

פרויקט גמר בסייבר

Rdroid

העברת מידע בין מכשירים ניידים ואתר אינטרנט

**שם התלמיד: יניב שרון**

**ת"ז: 206842858**

**מנחים: מיכאל צ'רנובילסקי וגד רוזנטל**

**בית הספר: אורט גוטמן נתניה**

**כיתה: י"ב 8**

**מאי 2016**

**תוכן עניינים**

[תקציר כולל ורציונל הפרויקט 3](#_Toc451518445)

[מבוא ורקע כללי 4](#_Toc451518446)

[תיאור המוצר המוגמר 5](#_Toc451518447)

[שפת התכנות וסביבת העבודה 6](#_Toc451518448)

[ניסוח, ניתוח ופתרון הבעיה האלגוריתמית 8](#_Toc451518450)

[תיאור מבנה המוצר 18](#_Toc451518451)

[מדריך התקנה למשתמש 27](#_Toc451518452)

[תיאור הממשק למשתמש 28](#_Toc451518453)

[מבט אישי על העבודה ותהליך הפיתוח 30](#_Toc451518454)

[ביבליוגרפיה 31](#_Toc451518455)

[קוד התוכנה 32](#_Toc451518456)

# 

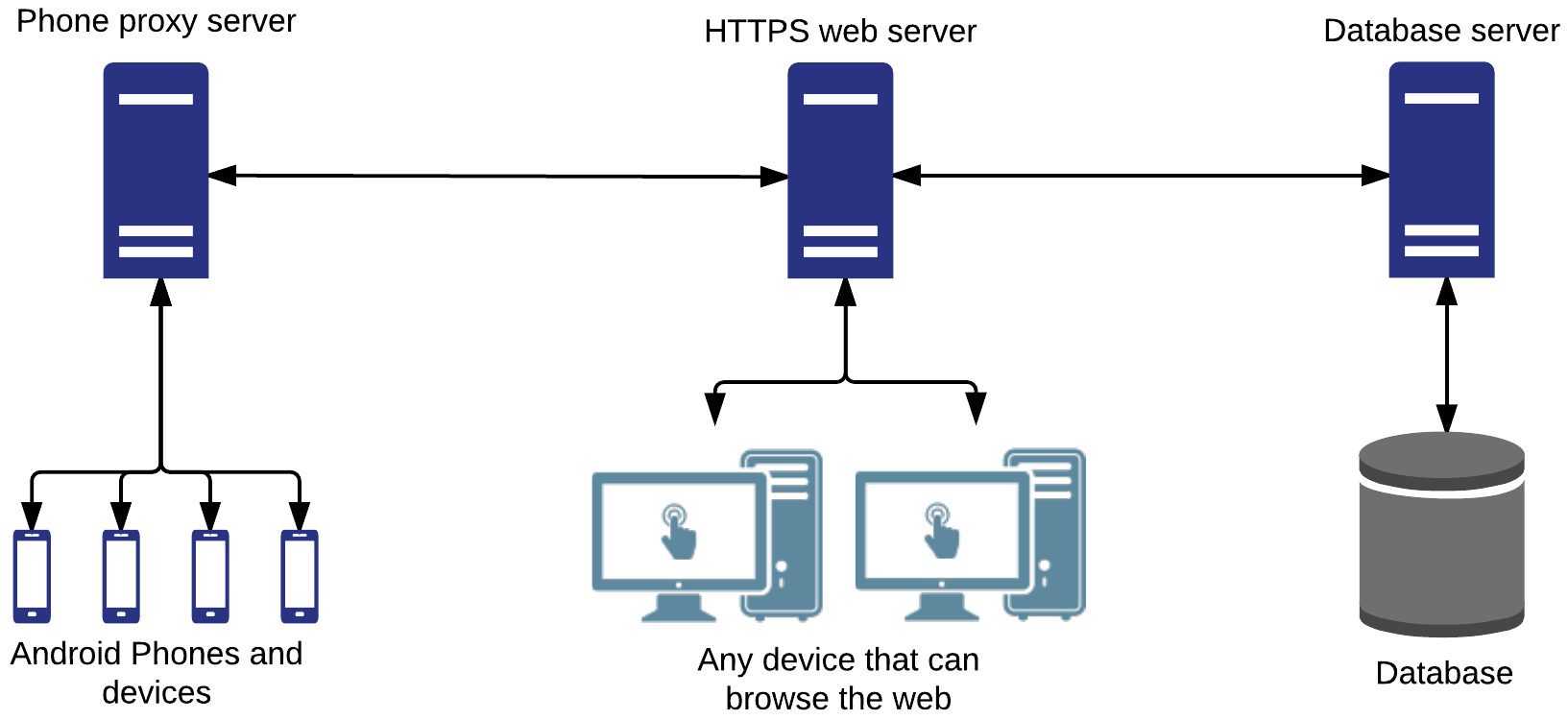
# תקציר ורציונל הפרויקט

בימינו אלה להאנשים יש שתי עמדות עבודה; העמדה הניידת שהיא הטלפון והעמדה הנייחת שהיא המחשב הנייח. האנשים משתמשים במחשב לצורך כתיבה וניהול מסמכים ומעוניינים לנהל את אותו מידע גם מהטלפון. ניהול המידע גם מהטלפון מצריך יכולת ניהול קבצים וסינכרון בין המחשב לבין הטלפון.

הפתרון שלנו לבעיה זו הוא ה"RDroid"; שירות שנותן ללקוח להעביר מידע מההמחשב לטלפון וגם מהטלפון למחשב.

השירות יינתן בעזרת אפליקציית אנדריוד בטלפון אשר תאפשר העברת מידע בין המשתמשים לבין הטלפון הפרטי שלהם. המידע יועבר על ידי כך שהמשתמשים יתממשקו לאתר אינטרנט אשר הוא יתקשר לאפליקציה המותקנת בטלפון שלהם.

הסיבה שבחרתי לעבוד בפרויקט על ניהול טלפונים מרחוק היא שלעיתים קרובות מצאתי את עצמי במצבים בהם לא הייתה לי דרך לגשת למידע חשוב שהיה זמין בטלפון בשעה שעבדתי על המחשב, ולפעמים לא היה לי המידע בטלפון כאשר המידע היה על המחשב.

מבנה התקשורת של השירות בנוי על העברת קבצים בין המחשב לטלפון באמצעות תקשורת שנעשית דרך שרת Proxy ושרת המשמש כאתר אינטרנט.

# מבוא ורקע כללי

הפרויקט מבוסס על תקשורת בין שרתים ומכשירים.

הלקוחות מתקשרים לשרת הראשי ע"י כניסה לאתר בשרת. באמצעות האתר הלקוחות מנהלים את קבציהם. השרת ראשי מתקשר דרך שרת PROXY לטלפונים עצמם.

הנושא המרכזי של הפרויקט הוא ניהול תקשורת עם לקוחות ועבודה עם המערכת ההפעלה אנדרואיד. פיתוח אפליקציות לאנדרואיד שונה מאוד מפיתוח תוכנות בווינדוס. לדוגמא: באנדרואיד App אינו מסוגל לרוץ אם הוא לא מופיע על המסך. לכן, צריך ליצור Service שיפעל ברקע ושה-App יתקשר איתו.

הפרויקט מממש את העברת הקבצים באמצעות העברת פקודות ומידע בין מכשירים שונים שמאפשר ניהול טלפונים/App מרחוק דרך האינטרנט.

# תיאור המוצר המוגמר

שם המוצר - Rdroid.

המוצר מבצע - העברת מידע מרחוק בין המחשב לטלפון דרך האינטרנט באופן יחודי לכל משתמש. המוצר עושה זאת בעזרת שרתים המספקים ממשק למשתמש לניהול, ומעבירים את המידע מהמחשב האישי למכשיר טלפון וממכשיר הטלפון למחשב האישי. השרתים מזהים את הלקוחות בעזרת כתובת אימייל וסיסמא, וכך מוודאים שהטלפון והלקוח שמתחבר לאתר הם זוג "חוק" (שמשתמשים לא יוכלו לגשת למידע שלא שלהם).

השאלות שהמערכת עונה עליהן:

1. **כיצד להגן על המידע הרגיש מפני גורמים לא מורשים?** התקשורת בין הטלפונים לשרת מוצפנת, כך שגורם לא מורשה לא יוכל להפיק שום תועלת מהאזנה למידע הנשלח, הוא לא יוכל לגלות שום מידע הנמצא על הטלפון ומה הפקודות שהלקוח מבקש ממנו לעשות.
2. **כיצד לעבוד בו-זמנית עם כל הטלפונים השונים?** כאשר טלפונים מתחברים לשרת, הוא פותח Thread נפרד לכל טלפון ובכך מאפשר לתקשר במקביל עם כולם, לקבל מהם מידע ולשלוח להם פקודות. כל טלפון ולקוח מתחבר עם שם משתמש וסיסמא כך שלא ניתן לגשת לטלפונים של אחרים.

# שפת התכנות וסביבת העבודה

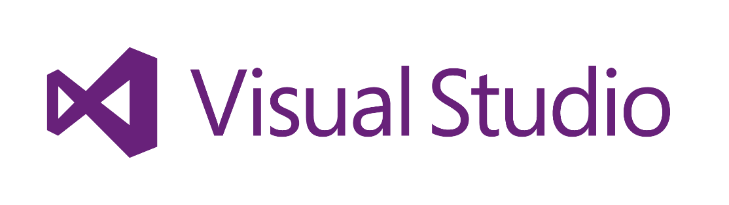
הפרויקט נכתב ב- Python, C# ו-Java.

* שרת הפרוקסי נכתב ב- Python בשל הפשטות והנוחות של השפה.
* השרת הראשי נכתב ב- C# כי התלמיד שמימש את השרת הראשי למד את סביבת .Net לעבודה בשירותי רשת.
* האפליקציה נכתבה ב-Java מכיוון שגוגל תומכים בשפה זו לסביבת אנדריוד.

סביבת העבודה עם Python הייתה PyCharm שהיא תוכנה מקצועית ונוחה לעבודה על פרויקטים מרובי מודולים וקבצים. היא שימשה גם לפיתוח הקוד ועבודה במקביל עם כל המודולים השונים וגם להרצתו בשביל בדיקות.

סביבת העבודה עם C# הייתה Microsoft Visual Studio 2015 ששימשה לכתיבת השרת הראשי באמצעות סביבת .Net, וה- debugger המובנה שימש לבדיקות. אני כתבתי מספר מחלקות שנועדו לפשט את התקשורת אל שרת הפרוקסי אך רוב העבודה בשפה זאת נעשתה ע"י התלמיד שעבד איתי, יובל ינושפולסקי.

סביבת העבודה על Java היא Android Studio שהיא סביבת העבודה של גוגל לפיתוח אפליקציות לאנדרואיד. הסביבה נוחה ותומכת במגוון פעולות העוזרות לפיתוח האפליקציה. בסביבת העבודה יש גם Debugger ו-Emulator למכשירי אנדרואיד.



# http://famouslogos.net/images/android-logo.jpghttp://www.eliteisinc.com/wp-content/uploads/2015/11/android-studio.pnghttp://www.logospike.com/wp-content/uploads/2014/11/Java_logo-2.jpg

