UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENŢĂ

**Secure Bit Exchange**

**propusă de**

***Ioana Alexandra Tănase***

**Sesiunea:** *Iulie, 2016*

**Coordonator ştiinţific**

**Conferențiar Dr. Ștefan Ciobâcă**

**UNIVERSITATEA ALEXANDRU IOAN CUZA IAŞI**

**FACULTATEA DE INFORMATICĂ**

**Secure Bit Exchange**

***Ioana Alexandra Tănase***

**Sesiunea:** *Iulie, 2016*

**Coordonator ştiinţific**

**Conferențiar Dr. Ștefan Ciobâcă**

DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licenţă cu titlul „*Secure Bit Exchange*” este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învăţământ superior din ţară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

* toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi deţin referinţa precisă a sursei;
* reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa precisă;
* codul sursă, imaginile etc. preluate din proiecte open-source sau alte surse sunt utilizate cu respectarea drepturilor de autor şi deţin referinţe precise;
* rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi, *data*

Absolvent *Ioana Alexandra Tănase*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Secure Bit Exchange”*, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” Iași să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iași, *data*

Absolvent *Ioana Alexandra Tănase*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

# CUPRINS

[INTRODUCERE 5](#_Toc455043808)

[CONTRIBUȚII 7](#_Toc455043809)

[I. Tehnologii folosite 8](#_Toc455043810)

[A. Android Studio IDE 8](#_Toc455043811)

[B. Limbajul de programare Java 8](#_Toc455043812)

[1. Prezentare 8](#_Toc455043813)

[C. Concluzii 9](#_Toc455043814)

[II. Descrierea Aplicației 10](#_Toc455043815)

[A. Analiză și proiectare 10](#_Toc455043816)

[1. Soluționarea problemei 10](#_Toc455043817)

[2. Expunerea ideii de rezolvare 10](#_Toc455043818)

[B. Arhitectură 12](#_Toc455043819)

[C. Implementare 15](#_Toc455043820)

[D. Tutorial 20](#_Toc455043821)

[CONCLUZII 24](#_Toc455043822)

[BIBLIOGRAFIE 25](#_Toc455043823)

# INTRODUCERE

Ultimii zece ani au reprezentat un mediu prosper pentru industria telefoanelor mobile ce rulează sub sistemul de operare Android. Încă de la lansarea platformei, noiembrie 2007, aspectul open source al acesteia a stârnit o mulțime de contradicții, dar s-a dovedit a fi un mare plus al acestui sistem.

Atât portabilitatea, cât și mâna liberă a utilizatorului spre content puternic customizabil, reprezintă aspecte unice ale sistemului de operare Android. Datorită acestor facilități împreună cu permisibilitatea Magazinului Play de a accepta promovarea aplicațiilor la un preț redus, appstore-ul a fost asaltat cu milioane de programe soft.

La scurt timp au început să apară diferite aplicații menite să ușureze munca de zi cu zi a oamenilor. Pe lângă altele, au apărut diferite instrumente soft care să ajute procesul de a lua o decizie prin intermediul aruncării unei monede. Popularitatea lor a început să crească în ultimul timp, dar toate dispun de un nivel scăzut de securitate și nu permit evaluarea deciziei a două persoane. Câteva exemple de astfel de aplicații: Coin Flip[[1]](#footnote-1), Coin Flip 3D[[2]](#footnote-2).

Am decis așadar, în urma motivelor expuse anterior, crearea unui software dezvoltat pe platforma Android care să combine interacțiunea oamenilor cu interesul pentru tehnologie prezentat de aceștia.

Obiectivul principal al acestei lucrări este de a oferi un mediu bazat pe încredere, ferit de orice tentativă de a trișa, necesar unei bune înțelegeri între două părți atunci când survine problema luării unei decizii. Propun în acest scop o aplicație care să satisfacă acest țel.

Proiectul de față este realizat cu ajutorul uneia dintre cele mai puternice instrumente de dezvoltare a programelor software, Java, rulat de Android Studio IDE v2.1.2.

Lucrarea este structurată pe trei părți: Prezentarea sistemului de operare Android, Tehnologii utilizate și Descrierea aplicației.

Prima parte este formată din două capitole în care se descrie pe scurt sistemul de operare Android și avantajele acestuia.

Cea de-a doua parte surprinde o expunere concisă a tehnologiilor utilizate în implementarea aplicației.

În cea de-a treia parte este prezentată aplicația din mai multe puncte de vedere. Primul capitol face analiza generală a funcționalităților descriind modul în care interacționează serviciul Network Service Discovery cu protocolul de comunicare, sub schema Server - Client.

