        break;
                }
            }, "json").error(function() {
                navigator.notification.alert("Could not connect to server", function() {});
                $("#forgotButton").parent().removeClass('ui-btn-active');
            });
        };
        return false;
});
$("#forgotPassword").on("pageshow", function() {
});
$("#forgotPassword").on("pagehide", function() {
});
}

var infoWindow = new google.maps.InfoWindow();
function createInfoWindow(marker, popupContent) {
    google.maps.event.addListener(marker, 'click', function () {
        infoWindow.setContent(popupContent);
        infoWindow.open(map, this);
    });
}
function main() {
    $("#mainPage").on("pageshow", function() {
        doConsole("showing main page");
    });
}
function showLocation(latitude, longitude) {
    $("#map-list").hide();
    $("#showList").removeClass("ui-btn-active");
    $("#map-canvas").show();
    $("#showMap").addClass("ui-btn-active");
    map.setZoom(17);
    map.setCenter(new google.maps.LatLng(latitude, longitude));
}

```

```

function createMapAndList(latitude, longitude) {
    var mylocation = 'images/mylocation.png';
    var shoplocation = 'images/shoplocation.png';
    $.post(host+"locations.php", {latitude:latitude, longitude: longitude}, function(msg) {
        switch(msg) {
            case -1:
                navigator.notification.alert("Could not connect to database", function()
                {});
                break;
            case 0:
                navigator.notification.alert("Sorry, no available locations", function()
                {});
                break;
            default:
                $.each(msg, function(index, value) {
                    var location = new google.maps.Marker({
                        position: new google.maps.LatLng(value.latitude,
value.longitude),
                        map:map,
                        title: value.name,
                        icon: shoplocation,
                        animation: google.maps.Animation.DROP
                    });
                    var popupContent = "<strong>" + value.name + "</strong><br/>" +
value.address + "<br/><br/>" +
"<span style='font-size:11px;'>" +
"p: " + value.phone + "<br/>" +
"e: " + value.email +
"</span>";
                    createInfoWindow(location, popupContent);
                    $("#map-list ul").append("<li class='listLocation' data-icon='flat-
location'><a href='javascript:showLocation("+value.latitude+", "+value.longitude+");'>"+value.name +
"+<br/><span style='font-weight:normal !important; font-size: 12px;'>" + value.address + "<br/>p: " +
value.phone+ "&nbsp;&nbsp;&nbsp;e: " + value.email+ "</span><span class='ui-li-
count'>" +Number(value.distanta).toFixed(2)+" km</span></a></li>");
                });
                $("#map-list ul").listview("refresh");
                break;
        };
    }, "json").error(function() {
        navigator.notification.alert("Could not connect to server", function() {});
    });

    var mapOptions = {
        zoom: 12,
        center: new google.maps.LatLng(latitude, longitude),
        mapTypeId: google.maps.MapTypeId.ROADMAP,
        mapTypeControl:false,
        streetViewControl: false
    };
    map = new google.maps.Map(document.getElementById('map-canvas'), mapOptions);

    var defaultLocation = new google.maps.Marker({
        position: new google.maps.LatLng(latitude, longitude),
        map: map,
        title: "current location",
        icon: mylocation
    });
}
function onSuccessLocation(position) {
    createMapAndList(position.coords.latitude, position.coords.longitude);
}
function onFailLocation(error) {
    navigator.notification.alert("Could not get location", function() {});
}

function locations() {
    $("#locationsPage").on("pagebeforeshow", function() {
        $("#locationsPage #showMap").addClass("ui-btn-active");
        $("#map-canvas").empty();
        $("#map-list ul").empty();
        $("#map-list input").val("").trigger("keyup");
    });
}

```

```

        $("#" + mapCanvas).height($(window).height() - 149);
        $("#" + mapList).hide();
        $("#" + mapCanvas).show();
    });
    $("#locationsPage").on("pageshow", function() {
        navigator.geolocation.getCurrentPosition(onSuccessLocation, onFailLocation);
        $("#" + showMap).on("click", function() {
            $("#" + mapList).hide();
            $("#" + mapCanvas).show();
        });
        $("#" + showList).on("click", function() {
            $("#" + mapCanvas).hide();
            $("#" + mapList).show();
        });
    });
}
function addBasket(productID) {
    $.post(host+"dobasket.php", {user: window.localStorage.getItem("userID"), product: productID},
function(msg) {
    switch(msg) {
        case -1:
            navigator.notification.alert("Could not connect to database", function()
{}); break;
        case 1:
            navigator.notification.alert("Added to Basket", function() {}); break;
        default:
            break;
    };
}, "json").error(function() {
    navigator.notification.alert("Could not connect to server", function() {});
});});
}
function checkFavourites(productID) {
    var action;
    if($("#fav"+productID).hasClass("ui-btn-active")) {
        action = "remove";
    } else {
        action = "add";
    }
    $.post(host+"dofavourites.php", {user: window.localStorage.getItem("userID"), product:
productID, action: action}, function(msg) {
        switch(msg) {
            case -1:
                break;
            case 1:
                if(action == "add") {
                    $("#" + fav + productID).addClass("ui-btn-active");
                } else {
                    $("#" + fav + productID).removeClass("ui-btn-active");
                };
                break;
            default:
                break;
        };
    }, "json").error(function() {
        navigator.notification.alert("Could not connect to server", function() {});
});});
}
function menu() {
    $("#menuPage").on("pagebeforeshow", function() {
        $("#menuItems").empty();
        $("#menuPage input").val("").trigger("keyup");
    });
    $("#menuPage").on("pageshow", function() {
        if($("#menuItems li").length < 1) {
            $.mobile.loading('show');
        }
        $.post(host+"menu.php", {user: window.localStorage.getItem("userID")}, function(msg) {
            switch(msg) {
                case -1:

```

```

        navigator.notification.alert("Could not connect to database", function()
    });
    break;
    case 0:
        navigator.notification.alert("Sorry, no available items", function() {});
        break;
    default:
        $.mobile.loading('hide');
        $("#menuItems").empty();
        $.each(msg, function(index, value) {
            var favClass;
            if(value.favourite != null) {
                favClass=" class='ui-btn-active' ";
            } else {
                favClass="";
            }
            $("#menuItems").append("<li class='listLocation'><div class='ui-grid-a'><div class='ui-block-a' style='width:90px !important;'><div class='circle' style='background-image:url(data:image/jpg;base64,"+value.photo+");'></div></div><div class='ui-block-b'><span class='menuTitle'>" +value.name+ "</span><div data-role='controlgroup' data-type='horizontal' data-mini='true' style='margin-top:15px;'><a "+favClass+" id='fav"+value.id+"' href='#' onclick='javascript:checkFavourites('"+value.id+"');' data-icon='flat-heart' data-theme='d' data-iconpos='notext' data-role='button'>Add to Favourites</a><a href='javascript:addBasket('"+value.id+"');' data-icon='flat-plus' data-theme='d' data-iconpos='notext' data-role='button'>Add to Basket</a></div></div><span class='ui-li-count'>" +value.price+ " RON</span></li>").trigger("create");
        });
        $("#menuItems").listview("refresh");
        break;
    };
    $.mobile.loading('hide');
}, "json").error(function() {
    navigator.notification.alert("Could not connect to server", function() {});
});
});
}

function removeFavourite(productID) {
    $.post(host+"dofavourites.php", {user: window.localStorage.getItem("userID"), product: productID, action: "remove"}, function(msg) {
        switch(msg) {
            case -1:
                break;
            case 1:
                $("#favItems").find("li[id*='"+productID+"']").remove();
                $("#favPag"+productID).remove();
                $("#favItems").listview("refresh");
                break;
            default:
                break;
        };
        "json").error(function() {
            navigator.notification.alert("Could not connect to server", function() {});
        });
    });
}

function basketAddBasket(item) {
    $.post(host+"dobasket.php", {user: window.localStorage.getItem("userID"), product: item}, function(msg) {
        switch(msg) {
            case -1:
                navigator.notification.alert("Could not connect to database", function() {});
            break;
            case 1:
                $("#item_"+item+" .quantity").text(parseFloat($("#item_"+item+" .quantity").text()) + 1);
                $("#shop .ui-btn-text").text(parseFloat($("#shop .ui-btn-text").text().split(" ")[0]) + parseFloat($("#item_"+item+" .mPrice").text().split(" ")[0]) + " RON");
                break;
            default:
                break;
        };
    });
}

```

