UNIVERSITATEA BABES-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICA SI INFORMATICA
SPECIALIZAREA INFORMATICA

LUCRARE DE LICENTA

Securitatea și confidențialitatea datelor în contextul aplicațiilor mobile

Conducator stiintific
Prof. Dr. Istvan CZIBULA

Absolvent
Lucian Edaurd Ghimpu

2019

Contents

1 Introducere

1.1 Problematica

În ultimi ani, piață dispozitivelor mobile a crescut în mod exponențial lucru datorat în primul rând dezvoltării tehnologice. Interesul oamenilor s-a schimbat ușor de la calculatoare de birou spre dispozitive mobile precum telefoanele și ceasurile smart.

Cantitatea de date pe care aceste dispozitive mobile o prelucrează este imensă, printre care și date sensibile și confidențiale. Adesea acest aspect este ignorat de dezvoltatori de aplicații mobile ducând astfel la aplicații mobile care în mod voluntar sau involuntar expun datele utilizatorului.

Lucrarea nu se limitează doar la problemele de securitate care pot apărea în cadrul unei aplicații, ci tratează și problematica confidențialității datelor și cum sunt ele prelucrate. Se tot pune problema dacă companii mari precum Facebook, Google și Amazon au dreptul de a avea acces la cantitatea uriașă de date fără consimțământul utilizatorului.

Scopul acestei lucrări este de a identifica astfel de probleme și de a propune soluții viabile. Trebuie făcută precizarea că această lucrare tratează doar probleme la nivel de aplicație și cum poate un dezvoltator să le evite sau să le prevină. Nu o să tratăm probleme de securitate de la nivel sistemelor de operare.

1.2 GDPR

The General Data Protection Regulation (GDPR) este un set de reglementări aduse în vederea protejării datelor persoanelor din Uniunea Europeană. Legea a fost adoptată în 25 Mai 2018 și are că principal scop împuternicirea indivizilor prin a le oferi control asupra datelor personale prelucrate de companii. Legea impune că orice proces comercial care gestionează date personale să o facă într-o manieră sigură și cu aprobarea utilizatorului.

Pentru a nu crea confuzii, legea definiste noțiunea de date personale ca orice formă de informație care poate fi folosită pentru identifica un idivit în mod direct sau indirect. În această definiție intră date precum: numele, numere de identitatea, locația sau date de o anumită natură (medicale, pshihologice, economice, genetice, economice, sociale, culturale) care pot fi asociate cu utilizatorul.

Printre reglementările propuse se numără:

- Abilitatea de a asigura confidențialitatea, integritatea, disponibilitatea și persistența serviciilor care prelucrează datele personale.
- Abilitatea de a asigura encriptarea datelor.
- Abilitatea de a oferi acces la date în cazul unui accident (backup).
- Impunerea unui proces de testare riguros care să asigure și să evalueze caracteristicile tehnice ale sistemului care procesează datele.

Abateri de la astfel de reglementări pot fi sancționate cu amenzi infunție de veniturile globale brute ale companiei.

Un alt aspect important al acestei legi îl reprezintă ideea de consimțământ, o declarație prin care se oferă în mod voluntar acces la anumite date unei anumite entități.

Legea este mult mai amplă și cuprinde mult mai multe aspecte legate de întregul proces de prelucrarea datelor, am reamintit aici doar câteva dintre ele. Putem totuși trage concluzia că aceste reglementări afectează în mod direct toți dezvoltatorii de soluții soft prin urmare naște necesitatea de a informa astfel de dezvoltatori de eventualele probleme și cum le pot prevenii.

1.3 Probleme etice si morale

Prelucrarea datelor cu caracter personal implică și o serie de probleme etice si morale pe care un dezvoltator de aplicații mobile și le poate pune.

Trebuie precizat de la început că utilizatorii au dreptul la confidențialitate, cea ce implică că dezvoltatorii să fie obligați să respecte acest drept. Pe lângă asta, există un set de principii etice care pot fi găsite în tratate internaționale legate de drepturile omului, a utilizatorului. Aceste drepturi și principii sunt incluse în documente precum:

- Declarația Universală a Drepturilor Omului din 1948
- Convenția Europeană a Drepturilor Omului din 1950
- Pactul Internațional privind Drepturile Civile și Politice din 1966

Unul din principalele principii care reiese din documentele menționate anterior este principiul autonomiei, principiu care se concetreaza asupra dreptului idividului la autodeterminare. Toate ființele umane au obligația de a respectă dreptul omului la autodeterminare. Din acest principiu se reflectă caracteristica de confidențialitate a unor anumite informații legate de persoane.

În cazul în care confidențialitate unei persoane nu este respectată asta poate duce în mod implicit și la o abatere de la drepturile omului. În același timp se pune o altă problema, dacă anumite date cu caracter personal care sunt deținute de anumite entiatați pot ajută acea persona în vederea prevenirii anumitor evenimente negative. De exemplu dacă o aplicația care prelucrează date medicale ar putea anticipa anumite boli ar trebui că această să aibe acces la date fără consințământul utilizatorului? Problema trebuie privită din ambele puncte de vedere și al companiei dar și al utilizatorului. În cazul în care datele respective pot avea un efect pozitiv asupra utilizatorilor dacă sunt prelucrate de companie, atunci utilizatorii ar trebuii să fie înștiințați.

Problemematica etică și morală atunci când e vorba de date confidențiale este una amplă și care trebuie analizată din mai multe puncte de vedere și prin mai multe filtre. Cert este că majoritatea companiilor și organizațiilor adopta sau sunt obligate să adopte un set de principii morale și etice pe care trebuie respectate.

1.4 Structura lucrării

În prima parte a lucrării vom aborda câteva aspecte teoretice legate de întregul proces prin care pot trece datele personale ale unui utlizator. Vom urmării procesul în patru etape:

1. Autentificarea și verificarea identității utilizatorilor. Metode prin pare un utilizator de poate autentifica și cum putem verifică identitatea lor. Vom încerca să găsim o soluție cât mai sigur dar și ușor de folosit pentru utilizatori.

2. Cererea de permisiuni către utilizator. Vom dezbate metode prin care putem cere consimțământul utlizatorului și de ce este el necesar.

3. Comunicarea și trasmiterea datelor. De multe ori există mai multe entități fizice (servere, cliente, baze de date etc...) prin care datele utilizatorilor trec, astfel în cât se impune folosirea unor canale de comunicare sigure.

4. Persistarea și criptarea datelor. În final o să vedem cum putem păstra datele utilizatorilor într-o manieră sigură.

Această nu este o lista exhaustivă ci doar urmărește unele dintre cele mai importante etape alese în funcție de regrementariile impuse de GDPR.

În a două parte o să prezint o aplicație mobilă, Medicarium, în care au fost aplicate aspectele teoretice discutate în prima parte. Aplicația urmărește prelucrarea datelor medicale ale utilizatorilor.

O să discutam despre cum putem proiecta o aplicație mobilă astfel în cât să devină flexibilă și ușor de modificat pentru a putea prevenii și soluționa orice problema legată de securitatea aplicației și de ce tehnologii ne putem folosi pentru a implementa o astfel de soluție soft.

2 Securitatea și confidențialitatea datelor în contextul aplicațiilor mobile

2.1 Autentificarea și verificarea identității

2.1.1 Importanța autentificării

Indiferent că vorbim de aplicații web, desktop sau mobile, majoritatea folosesc o metodă de autentificare. Auntentificarea șî înregistrarea stau la baza problematicii securității datelor. În clasamentul OWASP Top 10 din 2017 [1], problemele legate de autentificare și gestiunea sesiunii, sunt clasate pe locul 2. Iar în clasamentul OWASP top 10 Mobile din 2016 [2], autentificare nesigură și autorizarea necurespunzatoare sunt clasate pe locul 4, respectiv 6.

Unicitatea aplicaților mobile este dată de faptul că un dispozitiv mobil poate devenii accesibil oricărui persoane datorită portabilitatii lor. Un dispozitiv mobil poate fi furat, pierdut sau accesat temporal de o persoană necunoscută fără permisiunea posesorului. Prin urmare nevoia de un sistem de autentificare robust este mandatorie atunci când vorbim de aplicații care gestionează date sensibile (aplicații financiare, sociale, medicale, etc. . .).

În cadrul aplicațiilor mobile, autentificarea se poate face prin mai multe metode. De la simpla autentificare prin utilizator și parolă, până la utilizarea de sezori biometrici. Mitigari clasice precum impunerea unei parole sigure rămân valabile și în contextul aplicațiilor mobile.

Pentru alegerea metodei de autentificare trebuie să ne punem în primă instanța următoarele două întrebări:

- Care este scopul aplicației? O aplicație care notifica utilizatorul despre starea meteo poate că nu ar avea nevoie de autentificare prin senzori biometrici.
- Aplicația gestionează date confidențiale? Un exemplu potrivit ar fi o aplicație precum BT pay, care gestionează contul curent al unui utilizatorul, se folosește de mai multe metode de autentificare, o dată prin datele de logare, iar apoi prin senzori biometrici.

După ce ne este clar care este scopul aplicației și cu ce fel de date lucrează, putem include una sau mai multe metode de autentificare bazate după următorii factori:

1. Ceva ce utilizatorul știe (parolă, pin etc. . .)
2. Ceva ce il definește pe utilizator (amprenta, retina)
3. Ceva ce utilizatorul deține (parole generate temporal)

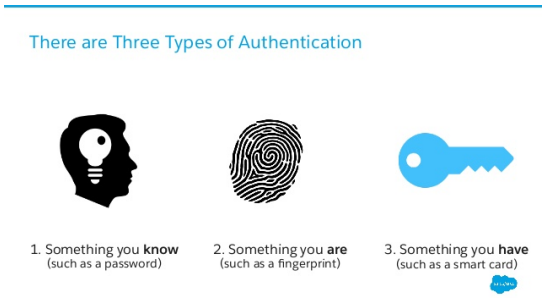

Figure 1: Metode de autentificare [3]

2.1.2 JWT

Majoritatea aplicațiilor de azi se folosesc de cea mai simplă formă de autentificare, prin folosirea de credentiale (ceva ce utilizatorul știe) și unui token de acces (ceva ce utilizatorul deține). Utilizatorul își creeza cont pentru o anumită aplicație, folosește credeințalele pentru a se loga, cererea de autentificare ajunge la server unde se verifică credențialele iar apoi se generează un token care va fi folosit de utilizator pentru a accesa diferite resurse în aplicație.

Scenariu descris anterior se referă la folosirea de JWT (JSON Web Token), un standard (RFC 7519) [4] adoptat de multe aplicații mobile în zilele noastre. JSON Web Token este o metodă sigură de autorizare a trasferului de informații între două părți [5], deobicei clientul mobil și serverul la care se face cererea. Clientul revendică de la server o dovadă, un token, care apoi este folosit de client pentru a accesa diferite resurse.

Din punct de vedere tehnic, un JWT are urmatoarea forma 11111.22222.33333 și este alcătuit din 3 parți:

1. Antetul (Header)
2. Datele utile (Payload)

3. Semnătura (Signature)

Figure 2: Antetul unui JWT, alcatuit din tipul de algoritm de înregistrare (HS256) și tipul de token (JWT).

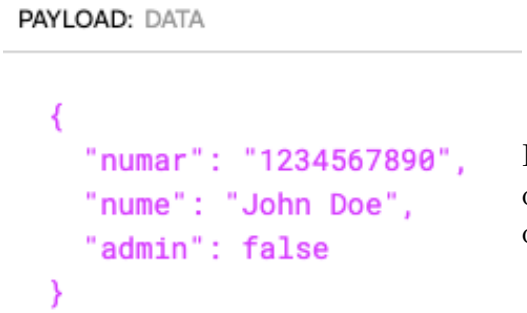

Figure 3: Partea utilă al unui JWT conține date sau permisiuni pe care clientul le are.

Figure 4: Semnătura unui JWT este alcătuită din antetul encodat, datele encodate, algoritmul folosit în antet și un secret. Semnătura are rolul de a oferi o metodă de verificare pentru a asigura ca conținutul nu a fost modificat

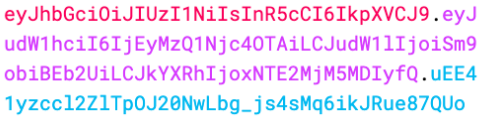

Figure 5: JWT va fi format în final de 3 siruri de caractere de tip Base64-URL separate prin punct.

O dată ce clientul deține un token, acesta trebuie tratat cu multă grijă în cadrul unei aplicații. Acesta poate fi folosit mai apoi în antetul tuturor cererilor de tip HTTP sub formă "Authorization: Bearer ¡token¿", din acest motiv cel mai probabil se dorește salvarea token-ului în memoria locală a dispozitivului pentru a putea fi apoi folosit în viitor. Acesta poate fi criptat iar la rândul lui la nivelul clientului, deși în mod nativ atât pe android cât și pe ios există metode sigure de stocare a datelor de tip primitiv (SharedPreferences în mod private pe Android și keychain pe iOS).

Pentru un nivel și mai mare de siguranță, se poate limita durata de timp pe care este valabil un token. Spre exemplu aplicația BT Pay folosește un token care este valid tip de 10-15 minute. După ce token-ul expiră, utilizatorul este nevoit să se autentifice din nou.