Al doilea capitol reprezintă aspecte legate de modalitatea de implementare a soluției.

Cel de-al treilea și ultim capitol face o prezentare amănunțită a programului din două perspective: sever și client.

# CONTRIBUȚII

Ideea construirii unei aplicații Android a fost propusă de mine. Alegerea tematicii a fost realizată în colaborare cu domnul profesor Ștefan Ciobâcă, acesta elaborând varianta finală.

Elementele de teorie folosite în construirea proiectului survin în proporție majoritară din partea profesorului meu coordonator, pe când soluțiile practice folosite în dezvoltarea lucrării provin în principal din propria experiență profesională, dobândită programând pe platforma Android, completată de noțiuni de Programare Orientată Obiect și Programare Avansată Java, acumulate pe parcursul anilor din facultate.

Sub îndrumare constantă am implementat un serviciu ce poate fi descoperit pe infrastructura Wi-Fi, ușurând astfel comunicarea între Server și Client, bazată pe protocolul de securitate de BitAgreement.

Colaborarea coordonator – student s-a produs sub forma de întâlniri săptămânale împreună cu numeroase convorbiri purtate cu ajutorul e-mailul, comunicarea realizându-se astfel într-un mod transparent și eficient.

Toți acești factori au contribuit la o bună și prosperă dezvoltare a prezentei lucrări și a aplicației practice.

# Tehnologii folosite

## Android Studio IDE

Mediu de dezvoltare[[3]](#footnote-3) oficial al programării sub platforma cu același nume, Android Studio asigură cele mai rapide instrumente necesare construirii de aplicații pentru Android.

Iată câteva dintre funcționalitățile oferite:

* IntelliJ IDE + Android Studio plugin (editor de cod intelligent)
* Unelte de debugging
* Sistem instant build/deploy
* Emulator rapid cu sitem de operare Android ce include Google Play Services
* Template-uri de cod
* Integrare GitHub
* O versiune a platformei Android

Toate acestea sunt oferite la un nivel de performanță ridicat și având drept suport flexibilitatea proiectelor.

## Limbajul de programare Java

Java reprezintă un limbaj de programare de nivel înalt, apărut mai ales ca urmare a eforturilor firmei JavaSoft, ulterior SUN Microsystems. Aceasta a încercat să dezvolte un limbaj de programare pentru echipamentele inteligente (telefonul, cuptor cu microunde, carduri, detectoare de fum şi a altor dispozitive mici) în cadrul așa numitului proiect Green. Mai mult decât atât, datorită provocării lansate de Internet, era nevoie de un limbaj care să permită utilizarea aplicaţiilor pe o mulţime de platforme.

Limbajul a apărut in August 1991 și îl are drept autor pe James Gosling. Autorul, Patrick Naughton şi Bill Joy sunt membrii echipei pentru implementarea proiectului Green, proiect care se ocupa de programarea dispozitivelor inteligente. Cei trei au pus bazele interpretorului Java, scris în C.

Tot ce a urmat după aceea a fost dezvoltarea ideii pentru ca noi astăzi să avem parte de un nou limbaj, superior din multe puncte de vedere limbajului C++. Astăzi, toate instrumentele necesare dezvoltării și rulării programelor Java sunt incluse în ceea ce se numește Java Development Kit, pe scurt JDK.

## Concluzii

Tehnologiile folosite în dezvoltarea aplicației „Secure Bit Exchange” sunt Android Studio IDE v.2.1.2 și Java SE Development 8[[4]](#footnote-4). Întreaga lucrare este scrisă sub API level echivalent versiunii de Android Jelly Bean, ce manifestă funcționalitate pe aproximativ 80% din dispozitive.

Am optat pentru acest mediu de dezvoltare deoarece, pe lângă faptul ca este cel oficial în materie de programare pe platforma Android, el oferă o mulțime de avantaje față de plug-in-ul ADT pentru Eclipse[[5]](#footnote-5).

Alegerea ultimei versiunii de Java, SDK 8, este evidentă. Acest update a adus atât inovație cât și stabilitate pentru dezvoltatori.

# Descrierea Aplicației

## Analiză și proiectare

### Soluționarea problemei

Competiția este resimțită tot mai puternic în zilele noastre. Odată cu spiritele aprinse și doritoare de câștig se pot strecura, din păcate, și participanți neserioși (trișori) la numeroasele evenimente provocatoare din viața noastră.

Date aceste fapte, menținerea aspectului cu adevărat aleratoriu până și la aruncarea unei monede poate ridica dificultăți. Se pot găsi infinite modalități pentru ca un individ să își asume câștigul pe nedrept.

