UNIVERSITATEA POLITEHNICA BUCUREȘTI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

|  |  |
| --- | --- |
| upb | cs |

PROIECT DE DIPLOMĂ

Get Together

O unealtă socială pentru interacțiuni reale

Radu-Ovidiu Catrangiu

**Coordonator științific:**

Conf. dr. ing. Alexandru Boicea

BUCUREŞTI

2019

UNIVERSITY POLITEHNICA OF BUCHAREST

FACULTY OF AUTOMATIC CONTROL AND COMPUTERS

COMPUTER SCIENCE DEPARTMENT

|  |  |
| --- | --- |
| upb | cs |

DIPLOMA PROJECT

Get Together

A social tool for real world interactions

Radu-Ovidiu Catrangiu

**Thesis advisor:**

Conf. dr. ing. Alexandru Boicea

BUCHAREST

2019

**CUPRINS**

[Sinopsis 2](#_Toc11780735)

[Abstract 2](#_Toc11780736)

[Introducere 3](#_Toc11780737)

[1.1 Context 3](#_Toc11780738)

[1.2 Problema 3](#_Toc11780739)

[1.3 Obiective 3](#_Toc11780740)

[1.4 Structura lucrării 4](#_Toc11780741)

[2 Analiza și specificarea cerințelor 5](#_Toc11780742)

[3 Studiu de piață / Abordări existente 6](#_Toc11780743)

[4 Soluția propusă 7](#_Toc11780744)

[4.1 Tehnologii Folosite 7](#_Toc11780745)

[5 Detalii de implementare 9](#_Toc11780746)

[5.1 Indicații formatare tabele 9](#_Toc11780747)

[6 Studiu de caz / Evaluarea rezultatelor 10](#_Toc11780748)

[7 Concluzii 12](#_Toc11780749)

[8 Bibliografie 13](#_Toc11780750)

[9 Anexe 15](#_Toc11780751)

# Sinopsis

TODO:

Sinopsisul proiectului are rol de introducere, conținând atât o descriere pe scurt a problemei abordate cât și o enumerare sumară a rezultatelor și a concluziilor. Se recomandă ca sinopsisul să fie redactat într-un limbaj accesibil unei persoane nefamiliarizate cu domeniul, dar în același timp destul de specific pentru a oferi rapid o vedere de ansamblu asupra proiectului prezentat.

Sinopsisul proiectului va fi redactat atât în română cât și în engleză. Ca dimensiunea recomandată aceasta secțiune va avea maxim 200 de cuvinte pentru fiecare variantă. Împreună, ambele variante se vor încadra într-o singură pagină.

# Abstract

TODO:

The abstract has an introductory role and should engulf both a brief description of the issue at hand, as well as an overview of the obtained results and conclusions. The abstract should be formulated such that even somebody that is unfamiliar with the projects’ domain can grasp the objectives of the thesis while, at the same time, retaining a specificity level offering a bird’s eye view of the project.

The projects’ abstract will be elaborated in both Romanian and English. The recommended size for this section is limited to 200 words for each version. Together, both versions will fit in one page.

# Introducere

## Context

TODO:

Get Together este simultan un produs menit să intre pe piața platformelor sociale, dar și o demonstrație de utilizare a tehnologiilor noi, în așa manieră încât să ușureze munca dezvoltatorilor atunci când vine vorba despre mentenanță și adăugarea de noi funcționalități.

## Problema

Get Together încearcă să rezolve două probleme. O problemă orientată către utilizatorii aplicației, respectiv o problemă orientată către dezvoltatorii acesteia.

Prima problemă, cea socială, constă în faptul că, după o vârstă mulți oameni nu mai au destul timp să planifice ieșiri în oraș cu prietenii. De multe ori apar probleme de sincronizare din cauza programului de zi cu zi al fiecăruia și din cauza locației la care se află fiecare într-un anumit moment al zilei, iar timpul de decizie asupra unui loc de întâlnire devine din ce în ce mai mare pe măsură ce se alătură grupului mai mulți oameni. Get Together își propune să ofere o soluție pentru a evita aceste probleme legate de planificare.

A doua problemă este una de inginerie, legată de implementare. Se dorește ca proiectul să capete dimensiuni mari, cu alte cuvinte să fie folosit de cât mai multă lume, deci este necesară o arhitectură ce poate susține un volum foarte mare de cereri. În timpul dezvoltării inițiale trebuie ținut în vedere faptul că un asemenea proiect se poate schimba după lansare atât în funcție de cererile utilizatorilor cât și în funcție de alți factori, atât interni cât și externi.

## Obiective

Se încearcă rezolvarea problemei sociale prin dezvoltarea unei aplicații web, cu un design atractiv, în care utilizatorii, după crearea contului, se pot alătura unui grup unde își pot raporta, pentru fiecare zi a săptămânii, intervalul orar la care ar putea fi disponibili să se întâlnească, împreună cu locația de unde vor pleca spre întâlnire. După raportarea locațiilor un server va compila toate datele legate de locațiile utilizatorilor din grupul respectiv și va sugera un număr de localuri la care s-ar putea întâlni, împreună cu un interval orar la care poate avea loc întâlnirea. Creatorul grupului poate seta tipul locațiilor ce pot fi sugerate, având la dispoziție trei variante: cafenele, baruri sau restaurante. De asemenea, aplicația trebuie să ofere posibilitatea utilizatorilor de a comunica rapid în cadrul platformei, deci va fi implementat și un sistem de comentarii în cadrul unui grup. Pentru a se putea distinge rapid de ceilalți utilizatori în secțiunea de comentarii, un utilizator are posibilitatea de a-și încărca propria imagine de profil. Un utilizator poate fi membru a mai multor grupuri, iar pentru a nu fi nevoit să verifice fiecare grup în parte pentru a afla ultimele noutăți va fi implementat un sistem de notificări în timp real.