Avantajul principal pe care îl oferă JWT este facilitatea prin care se demarează tot procesul de revendicare a datelor sau drepturile de la un server de catre client. Un alt aspect important îl reprezintă faptul că în spate, totul se produce folosit obiecte de tipul JSON, fapt ce il face extrem de ușor de implementat și folosit in orice limbaj de programare.

2.1.3 Autentificare prin senzori biometrici

Biometria este termenul tehnic folosit pentru măsurătorile și calculele făcute legate de corpul uman. Se folosește de metrici legate de caracteristicile umane. În cazul dezvoltării de software, biometria este folosită pentru autentificare.

Autentificarea prin senzori biometrici se folosește de un factor moștenit, ceva ce îl definește pe utilizator și prin urmare este una dintre cele mai comode și rapide metode de autentificare. Mai mult decât atât, datele biometrice precum aprentă sunt greu de furat sau compromis.

Din ce în ce mai multe aplicații încep să folosească autentificare prin senzori biometrici, un factor major îl joacă faptul că în ultimi ani, capabilitățile hardware ale dispozitivelor mobile a crescut exponențial, telefoanele vin încorporate cu diferinti senzori biometrici precum: senzori de amprenta și recunoaștere facială (iris și rețină).

Metricile biometrice pot varia de la caracteristici fizice până la aspecte ale comportamentului unei persoane. În cea ce privește dispozitivele mobile putem idetifica trei tipuri de autentificari biometrice:

1. Senzor de amprentă, extrem de sigur deoarece fiecare individ are o amprentă unică.

2. Recunoașterea vocii, avantajoasă deoarece nu necesitâ hardware in plus dar nepotrivit pentru situații unde utilizatorul trebuie să păstreze liniștea.

3. Recunoaștere facială, la fel ca cea a vocii, nu necesită hardware adițional dar nepotrivit pentru locuri în care luminozitatea este scăzută.

Utilizarea senzorilor biometrici implică anumite aspecte de care un dezvoltator de aplicații mobile trebuie să țină cont:

- Verificarea ca dispozitivul mobil este încorporat cu senzorii folosiți, în cazul în care un dispozitiv nu are senzorii biometrici, dezvoltator trebuie să ofere o metodă alternativă de autentificare. [6]

- Cererea de permisiunea petru folosirea senzorilor.

- Verificarea datele biometrice asociate dispozitivului să nu se modifică de la prima autentificare. Această măsură trebuie luată pentru a împiedică cazuri în care se adaugă noi date biometrice (amprenta nouă).

Deși autentificare prin senzori biometrici este mai rapidă și comodă, această nu ar trebuii să înlocuiască în mod complet autentificare făcută la nivel de server. Ambele metode pot coexista în funcție de context.

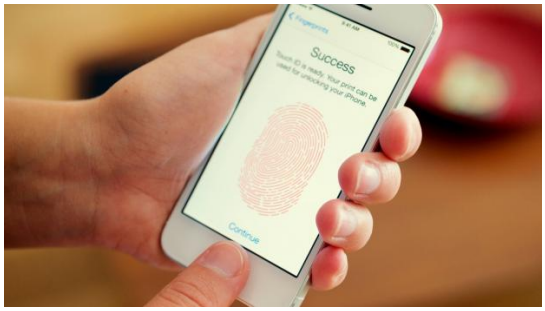

Figure 6: Autentificare prin senzor de amprentă

2.1.4 Autentificare prin factori multiplii

Autentificare prin factori multiplii (MFA) este un sistem de securizare a autentificării prin impunerea a mai multor metode de verificare a identității unui utilizator.

Autentificare prin factori multiplii combină mai multe metode independente de autentificare. Așa cum am văzut în capitolul anterior, autentificarea prin senzori biometrici și autentificarea prin credentiale lucrează cel mai bine împreună. Pe această idee, rolul autentificării prin factori multiplii este acela de a creea un sistem greu de compromis în vederea asigurării siguranței și confidentialității datelor utilizatorului. Astfel dacă unu din componentele sistemului este compromis, atacatorul este oprit de restul barierelor. În cazul în care cineva are acces la credentialele unui utilizator și la dispozitivul sau mobil, atacatorul poate fi oprit prin folosirea senzorului de amprenta.

Un alt caz de utilizare al autentificarii prin factori multiplii îl reprezintă autentificarea în doi pași sau OTP (one time password). Rolul autentificarii în doi pași este acela de a crea o barieră în plus în sistemul de securizare al autentificarii prin creeare de coduri/parole temporare unice.

După ce un utilizator se loghează folosind credențialele valide, un cod unic și temporar este generat și trimis utilizatorului prin diferite canale, deobicei prin SMS, email sau aplicații special făcute pentru generarea de coduri unice. Utilizatorul este apoi nevoit să introducă codul primit pentru a își confirmă identitatea. Utilizarea sa nu se limitează doar la autentificare, ci poate fi folosită în mod general pentru a confirmă identitatea persoanei. Un exemplu îl reprezintă aplicațiile financiare care atunci când se încearcă o plata, vor trimite un cod unic pentru a verifică identitatea persoanei care a inițiat acțiunea.

Deși folosirea autentificarii prin doi pași este destul de comună în cadrul aplicațiilor mobile, această metodă prezintă și anumite vulnerabilități, mai ales când canalul de comunicare a parolei este prin SMS sau prin apel telefonic. SMS-urile pot fi interceptate și redirecționate, la fel și apelurile telefonice. În astfel de cazuri se poate limita valabilitatea codului primit la un iterval scurt de timp (5-10 minute).

Utilizarea unui sistem de autentificare multiplu precum autentificare în doi pași este de altfel recomandată și de ENISA [7] într-un studiu făcut în vederea siguranței procesării datelor personale de către companii mari.

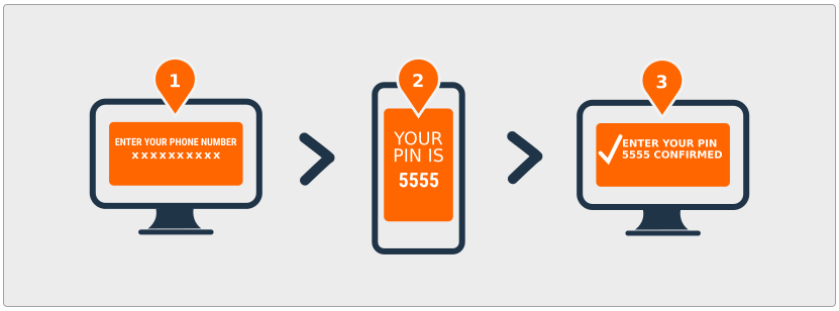

Figure 7: Autentificare în doi pași

2.2 Permisiuni

Permisiunile sunt un aspect important în vederea păstrării condifentialitatii datelor utilizatorului. O aplicație poate cere o anumită permisiune în diferite cazuri, pentru a accesa date personale (contacte, mesaje) sau pentru a accesa anumite funcționalități și senzori (apeluri telefonice, camera, senzori biometrici).

În medie un dispozitivil mobil are instalate 95 de aplicații, fiecare aplicație având nevoie de 5 permisiuni [8]. Problematica permisiunilor este adesea ignorată și de utilizator și de dezvoltatorul aplicației. În trecut, pe dispozitivele cu sistem de operare Android, când o aplicație trebuia să determine SSID-ul rețelei curente nici o permisiune nu era necesară. Acest lucru putând duce la determinarea locației utilizatorului bazat pe numele și SSID-ul rețelei [9]. Spre exemplu dacă rețeau s-ar numi "Gara Nord București", locația utilizatorului ar fi putut fi dezvăluită fără permisiunea lui. În versiunile noi de Android, determinarea SSID-ului se poate face doar dacă utilizatorul a dat consensul pentru folosirea locației sale.

În același timp, dezvoltatorul aplicației nu trebuie să abuzeze de permisiuni. Cererile directe către utilizator ar trebuii să se limiteze la funcționalitățile aplicației. Spre exemplu, o aplicație de timp calculator nu ar avea nevoie de permisiunea pentru apeluri telefonice.

Pentru a combate astfel de cazuri, google a început dezvoltarea unui algoritm pentru determinarea dacă o aplicație folosest ein mod eronat anumite permisiuni [10].

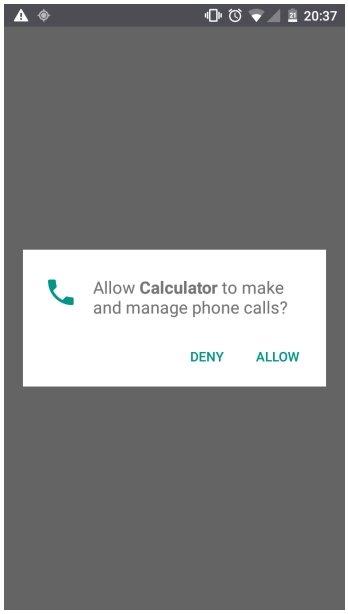

Figure 8: Abuzarea de permisiuni [11]

Pentru o gestiune corectă a permisiunilor se mai recomandă următoarele:

- Cererea unei anumite permisiuni să aibe loc doar atunci când este strict necesară. În cazul în care o aplicație depinde în mod permanent de o anumită permisiune (permisiunea de locație pentru aplicații de navigare), atunci cererea se poate face la deschiderea aplicației.
- Trebuie luat în considerare și diferitele versiuni ale sistemului de operare. Atât pe Android cât și pe iOS pot fi necesare diferite permisiuni pentru aceași acțiune. Exemplu cu SSID amintit anterior.
- Folosirea unei singure permisiuni în loc de un întreg grup. De multe ori mai multe permisiuni sunt grupate penru a facilita cererea lor. Dar în cazul în care doar un element din grup este folosit atunci este recomandat folosirea doar a elementului respectiv.
- Expunerea în mod clar a motivului necesității unei anumite permisiuni către utilizator.

2.3 Canale de comunicare

2.3.1 HTTP și HTTPS

Majoritatea aplicațiilor mobile se folosesc de unul sau mai multe servere pentru a își aduce date sau pentru a prelucra date preluate de la utilizator. Această practică este folosită pentru a evita stocarea de date pe dispozitivele utilizatorului având în vedere memoria limitată pe care o dețin. Din acest motiv este necesar folosire unor protocoale de comunicare. Cele mai folosite protocoale de comunicare în ecosistemul mobil sunt HTTP și HTTPS.

Aceste protocoale de comunicare sunt un set de reguli care descriu modalitatea prin care datele sun trimise și primite. În mediul mobil, HTTP și HTTPS sunt cele maifolosite protocoale pentru a trimite text imagini și sunete.

HTTPS este varianta sigură a lui HTTP, pe tot parcursul comunicării, datele sunt criptate de la un capăt la altul. Deși HTTP este mai frecvent folosit, este recomandată folosirea protocolului HTTPS mai ales pentru aplicații care gestionează date sensibile.

Dezavantul folosirii protocolului HTTPS îl reprezintă performanță. Criptare și decriptarea datelor trasmise sunt operații costisitoare. Deși există o pierdere de performanță, există studii [12] care demonstrează că diferența de performanță este modestă, și încurajează folosirea protocolului mai sigur. Un alt studiu [13] arată că pe anumite sisteme de operare mobile, precum Android, rata de adopție pentru HTTPS este în urmă față de rata de adopție pe desktop.

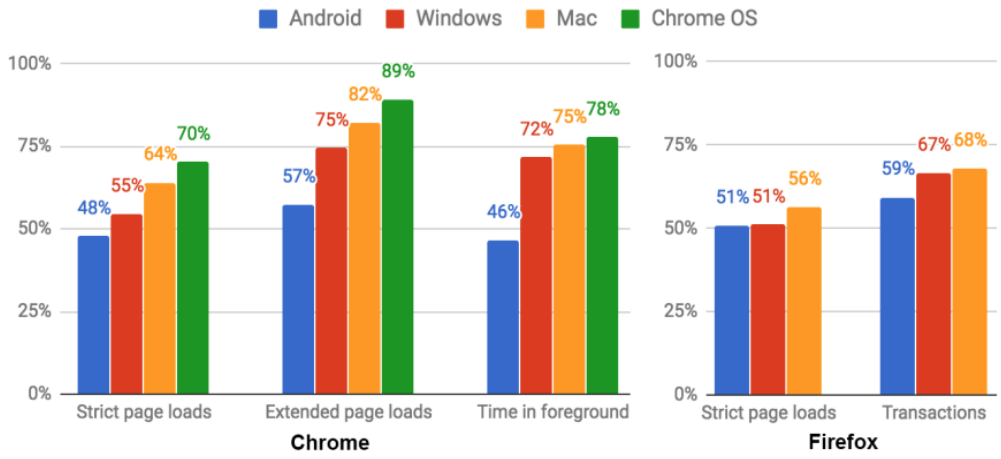

Figure 9: Procentul de utilizare HTTPS pe diferite sisteme de operare la sfarsitul unei saptamani din Febraruie 2017 [13]

2.3.2 SMS

SMS (Short Message Service) este un serviciu folosit de majoritatea dispozitivelor mobile. Cazurile de utilizare ale serviciului variază de la simpla folosire pentru a comunica mesaje scurte, până la autentificare cu parolă unică. În 2010, 6.1 trilioane de mesaje au fost trimise cu o medie de 193.000 de SMS-uri pe secundă [14].