```

        };
    }, "json").error(function() {
        navigator.notification.alert("Could not connect to server", function() {});
    });
}
function basketRemoveBasket(item) {
    $.post(host+"removebasket.php", {user: window.localStorage.getItem("userID"), product: item},
function(msg) {
    switch(msg) {
        case -1:
            navigator.notification.alert("Could not connect to database", function()
{});
            break;
        case 1:
            $("#shop .ui-btn-text").text(parseFloat($("#shop .ui-btn-
text").text().split(" ")[0]) - parseFloat($("#item_"+item+".mPrice").text().split(" ")[0]) + " RON");
            if($("#shop .ui-btn-text").text() == "0 RON") {
                $("#shop").hide();
            }
            if(parseFloat($("#item_"+item+".quantity").text()) == 1) {
                $("#item_"+item).remove();
            } else {
                $("#item_"+item+".quantity").text(parseFloat($("#item_"+item+
".quantity").text()) - 1);
            }
            break;
        default:
            break;
    };
}, "json").error(function() {
    navigator.notification.alert("Could not connect to server", function() {});
});
}
function basket() {
    $("#basketPage").on("pagebeforeshow", function() {
        $("#basketItems").empty();
        $("#basketPage input").val("").trigger("keyup");
        $("#shop").hide();
    });
    $("#basketPage").on("pageshow", function() {
        if($("#basketItems li").length < 1) {
            $.mobile.loading('show');
        };
        $.post(host+"showbasket.php", {user: window.localStorage.getItem("userID")}, function(msg) {
            switch(msg) {
                case -1:
                    navigator.notification.alert("Could not connect to database", function()
{});
                    break;
                case 0:
                    navigator.notification.alert("Basket is empty", function() {});
                    break;
                default:
                    $.mobile.loading('hide');
                    $("#basketItems").empty();
                    var initVal = 0;
                    $.each(msg, function(index, value) {
                        $("#basketItems").append("<li class='listLocation' id='item_"+
value.id+"><div class='floatLeft miniText'><span class='quantity'>" + value.quantity + "</span> x</div><div class='floatLeft'>" + value.name + "<br/><span class='mtext mPrice'>" + value.price + " RON</span></div><div class='clear'></div><span class='ui-li-count'><div data-role='controlgroup' data-type='horizontal' data-mini='true' style='margin-top:-10px;' class='basketOperation'><a href='javascript:basketRemoveBasket(' + value.id + ')' data-icon='minus' data-theme='d' data-iconpos='notext' data-role='button'>Decrease</a><a href='javascript:basketAddBasket(' + value.id + ')' data-icon='plus' data-theme='d' data-iconpos='notext' data-role='button'>Increase</a></div></span></li>").trigger("create");
                        initVal += value.price * value.quantity;
                    });
                    $("#shop .ui-btn-text").text(initVal + " RON");
                    $("#basketItems").listview('refresh');
                }
            }
        });
    });
}

```

```

        $("#shop").show();
        break;
    };
    $.mobile.loading('hide');
}, "json").error(function() {
    navigator.notification.alert("Could not connect to server", function() {});
});
});

function favourites() {
    $("#favouritePage").on("pagebeforeshow", function() {
        $("#favItems").empty();
        $("#favouritePage input").val("").trigger("keyup");
    });
    $("#favouritePage").on("pageshow", function() {
        if($("#favItems li").length < 1) {
            $.mobile.loading('show');
        }
        $.post(host+"showfavourites.php", {user: window.localStorage.getItem("userID")},
function(msg) {
    switch(msg) {
        case -1:
            navigator.notification.alert("Could not connect to database", function() {});
            break;
        case 0:
            navigator.notification.alert("Sorry, no available items", function() {});
            break;
        default:
            $.mobile.loading('hide');
            $("#favItems").empty();
            $.each(msg, function(index, value) {
                $("#favItems").append("<li class='listLocation' id='favPag"+value.id+"'><div
class='ui-grid-a'><div class='ui-block-a' style='width:90px !important;'><div class='circle'
style='background-image:url(data:image/jpg;base64,"+value.photo+");'></div></div><div class='ui-
block-b'><span class='menuTitle'>"+value.name+"</span><div data-role='controlgroup' data-
type='horizontal' data-mini='true' style='margin-top:15px;'><a class='ui-btn-active' href='#!'
onclick='javascript:removeFavourite(\""+value.id+"\")'; data-icon='flat-heart' data-theme='d' data-
iconpos='notext' data-role='button'>Add to Favourites</a><a href='javascript:addBasket(\""+value.id
+"\")'; data-icon='flat-plus' data-theme='d' data-iconpos='notext' data-role='button'>Add to
Basket</a></div></div><div><span class='ui-li-count'>"+value.price+" RON</span></
li>").trigger("create");
            });
            $("#favItems").listview("refresh");
            break;
        };
        $.mobile.loading('hide');
    }, "json").error(function() {
        navigator.notification.alert("Could not connect to server", function() {});
    });
});
}

function activity() {
    $("#activityPage").on("pagebeforeshow", function() {
        $("#activityItems").empty();
        $("#activityPage input").val("").trigger("keyup");
    });
    $("#activityPage").on("pageshow", function() {
        if($("#activityItems li").length < 1) {
            $.mobile.loading('show');
        }
        $.post(host+"showactivity.php", {user: window.localStorage.getItem("userID")}, function(msg)
{
    switch(msg) {
        case -1:
            navigator.notification.alert("Could not connect to database", function() {});
            break;
        case 0:
            navigator.notification.alert("Sorry, no available activity", function() {});
            break;
        default:
            $.mobile.loading('hide');
            $("#activityItems").empty();
    }
});
}
}

```

```

        $.each(msg, function(index, value) {
            var da = value.date.split(" ");
            var db = da[0].split("-");
            $("#activityItems").append("<li class='listLocation'><span style='font-size: 11px;'>" + value.message + "</span><span class='ui-li-count' style='font-size: 10px;'>" + db[2] + "/" + db[1] + "</span></li>").trigger("create");
        });
        $("#activityItems").listview("refresh");
        break;
    };
    $.mobile.loading('hide');
}, "json").error(function() {
    navigator.notification.alert("Could not connect to server", function() {});
});
});

function scannerSuccess(result) {
    if(result.cancelled) {
        $.mobile.changePage("#mainPage");
    } else {
        if(result.format == "QR_CODE") {
            $.post(host+"checkvoucher.php", {user: window.localStorage.getItem("userID"), code: result.text}, function(msg){
                switch(msg) {
                    case -1:
                        navigator.notification.alert("Could not connect to database", function() {});
                        $.mobile.changePage("#mainPage");
                        break;
                    case 0:
                        navigator.notification.alert("Please scan a valid voucher", function() {});
                        $.mobile.changePage("#mainPage");
                        break;
                    case 2:
                        navigator.notification.alert("You already used this voucher", function() {});
                        $.mobile.changePage("#mainPage");
                        break;
                    case 3:
                        navigator.notification.alert("The voucher is already in your wallet", function() {});
                        $.mobile.changePage("#mainPage");
                        break;
                    default:
                        $("#scanMsg").html("<center><h1 style='color:#b2382d;'>Congratulations!</h1><p>Your received a</p><h1 style='color:#b2382d;'>" + msg + "%</h1><p>discount for your next order</p></center>");
                        break;
                }
            }), "json").error(function() {
                navigator.notification.alert("Could not connect to server", function() {});
            });
        } else {
            navigator.notification.alert("Please scan a QR Code", function() {});
            $.mobile.changePage("#mainPage");
        }
    }
}

function scannerFailure(message) {
    navigator.notification.alert("Could not scan code", function() {});
    $.mobile.changePage("#mainPage");
}

