UNIVERSITATEA POLITEHNICA BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

watchr
Sesizarea problemelor comunitare de pe dispozitive mobile

Coordonator ştiinţific:
Conf. Dr. Ing. Alexandru Boicea

Absolvent:
Tudor-Ștefan Drăgan

BUCUREŞTI

2014

Abstract

O comunitate reprezintă o unitate socială care împărtășește valori și spații comune. În aceste comunități se pot întâmpla lucruri ce pot perturba buna stare a oamenilor din societatea modernă, spre exemplu accidente, incendii etc. În era internetului și a rețelelor sociale, aceste evenimente neplăcute ar trebui anunțate cât mai repede pentru ca societatea să conștientizeze problemele și pentru a încerca creșterea nivelului de responsabilitate a fiecărui individ din acea societate.

Soluția propusă în acest document este o aplicație care centralizează diferite evenimente (accident, gunoi aruncat, gropi în asfalt, drum blocat) și apoi transmite toate informațiile la utilizatorii serviciului și la instituțiile care se ocupă cu rezolvarea acestor probleme. Cetățenii pot adăuga diferite comentarii și diferite elemente cum ar fi poze, gradul de urgentă, detalii importante, etc. Întregul sistem este o rețea socială creată pentru folosul comunității.

Cuvinte cheie: aplicație, client, server, REST API, php, Objective-C, MySQL, croud-source, social network, OAuth 2.

Cuprins

Abstract 2
Introducere 7
1.1 Scurtă Descriere 7
1.1.1 Motivație 7
1.1.2 Caracteristici 8
1.2 Componentele rețelei watchr 8
1.2.1 Core Service 9
1.2.2 Aplicația Client 9
1.2.3 Interacțiunea Aplicația Client – Core Service 10
1.2.4 Instalarea 10
1.3 Analiza rețelelor sociale asemănătoare 11
Core Service 12
2.1 Tehnologii 12
2.1.1 Serverul Apache 12
2.1.2 Baza de date MySQL 13
2.1.3 Framework-ul PHP Laravel 4 13
2.2 Modelul bazei de date 14
2.2.1 Gruparea user profile 14
2.2.2 Gruparea *watchr* event 17
2.2.3 Mecanismul de autentificare 19
2.3 Implementarea API-ului REST *watchr* 20
2.3.1 Rutele in Laravel 21
2.3.2 Modele Eloquent pentru reprezentarea datelor 22
2.3.3 Validarea parametrilor din request 23
2.3.4 Query pentru găsirea celor mai apropiate evenimente 24
Securitate 26
3.1 Introducere 26
3.1.1 Roluri 27
3.1.2 Fluxul protocolului 27
3.1.3 Acordul de autorizatie 28
3.1.4 Folosirea parolei proprietarului de resursa 28
3.1.5 Tokenul de acces 28
3.1.6 Tokenul de refresh 29
3.2 Obținerea autorizației prin username si parola 29
3.2.1 Cererea tokenului de acces 30
3.2.2 Răspunsul tokenului de acces 31
Aplicația Client 32
4.1 Prezentare de ansamblu 32
4.1.1 Dezvoltarea pe terminale mobile 32
4.1.2 Tehnologii folosite 33
4.2 Designul aplicației *watchr* 34
4.2.1 Mock-up 34
4.2.2 Welcome screen-ul 34
4.2.3 Watchr Feed 35
4.2.4 Adăugarea unui eveniment 37
4.2.5 Meniul 38
4.2.6 Detaliile unui eveniment *watchr* 39
4.3 Implementarea aplicației iOS *watchr* 39
4.3.1 Design pattern MVC 39
4.3.2 Biblioteci open-source 40
4.3.3 Autentificarea/Inregistrarea utilizatorului 42
4.3.4 Exemplu de cerere 43
4.3.5 Afișarea unor cantități mari de adnotări pe harta 43

Evaluare și testare 49
Îmbunătățiri și planuri viitoare 50
6.1 Îmbunătățiri la Core Service 50
6.1.1 Dezvoltarea unui server de Push-Notification 50
6.1.2 Suport pentru fișiere video 50
6.1.3 Posibilitatea utilizatorilor sa urmărească un eveniment 50
6.1.4 Implementarea unor grupuri speciale 51
6.1.5 Sistem de promovare si recompensare 51
6.2 Îmbunătățiri *watchr* app 51
6.2.1 Suport pentru redarea și înregistrarea fișierelor video 51
6.2.2 Geo-fencing 51
6.2.3 Tratarea notificărilor 51
6.2.4 Relații de prietenie si suport pentru comment-uri 51
Concluzie 52

Lista de figuri

Fig. 1 Modelul de bază al serviciului watchr 8
Fig. 2 Screenshot cu Welcome Screen 10
Fig. 3 Arhitectura generală a runtime-ului serverului Apache 13
Fig. 4 Modelul bazei de date watchr 14
Fig. 5 Gruparea user profile 15
Fig. 6 Gruparea watchr event 17
Fig. 7 Gruparea mecanismului de autentificare 20
Fig. 8 Fluxul protocolului 27
Fig. 9 Fluxul autentificării cu username si parolă 29
Fig. 10 Mock-up aplicație 34
Fig. 11 Welcome Screen cu view-ul de "login" și "register" 35
Fig. 12 Ecranul de Feed 35
Fig. 13 Bara de navigare 36
Fig. 14 Exemplu de celulă din lista de evenimente 36
Fig. 15 Harta evenimentelor 37
Fig. 16 Ecranul de introducere eveniment nou 38
Fig. 17 Meniul 38
Fig. 18 Ecranul de afișare a detaliilor evenimentului 39
Fig. 19 Structura MVC 40
Fig. 20 Exemplu de Infinite Scroll 41
Fig. 21 Exemplu de MRProgress HUD 41
Fig. 22 Screenshot cu memory usage-ul aplicației 49

Notații si abrevieri

REST - Representational state transfer

API - Application programming interface

SGBD - Sistem de gestiune a bazelor de date

ORM - Object-relational mapping

URI - Uniform resource identifier

JSON - JavaScript Object Notation

IDE - Integrated development environment

Capitolul I

Introducere

1.1 Scurtă Descriere

1.1.1 Motivație

De multe ori suntem nevoiți să asistăm la anumite evenimente neplăcute fără voia noastră, alte ori problemele din cadrul comunității din care facem parte pur și simplu rămân nerezolvate o perioada îndelungata de timp datorită tendinței de a ignora asemenea mici inconveniente. Totuși, chiar dacă par lucruri minore, odată cu creșterea numărului lor, afectează cu buna stare a cetățenilor. Acele lucruri „minore" nu mai pot fi ignorate. Este datoria cetățeanului de a ajuta în curățarea și înfrumusețarea zonelor din care face parte.

Până acum, toate sesizările trebuiau făcute telefonic la entități specifice fiecărei probleme. Tot procesul este unul anevoios. Să presupunem că are loc un accident, iar un individ este martor la eveniment. Acesta trebuie să sune la numărul de urgentă, să aștepte aproximativ doua minute până când persoana de la centrală îi preia cererea. Apoi trebuie să piardă în jur de 5 minute pentru a descrie locul accidentului, poziția fixă și să ofere alte detalii importante. Această sesizare nu este transparentă. Lumea din jur nu va ști nimic despre acel accident, care îi implică în mod direct. Să presupunem ca se blochează drumul, ei nu au de unde să cunoască acest lucru decât atunci când vor observa că așteaptă deblocarea unei cozi interminabile.

Această lucrare încearcă să ofere o soluție la această problemă. Statisticile arată că aproape 60% din adulții care locuiesc in Statele Unite dețin un tip de telefon inteligent (smartphone), iar numărul crește de la an la an [1]. Se preconizează că până în anul 2020, 90% dintre cetățeni vor deține un smartphone. Acest lucru va rezulta în folosirea soluției propuse în această lucrare de cați mai mulți oameni. Sesizările problemelor comunitare de pe telefonul mobil vor avea un impact benefic in viața cetățenilor.

Această rețea socială încearcă automatizarea raportărilor de evenimente importante și distribuirea lor la entitățile care iau masuri în privința lor. Fiecare individ are un cuvânt de spus asupra unui anumit eveniment. Unele probleme sunt mai importante ca altele și este necesară categorisirea acestora.

Proiectul dorește să îmbunătățească condiția de trai a oamenilor din cadrul comunității și de a ajuta autoritățile să-și desfășoare activitățile întru-un mod cât mai ordonat și ușor posibil.

1.1.2 Caracteristici

watchr este o rețea socială creată special pentru gestionarea problemelor comunitare. Arhitectura serviciului este una asemănătoare cu rețeaua *Twitter.* Un astfel de serviciu are următoarele caracteristici:

- **Securitate** – În primul rând, tot serviciul este securizat point-to-point. Toate metodele din cadrul web service-ului sunt protejate de protocolul OAuth 2.0.
- **Modularitate** – Fiecare componentă din web service este concepută în așa fel încât să nu existe dependințe inutile intre module. De exemplu, sistemul de rating este detașat de cel de postare de evenimente.
- **Fiabilitate** – Serverul verifică fiecare intrare pentru toate metodele pentru a păstra puritatea bazei de date și pentru a valida contractul dintre client si server.
- **Tehnologii de ultimă generație** – Serviciul folosește ultima versiune de Apache Server 2.4.4. Web service-ul este scris folosind ultima versiune a limbajului PHP 5.4, iar aplicația mobila este special concepută pentru cel mai nou sistem de operare dezvoltat de Apple, iOS 7.1.
- **Interactivitate** – Aplicația trebuie să respecte cele mai stricte standarde de *User Interface Guidelines* impuse de Apple. Aceste *guidelines* ajută la creșterea numărului de utilizatori care vor folosi cu plăcere serviciul.
- **Scalare** – Rețeaua trebuie să suporte un număr foarte mare de requesturi concomitente. Sunt prezente diferite tehnici de creștere a performantei și a timpului de răspuns dintre client și server.
- **Crowd Sourced Content** – Rețeaua și informațiile pe care le deține aceasta cresc prin implicarea utilizatorilor, prin sesizarea diferitelor evenimente și prin folosirea aplicației pentru a spori acel sentiment de responsabilitate fața de comunitate.

1.2 Componentele rețelei watchr

Serviciul de sesizări *watchr* este compus din două subsisteme separate:

1. Core Service
2. Aplicația Client

Subsistemul **Core Service** reprezintă un web server ce îi permite utilizatorului să raporteze evenimentele în real-time, upload de fișiere, gestiune de utilizatori și multe altele. Toate evenimentele, userii și informațiile aferente sunt păstrate în baza de date.

Aplicația Client este un app care rulează nativ pe sistemul de operare iOS 7.1. Aplicația nu folosește wrappere cum ar fi PhoneGap sau Sencha Touch. Codul scris în Objective-C care rulează pe terminalul mobil, anume iPhone, este nativ sistemului de operare. Am decis să iau această cale datorită performanțelor și optimizărilor SDK-ului dezvoltat de Apple.

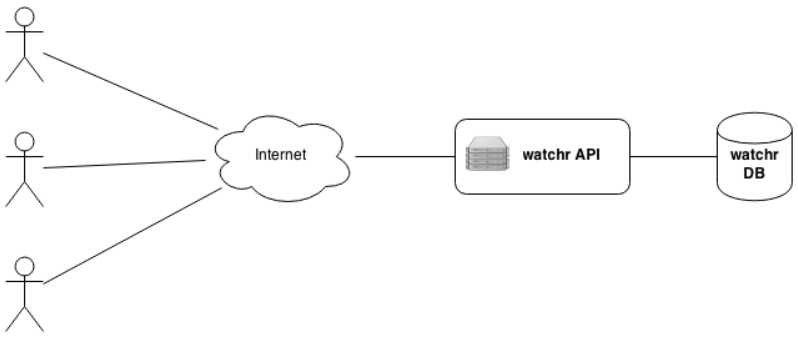

Fig. 1 Modelul de bază al serviciului watchr

1.2.1 Core Service

Pentru implementarea serverului am ales soluția open-source Apache. Apache este foarte răspândit și este folosit în aproape 55% din paginile web [2]. Apache este menținut de o comunitate deschisă de dezvoltatori sub licența Apache Software Foundation.

Comunicarea clientului cu serverul se face prin HTTP, suport pentru HTTPS este opțional. Web service-ul are expus un API pentru a permite interacțiunea dintre componentele software. API-ul este unul REST și vom intra în detaliile acestuia în următoarele capitole. Modulele de securitate sunt implementate în cadrul web service-ului și sunt concepute special pentru o astfel de rețea socială.

Serverul va reține toate informațiile necesare, precum atașamente si poze. Serverul trebuie să asigure că doar indivizii autorizați au acces la aceste fișiere, care șa rândul lor sunt legate de conturile utilizatorilor. Acest lucru este implementat prin atribuirea fiecărui utilizator al rețelei cu un nume de cont si parola.

Accesul la API se face folosind protocolul OAuth 2.0 prin HTTP/HTTPS. Protocolul este bazat pe *access tokens, sesiuni, granturi și scopeuri.* Voi prezenta protocolul, odată cu implementarea, în Capitolul III al acestei lucrări.

Sistemul de gestiune a bazelor de date folosit pentru stocarea și manipularea informațiilor serviciului *watchr* este MySQL. Este al doilea cel mai popular SGBD open-source la ora actuală [3], fiind o componentă cheie a stivei LAMP (Linux, Apache, MySQL, PHP). MySQL este folosit foarte des împreună cu limbajul de programare PHP și de aceea am considerat că frameworkul Laravel este perfect pentru implementarea aplicației *watchr*.

1.2.2 Aplicația Client

App-ul **watchr** este o aplicație special concepută pentru interacțiunea cu web service-ul descris mai sus. Este o aplicație nativă scrisă în Objective-C folosind SDK-ul furnizat de Apple Inc [4]. Aplicația are un design plăcut si atrăgător, ceea ce aduce un plus de funcționalitate serviciului. Ea trimite cereri către server pentru a îndeplini funcții precum postarea unui eveniment sau listarea evenimentelor active din jurul utilizatorului.

App-ul folosește funcțiile de geo locație ale telefonului pentru cereri. Acesta poate cere evenimentele care se petrec la o rază de 1 km pana la 20 km. Poate filtra căutările de evenimente după diferite criterii, cum ar fi distanța, data publicării și rating-ul.

Folosind acest software mobil, user-ul are posibilitatea de a adăuga evenimente care se întâmplă la fața locului, de a oferi informații concrete și de a se implica în problemele comunitare care sunt în desfășurare. Aplicația este o interfață pentru întregul serviciu și apelează metodele din API-ul REST.

Funcțiile de geo locație sunt un element cheie în funcționarea întregului serviciu. Fiecare eveniment trebuie să aibă o poziție fixă pe harta, date de coordonatele GPS. Sesizările sunt făcute în real-time, iar oamenii din cadrul comunității pot interveni prin adăugarea de informații noi și stabilirea de priorități.

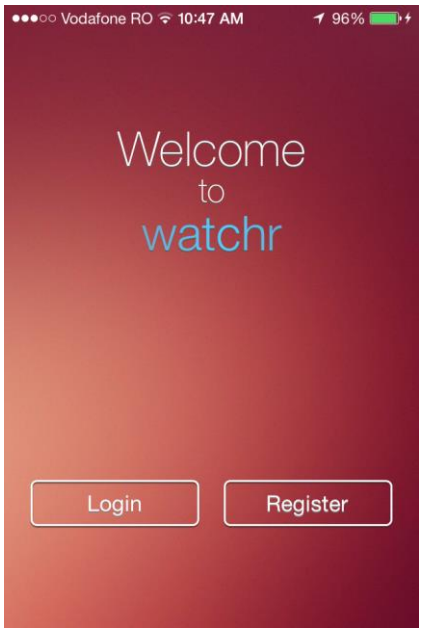

Fig. 2 Screenshot cu Welcome Screen

1.2.3 Interacțiunea Aplicația Client – Core Service

Comunicarea și transferul de fișiere și date dintre **aplicația client** și **core service** se face prin HTTP/HTPPS. Deoarece este necesară salvarea atașamentelor în diferite spații pe disk, serverul are access de *write* la diferite foldere cum ar fi *uploads/* și *profile_pictures/*. Permisiunile sunt date sistemul de permisiuni Posix ACL oferite de Unix și de configurarea serverului Apache.