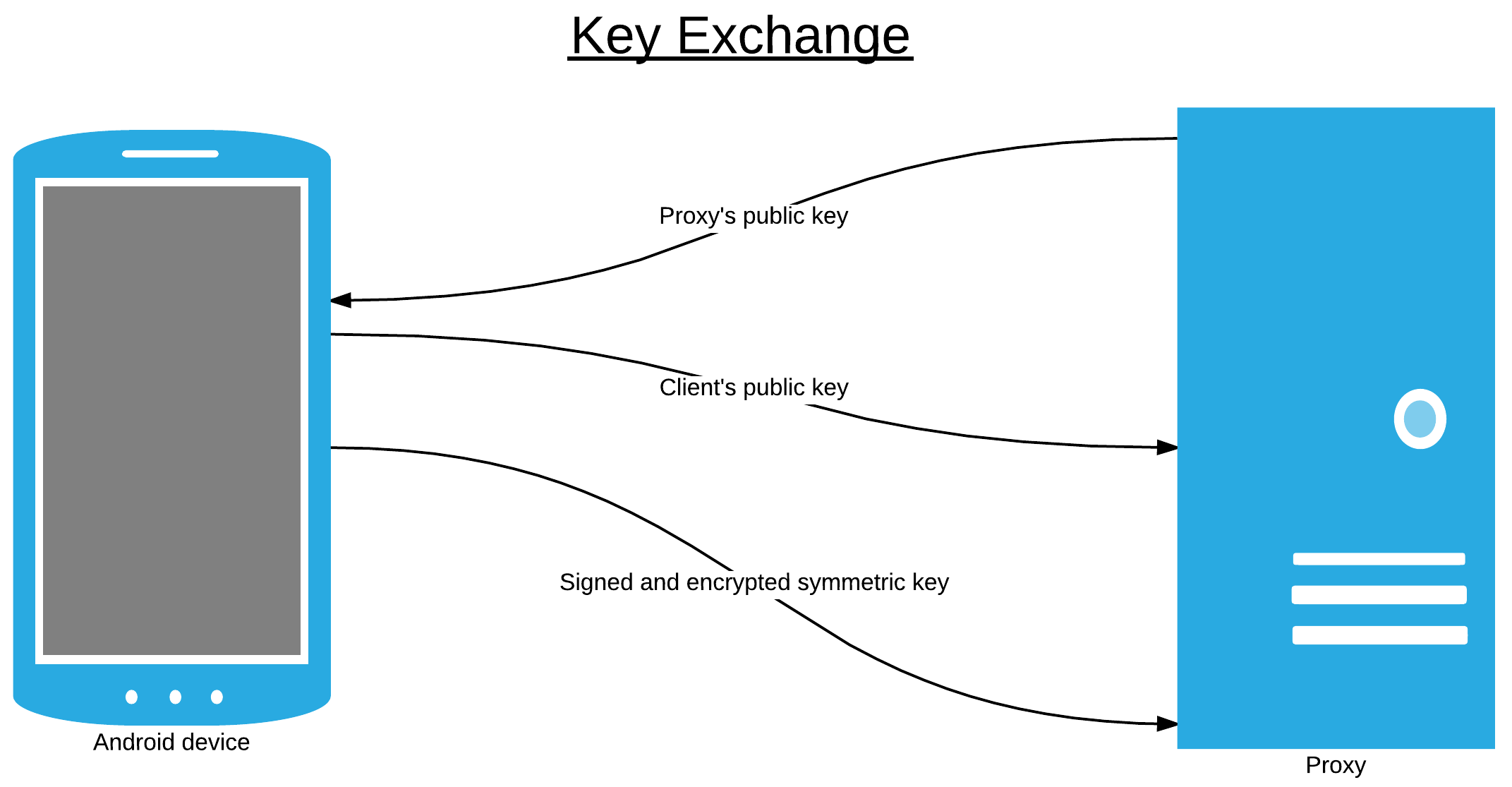
# ניסוח, ניתוח ופתרון הבעיה האלגוריתמית

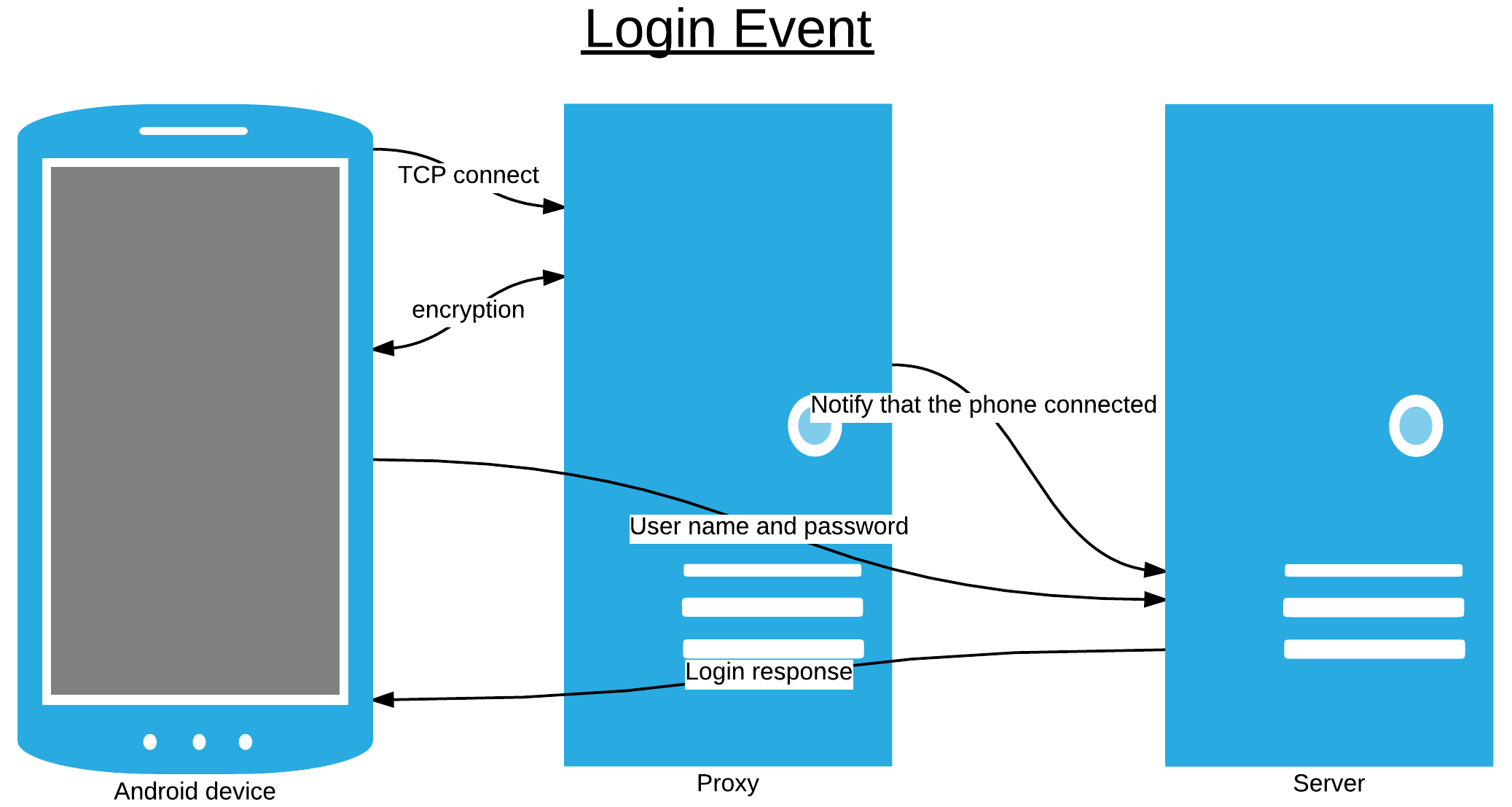
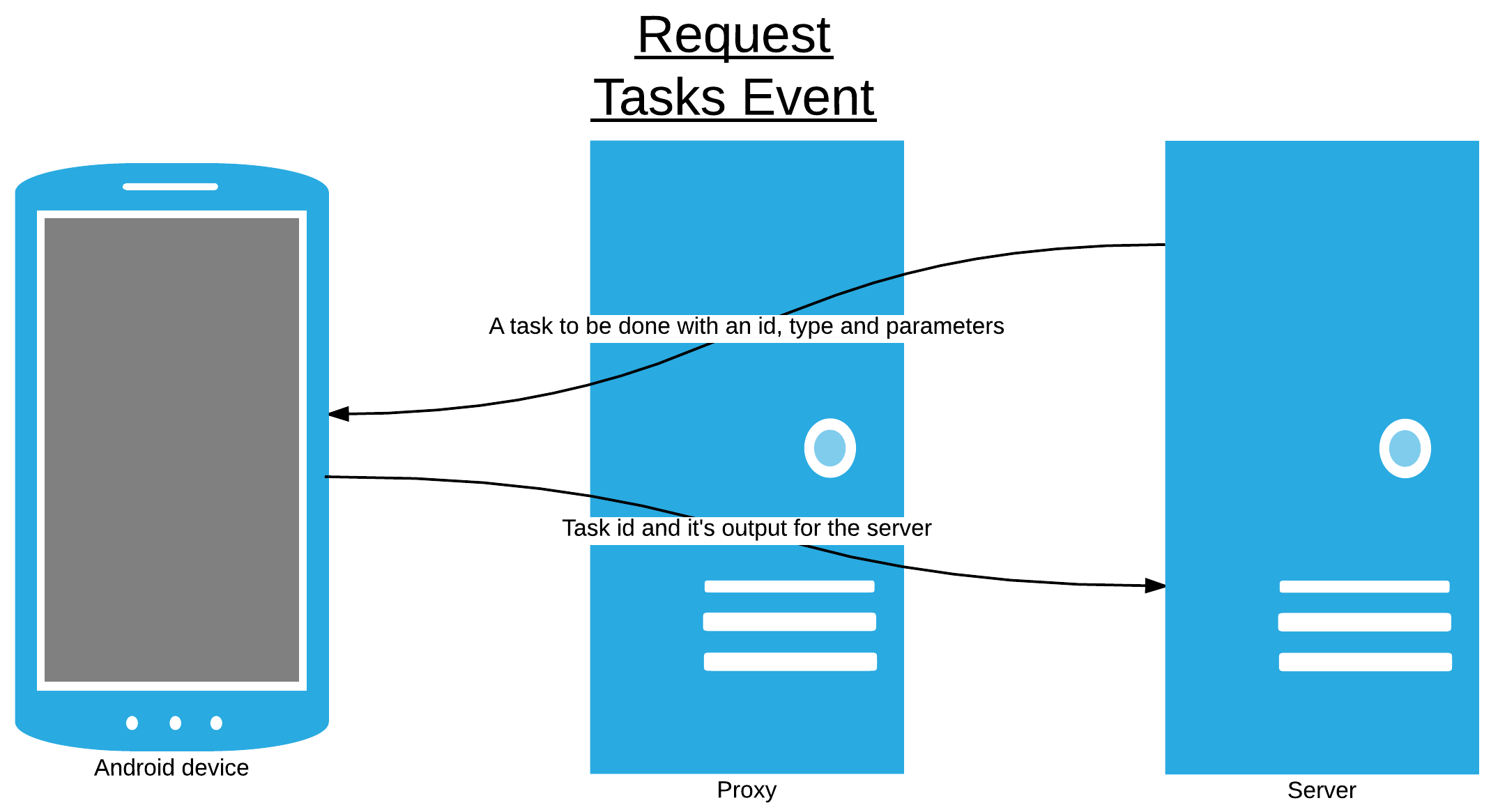
במהלך פיתוח המוצר נתקלתי בבעיות אלגוריתמיות רבות:

* **מימוש שרת מולטי-קליינט:**

הבעיה הראשונה שעמדה בפניי בפיתוח מוצר זה היתה יצירת שרת מולטי-קליינט. כלומר, הייתי צריך לממש שרת שמתחברים אליו מספר לקוחות בו-זמנית והוא מסוגל לדבר עם כולם. בתוך בעיה אלגוריתמית זו ישנן מספר תתי-בעיות:

* **פיתוח פרוטוקול משלי:** הפרוטוקול שפיתחתי צריך לנהל תקשורת בין הטלפונים לבין הפרוקסי ובה בעת לנהל את התקשורת בין הפרוקסי לבין השרת הראשי. במקביל הפרוטוקול צריך לתמוך בהצפנה ובתמיכה בכמה לקוחות במקביל.
* **מימוש אבטחת מידע:** כדי למנוע מהמידע לעבור בין השרת ללקוחות באופן חשוף הייתי צריך לממש אבטחת מידע, כלומר להוסיף לפרוטוקול יכולת להצפנה. הבעיה האלגוריתמית היא שליחה של מידע מוצפן בפרוטוקול ופענוח שלו בצד השני של הקו. המימוש נעשה ע"י שימוש במפתח הצפנה אסימטרי לצורך העברה ושימוש של מפתח הצפנה סימטרי.
* **העברת מידע בין השרתים:** מכיוון שהטלפונים לא מתחברים ישירות לשרת, היה צורך להוסיף לפרוטוקול חלק נוסף שנועד לניווט ההודעות בין השרת לטלפונים דרך שרת פרוקסי והתראות על מצבי החיבור איתם.

הנה פירוט הפרוטוקול:



המבנה של הפקטות:

כל הפקטות של הלקוח מתחילות ב-

Rdroid CLIENT

כל הפקטות של שרת הפרוקסי מתחילות ב-

Rdroid PROXY

כל הפקטות של השרת הראשי מתחילות ב-

Rdroid SERVER

בסוף כל שורה בפקטה כתוב-

/n

בשורה השנייה של הפקטה כתובה מטרת/משימה הפקטה-

רשימת משימה:

LOGIN

TASK

TASK\_RESULTS

NOTIFY\_SESSION\_ID

NOTIFY\_SESSION\_DISCONNECT

בשורה השלישית והלאה של הפקטה מצויינים הפרמטרים של הפעולה-

דוגמא של שורה שלישית ורביעית של פקטת LOGIN של הלקוח:

email:mighty

password:python

פרמטרים של פעולות:

DOWNLOAD (CLIENT):

requests a file to download from the server.

location - the location of the requested file.

LOGIN (CLIENT):

logins into the server.

username - the username of the client

password - the password of the client

LOGIN (SERVER):

informs the client whether the login was successful or not.

result - answers whether the login was successful or not.

TASK (SERVER):

sends a list of task that the client needs to do

task block - because this packet can send more than one task, each task will be defined in a block where all of its parameters are present.

<Task> - indicates the start of a task block.

</Task> - indicates the end of a task block.

id - the id of the task used to identify the results that are sent back to the server after the task was done.

type - the type of the task.

parameter - used when the task needs parameters. example: can provide a file address that is used by the client to request a file to download.

TASK\_RESULTS (CLIENT):

sends the results of the tasks that were given to the client.

id - the id of the task.

output - the results of the task or the information that it needs to return.

