UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Soluție de Mesagerie pentru Aplicațiile Mobile, cu Capabilități Offline

propusă de

Marian-Vasile Iordache

Sesiunea: iulie, 2019

Coordonator științific

Lect. dr. Panu Andrei

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Soluție de Mesagerie pentru Aplicațiile Mobile, cu Capabilități Offline

Marian-Vasile Iordache

Sesiunea: iulie, 2019

Coordonator științific

Lect. dr. Panu Andrei

Avizat,

Îndrumător lucrare de licență,

Lect. dr. Panu Andrei.

Data: ..... Semnătura: .....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Iordache Marian-Vasile** domiciliat în **România, jud. Iași, mun. Iași, Str. Ciric, nr. 34, bl. X3, et. 5, ap. 32**, născut la data de **14 septembrie 1996**, identificat prin CNP **1960914226723**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2019, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Soluție de Mesagerie pentru Aplicațiile Mobile, cu Capabilități Offline** elaborată sub îndrumarea domnului **Lect. dr. Panu Andrei**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: ..... Semnătura: .....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Soluție de Mesagerie pentru Aplicațiile Mobile, cu Capabilități Offline**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Marian-Vasile Iordache**

Data: ..... Semnătura: .....

ACORD PRIVIND PROPRIETATEA DREPTULUI DE AUTOR

Facultatea de Informatică este de acord ca drepturile de autor asupra programelor-calculator, în format executabil și sursă, să aparțină autorului prezentei lucrări, **Marian-Vasile Iordache**.

Încheierea acestui acord este necesară din următoarele motive:

- doresc continuarea dezvoltării acestui produs, fiind alcătuit din 3 module, pentru a fi publicat sub licența MIT pentru comunitatea open-source, în nume propriu;
- doresc adăugarea unor noi module peste acest proiect pentru a urma a fi comercializate ca pachete auxiliare.

Iasi,

Decan **Adrian Iftene**

Semnătura: .....

Absolvent **Marian-Vasile Iordache**

Semnătura: .....

Cuprins

Motivație

Aplicaţia de mesagerie este una dintre cele mai folosite de când a fost inventat telefonul mobil, însă în acelaşi timp cu evoluţia dispozitivelor, devenind din ce în ce mai performante, a apărut o cerere din partea consumatorilor de aplicaţii pentru a uşura modul de comunicare dintre aceştia.

Binecunoscutele aplicaţii Whatsapp, Viber, Snapchat, etc. au reuşit să-şi demonstreze utilitatea de-a lungul timpului prin oferirea unor soluţii proprii dezvoltate pentru a satisface cerinţele utilizatorilor, dar ca orice soluţie terţă disponibilă în momentul actual, confidenţialitatea şi integritatea datelor nu poate fi asigurată.

Prin această lucrare îmi propun să ofer o soluţie completă de mesagerie, open-source, portabilă, tolerantă la întreruperile de conexiune la internet, scalabilă, uşor de integrat în aplicaţii existente, interfaţă ajustabilă, proprietatea datelor şi extrem de potrivită pentru sistemele închise precum şcolile, instituţiile guvernamentale, grupuri sociale sau familii.

Introducere

Folosirea unor terţi ce oferă soluţii de mesagerie poate fi îngrijorătoare dacă ne uităm la frecvenţa breşelor de securitate din ultima vreme, fie că folosim o aplicaţie de sine stătătoare ce poate atrage utilizatorul printr-o interfaţă şi o experienţă plăcută, fie că alegem să dezvoltăm propria noastră aplicaţie de mesagerie folosind servicii existente (Chat as a Service), putem ajunge să fim ţinta atacurilor din partea unor persoane mai puţin binevoitoare.

Aplicaţiile moderne oferă soluţii din ce în ce mai extreme, dar în acelaşi timp inovative, pentru a asigura confidenţialitatea şi integritatea datelor, precum criptarea end-to-end, distrugerea mesajelor după ce acestea au fost vizualizate, autentificare în doi paşi, însă foarte mulţi utilizatori evită să revizuiască ”Termenii şi condiţiile de utilizare” sau ”Politica de confidenţialitate” şi uneori rămân uimiţi cum la următorul atac cibernetic o parte din datele acestora devin publice. Totul începe prin introducerea adresei de email sau numărul de telefon într-o aplicaţie terţă, urmată de cererea de permisiuni pentru geolocalizare, acces la agenda de contacte sau chiar citirea SMS-urilor.

Problema serviciilor terţe ar putea fi rezolvată dacă avem o soluţie ce ne oferă posibilitatea de a crea propria noastră platformă de mesagerie; soluţii open-source există deja pe piaţă, unde cele mai poluare sunt Rocket. Chat, Crisp, Zulipchat.

Aplicaţia realizată cu ocazia acestei lucrări reprezintă componentele de bază cum ar fi mesaje text, istoricul mesajelor, conexiune prin web sockets, offline messaging, interfaţă. Pentru alcătuirea unei soluţii de mesagerie, pe baza creată, se pot extinde alte componente auxiliare, dar la fel de importante, precum trimiterea de imagini, înregistrări audio, conţinut video, criptarea end-to-end şi altele.

Importanţa acestor produse software open-source vine odată cu nevoia si dificultăţile dezvoltatorilor de aplicatii mobile sau web de a întruni cerinţele utilizatorilor. Dezvoltarea unei astfel de soluţii poate fi foarte costisitoare, iar dacă luăm în calcul

cazurile din istorie:

- 11 septembrie 2001, în New York, unde exact după atentat, guvernul american începuse să bruieze sau chiar să oprească conexiunile la nivel de stat, iar posibilitatea de lua legătura cu membri ai familiei, prieteni sau colegi devenise aproape imposibilă. Soluţiile de mesagerie erau puţine, funcţionalitatea offline a aplicaţiilor era inexistentă, iar retrimiterea mesajelor trebuia executată manual.
- 2011, Revoluţia egipteană, guvernul blochează platformele de social media, urmată de o întrerupe a conexiunilor la reţeaua publică, iar cei care reuşeau totuşi să prindă semnal singura cale prin care puteau să comunice cu cei din exterior era să folosească servicii proxy, care de asemenea au fost blocate. Având posibilitatea de a crea platforme de mesagerie ajustabile, ce pot rula pe o reţea privată poate fi asigurată libertatea de exprimare.

Necesitatea unei soluţii ce utilizează o suită de tehnologii moderne a apărut în urma lipsei de adaptabilitate a unor companii sau comunităţi la ultimele standarde, soluţia pe care o voi prezenta în continuare este alcătuită din trei module, server, client şi UI ajustabil, dar accentul va cade pe primele două componente.

Capitolul 1

Server

Până la apariţia protocolului WebSocket în 2011, cel mai popular mod de a crea o aplicaţie de ”instant messaging” a fost prin implementarea Extensible Messaging and Presence Protocol (XMPP), împreună cu Bidirectional-streams Over Synchronous HTTP (BOSH). Considerat un standard în dezvoltarea aplicaţiilor în timp real, protocolul WebSocket a primit mult credit din partea dezvoltătorilor de software, depreciind BOSH şi oferind flexibilitatea de a înlocui XMPP la nivelul aplicaţiei.