Fiecare request este tratat independent pe sesiune, generând-se un nou proces, crescând astfel performanțele serviciului. User-ul care este logat în **aplicația client** poate trimite un request la API, iar cererea este tratată de controllerele frameworkului, bineînțeles dacă granturile sunt valide,.

1.2.4 Instalarea

Instalarea **Core Service-ului** este una destul de complexa ce necesită configurarea mai multor componente. Voi face aceeași clasificare cum am făcut și la prezentarea generală a componentelor principale.

- *Instalarea serverului Apache*

  Administratorul de sistem trebuie să instaleze serverul open-source Apache care se găsește pe site-ul dezvoltatorului și să ruleze scriptul de instalare. Apoi trebuie configurat pentru a spori performanta throughput-ului de requesturi și de a accepta upload-ul de fișiere. Instalarea necesită și setarea *datelor de autentificare* pentru administratorul de sistem.

- *Instalarea sistemului de gestiune a bazei de date - MySQL*

  Modelul bazei de date care este prezentat în Capitolul II este generat de către o aplicație de modelare numită *Vertabelo*. Pentru implementarea acesteia în Core Service trebuie importată întreaga schema in MySQL.

- **Inițializarea frameworkului Laravel**

  Laravel este bazat pe *package manager*-ul *Composer* creat special pentru gestiunea pachetelor PHP. Se va folosi acest package manager pentru generarea *controllerelor, modelelor, migrarilor si a testelor unit.* Totodată, Composer are posibilitatea de a instala pachete din repo-urile *The League of Extraordinary Packages si Github*. Astfel se vor încarcă toate pachetele necesare expuse în **composer.json** din root prin simpla executare a comenzii **composer self-update**.

Aplicația de iOS nu necesită instrucțiuni speciale pentru rularea sa, dar trebuie respectare obligatoriu un număr de condiții pentru a putea compila si rula aplicația pe device-ul de test:

- Obținerea unui cont de *developer iOS* de la Apple.
- Programatorul trebuie sa dețină un Mac sau cel puțin o mașina virtuala ce rulează Mac OS X 10.9.
- Softul pentru *development si deployment XCode*.

1.3 Analiza rețelelor sociale asemănătoare

Serviciul **watchr** este similar cu servicii bine cunoscute la momentul actual cum ar fi *Twitter si Foursquare*. Aceste rețele sociale reprezintă inspirația care a dus la implementarea acestui proiect.

Bineînțeles, rețeaua watchr nu este la scara imensa a lui Twitter, dar în esența elementele principale sunt aceleași. Twitter prezintă un feed de tweeturi, watchr un feed de evenimente cu locația specificată. Un watchr event poate conține poze, la fel ca un tweet. Fiecare user poate să-și exprime opiniile asupra unui eveniment, la fel cum se face și pe rețeaua Twitter sau Foursquare.

Watchr împrumuta sistemul de rating de la *reddit*. Utilizatorii pot crește sau scădea importanța unui eveniment printr-un simplu click. Comunitatea se poate implica activ în rezolvarea și promovarea evenimentelor.

Capitolul II

Core Service

watchr API

2.1 Tehnologii

2.1.1 Serverul Apache

Serverul web Apache este un software open source. Asta însemna că administratorul de sistem are acces la cod și poate crea o soluție speciala pentru cerințele speciale pe care le are. Multe dintre aceste soluții sunt scrise și sunt disponibile gratuit. Deoarece software-ul este open-source, Apache este actualizat constant. Mulți programatori din întreaga lume continuă dezvoltarea serverului, iar ultimele versiuni sunt întotdeauna disponibile.

Un al doilea avantaj pe care îl are designul open source este costul. Serverul Apache este oferit gratuit și poate fi instalat pe majoritatea sistemelor de operare existente. Un server asemănător produs de Microsoft poate duce la costuri de până la 1000 $. Soluția Apache este perfectă pentru businessuri ce lansează tehnologii noi, si care nu au un buget foarte mare pentru achiziționarea de servere [5].

Rețeaua socială ***watchr*** este dezvoltată în totalitate în PHP. Apache are suport *built-in* pentru acest limbaj. Serverul include suport pentru TLS și SSL, tehnologii cheie în prezervarea securității serviciului. Acestea sunt protocoale folosite pentru transferul de date criptate și sunt importante pentru dezvoltarea unor servicii sigure și de încredere, mai ales când vine vorba despre detaliile private ale clienților [5].

Apache este foarte portabil. Poate fi instalat pe o gamă largă de sisteme de operare. Serverul rulează pe toate versiunile de UNIX și pe toate versiunile bazate pe Windows NT și Mac OS. Apache este unul dintre cele mai versatile sisteme la ora actuală și este perfect pentru implementarea rețelei de socializare *watchr*.

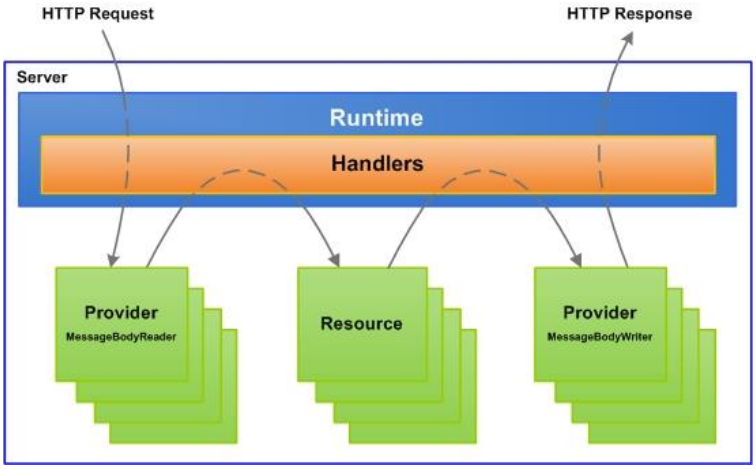

Fig. 3 Arhitectura generală a runtime-ului serverului Apache

2.1.2 Baza de date MySQL

Soluția *watchr* are caracteristicile unei rețele sociale. Gestionează utilizatori, evenimente și menține relațiile între acestea. Pentru a păstra și manipula toate datele necesare funcționării unui astfel de serviciu s-au folosit cele mai noi tehnologii din domeniu *(State of the Art)*. Este crucial ca toate datele și interacțiunile cu web service-ul să fie procesate corect.

Deoarece *watchr* este în esență un web app, utilizatorii trebuie să aibă acces la informațiile pe care ei le-au introdus. Aceste date sunt stocate într-o bază de date MySQL. Am ales această soluție datorită numeroaselor avantaje pe care MySQL le are [6]:

- Este un software open-source, ceea ce reduce costurile întreținerii unei astfel de implementări considerabil
- Este foarte rapidă, sigură si este constant îmbunătățită de programatorii care se ocupa de repo-ul MySQL.
- Este ușor de folosit, programatorii rețelei pot crea query-uri complexe prin folosirea funcțiilor simple din SQL.
- Este scalabilă: MySQL poate manipula până la 50 de milioane de rânduri sau mai multe. Limita fișierului poate crește la aproape 8 TB de date.

2.1.3 Framework-ul PHP Laravel 4

API-ul *watchr* este construit folosind framework-ul PHP open-source Laravel. Laravel este unul dintre cele mai complexe și robuste framework-uri de PHP cu suport pentru Eloquent ORM și sistem de migrare pentru MySQL, Postgres, SQL Server si SQL Lite.

Laravel 4 este cu totul și cu totul modular, iar fiecare modul este un package Composer. Aceste pachete pot fi incluse ca dependințe la proiect și pot fi cu ușurință descărcate din linia de comanda. Laravel este foarte bun în managementul rutelor, element cheie în dezvoltarea unui API REST (**Representational state transfer**). Fiecare metodă din API îi este stabilită o rută predefinită.

Unul dintre cele mai importante beneficii ale frameworkului este suportul pentru Eloquent ORM. Fiecare entitate din baza de date poate fi transpusă într-un model Eloquent care este folosit în manipularea datelor de către server. O entitate este tratată ca o clasă PHP și moștenește toate caracteristicile

acesteia. Deoarece PHP este un limbaj de programare orientat pe obiecte, se poate profita de toate beneficiile unei astfel de paradigme. Putem seta atribute și metode specifice fiecărui model.

Laravel este un framework foarte puternic și versatil, folosit pentru crearea unor aplicații web foarte complexe. Rețeaua *watchr* respectă toate definițiile unei rețele sociale modulare și scalabile. Structura proiectului este una extensibilă ce permite adăugarea a noi modele și controllere cu ușurință, fără a modifica elementele deja implementate.

2.2 Modelul bazei de date

În următoarele paragrafe voi descrie structura generală a bazei de date, câmpurile, relațiile logice dintre entități și legătura cu controllere scrise în PHP.

Diagrama a fost realizată in *Vertabelo*, un web app disponibil ca freeware pentru studenți.

Schema bazei de date se poate grupa în 3 porțiuni principale: **mecanismul de autentificare, gruparea user profile și gruparea *watchr event*.** Voi prezenta fiecare porțiune în detaliu în următoarele subcapitole.

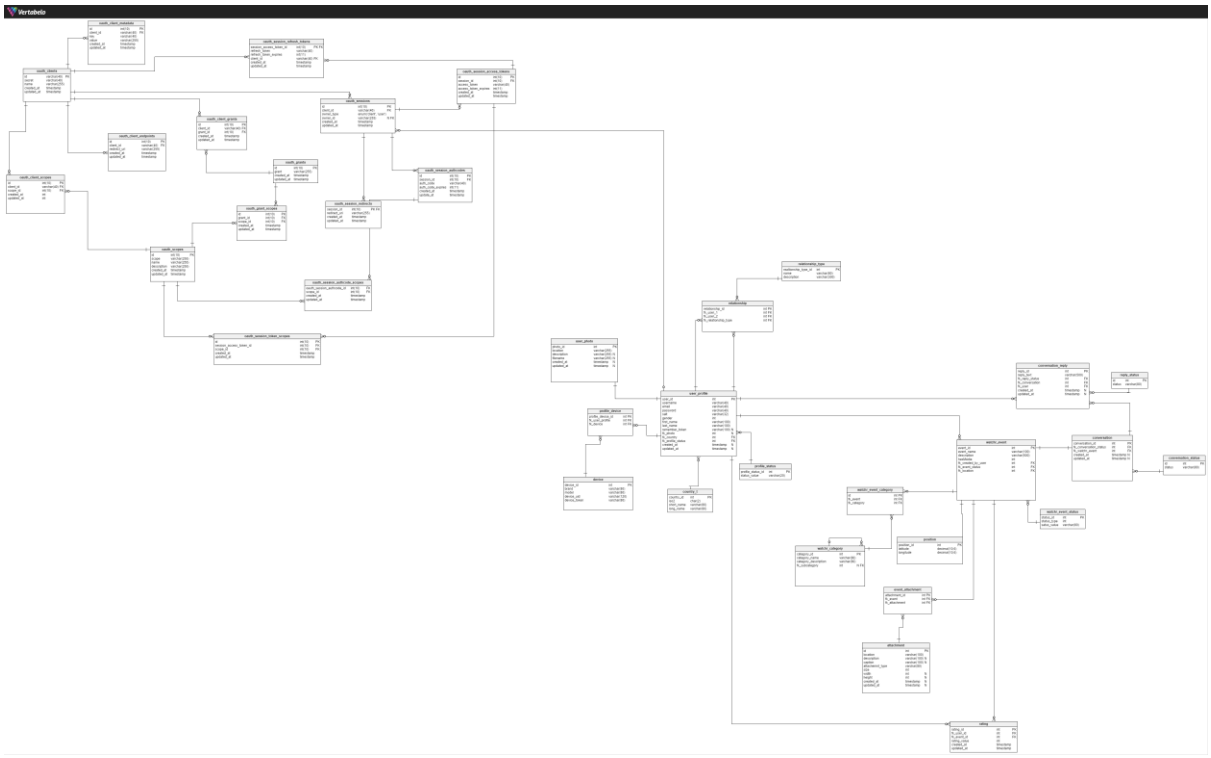

Fig. 4 Modelul bazei de date watchr

2.2.1 Gruparea user profile

După cum se poate observa în Fig. 4, nucleul acestei grupări este tabela **user_profile**. În această tabelă reținem toate informațiile necesare pentru a putea identifica un utilizator al aplicației *watchr*. O intrare în tabela **user_profile** reprezintă un cont asociat unui utilizator care are dreptul de a folosi aplicația și de a modifica baza de date. Aceste drepturi sunt totuși controlate de serviciu de autentificare

pe care îl vom prezenta în următoarele subcapitole. Tabela conține foreign key-uri la alte intrări cum ar fi poza utilizatorului, țara de care aparține și statusul profilului.

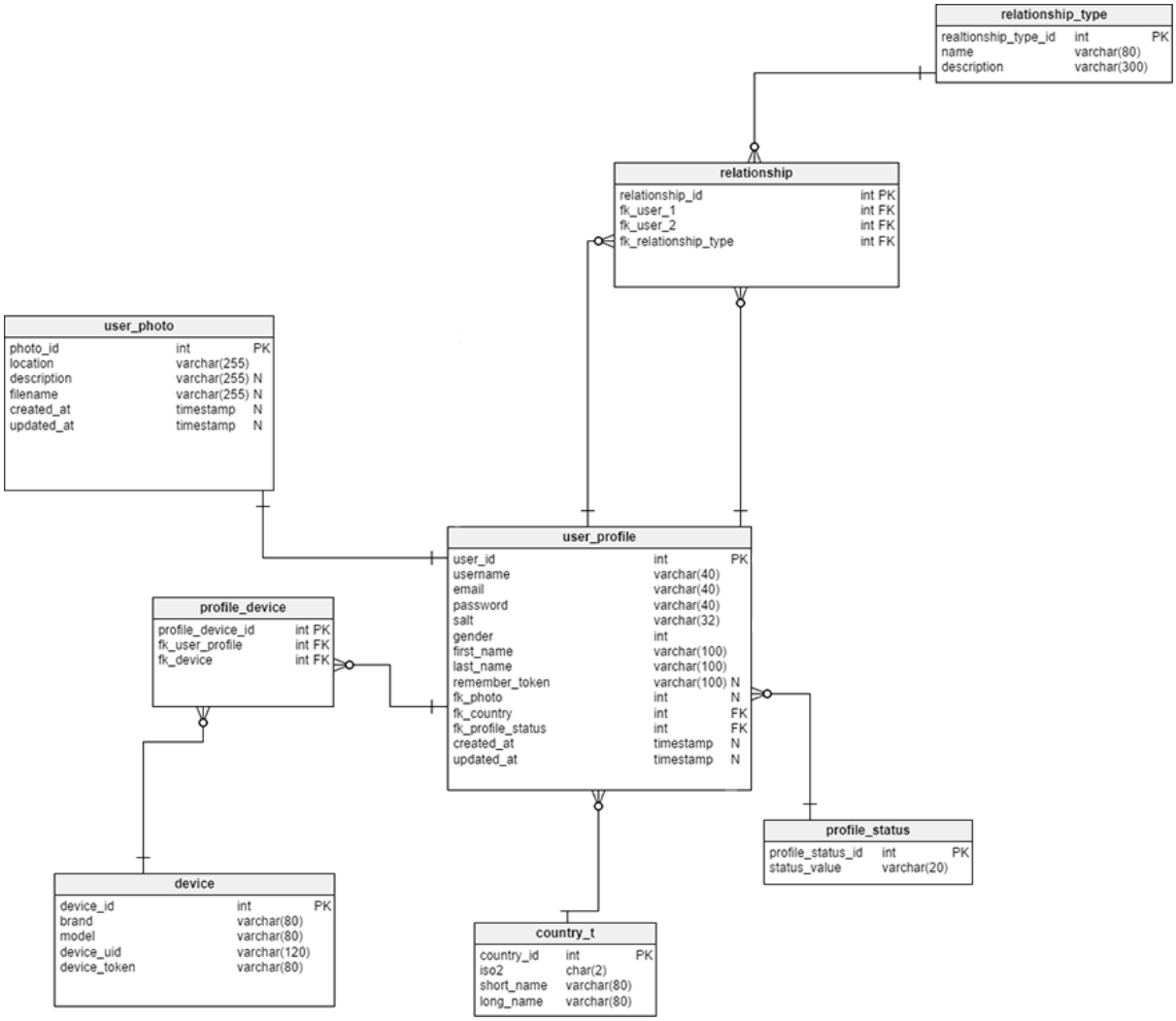

Fig. 5 Gruparea user profile

Tabela user_profile

Câmpuri:

- **user_id** – *int(11)* - id-ul unic al fiecărui user setat ca primary key.
- **username** – *varchar(40)* – username-ul fiecărui utilizator al serviciului, acesta trebuie să fie unic.
- **email** – *varchar(255)* - email-ul userului, care este de asemenea unic.
- **password** – *varchar(255)* - parola secretă a utilizatorului. În baza de date nu se păstrează parola in plain text , ci hashul acesteia. Pentru generarea hashului se folosește cel mai performant algoritm de hashing de parole, și anume *Bcrypt*, care folosește cifrul *MCRYPT_RIJNDAEL_128*, considerat cel mai securizat cifru [7].
- **gender** – *int(11)* - sexul utilizatorului. Se folosește standardul *ISO/IEC 5218* [8] care specifică 4 tipuri de cod:
  - **0** – necunoscut

    - **1** – masculin
    - **2** – feminin
    - **9** – nu se aplică
- **first_name, last_name** – *varchar(100)* - prenumele și numele sunt intrări opționale.
- **fk_photo, fk_country, fk_profile_status** – *int(11)* - foreign keys pentru poza de profil a utilizatorui, țara din care aparține și statusul profilului (activ, blocat, închis, stres)
- **created_at, updated_at** – *timestamp* - câmpuri standard care oferă informații despre data creări și data actualizării.