function scan() {
    $("#scanPage").on("pagebeforeshow", function() {
        $("#scanMsg").empty();
        window.plugins.barcodeScanner.scan(scannerSuccess, scannerFailure);
    });
}

function offers() {
    $("#offersPage").on("pagebeforeshow", function() {

```

```

$.post(host+"dooffer.php", function(msg){
    switch(msg) {
        case -1:
            navigator.notification.alert("Could not connect to database", function()
            {});
        break;
        default:
            $("#offersPage #offersText").text(msg.message);
        break;
    }
}, "json").error(function() {
    navigator.notification.alert("Could not connect to server", function() {});
});
});

function ios60bug() {
    $.ajaxPrefilter(function (options, originalOptions, jqXHR) {
        options.data = jQuery.param($.extend(originalOptions.data||{}, {
            timeStamp: new Date().getTime()
        }));
    });
}
function getUserProfile() {
    $.post(host+"userprofile.php", {user: window.localStorage.getItem("userID")}, function(msg) {
        switch(msg) {
            case -1:
                navigator.notification.alert("Could not connect to database", function()
                {});
            break;
            case 0:
                navigator.notification.alert("Could not find user in database", function()
                {});
            break;
            default:
                $("#userPicture").html("<div class='circleProfile' style='background-
image:url(data:image/jpg;base64,"+msg.photo+");'></div>");
                $("#username").val(msg.user);
                $("#ufname").val(msg.fname);
                $("#ulname").val(msg.lname);
                $("#uemail").val(msg.email);
                $("#uphone").val(msg.phone);
                $("#ugender").val(msg.gender);
                $("#ugender").selectmenu('refresh');
            break;
        }
    }, "json").error(function() {
        navigator.notification.alert("Could not connect to server", function() {});
    });
}
function getVouchers() {
    $.post(host+"vouchers.php", {user:window.localStorage.getItem("userID")}, function(msg) {
        switch(msg) {
            case -1:
                navigator.notification.alert("Could not connect to database", function()
                {});
            break;
            case 0:
                navigator.notification.alert("Voucher list is empty", function() {});
            break;
            default:
                var txt = "<div class='ui-grid-a'>";
                $.each(msg, function(index, value) {
                    var cod = value.code.substr(0,3) + "*****";
                    txt += "<div class='ui-block-a'>" + cod + "</div><div class='ui-block-b' style='text-align:right;'>discount: " + value.discount + "%</div><br/><br/>";
                });
                txt += "</div>";
                $("#contentVouchers").html(txt);
            break;
        }
    }, "json").error(function() {
        navigator.notification.alert("Could not connect to server", function() {});
    });
}

```

```

        });
    }

    function settings() {
        $("#showProfile").on("click", function() {
            $("#contentVouchers").hide();
            $("#contentProfile").show();
            document.getElementById('updateProfile').reset();
            getUserProfile();
        });

        $("#showVouchers").on("click", function() {
            $("#contentProfile").hide();
            $("#contentVouchers").show();
            getVouchers();
        });

        $("#updateProfile").on("submit", function() {
            var uusername = $("#updateProfile #uusername").val();
            var upassword = $("#updateProfile #upassword").val();
            var ufname = $("#updateProfile #ufname").val();
            var ulname = $("#updateProfile #ulname").val();
            var uemail = $("#updateProfile #uemail").val();
            var uphone = $("#updateProfile #uphone").val();
            var ugender = $("#updateProfile #ugender").val();
            $.post(host+"updateprofile.php", {id:window.localStorage.getItem("userID"), user: uusername,
            pass: upassword, fname: ufname, lname: ulname, email: uemail, phone: uphone, gender: ugender},
            function(msg){
                switch(msg) {
                    case -1:
                        navigator.notification.alert("Could not connect to database", function() {});
                        break;
                    case 1:
                        navigator.notification.alert("Profile updated", function() {});
                        if(upassword) {
                            window.localStorage.removeItem("userID");
                        };
                        checkLogin();
                        break;
                    default:
                        break;
                };
                }, "json").error(function() {
                    navigator.notification.alert("Could not connect to server", function() {});
                });
                return false;
            });
        $("#logoutButton").on("click", function() {
            window.localStorage.removeItem("userID");
            checkLogin();
        });
        $("#settingsPage").on("pagebeforeshow", function() {
            $("#settingsPage #showProfile").addClass("ui-btn-active");
            $("#contentProfile").show();
            document.getElementById('updateProfile').reset();
            getUserProfile();
            $("#contentVouchers").empty();
            $("#contentVouchers").hide();
        });
    }
}

function review() {
    var vchID = "";
    $("#reviewPage").on("pagebeforeshow", function() {
        $("#reviewVouchers").empty();
        $("#finishOrder .ui-btn-text").text($("#shop .ui-btn-text").text());
        $.post(host+"vouchers.php", {user:window.localStorage.getItem("userID")}, function(msg) {
            switch(msg) {
                case -1:
                    navigator.notification.alert("Could not connect to database", function() {});
                    break;
                case 0:
                    break;
            }
        });
    });
}

```

```

        default:
            var vch = "<span style=\"font-weight:bold;color:#b2382d;\">Use a voucher:</span><br/><fieldset data-role=\"controlgroup\" data-mini=\"false\">";
            $.each(msg, function(index, value) {
                vch += "<input type=\"radio\" name=\"vouchers\" id=\"" + value.id + "\" data-theme=\"b\" value=\"" + value.discount + "\"/>";
                vch += "<label for=\"" + value.id + "\">" + value.discount + "% discount</label>";
            });
            vch += "</fieldset>";
            $("#reviewVouchers").html(vch);
            $("#reviewVouchers input").checkboxradio();
            $("#reviewPage").trigger('pagecreate');
            $("#reviewVouchers input[type='radio']").bind("change", function(event, ui){
                var total = parseFloat($("#shop .ui-btn-text").text().split(" ")[0]);
                total -= total * parseFloat($(this).val()/100);
                $("#finishOrder .ui-btn-text").text(total + " RON");
                vchID = $(this).attr('id');
            });
            break;
        };
    },
    "json").error(function() {
        navigator.notification.alert("Could not connect to server", function() {});
    });
});
$("#finishOrder").on("click", function() {
    $.post(host+"order.php", {user:window.localStorage.getItem("userID"), voucher: vchID},
function(msg){
    switch(msg) {
        case -1:
            navigator.notification.alert("Could not connect to database", function() {});
            break;
        default:
            $("#qrOrder").empty();
            $("#qrOrder").html("<p>Your order has been confirmed. Scan the code at the cashier's desk.</p><img src=\"https://chart.googleapis.com/chart?chs=280x280&cht=qr&chl=" + msg + "&choe=UTF-8\" />");
            $.mobile.changePage("#orderPage");
            break;
    };
}, "json").error(function() {
    navigator.notification.alert("Could not connect to server", function() {});
});
return false;
});
}

function confirmRequests(status, requestID, requestorID) {
    $.post(host+"confirmrequest.php", {user:window.localStorage.getItem("userID"), request: requestID, requestor: requestorID, status: status}, function(msg){
        switch(msg) {
            case -1:
                navigator.notification.alert("Could not connect to database", function() {});
                break;
            case 0:
                break;
            case 1:
                $("#friendsItems").empty();
                $("#friendsPage input").val("").trigger("keyup");
                $("#friendsPage").trigger("pageshow");
                break;
            default:
                break;
        };
    },
    "json").error(function() {
        navigator.notification.alert("Could not connect to server", function() {});
    });
}
function getFriendRequest() {
    $.post(host+"getrequest.php", {user:window.localStorage.getItem("userID")}, function(msg){
        switch(msg) {
            case -1:

```