Implicând tehnologia introducem un nou tip de trișor în peisaj – supranumit uneori hacker. În realitate acesta poate fi pur și simplu un exploator ce doar a avut puțin noroc și nu chiar calitățile necesare spargerii de coduri sau reguli. Pentru a crea o aplicație care să permită un schimb de informații sigur, am analizat o multitudine de protocoale de securitate. Acestea funcționează pe diferite principii, dar în principal pe schimb de noțiuni criptate.

Pentru ca acest protocol să poate fi pus în aplicare a fost solicitată o structură de server – client a aplicației. Cum conexiunea prin socketuri de una singură poate provoca dificultăți a fost necesară implicarea unui API pentru a mijloci și stabiliza fluxul de informație.

În concluzie, am survenit cu o idee de rezolvare pentru problema prezentată urmând a fi prezentată mai în detaliu în capitolul care urmează.

### Expunerea ideii de rezolvare

Înainte de a specifica o posibilă soluție a problemei am decis că ar fi util să putem fixa cerințele aplicației „Secure Bit Exchange”. Prin urmare, regăsim drept funcționalități de bază următoarele:

1. Detectarea altor dispozitive ce rulează aplicația drept server în vederea stabilirii unei conexiuni sau crearea unui astfel de serviciu în cazul în care descoperirea nu returnează nicio legătură posibilă;
2. Determinarea cărei părți părtașă la conexiune îi revine dreptul de start al jocului;
3. Menținerea evidenței informațiilor transferate, ambele părți trebuind să aprobe finalizarea schimbului.

Găsirea dispozitivului ce își însușește meseria de server se realizează cu ajutorul API-ul Android, Network Service Discovery. Folosirea API-ului în aplicație permite utilizatorilor să identifice alte dispozitive la nivel local care suportă serviciul cerut de soft-ul în cauză. În caz contrar, cu ajutorul aceleiași tehnologii se poate transforma aplicația curentă într-un dispozitiv ce înregistrează tipul de serviciu dorit, urmând apoi să aștepte realizarea unei conexiuni.

Entitatea cu drept de start la joc este determinată folosind un protocol de commitment bazat pe criptografie. Fiecare participant alege un bit random 0 sau 1. Folosind o metodă de criptare cu cheie simetrică, amândoi recurg la codificarea acestuia. Clientul trimite bitul codat serverului și vice-versa. Când acest proces a luat sfârșit și ambii au luat la cunoștință veridicitatea informațiilor de până acum, își pot da acordul pentru a finaliza protocolul.

În urma sincronului de acorduri, fac schimb și de cheile lor folosite la criptare și decodează bitul oponentului. Alegerea participantului cu drept de start se alege, așadar, prin evaluarea expresiei XOR dintre cei doi biți. 0 indică câștigul serverului, iar 1 pe cel al clientului. Țin să menționez ca prin această metodă este exclusă orice tentativă de trișare a celor doi candidați.

## Arhitectură

Aplicația a fost gândită având drept suport modelul Server – Client. Uneori astfel de conexiuni pot ridica diverse provocări de stabilitate. Drept soluție am ales folosirea unui API[[6]](#footnote-6) care pune la dispoziția dezvoltatorilor posibilitatea de a crea servicii proprii cu scopul de a conecta dispozitive prin Wi-Fi.

Network Service Discovery (prescurtat NSD în ceea ce urmează) garantează:

1. ușurință de integrare;
2. securitate a datelor;
3. stabilitate în formarea de conexiuni.

La baza folosirii acestui instrument stau doar trei listeneri care utilizați cu succes constituie 90% din funcționalitate. Putem considera astfel ușurința cu care putem dezvolta aplicații prin mijlocul NSD.

Se pot realiza conexiuni numai între dispozitive ce rulează același tip de serviciu și sub același pseudonim. Aceste două informații nu sunt publice pentru utilizatori și pot fi ridica dificultate în a fi descoperite de către aplicații autorizate. Așa putem conchide că sistemul este ridicat peste un spațiu securizat.

Odată pus în practică, API-ul furnizează toate cele necesare formării unei conexiuni prin intermediul socketelor; timpul de execuție restrâns și o legătură asigurată contribuind la stabilitatea sitemului.

Acest serviciu de descoperire constă în trei pași: înregistrare, descoperire și soluționare.

Primul pas îl reprezintă înregistrarea serviciului propriu asupra rețelei de Internet Wireless. Din acest moment alte dispozitive vor putea găsi broadcastul acestui instrument.