Tabela user_photo

Entitate ce păstrează informații despre poza de profil a utilizatorului.

Câmpuri:

- **photo_id** – *int(11)* – id-ul unic al fiecărei poze setat ca primary key
- **location** – *varchar(255)* – calea relativa unde se găsește assetul respectiv.
- **description, filename** – *varchar(255)* – descrierea pozei. Câmpuri opționale, pot fi null.
- **created_at, updated_at** – *timestamp* - câmpuri standard care oferă informații despre data creări și data actualizării.

Tabelele profile_device **și** device

Tabela device reține informații despre terminalul de pe care utilizatorul este logat. Un terminal poate avea mai multe conturi, iar soluția rezolvării relației many-to-many constă în tabela profile_device. Informațiile care sunt stocate sunt folosite pentru identificarea device-ului și pentru a trimite notificări.

Câmpuri tabela **device**:

- **device_id** – *int(11)* – id-ul unic al fiecărui terminal setat ca primary key
- **brand, model** – *varchar(80)* – informații generice despre terminal.
- **device_uid, device_token** – *varchar(80*) – id-uri speciale folosite pentru identificarea dispozitivului și token-ul de push notification.

Tabelele relationship **si** relationship_type

Tabela relationship contine relațiile dintre 2 utilizatori. Tipul acestora sunt stabilite de foreign keyul **fk_relationship_type**.

Câmpuri tabela **relationship**:

- **relationship_id** – *int(11)* – id-ul unic al fiecărei relații setat ca primary key
- **fk_user_1, fk_user_2** – *int(11)* – foreign keys la cheia primară din tabela **user_profile**.
- **fk_relationship_type** – *int(11)* – foreign key la cheia primară din tabela **realtionship_type**. Tabela relationship_type conține următoarele intrări:
    - **pending** – cererea de prietenie a fost trimisă, dar nu a fost încă acceptată.
    - **friend** – cererea de prietenie a fost acceptată.
    - **blocked** – utilizatorul *fk_user_1* a blocat orice interacțiune directă cu *fk_user_2.*
    - **deleted** – Relația a fost setată ca inexistentă în urma închiderii unui cont de user_profile. Daca un user își redeschide contul, acesta va reveni la valoarea inițială (pending, friend, blocked).

2.2.2 Gruparea *watchr* event

Tabela **watchr_event** conține informațiile cheie pentru fiecare eveniment creat de utilizatorii rețelei *watchr*. Fiecare eveniment are asociată o listă de conversații unde utilizatorii autentificați pot posta comentarii. Un eveniment poate avea mai multe categorii, iar soluționarea relației many-to-many se face prin tabela pivot **watchr_event_category**. O intrare *watchr_event* poate avea și atașamente.

Tabela **event_attachment** este o tabelă pivot, deoarece un atașament poate fi legat de alt eveniment daca utilizatorul dorește sa împărtășească intrarea respectivă (poate să înregistreze poza sau video-ul la mai multe evenimente asemănătoare). Stabilirea importanței evenimentului se face cu ajutorul intrărilor din tabela **rating**.

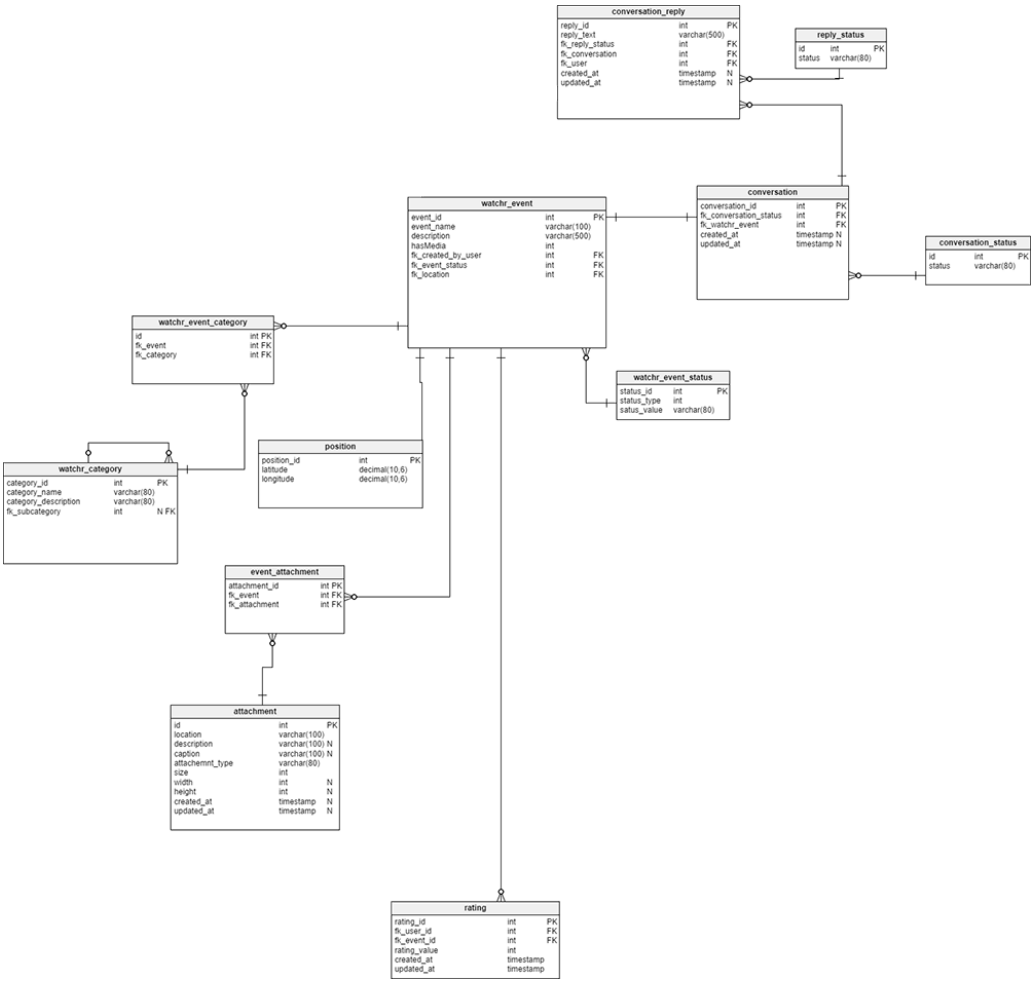

Fig. 6 Gruparea watchr event

Tabela watchr_event

Tabel conține informații despre eveniment cum ar fi numele acestuia și descrierea, id-ul utilizatorului care a creat evenimentul și poziția sa pe harta.

Câmpuri:

- **event_id** – *int(11)* – id-ul unic al evenimentului setat ca primary key
- **event_name** – *varchar(100)* – titlul evenimentului.
- **description** – *varchar(500)* – descrierea evenimentului. Câmp opțional, poate fi null.
- **hasMedia** – *int* – flag ce semnalizează dacă evenimentul prezintă atașamente.
- **fk_created_by_user** – *int(11)* – foreign key la tabela de **user_profile.**
- **fk_event_status** – *int(11)* – foreign key la tabel **watchr_event_status** ce conține următoarele intrări:
  - **open** – evenimentul este activ
  - **deleted** – evenimentul este stres de user-ul care l-a creat
  - **expired** – evenimentul a expirat
  - **resolved** – evenimentul a fost tratat
- **fk_location** – *int(11)* – foreign key la tabela **position** ce reține coordonatele GPS ale evenimentului
- **created_at, updated_at** – *timestamp* - câmpuri standard care oferă informații despre data creări și data actualizării.

Tabela watchr_category

Fiecare eveniment are o categorie, iar dacă aceasta nu este setată se va lua valoarea implicita *Other*. Se pot stabili mai multe categorii la un eveniment, iar categoriile pot avea alte subcategorii tot de tipul **watchr_category**.

Câmpuri:

- **category_id** – *int(11)* – id-ul unic al categoriei setat ca primary key
- **category_name** – *varchar(80)* – numele unei categorii.
- **category_description** – *varchar(80)* – descrierea categoriei respective. Câmp opțional, poate fi null.
- **categort_icon** – *varchar(50)* – este stabilită numele iconiței categoriei, care va fi folosită in aplicația client
- **fk_subcategory**– *int(11)* – foreign key reflexivă. Reține id-ul categoriei părinte.

Tabela attachment

Un eveniment *watchr* poate fi acompaniat de alte atașamente, cum ar fi poze, video-uri, etc. In tabela **attatchment** reținem toate informațiile necesare pentru stocarea fișierului.

Câmpuri:

- **id** – *int(11)* – id-ul unic al fișierului setat ca primary key
- **location** – *varchar(100)* – calea relativă a fișierului.
- **description, caption**– *varchar(80)* – elemente suplimentare. Câmpuri opționale, pot fi null.
- **attachment_type** – *varchar(80)* – este salvat MIME type-ul fișierului.
- **size** – *int* – Câmp opțional ce păstrează mărimea fișierului in bytes.
- **created_at, updated_at** – *timestamp* - câmpuri standard care oferă informații despre data creări și data actualizării.

Tabelele conversation **si** conversation_reply

Fiecare eveniment *watchr* are atașat un id de conversație. Relația este stabilită prin foreign key la id-ul unui eveniment. Fiecare conversație conține un **conversation_reply** ce reprezintă comentariul unui utilizator la eventul respectiv.

Câmpuri tabela **conversation**:

- **conversation_id** – *int(11)* – id-ul unic al conversației setat ca primary key
- **fk_watchr_event** – *int(11)* – foreign key la cheia primară din tabela **watchr_event**.
- **fk_conversation_status** – *int(11)* – foreign key către tabela **conversation_status**. Tabela conversation_status conține următoarele intrări
  - **open** – utilizatorii pot posta comentarii la eveniment.
  - **closed** – creatorul evenimentului poate dezactiva sistemul de comentarii.
  - **deleted** – conversația a fost ștearsa.
- **created_at, updated_at** – *timestamp* - câmpuri standard care oferă informații despre data creări și data actualizării.

Câmpuri tabela **conversation_reply**:

- **reply_id** – *int(11)* – id-ul unic al comentariului setat ca primary key
- **reply_text** – *varchar(500)* – comentariul user-ului pe care l-a postat la eveniment.
- **fk_conversation** – *int(11)* – foreign key către tabela **conversation**. Indică cui ii aparține comentariul.
- **fk_user** – *int(11)* – foreign key către tabel **user_profile**. Acest câmp reprezintă userul care a scris comentariul.
- **created_at, updated_at** – *timestamp* - câmpuri standard care oferă informații despre data creări și data actualizării.

Tabela rating

Un utilizator autentificat poate da un rating la fiecare eveniment. Acesta poate alege doar 3 valori: -1, 0, 1. Acest sistem de rating este împrumutat de la reddit și oferă o evaluare a evenimentului în funcție de popularitatea sa între oamenii din cadrul comunității.

Câmpuri tabela **conversation_reply**:

- **rating_id** – *int(11)* – id-ul unic al ratingului setat ca primary key
- **fk_user_id** – *int(11)* – foreign key către tabel **user_profile**. Acest câmp reprezintă userul care și-a exprimat votul asupra evenimentului.
- **fk_event_id** – *int(11)* – foreign key la cheia primară din tabela **watchr_event**.
- **rating_value** – *int(11)* – nota asociată cu valoarea -1, 0 sau 1.
- **created_at, updated_at** – *timestamp* - câmpuri standard care oferă informații despre data creări și data actualizării.

2.2.3 Mecanismul de autentificare

Autentificarea pentru accesarea API-ul *watchr* se face prin OAuth 2.0. Voi intra în detaliile acestui protocol în Capitolul III. Fiecare intrare din **user_profile** va putea folosi serviciul numai dacă acesta s-a autentificat. După autentificare serverul pornește o sesiune, cu informațiile acesteia aflându-se în tabela **oauth_session**. Această sesiune nu dispare odată ce utilizatorul nu mai folosește aplicația. Expirarea unei sesiuni este dată de expirarea tokenului, pe care îl vom trata în următorul capitol.

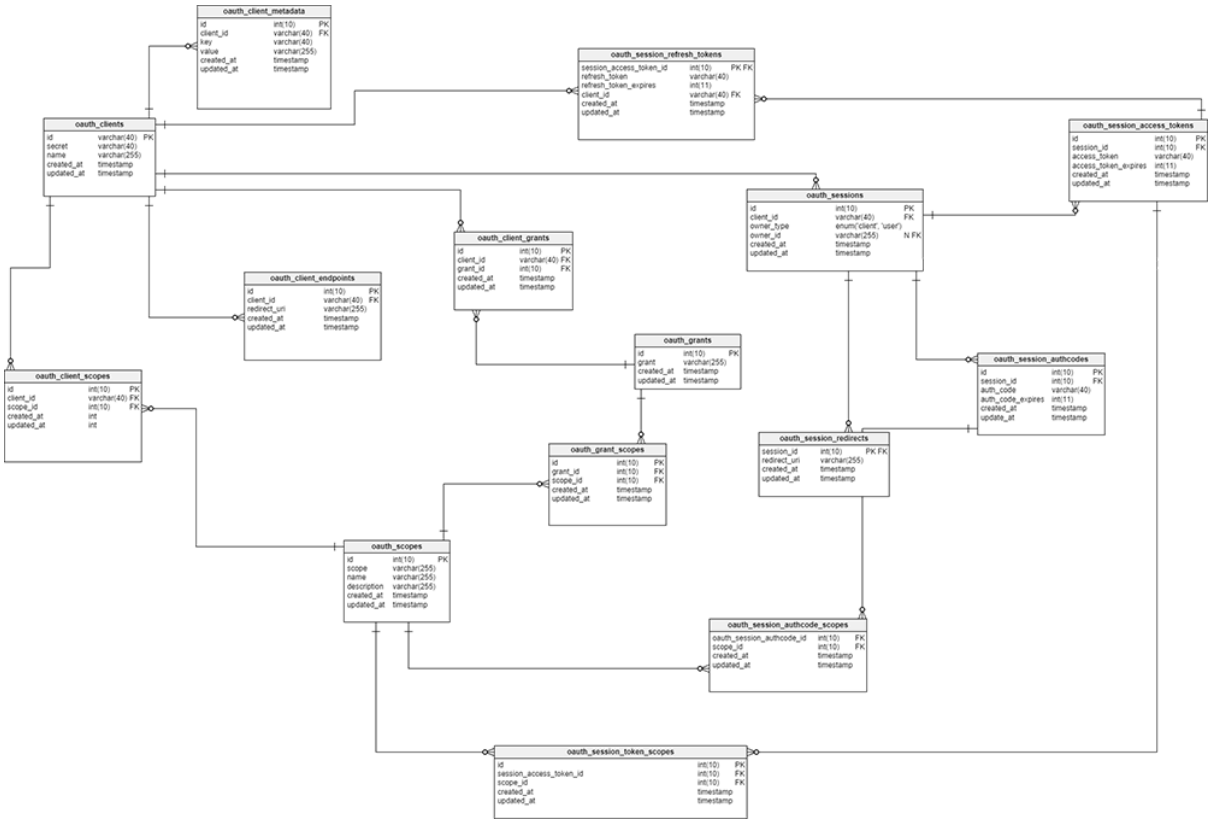

Fig. 7 Gruparea mecanismului de autentificare

2.3 Implementarea API-ului REST *watchr*

API-ul REST **watchr** este scris în totalitate în PHP. REST provine de la **Representational state transfer**. Acest termen a fost definit de Roy Fielding în anul 2000 în lucrarea sa de doctorat la UC Irvine [9].