```

        navigator.notification.alert("Could not connect to database", function() {});
        break;
    case 0:
        break;
    default:
        $.each(msg, function(index,value) {
            navigator.notification.confirm(toTitleCase(value.name) + " wants to
be your friend", function(btnIndex) {confirmRequests(btnIndex,value.id, value.request)}, "Friend
Request", 'Accept,Reject');
        });
        break;
    };
}, "json").error(function() {
    navigator.notification.alert("Could not connect to server", function() {});
});
}
function showFriendDetails(id, name) {
    $("#friendActivityItems").empty();
    $("#friendActivityPage input").val("").trigger("keyup");
    $("#friendActivityPage h1").text(toTitleCase(name)+"'s activity");
    $.mobile.changePage("#friendActivityPage");
    $.mobile.loading('show');
    $.post(host+"showactivity.php", {user: id}, function(msg) {
        switch(msg) {
            case -1:
                navigator.notification.alert("Could not connect to database", function() {});
                break;
            case 0:
                navigator.notification.alert("Sorry, no available activity", function() {});
                break;
            default:
                $.mobile.loading('hide');
                $.each(msg, function(index, value) {
                    var da = value.date.split(" ");
                    var db = da[0].split("-");
                    $("#friendActivityItems").append("<li class='listLocation'><span style='font-
size:11px;'>" + value.message + "</span><span class='ui-li-count' style='font-size:
10px;'>" + db[2] + "/" + db[1] + "</span></li>").trigger("create");
                });
                $("#friendActivityItems").listview("refresh");
                break;
            };
            $.mobile.loading('hide');
        }, "json").error(function() {
            navigator.notification.alert("Could not connect to server", function() {});
        });
    }
}
function friends() {
    $("#addFriend").on("click", function() {
        window.plugins.Prompt.show(
            "Enter friend's email address",
            function (userText) {
                $.post(host+"invite.php", {user:window.localStorage.getItem("userID"),
email:userText}, function(msg){
                    switch(msg) {
                        case -1:
                            navigator.notification.alert("Could not connect to database",
function() {});
                            break;
                        case 0:
                            navigator.notification.alert("Wrong email address", function()
{});
                            break;
                        case 1:
                            navigator.notification.alert("Your friend request was sent to
"+userText, function() {});
                            break;
                        case 2:
                            navigator.notification.alert("The user rejected your friend
request", function() {});
                            break;
                        case 3:
                            break;
                    }
                });
            }
        );
    });
}

```

```

                navigator.notification.alert("Your friend request is already
pending", function() {});
                break;
            case 4:
                navigator.notification.alert("User is already in your friend
list", function() {});
                break;
            default:
                break;
        };
    },
    "json").error(function() {
        navigator.notification.alert("Could not connect to server",
function() {});
    });
},
function () {
},
"Invite",
"Cancel"
);
return false;
});
$("#friendsPage").on("pagebeforeshow", function() {
getFriendRequest();
$("#friendsItems").empty();
$("#friendsPage input").val("").trigger("keyup");
});
$("#friendsPage").on("pageshow", function() {
$.mobile.loading('show');
$.post(host+"getfriends.php", {user:window.localStorage.getItem("userID")}, function(msg) {
switch(msg) {
case -1:
    navigator.notification.alert("Could not connect to database", function()
{});
    break;
case 0:
    navigator.notification.alert("You have no friends. Try to invite some",
function() {});
    break;
default:
    $.each(msg, function(index, value) {
        $("#friendsItems").append("<li class='listLocation'><a href=
\"javascript:showFriendDetails(\""+value.id+"\", '"+value.fullname+"\";\\")><div class='ui-grid-a'><div
class='ui-block-a' style='width:80px !important;'><div class='circle' style='background-
image:url(data:image/jpg;base64,"+value.photo+");\'></div></div><div class='ui-block-b'><span
class='menuTitleBig'>"+toTitleCase(value.fullname)+"</span></div></div></a></
li>").trigger("create");
    });
    $("#friendsItems").listview('refresh');
    break;
};
$.mobile.loading('hide');
}, "json").error(function() {
    navigator.notification.alert("Could not connect to server", function() {}));
});
});
}
function deviceReady() {
ios60bug();
checkLogin();
login();
register();
forgotPassword();
locations();
menu();
favourites();
basket();
activity();
scan();
offers();
settings();
review();
}

```

```

main();
friends();
}

```

## *functions.php*

```

<?php
$mode = "mongo";
$host = "localhost";
$user = "root";
$pass = "root";
$db = "coffee";
$resetpass = substr(md5(rand()), 0, 5);
function doConsole($message) {
    global $mode;
    $log = date("d.m.y - H:i:s");
    file_put_contents("output.txt", $log.":: [ ".$mode." ] ".$message."\n", FILE_APPEND);
}
function mongoEcho($output) {
    echo "\"".$output."\"";
}
function dbConnect() {
    global $host, $user, $pass, $db, $connection, $mode;
    switch($mode) {
        case "mysql":
            $connection = new mysqli($host, $user, $pass, $db);
            if ($connection->connect_error) {
                die('-1');
                doConsole("DATABASE: database connection error");
            }
            break;
        case "mongo":
            $conn = new MongoClient();
            if(!$conn) {
                die('-1');
                doConsole("DATABASE: database connection error");
            } else {
                $connection = $conn->selectDB($db);
            }
            break;
        default:
            break;
    }
}
function returnUser($username, $password) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $result = $connection->query("SELECT `id` FROM `users` WHERE `user`='".$username."' AND `pass`='".$password."' LIMIT 1");
            if($result->num_rows) {
                $output = $result->fetch_array(1);
                echo $output['id'];
            } else {
                echo '0';
            }
            break;
        case "mongo":
            $search = array('user' => $username, 'pass' => $password);
            $query = $connection->users->find($search)->limit(1);
            $output = $query->getNext();
            if($output) {
                mongoEcho($output['_id']);
            } else {
                echo '0';
            }
            break;
        default:
            break;
    }
}

```

```

}

function resetPassword($email) {
    global $connection, $mode, $resetpass;
    switch($mode) {
        case "mysql":
            $result = $connection->query("SELECT `id` FROM `users` WHERE `email`='".$email."' LIMIT 1");
            if($result->num_rows) {
                $output = $result->fetch_array(1);
                $temppass = $resetpass;
                $res = $connection->query("UPDATE `users` SET `pass`='".md5($temppass)."' WHERE `id`='".$output['id']."' ");
                if($res) {
                    echo '1';
                    file_put_contents("newpass.txt", $temppass);
                } else {
                    echo '-1';
                }
            } else {
                echo '0';
            }
            break;
        case "mongo":
            $search = array('email' => $email);
            $return = array('_id' => 1);
            $query = $connection->users->find($search, $return)->limit(1);
            $em = $query->getNext();
            if($em) {
                $temppass = $resetpass;
                $search = array('_id' => $em["_id"]);
                $newPass = array('$set' => array('pass' => md5($temppass)));
                $nq = $connection->users->update($search, $newPass);
                if($nq) {
                    echo '1';
                    file_put_contents("newpass.txt", $temppass);
                } else {
                    echo '-1';
                }
            } else {
                echo '0';
            }
            break;
        default:
            break;
    }
}

function registerUser($username,$password,$fname,$lname,$email,$phone,$gender,$picture) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("INSERT INTO `users`(`user`, `pass`, `fname`, `lname`, `email`, `phone`, `gender`, `photo`) VALUES ('".$username."', '".$md5($password)."', '".$fname."', '".$lname."', '".$email."', '".$phone."', '".$gender."', '".$picture."')");
            if($query) {
                echo '1';
            } else {
                echo '-1';
            }
            break;
        case "mongo":
            $newUser = array('user' => $username, 'pass' => md5($password), 'fname' => $fname, 'lname' => $lname, 'email' => $email, 'phone' => $phone, 'gender' => $gender, 'photo' => $picture);
            $query = $connection->users->insert($newUser);
            if($query) {
                echo '1';
            } else {
                echo '-1';
            }
            break;
        default:
    }
}

```