Pentru rezolvarea problemei de inginerie trebuie luată în calcul o arhitectură modulară, care să poată scala ușor, dar care totodată să țină costurile cât mai joase. De asemenea, se dorește dezvoltarea cât mai ușoară a componentelor necesare funcționării aplicației, astfel încât adăugarea de noi funcționalități să nu genereze probleme. Pentru realizarea acestor obiective se optează pentru o arhitectură a back-end-ului bazată pe microservicii. O astfel de arhitectură permite împărțirea aplicației pe mai multe componente independente, ce pot rula pe mașini diferite. Principalele avantaje al acestei abordări sunt posibilitatea de scalare a resurselor prin lansarea aceleiași componente pe mai multe mașini și redundanța, posibilitatea ca o componentă să răspundă la cererile altei componente identice care a întâmpinat o eroare și care trebuie să treacă prin etapele unei recuperări. Un alt avantaj îl prezintă flexibilitatea în dezvoltare. Atunci când sunt adăugate noi funcționalități unei componente sau când este adăugată o componentă nouă, nu este afectată întreaga aplicație, ci doar părți care au o oarecare legătură între ele.

## Structura lucrării

TODO: Un paragraf în care fiecare dintre secțiunile următoare este prezentată în 1-2 fraze, punând accentul pe elementele cele mai semnificative din fiecare secțiune.

# Analiza și specificarea cerințelor

TODO:

# Studiu de piață / Abordări existente

În prezent, aplicațiile și platformele de socializare sunt centrate mai mult pe utilizator ca individ. Acestea oferă posibilitatea de a încărca și posta imagini, de a comenta la respectivele imagini și de a partaja informații precum articole sau alte postări de la alte persoane. Facebook, de exemplu, oferă și posibilitatea de a crea un grup în care să-ți inviți prietenii, însă în cadrul grupului utilizatorii sunt încurajați să facă aceleași lucruri ca în spațiul public al platformei, astfel încurajând utilizatorii să petreacă cât mai mult timp în cadrul platformei, singura diferență fiind faptul că în cadrul grupului conținutul este accesibil numai membrilor. Scopul final al aplicațiilor existente, în afară de cel expus de brand, acela de a aduce împreună utilizatorii din toate colțurile lumii, este de fapt acela de a vinde spațiu pentru publicitate. De aceea, platforme precum Facebook sau Instagram îndeamnă utilizatorii să petreacă cât mai mult timp în fața unui ecran de calculator sau de telefon, pentru că traficul suplimentar justifică un preț mai mare pentru spațiul publicitar digital.

Get Together țintește să întoarcă utilizatorul dinspre mediul virtual către mediul real, ușurând planificarea întâlnirilor, stimulând astfel activitățile reale de grup. În prezent nu există o soluție, o unealtă, unde utilizatorii să își introducă date despre programul lor de zi cu zi, împreună cu locațiile aferente programului, și să primească sugestii de locații și intervale orare în care s-ar putea întâlni.

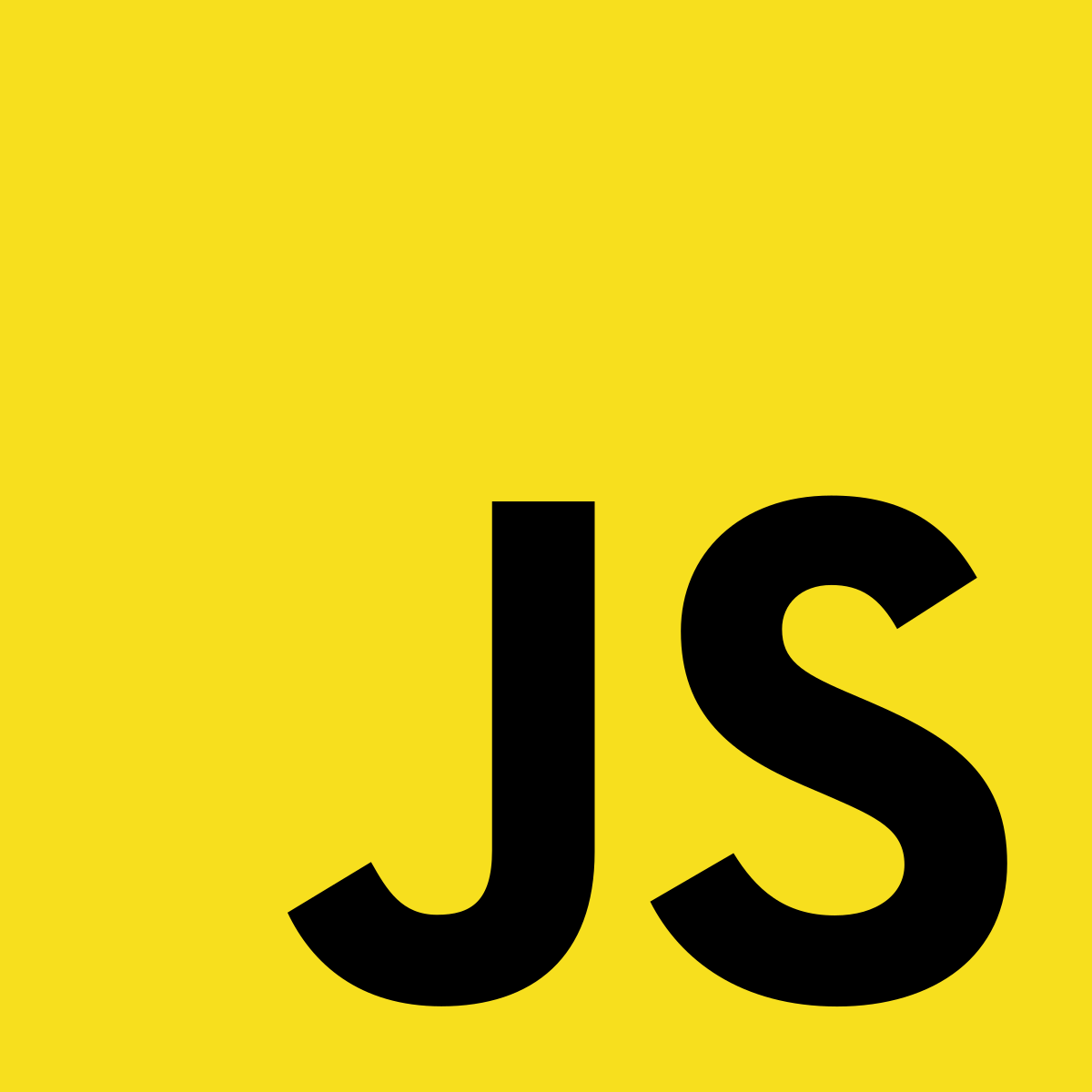
TODO:

[Dezvoltare de produs] Ce soluții similare există pe piață? Care sunt limitările lor / pentru ce cazuri de utilizare sau pentru ce tip de clienți produsele existente pe piață nu răspund cerințelor? Care sunt indicatorii pe baza cărora sunt evaluate aceste produse, de către potențiali clienți, și unde sunt lipsurile/ care este oportunitatea generată de lipsurile acestea?

# Soluția propusă

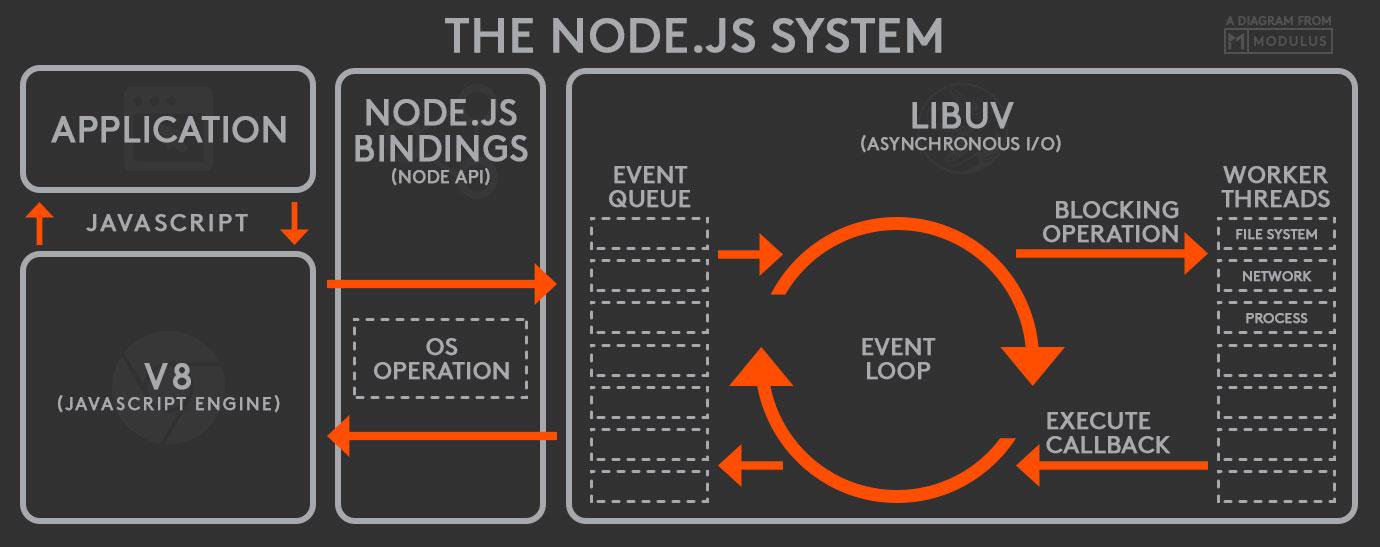
Pentru a rezolva problemele prezentate anterior vom oferi o soluție modernă, care se folosește de cele mai noi tehnologii și trenduri din programarea web. Întreg sistemul se bazează pe microservicii scrise în întregime în JavaScript, folosind Vue.JS pentru front-end, respectiv Node.js pentru back-end. Fiecare serviciu rulează independent unul de celălalt, în propriul container Docker, astfel având acces la un mediu curat, independent de mașina gazdă, pentru a avea setări și dependențe într-un singur pachet ce va rula întotdeauna fără să fie nevoie de configurări adiționale.