Rutele dintr-un API REST sunt de fapt metode pe care clientul le apelează printru-un request HTTP. Comunicarea dintre client și server trebuie să fie stateless, în sensul că nici un context din partea clientului nu este salvat pe server între requesturi. Fiecare request de la orice client conține toate informațiile necesare pentru a acționa cererea. Starea sesiunii se menține la client. Important de notat este că starea sesiunii poate fi transferată de către server unui alt serviciu cum ar fi baza de date care menține starea persistentă pentru a permite autentificarea. Clientul va trimite alte requesturi odată ce cunoaște response-ul de la server. Doar atunci poate continua tranziția intre stări.

Arhitectura unui API REST are o structură layered , în sensul că clientul nu știe dacă este conectat direct la end point-ul serverului sau dacă este conectat la un punct intermediar. Servere intermediare pot crește performanța și scalarea sistemului prin activarea load-balancing-ului și prin suportul pentru shared cache. Aceste servere pot implementa și politici de securitate.

API-uri REST au o interfața uniformă. Aceasta simplifică și decuplează arhitectura, ce permite fiecărei componente să evolueze independent.

Wew serverul *watchr* aderă la constrângerile REST. Un API RESTful este definit de aceste aspecte

- **base URI**, cum ar fi *http://watchr.com/resources*
- **un Internet media type** pentru data precum *JSON sau XML*

- **metode standar HTTP** (*GET, POST, PUT, DELETE etc*.)
- **linkuri hypertext pentru stări**
- **linkuri hypertext pentru resurse**

2.3.1 Rutele in Laravel

Toate rutele în afară de cea de cerere a tokenului de acces și de înregistrare utilizator nou sunt trecute printr-un filtru. Acel filtru verifică daca utilizatorul are acces la API și nu permite modificarea datelor rețelei dacă tokenul nu este valid. Prin prezența keywordului `'before'` Laravel cere ca utilizatorul care a accesat ruta, să fie autentificat. Această filtrare se face prin:

```
Route::group(array('before' => 'oauth'), function()
        {
                //watchr API resource routes
        };
```

Autentificare si cererea unui token de acces

```
Route::post('oauth/access_token', function()
    {
        return AuthorizationServer::performAccessTokenFlow();
    });
```

Autentificarea se face prin trimiterea unui request prin metoda **POST** la *oauth/access_token* cu următorii parametri:

- **grant_type:** metoda de autentificare, în cazul nostru este *password*
- **client_id:** id-ul clientului care trimite requestul
- **client_secret:** șir de caractere care este distribuit fiecărei aplicații de tip client
- **username:** numele de utilizator
- **password:** parola utilizatorului
- **scope:** scope-ul autentificării

Dacă user-ul a făcut un request valid, serverul va trimite un response cu **access_token,** timpul de expirare, tipul tokenului și tokenul de refresh folosit pentru reinițializarea tokenului atunci când acesta expiră.

```
{
    "access_token": "sx5QOdq3YqvnlADuTcaKD3ml98GTXK7H1wnLrebq",
    "token_type": "bearer",
    "expires": 1404473568,
    "expires_in": 604800,
    "refresh_token": "tMwdLwvUvyZIymAIP74KIg3cHSV69NjYYnISU5g5"
}
```

Exemplu de rute pentru managementul userilor

```
//User Profile routes

Route::get("users/me", array(
```

```
            "as"    => "user/me",
            "uses" => "UserProfileController@get_logged_in_user"
        ));
```

- ruta **GET** *users/me* va returna un response cu toate informațiile utilizatorului logat.

```
Route::get("users/{user_id}", [
            "as"    => "user/show",
            "uses" => "UserProfileController@show"
        ]);
```

- ruta **GET** *users/{user_id}* va returna informații despre un anumit user în funcție de id.

```
Route::post("users/delete/{user_id}", [
            "as"    => "user/destroy",
            "uses" => "UserProfileController@destroy"
        ]);
```

- ruta **GET** *users/delete/{user_id}* va seta contul utilizatorului cu statusul *deleted*.

2.3.2 Modele Eloquent pentru reprezentarea datelor

API-ul watchr folosește modele Eloquent. Eloquent ORM este un sistem de *ActiveRecord* folosit pentru lucrul cu baza de date. Fiecare tabel din baza de date îi corespunde un *model* care este apoi folosit pentru interacțiunea cu acel tabel. Eloquent suporta MySQL, Postgres, SQL Server si SQL Lite.

Pentru a explica cum vom folosi modele Eloquent vom lua drept exemplu modelul **User_profile** care reprezintă de fapt tabela *user_profile* din baza de date. Acest snippet de cod nu reprezintă întreaga clasă, dar aceste elemente sunt suficiente pentru a explica de ce am alege un ORM în schimbul manipulării manuale a datelor folosind SQL.

```
class User_profile extends \Eloquent implements UserInterface,
RemindableInterface {

    protected $fillable = [
        "username",
        "email",
        "first_name",
        "last_name",
    ];

    protected $table = "user_profile";

    protected $primaryKey = "user_id";

    protected $hidden = ["password", "fk_profile_status"];

    protected $guarded = [
        "user_id",
        "password"
    ];
```

```
    /**
     * Get user's country information.
     *
     * @return Country model
     */

    public function country(){
        return $this->hasOne('Country', 'fk_country', 'country_id');
    }


    /**
     * Get user's photo.
     *
     * @return Photo model
     */

    public function photo()
    {
        return $this->hasOne('User_photo', 'photo_id', 'fk_photo');
    }
}
```

După cum se poate observa clasa **User_profile** extinde o clasa de tip **Eloquent**. Array-ul **fillable** setează câmpurile ce pot fi mass-assignable in baza de date de către serviciul de migrare a frameworkului Laravel. Setarea tabelului la care face referire modelul se face prin variabila **table.** Variabila **primary_key** reprezintă numele câmpului care este setat ca cheie primară.

Câteodată se dorește limitarea atributelor care sunt incluse în array-ul modelului sau în format JSON, cum ar fi parole sau alte câmpuri secrete. Acest lucru se face folosind array-ul **hidden**, enunțând câmpurile ce nu vor fi vizibile la conversia în array sau JSON[1].

Array-ul **guarded** este inversul atributului **fillable**. Fiecare câmp enunțat nu este mass-assignable. De exemplu *user_id*-ul nu este asignabile pentru acesta este setat ca un int AUTO_INCREMENT în baza de date.

Relațiile în Eloquent sunt stabilite prin funcții. Funcția country întoarce modelul **Country** având drept cheie primara **country_id**, iar foreign key-ul din tabela **User_profile** este **fk_country**.

2.3.3 Validarea parametrilor din request

Fiecare request este validat înainte de modificarea acesteia în bazei de date. Astfel se va păstra puritatea și buna funcționare a acesteia. Validarea se face folosind clasa **Validator.** In snippetul de mai jos avem un exemplu de validare a *user_id*-ului.

```
        $validator = Validator::make(array(
                    'userId' => $_user_id
                ),array(
                    'userId' => 'required|integer|exists:user_profile,user_id'
                ));
```

[1] The JavaScript Object Notation (JSON) Data Interchange Format, http://tools.ietf.org/html/rfc7159

```
        if($validator->fails()){
              return Response::json(array(
                      "error"=>$validator->messages()->all(),
                  )
                  ,400);
          }
```

Se verifică **_user_id** dacă respectă regulile impuse de Validator. După cum se poate observa, câmpul trebuie setat, nu are voie să fie *null*, trebuie să fie de tipul *integer* și trebuie să existe în tabela **user_profile**. Dacă nu respectă una dintre reguli, se va trimite un *response* cu toate regulile care nu au fost respectate.

2.3.4 Query pentru găsirea celor mai apropiate evenimente [10]

Watchr folosește baza de date MySQL. Am explicat la începutul capitolului de avantajele folosirii unei astfel de tehnologii. Totuși, ea prezinta și câteva dezavantaje. Una din deficientele bazei de date MySQL este lipsa de suport pentru funcții de geo locație. Geo locația reprezintă un punct cheie în designul serviciului watchr. În alte soluții cum ar fi cele de la Oracle avem funcții speciale pentru query-uri care conțin elemente de geo locație, însă în MySQL acestea trebuie implementate manual.

Problema este următoarea: Având o tabela ce conține locația evenimentelor cu latitudine și longitudine, care dintre acele evenimente sunt mai apropiate față de o poziție anume?

Latitudinea și longitudinea sunt exprimate în grade. Latitudinea descrie cât de departe, în nord sau în sud, față de ecuator se află un punct. Punctele de-a lungul ecuatorului au latitudinea 0. Polul nord are o latitudine pozitivă de 90, iar polul sud -90. În consecința, locațiile din emisfera nordică au latitudine pozitivă, iar locațiile din emisfera sudică au latitudini negative.

Longitudinea descrie cât de departe este un punct față de primul meridian (o linie arbitrară care parcurge suprafața Terrei de la pol la pol). Observatorul Greenwich din apropierea Londrei, are, prin definiție, longitudinea 0.

Latitudinea se exprimă în valori cuprinse între [-90, 90], iar longitudinea în valori cuprinse între (-180, 180]. Aceste valori sunt exprimate în grade, minute și secunde, dar pentru a face calcule este necesară convertirea lor în valori decimale.

În implementarea serviciului *watchr* se folosește următoarea funcție pentru query-ul locațiilor:

```
SELECT *,latitude,longitude, distance,
       (SELECT SUM(r.rating_value)
        FROM rating r
        WHERE r.fk_event_id = E.event_id) AS event_rating
FROM watchr_event E,
     (SELECT latitude, longitude,position_id,r,
             111.045* DEGREES(ACOS(COS(RADIANS(latpoint))
                    * COS(RADIANS(latitude))
                    * COS(RADIANS(longpoint) - RADIANS(longitude))
                    + SIN(RADIANS(latpoint))
                    * SIN(RADIANS(latitude)))) AS distance
      FROM position
      JOIN (SELECT  :latidude AS latpoint,
                    :longitude AS longpoint,
```

```
                    :radius AS r ) AS p
            WHERE latitude
                  BETWEEN latpoint  - (r / 111.045)
                      AND latpoint  + (r / 111.045)
            AND longitude
                  BETWEEN longpoint - (r /(111.045*COS(RADIANS(latpoint))))
                      AND longpoint + (r /(111.045*COS(RADIANS(latpoint))))
      ) d
WHERE distance <= r
AND position_id = E.fk_location
AND E.fk_event_status=1
```

Pentru stabilirea distanței dintre două puncte de pe suprafața pământului se folosește **Formula Haversine** sau **Legea Cosinusului Sferic.** Sintaxa SQL pentru acesta formula este următoarea:

```
      111.045 * DEGREES(ACOS(COS(RADIANS(lat1)) * COS(RADIANS(lat2)) *
                             COS(RADIANS(long1) - RADIANS(long2)) +
                             SIN(RADIANS(lat1)) * SIN(RADIANS(lat2))))
```

Aceasta este distanța de-a lungul suprafeței terestre. Rezultatul, după cum se poate observa, este în grade. Asta înseamnă că valoarea trebuie multiplicată cu 111.045 (1 grad = 11.045 km) dacă vrem ca distanța să fie în km.

Totuși, implementarea aceasta nu este foarte eficientă și putem aduce câteva optimizări. Query-ul este lent deoarece trebuie calculată formula haversine pentru fiecare pereche de puncte posibilă. Rezultă automat că serverul va face multe computații matematice inutile și va forța verificarea fiecărei poziții din tabelă. Din conversia anterioară știm că un grad are o valoare de 111.045 km. Atunci, putem folosi clauze SQL să eliminăm punctele care sunt prea îndepărtate de raza cerută. Până în punctul acesta am creat un bounding-box, dar noi vrem să găsim evenimentele care se petrec în cadrul unei raze de cerc. Trebuie eliminate locațiile care nu se încadrează în limita impusă de raza.

```
      WHERE latitude
            BETWEEN latpoint  - (r / 111.045)
                AND latpoint  + (r / 111.045)
      AND longitude
            BETWEEN longpoint - (r /(111.045*COS(RADIANS(latpoint))))
                AND longpoint + (r /(111.045*COS(RADIANS(latpoint))))
```

În momentul acesta am reușit să implementăm un algoritm eficient de extragere a evenimentelor care se află în raza stabilită de utilizator.

Capitolul III

Securitate

Protocolul OAuth 2.0

3.1 Introducere

Framework-ul OAuth 2.0 permite unei aplicații third-party să obțină access limitat la un service HTTP, fie în numele proprietarului de resursă (*resource owner*), fie prin permiterea aplicației să obțină accesul în numele propriu. Acest protocol înlocuiește în totalitate versiunea anterioara, și anume OAuth 1.0.

OAuth a fost dezvoltat de către **Internet Engineering Task Force** (**IETF**) [11]. Framework-ul a fost publicat în Octombrie 2012. Graph API dezvoltat de Facebook folosește exclusiv OAuth 2.0 pentru autentificare. Toate API-urile oferite de Google suportă autentificare OAuth 2.0.

În modelul tradițional de autentificare client-server, clientul cere o resursă protejată de la server prin autentificarea sa cu acesta cu datele de autentificare ale proprietarului resursei. Pentru a permite aplicațiilor să acceseze acea resursă, proprietarul resursei trebuie să-și împărtășească datele de autentificare cu acea aplicație. Acest model are câteva dezavantaje:

- Aplicațiile trebuie să stocheze datele utilizatorului pentru utilizarea lor viitoare. De obicei, se salvează o parola în plain text.
- Serverele trebuie să suporte autentificare prin parole, ceea ce duce la o securitate mult mai slabă.
- Aplicațiile de la terți pot avea acces larg, neputând seta o constrângere asupra resurselor.
- Deținătorii de resurse nu pot revoca accesul unei aplicații third party individuale. Dacă se dorește revocarea accesului, se vor revoca accesul tuturor aplicațiilor prin schimbarea parolei.
- Compromiterea unei aplicații rezultă automat în compromiterea parolei end-user-ului și toate datele protejate de acea parola.

În locul folosirii datelor de autentificare pentru accesul la resursa protejată, clientul obține un token de acces – un șir de caractere care indică scopeul, timpul de viața și alte atribute de acces. Tokenurile sunt emise pentru clienți third-party de un server de autorizare cu aprobarea proprietarului de resursă. Clientul utilizează tokenu-ul de acces pentru a accesa resursa protejată, care este găzduită pe server.

API-ul *watchr* poate fi accesat doar prin autentificare cu OAuth 2.0, folosind biblioteca dezvoltata de Luca Degasperi [12].

3.1.1 Roluri

OAuth definește patru roluri:

1. **Proprietarul de resursa**

   O entitate capabilă să acorde acces la o resursă protejată. Când proprietarul este o persoană, acesta este referit ca end-user.

2. **Serverul de resurse**

   Serverul care găzduiește resursele protejate, capabil să accepte și să răspundă la requesturi folosind tokenuri de acces.