Din punct de vedere tehnic, SMS este un protocol de comunicare care permite schimbul de mesaj scurte. Un SMS poate fi format din 160 de caractere alfanumerice, mesajele mai mari de 160 de caractere sunt sparte în mai multe mesaje. Un mesaj trimits este mai întâi interceptat de SMSC (Short Message Service Center) care de cele mai multe ori este menținut de providerii de rețele telefonice. SMSC-ul trimite mai apoi mesajul către un alt SMSC (al receptorului) mesaj pe care în final îl trimite receptorului.

În ciudat popularității sale, comunicare prin SMS prezintă anumite vulnerabilități. Cel mai periculoasă vulnerabilitate este "SMS spoofing". Aceasta are loc când un atacator manipulează adresa mesajului pentru a impersona pe cineva. Aceste tipuri de atacuri pot fi prevenite prin verificarea datelor emițătorului înainte că mesajul să ajungă la receptor. O altă metodă de protejare, adoptată mai ales în cadrul autentificării prin 2 pași, este folosirea

unui SMS gateway, servicii special dedicate pentru astefel de operații, dotate cu diverse măsuri de protecție.

2.3.3 WebSocket

Standardizat în 2011 în RFC 6455 [15], WebSocket este un protocol de comunicare bidirecțional bazandu-se defapt o conexiune de tip TCP.

Principalul avantaj pe care îl oferă protocolul WebSocket este facilitatea prin care se permite trasferul de date în mod bidirecțional. În comparație cu protocolul HTTP, un server poate transmite date unui client fără ca acesta să fie făcut o cerere.

La fel ca HTTP, protocolul WebSocket (WS) este dublat de variată protejată WSS, oferind criptare de la un capăt la altul al comunicării.

Când vine vorba de aplicații mobile, WebScocket-urile sunt folosite predominant de aplicații care au nevoie de o comunicare de tip broadcast. Aplicațiile de mesagerie în grup precum WhatsApp folosesc WebSocket-uri pentru a notifica toți utilizatorii unui grup de mesajele noi primite.

Deși WebSocket rezolvă probleme de conectivitate, nu rezolva și problemele de securitate [16]. vulnerabilități la nivelul acestui protocol variază de la simple interceptări ale datelor în rețea, atunci când se folosește varianta neprotejată a protocolului, până la vulnerabilități mai severe precum blocarea serviciului (DDos) [17]. Pentru a evita astfel de probleme este sugerată folosirea variantei protajate (wss) și limitarea conexiunilor sau verificarea conexiuniilor când provin din aceasă sursă.

Fiind o tehnologie încă tânăra, WebSocket încă nu este la fel de răspândit precum HTTP sau SMS, dar că orice sistem de comunicare, includerea lui într-o soluție soft necesită atenție sporită pentru a prevenii eventuale probleme de securitate.

2.4 Persistența datelor

2.4.1 Metode de persistare a datelor

Una din principalele atribuții a majoritatea aplicațiilor mobile este aceea de a gestiona și/sau stoca datele utilizatorului. Fie că vorbim de aplicații cu servicii web dedicate sau jocuri simple, toate aplicațiile au nevoie de o metodă pentru persistare datelor.

Deși modalitatea de persistare a datelor diferă în funcție de platforma (iOS sau Adroid), se pot extrage câteva caracteristici generale comune.

Dezvoltatorul aplicației trebuie să decidă unde și cum va stoca datele, în funcție de gradul lor de confidențialitate și tipul de dată. Folosirea în mod eronat a sistemelor oferite de platforma pe care se dezvoltă aplicația poate duce la expunerea de date sensibile.

Există mai multe metode prin care se pot salva datele și unde pot apărea vulnerabilități din punct de vedere al securității:

1. Fișiere Log, scopul lor, în mod normal, este acela de a păstra un jurnal al activității pentru o anumită aplicație, fiind ușor accesibile. Date sensibile pot fi expuse în mod involuntar prin astfel de fișiere (credentiale, token-uri, date primite din cereri HTML).

2. Baze de date locale SQL. Folosite pentru a păstra un volum mai mare de date. Pe ambele platforme există variate necriptate și criptate.

3. Fișiere folosite pentru setări (SharedPreferences și NSUserDefaults). Folosite pentru a stoca date de dimensiune mică cum ar fi valori pentru setări. La fel că bazele de dată SQL, există variante criptate și necriptate. Trebuie menționat că atunci când fișierele de preferințe sunt folosite pentru a stoca date de autentificare (token-uri) este recomandată folosirea variantei criptate.

4. Memoria internă, sistemul de fișiere intern al dispositivului poate fi folosit pentru a păstra date. Deobicei se poate folosii pentru a păstra fișiere publice de tip multimedia, cum ar fi poze sau videoclipuri.

5. Memoria externă (doar pe Android), asemănătoare cu memoria internă, disponibilă prin folosirea de carturi SD sau alte extensii hardware.

Majoritatea metodelor enunțate anterior sunt protejate într-un fel sau altul de sistemul de operare. În cazul dispozitivelor mobile modificate (jailbreak sau rooted), sistemele de protecție oferite de sistemul de operare ajung a fi compromise.

2.4.2 Criptografie

Dupa cum am vazut pana acum, multe aspecte ale unei aplicatii mobile pot fi securizate prin folosirea de elemente native securizate, librarii sau implemtarii proprii. Toate acestea au la baza o caracteristica comuna, criptarea datelor.

După cum am văzut până acum, multe aspecte ale unei aplicații mobile pot fi securizate prin folosirea de elemente native securizate, librării sau implemtarii proprii. Toate acestea au la baza o caracteristică comună, criptarea datelor.

Criptografia este studiul tehnicilor matematice legate de aspectele securității informațiilor, cum ar fi confidențialitatea, integritatea datelor, autentificarea entității și autentificarea originii datelor [18].

Problematica securității informației nu este una modernă, de a lungul istoriei au fost nevoie de diferite metode de securizare a informației, mai ales în timp de război. Deobicei implica păstrarea datelor într-o maniere incopresibila. Pentru a decripta astfel de mesaje, se puteau folosii diferite tehnici de decodare.

Criptografia modernă se rezumă la algoritmi bazați pe teorie matematică, greu de spart datorită complixitatii și ipotezelor pe care se bazează. Practic astfel de algoritmi pot fi sparți dar necesită o putere de calcul foarte mare.

Deși există multe tehnici folosite în criptarea datelor, cea mai populară și folosită în zilele noastre este criptarea prin cheie publică (criptografie asimetrică), folosită de instituii guvernamentale, armata și corporații mari.

Criptografia asimetrică presupune folosirea a două chei, una pentru criptarea datelor (cheia publică) și una pentru decriptarea datelor (cheia privată). Fiecare receptor are o cheie privată unică pe care o folosește pentru a de cripta datele. Emițătorul oferă cheia publică oricui, folosită pentru a cripta datele. Cheia publică și privată sunt deobicei într-o relație matematică dar calcularea lor nu este fezabilă.

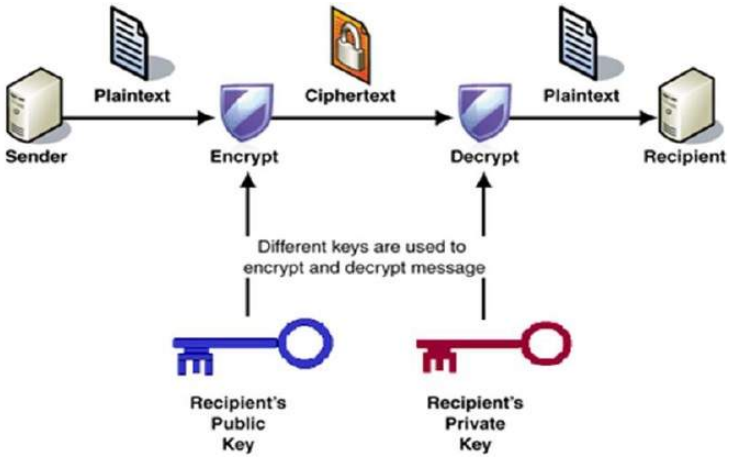

Figure 10: Criptografie Asimetrică [19]

RSA (Rivest–Shamir–Adleman) este cel mai folosit cripto sistem cu cheie publică. Relația matematică dintre cele două chei este securizată prin alegerea a două numere prime foate mari care sunt păstrate secrete [20].

3 Medicarium

3.1 Analiza aplicației

3.1.1 Problematica

"Mobile Health" (m-health) [21] este termenul folosit pentru a descrie utilizarea tehnologiilor mobile precum telefoanele pentru a sprijinii și îmbunătății sistemul de sănătate. Aplicațiile mobile au un potențial foarte mare când vine vorba de medicină, cazurile lor de utilizare variază de la simpla monitorizare a unor aspecte medicale până la comunicare între doctori, pacienți și instituții medicale.

În ultimii ani dispozitivele mobile au început să fie dotate cu din ce în ce mai mulți senzori pentru a oferii mai mult suport pentru astfel de aplicații. Creșterea în popularitate a adus ca urmare anumite reglementări. În iulie 2011, Administrația Statelor Unite pentru Alimente și Medicamente a emis un proiect de orientare privind reglementarea aplicațiilor medicale mobile [20]. Astfel se propune ca aplicațiile mobile medicale trebuie să gasesasca un echilibru intre înbunatațirea calității vieții utilizatoriilor și respectarea siguranței și confidențialității.

Medicarium este o aplicație care își propune centralizarea istoricului medical al utilizatorilor săi. În România există deja un astfel de sistem pin "Cardul de Sănătate". Din păcate nu toate persoanele din România au primit astfel de carduri, mai mult de cât atât, viitorul acestui sistem nu este cert. O altă problema a cardului de sănătate este faptul că utilitatea lui este limitată la sistemul de sănătate public din România.

Luând aceste probleme în calcul, Medicarium poate fi folosit oriunde, de oricine indiferent că vorbim de sistemul public sau private și indiferent de țară.

În prima faza Medicarium va fi folosit pentru a păstra istoricul medical al pacientului. Sunt stocate date generale ale pacientului (grupa de sânge, greutate, vârstă etc. . .), cat si documente medicale (analize, teste).

3.1.2 Cazuri de utilizare

Aplicatia Medicarium permite urmatoarele funcționalități:

1. Înregistrare și verificare cont.
2. Autentificare prin factori multipli (date de logare, pin și senzori biomerici).
3. Stocarea și editarea datelor medicale generale ale pacientului.
4. Stocarea documentelor medicale. Utilizatorul poate adaugă noi documente folosind camera sau din galaria telefonului.
5. Modificarea vizibilității datelor în caz de urgență.
6. Mod de urgență care paote fi folosit de orice persoană pentru a vedea datele medicale pe care utilizatorul le-a setat în prealabil.

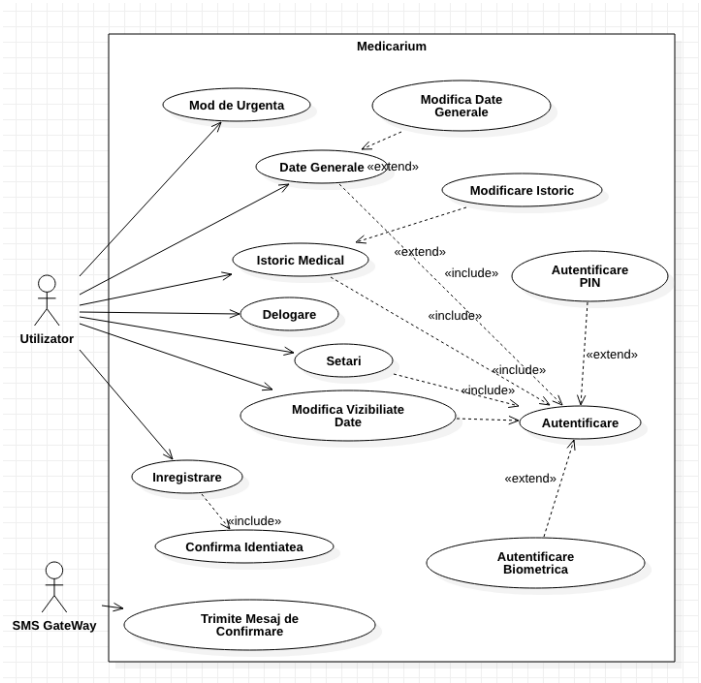

Figure 11: Diagrama Cazurilor de Utilizare

3.2 Proiectarea aplicației

3.2.1 Arhitectura

Solutia o sa fie compusa din 2 aplicatii mobile, clientul mobil propriu-zis si un "SMS Slave", o aplicatie separata folosita special pentru a trimite mesaje pentru autentificarea si verificarea utilizatorilor.

Clientul mobil este accesibil oricui pe cand aplicatia pentru mesaje este privata, fiind intretinuta impreuna cu serverele si baza de date.

La fel ca aplicatiile, avem doua server, un REST API care satisface cererile primite de clientii mobili si un SMS Gateway, care intermediaza comunicarea intre serverul REST si SMS Slave-ul.

Comunicarea intre servere se face prin HTTPS. Comunicarea intre SMS Gateway si SMS Slave se face prin WebSocket pentru a mentine un canal de comunicare bidirectional.

Server-ul REST comunica direct cu o baza de date nonrelationala de tip mongoDB.