```

        break;
    }
}

function returnLocations($latitude, $longitude) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT `name`, `latitude`, `longitude`,
`address`, `phone`, `email`, (6371 * acos(cos(radians($latitude)) * cos(radians(latitude)) *
cos(radians($longitude) - radians(longitude)) + sin(radians($latitude)) * sin(radians(latitude)))) AS distanta FROM `locations` ORDER BY distanta ASC");
            if($query) {
                if($query->num_rows) {
                    $locations = array();
                    while($location = $query->fetch_array()) {
                        array_push($locations, $location);
                    };
                    echo json_encode($locations);
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
            break;
        case "mongo":
            $query = $connection->locations->find();
            if($query) {
                if($query->count()) {
                    $locations = array();
                    while($location = $query->getNext()) {
                        $location["distanta"] = 6371 *
acos(cos(deg2rad($latitude)) * cos(deg2rad($location["latitude"])) * cos(deg2rad($longitude) -
deg2rad($location["longitude"]))) + sin(deg2rad($latitude)) * sin(deg2rad($location["latitude"])));
                        array_push($locations, $location);
                    };
                    echo json_encode($locations);
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
            break;
        default:
            break;
    }
}

function returnMenu($username) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT MENU.id as id, MENU.name as name,
MENU.price as price, MENU.photo as photo, FAV.product as favourite FROM `menu` MENU LEFT JOIN
`favourites` FAV ON MENU.id = FAV.product AND FAV.user = '".$username."'";
            if($query) {
                if($query->num_rows) {
                    $items = array();
                    while($item = $query->fetch_array()) {
                        array_push($items, $item);
                    };
                    echo json_encode($items);
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
            break;
        case "mongo":
            $query = $connection->menu->find();
            if($query) {

```

```

        if($query->count()) {
            $items = array();
            while($item = $query->getNext()) {
                $item["id"] = $item["_id"]."";
                $filter = array('user' => $username, 'product' =>
$item["_id"]."");
                $q = $connection->favourites->find($filter);
                if($q->getNext()) {
                    $item["favourite"] = 1;
                } else {
                    $item["favourite"] = null;
                }
                array_push($items, $item);
            };
            echo json_encode($items);
        } else {
            echo '0';
        }
    } else {
        echo '-1';
    }
    break;
default:
    break;
}
}

function checkFavourites($username, $product, $action) {
    global $connection, $mode;
    switch ($mode) {
        case "mysql":
            if($action == "add") {
                $query = $connection->query("INSERT INTO `favourites` (`user`,
`product`) VALUES ('".$username."', '".$product."')");
                if($query) {
                    echo '1';
                    $n = $connection->query("SELECT `name` FROM `menu` WHERE
id='".$product."'");
                    $msg = "Added ".$n->fetch_array()['name']."' to Favourites";
                    $q = $connection->query("INSERT INTO `history` (`user`,
`message`) VALUES ('".$username."', '".$msg."')");
                } else {
                    echo '-1';
                }
            } else {
                $query = $connection->query("DELETE FROM `favourites` WHERE `user`='".
$username."' AND `product`='".$product."'");
                if($query) {
                    echo '1';
                    $n = $connection->query("SELECT `name` FROM `menu` WHERE
id='".$product."'");
                    $msg = "Removed ".$n->fetch_array()['name']."' from
Favourites";
                    $q = $connection->query("INSERT INTO `history` (`user`,
`message`) VALUES ('".$username."', '".$msg."')");
                } else {
                    echo '-1';
                }
            }
            break;
        case "mongo":
            if($action == "add") {
                $data = array('user' => $username, 'product' => $product);
                $query = $connection->favourites->insert($data);
                if($query) {
                    echo '1';
                    $nd = array('_id' => new MongoDBId($product));
                    $fi = array('name' => 1);
                    $q = $connection->menu->find($nd,$fi);
                    $msg = "Added ".$q->getNext()['name']."' to Favourites";
                    $vdata = array('user' => $username, 'message' => $msg, 'date'
=> new MongoDate());
                    $v = $connection->history->insert($vdata);
                }
            }
    }
}

```

```

        } else {
            echo '-1';
        }
    } else {
        $data = array('user' => $username, 'product' => $product);
        $query = $connection->favourites->remove($data);
        if($query) {
            echo '1';
            $nd = array('_id' => new MongoDB\BSON\ObjectID($product));
            $fi = array('name' => 1);
            $q = $connection->menu->find($nd,$fi);
            $msg = "Removed ".$q->getNext()["name"]." from Favourites";
            $vdata = array('user' => $username, 'message' => $msg, 'date'
=> new MongoDB\BSON\DateTime());
            $v = $connection->history-
>insert($vdata);
        } else {
            echo '-1';
        }
    }
    break;
default:
    break;
}
}

function showUserFavourites($username) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT MENU.id, MENU.name, MENU.price,
MENU.photo FROM `favourites` FAV LEFT JOIN `menu` MENU ON MENU.id = FAV.product AND FAV.user = ''".
$username."'");
            if($query) {
                if($query->num_rows) {
                    $items = array();
                    while($item = $query->fetch_array()) {
                        if($item["id"] != null) {
                            array_push($items, $item);
                        };
                    };
                    if(count($items) > 0) {
                        echo json_encode($items);
                    } else {
                        echo '0';
                    }
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
        }
        break;
    case "mongo":
        $filter = array('user' => $username);
        $query = $connection->favourites->find($filter);
        if($query) {
            if($query->count()) {
                $items = array();
                while($item = $query->getNext()) {
                    $nf = array('_id' => new MongoDB\BSON\ObjectID($item["product"]));
                    $np = array('_id' => 1, 'name' => 1, 'price' => 1,
'photo' => 1);
                    $n = $connection->menu->find($nf, $np);
                    $newData = $n->getNext();
                    $newData["id"] = $newData["_id"]."";
                    array_push($items, $newData);
                };
                if(count($items) > 0) {
                    echo json_encode($items);
                } else {
                    echo '0';
                }
            }
        }
    }
}

```

```

        }
    } else {
        echo '0';
    }
} else {
    echo '-1';
}
break;
default:
    break;
}

}

function showUserActivity($username) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT * FROM `history` WHERE user='".$username."' ORDER BY `date` DESC");
            if($query) {
                if($query->num_rows) {
                    $items = array();
                    while($item = $query->fetch_array()) {
                        array_push($items, $item);
                    };
                    if(count($items) > 0) {
                        echo json_encode($items);
                    } else {
                        echo '0';
                    }
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
        }
        break;
    case "mongo":
        $filter = array('user' => $username);
        $query = $connection->history->find($filter)->sort(array('date' => -1));
        if($query) {
            if($query->count()) {
                $items = array();
                while($item = $query->getNext()) {
                    $item["date"] = date('Y-m-d H:i:s', $item["date"]->sec);
                    array_push($items, $item);
                };
                echo json_encode($items);
            } else {
                echo '0';
            }
        } else {
            echo '-1';
        }
    }
    break;
default:
    break;
}

}

function showUserBasket($username) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT MENU.id, MENU.name, MENU.price,
MENU.photo, BASKET.quantity FROM `menu` MENU LEFT JOIN `basket` BASKET ON MENU.id = BASKET.product
AND BASKET.user = '".$username."'");
            if($query) {
                if($query->num_rows) {
                    $items = array();
                    while($item = $query->fetch_array()) {
                        if($item["quantity"] != null) {
                            array_push($items, $item);
                        }
                    };
                    echo json_encode($items);
                } else {
                    echo '0';
                }
            }
        }
        break;
    case "mongo":
        $filter = array('user' => $username);
        $query = $connection->basket->find($filter)->sort(array('date' => -1));
        if($query) {
            if($query->count()) {
                $items = array();
                while($item = $query->getNext()) {
                    $item["date"] = date('Y-m-d H:i:s', $item["date"]->sec);
                    array_push($items, $item);
                };
                echo json_encode($items);
            } else {
                echo '0';
            }
        }
    }
    break;
default:
    break;
}