1.1 XMPP şi BOSH

BOSH a fost realizat pentru a îmbunătăţi performanţa şi pentru a crea un momentum pentru XMPP, bazându-se pe o tehnică numită ”long-polling”, aceasta a fost recunoscută încă de la început ca fiind o soluţie de moment până când o să apară o tehnologie mai puţin costisitoare. Fiind creat special pentru XMPP acesta deschide două conexiuni concurente (trimitere, primire), iar de fiecare dată după ce se trimite o cerere, conexiunea trebuie închisă ca mai apoi să fie redeschisă. Această soluţie este costisitoare din punct de vedere al scalabilităţii, deoarece la fiecare cerere este adăugat header-ul, cookies, etc. Iar transferul de date de vine uriaş. Trimiterea de fişiere este de asemenea costisitoare, rezultând în coderea fişierului în base64 unde în medie dimensiunea fişierului creşte cu până la 30%.

Capitolul la care XMPP se situează cel mai bine este securitatea datelor şi interoperabilitatea dintre instanţe, acţionând pe o arhitectură ce se bazează pe descentralizare (un utilizator poate fi client, dar să acţioneze şi ca server).

1.2 Subscripţie GraphQL peste WebSockets

Protocolul WebSocket este unul bidirecţional ce vine ca soluţie la problemele create de tehnologii precum BOSH. Avantajele folosirii acestui protocol sunt reducerea costurilor operaţionale, vine cu o schemă de compresie ce previne problemele de securitate întâlnite de XMPP peste HTTP, restul beneficiilor fiind aceleaşi oferite de tehnologii create pe baza modelului Comet însă reuşeşte să îmbunătăţească performanţa în general.

GraphQL este un query language ce oferă mai mult control programatorilor asupra datelor ce urmează a fi cerute dintr-un QueryType creat. Acesta a apărut ca soluţie la probleme des întâlnite de paradigma REST, precum 'overfetching' şi 'underfetching'.

GraphQL peste WebSocket vine ca o mâna de ajutor pentru o implementare scalabilă, permiţând ca logica aplicaţiei de mesagerie să fie pe evenimente bine definite anterior, iar pe baza acestor răspunsuri (payloads) să am mai mult control pe partea de client, dar şi pentru a evita blocarea operaţiilor pe baza de date.

Subscripţia

Subscripţia GraphQL este o operaţie ce urmăreşte modificările la nivelul serverului, depinde de utilizarea primitivei PubSub, astfel atunci când un utilizator trimite un mesaj către server, un grup de utilizatori (inclusiv emiţătorul) va primi ca răspuns un eveniment ce indică că un nou mesaj a fost adăugat.

Având în vedere existenţa altor metode de a actualiza datele utilizatorului, precum "polling" ce poate fi setat pentru a executa o cerere la un număr exact de secunde; o altă metodă "refetching" presupune ca la trimiterea unui mutation de către un utilizator acesta să urmeze execuţia query din care face parte.

Motivul pentru care am ales folosirea primitivei PubSub, în aplicaţiile de instant messaging actualizarea datelor trebuie să se realizeze sub o secundă, când utilizatorul deschide aplicaţia acesta trimite un "mesaj" pe o conexiune TCP (metoda UPGRADE) ce are ca rol trecerea la o conexiune bazată pe WebSockets. Când conexiunea este finalizată, primitiva 'onConnect' va trimite utilizatorului structura Chat, ce conţine ChatRooms, ChatMessages şi Metadata, ca mai apoi actualizarea acestora să fie bazată pe evenimente.

Legătura cu baza de date

GraphQL este agnostic faţă de baza de date utilizată. Legătura efectivă este realizată la nivelul rezolvărilor mutaţiilor definite însemnând ca la nivelul acestora totodată efectuăm operaţii pe colecţiile în cauză.

Structurarea si formatarea datelor

Grapher este un strat de data fetching peste Meteor şi mongoDB. În principal îl folosesc pentru abilitatea lui de a face mongodb să fie relaţionala. Aceasta este realizată prin declararea şi definirea unor legături între diferite colecţii din baza de date. Totodată Grapher ne oferă posibilitatea de crea query-uri dinamice ce ajută la utilizarea respectivelor legături între colecţii. De asemenea funcţionează foarte bine împreună cu Apollo Graphql.

1.3 Stocare de date (MongoDB si Redis)

Baza de date nerelationala (NoSQL) MongoDB oferă mai multă flexibilitate în structurarea şi stocarea datelor. Pentru o aplicaţie de instant messaging, soluţia optimă a fost să împart componentele principale în colecţii pentru a optimiza interogarea acestora.

Acţionând ca un serviciu Chat-as-a-Service, dinamica în modificarea schemei este una vitală pentru o aplicaţie de instant messaging. Odată cu scalarea aplicaţiilor într-un timp cât mai scurt duce la nevoia de date suplimentare, iar adăugarea unor câmpuri noi la colecţii în producţie este un punct forte pentru folosirea unei baze de date nerelationala.

Un alt avantaj pentru folosirea MongoDB este modelul arhitectural al acesteia ce presupune împărţirea bazei de date în mai multe baze de date mai mici. Astfel putem avea o baza de date distribuită pe mai multe noduri, partiţionând orizontal înregistrările din aceasta.

La nivel arhitectural, baza de date a aplicaţiei este structurată pe două colecţii, ChatRooms şi ChatMessages, această decizie ajutând la optimizarea operaţiilor de citire prin indexarea colecţiei în funcţie de grupul de utilizatori dintr-un Room, iar colecţia ChatMessages foloseşte ChatRoomId ca şi index.

1.4 Deployment

Pentru a urca aplicaţia pe un server m-am folosit de Meteor Up (MUP) un ‘command line interface‘ ce automatizează procesul de urcare al aplicaţiilor meteor pe server. Rularea aplicaţiilor pe server se face cu ajutorul imaginilor Docker.

1.5 Scalabilitate si Performanta

Arhitectura serverului este alcătuită într-o manieră uşor de scalat și modularizat odată cu creşterea numărului de utilizatori.

Meteor şi MongoDB la un nivel fundamental au fost create ca soluție pentru o scalare uşoară, deşi există potenţial pentru a susţine sute de mii de utilizatori, integrarea celor două tehnologii necesită anumite configurări.

1. **Indexarea pe baza de date.** MongoDB vine cu o indexare iniţială pe baza coloanei `_id` însă pe partea colecţiei ce stochează schimbul de mesaje am adăugat un index pe `chatRoomId` pentru a îmbunătăţi performanţa tranzacţiilor pe baza de date.

```
ChatMessages.rawCollection().createIndex({chatRoomId:1})
```

2. **Scalare verticală.** Primul instinct a unui administrator de sistem ce se confruntă cu probleme de performanţă rezultate de suprasolicitarea puterii computaţionale a serverului este de a adăuga mai multe resurse (RAM, CPU, GPU), ceea ce ar însemna că un singur server s-ar ocupa de toate tranzacţiile ce vin din partea utilizatorului.

   Scalarea verticală este folosită până la un prag, însă nu e o soluţie de durată deoarece performanţa şi limitarea tehnologiei hardware ar pune probleme la un moment dat.

3. **Load balancing.**

   Conceptul implică distribuirea traficului pe mai multe instanţe de server, un modul pentru CLI-ul MUP descris în **Secţiunea 1.4** poate fi facil în atingerea acestui scop.