3. **Clientul**

   Clientul reprezintă o aplicație care face requesturi la resurse protejate în numele proprietarului de resursă, folosind autorizația acestuia. Termenul „client” nu implică un tip specific (ex. server, desktop, mobil etc.).

4. **Serverul de autorizare**

   Serverul care emite tokenuri de acces clientului după autentificare. Serverul de autorizare poate fi același server ca și cel de resurse sau poate fi o entitate separată.

3.1.2 Fluxul protocolului

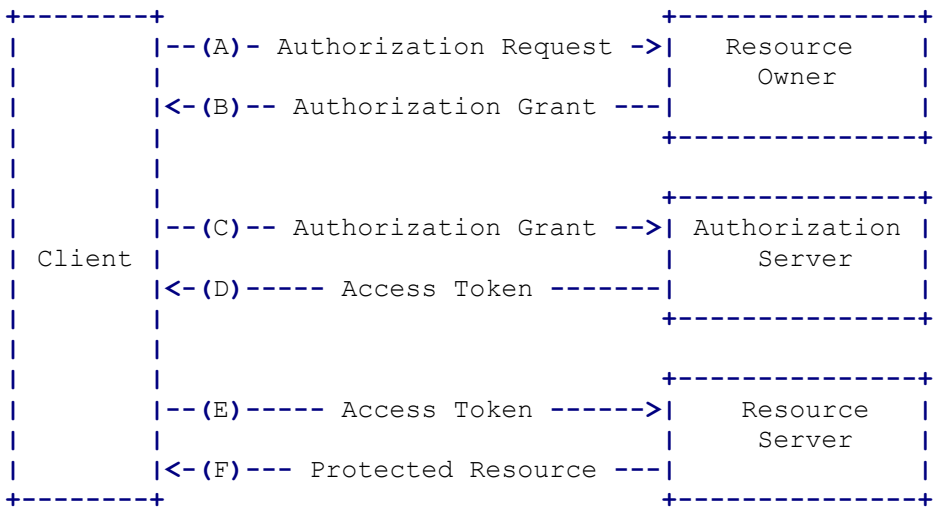

Fig. 8 Fluxul protocolului

Fluxul ilustrat din Fig. 8 descrie interacțiunea dintre cele 4 roluri și include următorii pași:

(A) Clientul cere autorizație de la proprietarul de resursă. Requestul de autorizație poate fi făcut direct la proprietar sau indirect via serverului de autorizație, folosit drept intermediar.

(B) Clientul primește un acord de autorizație, care este un element de autentificare reprezentând autorizarea proprietarului de resursa, exprimat folosind una din cele patru tipuri de acorduri de autorizație. Acordul de autorizație depinde de metoda folosită de client pentru requestul de autorizație.

(C) Clientul cere un token de acces pentru autentificarea sa cu serverul de autorizare și prezintă acordul de autorizare.

(D) Serverul de autorizare autentifică clientul și validează acordul de autorizare, iar dacă acesta este valid, se emite un token de acces.

(E) Clientul cere apoi resursa protejată de la serverul de resurse și se autentifică prin prezentarea tokenului de acces.

(F) Serverul de resurse validează tokenul de acces, iar dacă acesta este valid, ii va răspunde la cerere clientului.

3.1.3 Acordul de autorizatie

Un acord de autorizație este un element de autentificare reprezentând tipul de autorizație pe care proprietarul de resursă cere. Acesta este folosit de client pentru a obține tokenul de acces. OAuth definește patru tipuri de acorduri:

1. codul de autorizație
2. implicit
3. folosirea parolei proprietarului de resursa
4. datele clientului

API-ul *watchr* folosește momentan doar cel de-al doilea serviciu, autorizare prin folosirea parolei utilizatorului. De aceea voi detalia doar acest tip de acord în subcapitolele următoare.

3.1.4 Folosirea parolei proprietarului de resursa

Datele de autentificare ale utilizatorului (username și parola) pot fi folosite direct ca un acord de autorizație pentru a obține tokenul de acces. Datele ar trebui folosite doar de către clienți cu grad de încredere foarte ridicat și doar când celelalte acorduri nu sunt disponibile (spre exemplu un cod de autorizatie). Aplicația *watchr* face parte din întregul serviciu și prezinta cel mai mare grad de încredere.

Chiar dacă acest tip folosește legătura directa la datele de autentificare ale proprietarului de resursa, acestea sunt folosite doar o singură dată la cererea tokenului de acces. Acest tip de autorizație elimină necesitatea stocării numelui de utilizator și a parolei prin schimbarea lor cu un token de acces sau token de refresh.

3.1.5 Tokenul de acces

Token-uri de acces sunt date de autentificare folosite pentru a accesa resurse protejate. Un token de acces este un șir de caractere ce reprezintă autorizația acordată clientului. Token-urile prezintă un domeniu specific și timp de viață, acordat de proprietarul de resursă și impus de serverul de resurse și serverul de autorizare.

Tokenul denotă un identificator folosit pentru a prelua informațiile de autorizație într-o manieră verificabilă. (de exemplu, un semn constând în câteva date și o semnătură).
Tokenul de acces oferă un strat de abstractizare, înlocuind diferite modele (de ex. autentificarea cu username și parola) cu un singur token care este înțeles de serverul de resurse. Această abstractizare asigură că serverul de resurse înțelege domeniul de autorizație pe care clientul îl are și nu mai are nevoie de alte metode diferite de autentificare.

Tokenurile de acces au diferite formate, structuri și metode de utilizare. API-ul *watchr* folosește un token de 40 de caractere de forma:

```
"access_token": "sx5QOdq3YqvnlADuTcaKD3ml98GTXK7H1wnLrebq"
```

3.1.6 Tokenul de refresh

Token-uri de refresh sunt date de autentificare folosite pentru a obține tokenuri de acces. Tokenurile de refresh sunt emise clientului de serverul de autorizare și sunt folosite pentru obținerea unor tokenuri de acces noi atunci când acesta devine invalid sau expiră. Emiterea unui token de refresh este opțională. Dacă serverul de autorizare emite un token de refresh, acesta este inclus în emiterea unui token de acces.

Un token de refresh este un șir de caractere care reprezintă autorizația acordată clientului de proprietarul de resursă. Spre deosebire de tokenurile de acces, un token de refresh este destinat serverelor de autorizare și nu sunt niciodată transmise serverului de resurse.

3.2 Obținerea autorizației prin username si parola

Momentan serviciul *watchr* suportă doar autorizarea prin username și parolă. În următoarele subcapitole voi explica felul în care acest mod de autorizare funcționează.

Acest tip de acord de autorizare este potrivit pentru clienții ce au capabilitatea de a obține datele de autentificare ale proprietarului de resursă (username și parola). De asemenea, este utilizat pentru a migra clienții existenți folosind scheme de autentificare directe, precum HTTP Basic sau autentificare Digest, prin convertirea datelor de autentificare într-un token de acces.

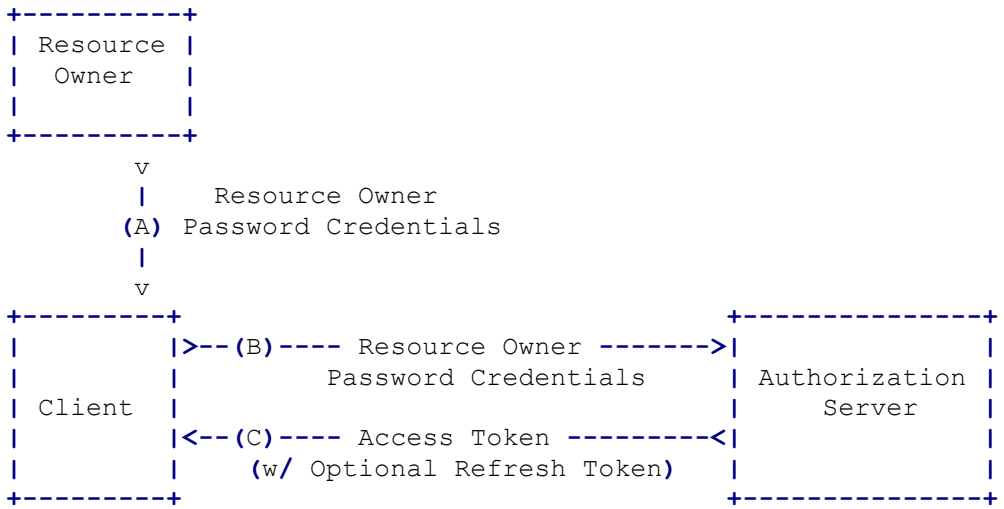

Fig. 9 Fluxul autentificării cu username si parolă

Diagrama de sus denotă următorii pași:

(A) Proprietarul de resursă îi oferă clientului numele de utilizator și parola.

(B) Clientul solicită un token de acces de la serverul de autorizare prin includerea datelor primite de la proprietarul de resursa. Atunci când se face cererea, clientul se autentifică cu serverul de autorizare.

(C) Serverul de autorizare autentifică clientul și validează datele proprietarului, iar dacă acestea sunt valide, va răspunde cu un token de acces.

3.2.1 Cererea tokenului de acces

Clientul face un request la server prin adăugarea următorilor parametri folosind formatul „application/x-www-form-urlencoded” cu encodarea UTF-8 in body-ul cererii HTTP:

- **grant_type**

  NECESAR. valoarea trebuie setată „*password*”

- **username**

  NECESAR. username-ul proprietarului de resursă

- **password**

  NECESAR. parola proprietarului de resursă

- **scope**

  OPTIONAL. domeniile de acces

De exemplu, un client face următoarea cerere la serverul de autorizare:

```
POST /oauth/access_token HTTP/1.1
     Host: api.watchr.com
     Authorization: Basic sadSALJasdjasdaljASJSASKASLS
     Content-Type: application/x-www-form-urlencoded

     grant_type=password&username=tudorgk&password=sfkl124
```

Serverul de autorizare trebuie să:

- autentifice clientul dacă autentificarea clientului este inclusă;
- valideze parola proprietarului de resursă folosind algoritmul existent de validare a parolei.

Pentru că tokenul de acces folosește parola utilizatorului, serverul de autorizare trebuie să fie protejat împotriva atacurilor de tip brute force (de ex., generarea unor alerte sau limitarea benzii).

3.2.2 Răspunsul tokenului de acces

Dacă cererea tokenului de acces este validă și autorizată, serverul de autorizare emite un token de acces și opțional un token de refresh. Dacă cererea este invalidă sau nu a reușit să se autentifice, va returna un răspuns de eroare.

Un exemplu de răspuns valid este:

```
HTTP/1.1 200 OK
     Content-Type: application/json;charset=UTF-8
     Cache-Control: no-store
     Pragma: no-cache

     {
       "access_token":"sx5QOdq3YqvnlADuTcaKD3ml98GTXK7H1wnLrebq",
       "token_type":"bearer",
       "expires_in":3600,
       "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIAvnlADuTcaKD3ml98GTXK",
     }
```

Capitolul IV

Aplicația Client

watchr iOS app

4.1 Prezentare de ansamblu

Aplicația iOS *watchr* este cea de-a doua componenta principală a rețelei de socializare și de raportare a problemelor comunitare. O aplicație mobila este un program creat special pentru dispozitive mobile precum smartphone-uri și tablete. Termenul de „app” este o prescurtare a termenului „application software”.

Aplicațiile mobile au fost inițial concepute pentru productivitate și consumarea informațiilor generale cum ar fi citirea emailurilor, accesarea calendarului, folosirea contactelor și accesarea informațiilor despre vreme și a bursei de valori. Cu toate acestea, cererea generată de câți mai mulți utilizatori și disponibilitatea instrumentelor foarte puternice de dezvoltare au condus la expansiunea rapidă în alte categorii, cum ar fi jocuri pentru telefoane mobile, automatizări industriale, GPS și servicii de localizare, soluții bancare, achiziții de bilete și multe altele. Această explozie în numărul și varietatea app-urilor au făcut descoperirea serviciilor noi o provocare, care la rândul ei a adus la crearea unor modele de business cu totul și cu totul unice și inovatoare.

Popularitatea aplicațiilor mobile a continuat să crească, iar utilizarea acestora a devenit tot mai răspândită în toata domeniile. Un studiu comScore din mai 2012 a raportat că în trimestrul anterior, mai mulți abonați au folosit aplicațiile mobile in locul browser-ului de pe dispozitivul mobil: 51,1% vs. 49.8% [13]. Cercetătorii au descoperit că utilizarea de aplicații mobile se corelează puternic cu contextul utilizatorului, cu poziția acestuia și momentul zilei. [14]

Aplicația *watchr* reprezintă panoul de comandă al întregului sistem. Cu ajutorul acesteia utilizatorii pot introduce și modifica evenimente noi, comenta asupra acestora și pot filtra anumite date în funcție de preferințe. Fiecare utilizator este liber să folosească aplicația cum dorește, fie doar să ocupe rolul de observator al comunității, fie să introducă date noi in sistem.

În subcapitolele următoare voi prezenta designul aplicației, deciziile importante luate în proiectarea sa și implementarea unor caracteristici cheie.

4.1.1 Dezvoltarea pe terminale mobile

Dezvoltarea aplicațiilor pe terminale mobile este una foarte complexa deoarece trebuie luate în considerare constrângerile și funcționalitățile acestor dispozitive. Dispozitivele mobile nu au o sursă de energie infinită și au procesoare mult mai slabe în comparație cu calculatoarele personale. Ele au și elemente diferite, cum ar fi sisteme de localizare și camere video. Programatorul trebuie să ia în considerare numărul foarte mare de ecrane cu rezoluții și formate diferite, specificații hardware diferite pentru fiecare model cât și schimbările ce au loc la fiecare actualizare a platformei.

Programarea pentru aplicații mobile necesită utilizarea unor medii de dezvoltare integrate specializate. Pentru dezvoltarea aplicației *watchr* s-a folosit IDE-ul XCode 5.4.

Ca parte a procesului de dezvoltare, **Mobile User Interface (UI) Design** este, de asemenea, o componentă esențială în implementarea unei aplicații. UI-ul pentru dispozitive mobile trebuie să țină cont de constrângeri și contexte, dimensiunile ecranului, tipurile de input și mobilitatea. Utilizatorul este deseori cel mai important element când vine vorba de interacțiunea sa cu device-ul. Interfața are elemente atât software cât și hardware. Inputurile utilizatorului pot manipula sistemul, iar outputul reflectă efectele acestor evenimente. În general, obiectivul unui design de UI pentru terminale mobile este cel de a oferi o interfață user-friendly, să minimizeze intrările de la tastatură, de a fi ușor de înțeles și de a oferi cât mai multe opțiuni folosind un set minimal de instrucțiuni.

4.1.2 Tehnologii folosite

Aplicația *watchr* este creată special pentru platforma iOS. iOS este sistemul de operare dezvoltat de compania Apple și este distribuit exclusiv hardware-ului Apple. Acesta este prezent pe iPhone, iPad, iPod Touch si Apple TV.

Interfața platformei iOS este bazată pe conceptul de manipulare directă, folosind capabilitățile multi-touch. Interfața consta în elemente de control precum butoane, slidere și switchuri. Terminalele au diferite componente hardware speciale, spre exemplu GPS și accelerometru, ce schimba modul de interacțiune. Aplicația folosește poziția utilizatorului pentru a oferi informații relevante la un moment specific în timp și spațiu.

Cocoa Touch

iOS împărtășește majoritatea framework-urilor cu OS X, sistemul de operare desktop dezvoltat de Apple. Core Foundation și Foundation sunt unele din aceste framework-uri. Totuși, platforma mobilă folosește Cocoa Touch, ce reprezintă un framework de UI creat special pentru ecrane tactile și terminale mobile. Cocoa Touch oferă un strat de abstractizare al sistemului de operare și este bazat pe toolsetul bibliotecii Cocoa din OS X. Frameworkul este scris in limbajul proprietar Apple, Objective-C. Cocoa Touch permite utilizarea hardware-ului și a caracteristicilor device-ului. La fel ca frameworkul Cocoa pe OS X, Cocoa Touch urmează aceeași arhitectură software MVC.