}

```

```

        };
    };
    if(count($items) > 0) {
        echo json_encode($items);
    } else {
        echo '0';
    }
} else {
    echo '0';
}
} else {
    echo '-1';
}
break;
case "mongo":
    $data = array('user' => $username);
    $query = $connection->basket->find($data);
    if($query) {
        if($query->count()) {
            $items = array();
            while($item = $query->getNext()) {
                $filter = array('_id' => new
MongoId($item["product"]));
                $cols = array('_id' => 1, 'name' => 1, 'price' => 1,
'photo' => 1);
                $q = $connection->menu->find($filter, $cols);
                $newData = $q->getNext();
                $newData["id"] = $newData["_id"]."";
                $newData["quantity"] = $item["quantity"];
                array_push($items,
$newData);
            };
            echo json_encode($items);
        } else {
            echo '0';
        }
    } else {
        echo '-1';
    }
}
break;
default:
    break;
}
}
function addBasket($username, $product) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT * FROM `basket` WHERE `user`='".
$username."' AND `product`='".$product."'");
            if($query) {
                if($query->num_rows) {
                    $q = $connection->query("UPDATE `basket` SET
`quantity`=`quantity`+1 WHERE `user`='".$username."' AND `product`='".$product."'");
                    doConsole("BASKET: update product quantity");
                } else {
                    $q = $connection->query("INSERT INTO `basket` (`user`,
`product`) VALUES('".$username."', '".$product."')");
                }
                echo '1';
            } else {
                echo '-1';
            }
        }
        break;
    case "mongo":
        $data = array('user' => $username, 'product' => $product);
        $query = $connection->basket->find($data);
        if($query) {
            if($query->count()) {
                $udata = array('$inc' => array('quantity' => 1));
                $q = $connection->basket->update($data, $udata);
            } else {

```

```

        $bdata = array('user' => $username, 'product' => $product,
'quantity' => 1);
                $q = $connection->basket->insert($bdata);
            }
            echo '1';
        } else {
            echo '-1';
        }
        break;
    default:
        break;
}
}

function removeItemBasket($username, $product) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT * FROM `basket` WHERE `user`='".$username."' AND `product`='".$product."' LIMIT 1");
            if($query) {
                if($query->num_rows) {
                    if($query->fetch_array()['quantity'] <= 1) {
                        $q = $connection->query("DELETE FROM `basket` WHERE `user`='".$username."' AND `product`='".$product."'");
                    } else {
                        $q = $connection->query("UPDATE `basket` SET `quantity`=`quantity`-1 WHERE `user`='".$username."' AND `product`='".$product."'");
                    }
                    echo '1';
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
            break;
        case "mongo":
            $data = array('user' => $username, 'product' => $product);
            $query = $connection->basket->find($data)->limit(1);
            if($query) {
                if($query->count()) {
                    if($query->getNext()['quantity'] <= 1) {
                        $q = $connection->basket->remove($data);
                    } else {
                        $udata = array('$inc' => array('quantity' => -1));
                        $q = $connection->basket->update($data, $udata);
                    }
                    echo '1';
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
            break;
        default:
            break;
    }
}

function doVoucher($username, $voucher) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT * FROM `voucher` WHERE `code` = '".$voucher."' LIMIT 1");
            if($query) {
                if($query->num_rows) {
                    $vch = $query->fetch_array();
                    $q = $connection->query("SELECT * FROM `users_voucher` WHERE `user` = '".$username."' AND `voucher` = '".$vch["id"]."')");
                    if($q->num_rows) {
                        if($q->fetch_array()['used'] != 0) {

```

```

                echo '2';
            } else {
                echo '3';
                doConsole("SCAN: voucher already in wallet");
            };
        } else {
            $n = $connection->query("INSERT INTO `users_voucher`(`user`, `voucher`) VALUES ('".$username."', '".$vch["id"]."')");
            echo $vch["discount"];
            $msg = "Scanned a voucher with a discount of ".
$vch["discount"]."%";
            $v = $connection->query("INSERT INTO `history` (`user`, `message`) VALUES ('".$username."', '".$msg."')");
        }
    } else {
        echo '0';
    }
} else {
    echo '-1';
}
break;
case "mongo":
    $data = array('code' => $voucher);
    $query = $connection->voucher->find($data)->limit(1);
    if($query) {
        if($query->count()) {
            $vch = $query->getNext();
            $filter = array('user' => $username, 'voucher' =>
$vch["_id"]."");
            $q = $connection->users_voucher->find($filter);
            if($q->count()) {
                if($q->getNext()["used"] != 0) {
                    echo '2';
                } else {
                    echo '3';
                };
            } else {
                $ndata = array('user' => $username, 'voucher' =>
$vch["_id"]."", 'used' => 0);
                $n = $connection->users_voucher->insert($ndata);
                echo $vch["discount"];
                $msg = "Scanned a voucher with a discount of ".
$vch["discount"]."%";
                $hdata = array('user' => $username, 'message' => $msg,
'date' => new MongoDate());
                $v = $connection->history->insert($hdata);
            }
        } else {
            echo '0';
        }
    } else {
        echo '-1';
    }
    break;
default:
    break;
}
}

function getOffers() {
    global $connection, $mode;
    switch($mode) {
        case "mysql";
            $query = $connection->query("SELECT * FROM `offer` ORDER BY RAND() LIMIT
0,1");
            if($query) {
                echo json_encode($query->fetch_array());
            } else {
                echo '-1';
            }
        break;
        case "mongo":

```

```

        $nr = rand(0,2);
        $query = $connection->offer->find()->skip($nr)->limit(1);
        if($query) {
            echo json_encode($query->getNext());
        } else {
            echo '-1';
        }
        break;
    default:
        break;
    }
}

function getUserProfile($username) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT * FROM `users` WHERE id='".
$username."'");
            if($query) {
                if($query->num_rows) {
                    $user = $query->fetch_array();
                    echo json_encode($user);
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
            break;
        case "mongo":
            $data = array('_id' => new MongoDB\BSON\ObjectID($username));
            $query = $connection->users->find($data);
            if($query) {
                if($query->count()) {
                    $user = $query->getNext();
                    echo json_encode($user);
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
            break;
        default:
            break;
    }
}

function doUpdateProfile($id, $username, $password, $fname, $lname, $email, $phone, $gender) {
    global $connection, $mode;
    switch ($mode) {
        case "mysql":
            if($password) {
                $query = $connection->query("UPDATE `users` SET `user` = '"..
$username."', `pass` = '".md5($password)."', `fname` = '".$fname."', `lname` = '".$lname."', `email` =
".$email."', `phone` = '".$phone."', `gender` = '".$gender."' WHERE `id` = '".$id."' ");
                if($query) {
                    echo '1';
                    $msg = "Updated profile (password)";
                    $v = $connection->query("INSERT INTO `history` (`user`, `message`) VALUES ('".$id."', '".$msg."')");
                } else {
                    echo '-1';
                };
            } else {
                $query = $connection->query("UPDATE `users` SET `user` = '"..
$username."', `fname` = '".$fname."', `lname` = '".$lname."', `email` = '".$email."', `phone` = '".$.
$phone."', `gender` = '".$gender."' WHERE `id` = '".$id."' ");
                if($query) {
                    echo '1';
                    $msg = "Updated profile";
                    $v = $connection->query("INSERT INTO `history` (`user`, `message`) VALUES ('".$id."', '".$msg."')");
                }
            }
    }
}

```

```

        } else {
            echo '-1';
        };
    };
    break;
case "mongo":
    $filter = array('_id' => new MongoDB\BSON\ObjectID($id));
    if($password) {
        $newdata = array('$set' => array('user' => $username, 'pass' =>
md5($password), 'fname' => $fname, 'lname' => $lname, 'email' => $email, 'phone' => $phone, 'gender' => $gender));
        $query = $connection->users->update($filter, $newdata);
        if($query) {
            echo '1';
            $msg = "Updated profile (password)";
            $mh = array('user' => $id, 'message' => $msg, 'date' => new
MongoDate());
            $q = $connection->history-
>insert($mh);
        } else {
            echo '-1';
        }
    } else {
        $newdata = array('$set' => array('user' => $username, 'fname' =>
$fname, 'lname' => $lname, 'email' => $email, 'phone' => $phone, 'gender' => $gender));
        $query = $connection->users->update($filter, $newdata);
        if($query) {
            echo '1';
            $msg = "Updated profile";
            $mh = array('user' => $id, 'message' => $msg, 'date' => new
MongoDate());
            $q = $connection->history->insert($mh);
        } else {
            echo '-1';
        }
    }
    break;
default:
    break;
}
}

function getVouchers($username) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT voucher.code, voucher.discount,
voucher.id, user.used FROM `users_voucher` user LEFT JOIN `voucher` voucher ON user.voucher =
voucher.id AND user.user = '". $username . "'");
            if($query) {
                if($query->num_rows) {
                    $items = array();
                    while($item=$query->fetch_array()) {
                        if($item["discount"] != null && $item["used"] == '0') {
                            array_push($items, $item);
                        };
                    };
                    if(count($items) > 0) {
                        echo json_encode($items);
                    } else {
                        echo '0';
                    }
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
        }
        break;
    case "mongo":
        $data = array('user' => $username, 'used' => 0);
        $query = $connection->users_voucher->find($data);
        if($query) {

```