NOTIFY\_SESSION\_ID (PROXY)

notifies the server of a new session id that is given to a phone by the proxy.

session\_id - the id of the session

NOTIFY\_SESSION\_DISCONNECT(PROXY)

notifies the server of a session that disconnected.

session\_id - the id of the session.

כל הפקטות מתחילות במספר המייצג את אורך הפקטה (ללא המספר) ואחריו התו : (שגם לא נכלל בספירה).

כל הפקטה חוץ מהחלק שכותב את אורכה מומרת ל-Base64 ובמידה ומסודרת הצפנה אז היא גם מוצפנת לפני ההמרה ל-Base64.

בנוסף, במעבר בין השרת הראשי לשרת הפרוקסי ובין שרת הפרוקסי לשרת הראשי כל פקטה תקבל בתחילתה את ה-id של ה-session. לדוגמא (id=123):

123:message

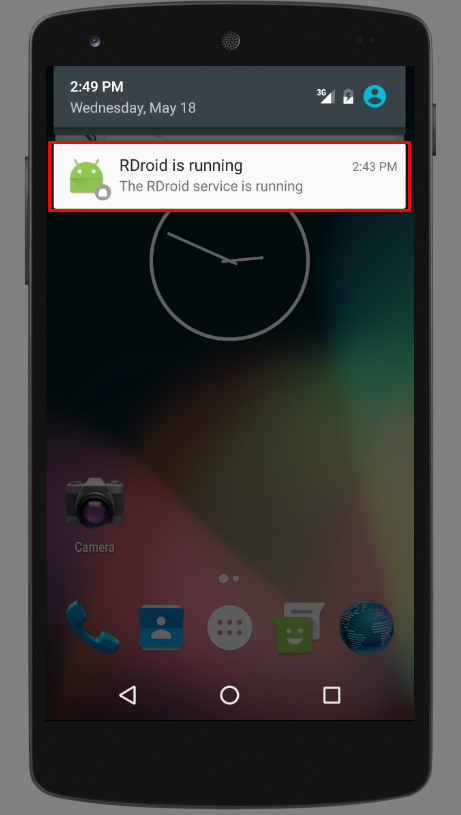
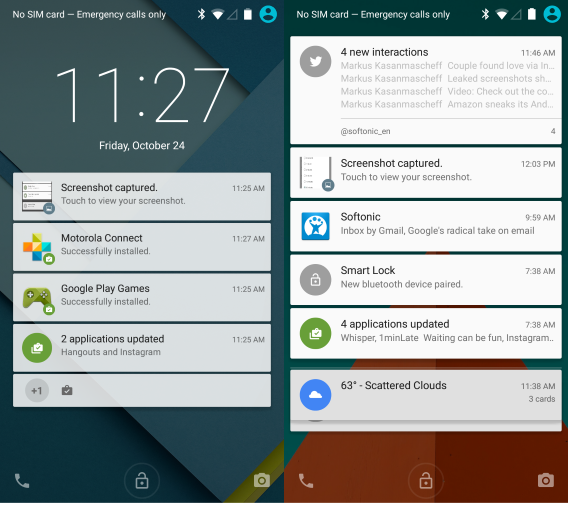
הנקודותיים מסמלות את סוף ה-id ותחילת ההודעה עצמה.

* **העברת קבצים:**

לקראת סוף הפרוייקט נתקלתי בבעיה בהעברת קבצים מכיוון שלא יכולתי לטפל במידע גדול. לעומת השרתים שפועלים על המחשב ויכולים להשתמש בהמון זיכרון Ram בכדי לשלוח קבצים בשלמותם, הטלפונים אינם מסוגלים להקצות להם הרבה זיכרון ולרוב גם מגבילים את הזיכרון שניתן לאפליקציות ושירותים. ישנם טלפונים יחסית חדשים שמקצים כ-30MB לאפליקציות. בכדי לשלוח קבצים בשלמותם צריך לשלוח אותם בחלקים. בשביל לשלוח את הקבצים השתמשתי ב-Streams שבעזרתם קיבלתי ושלחתי את המידע בחלקים. שליחת ההודעה בחלקים גרם לבעיה עם ההצפנה הסימטרית AES. היה צורך לשנות את המצב שימוש בהצפנה ל-CBC בכדי שיהיה ניתן לשלוח את ההודעות הללו ובסופו של דבר, היה יותר מדי קוד לשנות ולא מספיק זמן לעשות זאת בצורה מוצפנת.

* **הרצת האפליקציה ברקע:**

הפרוייקט שלי מבוסס על הצורך שהאפליקציה תעבוד כל הזמן ותגיב לשרת בכל עת. מכיוון שבאנדרואיד לא ניתן להריץ תהליך שהמשתמש לא יכול לראות, יצרתי שירות שמותחל ע"י האפליקציה ובפתיחת המכשיר. בכדי שהשירות ישאר דלוק הוא מייצר “Ongoing Notification” שהיא הודעה שמופיעה בחלקו העליון של המסך ולא נעלמת (אלא אם השירות נסגר או מגדיר את הנוטיפיקציה אחרת או שהמשתמש מגדיר שלא לראות את הנוטיפיקציה יותר).



* **אפשרות לקביעת הגדרות לשרת הפרוקסי:**

רציתי לאפשר קינפוג מהיר של שרת הפרוקסי בכדי שיהיה ניתן להתאים אותו לתנאים שונים. החלק העיקרי שניתן לקנפג הוא הכתובות IP ופורטים של שרת הפרוקסי והשרת הראשי, כך שיהיה ניתן להתאים את השרת במקרה שהשרת הראשי עובר לכתובת אחרת או משנה Port.

הסבר על שמירת ההגדרות בקובץ:

ההגדרות של השרת פרוקסי יישמרו במסמך טקסטואלי. ההגדרות יישמרו בצורה של מילון: כל הגדרה תשמר כמפתח וערכו. כל שורה במסמך תהיה הגדרה נפרדת. בכל שורה, המפתח הוא הטקסט עד ה-= וערכו הוא כל שאר הטקסט. שורות המתחילות ב-# הינן שורות הערה ולכן אינן נקלטות כהגדרות בתוכנה.

דוגמא:

#network

my\_ip=133.74.20.42

my\_port=8888

במקרה הזה שמור שכתובת השרת היא 133.74.20.42:8888.

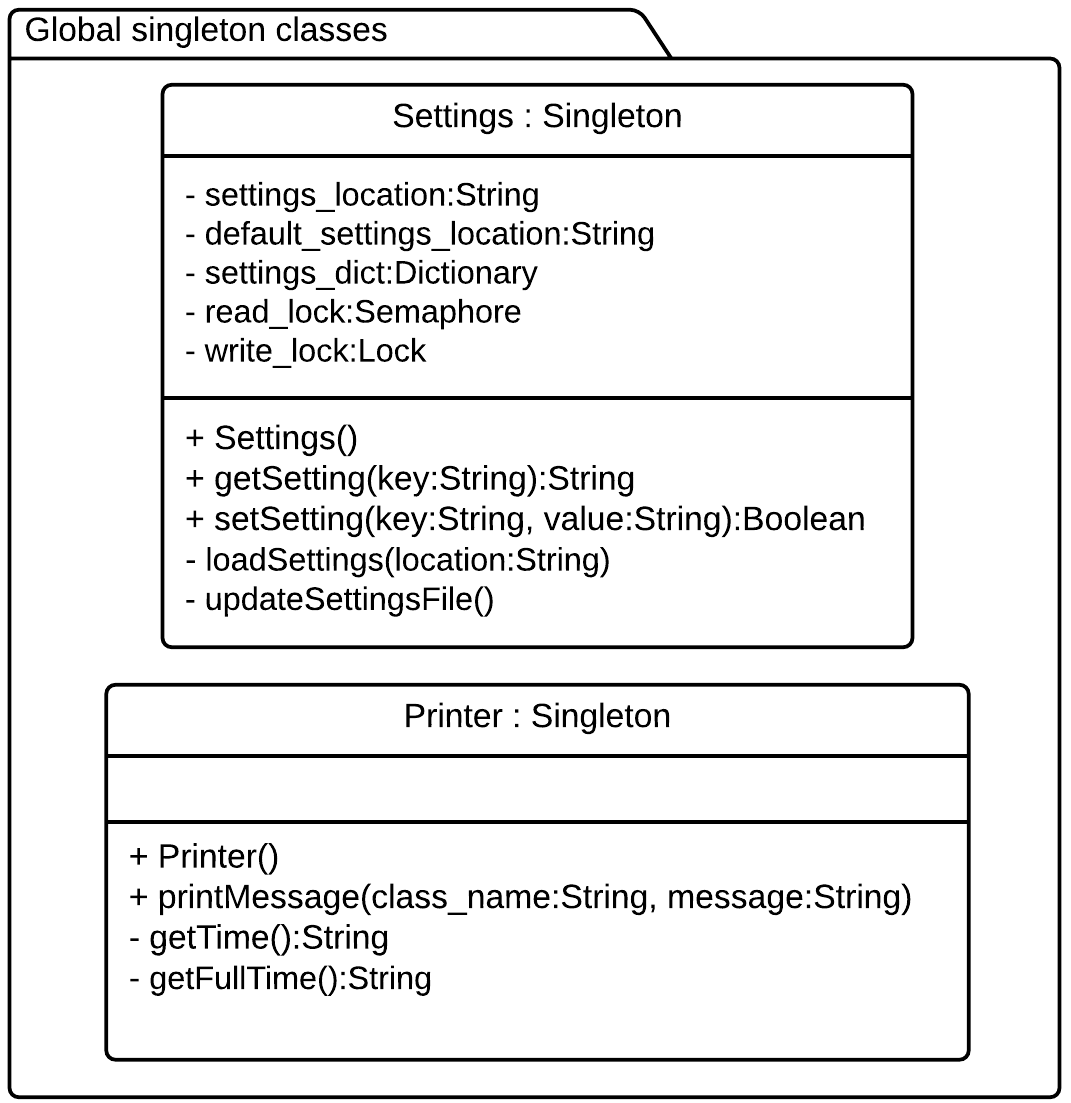
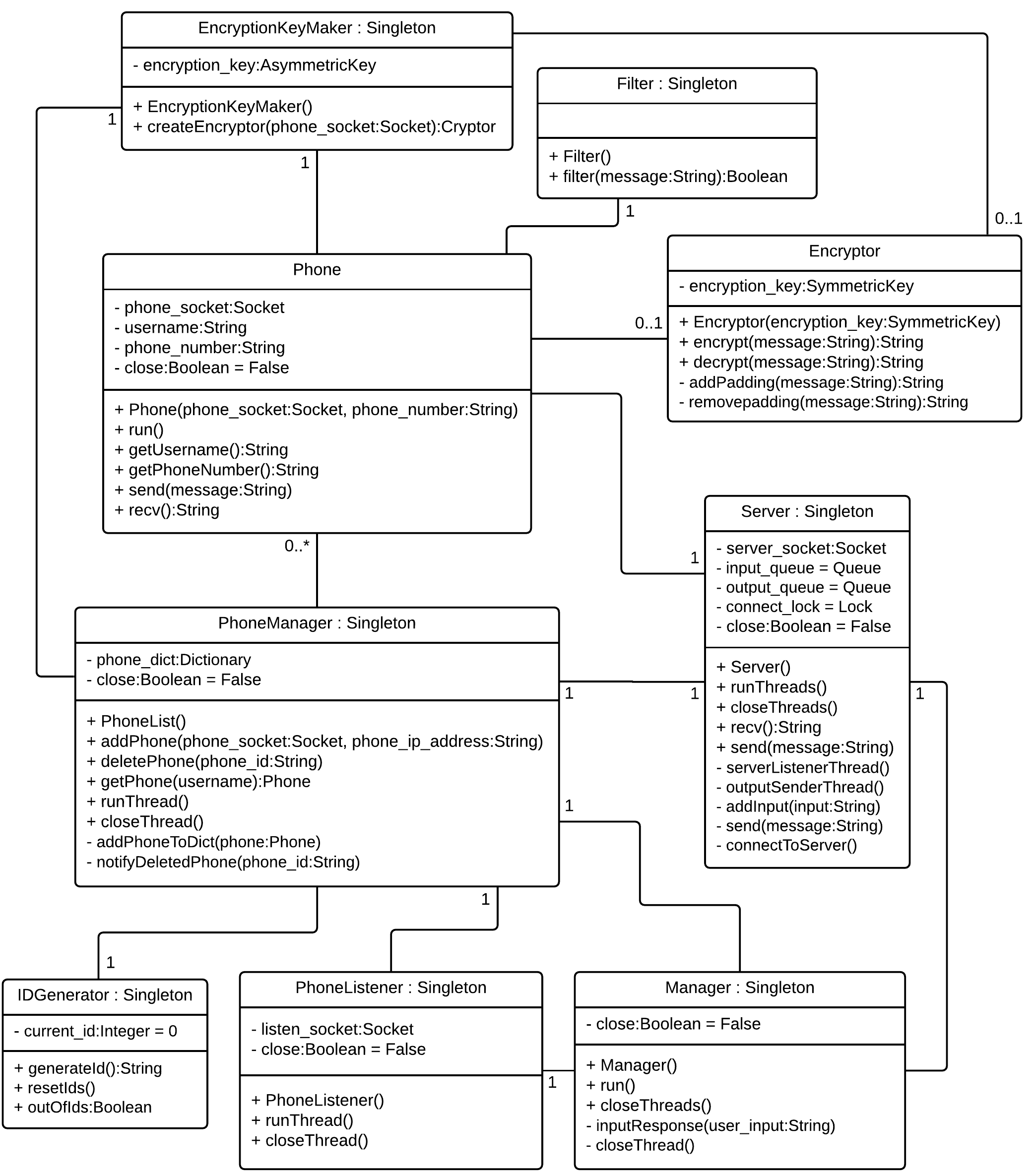
* **שמירת מידע באפליקציה:**

לאנדרואיד יש מערכת קבצים הדומה לזו של וינדוס. בנוסף למערכת הקבצים הזו יש עוד מערכת המנהלת את אחסון המידע של אפליקציות. המערכת הזאת שומרת את המידע של התוכנה בצורה של מילון ערכים. בנוסף ניתן להגדיר אם חלק מהמידע יהיה נגיש ע"י אפליקציות אחרות וכמובן לגשת למידע פומבי של אפליקציות אחרות.