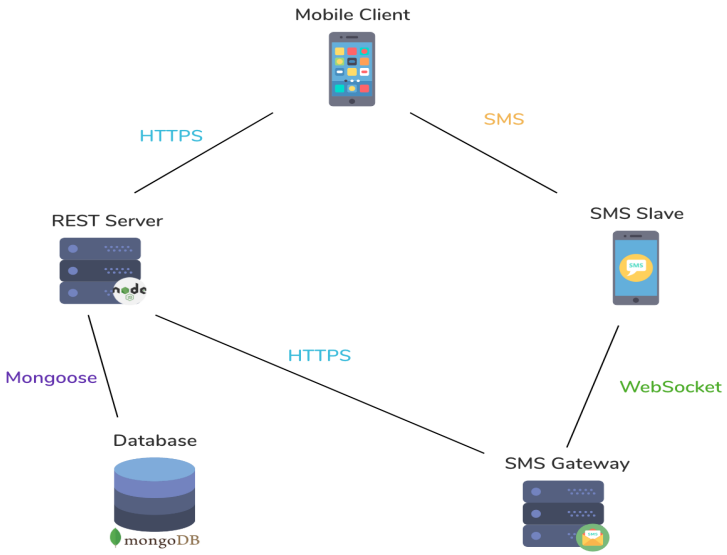

Figure 12: Diagrama de arhitectura a aplicatiei

Server-ul REST o sa fie impărțit in mai multe module dupa cum urmeaza:

- Un modul pentru Modele (Models), cara să conțină difinițiile schemelor entităților din baza de date.
- Un modul pentru Rute (Routes), care gestioneaza cererile primite la server
- Un modul pentru interceptorul de token (Middleware), cărui rol e sa verifice daca cererile către server au un token valid.
- Un modul de teste
- Modulul principal care cuprinde restul modulelor si contine fișierele server.js si index.js a caror scop e să centralizeze restul modulelor

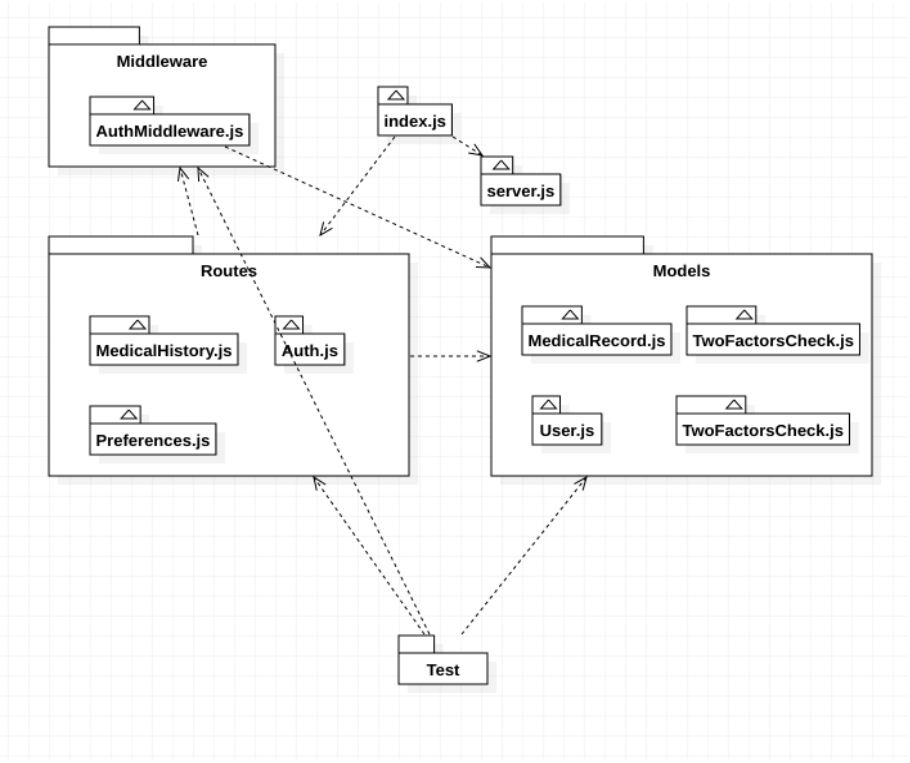

Figure 13: Diagrama de pachete pentru server

Clientul mobil o să fie și el împărțit in pachete:

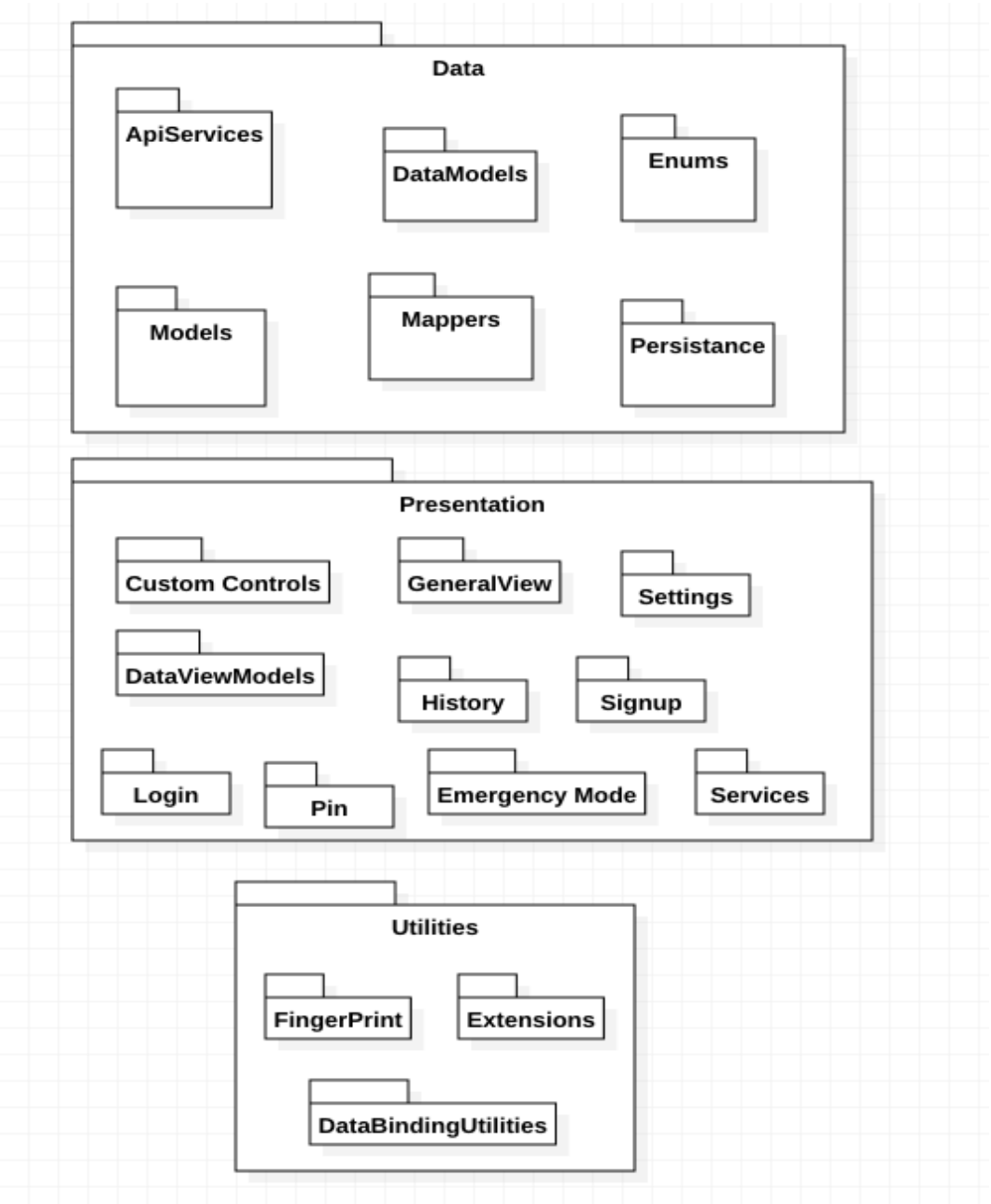

Figure 14: Diagrama de pachete a clientului mobil

Conținutul parțial al pachetului Data prin urmatoarea diagramă de clase:

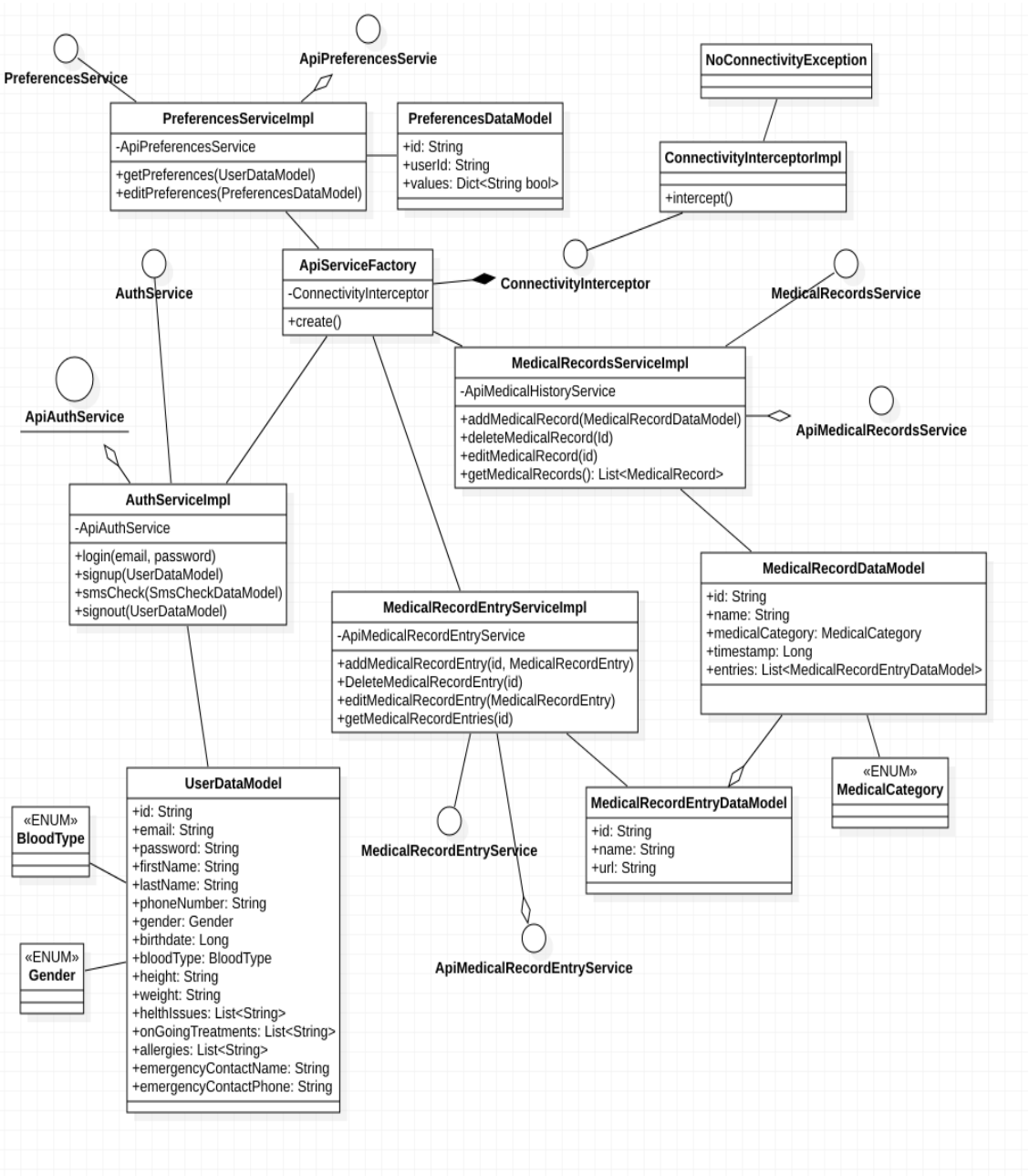

Figure 15: Diagrama de clase parțială a pachetului Data

3.2.2 MVVM

Pentru a crea o arhitectură ușor de menținut și ușor de modificat am optat să folosesc șablonul arhitectural MVVM.