În cazul în care vrem să descoperim serivicii deja înregistrate, ne folosim de listenerul de discovery al API-ului. La găsirea unor servicii se poate înainta la pasul de resolve. Dacă ambele părți, atât serviciul cât și clientul rulează sub același tip de conexiune NSD, soluționarea legăturii peste rețea va fi una pozitivă.

După ce pasul de soluționare a avut loc, se pot folosi informațiile prelucrate până în acest moment de API-ul nostru (host și port) în vederea stabilirii unei conexiuni bazată pe comunicarea între sockete.

Ulterior serviciului, am folosit un server TPC/IP iterativ pentru a comunica cu clienții. Motivul pentru care am optat pentru un astfel de model de server este natura aplicație; ea va putea înregistra o singură conexiune Server – Client la un anumit moment.

Odată activ, serverul așteaptă sosirea cererile clienților. În momentul în care legătura dintre părțile participante se stabilește, clientul poate iniția startul protocolului de commitment pentru a soluționa dispozitivul ce are dreptul de a fi primul la joc.

Protocolul constă în schimb de informații codate între părtași. Fiecare dintre aceștia vor accesa printr-un task asincron[[7]](#footnote-7) serviciul web Random.org[[8]](#footnote-8) și vor genera astfel câte un bit random (0 sau 1). Acesta va fi codat folosind un algoritm de criptare cu cheie simetrică și trimis părții adverse. Odată primiți cu succes, clientul și serverul fac schimb și de cheile lor de criptare. Cu ajutorul acestora reușesc să decodifice bitul trimis inițial.

Câștigătorul, sau norocosul ce va avea drept să fie primul, va fi ales în funcție de rezultatul generat de evaluarea expresiei logice XOR[[9]](#footnote-9). În cazul în care valoarea disjuncției este egală cu 0 serverul va fi desemnat primul jucător, altfel, clientul va avea drept de start în caz contrar (expresia va indica valoarea 1).

Pentru a înțelege mai ușor modul de funcționare al acestui protocol am recurs la desenarea unui diagrame UML care descrie întregul schimb de informații.

În figura 2 se poate vedea modul în care comunică serverul cu clientul în scopul implementării protocolului de commitment.

Cu ajutorul acestui protocol se elimină una dintre problemele ce pot surveni în timpul alegerii primului jucător în cadrul oricărei competiții. Totul se bazează pe noroc pur și nu există mijloc de a trișa sau de a prezice rezultatul.



Figura 1: Diagramă a protocolului de bit agreement

## Implementare

În acest capitol voi descrie în detaliu procesul tehnic care a dus la asamblarea aplicației.

În Figura 2 se regăsește structura proiectului Secure Bit Exchange.

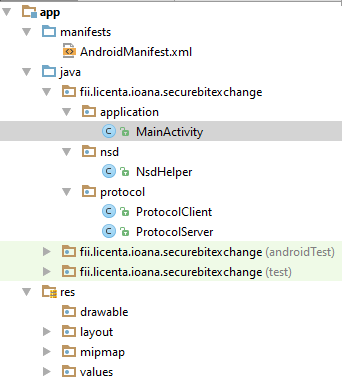


Figura 2: Organizarea codului sursă

Un proiect de tip Android este format din trei foldere de bază:

* **Manifests** – conține fișierul de configurare AndroidManifest.xml, informații esențiale ce trebuie cunoscute înainte de rularea aplicației;
* **Java** – include partea de cod sursă;
* **Res** – destinat părții de interfață grafică, cuprinde structurile activităților, stiluri de scris și de culori, imagini și perechi cheie – valoare folosite în program.

Codul sursă al aplicației a fost împărțite în trei pachete:

* **App** – conține clasa activității de bază, MainActivity
* **Nsd** – include clasa corespunzătoare API-ului NSD, NsdHelper
* **Protocol** – include serverul și clientul TCP / IP

După cum putem observa, arhitectura aplicației este axată pe principiul simplității. Cum întreaga funcționalitate se învârte în jurul comunicării client – server, clasele din pachetul de protocol sunt singurele mai stufoase, restul având o complexitate scăzută.

O scurtă prezentare a claselor, după cum urmează:

* *MainActivity* este clasa de bază a aplicației, aici se află funcțiile ce denotă comportamentul softului pe toată durata ciclului de viață (create, start, stop, pause, destroy) a programului. Are ca scop invocarea tuturor serviciilor folosite, controlul modului de rulare al aplicației, client sau server, și inițierea comunicării între dispozitive.
* *NsdHelper* reprezintă implementarea funcționalității API-ului de Network Service Discovery. Cuprinde trei obiecte de tip listener pentru fiecare pas necesar conectării sub acest API: registerListener, discoverListener și resolveListener , organizate de un obiect de tip NsdManager;
* *ProtocolServer* clasa ce cuprinde modul de funcțiune al serverului TCP/IP, crearea portului la care se așteaptă clienții, criptarea bitului random, transferul de date;
* *ProtocolClient* clasă similară celei anterioare, se ocupă cu modul în care se comportă partea de client a conexiunii. Clientul trimite cererea către server, dacă s-a efectuat cu succes conexiunea, se începe schimbul de informații.