בכדי לשמור את המידע חקרתי על השיטות השונות של שמירה ובסופו של דבר השתמשתי במערכת האחסון האישית לאפליקציה.

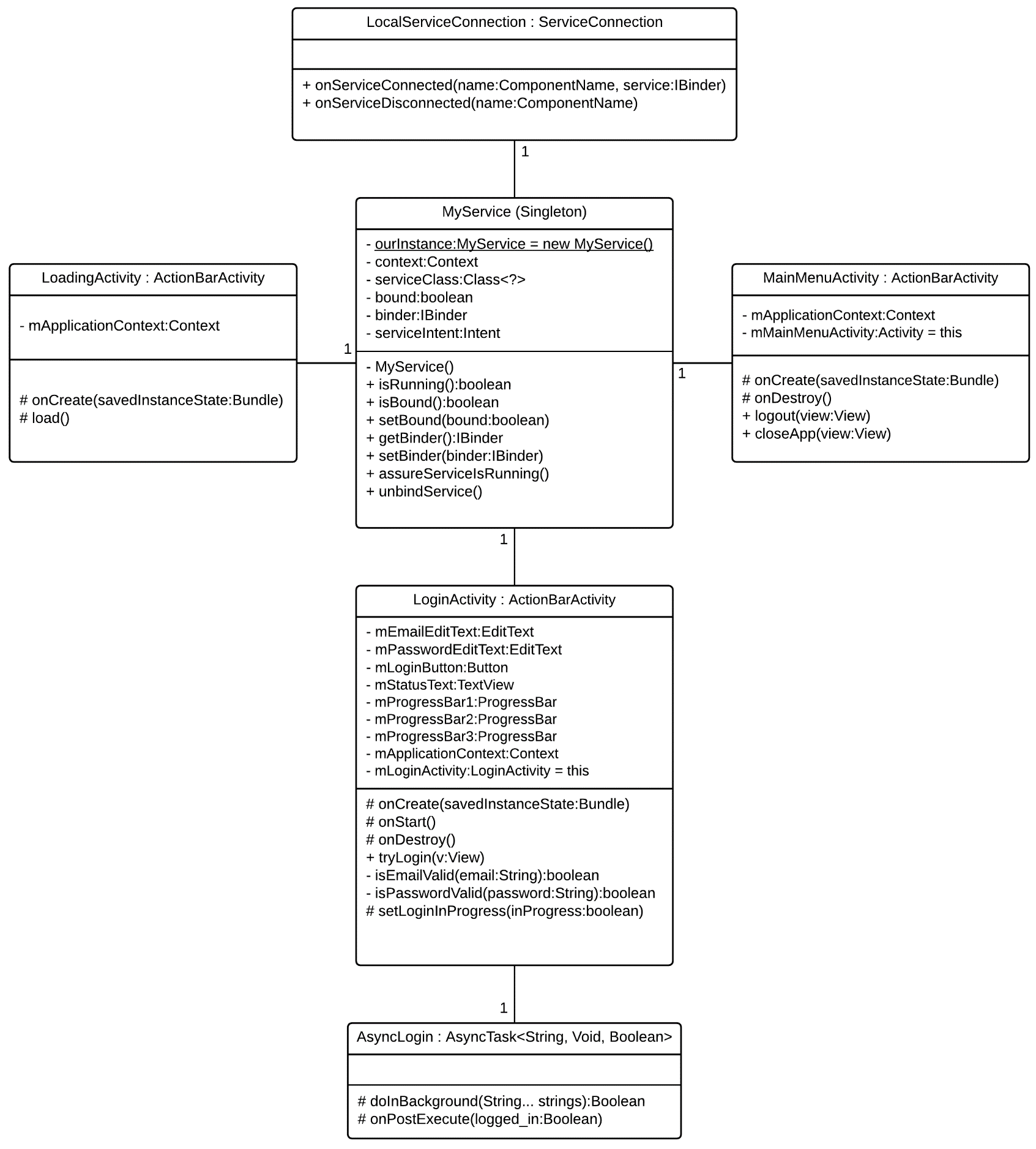
# תיאור מבנה המוצר

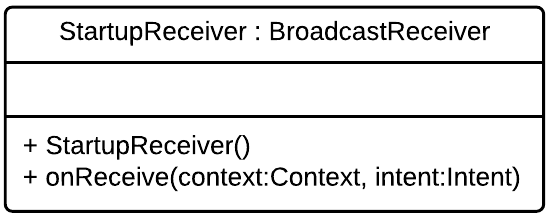
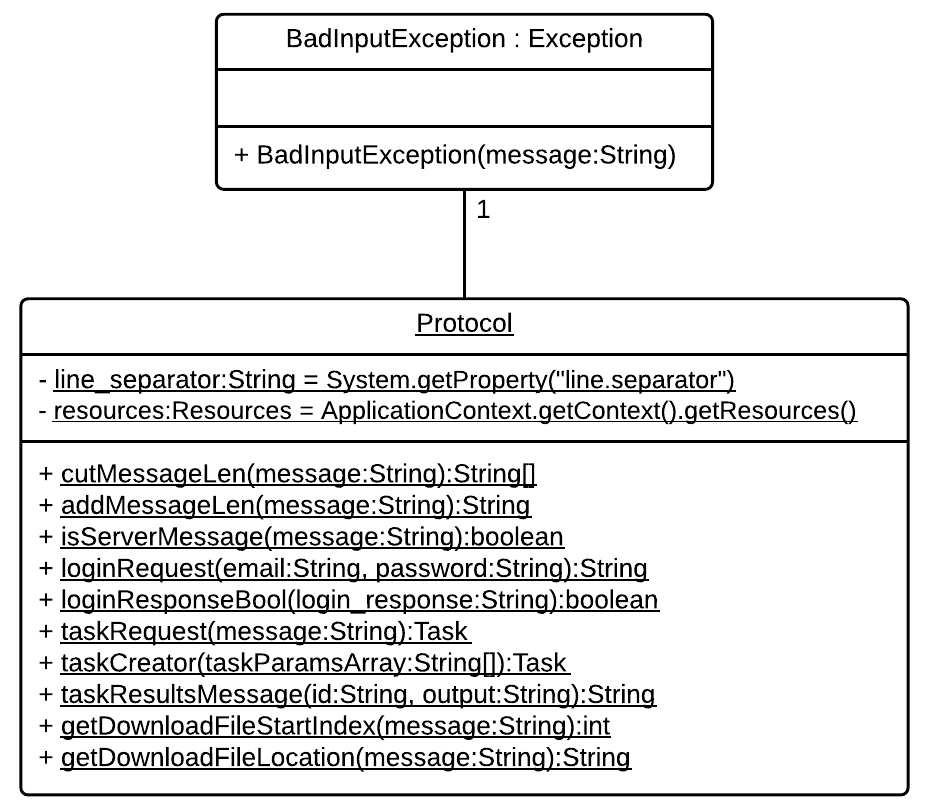
תרשימי UML של שרת ה-Proxy:

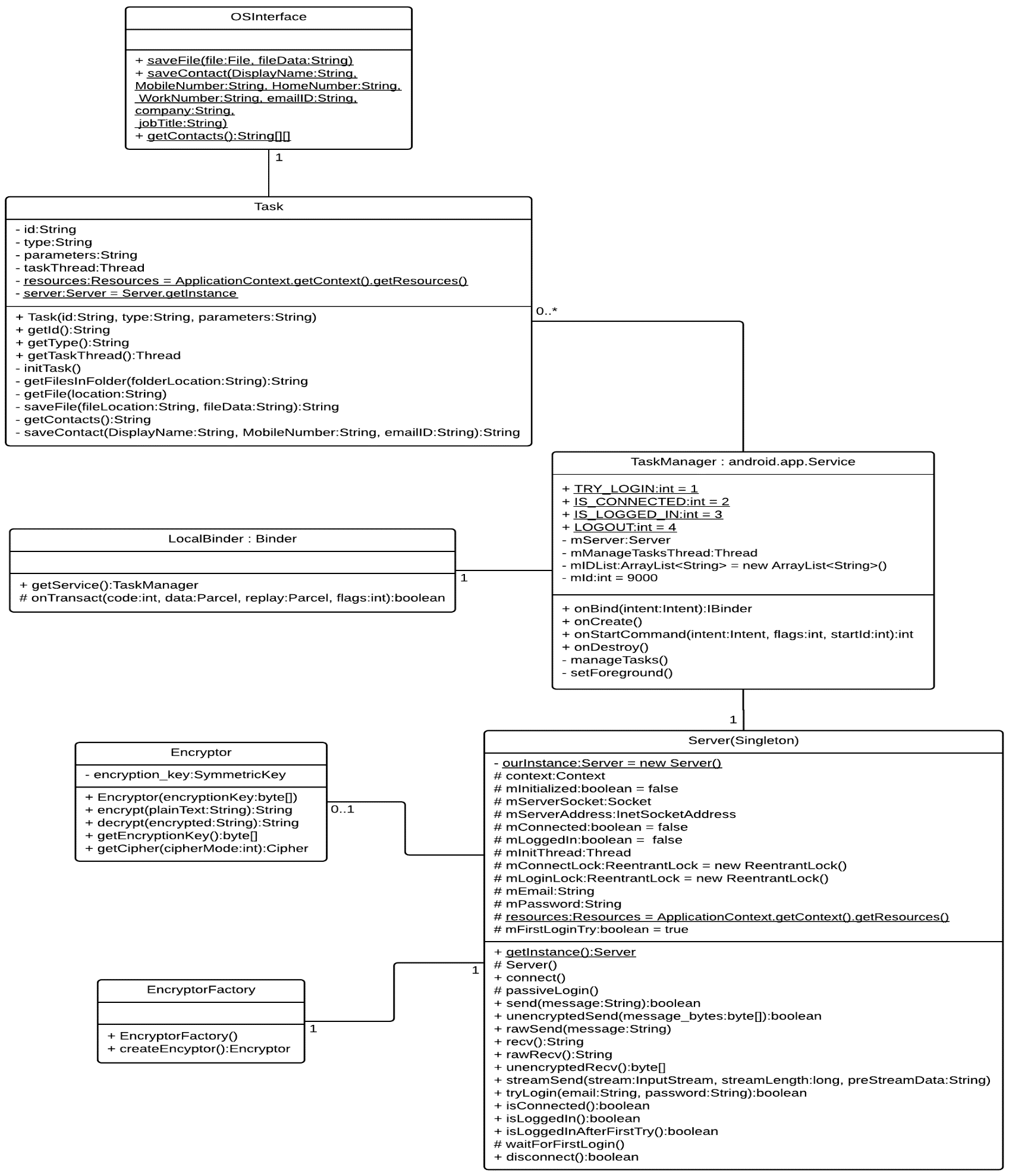


המודולים הם:

* **Main –** המודול שמתחיל את ריצת השרת. יוצר Manager חדש ומריץ אותו.
* **Manager** – המודול האחראי על התחלת Threads של תקשורת וניהול קלט מהמנהל (בממשק טקסטואלי).
* **Server** –המודול האחראי על ניהול התקשורת עם השרת הראשי.
* **Phone** – מודול האחראי על ניהול תקשורת עם טלפון.
* **PhoneListener** – מודול האחראי על קבלת חיבורים חדשים מטלפונים.
* **PhoneManager** – מודול האחראי על שמירת אובייקטי Phone הקיימים וניהול התקשורת איתם.
* **EncryptionKeyMaker** – אחראי על קביעת מפתח משותף עם טלפון ויצירת Encryptor למפתח הזה.
* **Encryptor** – אחראי על הצפנה ופיענוח של הודעות לפי המפתח שהוא הותחל איתו.
* **Filter** – פילטר להודעות שלא תואמות לפרוטוקול.
* **IDGenerator** – מייצר מזהים ייחודים שנועדו לזיהוי הטלפונים השונים.
* **Printer** – מדפיס הודעות לממשק של מנהל השרת (ממשק טקסטואלי). מדפיס את הזמן ואת המודול שמדפיס את ההודעה לפני ההודעה.
* **Settings –** אחראי על שמירה וקריאת הגדרות מקובץ ההגדרות.
* **Singleton –** Metaclass שהופך את מודולים ל-Singletons.