Xcode

Xcode este un IDE ce conține suita tool-uri pentru dezvoltarea software implementate de Apple. Suita Xcode include majoritatea tehnologiilor pentru dezvoltarea pe dispozitive Apple, dar conține și Interface Builder, o aplicație folosita pentru construirea interfețelor grafice.

Versiunea folosită pentru implementarea aplicației *watchr* este 5.1.1 ce oferă suport pentru iOS 7.1 SDK și un compilator LLVM (Low Level Virtual Machine) pe 64 de biți.

4.2 Designul aplicației *watchr*

În următoarele subcapitole vom discuta despre una dintre cele mai importante etape din dezvoltarea unei aplicații mobile, și anume **designul**. În acest stagiu se va analiza audiența, se vor concepe cazuri de uz și se vor aprofunda detaliile interacțiunii.

4.2.1 Mock-up

Aplicația *watchr* a fost concepută să ruleze numai pentru platforma Apple, dar designul nu este constrâns de acest aspect. În Fig. 10 avem primele mock-up-uri ale interfeței. Bineînțeles nu toate elementele au fost incluse în implementarea finală. În conceperea sa s-a pus accent pe claritatea informațiilor afișate. S-a căutat eliminarea informațiilor care duceau la încărcarea interfeței, fapt ce ducea la scăderea lipsei de interes al utilizatorului.

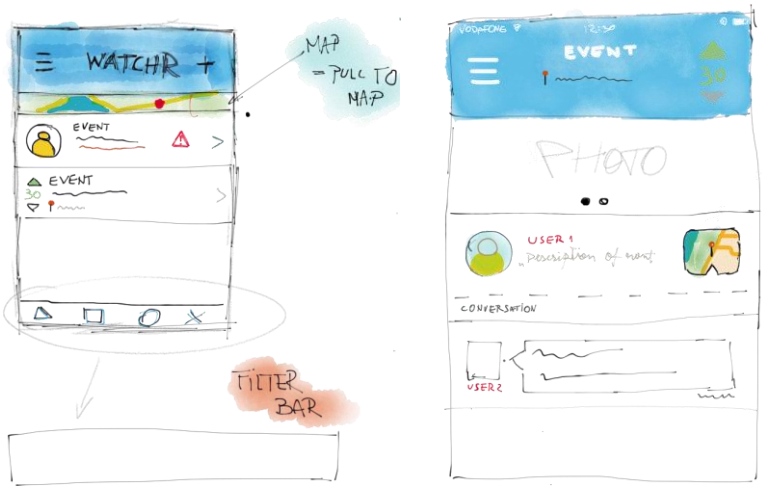

Fig. 10 Mock-up aplicație

Culorile au un rol foarte important în imersiunea utilizatorului în aplicație. Au fost alese culori vibrante pentru a capta atenția user-ului. Se folosește un font subțire, specific schițelor impuse de Apple. prin folosirea unui font subțire și modern se sporește claritatea și lizibilitatea textului. Iconițele sunt precise și luminoase, iar ornamentele de design sunt subtile și adecvate.

Animațiile poartă un rol la fel de important, ele oferind utilizatorului sentimentul de tranziție dintre o stare în alta. Ele ajută la înțelegerea poziției în ierarhie ce se modifică odată cu navigarea printre diferite view-uri.

În continuare vom trata interfața aplicației de iOS, deciziile luate pentru a îmbunătăți **User Experience**-ul și navigarea între view-uri.

4.2.2 Welcome screen-ul

Orice aplicație pornește cu un *welcome/splash screen*. Acest lucru este necesar pentru stabilirea primelor informații, cum ar fi folosirea contului care va accesa serviciul sau înregistrarea unui nou utilizator. În Fig. 11 avem welcome screenu-ul. După cum se poate observa, utilizatorul nu este logat în aplicație și are doar două opțiuni: cea de **Login** și cea de **Register.** Daca aplicația deține deja un cont activ, utilizatorului îi va fi prezentat **Feed**-ul direct.

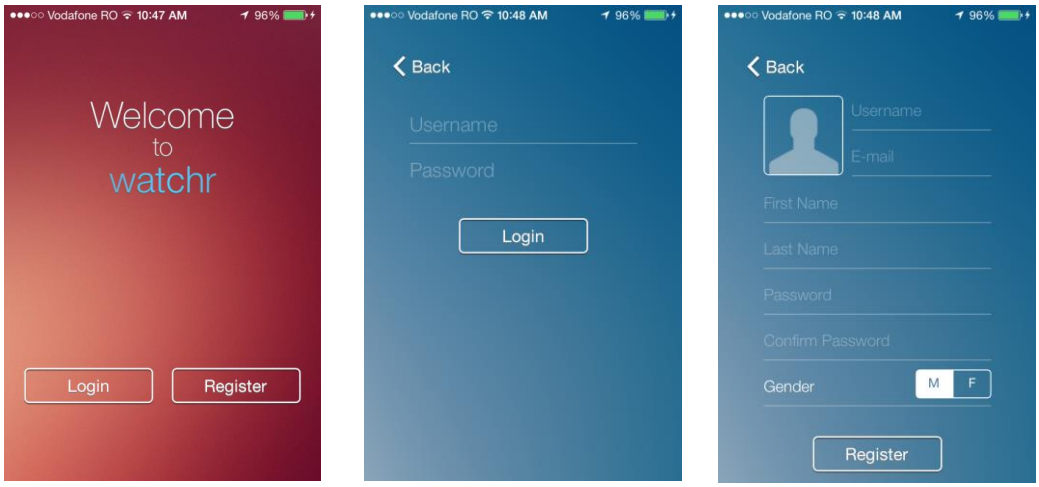

Fig. 11 Welcome Screen cu view-ul de "login" și "register"

4.2.3 Watchr Feed

Feed-ul este una dintre cele mai importante componente ale aplicației *watchr*. În aceasta lista scrollabilă sunt afișate toate evenimentele care respectă filtrele stabilite de utilizator. Observăm că în Fig. 12 sunt afișate evenimentele care sunt cuprinse în raza de 20 de km, iar acestea sunt ordonate după rating în ordine descrescătoare.

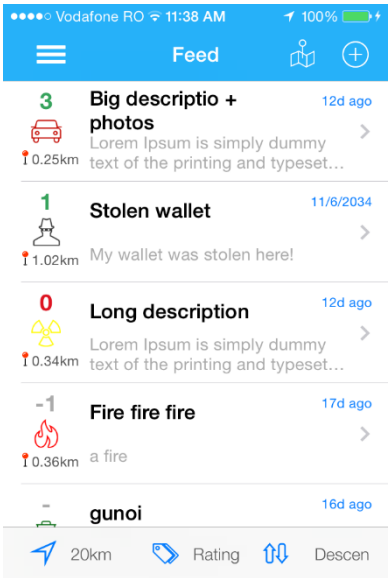

Fig. 12 Ecranul de Feed

Schițele Apple impun o structură atunci când vine vorba despre navigarea prin aplicație. Această structură este dată de Bara de navigare prezentată în Fig. 13. Oamenii tind să nu fie conștienți de experiența de navigare dintr-o aplicație, decât daca aplicația nu le satisface așteptările lor. Navigarea trebuie implementată într-un mod în care susține structura și scopul aplicației, fără sa atragă atenție asupra navigării în sine.

Navigarea

În general există trei stiluri principale de navigare, fiecare dintre ele fiind potrivită pentru o structura specifică:

- ierarhică
- plată
- content- sau experience-driven

Aplicația *watchr* are o structură ierarhică. Utilizatorul navighează făcând o alegere pe ecran pană când ajunge la destinație. Pentru a naviga la alta destinație, utilizatorul trebuie să reconstituie o parte din pași sau să revină la început și să facă alegeri diferite. Acest lucru se poate observa în **Feed.** Utilizatorul alege un eveniment și va fi dus la detaliile acestuia.

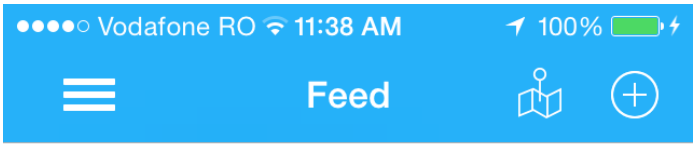

Fig. 13 Bara de navigare

În aplicația *watchr* se folosește o bara de navigare (Fig. 13) pentru a oferi utilizatorilor o modalitate ușoara de a traversa o ierarhie de date. Titlul oferă informații utilizatorilor despre poziția lor în ierarhie; butonul de „back” face ca tranziția la nivelul anterior să fie foarte ușor de realizat.

Celula din lista de evenimente

În continuare vom vorbi despre o celulă din lista de evenimente. Celula conține următoarele elemente:

- **titlul** evenimentului
- **subtitlu** cu câteva rânduri din descrierea evenimentului
- **data**, care este generată dinamic în funcție de data vizualizării
- **iconița**, care oferă o reprezentare vizuală a tipului evenimentului
- **distanța** la care se află evenimentul de poziția utilizatorului
- **ratingul,** ce este colorat diferit în funcție de ratingul acordat de utilizator

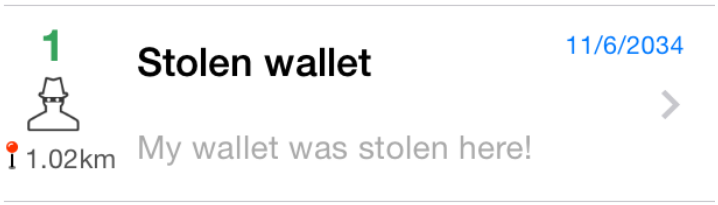

Fig. 14 Exemplu de celulă din lista de evenimente

Harta evenimentelor

Prin apăsarea butonului de afișare a harții din bara de navigație, interfața va face tranziția către harta după cum se poate vedea în Fig. 15. Pe această hartă sunt afișate toate pozițiile evenimentelor din listă. Deoarece putem avea un număr foarte mare de evenimente pe o anumită rază și trebuie să putem afișa eficient aceste locații, am optat pentru folosirea unui algoritm complex de gruparea punctelor de pe harta.

În Fig. 15 se poate observa că avem două pozații notate cu cifra 3. Daca vom face zoom-in la hartă în regiunea acelor poziții, se vor afișa alte 3 locații separate. Voi explica folosirea acestui algoritm în următoarele subcapitole.

Fig. 15 Harta evenimentelor

4.2.4 Adăugarea unui eveniment

În Fig. 16 este prezentat view-ul de adăugare a unui eveniment. View-ul are o structură de formular unde putem introduce:

- **titlul** evenimentului
- **descrierea** acestuia
- **categoria** din care face parte evenimentul (de ex. accident, inundație etc.)
- **poziția** evenimentului pe hartă
- **atașamente,** precum poze de la locul accidentului.

Utilizatorul apoi apasă butonul *Submit* și va trimite informațiile serverului *watchr.*

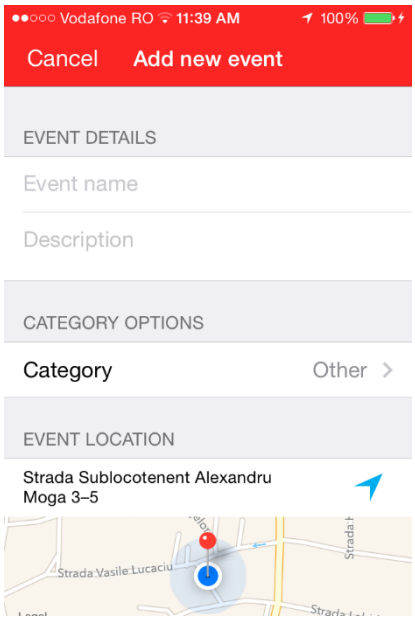

Fig. 16 Ecranul de introducere eveniment nou

4.2.5 Meniul

Meniul se accesează prin apăsarea butonului cu 3 dash-uri din colțul din stânga ale ecranului, iar în caz că acesta nu este prezent se poate trage cu degetul view-ul afișat. În meniu avem acces la **Feed**, **Locatiile mele** și **Prieteni.** În headerul meniului avem informații despre utilizatorul care este logat, notificările acestuia și cererile de prietenie.

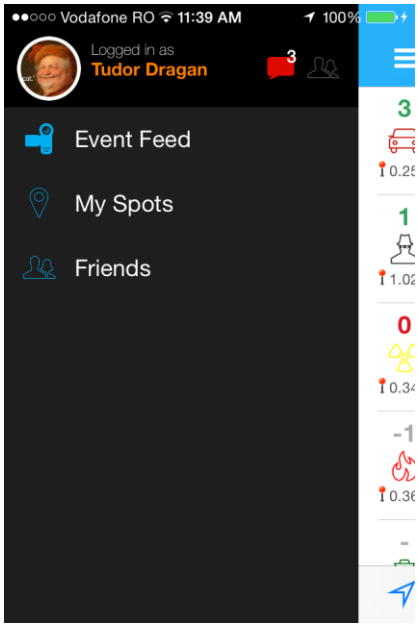

Fig. 17 Meniul

4.2.6 Detaliile unui eveniment *watchr*

Accesarea unui eveniment va duce la afișarea acestui view (Fig. 18). De aici utilizatorul are opțiunea de a afla mai multe informații despre evenimentul respectiv, de a contribui cu un rating, de a comenta și de a oferi noi informații daca dorește.

În headerul listei se află un carusel cu pozele care au fost atașate evenimentului. Sub acest header sunt afișate informații despre utilizatorul care a postat evenimentul, precum poza sa de profil, numele și prenumele și username-ul. De asemenea se folosește *reverse geocoding* pentru afișarea adresei evenimentului.

Cele 4 taburi reprezintă diferite secțiuni, fiecare având un rol important;

- **EVENT DETAILS** – sunt afișate detalii despre eveniment, de exemplu statusul evenimentului si descrierea sa
- **SHOW MAP** – va afișa poziția evenimentului pe harta
- **COMMENTS** – secțiune unde se for afișa comentariile utilizatorilor aplicației *watchr*
- **FOLLOWERS** – lista utilizatorilor care urmăresc evenimentul și sunt implicați direct prin folosirea serviciului de notificări.

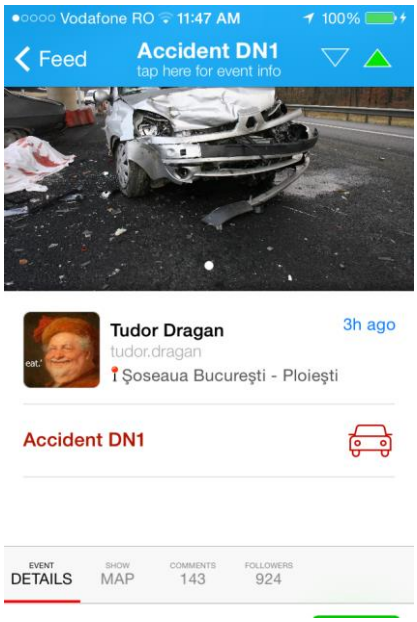

Fig. 18 Ecranul de afișare a detaliilor evenimentului

4.3 Implementarea aplicației iOS *watchr*

În subcapitolele ce urmează, vom trata structura aplicației, bibliotecile folosite, implementarea unor funcții cheie și testarea aplicației. Vom începe prin explicarea design pattern-ului folosit pentru implementarea aplicației *watchr.*

4.3.1 Design pattern MVC

Design pattern-ul **Model-View-Controller (MVC)** este esențial în proiectarea unei aplicații mobile. MVC atribuie obiectelor aplicației unul dintre cele trei roluri: *model*, *view* și *controller*. În acest pattern, modelele țin evidența datelor aplicației, view-urile reprezintă interfața (UI-ul) și construiesc conținutul aplicației, iar controllerele gestionează view-urile. Prin răspunderea la interactive utilizatorului și popularea view-urilor cu conținut, controllerele servesc ca o poartă între modele și viewuri.