MVVM este un șablon arhitectural original dezvoltat de Microsoft pentru Windows Presentation Foundation (WPF) framework-ul din .NET destinat dezvoltării de aplicații desktop. Este o variată a MVP-ului, un alt șablon arhitectural. Este predominant folosit pentru dezvoltarea de aplicații cu interfață grafică. Structural el este alcătuit din 3 componente:

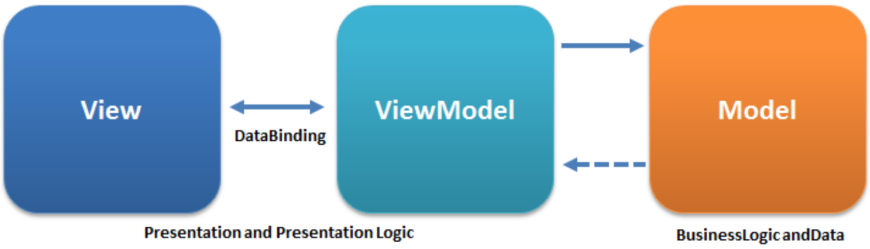

Figure 16: Componentele din MVVM

1. View. Se referă la un ecran sau la o parte din interfață grafică, ce e văzut de utilizator. Este responsabil de input-ul dat de utilizator. În cazul aplicației Medicarium, view-urile sunt toate ecranele pe care aplicația le are

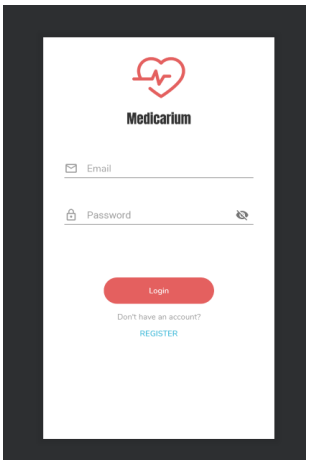

Figure 17: Exemplu de View bazat pe cod XML

2. ViewModel. Descrie starea datelor din model, care apoi este observată de View. Deobicei include un Binder carenotifică View-ul când au loc schimbări în Model. ViewModel-ul este decuplat de View, nu are nici o referință la el.

```
class LoginViewModel(
    application: Application,
    private val authService: AuthService
) : BaseAndroidViewModel(application) {

    private val _showDialog = MutableLiveData<String>()

    val email = MutableLiveData<String>()
    val password = MutableLiveData<String>()
    val isBusy = MutableLiveData<Boolean>()

    val showDialog: LiveData<String>
        get() = _showDialog

    val navigateToPinSetup = MutableLiveData<Event<Boolean>>()
    val navigateToSmsCheck = MutableLiveData<Event<Boolean>>()

    val isLoginEnabled: LiveData<Boolean> =
        Transformations.map(LiveDataDoubleTrigger(email, password)) {
            it.first!!.isEmailValid() && it.second!!.isPasswordValid()
        }
```

Figure 18: Exemplu de ViewModel pentru View-ul anterior

3. Model. Se referă la datele care sunt observate. Poate fi un model din aplicație sau un set de date. Unicul său scop e să păstreze datele.

```
data class User(
    var id: String = String.empty(),
    var email: String = String.empty(),
    var password: String = String.empty(),
    var confirmPassword: String = String.empty(),
    var firstName: String = String.empty(),
    var lastName: String = String.empty(),
    var phoneNumber: String = String.empty(),
    var gender: Gender = Gender.UNSPECIFIED,
    var birthDate: Long = DateTimeUtility.getCurrentDateInMs(),
    var bloodType: BloodType = BloodType.UNSPECIFIED,
    var height: String = String.empty(),
    var weight: String = String.empty(),
    var healthIssues: List<String> = ArrayList(),
    var onGoingTreatments: List<String> = ArrayList(),
    var allergies: List<String> = ArrayList(),
    var emergencyContactName: String = String.empty(),
    var emergencyContactPhoneNumber: String = String.empty()
)
```

Figure 19: Exemplu de Model din aplicația Medicarium

Deși Google nu impune un anumit stil arhitectural pentru aplicațiile mobile Android, MVP era cel mai folosit șablon. Asta s-a schimbat recent cu introducerea lui Android Jetpack, un set de librării care vin să ajute dezvoltatorii de aplicații și care favorizează folosirea MVVM.

Principalele avantaje a MVVM-ului față de MVP sunt:

- O separare mai bună a responsabilităților care implicit duce la o cuplare redusă. ViewModel-ul nu are nici o referință la View, el doar oferă o reprezentare a datelor care poate fi observată de cineva sau nu.
- Ușurează testarea. Componentele fiind redus cuplate între ele pot fi mai ușor testate. ViewModel-ul este o simplă clasa POJO sau POCO care conține doar propietăți și metode ușor de testat.
- Codul din ViewModel este ușor de refolosit, fâcand MVVM destul de popular pentru proiecte pe diferite platforme.
- Cod mai ușor de menținut și mai adecvat pentru metodologii de lucru precum agile.
- Funcționează foarte bine cu alte șabloane de proiectare (Command, Singleton, Factory etc. . .)

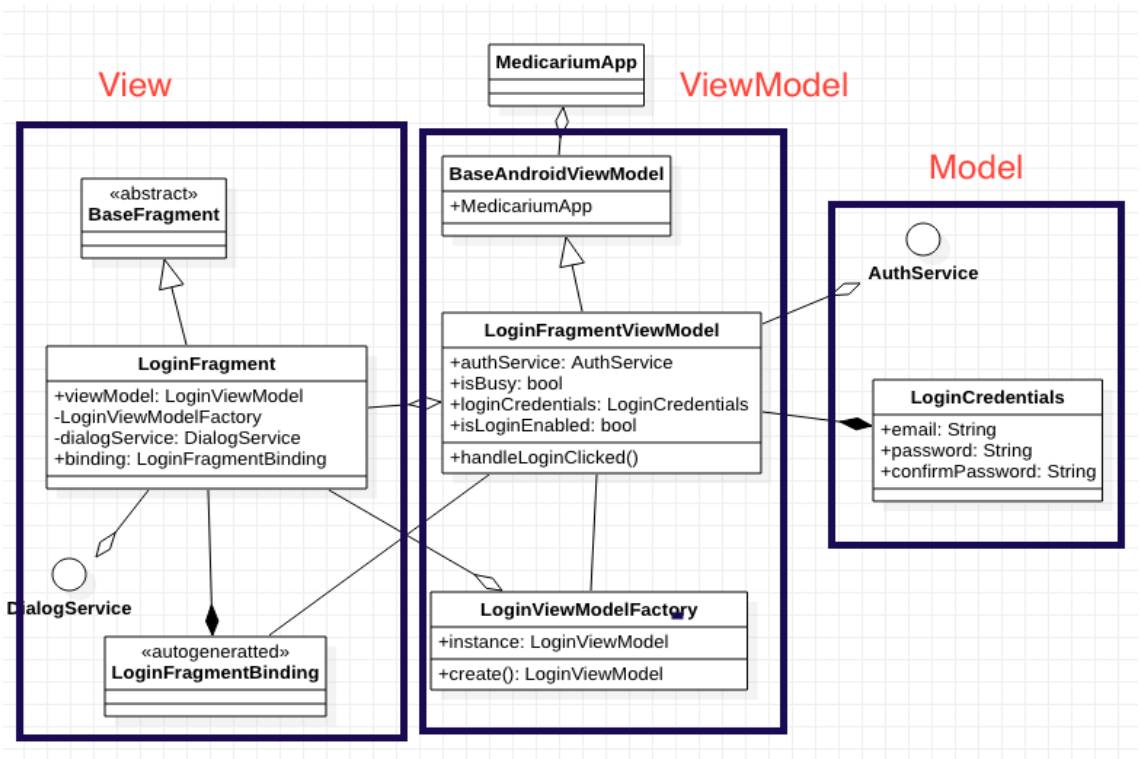

Figure 20: Diagrama de clase care evidențiază structura MVVM pentru funcționalitatea de login

Ca orice alt șablon arhitectural și MVVM prezintă puncte slabe:

- Mai mult cod. Deși codul devin mai ușor de menținut, cantitatea lui poate crește considerabil.
- Depanarea codului poate fi anevoiasă datorită sistemului de Databinding.
- Poate fi mai greu de înțeles la prima vedere, comparat cu MVC și MVP.

Deși MVVM se pretează cel mai bine pentru aplicații mari cu interfață grafică unde testarea este esențială, poate fi folosit și pentru aplicații de dimensiuni medii. Folosit cum trebuie poate ajută la o dezvoltare mai rapidă și la o calitate crescută a codului.

3.3 Implementarea aplicației - Serverul și serviciile

3.3.1 Server REST

Representational State Transfer (REST) este stil arhitectural software care definește un set de constrângeri pentru creearea de servicii Web. Un serviciu Web este un server web constriuit pentru a satisface toate nevoile unui site sau unei aplicații [23].

Principala caracteristică a acestor tipuri de servicii îl reprezintă faptul care serverul nu stochează nici o stare a sesiunii clientului. Fiecare cerere conține toate informațiile necesare pentru a putea înțelege cererea. Cererile au loc în izolare și sunt idependente, serverul nu se folosește de informații din alte cereri iar clientul este responsabil pentru trimitarea de orice dată e necesară pentru a obține răspunsul dorit.

Pricipalele avnataje ale unui serviciu REST sunt:

- Suport pentru diferite format de date (JSON, XML, text etc. . .)
- Performanță crescută și eficientă, serviciile REST folosesc puțină lățime de bandă
- Ușor de modificat, datorită unicității arhitecturii, coponentele sunt izolate ușuarând eventuale modificări
- Scalabilitate, fiind principalul motiv care a adus la adoptatea serviciilor REST [23]

În Medicarium, serverul REST este responsabil de a servii clienții mobili cu resurse de care au nevoie. Există o serie de rute la dispoziția clienților accesibile doar pe baza de Json Web Token (JWT).

Exemple de rute folosite în aplicatiie:

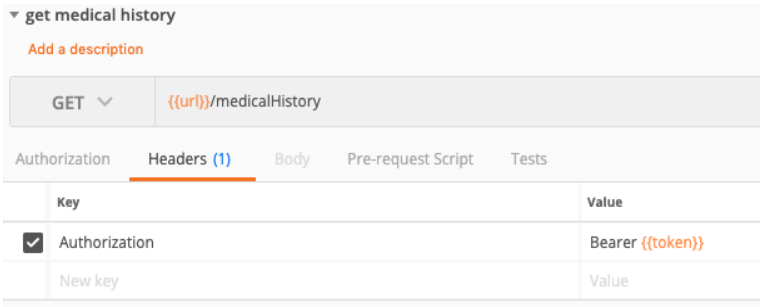

Figure 21: Exemplu de metoda GET cu JWT în antetul cererii

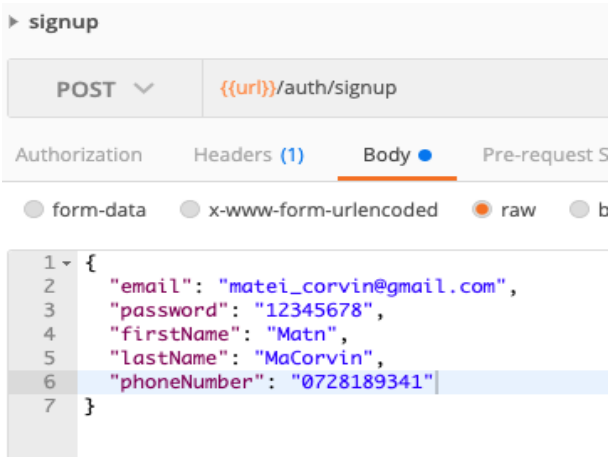

Figure 22: Exemplu de metoda POST cu JSON în corpul cererii

Figure 23: Exemplu de metoda DELETE cu mesaj de eroare

3.3.2 Node.js

Pentru implementarea serverlui REST s-a folosit Node.js, un mediu de rulare care permite execuția de cod javascript în afară browser-ului.

Node.js este dublat de npm, un sistem de gestionarea a pachetelor care permite customizarea unui proiect Node.js cu diferite pachete.

Există numeroase avantaje atunci când se folosește Node.js pentru dezvoltarea unui server. Fiind javascript, serverul în sine este mai lejer și mai ușor de întreținut. Mai mult de cât atât, este și mult mai rapid, operațiile de I/O fiind foarte rapide. Un alt avantaj important îl reprezintă numeroase librării și pachete disponibile în npm.

Servicii mari precum Ebay, Netflix și PayPal [24], folosesc Node.js, iar popularitatea acestui stă doar să crească.

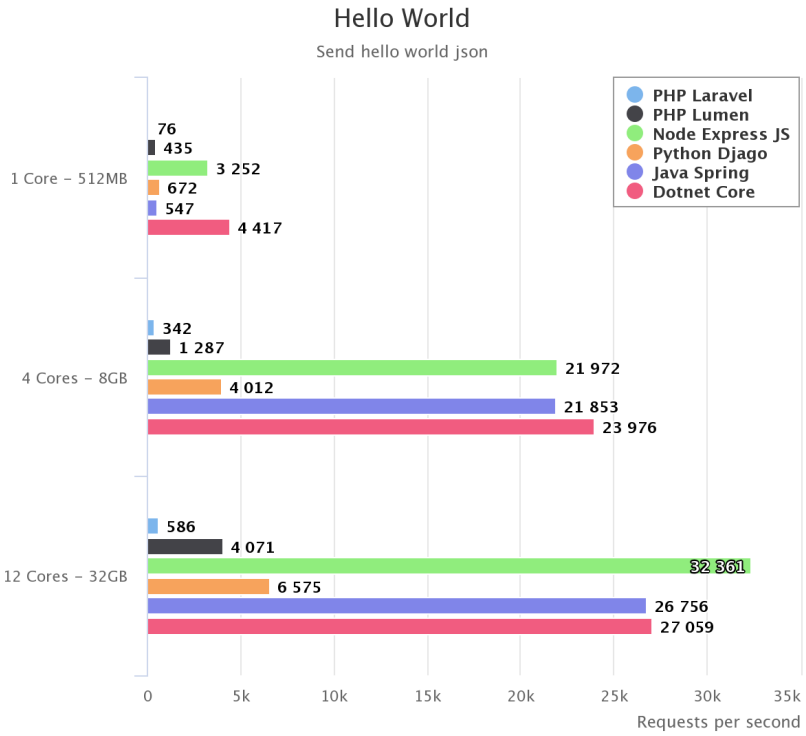

Figure 24: Performanță a diferitor framework-uri [25]

3.3.3 MongoDB

MongoDB este un tip de baza de date flexibi și scalabil. Combină abilitatea de a scala cu o multe funcții secundare precum: indexi secundari, interogări range, sortări, agregări și indexi geospațiali [26].