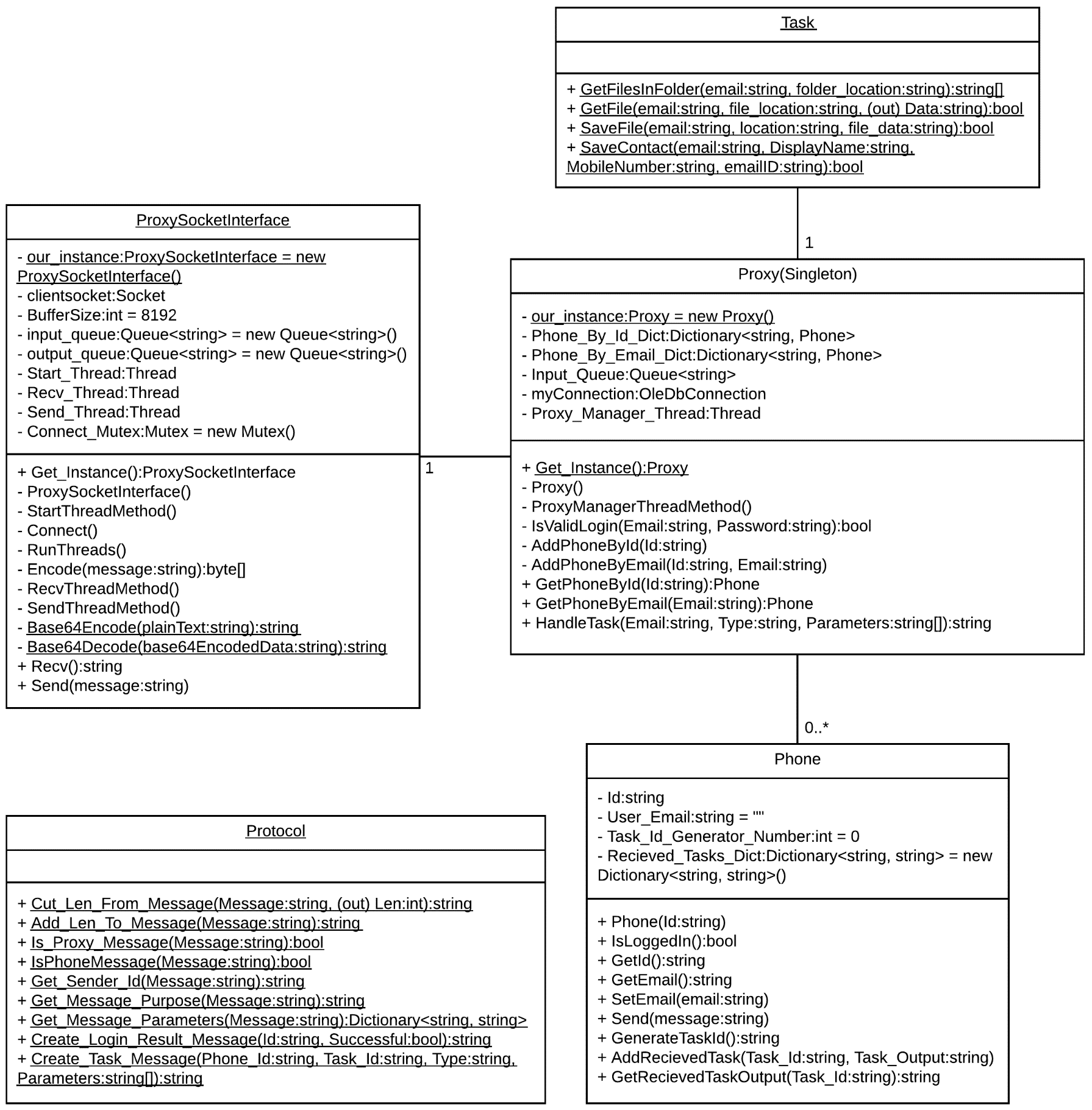
תרשימי UML של האפליקציה:





המודולים הם:

* **ה-Activities** – באפליקציה יש 3 Activities. התפקיד של המודולים הינו לנהל את ממשק המשתמש.
* **MyService** – מודול האחראי על ניהול התקשורת של האפליקציה עם ה-Service.
* **StartUpReciever** – מודול שמתחיל את ה-Service כאשר המכשיר נדלק (מריץ את TaskManager).
* **Protocol –** אחראי על כתיבת וניתוח הודעות לפי הפרוטוקול.
* **TaskManager –** אחראי על קבלת Tasks מ-Server, התחלתם ושליחת התוצאות שלהם חזרה לשרת. אחראי גם על התקשורת עם האפליקציה.
* **Server –** אחראי על תקשורת עם השרת.
* **EncryptorFactory –** אחראי על יצירת Encryptor להצפנת הודעות עם השרת.
* **Encryptor –** אחראי על הצפנה ופיענוח של הודעות עם השרת.
* **Task –** אחראי על ביצוע מטלות הנשלחות מהשרת והחזרת פלט מהן.
* **OSInterface –** ממשק להרצת פעולות של מערכת ההפעלה.

תרשים UML של הספריות לשרת הראשי:

המודולים הם:

* **Proxy –** אחראי על ניהול התקשורת עם שרת הפרוקסי, מעקב אחרי אילו טלפונים מחוברים ואילו התחברו למשתמש.
* **ProxySocketInterface –** אחראי על התקשורת עם שרת הפרוקסי.
* **Phone –** מייצג טלפון שמחובר לשרת הפרוקסי.
* **Task –** ממשק המפשט את התקשורת עם הטלפונים בכך שהוא מייצר הודעות של משימות ומחזיר את התשובה של הטלפון.
* **Protocol –** מייצר ומנתח הודעות לפי הפרוטוקול.

# מדריך התקנה למשתמש

אילוצים ודרישות מערכת

* השרתים פותחו עבור מערכת ההפעלה Windows. הם נבדקו ונוסו בגרסת **Windows 10** בלבד, ועל כן השימוש בהם בגרסה אחרת של Windows הוא באחריות המשתמש בלבד. האפליקציה עובדת על כל מכשיר אנדרואיד מגרסא 5 ומעלה.
* בנוסף, יש לוודא שמותקן בכל מחשב קצה ובשרת**Python 2.7** ושמותקנת בו ספריית **PyCrypto** עבור גרסה זו של Python. בביליוגרפיה נמצאים לינקים להורדת Python ולהתקנת הספרייה.
* כמו כן, צריכה להיות מותקנת במחשב השרת **.NET Framework 4.0** או גרסה עדכנית יותר (התוכנה נבדקה בגרסאות **4.0** ו- **4.5**).

התקנת והרצת התוכנה

1. צריך להוריד את הקוד של השרתים והאפליקציה מ-GitHub.
2. צריך להתאים את הגדרות השרתים בכדי שיתחברו אחד לשני וירוצו על הפורטים הרצויים, לעשות Port Forwarding לפורטים אם צריך, וגם באפליקצייה צריך להתאים את כתובת ה-ip והפורט למיקום של השרתים בקובץ strings.xml.
3. צריך ליצור קובץ Apk של האפליקציה עם השינויים בקובץ strings.xml ולהתקין אותו במכשיר.

# תיאור הממשק למשתמש

ממשק המשתמש הינו מאוד מינימליסטי.

אני לא אפרט על הממשק של האתר מכיוון שאני לא עיצבתי אותו.

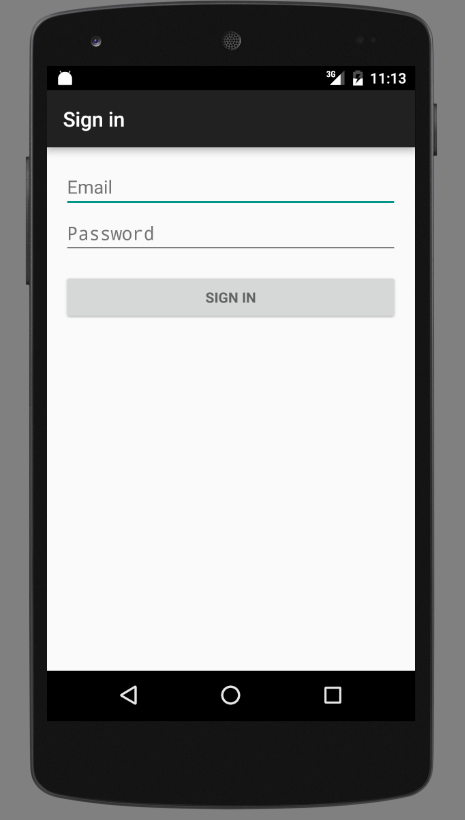
שרת הפרוקסי:

שרת הפרוקסי כתוב בפייטון ולכן יש לו ממשק טקסטואלי בסגנון CMD. השרת מדפיס הודעות על פעולתו וניתן לכתוב לו stop כדי שיסגור את עצמו.

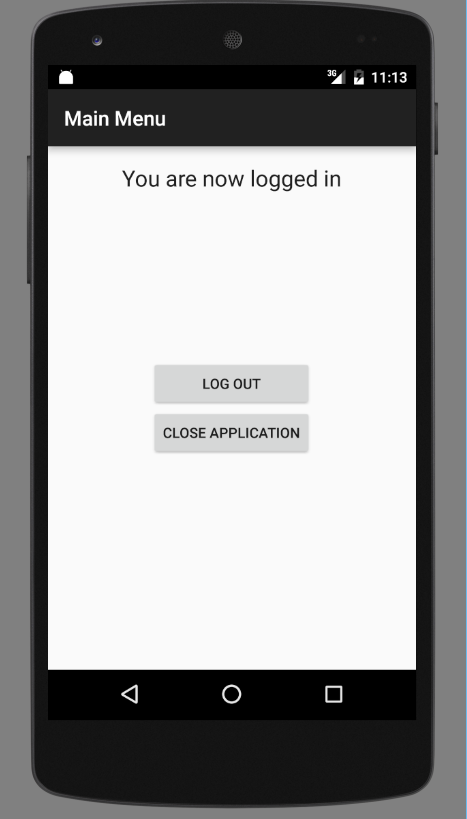
האפליקציה:

אפליקציות בטלפונים מורכבות וויזואלית מ-Activities, שהן כמו Forms ב-Windows, ועמודים באתרי אינטרנט. באפליקציה שלי ישנן 3 Activities:

* **Loading Activity** – נועדה בשביל להוות ממשק למשתמש בזמן שהאפליקציה מתחילה לרוץ. אם ישנם שם משתמש וסיסמא בזיכרון, ה-Activity מתחילה את Main Menu Activity, אחרת, היא מתחילה את Login Activity.
* **Login Activity** – נועדה בכדי שהמשתמש יוכל להזין שם משתמש וסיסמא כדי להתחבר לחשבון שלו. ב-Activity יש שתי תיבות טקסט להזנת שם משתמש וסיסמא וכפתור להיתחברות. אם המשתמש מתחבר בהצלחה, ה-Activity שומרת בזיכרון את שם המשתמש והסיסמא.

תמונה:

* **Main Menu Activity** – מהווה ממשק למשתמש שבו הוא יכול לבצע פעולות אחרי ההתחברות. יש בה שני כפתורים: כפתור להיתנתקות מהמשתמש שמעביר את האפליקציה ל-Login Activity וכפתור לסגירת האפליקציה (שהוא קצת מיותר).

תמונה:

# מבט אישי על העבודה ותהליך הפיתוח

העבודה על הפרויקט ערכה מספר חודשים בהם נחשפתי לטכנולוגיות שונות ומעניינות אשר בחזית עולם התכנות.

בעבודתי הייתי צריך לשלב תקשורות בין תוכנות שונות כאשר התוכנות עצמן מתייחסות לתקשורות באופן שונה אחת מהשניה.

אני רוצה לשים דגש על השקעתי בלמידת Android ואיך לעבוד בסביבת העבודה Android Studio ולשלב את החלק הויזואלי של ה-Activities. החלק הויזוראלי נכתב ב-Xml ולכן גם למדתי את השפה הזו לאורך הפיתוח (אומנם יש עורך מופשט שבו ניתן לגרור עצמים למסך אך עבודה עם Xml מדויקת יותר ויוצרת תוצאות חלקות יותר).

האתגרים והקשיים שנתקלתי בהם במהלך פיתוח הפרויקט הם:

* ניתוח הבעיה והצגה פורמאלי שלה
* לימוד סביבות עבודה ושפות חדשות
* תכנות Multitreads
* תקשורת מוצפנת חוצה סביבות ושפות שונות
* פיתוח פרוטוקול עצמאי ויעודי למשימה
* שילוב עבודה עם שרת ראשי שתוכנת ע"י תלמיד אחר תוך מציאת בגים ותיקוני תכנון ותכנות בתהליך

הערכת הפתרון לעומת התכנון:

* במקור רציתי להוסיף עוד תכונות שיאפשרו שליטה מרחוק בטלפונים אך לא נמצא מספיק זמן למימוש הרעיונות
* אין הצפנה להעברת הקבצים עקב קשיי מימוש (מחוסר זמן) למצבים שונים של AES שנועדו להצפין הודעות בחלקים
* הייתי מעוניין לשפר את הממשק הגרפי הן של ה APP והן של השרת הראשי.