## Tehnologii Folosite



JavaScript este un limbaj de programare multi-paradigmă ce suportă stilul clasic de programare imperativă (precum C, C++ sau Java), programarea funcțională (precum Haskell și mai nou Java sau Scala) și nu în ultimul rând programarea bazată pe evenimente. Acest ultim tip de programare menționat joacă un rol foarte important în tehnologia din zilele noastre. Datorită programării bazate pe evenimente astăzi ne putem bucura de aplicații web reactive, cu mult diferite de paginile statice scrise doar in HTML de la începuturile internetului. Cu acest tip de programare putem adăuga acțiuni specifice pentru fiecare click din pagină, putem asculta pentru evenimente precum schimbări în pagină, mișcarea mouse-ului sau utilizarea tastaturii. Astfel ne putem bucura de interfețe grafice mult mai complexe, mai intuitive, menite să ajute utilizatorul oferind o experiență mult mai plăcută în navigarea internetului. De asemenea, dinamismul oferit de programarea bazată pe evenimente este o principală caracteristică regăsită în tehnologiile de back-end. De aceea încă din 2009, de la lansarea Node.js, JavaScript, un limbaj inițial conceput pentru front-end, a migrat și către partea nevăzută a tehnologiei web, devenind și un limbaj de back-end, astfel oferind posibilitatea dezvoltatorilor să conceapă soluții „full-stack” fără a fi nevoiți să învețe și să folosească două limbaje de programare diferite.



Node.js este un mediu de rulare pentru JavaScript menit să aducă limbajul de programare din mediul browser-ului în mediul sistemului de operare. Fiecare browser dispune de un motor de rulare pentru JavaScript, o componentă care transformă codul din script-uri în cod binar, pe care îl poate înțelege și executa procesorul. Câteva exemple de asemenea motoare de rulare sunt „SpiderMonkey” pentru Mozilla Firefox, „Chakra” pentru Microsoft Edge și, nu în ultimul rând, „V8” pentru Google Chrome. Ultimul motor de rulare menționat, „V8”, stă la baza Node.js, însă nu reprezintă în totalitate mediul de rulare oferit de Node. Așa cum browser-ul pune la dispoziție dezvoltatorului documentul paginii, sau DOM – Document Object Model, pentru a putea prelucra informațiile din pagină, împreună cu evenimentele ce au loc pe aceasta, Node.js pune la dispoziție dezvoltatorului o suită de module ce mediază accesul la componentele sistemului de operare. Câteva dintre aceste module sunt „fs” (File System) pentru accesul la hard disk și lucrul cu sistemul de fișiere, „http” sau „https” pentru accesul la placa de rețea, deschiderea porturilor de rețea sau realizarea apelurilor în internet prin protocoalele standard TCP și UDP, modulul „crypto” care pune la dispoziție dezvoltatorilor câțiva algoritmi consacrați de criptare și decriptare și multe alte module utile în funcție de proiectul pentru care sunt folosite. Node.js strălucește atunci când proiectul impune multe scrieri și citiri atât din sistemul de fișiere, cât și din rețea, însă atunci când este nevoie de calcul intens, ocuparea pentru mult timp a procesorului cu o singură problemă, se va observa un impact major asupra performanței. Acest lucru se datorează modului de funcționare a limbajului de programare și a mediului de rulare. JavaScript este un limbaj de programare ce rulează pe un singur fir de execuție, spre deosebire de alte limbaje de programare existente care se pot folosi de toate nucleele și firele de execuție ale procesorului pe care rulează programul rezultat. Ce este special la Node.js este faptul că maximizează beneficiile oferite de natura asincronă a limbajului de programare, adică de paradigma de programare pe bază de evenimente. Codul scris de programator rulează pe un singur fir de execuție, însă fiecare funcție asincronă care accesează o componentă blocantă, de intrare-ieșire, este executată de Node.js pe un fir de execuție separat, urmând ca rezultatul să fie adăugat în coada de evenimente, coadă monitorizată în permanență de firul principal de execuție. Această arhitectură avantajează programatorul deoarece nu trebuie să aibă grijă să nu blocheze firul de execuție, deci reducând complexitatea codului, devenind mai accesibil pentru noi programatori.

Un alt aspect foarte important în favoarea acestei noi tehnologii este disponibilitatea unui manager de pachete, NPM – Node Package Manager, care oferă acces la o mulțime de module open-source. Apariția NPM este principalul motiv pentru creșterea în popularitate a Node.js, în mod special datorită modulelor „express” și „request”. Aceste două module au făcut mult mai accesibilă construirea unui server și comunicarea prin protocolul HTTP. Astfel, un simplu server poate fi scris în doar cinci linii de cod, iar un simplu apel de tip `GET` poate fi scris în doar două linii.



Odată cu popularizarea acestei noi tehnologii, multe dintre marile companii de IT ale lumii au început să își migreze serverele de la soluții tradiționale în domeniul business, precum Java Spring, la Node.js. Câteva companii mari care folosesc servere de producție scrise în Node.js sunt Netflix, PayPal, Uber, eBay, NASA și Yahoo. Paypal a raportat în urma dezvoltării aceluiași produs folosind Java Sprint și Node.js că folosind Node aplicația web a fost realizată aproape de două ori mai rapid de mai puțini dezvoltatori decât aplicația Java și a fost compusă din 40% mai puține fișiere și 33% mai puține linii de cod. Din punct de vedere al performanței, aplicația scrisă în Node.js a fost capabilă să răspundă la de două ori mai multe cereri decât aplicația scrisă în Java, cele două rulând pe mașini cu un fir de execuție pentru Node.js, respectiv cu cinci fire de execuție pentru Java Spring.