   Scalarea orizontală se realizează prin adăugarea mai multor instanțe la reţea. Această scalare se foloseşte de un Load-Balancer ce are ca scop distribuirea unei cereri a utilizatorului pe unu din nodurile create. Fiecare nod rulează o copie a serverului capabilă să gestioneze orice cerere redirecţionată de Load-Balancer.

   Distribuirea este influenţată prin gestionarea stării nodurilor identificând care nod este supraîncărcat cu cereri, care este în aşteptare sau cât de aproape este utilizatorul de un nod.

   Mecanismul de selectare a unui nod şi redirecţionarea cererii este una mai complexă, însă prin această metodă putem asigura scalabilitatea dar şi optimizarea timpilor în care un utilizator ar trebui să primească un răspuns.

   O altă responsibilitate a acestui mecanism este pentru a verifica starea nodurilor din cluster prin schimbarea de mesaje cu acesta sau prin implementarea protocolului ”ping echo”.

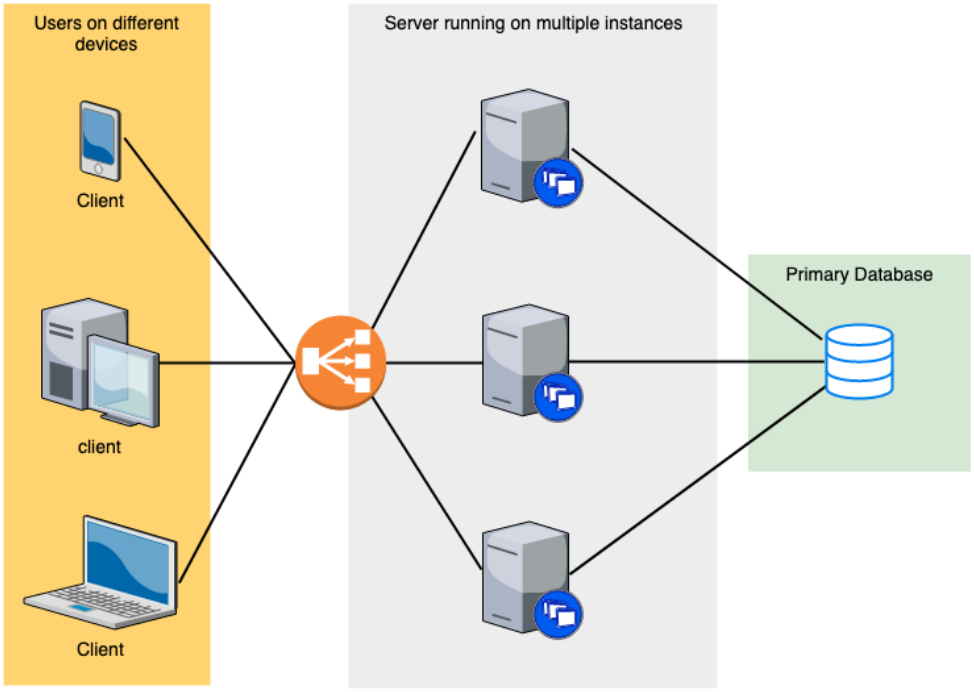

Figura 1.1: Scalare orizontală

4. **Sharding**

Adăugarea unui mecanism Load Balancer poate să rezolve parţial scalabilitatea unui serviciu, însă având mai multe instanţe de server conectate la aceeaşi baza de date ajungem în punctul în care dimensiunea datelor stocate creşte semnificativ.

Funcţionând pe acelaşi principiu ca Load Balancer (Master-Slave), pentru scalarea bazei de date putem folosi în primul rând conceptul de `Sharding` ce presupunea împărţirea bazei de date în mai multe partiţii cu dimensiuni aproximativ egale. Tranzacţiile pe baza de date se realizează într-o maniera eficientă asigurând o distribuţie egală pe `shards`; selectarea unui shard se face pe baza unui `shard key`.

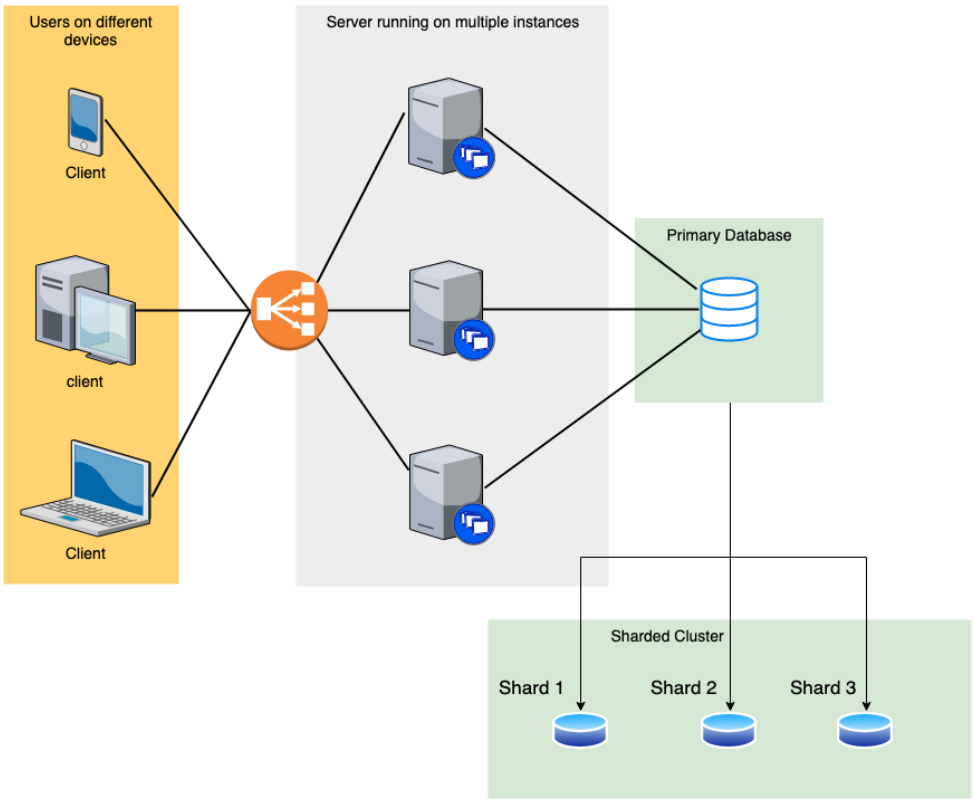

Figura 1.2: Sharding pe baza de date

5. **Replica Set**

Partiţionarea bazei de date în shards, deşi este o soluţie ce îmbunătăţeşte foarte mult performanţa tranzacţiilor, poate să expună arhitectura acesteia la eşuarea unor operaţii în caz că unul din shards nu este disponibil.

Într-o aplicaţie de instant messaging nu putem permite ca acest lucru, astfel ajungem în situaţia în care utilizatorii nu pot să primească istoricul chatului. Pentru a rezolva imposibilitatea de a accesa datele dintr-un shard trebuie să creăm clone a acestuia pentru a acţiona ca plase de siguranţă în caz că Shard Master nu poate fi accesibil.