Pentru o mai bună înțelegere a modului în care rulează programul voi expune principalele metode folosite în cadrul soluției, în ordinea cronologică apelării lor.

Execuția aplicației începe cu invocarea metodei onCreate aparținând acțivității, cod ce se execută înainte de startul efectiv al softului.

|  |
| --- |
| @Override **protected void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.***activity\_nsd***);  **textView** = (TextView) findViewById(R.id.***textView1***);  **button** = (Button) findViewById(R.id.***button1***);  **button**.setVisibility(View.***INVISIBLE***); } |

Figura 3: Funcția onCreate() din ActivityMain

Rolul principal al funcției onCreate, este stabilirea contextului de lucru și inițializarea componentelor corespunzătoare view-ului.

Ulterior se execută metodă onStart, cu scop principal de a distinge comportamentul aplicației: tip server sau tip client.

Pentru partea de server se instanțiază un obiect de tipul NsdHelper cu ajutorul căruia se pornește broadcastul serviciului nostru peste rețeaua Wi-Fi la care suntem conectați în acel moment.

După trecerea a trei secunde se inițiază partea de server a conexiunii TCP și se așteaptă cereri de la clienți. De remarcat este faptul că procesul de așteptare nu începe până nu se apasă pe butonul Start al aplicației. Acest buton devine vizibil atunci când serviciul a fost distribuit cu succes.

|  |
| --- |
| @Override **protected void** onStart() {  **super**.onStart();  **mNsdHelper** = **new** NsdHelper(**this**);  **mNsdHelper**.initializeNsd();  Handler handler = **new** Handler(Looper.*getMainLooper*());  **mNsdHelper**.initializeServerSocket();  handler.postDelayed(**new** Runnable() {  @Override  **public void** run() {  Log.i(***TAG*** + **" S-a inregistrat? "** + **mNsdHelper**.**registered**);  Log.i(***TAG*** + **" Terminat."**);  **textView**.setText(**"Server terminal."**);  initServerButton();  **button**.setVisibility(View.***VISIBLE***);  }  }, 3000); } |

Figura 4: Funcția onStart() din ActivityMain pe partea de server

În partea de client se pornește prin descoperirea serviciului, se continuă cu conectarea la acesta după un timp de așteptare de trei secunde, și se încheie prin formarea legăturii între cele două dispozitive. După ce host-ul si port-ul au fost asigurate, se încheie procesul de discovery și se afișează pe ecran un buton ce servește la pornirea clientului.

|  |
| --- |
| @Override **protected void** onStart() {  **super**.onStart();  **mNsdHelper** = **new** NsdHelper(**this**);  **mNsdHelper**.initializeNsd();  Handler handler = **new** Handler(Looper.*getMainLooper*());  **mNsdHelper**.discoverServices();  handler.postDelayed(**new** Runnable() {  @Override  **public void** run() {  NsdServiceInfo service = **mNsdHelper**.getChosenServiceInfo();  }  }, 3000);  handler.postDelayed(**new** Runnable() {  @Override  **public void** run() {  NsdServiceInfo service = **mNsdHelper**.getChosenServiceInfo();  **if** (service != **null** && **mNsdHelper**.**resolved**) {  **port** = service.getPort();  **host** = service.getHost();  System.***out***.println(**"Conectat la "** + String.*valueOf*(**host**) + **" "** + **port**);  }  **mNsdHelper**.stopDiscovery();  **textView**.setText(**"Client terminal."**);  initClientButton();  **button**.setVisibility(View.***VISIBLE***);  }  }, 3000); } |

Figura 5: Funcția onStart() din ActivityMain pe partea de client

Putem observa că instanțierea serverului și a clientului se face cu ajutorul funcțiilor initServerButton (fig. 7) și initClientButton (fig. 8). Aceste două metode cuprind câte un onClickListener pentru butonul de start al componentei corespunzătoare fiecărui tip de aplicație. Atât serverul cât și clientul sunt inițializați de câte un thread separat celui principal (ActivityMain), lucru ce sporește eficiența lucrării și nu blochează userul de la a accesa partea de view în timpul execuției protocolului.