Un alt motiv pentru creșterea în popularitate a Node.js este MongoDB. MongoDB este o bază de date de tip NoSQL unde datele sunt reprezentate de documente (echivalentul rândurilor într-o tabelă) stocate în colecții (echivalentul tabelelor). MongoDB utilizează pentru documente un format de date numit JSON, un acronim pentru JavaScript Object Notation, însă aceste date sunt stocate în format binar, BSON – Binary JSON. Din nume reiese faptul că acest format de date este nativ pentru aplicațiile scrise în JavaScript, deci nu apare costul suplimentar necesar transformării datelor dintr-un tip în altul pentru prelucrare pe server. MongoDB este o soluție pentru aplicațiile în care au loc foarte multe citiri și relativ puține scrieri în colecții foarte mari de date. Chiar numele „mongo” provine din englezescul informal „humongous” care se poate traduce ca „foarte mare”, deci și din nume se poate înțelege că această bază de date suportă gestionarea unor cantități imense de date. Spre deosebire de bazele de date SQL, MongoDB, o bază de date de tipul NoSQL, nu impune o schemă, un format, pentru datele stocate, deci într-o colecție putem regăsi documente ce conțin date variate, un lucru ce poate fi foarte util în dezvoltarea pe termen lung al unui produs. De exemplu, după o perioadă îndelungată de timp în care s-au acumulat foarte multe date într-o colecție se dorește adăugarea unui nou parametru opțional. Adăugarea noului parametru se face prin modificarea codului care adaugă sau actualizează datele pe server, fără să necesite alte operațiuni care rulează în fundal în cadrul bazei de date, afectând performanțele pentru o perioadă de timp. Un alt punct forte pentru acest tip de baze de date este posibilitatea de replicare. MongoDB oferă o soluție pentru păstrarea redundanței prin posibilitatea configurării unui „Replica Set”, astfel dacă o mașină dintr-un „Replica Set” pățește ceva, datele nu sunt pierdute. Alt mare avantaj pe care îl oferă această soluție pentru baza de date este posibilitatea de a scala orizontal. Multe dintre soluțiile SQL nu au posibilitatea de a scala orizontal, ci doar vertical, adică singura posibilitate de scalare este cea de a mai adăuga memorie și putere de procesare mașinii pe care rulează baza de date. Acest tip de scalare este limitată deoarece după un anumit prag nu se mai poate adăuga putere de procesare, iar scalarea orizontală este uneori imposibilă deoarece nu se pot împărți datele din tabele între mai multe mașini în cadrul unui cluster. Soluția NoSQL permite scalarea orizontală. În cazul MongoDB aceasta se materializează sub forma conceptului de „sharding”, posibilitatea de a avea mai multe „shard-uri”, adică mașini sau grupuri de mașini („replica set-uri”) care stochează datele în funcție de anumite criterii. Această soluție este una transparentă, utilizatorul bazei de date neavând posibilitatea de a ști că datele i-au fost livrate dintr-un „shard” sau dintr-un simplu „replica set”.



În cadrul aplicației Get Together au fost necesare nouă colecții afișate în următoarea schemă (figura 1). Majoritatea ID-urilor folosite pentru indexarea și interogarea bazei de date sunt de forma „UUID” – „Universally Unique Identifier”, în mod normal un număr de 128 de biți reprezentat ca 32 de cifre hexazecimale („123e4567-e89b-12d3-a456-426655440000”), dar pe care îl stochez ca pe un șir de caractere, un obiect de tip String.

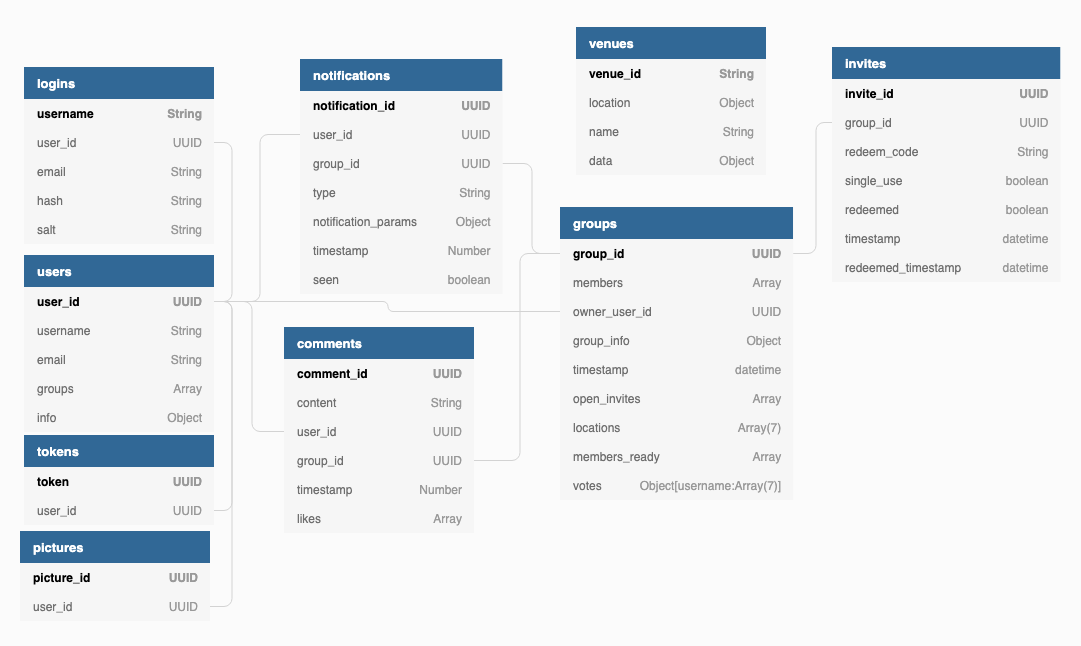


Figura 1 Structura bazei de date

Colecția „logins” conține datele adăugate în urma procesului de înregistrare pe platformă, utilizate pentru autentificare. Cheia principală, unică, este numele utilizatorului. Nu pot exista doi utilizatori cu același nume. De asemenea este adăugat și „user\_id”-ul utilizatorului nou creat pentru a stabili o relație între colecțiile „logins” și „users”. Cele mai importante date în această colecție, în afară de „username”, sunt „hash”-ul și „salt”-ul, date rezultate în urma criptării parolei utilizatorului folosind „PBKDF2” („Password-Based Key Derivation Function 2”).