Un alt beneficiu al clonării bazei de date este accesibilitatea în funcţie de locaţia utilizatorului, îmbunătăţind performanţa operaţiei Read interogând direct pe o clonă a bazei de date; operaţia Write rămâne neschimbată din punct de vedere al timpilor de executare, aceasta va trebui să fie trimisă spre baza de date Master ca apoi să fie clonată pe un shard şi pe baza de data replicată.

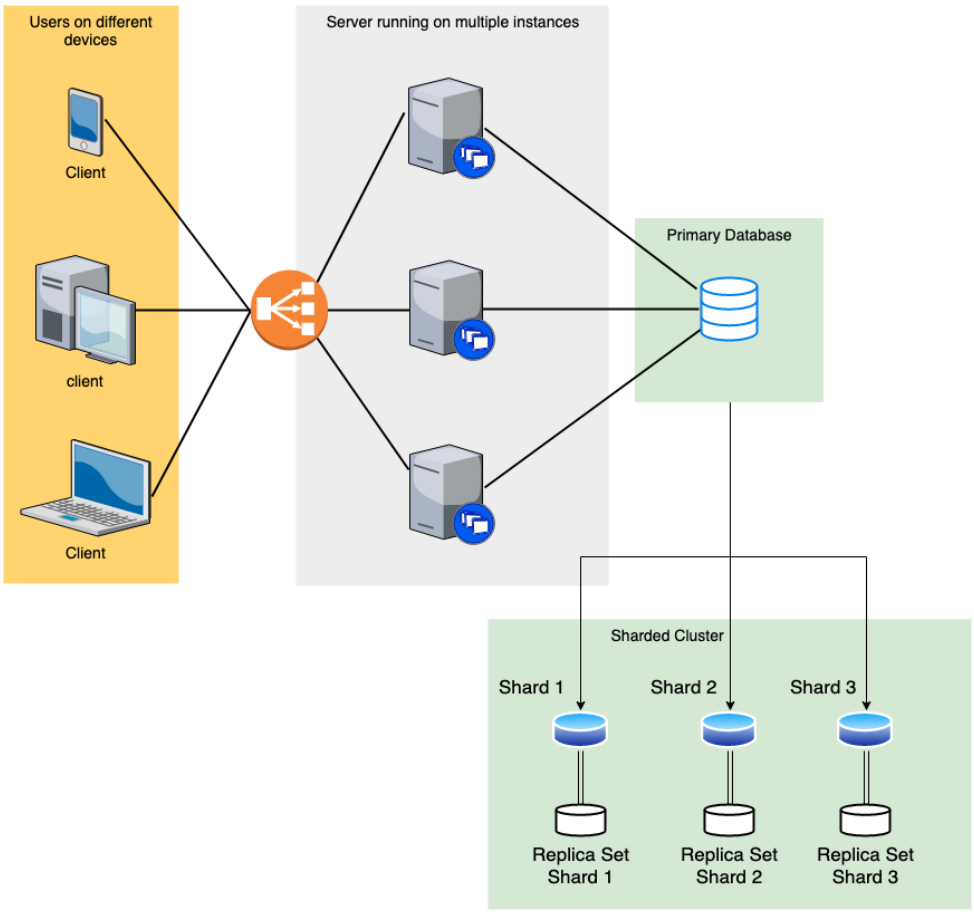

Figura 1.3: Shards on Replica Set

6. **Comunicarea prin WebSockets pe mai multe instanțe de server.**

   Scalabilitatea până în acest punct duce la un regres a funcţionalităţii de baza a acestui proiect. Într-o arhitectură ce a fost construită pe mai multe instanţe, un client poate trimite o cerere de conectare la WebSocket prin Load Balancer, însă datorită mecanismului de distribuire a fluxului de cereri, doi utilizatori ce vor să comunice între ei se pot conecta la două instanţe de server diferite, ceea ce ar face imposibilă comunicarea între cei doi.

   Ca soluţie la acest scenariu am ales să externalizez serviciul de PubSub pe un server Redis, astfel instanţele putând comunica între ele folosind o tehnologie rapidă bazată pe memorie cache şi un store avansat bazat pe cheie-valoare.

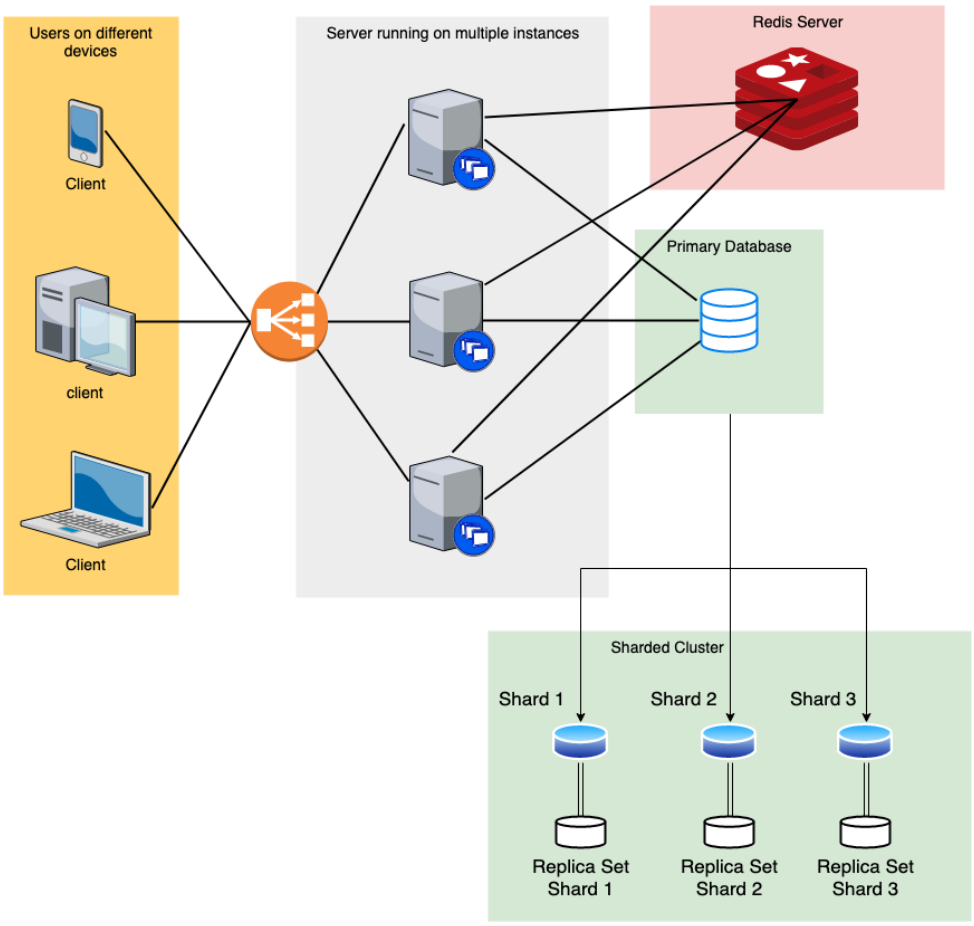

Figura 1.4: Redis ca serviciu extern de comunicare intre instanțe

1.6 Securitate

Un serviciu de instant messaging nu trebuie să ducă lipsă de un nivel de securitate, astfel am urmărit câteva concepte pentru a fi integrate la nivelul de gestionare a cererilor, totodată lăsând oportunitatea de îmbunătăţire a aplicaţiei la acest capitol ce se poate regăsi în **Capitolul 4. Posibilitatea de extindere a aplicației**.