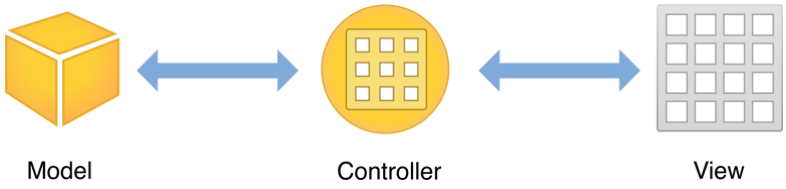

Fig. 19 Structura MVC

Obiectul Model

Un obiect model încapsulează datele specifice pentru o aplicație și definește logica care manipulează și procesează acele date. De exemplu, în aplicația *watchr* un obiect model este reprezentat de un eveniment primit de la server. Acest obiect poate avea relații one-to-many sau many-to-many cu alte obiecte model. Ideal, un obiect model nu ar trebui să aibă o confecțiune explicită cu obiectele de view, care doar prezinta datele utilizatorului.

Obiectul Model

Un obiect view este un obiect din aplicație pe care utilizatorii îl pot vedea. Un obiect view știe cum să se deseneze și poate răspunde la interacțiunile userilor. Un rol major al obiectelor view este acela de a afișa datele din obiectele model și de a permite editarea acelor date. În ciuda acestui fapt, obiectele view sunt decuplate de modelele dintr-o aplicație.

Obiectul Controller

Un obiect controller acționează ca un intermediar între una sau mai multe obiecte view cu una sau mai multe obiecte model. Obiectele controller sunt folosite în coordonarea dintre taskuri și managementul ciclului de viată a obiectelor.

4.3.2 Biblioteci open-source

În implementarea unei aplicații atât de complexe au fost nevoie de câteva biblioteci open-source pentru a finaliza proiectul într-un timp rezonabil.

OAuth2Client
https://github.com/nxtbgthng/OAuth2Client

Aceasta bibliotecă implementează profilul nativ al aplicației, ce suportă autorizarea end-userului la un serviciu ce are implementat sistemul de autentificare OAuth 2.0. În plus, această bibliotecă are suport pentru autorizarea cu username și parola pe care le folosește pentru a obține un token de acces.

InfiniteScroll
https://github.com/pronebird/UIScrollView-InfiniteScroll

Infinte scrooll ajută în organizarea unui flux "infinit" de conținut. Biblioteca ține evidența poziției utilizatorului în scroll view, iar atunci când ajunge la capătul listei, va prezenta un *activity indicator* care va declanșa un handler. Acel handler va încărca mai mult conținut de la server. În Fig. 20 este prezentat un exemplu al acestei funcționalități.

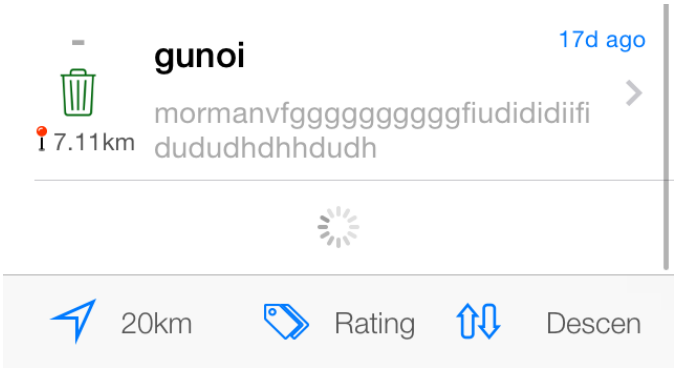

Fig. 20 Exemplu de Infinite Scroll

MRProgress
https://github.com/mrackwitz/MRProgress

MRProgress este o colecție de componente care prezintă un *blurred box overlay* cu un indicator în timp ce se execută taskuri într-un background thread. Această bibliotecă este folosită atunci când postăm un eveniment și dorim să prezentăm un feedback utilizatorului cu progresul.

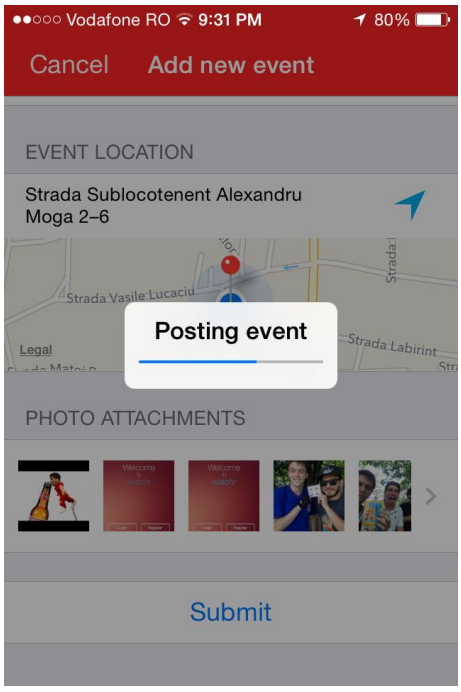

Fig. 21 Exemplu de MRProgress HUD

XYOrigami
https://github.com/xyfeng/XYOrigami

XYOrigami este un engine de tranziții între view-uri pentru platforma iOS. Acest engine este folosit pentru animarea efectului de "împăturire" a view-ului. Engine-ul oferă un element vizual ce ajută în crearea senzației de imersiune a utilizatorului.

ECSlidingViewController
https://github.com/ECSlidingViewController/ECSlidingViewController

ECSlidingViewController este un container care gestionează o interfața bazată pe straturi. Nivelul de top se ancorează pe partea din dreapta sau stânga, în timp ce afișează un view în spatele acestuia. Este cunoscut drept "Side Menu" sau "Hamburger Menu/Drawer/Sidebar". Acest container a fost folosit în implementarea meniului.

FBShimmering
https://github.com/facebook/Shimmer

Shimmer este un efect vizual ce este prezent în **Welcome Screen**. Acesta este dezvoltat de către Facebook și oferă un element în plus designului aplicației.

4.3.3 Autentificarea/Inregistrarea utilizatorului

Pentru a putea folosi funcțiile aplicației, utilizatorul trebuie să se autentifice cu serverul *watchr*. La prima pornire se va afișa **Welcome Screen-ul** unde user-ul are două opțiuni: dacă deține un cont, el va trebui sa se autentifice cu username și parola, iar daca este un utilizator nou, el va trebui sa introducă toate datele necesare pentru înregistrare.

Odată ce utilizatorul completează câmpurile, se face o cerere către API-ul watchr astfel:

```
//attempt login
[[NXOAuth2AccountStore sharedStore] requestAccessToAccountWithType : @"watchrAPI"
username:_username password:_password];
```

Pentru a primi înapoi răspunsul de la server trebuie setat un *observer* care să „asculte” dacă autentificarea a fost validă:

```
[[NSNotificationCenter defaultCenter]
addObserverForName:NXOAuth2AccountStoreAccountsDidChangeNotification
            object:[NXOAuth2AccountStore sharedStore]
             queue:nil
        usingBlock:^(NSNotification *aNotification)
                   {

                   // the account has been validated
                   // application logic for dismissing the welcome screen

                   }];
```

4.3.4 Exemplu de cerere

Toate cererile către API-ul *watchr* vor avea următoarea structură. Diferențele vor fi în apelarea URL-urilor cu parametrii necesari și tratarea răspunsului primit de la server. Se va folosi același cont cu care utilizatorul s-a logat.

```
[NXOAuth2Request performMethod:@"GET"
                     onResource:[NSURL URLWithString:[NSString
stringWithFormat:@"%@%@", TDAPIBaseURL, @"/events/active"]]
                 usingParameters:[_dashboardFilters filtersToDictionary]
                     withAccount:[[NXOAuth2AccountStore sharedStore]
accountWithIdentifier:[[NSUserDefaults standardUserDefaults] objectForKey:
TDWatchrAPIAccountIdentifier] ]
            sendProgressHandler:^(unsigned long long bytesSend, unsigned long long
bytesTotal)
                 {

                 //change the progress bar value

                 }
                responseHandler:^(NSURLResponse *response, NSData *responseData,
NSError *error)
                 {

                 //if error, display an alert view
                 //else if everything is ok, repopulate the table


                 }];
```

4.3.5 Afișarea unor cantități mari de adnotări pe harta

Deoarece putem avea un număr foarte mare de evenimente pe o anumita rază, trebuie să putem afișa eficient pozițiile lor pe hartă. Am optat pentru folosirea unui algoritm complex de grupare a punctelor de pe harta. Algoritmul a fost implementat de Theodore Calmes în limbajul C aplicație iOS [15]. Acest algoritm trebuie să fie performant pentru a nu îngreuna User Experience-ul.

Structura de date

Pentru a ajunge la o strucură de date optimă pentru această problemă, trebuie analizate datele ce vin de la server. Datele sunt reprezentate printr-un set de coordonate (latitudine, longitudine) care vor fi prezentate pe harta. Harta are posibilitatea de zoom și de mutare a regiunii, însemnând că nu toate punctele vor fi vizibile pe ecran. Trebuie găsite punctele care aparțin unei zone specificate, de exemplu un dreptunghi.

O soluție simplă ar fi iterarea prin toate punctele si interogarea coordonatelor dacă acestea respectă următoarea ecuație: xMin <= x <= xMax și yMin <= y <= yMax. Din păcate această soluție are o complexitate de O(n) și nu este optimă pentru scenariul acesta.

O abordare bună la probleme de căutare este găsirea simetriilor care vor reduce domeniul de căutare. Pentru a reduce domeniul, în loc să indexăm punctele, indexăm după regiunea în care se află acele poziții. Acest tip de indexare spațială este implementată prin folosirea unui *quad tree*, ce are o complexitate de O(h) (h fiind înălțimea arboretului la un anumit punct).

Quad Tree

Un arbore quad este o structură de date compusa din noduri care salvează un *pool* de poziții și un *bounding box*. Orice punct care aparține *bounding box*-ului nodului este adăugat în *pool*. Odată ce *pool*-ul este plin, nodul se desparte în alte 4 noduri, fiecare având propriul *bounding box* care corespunde cu un sfert din *bounding box*-ul părintelui. Toate punctele care ar încăpea în *pool*-ul părintelui, vor aparține acum de pool-ul copiilor.

Structura de date este definită astfel:

```
typedef struct TBQuadTreeNodeData {
    double x;
    double y;
    void* data;
} TBQuadTreeNodeData;
TBQuadTreeNodeData TBQuadTreeNodeDataMake(double x, double y, void* data);

typedef struct TBBoundingBox {
    double x0; double y0;
    double xf; double yf;
} TBBoundingBox;
TBBoundingBox TBBoundingBoxMake(double x0, double y0, double xf, double yf);

typedef struct quadTreeNode {
    struct quadTreeNode* northWest;
    struct quadTreeNode* northEast;
    struct quadTreeNode* southWest;
    struct quadTreeNode* southEast;
    TBBoundingBox boundingBox;
    int bucketCapacity;
    TBQuadTreeNodeData *points;
    int count;
} TBQuadTreeNode;
TBQuadTreeNode* TBQuadTreeNodeMake(TBBoundingBox boundary, int bucketCapacity);
```

Structura `TBQuadTreeNodeData` va conține coordonatele și informațiile suplimentare ale evenimentului.

Structura `TBBoundingBox` reprezintă un dreptunghi care este definit ca o regiune. Punctul din stânga sus este definit de valorile (xMin, yMin) iar cel din dreapta jos (xMax, yMax).

Structura `TBQuadTreeNode` conține patru pointeri. Fiecare pointer este o referință la cuadrantul său. Ea conține pool-ul și granița.

Crearea arborelui quad folosind

Odată ce avem un array cu evenimentele de la server, se începe construirea arborelui. Codul ce face acest lucru este cel prezentat aici:

```
typedef struct WatchrEventInfo {
    char* eventName;
    char* eventDescription;
       unsigned int index;
} WatchrEventInfo;


TBQuadTreeNodeData TBDataFromObject(NSDictionary *object, unsigned int index){
```

```
    NSDictionary * position = [object objectForKey:@"position"];

    //get event coordinates
    double latitude = [[position objectForKey:@"latitude"] doubleValue];
    double longitude = [[position objectForKey:@"longitude"] doubleValue];

    //initialize event info
    WatchrEventInfo* eventInfo = malloc(sizeof(WatchrEventInfo));

    //get the event's name
    NSString * eventName = [object objectForKey:@"event_name"];
    if (eventName == nil || [eventName isKindOfClass:[NSNull class]])
       eventName = @"Unnamed event";

    eventInfo->eventName = malloc(sizeof(char) * eventName.length + 1);
    strncpy(eventInfo->eventName, [eventName UTF8String], eventName.length + 1);

    //get the event's description
    NSString * eventDescription = [object objectForKey:@"description"];
    if (eventDescription == nil || [eventDescription isKindOfClass:[NSNull class]])
       eventDescription = @"Description not available";

    eventInfo->eventDescription = malloc(sizeof(char) * eventDescription.length +
1);
    strncpy(eventInfo->eventDescription, [eventDescription UTF8String],
eventDescription.length + 1);

    //set the event index
    eventInfo->index = index;

    return TBQuadTreeNodeDataMake(latitude, longitude, eventInfo);
}


-(void) buildTreeWithArray:(NSArray *)array{
       @autoreleasepool {
              //get the data for the annotations;
              TBQuadTreeNodeData *dataArray = malloc(sizeof(TBQuadTreeNodeData) *
[array count]);
              for (unsigned int i =0; i<[array count]; i++) {
                     NSDictionary * eventData =[array objectAtIndex:i];
                     dataArray[i] = TBDataFromObject(eventData, i);
              }

              //Change it to Romania
              TBBoundingBox world = TBBoundingBoxMake(44, 20, 49, 30);
         _root = TBQuadTreeBuildWithData(dataArray, [array count], world, 4);

       }
}
```

Acum arborele este alocat și populat cu datele necesare; următorul pas este *clustering*-ul.

Clustering

Arborele este acum populat cu datele evenimentelor, trebuie rezolvată problema clustering-ului. Deoarece nu se dorește afișarea tuturor punctelor pe hartă în același timp, trebuie luată în considerare gruparea locațiilor pe clustere. Sunt mai multe metode prin care se poate face acest lucru. Metoda aleasă aici este cea care depinde de nivelul de zoom. În funcție de nivelul de zoom, adnotările se despart pentru a afișa evenimentele pe hartă. S-a ales metoda clustering-ului pentru că permite afișarea unui număr mare de date, reducând de puncte de pe hartă.

```
- (NSArray *)clusteredAnnotationsWithinMapRect:(MKMapRect)rect
withZoomScale:(double)zoomScale
{
    double TBCellSize = TBCellSizeForZoomScale(zoomScale);
    double scaleFactor = zoomScale / TBCellSize;

    NSInteger minX = floor(MKMapRectGetMinX(rect) * scaleFactor);
    NSInteger maxX = floor(MKMapRectGetMaxX(rect) * scaleFactor);
    NSInteger minY = floor(MKMapRectGetMinY(rect) * scaleFactor);
    NSInteger maxY = floor(MKMapRectGetMaxY(rect) * scaleFactor);

    NSMutableArray *clusteredAnnotations = [[NSMutableArray alloc] init];

    for (NSInteger x = minX; x <= maxX; x++) {
        for (NSInteger y = minY; y <= maxY; y++) {

            MKMapRect mapRect = MKMapRectMake(x / scaleFactor, y / scaleFactor, 1 /
            scaleFactor, 1.0 / scaleFactor);

            __block double totalX = 0;
            __block double totalY = 0;
            __block int count = 0;

            TBQuadTreeGatherDataInRange(self.root,
TBBoundingBoxForMapRect(mapRect), ^(TBQuadTreeNodeData data) {
                totalX += data.x;
                totalY += data.y;
                count++;
          });

            if (count >= 1) {
                CLLocationCoordinate2D coordinate =
CLLocationCoordinate2DMake(totalX / count, totalY / count);
                TBClusterAnnotation *annotation = [[TBClusterAnnotation alloc]
initWithCoordinate:coordinate count:count];
                [clusteredAnnotations addObject:annotation];
            }
        }
    }

    return [NSArray arrayWithArray:clusteredAnnotations];
}
```

Adăugarea adnotărilor necesare

Din considerente de performanță, se dorește folosirea main thread-ului cât mai puțin, ce înseamnă că toate metodele de manipulare a arborelui *quad* trebuie apelate într-o coadă de operații din background. Pentru a reduce timpul petrecut pe *main thread* trebuie să fim siguri că se vor desena doar adnotările care sunt necesare. Astfel se vor evita blocuri inutile atunci când se navighează pe hartă și asigură o experiența fără întreruperi.