```

        if($query->count()) {
            $items = array();
            while($item = $query->getNext()) {
                $filter = array('_id' => new
MongoId($item["voucher"]));
                $q = $connection->voucher->find($filter);
                $newData = $q->getNext();
                $newData["id"] = $newData["_id"]."";
                array_push($items, $newData);
            }
            echo json_encode($items);
        } else {
            echo '0';
        }
    } else {
        echo '-1';
    }
}
break;
default:
    break;
}

}

function makeOrder($username, $voucher) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT * FROM `basket` WHERE `user` = '".
$username."'");
            if($query) {
                $date = new DateTime();
                $order = $date->getTimestamp()."-".$username;
                while($item=$query->fetch_array()) {
                    $q = $connection->query("INSERT INTO `basket_order` (`user`,
`product`, `quantity`, `code`) VALUES('".$item["user"]."', '".$item["product"]."',
".$item["quantity"]."', '".$order."')");
                }
                $r = $connection->query("DELETE FROM `basket` WHERE `user`='".
$username."'");
                if($voucher) {
                    $v = $connection->query("UPDATE `users_voucher` SET `used` =
'1', `code` = '".$order."' WHERE `user` = '".$username."' AND `voucher` =
'".$voucher."'");
                };
                $msg = "Bought some hot coffee!";
                $v = $connection->query("INSERT INTO `history` (`user`, `message`)
VALUES ('".$username."', '".$msg."')");
                echo json_encode($order);
            } else {
                echo '-1';
            };
            break;
        case "mongo":
            $filter = array("user" => $username);
            $query = $connection->basket->find($filter);
            if($query) {
                $date = new DateTime();
                $order = $date->getTimestamp()."-".$username;
                while($item=$query->getNext()) {
                    $newdata = array('user' => $item["user"], 'product' =>
$item["product"], 'quantity' => $item["quantity"], 'code' => $order);
                    $q = $connection->basket_order->insert($newdata);
                };
                $r = $connection->basket->remove($filter);
                if($voucher) {
                    $vf = array('user' => $username, 'voucher' => $voucher);
                    $nd = array('$set' => array('used' => 1, 'code' => $order));
                    $v = $connection->users_voucher->update($vf, $nd);
                };
                $msg = "Bought some hot coffee!";
                $hdata = array('user' => $username, 'message' => $msg, 'date' => new
MongoDate());
                $h = $connection->history->insert($hdata);
                echo json_encode($order);
            }
    }
}

```

```

        } else {
            echo '-1';
        }
        break;
    default:
        break;
    }
}

function inviteUser($username, $email) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT `id` FROM `users` WHERE `email` = '".$email."'");
            if($query) {
                if($query->num_rows) {
                    $id = $query->fetch_array()["id"];
                    $c = $connection->query("SELECT * FROM `friends` WHERE `request` = '".$username."' AND `accept` = '".$id."'");
                    if($c->num_rows) {
                        $status = $c->fetch_array()["status"];
                        switch($status) {
                            case '0':
                                echo '2';
                                break;
                            case '1':
                                echo '3';
                                break;
                            case '2':
                                echo '4';
                                break;
                            default:
                                break;
                        }
                    } else {
                        $q = $connection->query("INSERT INTO `friends`(`request`, `accept`, `status`) VALUES('".$username."','".$id."','".$1"')");
                        echo '1';
                    };
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
            break;
        case "mongo":
            $filter = array('email' => $email);
            $data = array('_id' => 1);
            $query = $connection->users->find($filter,$data);
            if($query) {
                if($query->count()) {
                    $id = $query->getNext()["_id"]."";
                    $filter = array('request' => $username, 'accept' => $id);
                    $q = $connection->friends->find($filter);
                    if($q->count()) {
                        $status = $q->getNext()["status"];
                        switch($status) {
                            case '0':
                                echo '2';
                                break;
                            case '1':
                                echo '3';
                                break;
                            case '2':
                                echo '4';
                                break;
                            default:
                                break;
                        }
                    } else {
                }
            }
        }
    }
}

```

```

        $newdata = array('request' => $username, 'accept' =>
$id, 'status' => '1');
                $v = $connection->friends->insert($newdata);
                echo '1';
            }
        } else {
            echo '0';
        }
    } else {
        echo '-1';
    }
}
break;
default:
    break;
}
}

function getFriendRequest($username) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT friends.id, friends.accept,
friends.status, friends.request, CONCAT(users.fname,\\" \",users.lname) as name FROM `friends`-
friends LEFT JOIN `users` users ON users.id = friends.request");
            if($query) {
                if($query->num_rows) {
                    $items = array();
                    while($item=$query->fetch_array()) {
                        if($item["accept"] == $username && $item["status"] ==
'1') {
                            array_push($items, $item);
                        }
                    }
                    if(count($items) > 0) {
                        echo json_encode($items);
                    } else {
                        echo '0';
                    }
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
        }
        break;
    case "mongo":
        $filter = array('accept' => $username, 'status' => '1');
        $query = $connection->friends->find($filter);
        if($query) {
            if($query->count()) {
                $items = array();
                while($item = $query->getNext()) {
                    $nf = array('_id' => new MongoDB\BSON\ObjectID($item["request"]));
                    $nd = array('fname' => 1, 'lname' => 1);
                    $q = $connection->users->find($nf, $nd);
                    $item["id"] = $item["_id"]."";
                    $userdata = $q->getNext();
                    $item["name"] = $userdata["fname"]." ".
$userdata["lname"];
                    array_push($items, $item);
                };
                echo json_encode($items);
            } else {
                echo '0';
            }
        } else {
            echo '-1';
        }
    }
}
break;
default:
    break;
}
}
}

```

```

function doConfirmRequest($username, $requestID, $requestor, $status) {
    global $connection, $mode;
    $newstatus = "";
    if($status == '2') {
        $newstatus = "0";
    } else {
        $newstatus = "2";
    }
    switch($mode) {
        case "mysql":
            $query = $connection->query("UPDATE `friends` SET `status` = '".$newstatus."' WHERE `id` = '".$requestID."'");
            if($query) {
                if($newstatus == '2' && $username != null && $requestID != null && $requestor != null) {
                    $q = $connection->query("INSERT INTO `friends` (`request`, `accept`, `status`) VALUES('".$username."','".$requestor."', '2')");
                    echo '1';
                    $msg = "Has a new friend :";
                    $v = $connection->query("INSERT INTO `history` (`user`, `message`) VALUES ('".$username."', '".$msg."')");
                    $u = $connection->query("INSERT INTO `history` (`user`, `message`) VALUES ('".$requestor."', '".$msg."')");
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
            break;
        case "mongo":
            $filter = array('_id' => new MongoDB\BSON\ObjectID($requestID));
            $data = array('$set' => array('status' => $newstatus));
            $query = $connection->friends->update($filter, $data);
            if($query) {
                if($newstatus == '2' && $username != null && $requestID != null && $requestor != null) {
                    $nd = array('request' => $username, 'accept' => $requestor, 'status' => '2');
                    $q = $connection->friends->insert($nd);
                    echo '1';
                    $msg = "Has a new friend :";
                    $v = $connection->history->insert(array('user' => $username, 'message' => $msg, 'date' => new MongoDB\BSON\DateTime()));
                    $u = $connection->history->insert(array('user' => $requestor, 'message' => $msg, 'date' => new MongoDB\BSON\DateTime()));
                } else {
                    echo '0';
                }
            } else {
                echo '-1';
            }
            break;
        default:
            break;
    }
}
function getUserFriends($username) {
    global $connection, $mode;
    switch($mode) {
        case "mysql":
            $query = $connection->query("SELECT users.id, CONCAT(users.fname, ' ', users.lname) as fullname, users.photo FROM users LEFT JOIN friends ON friends.accept = users.id WHERE friends.request = '".$username."' AND friends.status = '2'");
            if($query) {
                if($query->num_rows) {
                    $items = array();
                    while($item=$query->fetch_array()) {
                        if($item["id"] != null) {
                            array_push($items, $item);
                        }
                    }
                }
            }
}

```