Validare cerere utilizator

Cea mai comună greşeală în dezvoltarea unui API este apelarea unor metode publice având `userID` ca parametru, acest obicei duce la furarea datelor altor utilizatori dacă aceştia cunosc identificatorul utilizatorului, trimiterea de date false în numele victimei, şantajare folosind datele sensibile preluate de atacator, etc.

DDP Rate Limiting

Pentru prevenirea atacurilor DoS (Denial of Service) Meteor poate fi integrat cu o bibliotecă ce propune rezolvarea acestui scenariu. Fără o astfel de protecţie, utilizatorii răuvoitori pot încerca să supra solicite serverul cu un număr mare de cereri. Deşi avem implementată auto scalarea instanţelor de server şi a bazei de date este mult mai sigur să blocăm surplusul de cereri din punct de vedere al costurilor operaţionale.

SSL

Adăugarea certificatelor SSL a devenit un standard în zilele noastre, platformele de navigare sau de distribuţie au început să blocheze aplicaţiile atât la nivel de browser web cât şi la aprobarea aplicaţiilor mobile în magazinele virtuale consacrate.

Firewall la baza de date

Aceasta este opţiune uşor de configurat din platforma furnizorului de servicii pentru o baza de date în cloud, permiterea conexiunii la acesteia poate fi configurată astfel încât doar instanţele de server să se poată conecta la aceasta.

Capitolul 2

Client

Înainte de dezvoltarea aplicaţiei, am urmărit cu atenţie soluţiile de instant messaging pe care le folosim zilnic pentru a identifica componentele esenţiale ce trebuie realizate pentru a satisface cerinţele de bază ale utilizatorului. Prin urmare, având ca scop crearea unei aplicaţii care să fie familiară utilizatorului, componentele incluse în aplicaţie sunt oferite prin realizarea unei interfeţe comune, curată şi plăcuta cititului, de asemenea oferă o experienţă inedită prin menţinerea animaţiilor la 60 fps, istoricul mesajelor salvat într-un cache la nivelul aplicaţiei pentru a putea oferi atât o experienţă offline cât şi comoditate în privinţa retrimiterii mesajelor în caz că acestea au avut un răspuns de la server negativ.

2.1 Conexiunea Client - Server

Conexiunea la server se realizeaza prin instanţiere a unui client utilizând biblioteca Apollo. Această bibliotecă ne permite să realizăm un tunel prin care vor trece cererile clientului, astfel e posibilă modelarea acestora în funcţie de necesităţile aplicaţiei.

În realizarea tunelului, am inclus permiterea de a trimite cereri HTTP cât şi de a crea o conexiune prin WebSocket la server, inmemory cache ce permite reţinerea datelor la nivel de client, de exemplu dacă se execută o cerere de tip GET de două ori, prima dată acesta va trimite cererea la server, răspunsul oferit este stocat în cache, iar al doilea cerere va interoga mai întâi cache-ul, va returna mai întâi datele din cache ca mai apoi să fie actualizate cu răspunsul de la server. Bineînţeles, prin standardizarea tunelului prin care va trece cererea clientului, în caz că un utilizator va încerca să multiplice, intenţionat sau nu, avem tehnica ”deduplication” care va încerca să minimizeze

numărul de cereri prin reducerea duplicatelor.

De asemenea am inclus tratarea cazurilor când un utilizator dorește să trimită fişiere către APIul GraphQL, realizând astfel un stream pentru fiecare fişier trimis (multipart/form-data).

2.2 Structurarea datelor

Având în vedere că subscripţia realizată pe server poate trimite diferite evenimente în funcţie de modificările realizate pe baza de date, în primul rând avem nevoie de o metodă de a diferenţia tipurile de payloads ce urmează să le primească utilizator. Aşadar, subscripţia chat este realizată dintr-o uniune de cinci interfeţe, iar pentru a le putea deosebi la nivel de client am utilizat IntrospectionFragmentMatcher.

Pentru stocarea datelor la nivel de client am ales să folosesc AyncStorage, fiind un wrapper peste implementări native din Android şi iOS, ce serializează datele şi le salvează fie în RocksDB fie în SQLite. Pentru a avea o gestiune mai bună a datelor am implementat un mecanism folosind tehnologia Redux, aceasta îmi oferă un control mai bun asupra stării aplicaţiei, a datelor iar împreună cu sistemul de rehidratare realizat am putut să redau utilizatorilor posibilitatea de a trimite mesaje şi de a utiliza aplicaţia când aceasta prezintă pierderi a conexiunii la internet sau este complet inexistentă.

2.3 Portabilitate, Compatibilitatea şi alte consideraţii

Dezvoltarea soluţiei de mesagerie este una vitală în zilele noastre, fie că vorbim de crearea unui joc banal sau că vorbim despre crizele din ţările cu dictatori în poziţii de conducere ce limitează libertatea de exprimare a cetăţenilor, portabilitatea şi rapiditatea de integrare a tehnologiilor sunt două puncte cheie în luarea deciziilor atât din punct de vedere al unei afaceri cât şi din punct de vedere al situaţiei critice din ţara utilizatorului.

React Native este soluţia perfectă pentru o astfel de aplicaţie, fiind construit peste biblioteca React şi implementând JavaScript Engine. Pentru a realiza portabilitatea codului javascript către threadurile native pentru iOS respectiv Android este necesar un bridge între cele două threaduri (JavaScriptCore şi Main Thread) prin care se face schimbul de mesaje serializate.

Furnizarea unui API ce extinde componente native, de exemplu componenta View extinde clasele native ViewManager pentru Android şi RCTViewManager pentru iOS. Acest API ne permite reutilizarea codului până la 85%, restul fiind alcătuite din configurări specifice pentru fiecare platformă şi soluţii ajustate în funcţie de experienţa pe care vrem să o aibă un utilizator pe aplicaţie.

Ideea de portabilitate a aplicaţiilor realizate în React Native s-a extins dincolo de barierele mobile-ului, dacă am văzut compania Windows oferind suport, în mod exclusiv şi activ, pentru a dezvolta aplicaţii folosind React Native pentru platforma Windows Phone, comunitatea React a realizat pachete inclusiv pentru platformele de Desktop, Ubuntu, tvOS, macOS, dar cel mai important este pachetul pentru Web, astfel putând rulezi aplicaţia în mai multe medii de dezvoltare.

De la bun început arhitectura şi suita de tehnologii aleasă a fost centrată pe crearea unei soluţii în mai multe medii de dezvoltare, astfel clientul inițializat cu Apollo, poate fi configurat astfel încât să folosească tehnologii specifice mediului mobile, Webului dar şi alte.

Spre exemplu persistarea cache-ului presupune configurarea acestuia cu un storage, acesta poate fi atât `AsyncStorage` pentru React Native, cât şi `localStorage` pentru Web, de asemenea alte soluţii asemănătoare pot fi folosite atât cât respectă logica tehnologiilor menţionate anterior.

```
await persistCache({
  cache,
  storage: AsyncStorage || localStorage,
  maxSize: false,
  debug: true
});
```

Implementarea proiectului a avut la bază principiul singurei responsabilități, utilizând biblioteca React ce oferă flexibilitatea de a structura proiectul pe componente si micro componente (componente tip container şi de tip prezentaționale). Astfel, componentele de tip container au sarcina de a organiza datele şi evenimentele ce apar în urma interacţiunii dintre utilizator şi alţi membri ce folosesc aplicaţia, expediind diferite tipuri de payloads către store-ul global Redux.