|  |
| --- |
| **public void** initServerButton() {  **button**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  **button**.setVisibility(View.***INVISIBLE***);  **final** ProtocolServer protocolServer = **new** ProtocolServer(**mNsdHelper**.**mServerSocket**);  Thread server = **new** Thread(protocolServer);  server.start();  Handler handler = **new** Handler(getMainLooper());  handler.postDelayed(**new** Runnable() {  @Override  **public void** run() {  **if**(protocolServer.*ended*){  **byte** xor = protocolServer.*xor*;  exposeWinner(xor);  }  }  }, 3000);  }  }); } |

Figura 6: Funcția initServerButton() din ActivityMain

|  |
| --- |
| **public void** initClientButton() {  **button**.setOnClickListener(**new** View.OnClickListener() {  @Override  **public void** onClick(View v) {  **final** ProtocolClient protocolClient = **new** ProtocolClient(**host**, **port**);  Thread client = **new** Thread(protocolClient);  client.start();  Handler handler = **new** Handler(getMainLooper());  handler.postDelayed(**new** Runnable() {  @Override  **public void** run() {  **if**(protocolClient.**ended**){  **byte** xor = protocolClient.**xor**;  exposeWinner(xor);  }  }  }, 3000);  }  }); } |

Figura 7: Funcția initClientButton() din ActivityMain

Execuția aplicației se încheie atunci când ambii utilizatori și-au datul acordul de terminare a schimbului de informații și rezultatul se va afișa pe ecran.

Activitatea poate fi încheiată și forțat de către utilizator, caz în care este apelată funcția onDestroy(), moment în care se distruge API-ul NSD și se închide terminalul TCP, după caz.

|  |
| --- |
| @Override **protected void** onDestroy() {  mNsdHelper.tearDown();  **super**.onDestroy(); } |

Figura 8: Funcția onDestroy() din ActivityMain

|  |
| --- |
| **public void** eardown() {  mNsdManager.unregisterService(mRegistrationListener); //pentru server  mNsdManager.stopServiceDiscovery(mDiscoveryListener); //pentru client } |

Figura 9: Funcția tearDown() din NsdHelper

## Tutorial

La rularea aplicației se va decide ce tip de terminal se instanțiază. Serviciul denumit BitExchange este deja existent se va porni terminalul de client, în cazul în care nu se poate găsi instrumentul dorit se rulează aplicația sub modelul serverului.

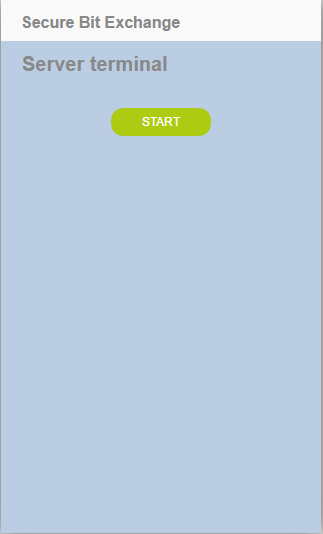
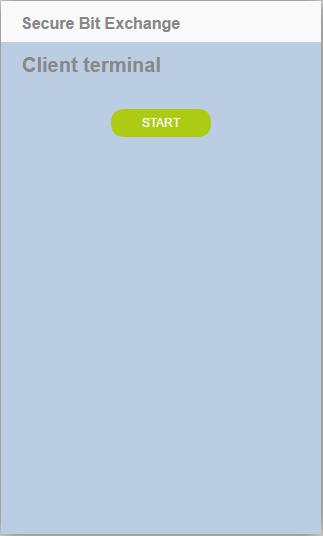
 

Figura 10: Pornirea aplicației

Pe partea de server se inițializează serviciul BitExchange și se începe broadcastul lui asupra rețelei Wi-Fi.

Pe partea de client este început procedeul de descoperire al serviciului și se pregătește de inițierea conexiunii.

Când acești pași au avut loc, iar soluționarea serviciului a fost una evaluată cu succes, butonul ‘Start’ devine vizibil. Odată apăsat de pe terminalul server, acesta creează serverSocketul și așteaptă conexiunea din partea clientului. Clientul așteaptă și urmează să apese el însuși butonul pentru a realiza legătura între dispozitive.

Când conexiunea a fost una reușită, pe ecran va apărea o secțiune albă ce va surveni ca scop afișajul anumitor informații necesare la buna desfășurare a protocolului.