Colecția „users” dispune de o cheie principală, unică, stocată sub numele „user\_id”. Această colecție conține date despre utilizatori precum „username”, „email”, o listă „groups” cu id-urile grupurilor din care face parte utilizatorul, respectiv un obiect „info” cu date opționale despre utilizator precum numele și prenumele real, data nașterii, sexul și id-ul pozei de profil. În această colecție și în colecția „logins” sunt adăugate date simultan, la înregistrarea unui nou utilizator.

Colecția „tokens” conține o asociere între „user\_id” și „token”-ul de autentificare pe baza căruia se permite accesul în back-end. Această colecție conține datele ce fac distincția între utilizatori într-o manieră opacă, fără să expună public date sensibile ce ar putea da de gol publicului structura bazei de date, precum „user\_id”-ul, deschizând o vulnerabilitate la atacuri. În colecția „tokens” sunt adăugate date în urma execuției procesului de autentificare pe bază de nume de utilizator și parolă.

Colecția „pictures” conține o asociere între id-ul pozei de profil al fiecărui utilizator și id-ul acestuia. În această colecție se adaugă date în urma procesului de încărcare a unei imagini. Principalul rol al acestei colecții este de a păstra o referință a fișierelor aflate în volumul de memorie a serviciului însărcinat cu servirea și stocarea pozelor de profil, „picture\_id”-ul fiind același cu numele fișierului stocat.

Colecția „notification” conține cheia unică „notification\_id” împreună cu id-urile pentru grupul unde a fost generată, respectiv al utilizatorului căruia i se adresează notificarea. De asemenea mai sunt stocate date precum tipul notificării, un „timestamp”, un număr care reprezintă milisecundele trecute de la 1 Ianuarie 1970, un parametru boolean, „seen”, care marchează dacă a fost văzută, sau nu, notificarea. Alături de datele obligatorii menționate este prezent și un obiect numit „notification\_params” în care sunt adăugate date opționale utilizate pentru generarea textului ce va fi afișat în front-end.

Colecția „comments” conține date despre comentariile adăugate într-un grup. Cheia unică este „comment\_id”, dar pentru o identificare rapidă sunt indexate și id-urile grupului și a utilizatorului care a adăugat comentariul. Alături de aceste date de identificare sunt prezente câmpurile „content”, pentru conținutul efectiv al comentariului, un timestamp pentru a marca data exactă când a fost adăugat comentariul și o listă cu id-urile utilizatorilor care au apăsat butonul de „like”.

Colecția „invites” conține date despre invitațiile generate. Această colecție conține două chei unice, „invite\_id”, respectiv „redeem\_code”. Id-ul invitației, un UUID, este folosit în operațiile de ștergere și de actualizare, iar codul de revendicare are o formă mai ușor de introdus manual. Acesta este format din 12 caractere alfanumerice, grupate câte trei și separate folosind caracterul cratima (Ex: „QBK3-BAPF-25HM”). Un astfel de document mai conține id-ul grupului aferent invitației, un parametru boolean „single\_use” care indică dacă invitația poate fi folosită de mai multe ori, respectiv două obiecte de tip „Date”, „timestamp” – pentru data la care a fost creată invitația, respectiv „redeem\_timestamp” – pentru data la care a fost revendicată invitația.

Colecția „venues” are un rol foarte important în menținerea costurilor la un nivel cât mai scăzut, împreună cu serviciul ce se ocupă de gestionarea acestei colecții. Aici se stochează date despre localurile obținute prin apeluri către serviciul „Google Places”, metoda „Nearby Search”. Acest serviciu este unul relativ scump ajungând să coste 32.00 de dolari americani pentru 1000 de cereri atunci când aplicația realizează într-o lună un număr de cereri aflat între 0 și 100 000, 25.69 de dolari americani pentru 1000 de cereri atunci când aplicația realizează între 100 001 și 500 000 de cereri pe lună. Pentru un volum mai mare de 500 000 de cereri HTTP pe lună se poate negocia prețul cu reprezentanții de vânzări ai companiei Google. Astfel, accesul la acest serviciu poate genera costuri foarte mari în timp. Pentru a combate acest fenomen toate cererile de listare a sugestiilor către back-end-ul Get Together, pentru o anumită zonă, se vor face doar după interogarea colecției „venues”. În cazul în care interogarea bazei de date nu a oferit destule rezultate se încearcă aceeași interogare, dar către serviciul „Google Places”, apoi rezultatul este stocat în această colecție. Câmpul „venue\_id” reprezintă un șir de caractere unic atribuit de serviciul oferit de Google. Acesta este folosit pentru a nu stoca de două ori aceleași date. Alte câmpuri ce se regăsesc într-un document din această colecție sunt un șir de caractere ce reprezintă numele localului, un obiect „data” ce conține răspunsul nealterat oferit de metoda „Nearby Search” oferită de „Google Places”, respectiv un obiect numit „location”. Acest ultim obiect este ceea ce face colecția „venues” să fie specială. Acesta este un index geospațial, mai precis un obiect „GeoJSON”, definit urmărind documentația MongoDB. Acesta conține un șir de caractere, „type”, care are întotdeauna valoarea „Point”, deoarece stocăm puncte pe hartă, respectiv o listă, „coordinates”, ce conține două valori numerice, longitudinea și latitudinea, în această ordine. Folosind acest index spațial putem interoga această colecție la fel cum am căuta date pe o hartă sau pe un glob pământesc. Putem seta un punct și o rază, urmând să obținem toate datele ce se intersectează cu sfera rezultată, sau putem căuta date ce se află în interiorul unui poligon. Faptul că putem realiza cu ușurință interogări geospațiale este un alt motiv pentru care am ales MongoDB pentru această aplicație.