Un alt avantaj al integrării bibliotecii React cu Redux este conexiunea dintre cele două, React actualizează componentele doar pe baza schimbării de proprietăţi sau a

stării acestuia şi oferind posibilitatea de a le lega la anumite părţi din store-ul Redux.

Aşadar, modularitatea şi reutilizarea componentelor, fie ele prezentaţionale sau de tip container se pot utiliza fără să sufere nicio modificare pe alte ecrane din aplicaţie sau chiar într-un produs nou.

2.4 Offline Link

În zilele noastre, să te afli într-o situaţie fără conexiune la internet ni s-ar părea ceva imposibil. Deşi conexiunea 5G încă este în discuţii adoptată de unele ţări şi interzisă complet de altele (la nivel de oraş), aria de acoperire a reţelei actuale are câteva puncte oarbe ce îţi poate crea dificultate în a utiliza o aplicaţie care să nu suporte funcţionalitatea de a o folosi fără conexiune la internet.

La sfârşitul anului 2018, Uniunea Internaţională de Telecomunicaţii, contrar opiniei publice, estima că doar 51.2% din populaţia globului are conexiune la internet, luând în calcul şi conexiunile de câţiva kbit/s unde astfel de conexiuni sunt preponderente în regiunile subdezvoltate din Africa, Orientul Mijlociu şi America de Sud.

O actualizare a datelor (Martie, 2019) a celor de la Internet World Stats arată o creştere considerabilă la nivel mondial, însă cea mai mare creştere a fost în Africa.

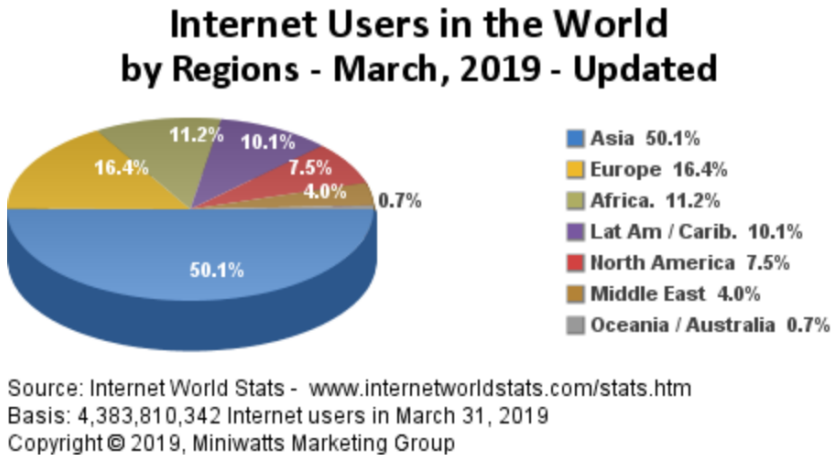

Figura 2.1: Numarul de utilizatori de internet

Cu toate acestea 5 milioane de americani nu sunt conectaţi la internet, aflându-se

în zone rurale unde costurile pentru dezvoltarea infrastructurii sunt destul de ridicate.

Chiar şi unele industrii suferă de lipsa conexiunii, cei ce practică expertize auto în hale bine izolate sau agenţii de vânzări ce se află pe teren şi se regăsesc într-un punct mort din care contactarea clienţilor devine mai dificilă.

2.4.1 Conceptul de bază

Realizarea unui mecanism ce poate adăuga capabilităţi de rulare în mediul offline a aplicaţiilor mobile cât şi web, este unul la fel de dificil precum problema în sine.

Apollo ne oferă posibilitatea de a crea şi de a compune lanţul de linkuri prin care cererea trimis de client ajunge la server.

Figura 2.2: Apollo Link

Astfel prin concatenarea offline linkului la lanţul prin care trece o cerere a unui utilizator putem să adaugăm diferite capabilităţi de modelare şi administrare a datelor ce urmează a fi trimise că răspuns.

Figura 2.3: Concatenarea unui Link la Apollo

2.4.2 Abordări anterioare

Merită menţionat abordările anterioare ale comunităţii apollo graphql ce au încercat să vină cu o soluţie pentru adăugarea capabilităţilor offline în integrarea cu graphql.

În prima versiune a clientului apollo, arhitectura tehnologiei se baza în mare parte pe integrarea cu redux, astfel se putea face o administrare şi stocare a datelor într-un singur store. Decizia de a se îndepărta de redux şi de a veni cu o soluţie proprie la problema menţinerii datelor la nivelul clientului a nemulţumit comunitatea, însă pe termen lung acea decizia a schimbat modul în care gândim structurarea datelor.

O abordare nouă, un mecanism de caching nou şi adăugarea unui link pentru prelucrarea datelor la nivel de client a creat şi mai mult confuzie pentru implementarea unor capabilităţi ce transformă aplicaţiile să funcţioneze în modul offline-first.

În continuare voi prezenta anumite pachete ce poate atinge un nivel minimal pentru rezolvarea problemei.

- apollo-link-retry, este o soluţie simplă pentru a rezolva problema siguranţei conexiunii la internet. Soluţii ajustabile pot fi adăugate cu uşurinţă, dacă vorbim despre un maximum de încercări admise pentru a executa o cerere cu un status OK. Fiind o abordare minimale, reuşeşte totuşi să acopere un scenariu în care utilizatorul suferă o pierdere a conexiunii. O îmbunătăţire a acestui scenariu poate fi adăugarea unei întârzieri în execuţia re încercării, dar utilizatorul poate rămâne fără conexiune pentru o perioadă mai îndelungată şi când vine vorba de instant messaging, se poate crea o situaţie cel puţin neplăcuta când un mesaj eşuează în a fi transmis destinatarului.

- apollo-cache-persist, un mecanism de persistare a întregului cache pentru a evita situaţia în care utilizatorul face o cerere ce are că timpi de execuţie mai mare. Astfel implementând acest pachet în aplicaţia de instant messaging, putem să-i oferim utilizatorului access la datele deja cerute într-o sesiune anterioare că mai apoi să fie actualizate cu noile date ce vin că răspuns la noile cereri. Cu toate acestea, nu poate să stocheze mutaţiile (POST method) pe client.

- apollo-link-queue, este o altă abordare în ceea ce priveşte rezolvarea cererilor de pe client, în funcţie de starea aplicaţiei, coadă poate fi închisă sau deschisă. De exemplu, dacă utilizatorul este conectat la internet, cererile acestuia nu vor fi afectate în niciun fel, pe de altă parte dacă acesta suferă pierderi ale conexiunii coada prin care trec cererile va fi închisă şi va urma să fie redeschisă când reconectarea la internet este cu succes. Deşi ideea acestei abordări este una destul de acceptabilă, consumatorul de aplicaţii de instant messaging poate pierde datele respective prin închiderea şi redeschiderea aplicaţiei.