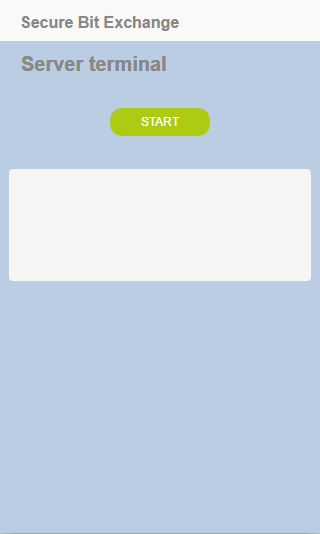
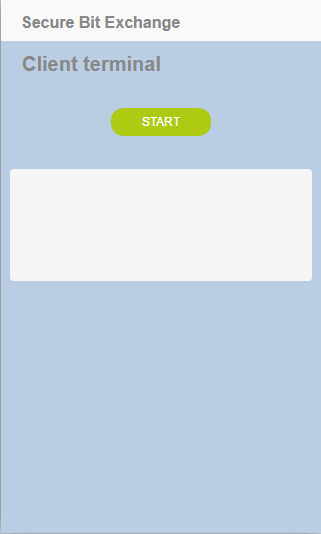
** **

Figura 11: Serviciul setat corect și inițierea conexiunii TCP/IP

Dacă legătura între sockete a fost realizată cu succes, în partea destinată afișării text vor începe să apară rezultatele.

Primul pas îl reprezintă alegerea a unui bit aleator de către fiecare utilizator, posibilă cu ajutorul API-ului Random.org. Prima linie a ariei de text este rezervată afișării acestui bit.

Se generează o cheie unică de criptare pentru fiecare participant.

În continuare, algoritmul de criptare folosit codează informația și arată rezultatul ei pe ecran.

Se face schimbul de biți codați și se așteaptă confirmarea utilizatorului.

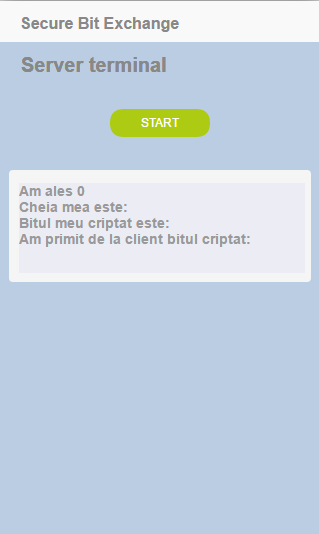
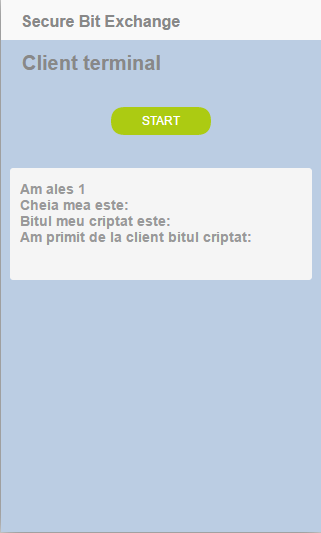
** **

Figura 12: Se realizează schimbul de biți codați

Următorul pas îl reprezintă luarea la cunoștință de ambele părți faptul că schimbul de informații a decurs cu succes și că sunt de acord să trimită reciproc cheile lor secrete necesare în procesul de decriptare.

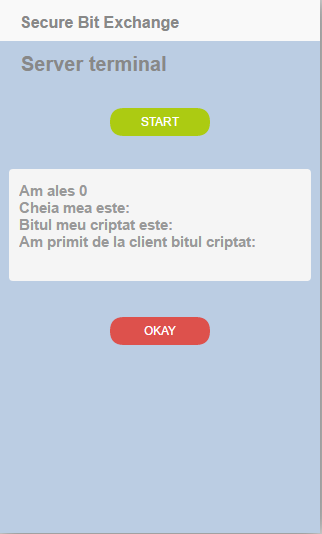
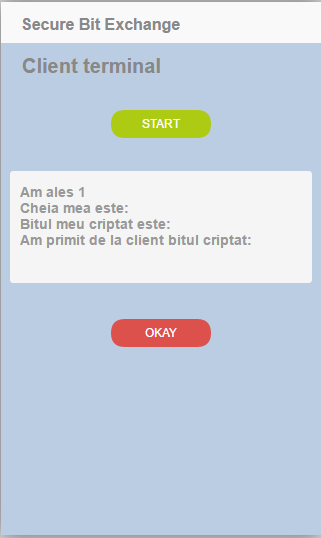
** **

Figura 13: Se așteaptă confirmarea utilizatorului

În final, după terminarea transmisiei și închiderea conexiunilor fiecare participant procesează rezultatul expresiei și exclusiv între biții aleși. Aceste informații sunt și ele precizate pe ecran, în josul display-ului. Cazul în care s-a ajuns la valoarea 0, serverul este favorabil, clientul având drept de start atunci când rezultatul este 1.