Mai mult decât atât, MongoDB intră în categoria bazelor de date non relaționale (NoSSQL). Astfel de baze de date oferă alternative când vine vorba de modelarea datelor, nu înlocuiesc complet sistemul relațional ci doar îl extind.

Câteva din avantaje folosirii unei baze de date NoSQL [27]:

- Mai rapide și eficiente
- Oferă o gama largă de modele de date
- Ușor de scalat
- Nu necesită administratori de baze de date
- Foarte populare și evoluează rapid

Există însă și câteva dezavantaje față de o baza de date relațională:

- Devenind populare destul de recent, încă sunt imature
- Nu există un limbaj standard pentru interogări
- Unele interpretări nu respectă ACID
- Mentenanță poate fi dificilă

Din punct de vedere tehnic, MongoDB lucrează cu "documente", obiecte de tip JSON care mapează entitatiile din aplicație. Este important de precizat că nu există o stuctură pe care documentele care mapează aceași entitate trebuie să o respecte. Spre exemplu, putem aveam două documente diferite care mapează o persoana, unul să conțină doar numele și vârstă, iar al doilea să conțină nume, vârstă și înălțime. Acest lucru permite o scalabilitate mai ușoară și în general este mai ușor de lucrat direct cu obiecte de tip JSON.

În aplicația Medicarium, folosirea unei baze de date MongoDB facilitează în primul rând comunicare între toate entitățiile care se ocupă de datele

aplicației. Un alt avantaj pe care îl aduce aplicației este eficientă și rapiditatea cu care sunt aduse datele. Diferența de tip pe care o aduce la citirea si scrierea de date conteaza foarte mult pentru o aplicație mobilă.

Acest sistem implica toate partile solutiei soft, de la aplicatia in sine, pana la aplicatia dedicata sa trimita doar mesaje.

Figure 25: Exemplu de documente păstrate în baza de date

3.3.4 Verificarea in doi pași

Pentru a crește nivelul de securitate al aplicației există un sistem de verificare în doi pași. Așa cum am precizant în prima parte a lucrării, adăugarea unui astfel de sistem poate fi foarte benefic pentru aplicații. Mai mult de cât atât, așa cum este precizat și în studiul făcut de ENISA [28], un astfel de sistem este recomandat pentru a face aplicația să respecte prevederile aduse în GDPR.

Un flow detaliat al unei astfel de verificări:

1. Un utilizator nou dorește să se înregistreze SAU un utilizator cu cont deja creat nu are contul verificat
2. Serverul REST primește cererea de la clientul mobil și va aduce date din baza de date pentru a verifică dacă utilizator are cont verificat
3. În cazul în care utilizatorul are cont verificat, poate intră în aplicație. În caz contrar, o cerere este trimisă la serverul care se ocupă de mesaje (SMS Gateway)
4. SMS Gateway o să salveze cererea în baza de date și va notifica dispozitivul mobil care se ocupă cu trimiterea de mesaje, care la rândul lui va trimite codul la utilizator.
5. Utilizatorul primește mesajul și introduce codul în dispozitiv, care apoi v-a fi verificat pe server. Dacă codul este valid, utilizatorul își poate continuă activitatea, altfel o să fie nevoit să mai încerce o dată.
6. Verificarea mai poate eșua dacă utilizatorul nu își verifică codul în decursul a 5 minute, valabilitatea codului fiind limitată din motive de securitate

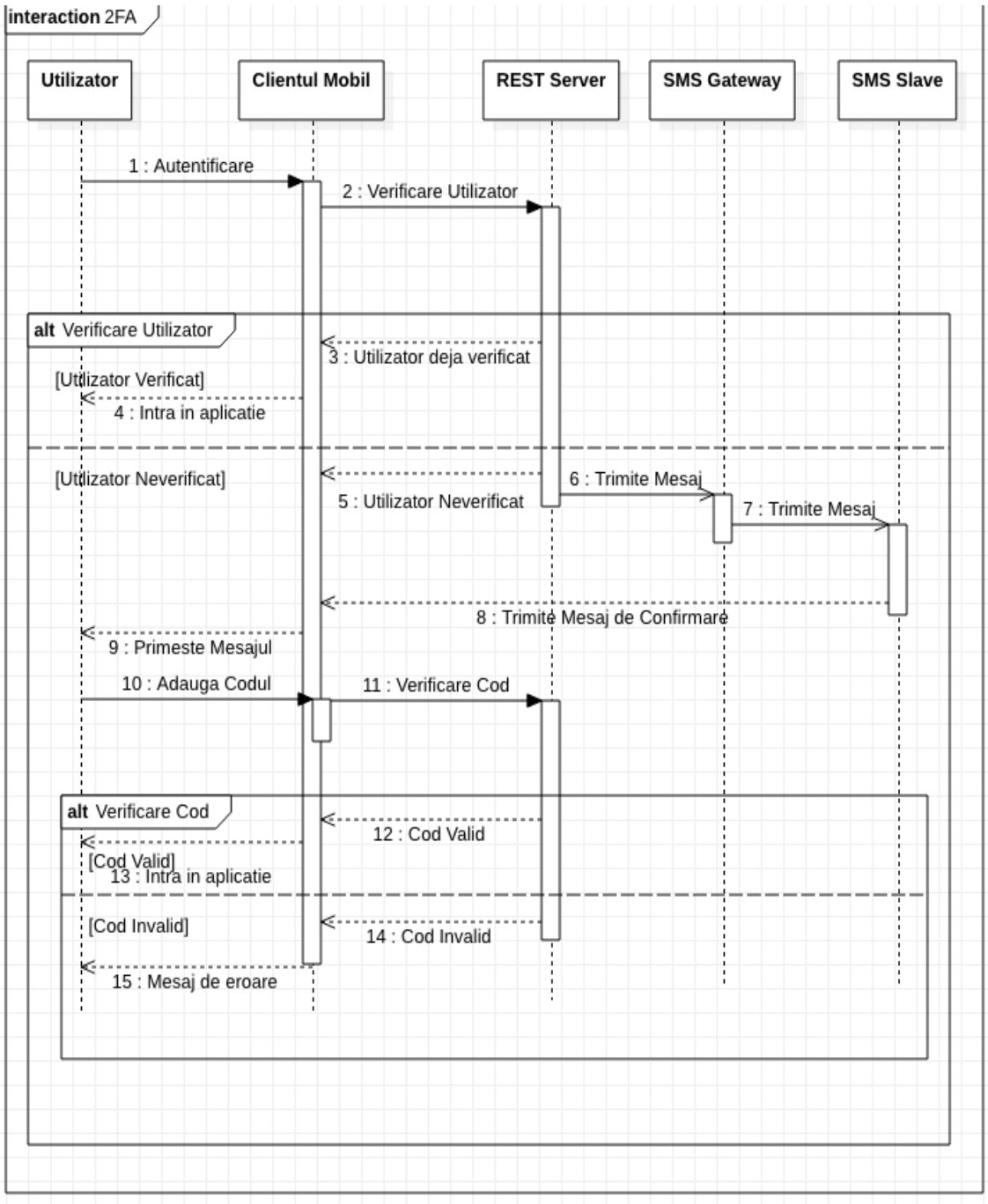

Figure 26: Diagrama de secvență pentru verificarea în doi pași
(baza de date este exclusă pentru a simplifica diagrama)

3.4 Implementarea aplicației - Clientul mobil

3.4.1 Android Jetpack

Pentru implementarea ambelor aplicatii, clientul mobil Medicarium si aplicatia pentru mesaje, am optat sa dezvolt pe Android, sistem de operare mentinut de Google. Este bazat pe un kernel Linux si este optimizat pentru a lucra pe device-uri cu touchscreen.

Mai mult de cat atat, am folosit Android Jetpack, o colectie de librarii dezvoltate recent care au in vedere imbunatatirea procesului de dezvoltare a aplicatiilor mobile.

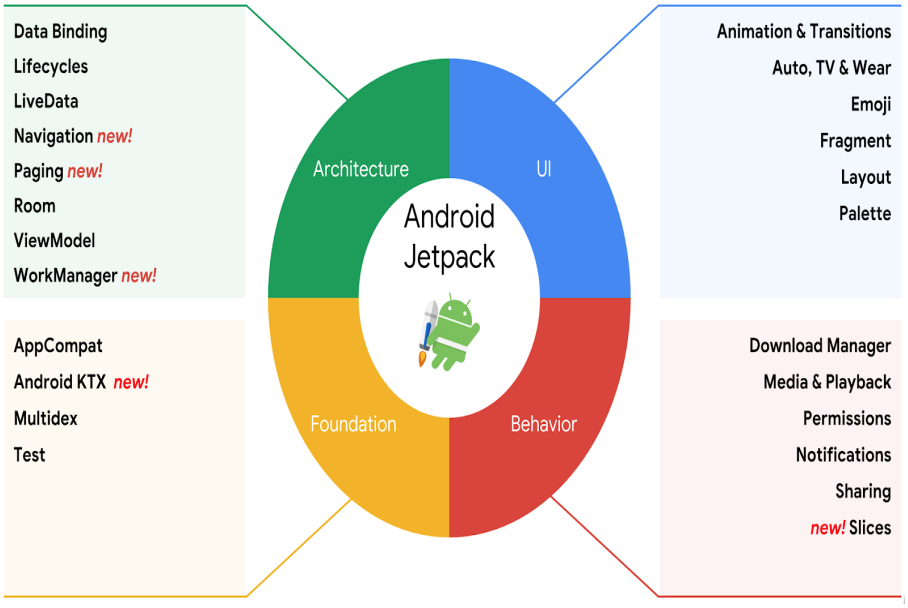

Figure 27: Componentele din Android Jetpack [29]

Motivele folosirii platformei Android:

1. Popularitea crescuta a sisemului de operare Android. Conform unei statistici [30] sistemul de operare Android ocupa 75% din piata mobila, pe cand iOS doar 22%

2. Sistemul de operare Android este mai vulnerabil de cat iOS. Acest lucru se datoreaza faptului ca fiecare vanzator de telefoane cu Android

are tendinta de a modifica putin sistemul de operare pentru al ajusta nevoilor proprii. Sistemul in sine este mentinut doar de Google, astfel se creaza discrepante intre diposizivele mobile cu acelasi sistem de operare dar producatori diferiti de hardware. Pe de alta parte, Apple detine controlul si al software-ului (sistemul de operare iOS) si al hardware-ului, fiind mai usor de impus masuri de securitate intr-un sistem inchis.

3.4.2 Kotlin

Daca pana acum nu mult timp sigura metoda pentru a dezvolta aplicatii pentru Android era Java, in ultimii ani un nou limbaj este disponibil, Kotlin. Kotlin este un limbaj care a aparut din dorinta de a rezolva anumite probleme de proiectare pe care Java le are si pentru a imbunatatii procesul de dezvoltare.

Kotlin este un limbaj nou, stabil care poate functiona pe orice device cu Android si rezolva multe probleme pe care Java nu a putut. Aduce multe concepte in plus, este un limbaj a carui scop e sa faca viata dezoltatorilor mult mai usoara [31]. Kotlin ruleaza pe JVM facand astfel ca orice cod scris in Java sa fie 100% compatibil.

Cateva avantaje pe care le ofera Kotlin fata de Java:

- Cod mai putin. Kotlin este un limbaj mai concis, codul este mai usor de scris si citit, crescand astfel eficienta dezvoltatorilor si reducand codebase-ul.
- Suport pentru programare functionala. Exista numeroase constructii si concepte ale programarii functionale care pot fi folosite in Kotlin.
- Null safety. In compartie cu Java, in Kotlin orice obiect care poate avea asignata valoarea null trebuie marcat cu nullable in caz contrar codul o sa esueze la compilare.
- Complet compatibil cu Java, trecerea de la Java la Kotlin fiind triviala.
- Multe alte concepte moderne (operatori ternari, corutine, extensii de metode etc. . .)

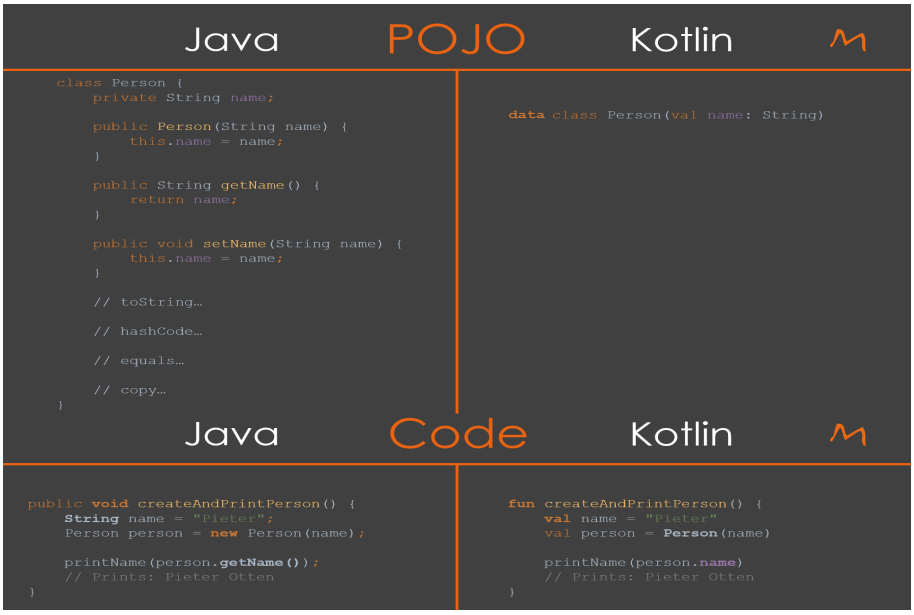

Figure 28: Comparatie intre cod Java si cod Kotlin [32]

Multe librarii au inceput deja migrarea din Java in Kotlin. Iar Google a anuntat ca limbajul principal pentru Android a devenit in mod oficial Kotlin.