Prin urmare, îmbinarea celor trei pachete poate acoperi o bună parte din rezolvarea problemei conectivităţii la internet, dar avem nevoie de un mecanism mai bune pentru administrarea cererilor. Un mecanism care să ne asigure actualizarea datelor dacă acestea au fost editate, dacă s-au create alte noi în timp ce conexiunea la server nu era asigurată sau chiar dacă s-au şters din acestea.

2.4.3 Implementare

În această implementare am reuşit să implementez un mecanism ce abordează câteva dintre cele mai populare scenarii în care se pot afla utilizatorii.

Scenarii

1. **Scenariul Multi Device Connection.** În viaţa cotidiană tindem să fim conectaţi pe mai multe dispozitive în acelaşi timp. Cazurile meteorologice severe pot afecta reţeaua iar pierderea conexiunii în acea zonă ar fi inevitabilă. În schimb smartphone-ul poate avea că plasă de siguranţa folosirea datelor mobile şi prin mecanismul de reconectare a websocketului schimbul de mesaje cu alţi utilizatori nu ar fi deloc afectat.

   Scenariul reprezentat în **Figura 2.1** este cel mai des întâlnit când pierderea conexiunii nu este sesizată de utilizator. Dacă comunicarea se va realiza prin aplicaţia de instant messaging de pe platforma web, mesajele trimise de utilizatorul în cauză vor fi puse în aşteptare, dar pentru a putea a oferi un răspuns utilizatorului în ChatRoom va apărea mesajul trimis cu notificarea că acesta este în aşteptare. Conceptul este denumit Optimistic UI.

2. **Scenariul WiFi public în zone speciale.** Un alt caz deosebit pentru necesitatea unei aplicaţii cu capabilităţi offline este atunci când utilizatorul foloseşte un WiFi public. Fiind o conexiune nesecurizată, smartphone-ul poate să o detecteze şi să iniţializeze procesul de conectare la aceasta.

   Problema apare atunci când utilizatorul este în tranzit şi nu deţine detalii despre raza de acţiune a acelei reţele. Într-o aplicaţie de instant messaging această

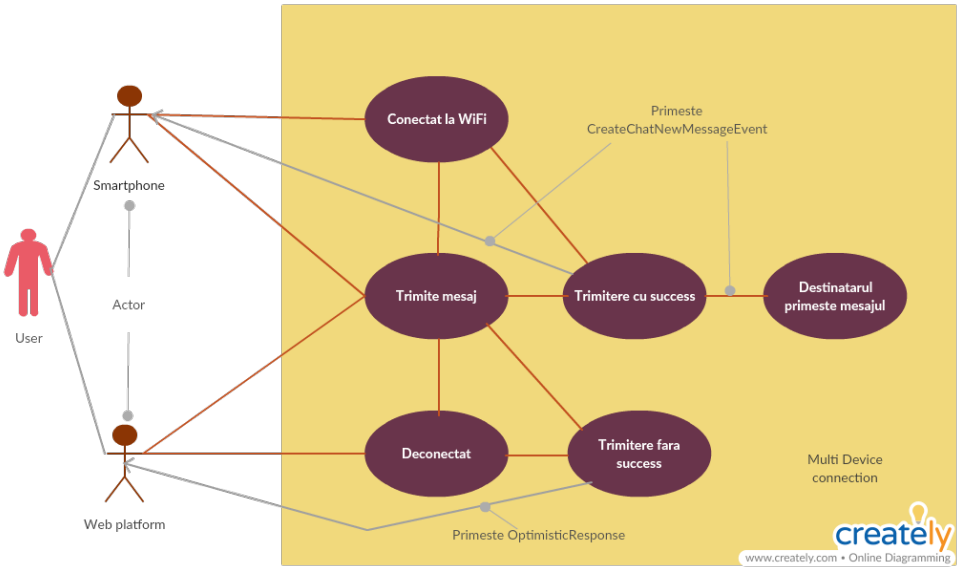

Figura 2.4: Multi Device Connection

situaţie devine îngrijorătoare deoarece datele introduse de consumator se pot pierde dacă nu este implementat un mecanism de persistenţă a acestora.

3. **Scenariul Colaborare.** Acest Offline Link dezvoltat fiind agnostic faţă de framework-uri sau platforme, dependent însă de Apollo GraphQL, este complet modular şi poate fi utilizat împreună cu alte tehnologii precum Vue.js, Cordova, etc.

   Un scenariu de utilizare pentru lucrul în echipă într-un mediu fără conexiune la internet trece peste limitele acestui mecanism, implementarea fiind una trivială în comparaţie cu alte tehnologii sau concepte existente. Acesta se aproprie de un concept bazat pe pesimism, regăsit în prelucrarea unui document Word pe platforma Microsoft Drive.

   Conceptul de pesimism presupune blocarea unei entităţi pentru restul utilizatorilor dacă există deja o persoană care şi-a atribuit rolul de editor a acelei entităţi, cel din urmă va avea drepturi complete de Write peste acea entitate în timp ce restul doar drepturi de Read.

   Neil Fraser oferă o soluţie mai bună la produsele bazate pe colaborare în timp real numită ”Differential Syncronization”, soluţie ce se regăseşte în produsele Google Docs, Google Sheets etc.

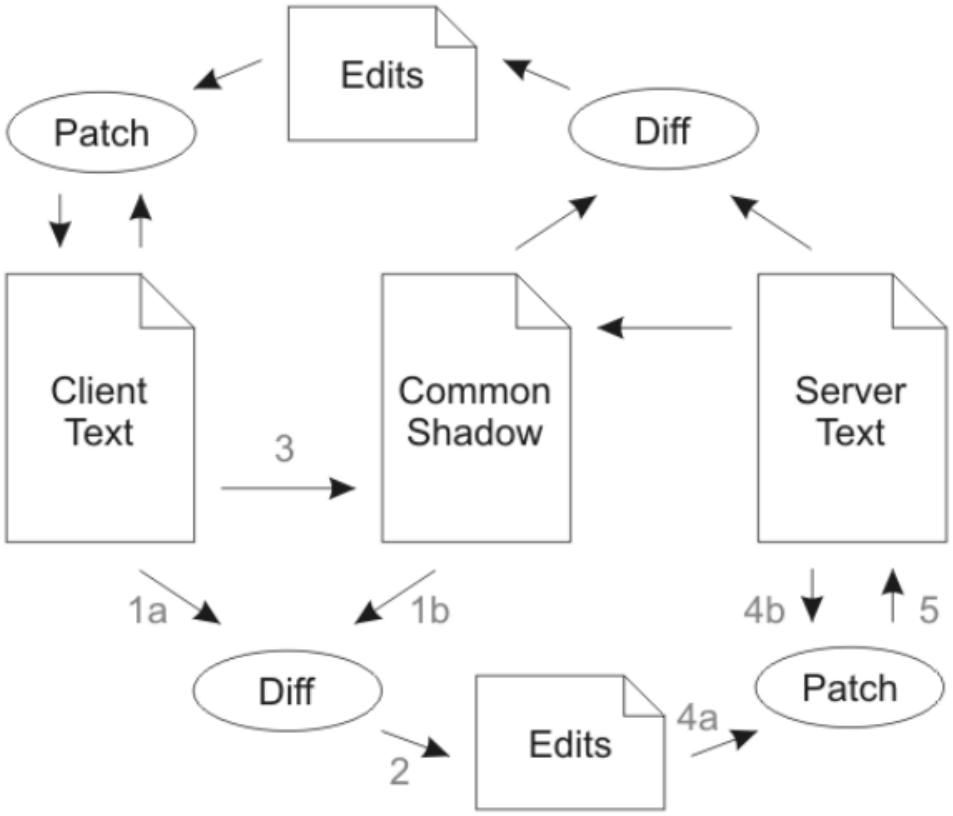

Figure 2: Differential Synchronization without a network.