# ביבליוגרפיה

1. אתר הדרכה לסייבר שיוסד על ידי המנחה שלי מיכאל:  
   [https://sites.google.com/site/coursecyber](https://sites.google.com/site/coursecyber/)
2. הפורום הגדול ביותר לשאלות ותשובות בנושאי תכנות:  
   [http://stackoverflow.com](http://stackoverflow.com/)
3. MSDN, הרשת הרשמית של Microsoft למפתחים:  
   <http://msdn.microsoft.com/library>
4. התיעוד הרשמי של Python:  
   [https://docs.python.org/2.7](https://docs.python.org/2.7/)
5. הוויקי הרשמית של Python:  
   [https://wiki.python.org/moin](https://wiki.python.org/moin/)
6. אתר ללימוד Pythonו-Java:  
   <https://www.codecademy.com>
7. להורדת Python:

<https://www.python.org>

1. להתקנת ספריית PyCrypto בשביל Python:

<http://www.voidspace.org.uk/python/pycrypto-2.6.1>

1. להורדת Android Studio:

<https://developer.android.com/studio/index.html>

1. להורדת Visual Studio:

<https://www.visualstudio.com>

**קוד התוכנה**

**האפליקציה:**

קבצי המידע:

**Strings.xml :**

<resources>  
 <string name="app\_name">Rdroid</string>  
 <string name="label\_activity\_loading">Rdroid</string>  
 <string name="label\_activity\_login">Sign in</string>  
 <string name="label\_activity\_main\_menu">Main Menu</string>  
 <string name="user\_data">user\_data</string>  
  
 <!-- Strings related to loading activity -->  
 <string name="loading\_text\_view\_text">Connecting to the server</string>  
  
 <!-- Strings related to login activity -->  
 <string name="prompt\_email">Email</string>  
 <string name="prompt\_password">Password</string>  
 <string name="action\_sign\_in">Sign in</string>  
 <string name="action\_sign\_in\_short">Sign in</string>  
 <string name="status\_login\_attempt">Logging in</string>  
 <string name="status\_login\_not\_connected">Not connected to server</string>  
 <string name="status\_login\_wrong\_parameters">Wrong email or password</string>  
 <string name="status\_login\_bad\_parameters">Invalid email or password</string>  
 <string name="status\_login\_successful">Login successful</string>  
  
 <!-- Strings related to main menu activity -->  
 <string name="log\_out\_button\_text">Log Out</string>  
 <string name="close\_application\_button\_text">Close Application</string>  
  
 <!-- Strings related to protocol -->  
 <string name="server\_address">79.178.66.183</string>  
 <string name="server\_port">9000</string>  
 <string name="protocol\_client\_header">Rdroid CLIENT</string>  
 <string name="protocol\_server\_header">Rdroid SERVER</string>  
 <string name="protocol\_parameter\_separator">:</string>  
 <!-- Strings related to protocol login -->  
 <string name="protocol\_client\_login\_announcement">LOGIN</string>  
 <string name="protocol\_client\_login\_email">email</string>  
 <string name="protocol\_client\_login\_password">password</string>  
 <string name="protocol\_server\_login\_announcement">LOGIN</string>  
 <string name="protocol\_server\_login\_bool">result</string>  
 <string name="protocol\_server\_login\_success">success</string>  
 <string name="protocol\_server\_login\_failure">failure</string>  
 <!-- Strings related to protocol task request -->  
 <string name="protocol\_server\_task\_request\_announcement">TASK</string>  
 <string name="protocol\_server\_task\_request\_id">id</string>  
 <string name="protocol\_server\_task\_request\_type">type</string>  
 <string name="protocol\_server\_task\_request\_parameters">parameters</string>  
 <string name="protocol\_server\_task\_request\_parameters\_separator">,</string>  
 <string name="protocol\_client\_task\_results\_announcement">TASK\_RESULTS</string>  
 <string name="protocol\_client\_task\_results\_id">id</string>  
 <string name="protocol\_client\_task\_results\_output">output</string>  
 <string name="protocol\_client\_task\_results\_output\_separator">/</string>  
  
</resources>

**Activity\_loading.xml:**

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:paddingBottom="@dimen/activity\_vertical\_margin"  
 android:paddingLeft="@dimen/activity\_horizontal\_margin"  
 android:paddingRight="@dimen/activity\_horizontal\_margin"  
 android:paddingTop="@dimen/activity\_vertical\_margin"  
 tools:context="com.yanoonigmail.rdroid.app.LoadingActivity">  
  
 <ProgressBar android:indeterminate="true"  
 android:layout\_width="50dp" android:layout\_height="50dp"  
 android:id="@+id/marker\_progress1" style="?android:attr/progressBarStyle"  
 android:layout\_gravity="center\_vertical|center\_horizontal"  
 android:layout\_centerVertical="true"  
 android:layout\_centerHorizontal="true" />  
 <ProgressBar android:indeterminate="true"  
 android:layout\_width="100dp" android:layout\_height="100dp"  
 android:id="@+id/marker\_progress2" style="?android:attr/progressBarStyle"  
 android:layout\_gravity="center\_vertical|center\_horizontal"  
 android:layout\_centerVertical="true"  
 android:layout\_centerHorizontal="true" />  
 <ProgressBar android:indeterminate="true"  
 android:layout\_width="150dp" android:layout\_height="150dp"  
 android:id="@+id/marker\_progress3" style="?android:attr/progressBarStyle"  
 android:layout\_gravity="center\_vertical|center\_horizontal"  
 android:layout\_centerVertical="true"  
 android:layout\_centerHorizontal="true" />  
  
 <TextView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:textAppearance="?android:attr/textAppearanceLarge"  
 android:text="@string/loading\_text\_view\_text"  
 android:id="@+id/loadingTextView"  
 android:layout\_below="@+id/marker\_progress3"  
 android:layout\_centerHorizontal="true" />  
</RelativeLayout>

**Activity\_login.xml:**

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:paddingBottom="@dimen/activity\_vertical\_margin"  
 android:paddingLeft="@dimen/activity\_horizontal\_margin"  
 android:paddingRight="@dimen/activity\_horizontal\_margin"  
 android:paddingTop="@dimen/activity\_vertical\_margin"  
 tools:context=".app.LoginActivity">  
  
 <ScrollView  
 android:id="@+id/login\_form"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <LinearLayout  
 android:id="@+id/email\_login\_form"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="vertical">  
  
 <EditText  
 android:id="@+id/email"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="@string/prompt\_email"  
 android:inputType="textEmailAddress"  
 android:maxLines="1"  
 android:singleLine="true" />  
  
 <EditText  
 android:id="@+id/password"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="@string/prompt\_password"  
 android:inputType="textPassword"  
 android:maxLines="1"  
 android:singleLine="true" />  
  
 <Button  
 android:id="@+id/login\_button"  
 style="?android:textAppearanceSmall"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="16dp"  
 android:text="@string/action\_sign\_in"  
 android:textStyle="bold"  
 android:onClick="tryLogin" />  
  
 <TextView  
 android:id="@+id/status\_text"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:textAppearance="?android:attr/textAppearanceLarge"  
 android:layout\_gravity="center\_horizontal" />  
 <RelativeLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
 <ProgressBar android:indeterminate="true"  
 android:layout\_width="50dp" android:layout\_height="50dp"  
 android:id="@+id/marker\_progress1" style="?android:attr/progressBarStyle"  
 android:layout\_gravity="center\_vertical|center\_horizontal"  
 android:layout\_centerVertical="true"  
 android:layout\_centerHorizontal="true"  
 android:visibility="invisible" />  
 <ProgressBar android:indeterminate="true"  
 android:layout\_width="100dp" android:layout\_height="100dp"  
 android:id="@+id/marker\_progress2" style="?android:attr/progressBarStyle"  
 android:layout\_gravity="center\_vertical|center\_horizontal"  
 android:layout\_centerVertical="true"  
 android:layout\_centerHorizontal="true"  
 android:visibility="invisible" />  
 <ProgressBar android:indeterminate="true"  
 android:layout\_width="150dp" android:layout\_height="150dp"  
 android:id="@+id/marker\_progress3" style="?android:attr/progressBarStyle"  
 android:layout\_gravity="center\_vertical|center\_horizontal"  
 android:layout\_centerVertical="true"  
 android:layout\_centerHorizontal="true"  
 android:visibility="invisible" />  
 </RelativeLayout>  
 </LinearLayout>  
 </ScrollView>  
</RelativeLayout>

**Activity\_main\_menu.xml:**

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:paddingBottom="@dimen/activity\_vertical\_margin"  
 android:paddingLeft="@dimen/activity\_horizontal\_margin"  
 android:paddingRight="@dimen/activity\_horizontal\_margin"  
 android:paddingTop="@dimen/activity\_vertical\_margin"  
 tools:context=".app.MainMenuActivity"  
 android:onClick="logOut">  
  
 <TextView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:textAppearance="?android:attr/textAppearanceLarge"  
 android:text="You are now logged in"  
 android:id="@+id/textView"  
 android:layout\_alignParentTop="true"  
 android:layout\_centerHorizontal="true" />  
  
 <ScrollView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/scrollView"  
 android:layout\_centerVertical="true"  
 android:layout\_centerHorizontal="true">  
  
 <TableLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content">  
  
 <TableRow  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
 <Button  
 android:id="@+id/logoutButton"  
 android:text="@string/log\_out\_button\_text"  
 android:onClick="logout" />  
 </TableRow>  
  
 <TableRow  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
 <Button  
 android:id="@+id/closeAppButton"  
 android:text="@string/close\_application\_button\_text"  
 android:onClick="closeApp" />  
 </TableRow>  
 </TableLayout>  
 </ScrollView>  
</RelativeLayout>

קבצים כלליים:

**ApplicationContext.java:**

package com.yanoonigmail.rdroid;  
  
import android.app.Application;  
import android.content.Context;  
  
*/\*\*  
 \* Created by yanoo on 03-Mar-16.  
 \*/*public class ApplicationContext extends Application {  
  
 private static Context *mContext*;  
  
 @Override  
 public void onCreate() {  
 super.onCreate();  
 *mContext* = this;  
 }  
  
 public static Context getContext(){  
 return *mContext*;  
 }  
}

קבצי האפליקציה:

**LoadingActivity.java:**

package com.yanoonigmail.rdroid.app;  
  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.Parcel;  
import android.os.RemoteException;  
import android.support.v7.app.ActionBarActivity;  
import android.os.Bundle;  
  
import com.yanoonigmail.rdroid.ApplicationContext;  
import com.yanoonigmail.rdroid.R;  
import com.yanoonigmail.rdroid.service.TaskManager;  
  
import static com.yanoonigmail.rdroid.R.string.*user\_data*;  
import static com.yanoonigmail.rdroid.service.TaskManager.*IS\_LOGGED\_IN*;  
  
public class LoadingActivity extends ActionBarActivity {  
 private MyService mMyService = MyService.*getInstance*();  
 private Context mApplicationContext = ApplicationContext.*getContext*();  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_loading*);  
 load();  
 }  
  
 protected void load() {  
 Thread loadingThread = new Thread(new Runnable() {  
 public void run() {  
 mMyService.assureServiceIsRunning();  
 Parcel input\_parcel = Parcel.*obtain*();  
 Parcel output\_parcel = Parcel.*obtain*();  
 try {  
 mMyService.getBinder().transact(*IS\_LOGGED\_IN*, input\_parcel, output\_parcel, 0);  
 } catch (RemoteException e) {  
 e.printStackTrace();  
 }  
 boolean[] bool\_array = new boolean[1];  
 output\_parcel.readBooleanArray(bool\_array);  
 output\_parcel.recycle();  
 Intent i;  
 if (bool\_array[0]) {  
 i = new Intent(mApplicationContext, MainMenuActivity.class);  
 } else {  
 i = new Intent(mApplicationContext, LoginActivity.class);  
 }  
 i.addFlags(Intent.*FLAG\_ACTIVITY\_NEW\_TASK*);  
 mApplicationContext.startActivity(i);  
 finish();  
 }  
 });  
 loadingThread.start();  
 }  
}