```

                if(count($items) > 0) {
                    echo json_encode($items);
                } else {
                    echo '0';
                }
            } else {
                echo '0';
            }
        } else {
            echo '-1';
        }
    }
    break;
}
case "mongo":
    $filter = array('request' => $username, 'status' => '2');
    $query = $connection->friends->find($filter);
    if($query) {
        if($query->count()) {
            $items = array();
            while($item = $query->getNext()) {
                $nf = array('_id' => new MongoDB($item["accept"]));
                $nd = array('_id' => 1, 'fname' => 1, 'lname' => 1,
                'photo' => 1);
                $q = $connection->users->find($nf, $nd);
                $newData = $q->getNext();
                $newData["id"] = $newData["_id"]."";
                $newData["fullname"] = $newData["fname"]." ".
                $newData["lname"];
                array_push($items, $newData);
            };
            echo json_encode($items);
        } else {
            echo '0';
        }
    } else {
        echo '-1';
    }
}
break;
default:
    break;
}
}
function order($username, $voucher) {
    global $connection;
    dbConnect();
    makeOrder($username, $voucher);
}
function doLogin($username, $password) {
    global $connection;
    dbConnect();
    returnUser($username, $password);
}

function doReset($email) {
    global $connection;
    dbConnect();
    resetPassword($email);
}
function doRegister($username,$password,$fname,$lname,$email,$phone,$gender,$picture) {
    global $connection;
    dbConnect();
    registerUser($username,$password,$fname,$lname,$email,$phone,$gender,$picture);
}
function getLocations($latitude, $longitude) {
    global $connection;
    dbConnect();
    returnLocations($latitude, $longitude);
}
function getMenu($username) {
    global $connection;
    dbConnect();
    returnMenu($username);
}
function doFavourites($username, $product, $action) {

```

```

        global $connection;
        dbConnect();
        checkFavourites($username, $product, $action);
    }
    function showFavourites($username) {
        global $connection;
        dbConnect();
        showUserFavourites($username);
    }
    function showActivity($username) {
        global $connection;
        dbConnect();
        showUserActivity($username);
    }
    function doBasket($username, $product) {
        global $connection;
        dbConnect();
        addBasket($username, $product);
    }
    function removeBasket($username, $product) {
        global $connection;
        dbConnect();
        removeItemBasket($username, $product);
    }
    function showBasket($username) {
        global $connection;
        dbConnect();
        showUserBasket($username);
    }
    function checkVoucher($username, $voucher) {
        global $connection;
        dbConnect();
        doVoucher($username, $voucher);
    }
    function doOffer() {
        global $connection;
        dbConnect();
        getOffers();
    }
    function userProfile($username) {
        global $connection;
        dbConnect();
        getUserProfile($username);
    }
    function updateProfile($id, $username, $password, $fname, $lname, $email, $phone, $gender) {
        global $connection;
        dbConnect();
        doUpdateProfile($id, $username, $password, $fname, $lname, $email, $phone, $gender);
    }
    function vouchers($username) {
        global $connection;
        dbConnect();
        getVouchers($username);
    }
    function invite($username, $email) {
        global $connection;
        dbConnect();
        inviteUser($username, $email);
    }
    function getRequest($username) {
        global $connection;
        dbConnect();
        getFriendRequest($username);
    }
    function confirmRequest($username, $requestID, $requestor, $status) {
        global $connection;
        dbConnect();
        doConfirmRequest($username, $requestID, $requestor, $status);
    }
    function getFriends($username) {
        global $connection;
        dbConnect();

```

```

        getUserFriends($username);
    }
    function sanitize($str) {
        $str = @strip_tags($str);
        $str = @stripslashes($str);
        $str = mysql_real_escape_string($str);
        return $str;
    }
?>
```

### ***checkvoucher.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
$voucher = sanitize($_POST['code']);
checkVoucher($username, $voucher);
?>
```

### ***dobasket.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
$product = sanitize($_POST['product']);
doBasket($username, $product);
?>
```

### ***dofavourites.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
$product = sanitize($_POST['product']);
$action = sanitize($_POST['action']);
doFavourites($username, $product, $action);
?>
```

### ***menu.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
getMenu($username);
?>
```

### ***order.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
$voucher = sanitize($_POST['voucher']);
order($username, $voucher);
?>
```

### ***showactivity.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
showActivity($username);
?>
```

### ***doregister.php***

```

<?php
include("functions.php");
$username = sanitize($_POST['username']);
$password = sanitize($_POST['password']);
$fname = sanitize($_POST['fname']);
$lname = sanitize($_POST['lname']);
$email = sanitize($_POST['email']);
$phone = sanitize($_POST['phone']);
$gender = sanitize($_POST['gender']);
$picture = $_POST['picture'];
$body = "
username: ".$username."\n
password: ".$password."\n
fname: ".$fname."\n
lname: ".$lname."\n
email: ".$email."\n
phone: ".$phone."\n
gender: ".$gender."\n
picture: ".$picture."\n";
doRegister($username,$password,$fname,$lname,$email,$phone,$gender,$picture);
?>

```

### ***doreset.php***

```

<?php
include("functions.php");
$email = sanitize($_POST['mail']);
doReset($email);
?>

```

### ***invite.php***

```

<?php
include("functions.php");
$username = sanitize($_POST['user']);
$email = sanitize($_POST['email']);
invite($username, $email);
?>

```

### ***confirmrequest.php***

```

<?php
include("functions.php");
$username = sanitize($_POST['user']);
$requestID = sanitize($_POST['request']);
$requestor = sanitize($_POST['requestor']);
$status = sanitize($_POST['status']);
confirmRequest($username, $requestID, $requestor, $status);
?>

```

### ***showbasket.php***

```

<?php
include("functions.php");
$username = sanitize($_POST['user']);
showBasket($username);
?>

```

### ***dologin.php***

```

<?php
include("functions.php");
$username = sanitize($_POST['username']);
$password = md5(sanitize($_POST['password']));
doLogin($username, $password);
?>

```

### ***removebasket.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
$product = sanitize($_POST['product']);
removeBasket($username, $product);
?>
```

### ***dooffer.php***

```
<?php
include("functions.php");
doOffer();
?>
```

### ***vouchers.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
vouchers($username);
?>
```

### ***showfavourites.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
showFavourites($username);
?>
```

### ***locations.php***

```
<?php
include("functions.php");
$latitude = sanitize($_POST['latitude']);
$longitude = sanitize($_POST['longitude']);
getLocations($latitude, $longitude);
?>
```

### ***getrequest.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
getRequest($username);
?>
```

### ***userprofile.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
userProfile($username);
?>
```

### ***getfriends.php***

```
<?php
include("functions.php");
$username = sanitize($_POST['user']);
getFriends($username);
?>
```

### ***updateprofile.php***

```
<?php
include("functions.php");
$id = sanitize($_POST['id']);
$username = sanitize($_POST['user']);
$password = sanitize($_POST['pass']);
$fname = sanitize($_POST['fname']);
$lname = sanitize($_POST['lname']);
$email = sanitize($_POST['email']);
$phone = sanitize($_POST['phone']);
$gender = sanitize($_POST['gender']);
updateProfile($id, $username, $password, $fname, $lname, $email, $phone, $gender);
?>
```