Colecția „groups” este cea mai complexă dintre cele nouă utilizate în cadrul aplicației. Aceasta conține un index unic „group\_id”, o listă „members” cu toate id-urile utilizatorilor ce aparțin grupului, un „owner\_user\_id” pentru a reține cine a fost creatorul grupului, un obiect de tip dată numit „timestamp” menit să marcheze data la care a fost creat grupul.

TODO:

Microservices architecture

Docker

Kubernettes, but docker-compose

Mosquitto / MQTT.

Redis for tokens caching

VueJS.

WebSockets.

# Detalii de implementare

În plus fata de capitolul precedent acesta conține elemente specifice ale rezolvării problemei care au presupus dificultăți deosebite din punct de vedere tehnic. Pot fi incluse configurații, secvențe de cod, pseudo-cod, implementări ale unor algoritmi, analize ale unor date, scripturi de testare. De asemenea, poate fi detaliat modul în care au fost utilizate tehnologiile introduse in capitolul 3.

Criterii pentru calificativul Nesatisfăcător:

* Sunt prezentate pe scurt scheme și pseudo-cod.

Criterii pentru calificativul Satisfăcător:

* Descriere sumara a implementării, prezentarea unor secvențe nerelevante de cod, scheme, etc.

Criterii pentru calificativul Bine:

* Descrierea detaliată a algoritmilor/structurilor utilizați; Prezentarea etapizată a dezvoltării, inclusiv cu dificultăți de implementare întâmpinate, soluții descoperite; (dacă este cazul) demonstrarea corectitudinii algoritmilor utilizați.

## Indicații formatare tabele

Se recomandă utilizarea tabelelor de forma celui de mai jos. Font: Calibri, 9.

Orice tabel prezent în teză va fi referit în text; exemplu: a se vedea Tabel 1.

# Studiu de caz / Evaluarea rezultatelor

Acest capitol trebuie să răspundă, în principiu, la **2 întrebări** și să se încheie cu **o discuție** a rezultatelor obținute. Cele doua întrebări la care trebuie sa se răspundă sunt:

1) **Merge corect**? (Conform specificațiilor extrase în capitolul 2);

Evaluarea dacă merge corect se face pe baza cerințelor identificate în capitolele anterioare.

2) Cât de bine merge / cum se compară cu soluțiile existente? (pe baza unor metrici clare).

Evaluarea cât de bine merge trebuie să fie bazată pe procente, timpi, cantitate, numere, **comparativ cu soluțiile prezentate în capitolul 3**. Poate fi vorba de performanță, overhead, resurse consumate, scalabilitate etc.

În realizarea discuției, se vor utiliza tabele cu procente, rezultate numerice și grafice. În mod obișnuit, aici se fac comparații și teste comparative cu alte proiecte similare (dacă există) și se extrag puncte tari și puncte slabe. Se ține cont de avantajele menționate și se demonstrează viabilitatea abordării / aplicației, de dorit prin comparație cu alte abordări (dacă acest lucru este posibil). Cuvântul cheie la evaluare este „metrică“: trebuie să aveți noțiuni măsurabile și cuantificabile. În cadrul procesului de evaluare, explicați datele, tabelele și graficele pe care le prezentați și insistați pe relevanța lor, în următorul stil: „este de preferat ... deoarece …“; explicați cititorului nu doar datele ci și semnificația lor și cum sunt acestea interpretate. Din această interpretare trebuie să rezulte poziționarea proiectului vostru printre alternativele existente, precum și cum poate fi acesta îmbunătățit în continuare.

Criterii pentru calificativul *Nesatisfăcător*:

* Aplicația este testată dar rulează pe calculatorul studentului, nu există posibilități de testare, nu a fost validată cu clienți / utilizatori;
* Nu au fost realizate comparații cu alte sisteme similare.

Criterii pentru calificativul *Satisfăcător*:

* [Dezvoltare de produs] Există teste unitare și de integrare, există o strategie de punere în funcțiune (*deployment*), există validare minimală cu clienții / utilizatorii.
* [Cercetare] Principalele componente și soluția în ansamblu au fost evaluate din punct de vedere al performanței, însă nu sunt folosite seturi de date standard, există unele erori de interpretare a datelor.
* [Ambele] Discuție minimală asupra relevanței rezultatelor prezentate, comparație minimală cu alte sisteme similare.

Criterii pentru calificativul *Bine*:

* [Dezvoltare de produs] Teste unitare și de integrare, instrumente de punere in funcțiune (*deployment*) utilizate și care arată lucru constant de-a lungul semestrului, lucrare validată cu clienții / utilizatorii, produs în producție.
* [Cercetare] Componentele și soluția în ansamblu au fost evaluate din punct de vedere al performanței, folosind seturi de date standard și cu o interpretare corectă a rezultatelor.
* [Ambele] Discuție cu prezentarea calitativă și cantitativă a rezultatelor, precum și a relevanței acestor rezultate printr-o comparație complexă cu alte sisteme similare.