**LoginActivity.java:**

package com.yanoonigmail.rdroid.app;  
  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.os.AsyncTask;  
import android.os.RemoteException;  
import android.support.v7.app.ActionBarActivity;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.ProgressBar;  
import android.widget.TextView;  
import android.os.Parcel;  
  
import com.yanoonigmail.rdroid.ApplicationContext;  
import com.yanoonigmail.rdroid.service.TaskManager;  
  
import java.lang.Thread;  
  
import static com.yanoonigmail.rdroid.R.id.*email*;  
import static com.yanoonigmail.rdroid.R.id.*login\_button*;  
import static com.yanoonigmail.rdroid.R.id.*marker\_progress1*;  
import static com.yanoonigmail.rdroid.R.id.*marker\_progress2*;  
import static com.yanoonigmail.rdroid.R.id.*marker\_progress3*;  
import static com.yanoonigmail.rdroid.R.id.*password*;  
import static com.yanoonigmail.rdroid.R.id.*status\_text*;  
import static com.yanoonigmail.rdroid.R.layout.*activity\_login*;  
import static com.yanoonigmail.rdroid.R.string.*status\_login\_attempt*;  
import static com.yanoonigmail.rdroid.R.string.*status\_login\_bad\_parameters*;  
import static com.yanoonigmail.rdroid.R.string.*status\_login\_not\_connected*;  
import static com.yanoonigmail.rdroid.R.string.*status\_login\_wrong\_parameters*;  
import static com.yanoonigmail.rdroid.R.string.*user\_data*;  
import static com.yanoonigmail.rdroid.service.TaskManager.*IS\_CONNECTED*;  
import static com.yanoonigmail.rdroid.service.TaskManager.*TRY\_LOGIN*;  
  
public class LoginActivity extends ActionBarActivity {  
 private EditText mEmailEditText;  
 private EditText mPasswordEditText;  
 private Button mLoginButton;  
 private TextView mStatusText;  
 private ProgressBar mProgressBar1;  
 private ProgressBar mProgressBar2;  
 private ProgressBar mProgressBar3;  
 private MyService mMyService = MyService.*getInstance*();  
 private Context mApplicationContext = ApplicationContext.*getContext*();  
 private LoginActivity mLoginActivity = this;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(*activity\_login*);  
 mEmailEditText = (EditText) findViewById(*email*);  
 mPasswordEditText = (EditText) findViewById(*password*);  
 mLoginButton = (Button) findViewById(*login\_button*);  
 mStatusText = (TextView) findViewById(*status\_text*);  
 mProgressBar1 = (ProgressBar) findViewById(*marker\_progress1*);  
 mProgressBar2 = (ProgressBar) findViewById(*marker\_progress2*);  
 mProgressBar3 = (ProgressBar) findViewById(*marker\_progress3*);  
 }  
  
 @Override  
 protected void onStart() {  
 super.onStart();  
 mMyService.assureServiceIsRunning();  
 }  
  
 @Override  
 protected void onDestroy() {  
 super.onDestroy();  
 }  
  
 public void tryLogin(View v) {  
 setLoginInProgress(true);  
 Parcel input\_parcel = Parcel.*obtain*();  
 Parcel output\_parcel = Parcel.*obtain*();  
 try {  
 mMyService.getBinder().transact(*IS\_CONNECTED*, input\_parcel, output\_parcel, 0);  
 boolean[] val = new boolean[1];  
 output\_parcel.readBooleanArray(val);  
 output\_parcel.recycle();  
 if (!val[0]) {  
 mStatusText.setText(mApplicationContext.getString(*status\_login\_not\_connected*));  
 setLoginInProgress(false);  
 } else {  
 mStatusText.setText(mApplicationContext.getString(*status\_login\_attempt*));  
 String given\_email = mEmailEditText.getText().toString();  
 String given\_password = mPasswordEditText.getText().toString();  
 if (!isEmailValid(given\_email) || !isPasswordValid(given\_password)) {  
 mStatusText.setText(mApplicationContext.getString(*status\_login\_bad\_parameters*));  
 setLoginInProgress(false);  
 } else {  
 try {  
 new AsyncLogin().execute(given\_email, given\_password);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 } catch (RemoteException e) {  
 e.printStackTrace();  
 setLoginInProgress(false);  
 output\_parcel.recycle();  
 }  
 }  
  
  
 private boolean isEmailValid(String email) {  
 return email.contains("@");  
 }  
  
 private boolean isPasswordValid(String password) {  
 return (password.length() >= 1);  
 }  
  
 protected void setLoginInProgress(boolean inProgress) {  
 mLoginButton.setEnabled(!inProgress);  
 int visibility;  
 if (inProgress) {  
 visibility = View.*VISIBLE*;  
 }  
 else {  
 visibility = View.*INVISIBLE*;  
 }  
 mProgressBar1.setVisibility(visibility);  
 mProgressBar2.setVisibility(visibility);  
 mProgressBar3.setVisibility(visibility);  
 }  
  
 private class AsyncLogin extends AsyncTask<String, Void, Boolean> {  
 @Override  
 */\*\*  
 String email = strings[0];  
 String password = strings[1];  
 \* \*\*/* protected Boolean doInBackground(String... strings) {  
 if (strings.length != 2) {  
 throw new IllegalArgumentException("Method requires exactly 2 strings");  
 }  
 if (!mMyService.isRunning()) {  
 throw new Error("MyService is ded");  
 }  
 Parcel input\_parcel = Parcel.*obtain*();  
 input\_parcel.writeStringArray(strings);  
 Parcel output\_parcel = Parcel.*obtain*();  
 boolean login\_successful = false;  
 try {  
 mMyService.getBinder().transact(*TRY\_LOGIN*, input\_parcel, output\_parcel, 0);  
 boolean[] val = new boolean[1];  
 output\_parcel.readBooleanArray(val);  
 login\_successful = val[0];  
 if (login\_successful) {  
 SharedPreferences preferences = getSharedPreferences (getString(*user\_data*), Context.*MODE\_PRIVATE*);  
 SharedPreferences.Editor preferences\_editor = preferences.edit();  
 preferences\_editor.putString("email", strings[0]);  
 preferences\_editor.putString("password", strings[1]);  
 preferences\_editor.apply();  
 }  
 } catch (RemoteException e) {  
 e.printStackTrace();  
 }  
 output\_parcel.recycle();  
 return login\_successful;  
 }  
  
 @Override  
 protected void onPostExecute(Boolean logged\_in) {  
 setLoginInProgress(false);  
 if (logged\_in) {  
 Intent mainMenuIntent = new Intent(mLoginActivity, MainMenuActivity.class);  
 mainMenuIntent.setFlags(Intent.*FLAG\_ACTIVITY\_NEW\_TASK*);  
 mApplicationContext.startActivity(mainMenuIntent);  
 finish();  
 }  
 else {  
 mStatusText.setText(getString(*status\_login\_wrong\_parameters*));  
 }  
 }  
 }  
}

**MainMenuActivity.java:**

package com.yanoonigmail.rdroid.app;  
  
import android.app.Activity;  
import android.content.Context;  
import android.content.Intent;  
import android.os.Parcel;  
import android.os.RemoteException;  
import android.support.v7.app.ActionBarActivity;  
import android.os.Bundle;  
import android.view.View;  
  
import com.yanoonigmail.rdroid.ApplicationContext;  
import com.yanoonigmail.rdroid.R;  
  
import static com.yanoonigmail.rdroid.service.TaskManager.*LOGOUT*;  
  
public class MainMenuActivity extends ActionBarActivity {  
 private MyService mMyService = MyService.*getInstance*();  
 private Context mApplicationContext = ApplicationContext.*getContext*();  
 private Activity mMainMenuActivity = this;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main\_menu*);  
 }  
  
 @Override  
 protected void onDestroy() {  
 super.onDestroy();  
 }  
  
 public void logout(View view) {  
 Parcel output\_parcel = Parcel.*obtain*();  
 Parcel input\_parcel = Parcel.*obtain*();  
 try {  
 boolean[] logged\_out = new boolean[1];  
 mMyService.getBinder().transact(*LOGOUT*, input\_parcel, output\_parcel, 0);  
 output\_parcel.readBooleanArray(logged\_out);  
 if (logged\_out[0]) {  
 Intent loginIntent = new Intent(mMainMenuActivity, LoginActivity.class);  
 loginIntent.setFlags(Intent.*FLAG\_ACTIVITY\_NEW\_TASK*);  
 mApplicationContext.startActivity(loginIntent);  
 finish();  
 }  
 } catch (RemoteException e) {  
 e.printStackTrace();  
 }  
 output\_parcel.recycle();  
 }  
  
 public void closeApp(View view) {  
 finish();  
 }  
}

**MyService.java:**

package com.yanoonigmail.rdroid.app;  
  
import android.app.ActivityManager;  
import android.content.ComponentName;  
import android.content.Context;  
import android.content.Intent;  
import android.content.ServiceConnection;  
import android.os.IBinder;  
  
import com.yanoonigmail.rdroid.ApplicationContext;  
import com.yanoonigmail.rdroid.service.TaskManager;  
  
*/\*\*  
 \* Created by 34v7 on 02/03/2016.  
 \*/*public class MyService {  
 private static MyService *ourInstance* = new MyService();  
 private Context context;  
 private Class<?> serviceClass;  
 private boolean bound;  
 private IBinder binder;  
 private Intent serviceIntent;  
 private LocalServiceConnection serviceConnection;  
  
 public static MyService getInstance() {  
 return *ourInstance*;  
 }  
  
 private MyService() {  
 serviceClass = TaskManager.class;  
 context = ApplicationContext.*getContext*();  
 serviceIntent = new Intent(context, serviceClass);  
 serviceConnection = new LocalServiceConnection();  
 assureServiceIsRunning();  
 bound = false;  
 }  
  
 public boolean isRunning() {  
 ActivityManager manager = (ActivityManager) context.getSystemService(Context.*ACTIVITY\_SERVICE*);  
 for (ActivityManager.RunningServiceInfo service : manager.getRunningServices(Integer.*MAX\_VALUE*)) {  
 if (serviceClass.getName().equals(service.service.getClassName())) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 public boolean isBound() {  
 return bound;  
 }  
  
 public void setBound(boolean bound) {  
 this.bound = bound;  
 }  
  
 public IBinder getBinder() {  
 while (true) {  
 if (isBound()) {  
 return binder;  
 } else {  
 assureServiceIsRunning();  
 }  
 }  
 }  
  
 public void setBinder(IBinder binder) {  
 this.binder = binder;  
 }  
  
 public void assureServiceIsRunning() {  
 if (!isRunning()) {  
 context.startService(serviceIntent);  
 }  
 if (!isBound()) {  
 boolean cont = false;  
 while (!cont) {  
 cont = context.bindService(serviceIntent, serviceConnection, Context.*BIND\_ABOVE\_CLIENT*);  
 if (!cont) {  
 try {  
 Thread.*sleep*(5000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
 }  
  
 public void unbindService() {  
 if (isBound()) {  
 ApplicationContext.*getContext*().unbindService(serviceConnection);  
 }  
 }  
  
 private class LocalServiceConnection implements ServiceConnection {  
 @Override  
 public void onServiceConnected(ComponentName name, IBinder service) {  
 setBinder(service);  
 setBound(true);  
 }  
  
 @Override  
 public void onServiceDisconnected(ComponentName name) {  
 setBound(false);  
 assureServiceIsRunning();  
 }  
 }  
}

קבצי השירות (service):

**Encryptor.java:**

package com.yanoonigmail.rdroid.service;  
  
import javax.crypto.Cipher;  
import javax.crypto.KeyGenerator;  
import javax.crypto.SecretKey;  
import javax.crypto.spec.IvParameterSpec;  
import javax.crypto.spec.SecretKeySpec;  
  
import android.util.Base64;  
import android.util.Log;  
  
import java.io.OutputStream;  
import java.security.NoSuchAlgorithmException;  
import java.security.SecureRandom;  
  
*/\*\*  
 \* Created by Yaniv Sharon on 17/02/2016.  
 \*/*public class Encryptor {  
 private byte[] encryptionKey;  
  
 public Encryptor(byte[] encryptionKey)  
 {  
 this.encryptionKey = encryptionKey;  
 }  
  
 public String encrypt(String plainText) throws Exception {  
 Cipher cipher = getCipher(Cipher.*ENCRYPT\_MODE*);  
 byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());  
 return Base64.*encodeToString*(encryptedBytes, Base64.*DEFAULT*);  
 }  
  
 public String decrypt(String encrypted) throws Exception {  
 Cipher cipher = getCipher(Cipher.*DECRYPT\_MODE*);  
 byte[] encryptedBytes = Base64.*decode*(encrypted, Base64.*DEFAULT*);  
 byte[] decryptedBytes = cipher.doFinal(encryptedBytes);  
 return new String(decryptedBytes);  
 }  
  
 public byte[] getEncryptionKey() {  
 return this.encryptionKey;  
 }  
  
 public Cipher getCipher(int cipherMode) throws Exception {  
 String encryptionAlgorithm = "AES";  
 SecretKeySpec keySpecification = new SecretKeySpec(  
 encryptionKey, encryptionAlgorithm);  
 Cipher cipher = Cipher.*getInstance*(encryptionAlgorithm);  
 cipher.init(cipherMode, keySpecification);  
 return cipher;  
 }  
}