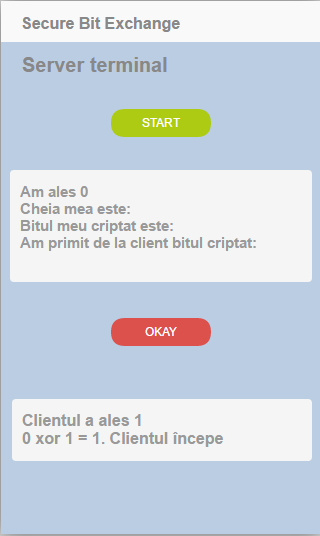
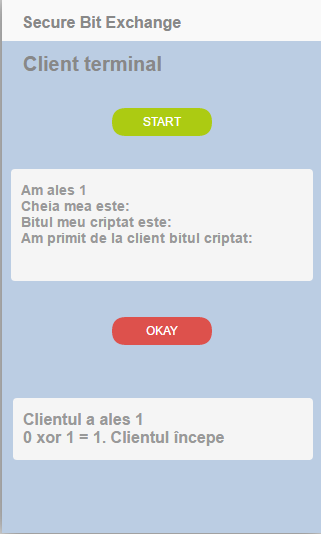
** **

Figura 14: Sfârșitul protocolului

# CONCLUZII

Obiectivul principal al acestei lucrări a fost căutarea unui mijloc de a angaja oamenii la a se pune de acord în privința luării unei decizii. În acest scop au fost necesare soluții anti-trișare și anti-exploatare.

Consider că aplicația a reușit să atingă toate funcționalitățile propuse, implementarea lor ridicând uneori dificultăți.

Un mare hop ce a trebuit depășit încă de la primele linii de cod a fost stabilizarea API-ului NSD și realizarea unei conexiuni stabile. Odată ce aceste două funcții au fost stăpânite totul a decurs mult mai fluent. Am văzut cum implementarea unei criptări relativ simplă poate aduce atât de multe beneficii și poate asigura un spațiu sigur peste care se pot construi o multitudine de alte funcționalități.

O posibilă direcție de dezvoltare este crearea unui sistem de menținere a scorului pentru jocuri de cărți și a unei baze de date care să stocheze informații despre meciurile trecute, incluzând totodată un sistem de logare în aplicație; se poate extinde și refolosi criptarea în securizarea procesului de login.

Bineînțeles o altă idee de îmbunătățire este aceea de a modulariza aplicația încât să poată extinde pentru mai multe tipuri de jocuri, nu numai de cărți. Implementarea unui zar poate fi folositoare în acest scop.

Pentru a fi accesibilă și mai multor persoane care nu cunosc bine limba suportată, s-ar putea include mai multe limbi în componența aplicației cum ar fi Franceză, Italiană, Germană, etc.

Conchidem astfel că tema propusă este una de actualitate, în ton cu tendințele curente și ce lasă loc de oportunități și prosperitate.

# BIBLIOGRAFIE

**[1] Wikipedia:** https://en.wikipedia.org/wiki/Commitment\_scheme

**[2] Android Developer**: https://developer.android.com/index.html

**[3] Tutorialspoin:** http://www.tutorialspoint.com/java/java\_networking.htm

**[4] Random engine:** https://www.random.org/

1. Coin Flip App: <https://play.google.com/store/apps/details?id=com.ajas.coinflip> [↑](#footnote-ref-1)
2. Coin Flip 3D App: <https://play.google.com/store/apps/details?id=com.fithx.coinflip> [↑](#footnote-ref-2)
3. Integrated Development Environment guide: <http://mashable.com/2010/10/06/ide-guide> [↑](#footnote-ref-3)
4. Java SE Development Kit Guide: <https://docs.oracle.com/javase/8/> [↑](#footnote-ref-4)
5. Eclipse plugin ADT: <https://developer.android.com/studio/tools/sdk/eclipse-adt.html> [↑](#footnote-ref-5)
6. Application Programming Interface: <https://ro.wikipedia.org/wiki/Application_Programming_Interface> [↑](#footnote-ref-6)
7. Documentație AsyncTask: <https://developer.android.com/reference/android/os/AsyncTask.html> [↑](#footnote-ref-7)
8. Serviciul Random.org bazat pe sunetele din atmosferă: <https://www.random.org/> [↑](#footnote-ref-8)
9. XOR (Disjuncție exclusivă): <https://ro.wikipedia.org/wiki/Disjunc%C8%9Bie_exclusiv%C4%83> [↑](#footnote-ref-9)