3.4.3 Implementarea autentificării

În această ultimă parte vom vedeam cum am implementat în aplicație anumite necesități discutate în partea teoretic. Vom începe cu autentificarea și verificarea identității. Ca în orice aplicație, în prima instanța suntem întâmpinați cu două opțiuni, a ne loga cu un cont deja creat sau a crea un cont nou:

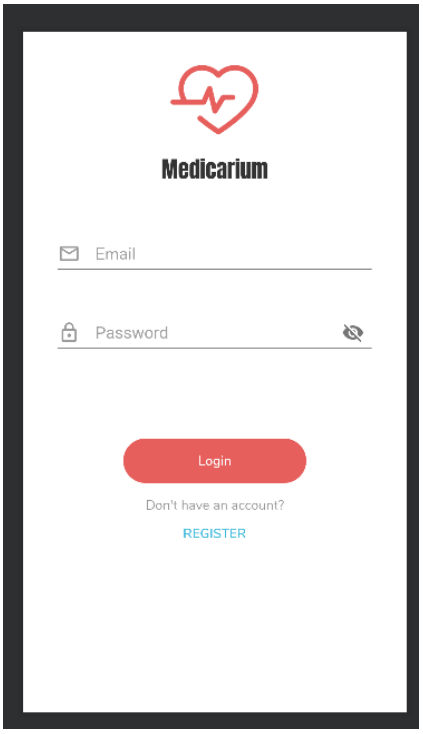

Figure 29: Ecranul de Login

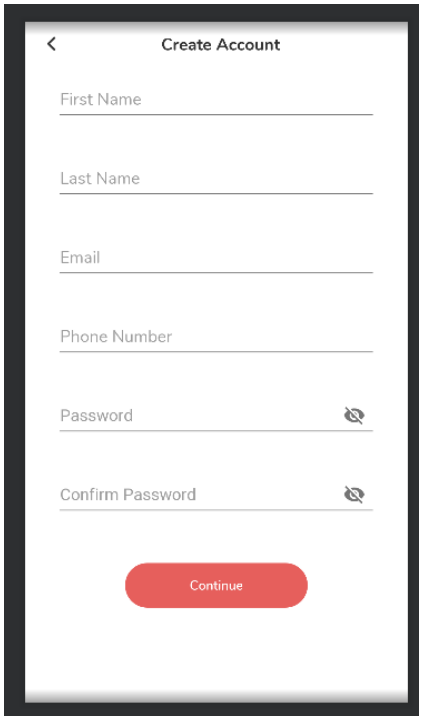

Figure 30: Ecranul de Register

În cazul în care contul utilizatorului nu este verificat, un mesaj o să fie trimis către numărul lui personal pentru a își verifică identitatea. Mesajul este trimis prin SMS Gateway, serverul separat care se ocupă doar cu trimiterea de mesaje de verificare.

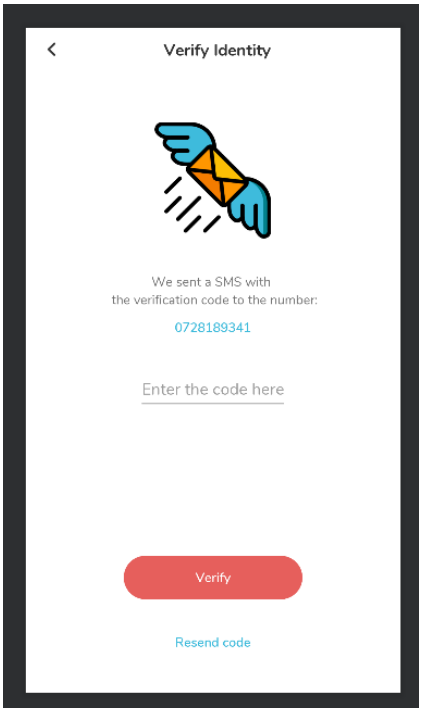

Figure 31: Ecranul de verificare a codului primit prin SMS

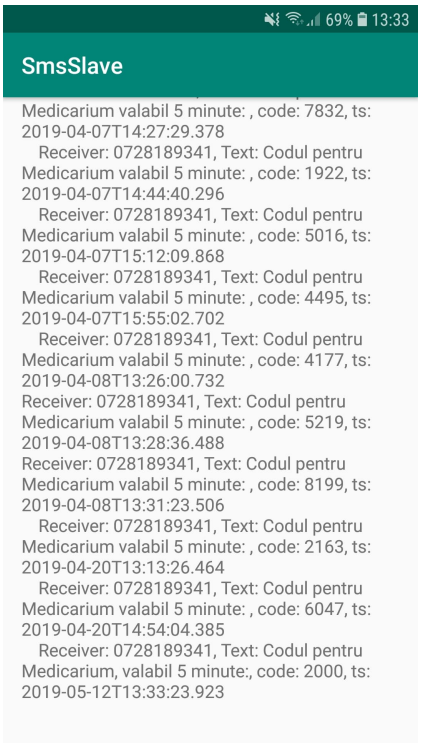

Figure 32: Aplicația SMS slave

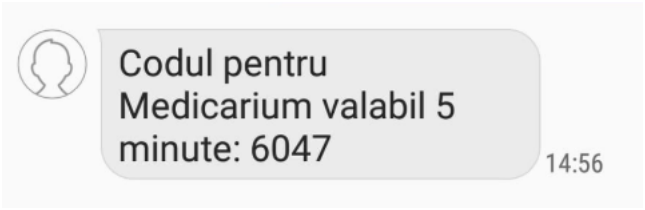

Figure 33: Model de SMS cu cod de verificare

O dată ce un utlizator are contul verificat acesta poate intra în aplicație în dou feluri, prin amprentă sau prin cod PIN. Deși varianta cea mai facilă ar fi prin aprentă, nu toate telefoanele au senzor astfel în cât se impune necesitatea implementării și a autentificării prin pin.

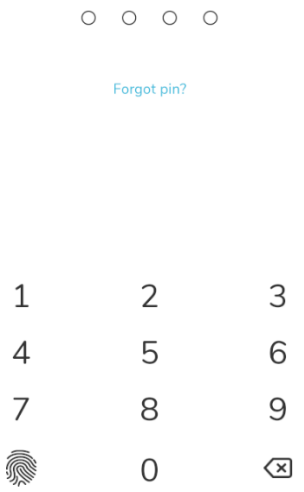

Figure 34: Autentificare prin cod PIN

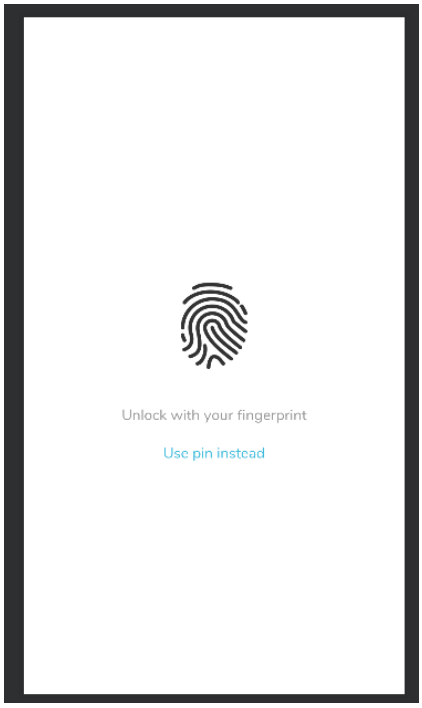

Figure 35: Autentificae prin aprentă

Pentru a asigura că dispozitivul mobil suportă senzor de amprenta se fac o serie de verificări. Se verifică dacă versiunea sistemului de operare este potrivită, dacă există senzor de amprenta, dacă avem permisiunea de a folosi amprenta utilizatorului și dacă există cel puțîn o aprentă setată pe dispozitivul respectiv.

```
fun hasFingerprintSupport(context: Context): Boolean {

    // check if droid.M
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        val keyguardManager = context.getSystemService(KEYGUARD_SERVICE) as? KeyguardManager
        val fingerprintManager = context.getSystemService(FINGERPRINT_SERVICE) as? FingerprintManager

        // check hardware
        if (!fingerprintManager!!.isHardwareDetected) {
            return false
        }

        // check permission
        if (ActivityCompat.checkSelfPermission(context, Manifest.permission.USE_FINGERPRINT)
            != PackageManager.PERMISSION_GRANTED) {
            return false
        }

        // check if at least 1 fingerprint is enrolled
        if (!fingerprintManager.hasEnrolledFingerprints()) {
            return false
        }

        if (!keyguardManager!!.isKeyguardSecure) {
            return false
        }

        return true
    }

    return false
}
```

Figure 36: Metoda care verifică dacă autentificarea prin amprenta este posibilă

În cazul în care verificarea prin amprenta nu este posibilă, utilizatorul poate folosi autentificarea prin cod PIN.

3.4.4 Gestionarea permisiunilor

Aplicația necestia o serie de permisiuni pentru a funcționa la capabilitățile sale maxime. Asta nu însemna că fără anumite permisiuni aplicație nu funcționează ci doar o să limiteze funcționalitățile disponibile. Spre exemplu dacă utilizatorul nu permite accesul la camera, el o să poată folosii restul aplicație dar nu funcționalitățile care necesită camera.

Permisiunile sunt cerute în general atunci când sunt necesare.

```
<uses-feature
    android:name="android.hardware.camera"
    android:required="false" />
<uses-feature
    android:name="android.hardware.fingerprint"
    android:required="false" />
<uses-feature
    android:name="android.hardware.camera.autofocus"
    android:required="false" />

<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.USE_BIOMETRIC" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.VIBRATE"/>
```

Figure 37: Lista de permisiuni folosite în Medicarium

- Permisiunea pentru Internet și starea rețelei sunt folosite pentru a accesa rețeau și pentru a verifică dacă există conexiune
- Permisiunea la senzorii biometrici și senzorul de amprenta sunt folosite pentru autentificare
- Camera, autofocusul și accesul la datele externe sunt folosite atunci când se adugă anlize noi
- Permisiunea pentru vibrații este folosită pentru a oferii feedback utilizatorului

3.4.5 Persistența datelor și comunicarea cu serverul

Majoritatea datelor nu sunt păstrate pe dispozitivul utilizatorului ci sunt trimise direct serverului prin canale siguri de comunicare, folosind HTTPS.

Putem exemplifica un astfel de flow pentru funcționalitatea de "adaugă analiză medicală":

1. În prima faza utilizatorul completează datele necesare și sunt validate în clientul mobil.

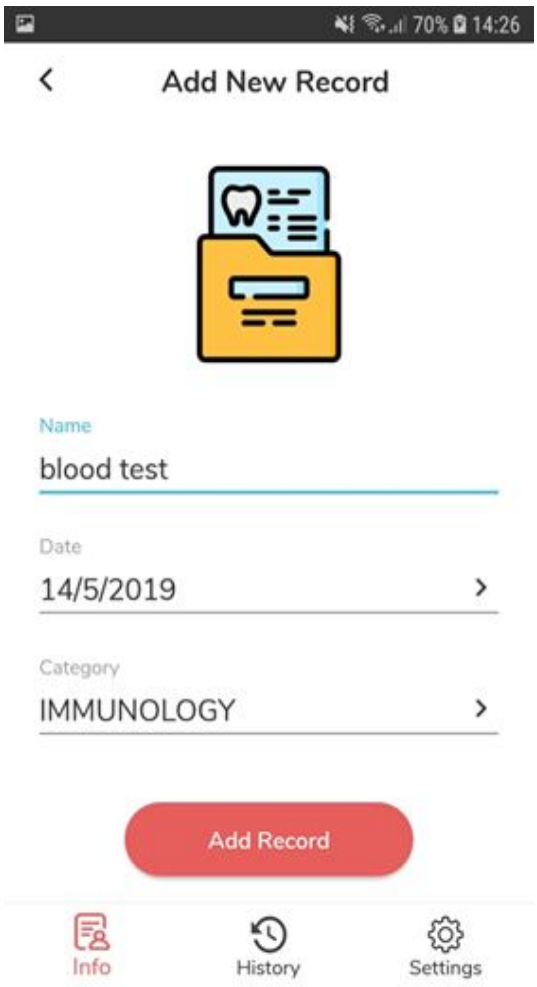

Figure 38: Ecranul unde sunt completate datele

2. O cerere HTTP POST este apoi făcută către server.

```
@POST( value: "/medicalHistory")
fun addMedicalRecord(@Body medicalRecord: MedicalRecordDataModel) : Observable<MedicalRecordDataModel>
```

Figure 39: Metodă HTTP și ruta la care se face cererea

```
fun addMedicalRecord() {
    isBusy.value = true
    medicalRecordsService
        .addMedicalRecord(medicalRecordToAdd.value!!.toMedicalRecord())
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe({ it: MedicalRecord!
            Log.i(LoggerConstants.API_REQ, msg: "AddMedicalRecord request succeeded: $it")
            _medicalRecords.add(it)
            navigateBack.value = Event( content: true)
        }, { it: Throwable!
            Log.e(LoggerConstants.API_REQ, msg: "AddMedicalRecord request failed: ${it.message}")
            isBusy.value = false
        }, {
            isBusy.value = false
        })
        .addTo(compositeDisposable)
}
```