Figura 2.5: Google Differential Synchronization

Contribuţii personale în realizarea Offline Link

Pentru că o aplicaţie să ofere suport modului offline trebuie să rezolve anumite probleme specifice:

- **Persistenţa mutaţiilor.** Permiterea utilizatorului de a utiliza aplicaţia fără conexiune la internet presupune implicit şi executarea unor cereri (Mutations) către server. Acesta abilitate a aplicaţiei rezultă în creşterea retenţiei utilizatorilor şi le oferă o experienţă inedită în ceea ce priveşte eventualele întreruperi sau un semnal semn de la reţea.

  În arhitectura Apollo sunt două scenarii posibile ce nu au fost incluse, când utilizatorul este în modul offline sau când instanţa unde a fost găzduit serviciul API nu mai este accesibilă.

  Soluţionarea acestei probleme vine prin crearea unui serviciu la nivel de client ce stochează cererile care eşuează în a fi executate cu succes într-o coadă. Folosind această structură de date ne permite să reluăm executarea cererilor în cauză într-un mod secvenţial când utilizatorul revine online.

Că acest scenariu să funcţioneze avem nevoie că cererea respectivă să fie ajustată prin formarea unei entităţi optimiste.

- **Interogarea entităţilor la nivel de client**

  Modulul offline include stocarea datelor (ChatRooms, ChatMessages, metadate, etc.) la nivelul clientului folosind tehnologia Redux pentru o gestionare mai bună a acţiunilor. Astfel când un utilizator eşuează să se conecteze la server, acesta va putea totuşi să folosească aplicaţia.

  Dezvoltarea unui astfel de mecanism implică creşterea complexităţii pentru gestionarea acţiunilor şi este predispus la apariţia conflictelor. De exemplu când un utilizator doreşte să creeze un ChatRoom, execuţia cererii va eşua, însă la nivel de client ştim cum arată structura de date ce urmează a fi trimisă, astfel putem genera un id unic la care concatenăm volumul de date ce urma a fi trimis către server.

  Dificultatea creşte când utilizatorul începe să trimită mesaje către destinatar deşi este în modul offline. Asigurând un răspuns optimistic la trimiterea de mesaje, putem să presupunem că toate cererile vor urma să fie rezolvate cu succes. Mutaţiile fiind persistente, acestea vor fi salvate că modificări pe o entitate ce nu se află în baza de date, iar reconectarea la internet declanşează sincronizarea ce trebuie să asigure actualizarea mutaţiilor cu noul ChatRoomId dar şi datele stocate la nivel de client.

Capitolul 3

Posibilitatea de extindere a aplicației

FCM si APNs

Implementarea actuală a modulelor ce construiesc aplicaţia de instant messaging a fost concentrată pe scenariile când un utilizator foloseşte aplicaţia în mod activ (starea aplicaţiei este activă; în foreground). Complexitatea creşte atunci când aplicaţia se află în modul background sau instanţa acesteia este complet terminată.

Când aplicaţia este în una din stările mai sus menţionate, conexiunea prin websocket este de asemenea terminată, ceea ce face imposibilă comunicarea cu serverul. Rezumându-mă strict la aplicaţia mobile realizată cu frameworkul cross-platform React Native, această limitare se poate rezolva prin înregistrarea şi integrarea certificatelor de Push Notifications.

Plecând de la ipoteza acestei lucrări, am evitat să folosesc serviciile FCM şi APNs pentru a acoperi şi acest scenariu, scopul acestei aplicaţii fiind să poată fi uşor de folosit şi de configurat în reţele închise sau chiar distribuite. O soluţie independentă de serviciile de top din piaţă ar fi că la nivel de client să se configureze modul de rulare a unor sarcini în background, de exemplu o interogare a serverului pentru actualizarea datelor (mesaje noi) şi lansarea unei notificări la nivel de aplicaţie pentru a atenţiona utilizatorul. O limitare a acestei abordări este frecvenţa cu care se execută sarcina respectivă, atât pe iOS cât şi pe Android, aceasta se adaptează după comportamentul utilizatorului cu dispozitivul respectiv (este posibil ca la executarea sarcinii respective utilizatorul să nu fie conectat la internet)

Concluzii

Prin această lucrare am încercat să ofer o alternativă la serviciile existente de pe piață, deşi cele open-source oferă şi posibilitatea de integrare în aplicaţii deja existente, scopul acestora este doar de marketing pentru a creşte audienţa produselor ce trebuiesc cumpărate pentru a fi folosite.

Fiind o soluţie ce oferă posibilitatea de a rula serviciul de instant messaging, am luat în considerare mai multe tipuri de utilizatori, putând acţiona ca parte integrată în aplicaţii comerciale oferind şi soluţiile de scalabilitate a serviciului de messaging în **Secţiunea 1.5**, dar şi ca o instantă într-o reţea privată, cum ar fi şcolile sau zonele lovite de conflict.

Soluţia propusă poate acţiona ca un întreg sau se pot folosi modulele integrate în realizarea acesteia, fie că dezvoltatorul foloseşte doar serverul pentru soluţiile de scalabilitate, fie că foloseşte doar **2.4 Offline Link** pentru integrarea pe o aplicaţie web ce foloseşte un alt framework decât React, sau fie că foloseşte interfaţa realizată în React Native pentru portabilitate, consider că am reuşit prin această lucrare să ating performanţa de a rezolva mai multe probleme, dar folosind o suită de tehnologii moderne.

Bibliografie

- Lianne Caetano, *https://securingtomorrow.mcafee.com/consumer/mobile-and-iot-security/viber-app-sends-data-unencrypted/*, 2014
- Author2, *Boook2*, 2017
- https://github.com/apollographql/graphql-subscriptions
- https://docs.meteor.com/api/pubsub.html
- https://galaxy-guide.meteor.com/apm-know-your-observers.html
- https://gist.github.com/OlegIlyenko/a5a9ab1b000ba0b5b1a
- https://github.com/apollographql/graphql-subscriptions
- https://github.com/davidyaha/graphql-redis-subscriptions
- https://www.theguardian.com/world/2011/jan/26/egypt-blocks-social-media-websites
- https://xmpp.org/extensions/xep-0124.htm#technique
- https://tools.ietf.org/html/rfc7692
- https://www.apollographql.com/docs/react/advanced/fragments/#fragment-matcher
- https://tools.ietf.org/html/rfc7395
- https://tools.ietf.org/html/rfc6455
- https://tools.ietf.org/html/rfc6120
- https://tools.ietf.org/html/rfc7692

- https://tools.ietf.org/html/rfc7578
- https://qz.com/1233010/before-we-solve-the-worlds-problems-we-need-to-connect-it-to-the-internet/
- https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx
- https://ai.google/research/pubs/pub35605