Screenshot-ul din stânga este un snapshot al harții înainte ca aceasta să-și schimbe regiunea. Adnotările de pe hartă sunt exact cele prezente în mapView-ul curent. Acest set se numește setul *before.*

Screenshot-ul din dreapta este un snapshot al harții imediat după ce regiunea s-a modificat. Vrem să păstram o parte din adnotările anterioare. Se dorește păstrarea adnotărilor care aparțin atât setului *before* cat și setului *after*. Se va face intersecția acestora.

```
- (void)updateMapViewAnnotationsWithAnnotations:(NSArray *)annotations
{
    NSMutableSet *before = [NSMutableSet setWithArray:self.mapView.annotations];
    NSSet *after = [NSSet setWithArray:annotations];

    // Annotations circled in blue shared by both sets
    NSMutableSet *toKeep = [NSMutableSet setWithSet:before];
    [toKeep intersectSet:after];

    // Annotations circled in green
    NSMutableSet *toAdd = [NSMutableSet setWithSet:after];
    [toAdd minusSet:toKeep];

    // Annotations circled in red
    NSMutableSet *toRemove = [NSMutableSet setWithSet:before];
    [toRemove minusSet:after];

    // These two methods must be called on the main thread
    [[NSOperationQueue mainQueue] addOperationWithBlock:^{
        [self.mapView addAnnotations:[toAdd allObjects]];
        [self.mapView removeAnnotations:[toRemove allObjects]];
    }];
}
```

Astfel se asigură evitarea computațiilor inutile pe *thread-ul* principal și va îmbunătăți performanța. Următorul pas este cel de desenare a punctelor pe hartă pentru a afișa numărul de evenimente care aparțin unui cluster.

Desenarea adnotărilor

Pentru că se reduce numărul de puncte pe care un utilizator poate să le vadă, trebuie ca fiecare dintre acele puncte să reflecte numărul de evenimente pe care le acoperă.

Metodele de desenare a adnotărilor sunt prezentate mai jos:

```
- (void)setupLabel
{
    _countLabel = [[UILabel alloc] initWithFrame:self.frame];
    _countLabel.backgroundColor = [UIColor clearColor];
    _countLabel.textColor = [UIColor whiteColor];
    _countLabel.textAlignment = NSTextAlignmentCenter;
    _countLabel.shadowColor = [UIColor colorWithWhite:0.0 alpha:0.75];
    _countLabel.shadowOffset = CGSizeMake(0, -1);
    _countLabel.adjustsFontSizeToFitWidth = YES;
    _countLabel.numberOfLines = 1;
    _countLabel.font = [UIFont boldSystemFontOfSize:12];
    _countLabel.baselineAdjustment = UIBaselineAdjustmentAlignCenters;
    [self addSubview:_countLabel];
}

- (void)setCount:(NSUInteger)count
{
    _count = count;

    CGRect newBounds = CGRectMake(0, 0, roundf(44 * TBScaledValueForValue(count)),
roundf(44 * TBScaledValueForValue(count)));
    self.frame = TBCenterRect(newBounds, self.center);

    CGRect newLabelBounds = CGRectMake(0, 0, newBounds.size.width / 1.3,
newBounds.size.height / 1.3);
    self.countLabel.frame = TBCenterRect(newLabelBounds, TBRectCenter(newBounds));
```

```
    self.countLabel.text = [@(_count) stringValue];

    [self setNeedsDisplay];
}

- (void)drawRect:(CGRect)rect
{
    CGContextRef context = UIGraphicsGetCurrentContext();

    CGContextSetAllowsAntialiasing(context, true);

    UIColor *outerCircleStrokeColor = [UIColor colorWithWhite:0 alpha:0.25];
    UIColor *innerCircleStrokeColor = [UIColor whiteColor];
    UIColor *innerCircleFillColor = [UIColor colorWithRed:0.87 green:0.08 blue:0.13
alpha:1];

    CGRect circleFrame = CGRectInset(rect, 4, 4);

    [outerCircleStrokeColor setStroke];
    CGContextSetLineWidth(context, 5.0);
    CGContextStrokeEllipseInRect(context, circleFrame);

    [innerCircleStrokeColor setStroke];
    CGContextSetLineWidth(context, 4);
    CGContextStrokeEllipseInRect(context, circleFrame);

    [innerCircleFillColor setFill];
    CGContextFillEllipseInRect(context, circleFrame);
}
```

Astfel se încheie modul de afișare a evenimentelor pe harta. Utilizatorul va putea folosit aplicația într-un mod eficient, fără sa aibă parte de scăderi în performanțe la randare.

Capitolul V

Evaluare și testare

Aplicația *watchr* a fost testată folosind suita de instrumente dezvoltată de către Apple. În Fig. 22 este prezentat graficul de alocări pentru obiectele folosite în aplicație. După cum se poate observa, aplicația nu trece de pragul critic și se află în limitele normale a unei aplicații de acest tip.

Totuși, au existat doua *leak-uri de memorie* pe parcursul rulării, dar acestea au fost date de *garbagge collector* care nu a reușit sa dealoce memoria în timp rezonabil. *Framerate*-ul este unul stabil între 40 și 60 de FPS. Consumul de energie este unul normal.

Testarea a fost făcută pe un iPhone 4S cu sistemul de operare iOS 7.1.1.

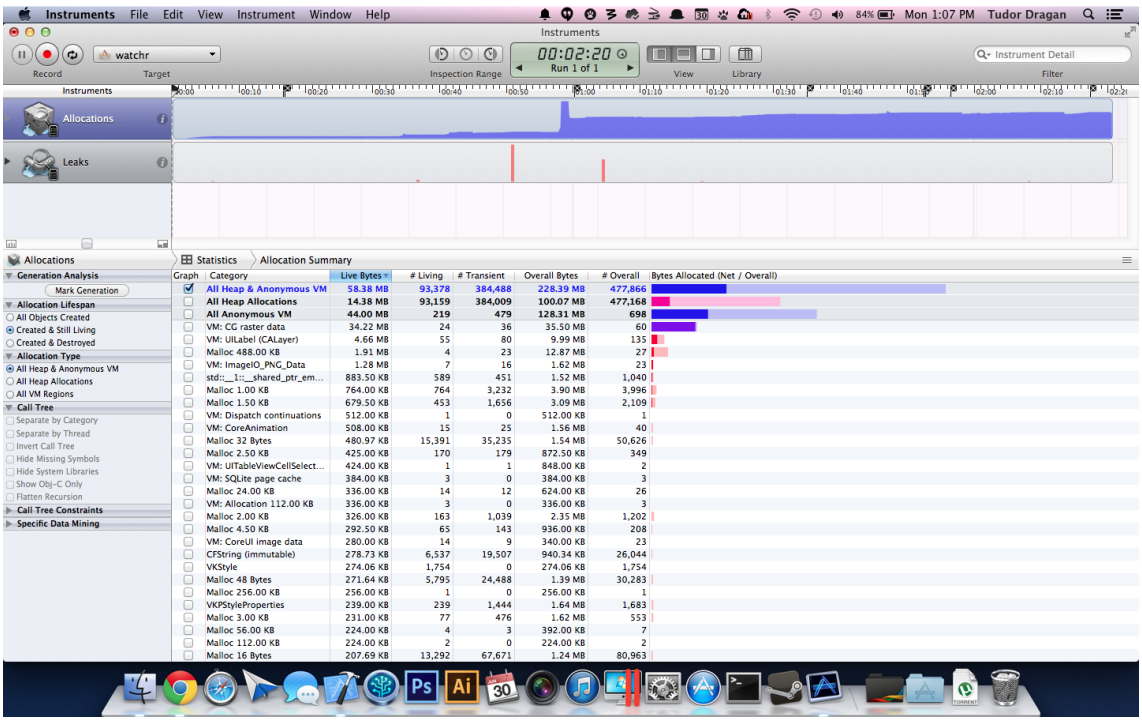

Fig. 22 Screenshot cu memory usage-ul aplicației

Capitolul VI

Îmbunătățiri și planuri viitoare

Serviciul *watchr* este o soluție complexă cu funcționalități multiple. Bazele sale au fost incluse în această lucrare, iar dezvoltarea lui nu se termină aici. **Core Service-ul** și **App-ul watchr** vor fi extinse cu alte caracteristici ce vor fi incluse în viitoarele versiuni.

6.1 Îmbunătățiri la Core Service

6.1.1 Dezvoltarea unui server de Push-Notification

Serviciul, momentan, nu oferă suport pentru *Push-Notifications*. **APNS (Apple Push Notification Service)** este un serviciu creat de Apple care îi permite serverului să comunice cu aplicația atunci când aceasta nu este pornită pe device. Serviciul folosește *tehnologia de push* ce menține o conexiune directă și constantă cu serverele Apple, care la rândul lor trimit mai departe mesajele serverului *watchr.*

Dacă aplicația nu este deschisă atunci când notificarea este primită, aplicația va afișa un *badge* în coltul iconiței, iar notificarea va apărea în Notification Center.

6.1.2 Suport pentru fișiere video

În momentul de fată, serviciul *watchr* suportă doar atașamente de tip foto (JPEG, PNG etc). Deoarece popularitatea altor rețele sociale care folosesc atașamente video a crescut, este necesar ca și rețeaua prezentată în această lucrare să suporte atașamente de tip video.

Aceste atașamente video oferă mult mai multe informații despre evenimentul care are loc în comunitate și scoate în evidenta detalii pe care celelalte medii (text & foto) nu o pot face.

6.1.3 Posibilitatea utilizatorilor sa urmărească un eveniment

În prezent, utilizatorul are opțiuni limitate când vine vorba de interacțiunea cu un anumit eveniment. În viitoarele iterații, serviciul *watchr* va suporta acțiunea de „Follow”. Acest funcționalitate este împrumutat de la rețeaua de socializare Twitter și presupune ca utilizatorul să fie în permanență anunțat la orice modificare în pagina evenimentului respectiv. Aceste pot modificări pot fi: adăugarea de noi atașamente, schimbarea statusului, un utilizator care a comentat la un eveniment, șmd.

El va fi în permanență implicat în evenimentele pe care îl interesează și poate să fie mai bine informat în legătura cu desfășurarea acestora.

6.1.4 Implementarea unor grupuri speciale

În viitor, serviciul va suporta grupuri speciale ce vor fi compuse din utilizatori cu profesii speciale, de exemplu un grup de medici voluntari sau de polițiștii comunitari. Ei vor avea posibilitatea să afle informații despre evenimentele ce se petrec în jurul lor direct de la instituții și vor avea opțiunea de a interveni sau nu.

6.1.5 Sistem de promovare si recompensare

Viitoarele iterații ale serviciului *watchr* vor conține modulul de *achievement-uri.* Utilizatorii vor fi recompensați pentru că folosesc serviciu prin diferite medalii și embleme virtuale. În cele mai recente studii s-a observat că o creștere masivă a utilizării unui serviciu web este dat în mare parte de aceste recompense. Utilizatorii pot câștiga diferite „titluri" virtuale prin folosirea aplicației și prin ajutarea comunității.

6.2 Îmbunătățiri *watchr* app

6.2.1 Suport pentru redarea și înregistrarea fișierelor video

Aplicația actuală suportă doar afișarea elementelor foto. Odată cu implementarea suportului pentru fișiere video, se vor face modificările necesare pentru înregistrarea și redarea fișierelor video pe device.

6.2.2 Geo-fencing

Geo-fencing este un perimetru virtual pentru o arie geografică reală. Un geo-fence poate fi generat dinamic în jurul unui eveniment, iar orice device care se află în acel perimetru va primi notificări din partea serverului. Utilizatorii vor fi mai implicați, iar serviciul va fi mai accesibil.

6.2.3 Tratarea notificărilor

Momentan aplicația nu oferă suport pentru tratarea notificărilor deoarece serverul nu are implementat acest modul. Odată ce serverul va putea trimite notificări către device-uri, aplicația va fi modificată în prealabil.

6.2.4 Relații de prietenie si suport pentru comment-uri

Menționând ca pe serverul *watchr* sunt deja implementate aceste module, aplicația nu oferă încă suport pentru ele. Obiectivul versiunii curente a fost să ofere fără probleme experiența de bază al serviciului. Includerea modulelor instabile în această versiune nu aduceau decât frustrarea utilizatorilor, lucru ce trebuie evitat.

Capitolul VII

Concluzie

Acest proiect nu este doar o implementare client-server. El definește o rețea specială, un grup de oameni ce doresc să contribuie și să ajute comunitatea din care fac parte. *Watchr* este un serviciu ușor de folosit și cu potențial de răspândire foarte ridicat, dacă urmărim trendurile rețelelor sociale din ziua de astăzi. Folosirea aplicației mobile va duce la creșterea numărului utilizatorilor și, automat, la creșterea întregii rețele.

Proiectul acesta a expus doar bazele rețelei. Funcționalitățile enumerate în Capitolul VI sunt deja în curs de implementare și voi participa în continuare în mod activ la creșterea rețelei pentru o comunitate deschisă, frumoasă și vie.

Bibliografie

[1] PewResearch, „Smartphone Ownership," 5 Iunie 2013. [Interactiv]. Available: http://www.pewinternet.org/2013/06/05/smartphone-ownership-2013/.

[2] Netcraft, „Web Server Survey," Iunie 2013. [Interactiv]. Available: http://news.netcraft.com/archives/2013/06/06/june-2013-web-server-survey-3.html.

[3] DB-Engines, „DB-Engines Ranking," Iulie 2013. [Interactiv]. Available: http://db-engines.com/en/ranking.

[4] Apple Inc., „iOS Dev Center," [Interactiv]. Available: https://developer.apple.com/devcenter/ios/index.action.

[5] eHow, „Apache Web Server Advantages," Mai 2014. [Interactiv]. Available: http://www.ehow.com/list_6292494_apache-server-advantages.html.

[6] Novell , „NW 6.5 SP8: Novell MySQL Administration Guide," [Interactiv]. Available: http://www.novell.com/documentation/nw65/web_mysql_nw/data/aj5bj52.html.

[7] PHP Foundation, „Cryptography Extensions," 2013. [Interactiv]. Available: http://www.php.net/manual/en/book.mcrypt.php.

[8] ISO, „ISO/IEC 5218:2004," 2004. [Interactiv]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=36266.

[9] R. Fielding, „Representational State Transfer (REST)," UC Irvine, Irvine, 2000.

[10] Ollie, „Nearest-location finder for MySQL," Plum Island Media, 12 Martie 2014. [Interactiv]. Available: http://www.plumislandmedia.net/mysql/haversine-mysql-nearest-loc/.

[11] I. E. T. F. (IETF), „The OAuth 2.0 Authorization Framework," IETF, Octombrie 2012. [Interactiv]. Available: http://tools.ietf.org/html/rfc6749.

[12] L. Degasperi, „PHP OAuth 2.0 Server for Laravel," 2013. [Interactiv]. Available: https://github.com/lucadegasperi/oauth2-server-laravel.

[13] S. Perez, „comScore: In U.S. Mobile Market, Samsung, Android Top The Charts; Apps Overtake Web Browsing," TechCrunch, 2 Iulie 2012. [Interactiv]. Available: http://techcrunch.com/2012/07/02/comscore-in-u-s-mobile-market-samsung-android-top-the-charts-apps-overtake-web-browsing/.

[14] B. H. J. S. A. K. a. G. B. Matthias Böhmer, „Falling asleep with Angry Birds, Facebook and Kindle: a large scale study on mobile application usage," 2011. [Interactiv]. Available: http://dl.acm.org/citation.cfm?doid=2037373.2037383.

[15] T. Calmes, „How To Efficiently Display Large Amounts of Data on iOS Maps," Toughbot, 1 Noiembrie 2013. [Interactiv]. Available: http://robots.thoughtbot.com/how-to-handle-large-amounts-of-data-on-maps.

[16] M. Hillyer, „MANAGING HIERARCHICAL DATA IN MYSQL," [Interactiv]. Available: http://mikehillyer.com/articles/managing-hierarchical-data-in-mysql/.