# Concluzii

În acest capitol este sumarizat întreg proiectul, de la obiective, la implementare, si la relevanta rezultatelor obținute. În finalul capitolului poate exista o subsecțiune de „Dezvoltări ulterioare“.  
Criterii pentru calificativul *Nesatisfăcător*:

* Concluziile nu sunt corelate cu conținutul lucrării;

Criterii pentru calificativul *Satisfăcător*:

* Concluziile sunt corelate cu conținutul lucrării, însă nu se oferă o imagine asupra calității și relevantei rezultatelor obținute;

Criterii pentru calificativul *Bine*:

* Concluziile sunt corelate cu conținutul lucrării, și se oferă o imagine precisa asupra relevantei și calității rezultatelor obținute în cadrul proiectului.

# Bibliografie

* Trebuie respectat **un singur standard** de trimiteri bibliografice (citare), **dintre** următoarele alternative:
  + APA (<http://pitt.libguides.com/c.php?g=12108&p=64730>)
  + IEEE (<https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf>)
  + Harvard (<https://libweb.anglia.ac.uk/referencing/harvard.htm>)
  + Cu numerotarea referințelor în ordine alfabetică sau în ordinea apariției în text (de exemplu, stilul cu numere folosit de unele publicații ACM - <https://www.acm.org/publications/authors/reference-formatting>)
* Toate referințele din acest capitol trebuie să fie referite în text. Exemple:
  + [Articol jurnal]: [2];
  + [Articol conferință]: [3];
  + [Carte]: [4]
  + [Weblink]: [5]
  + [Application report] [6]

|  |  |
| --- | --- |
| [1] | J. Silva-Martinez, "ELEN-325. Introduction to Electronic Circuits: A Design Approach," 2008. [Online]. Available: http://www.ece.tamu.edu/~spalermo/ecen325/Section%20III.pdf. |
| [2] | H. Baali, H. Djelouat, A. Amira and F. Bensaali, "Empowering Technology Enabled Care Using IoT and Smart Devices: A Review," *IEEE Sensors Journal,* vol. 18, no. 5, pp. 1790-1809, 2018. |
| [3] | A. Haroon, S. Akram, M. A. Shah and A. Wahid, "E-Lithe: A Lightweight Secure DTLS for IoT," in *IEEE 86th Vehicular Technology Conference (VTC-Fall)*, Toronto, 2017. |
| [4] | A. K. Jain and R. C. Dubes., Algorithms for Clustering Data, Upper Saddle River: Prentice-Hall, Inc., 1988. |
| [5] | "Kernel panic! What are Meltdown and Spectre, the bugs affecting nearly every computer and device?," techcrunch.com, 2018. [Online]. Available: https://techcrunch.com/2018/01/03/kernel-panic-what-are-meltdown-and-spectre-the-bugs-affecting-nearly-every-computer-and-device. [Accessed 14 02 2018]. |
| [6] | E. Rogers, "Understanding Buck-Boost Power Stages in Switch Mode Power Supplies," Texas Instruments, 2007. |

* NU utilizați referințe la Wikipedia sau alte surse fără autor asumat.
* Pentru referințe la articole relevante accesibile în web (descrise prin URL) se va nota la bibliografie și data accesării.
* Mai multe detalii despre citarea referințelor din internet se pot regăsi la:
  + <http://www.writinghelp-central.com/apa-citation-internet.html>
  + <http://www.webliminal.com/search/search-web13.html>
* Note de subsol se utilizează dacă referiți un link mai puțin semnificativ o singură dată; Dacă nota este citată de mai multe ori, atunci utilizați o referință bibliografică.
* Dacă o imagine este introdusă în text și nu este realizată de către autorul lucrării, trebuie citată sursa ei (ca notă de subsol sau referință - este de preferat utilizarea unei note de subsol).
* Referințele se pun direct legate de text (de exemplu „KVM [1] uses“, „as stated by Popescu and Ionescu [12]”, etc.). Nu este recomandat să folosiți formulări de tipul „[1] uses”, „as stated in [12]“, „as described in [11]“ etc.
* Afirmațiile de forma „are numerous“, „have grown exponentially“, „are among the most used“, „are an important topic“ trebuie să fie acoperite cu citări, date concrete si analize comparative.
  + Mai ales în capitolele de introducere, „state of the art“, „related work“ sau „background“ trebuie să vă argumentați afirmațiile prin citări. Fiți autocritici și gândiți-vă dacă afirmațiile au nevoie de citări, chiar și cele pe care le considerați evidente.
  + Cea mai mare parte dintre citări vor fi în capitolele de introducere „state of the art“, „related work“ sau „background“.
* Toate intrările bibliografice trebuie citate în text. Nu le adăugați pur și simplu la final.
* Nu copiați sau traduceți niciodată din surse de informație de orice tip (online, offline, cărți, etc.). Dacă totuși doriți să oferiți, prin excepție, un citat celebru - de maxim 1 frază- utilizați ghilimele și evident menționați sursa.
* Dacă reformulați idei sau creați un paragraf rezumat al unor idei folosind cuvintele voastre, precizați cu citare (referință bibliografică) sau cu notă de subsol sursa sau sursele de unde ați preluat ideile.

# Anexe

Anexele sunt opționale.

Ce poate intra în anexe:

* Exemplu de fișier de configurare sau compilare;
* Un tabel mai mare de ½ pagină;
* O figura mai mare mai mare de ½ pagină;
* O secvență de cod sursa mai mare de ½ pagină;
* Un set de capturi de ecran („screenshot”-uri);
* Un exemplu de rulare a unor comenzi plus rezultatul („output”-ul) acestora;
* În anexe intră lucruri care ocupă mai mult de o pagină ce ar întrerupe firul natural de parcurgere al textului.