Figure 40: Codul prin care se face cererea la server

3. În final datele ajung la server și sunt salvate în baza de date.

```
router.post("/", checkAuth, async (req, res, next) => {

    const medicalRecord = new MedicalRecord({
        _id: mongoose.Types.ObjectId(),
        userId: req.userData.userId,
        name: req.body.name,
        timestamp: req.body.timestamp,
        category: req.body.category
    })

    try {
        const saved = await medicalRecord.save()
        return res.status(200).json(saved)
    } catch(e) {
        console.log(`Error at save: ${e}`)
        return res.status(500).json({
            error: e
        })
    }
})
```

Figure 41: Cod din server

Datele care sunt salvate în memoria telefonului sunt datele generale ale utilizatorului care pot fi folosite și fără conexiune la internet pentru modul de urgență. Ele sunt păstrate într-o baza de date locală criptată. Sunt folosite și fișierele de preferință pentru a salva date de dimensiuni mici cu scop auxiliar. La fel că baza de date locală, și fișierele de preferințe sunt criptate.

```
<?xml version ="1.0" encoding ="utf-8"?>
<map>
    <boolean name="isVerified" value="true" />
    <string name="token" value="token_value" />
</map>
```

Figure 42: Fișier de preferințe fără criptare

```
<?xml version ="1.0" encoding ="utf-8"?>
<map>
    <boolean name="S2E3QmlYamlGL0JHLy9jWHZudUFmdz09"
        value="MHdEcC9Zb002cjJpVGxZMVRrNmVGdz09" />
    <string name="cFY4TnJWRnNWVUR4QWZZVEhKMlhvdz09"
        value="YXlSSWIyc2E2bm9iSTJLMGZSekVlQT09" />
</map>
```

Figure 43: Fișier de preferințe cu criptate

3.5 Testarea

Pentru a asigura calitatea soluției soft am dezvoltat în paralel cu aplicația și o suită de teste. Majoritatea testelor sunt de tip unitar, verificând o anumită funtionalitate. De asemenea testele sunt separate în funcție de entitatea pe care o testează (serverul sau clientul).

Pentru a testa serverul REST făcut în Node.js am folosit o serie de pachete ajutătoare: mocha, chai și supertest. Am creat teste pentru rutele disponibile. Testele rulează într-o configurarea separată care validează token-ul de autentificare în mod implicit.

```
describe('Unit testing for /medicalHistory DELETE', function() {
    it('should return 200', async function() {

        const rand = Math.floor(Math.random());
        const randomDoc = await MedicalRecord.findOne().skip(rand);

        const response = await request(app)
            .delete('/medicalHistory/'+randomDoc._id)
            .set('Accept', 'application/json')
            .expect("Content-type", /json/)
            .then(function(response) {
                assert.equal(response.status, 200)
            })
    })
})

describe('Unit testing for /medicalHistory PATCH', function() {
    it('should return 500, invalid data', function() {
        return request(app)
            .patch('/medicalHistory')
            .send({category: "OPHTHALMOLOGY"})
            .set('Accept', 'application/json')
            .expect("Content-type",/json/)
            .then(function(response) {
                assert.equal(response.status, 500)
            })
    })
})
```

Figure 44: Exemple de teste unitare în Node.js

Pentru a rula un test pe o anumită ruta se definește ruta la care se va execută testul, metodă (GET, POST, DELETE, etc...) și eventualele date de care e nevoie. Apoi se definește ce cod ar trebuii să aibe răspunsul primit și eventual ce date sau tip de date ar trebuii primite.

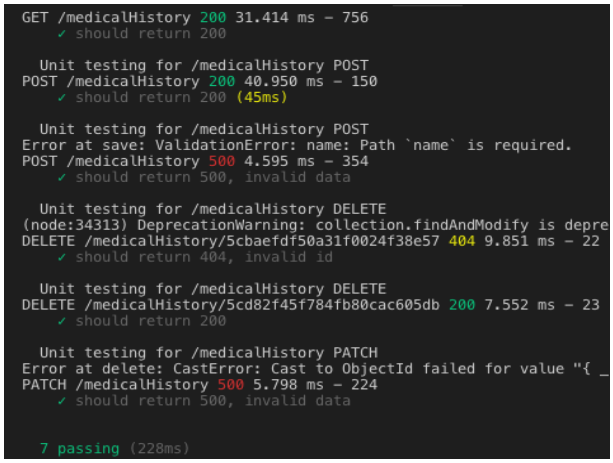

Figure 45: Rularea unei suite de teste pe ruta /medicalHistory

Pentru clientul mobil am folosit JUnit, o librărie specializate pe unit testing și diferite librării auxiliare oferite de Kotlin și Android pentru a elabora și teste de integrare.

```kotlin
class MedicalRecordsViewModelTest {
    private lateinit var viewModel: MedicalRecordsViewModel
    private lateinit var connectivityService: ConnectivityService
    private lateinit var connectivityInterceptor: ConnectivityInterceptor
    private lateinit var apiServiceFactory: ApiServiceFactory
    private lateinit var medicalRecordService: MedicalRecordService
    private var syncObject = Object()

    @get:Rule
    var instantTaskExecutorRule = InstantTaskExecutorRule()

    @Before
    fun doBeforeTest() {
        val app = InstrumentationRegistry.getTargetContext().applicationContext as Application
        connectivityService = ConnectivityServiceImpl()
        connectivityInterceptor = ConnectivityInterceptorImpl(connectivityService)
        apiServiceFactory = ApiServiceFactory(connectivityInterceptor)
        medicalRecordService = MedicalRecordServiceImpl(apiServiceFactory)
        viewModel = MedicalRecordsViewModel(app, medicalRecordService)
    }

    @Test
    fun addMedicalRecord() {
        val oldCount = viewModel.medicalRecords.value!!.count()
        Log.e( tag: "TEST",  msg: "Old count: $oldCount")

        if (connectivityService.hasConnection(InstrumentationRegistry.getTargetContext())) {
            Log.e( tag: "TEST",  msg: "Has Internet connection")
            viewModel.medicalRecordToAdd.value!!.timestamp = Calendar.getInstance().timeInMillis
            viewModel.medicalRecordToAdd.value!!.medicalCategory = MedicalCategory.OTOLOGY

            assertFalse(viewModel.isBusy.value!!)
            viewModel.addMedicalRecord()
            assertTrue(viewModel.isBusy.value!!)
            Thread.sleep( millis: 3000)
            assertTrue(viewModel.medicalRecords.value!!.count() > oldCount)
            assertFalse(viewModel.isBusy.value!!)
        } else {
            Log.e( tag: "TEST",  msg: "No Internet connection")
        }
    }
}
```

Figure 46: Clasa cu test de integrare care cuprinde mai multe servicii și un ViewModel

4 Concluzii

Acestă lucrare a avut ca scop exemplificarea în mod practic și teoretic a anumitor probleme pe care o aplicație mobilă le poate avea atunci când lucrează cu date personale. Având în vedere mișcările din lumea politică și adopotarea unor legi precum GDPR, astfel de subiecte devin din ce în ce mai relevante pentru utilizatori dar și pentru dezvoltatorii de aplicații.

Aplicația Medicarium a fost dezvoltată că și un reper. Pas cu pas, de la proiectarea aplicației până la testare, există decizii importante pe care trebuie luate pentru a asigura integritatea, confidențialitatea și disponibilitatea datelor utilizatorilor.

Aplicația poate fi extinsă și imbunatatia, ce am tratat în lucrare nu cuprinde amplul spectru a cea ce înseamnă securitate și confidențialitate. În lucrare am observat câteva din cele mai importante etape a unei aplicații. O îmbunătățire ar putea fi creare unui sistem de backup al datelor mai riguros. La momentul de față singurul backup disponibil este la nivelul bazei de date. O altă îmbunătățire ar putea fi extinderea modului offline astfel în cât să cuprindă toate funcționalitățile aplicației, nu doar modul de urgență.

În concluzie, se recomandă ca dezvoltatorii de aplicații mobile să ia în serios problematica securității și confidentialitații datelor, în caz contrar, se pot lua măsuri legale care pot afecta în mod negativ compania sau persoană respectivă.

5 Bibliografie

Bibliografie

[1] OWASP. *OWASP Top 10 2017*. URL: https://www.owasp.org/images/7/72/OWASP%5C_Top%5C_10-2017%5C_%5C%28en%5C%29.pdf.pdf.

[2] OWASP. *Top 10 Mobile 2016*. URL: https://www.owasp.org/index.php/Mobile%5C_Top%5C_10%5C_2016-Top%5C_10.

[3] *3 types of Authentication*. URL: https://www.slideshare.net/awesomeadmin/secure-your-salesforce-org-with-twofactor-authentication.

[4] *RFC 7519 JWT*. URL: https://tools.ietf.org/html/rfc7519.

[5] *JSON Web Token (JWT)*. URL: https://tools.ietf.org/html/rfc7519.

[6] ENISA. *Smartphone Secure Development Guidelines*. 2017. URL: https://www.enisa.europa.eu/publications/smartphone-secure-development-guidelines-2016.

[7] ENISA. *Guidelines for SMEs on the security of personal data processing*. 2017. URL: https://www.enisa.europa.eu/publications/guidelines-for-smes-on-the-security-of-personal-data-processing.

[8] Bin Liu et al. "Follow my recommendations: A personalized privacy assistant for mobile app permissions". In: *Twelfth Symposium on Usable Privacy and Security ({SOUPS} 2016)*. 2016, pp. 27–41.

[9] davx5. *Why does WiFi SSID restriction need the location permission?* URL: https://www.davx5.com/faq/wifi-ssid-restriction-location-permission.

[10] Giles Hogben Martin Pelikan and Ulfar Erlingsson. *Identifying Intrusive Mobile Apps Using Peer Group Analysis*. URL: https://security.googleblog.com/2017/07/identifying-intrusive-mobile-apps-using.html.

[11] Reddit Post. *Since when did a calculator need to make calls?* URL: https://www.reddit.com/r/assholedesign/comments/8nlh3k/since_when_did_a_calculator_need_to_make_calls/.

[12] Arthur Goldberg, Robert Buff, and Andrew Schmitt. "A comparison of HTTP and HTTPS performance". In: *Computer Measurement Group, CMG98* 8 (1998).

[13] Adrienne Porter Felt et al. "Measuring {HTTPS} Adoption on the Web". In: *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 2017, pp. 1323–1338.

[14] "The rise of 3G". In: URL: `http://www.itu.int/ITU-D/ict/material/FactsFigures2010.pdf`.

[15] Ian Fette and Alexey Melnikov. *The websocket protocol*. Tech. rep. 2011.

[16] Jussi-Pekka Erkkilä. "Websocket security analysis". In: *Aalto University School of Science* (2012), pp. 2–3.

[17] *Testing WebSockets*. URL: `https://www.owasp.org/index.php/Testing_WebSockets_(OTG-CLIENT-010)`.

[18] Jonathan Katz et al. *Handbook of applied cryptography*. CRC press, 1996.

[19] Tutorialspoint. *Public Key Encryption*. URL: `https://www.tutorialspoint.com/cryptography/public_key_encryption.htm`.

[20] Amy J Barton. "The regulation of mobile health applications". In: *BMC medicine* 10.1 (2012), p. 46.

[21] James G Kahn, Joshua S Yang, and James S Kahn. "'Mobile'health needs and opportunities in developing countries". In: *Health Affairs* 29.2 (2010), pp. 252–258.

[22] Wikipedia. *Model-view-viewmodel*. URL: `https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel`.

[23] Mark Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces.* " O'Reilly Media, Inc.", 2011.

[24] Stefan Tilkov and Steve Vinoski. "Node. js: Using JavaScript to build high-performance network programs". In: *IEEE Internet Computing* 14.6 (2010), pp. 80–83.

[25] Mihai Cracan. *Web REST API Benchmark on a Real Life Application*. URL: `https://medium.com/@mihaigeorge.c/web-rest-api-benchmark-on-a-real-life-application-ebb743a5d7a3`.

[26] Kyle Banker. *MongoDB in action*. Manning Publications Co., 2011.

[27] Ameya Nayak, Anil Poriya, and Dikshay Poojary. "Type of NOSQL databases and its comparison with relational databases". In: *International Journal of Applied Information Systems* 5.4 (2013), pp. 16–19.

[28] ENISA. *Privacy and data protection in mobile applications*. 2019. URL: https://www.enisa.europa.eu/publications/privacy-and-data-protection-in-mobile-applications.

[29] Asfak Saiyed. *What is Android Jetpack?* 2018. URL: https://android.jlelse.eu/what-is-android-jetpack-737095e88161.

[30] StatCounter. *StatCounter*. 2019. URL: http://gs.statcounter.com/os-market-share/mobile/worldwide.

[31] Marcin Moskala and Igor Wojda. *Android Development with Kotlin*. Packt Publishing Ltd, 2017.

[32] Pieter Otten. *Kotlin vs Java*. URL: https://www.mediaan.com/kotlin-vs-java/.

[33] Jakob Jonsson and Burt Kaliski. *Public-key cryptography standards (PKCS)# 1: RSA cryptography specifications version 2.1*. Tech. rep. 2003.