**EncryptorFactory.java:**

package com.yanoonigmail.rdroid.service;  
  
import android.util.Base64;  
  
import java.io.IOException;  
import java.security.InvalidKeyException;  
import java.security.KeyFactory;  
import java.security.NoSuchAlgorithmException;  
import java.security.PublicKey;  
import java.security.spec.InvalidKeySpecException;  
import java.security.spec.X509EncodedKeySpec;  
import java.util.Arrays;  
  
import javax.crypto.BadPaddingException;  
import javax.crypto.Cipher;  
import javax.crypto.IllegalBlockSizeException;  
import javax.crypto.KeyGenerator;  
import javax.crypto.NoSuchPaddingException;  
import javax.crypto.SecretKey;  
  
*/\*\*  
 \* Created by Yaniv Sharon on 17/02/2016.  
 \*/*public class EncryptorFactory {  
 private Server server = Server.*getInstance*();  
 public EncryptorFactory() {  
 }  
  
 public Encryptor createEncryptor() throws NoSuchPaddingException, NoSuchAlgorithmException, InvalidKeySpecException, InvalidKeyException, BadPaddingException, IllegalBlockSizeException, IOException {  
 byte[] keyBytes = server.unencryptedRecv();  
 X509EncodedKeySpec keySpec = new X509EncodedKeySpec(keyBytes);  
 PublicKey key = KeyFactory.*getInstance*("RSA").generatePublic(keySpec);  
 Cipher cipher = Cipher.*getInstance*("RSA/ECB/PKCS1Padding");  
 cipher.init(Cipher.*ENCRYPT\_MODE*, key);  
 KeyGenerator keyGen = KeyGenerator.*getInstance*("AES");  
 keyGen.init(256);  
 SecretKey secretKey = keyGen.generateKey();  
 server.unencryptedSend(cipher.doFinal(secretKey.getEncoded()));  
 return new Encryptor(secretKey.getEncoded());  
 }  
}

**OSInterface.java:**

package com.yanoonigmail.rdroid.service;  
  
import android.content.ContentProviderOperation;  
import android.content.ContentResolver;  
import android.content.OperationApplicationException;  
import android.database.Cursor;  
import android.os.RemoteException;  
import android.provider.ContactsContract;  
import android.widget.Toast;  
  
import com.yanoonigmail.rdroid.ApplicationContext;  
  
import java.io.BufferedReader;  
import java.io.DataInputStream;  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.nio.charset.StandardCharsets;  
import java.util.ArrayList;  
import java.util.Arrays;  
  
*/\*\*  
 \* Created by yanoo on 19-Apr-16.  
 \*/*public class OSInterface {  
  
 public static void saveFile(File file, String fileData) {  
 String[] data = fileData.split(System.*getProperty*("line.separator"));  
 FileOutputStream fos = null;  
 try  
 {  
 fos = new FileOutputStream(file);  
 }  
 catch (FileNotFoundException e) {e.printStackTrace();}  
 if (fos != null){  
 try {  
 try {  
 for (int i = 0; i < data.length; i++) {  
 fos.write(data[i].getBytes());  
 if (i < data.length - 1) {  
 fos.write("\n".getBytes());  
 }  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 } finally {  
 try {  
 fos.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
  
 public static void saveContact(String DisplayName, String MobileNumber, String HomeNumber,  
 String WorkNumber, String emailID, String company,  
 String jobTitle) throws OperationApplicationException, RemoteException {  
 ArrayList< ContentProviderOperation > ops = new ArrayList < ContentProviderOperation > ();  
  
 ops.add(ContentProviderOperation.*newInsert*(  
 ContactsContract.RawContacts.*CONTENT\_URI*)  
 .withValue(ContactsContract.RawContacts.*ACCOUNT\_TYPE*, null)  
 .withValue(ContactsContract.RawContacts.*ACCOUNT\_NAME*, null)  
 .build());  
  
 //------------------------------------------------------ Names  
 if (DisplayName != null) {  
 ops.add(ContentProviderOperation.*newInsert*(  
 ContactsContract.Data.*CONTENT\_URI*)  
 .withValueBackReference(ContactsContract.Data.*RAW\_CONTACT\_ID*, 0)  
 .withValue(ContactsContract.Data.*MIMETYPE*,  
 ContactsContract.CommonDataKinds.StructuredName.*CONTENT\_ITEM\_TYPE*)  
 .withValue(  
 ContactsContract.CommonDataKinds.StructuredName.*DISPLAY\_NAME*,  
 DisplayName).build());  
 }  
  
 //------------------------------------------------------ Mobile Number  
 if (MobileNumber != null) {  
 ops.add(ContentProviderOperation.  
 *newInsert*(ContactsContract.Data.*CONTENT\_URI*)  
 .withValueBackReference(ContactsContract.Data.*RAW\_CONTACT\_ID*, 0)  
 .withValue(ContactsContract.Data.*MIMETYPE*,  
 ContactsContract.CommonDataKinds.Phone.*CONTENT\_ITEM\_TYPE*)  
 .withValue(ContactsContract.CommonDataKinds.Phone.*NUMBER*, MobileNumber)  
 .withValue(ContactsContract.CommonDataKinds.Phone.*TYPE*,  
 ContactsContract.CommonDataKinds.Phone.*TYPE\_MOBILE*)  
 .build());  
 }  
  
 //------------------------------------------------------ Home Numbers  
 if (HomeNumber != null) {  
 ops.add(ContentProviderOperation.*newInsert*(ContactsContract.Data.*CONTENT\_URI*)  
 .withValueBackReference(ContactsContract.Data.*RAW\_CONTACT\_ID*, 0)  
 .withValue(ContactsContract.Data.*MIMETYPE*,  
 ContactsContract.CommonDataKinds.Phone.*CONTENT\_ITEM\_TYPE*)  
 .withValue(ContactsContract.CommonDataKinds.Phone.*NUMBER*, HomeNumber)  
 .withValue(ContactsContract.CommonDataKinds.Phone.*TYPE*,  
 ContactsContract.CommonDataKinds.Phone.*TYPE\_HOME*)  
 .build());  
 }  
  
 //------------------------------------------------------ Work Numbers  
 if (WorkNumber != null) {  
 ops.add(ContentProviderOperation.*newInsert*(ContactsContract.Data.*CONTENT\_URI*)  
 .withValueBackReference(ContactsContract.Data.*RAW\_CONTACT\_ID*, 0)  
 .withValue(ContactsContract.Data.*MIMETYPE*,  
 ContactsContract.CommonDataKinds.Phone.*CONTENT\_ITEM\_TYPE*)  
 .withValue(ContactsContract.CommonDataKinds.Phone.*NUMBER*, WorkNumber)  
 .withValue(ContactsContract.CommonDataKinds.Phone.*TYPE*,  
 ContactsContract.CommonDataKinds.Phone.*TYPE\_WORK*)  
 .build());  
 }  
  
 //------------------------------------------------------ Email  
 if (emailID != null) {  
 ops.add(ContentProviderOperation.*newInsert*(ContactsContract.Data.*CONTENT\_URI*)  
 .withValueBackReference(ContactsContract.Data.*RAW\_CONTACT\_ID*, 0)  
 .withValue(ContactsContract.Data.*MIMETYPE*,  
 ContactsContract.CommonDataKinds.Email.*CONTENT\_ITEM\_TYPE*)  
 .withValue(ContactsContract.CommonDataKinds.Email.*DATA*, emailID)  
 .withValue(ContactsContract.CommonDataKinds.Email.*TYPE*, ContactsContract.CommonDataKinds.Email.*TYPE\_WORK*)  
 .build());  
 }  
  
 //------------------------------------------------------ Organization  
 if (!company.equals("") && !jobTitle.equals("")) {  
 ops.add(ContentProviderOperation.*newInsert*(ContactsContract.Data.*CONTENT\_URI*)  
 .withValueBackReference(ContactsContract.Data.*RAW\_CONTACT\_ID*, 0)  
 .withValue(ContactsContract.Data.*MIMETYPE*,  
 ContactsContract.CommonDataKinds.Organization.*CONTENT\_ITEM\_TYPE*)  
 .withValue(ContactsContract.CommonDataKinds.Organization.*COMPANY*, company)  
 .withValue(ContactsContract.CommonDataKinds.Organization.*TYPE*, ContactsContract.CommonDataKinds.Organization.*TYPE\_WORK*)  
 .withValue(ContactsContract.CommonDataKinds.Organization.*TITLE*, jobTitle)  
 .withValue(ContactsContract.CommonDataKinds.Organization.*TYPE*, ContactsContract.CommonDataKinds.Organization.*TYPE\_WORK*)  
 .build());  
 }  
  
 // Asking the Contact provider to create a new contact  
 ApplicationContext.*getContext*().getContentResolver().applyBatch(ContactsContract.*AUTHORITY*, ops);  
 }  
  
 public static String[][] getContacts() {  
 ArrayList<String[]> outputList = new ArrayList<>();  
 ContentResolver cr = ApplicationContext.*getContext*().getContentResolver();  
 Cursor cur = cr.query(ContactsContract.Contacts.*CONTENT\_URI*,  
 null, null, null, null);  
  
 if (cur != null) {  
 if (cur.getCount() > 0) {  
 while (cur.moveToNext()) {  
 String id = cur.getString(  
 cur.getColumnIndex(ContactsContract.Contacts.*\_ID*));  
 String name = cur.getString(cur.getColumnIndex(  
 ContactsContract.Contacts.*DISPLAY\_NAME*));  
  
 if (Integer.*parseInt*(cur.getString(cur.getColumnIndex(  
 ContactsContract.Contacts.*HAS\_PHONE\_NUMBER*))) > 0) {  
 Cursor pCur = cr.query(  
 ContactsContract.CommonDataKinds.Phone.*CONTENT\_URI*,  
 null,  
 ContactsContract.CommonDataKinds.Phone.*CONTACT\_ID* + " = ?",  
 new String[]{id}, null);  
 if (pCur != null) {  
 while (pCur.moveToNext()) {  
 String phoneNumber = pCur.getString(pCur.getColumnIndex(  
 ContactsContract.CommonDataKinds.Phone.*NUMBER*));  
 String email = pCur.getString(pCur.getColumnIndex(  
 ContactsContract.CommonDataKinds.Email.*ADDRESS*));  
 Toast.*makeText*(ApplicationContext.*getContext*(), "Name: " + name  
 + ", Phone No: " + phoneNumber, Toast.*LENGTH\_SHORT*).show();  
 String[] contactArray = new String[3];  
 contactArray[0] = name;  
 contactArray[1] = phoneNumber;  
 contactArray[2] = email;  
 outputList.add(contactArray);  
 // *todo* }  
 pCur.close();  
 }  
 }  
 }  
 }  
 cur.close();  
 }  
 String[][] output = new String[outputList.size()][];  
 for (int i = 0; i < outputList.size(); i++) {  
 output[i] = outputList.get(i);  
 }  
 return output;  
 }  
}

**Protocol.java:**

package com.yanoonigmail.rdroid.service;  
  
import android.content.res.Resources;  
  
import com.yanoonigmail.rdroid.R;  
import com.yanoonigmail.rdroid.ApplicationContext;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
  
import static com.yanoonigmail.rdroid.R.string.*protocol\_parameter\_separator*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_client\_header*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_client\_login\_announcement*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_client\_login\_password*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_client\_login\_email*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_client\_task\_results\_announcement*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_client\_task\_results\_id*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_client\_task\_results\_output*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_server\_task\_request\_parameters\_separator*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_server\_task\_request\_announcement*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_server\_task\_request\_id*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_server\_header*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_server\_login\_announcement*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_server\_login\_bool*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_server\_login\_success*;  
  
*/\*\*  
 \* Created by Yaniv on 27-Feb-16.  
 \*/*public class Protocol {  
 private static String *line\_separator* = System.*getProperty*("line.separator");  
 private static Resources *resources* = ApplicationContext.*getContext*().getResources();  
  
 public static String[] cutMessageLen(String message){  
 return message.split(*resources*.getString(*protocol\_parameter\_separator*), 2);  
 }  
  
 public static String addMessageLen(String message) {  
 return String.*valueOf*(message.length()) + *resources*.getString(*protocol\_parameter\_separator*)  
 + message;  
 }  
  
 public static boolean isServerMessage(String message) {  
 return message.startsWith(*resources*.getString(*protocol\_server\_header*));  
 }  
  
 public static String loginRequest(String email, String password) {  
 String message;  
 message = *resources*.getString(*protocol\_client\_header*) +  
 *line\_separator*;  
 message += *resources*.getString(*protocol\_client\_login\_announcement*) +  
 *line\_separator*;  
 message += *resources*.getString(*protocol\_client\_login\_email*) +  
 *resources*.getString(*protocol\_parameter\_separator*) +  
 email +  
 *line\_separator*;  
 message += *resources*.getString(*protocol\_client\_login\_password*) +  
 *resources*.getString(*protocol\_parameter\_separator*) +  
 password +  
 *line\_separator*;  
 return message;  
 }  
  
 public static boolean loginResponseBool(String login\_response) {  
 if (!*isServerMessage*(login\_response))  
 return false;  
 String[] lineArray = login\_response.split(*line\_separator*);  
 if (lineArray.length != 3)  
 return false;  
 if (!lineArray[1].equals(*resources*.getString(*protocol\_server\_login\_announcement*)))  
 return false;  
 if (!lineArray[2].startsWith(*resources*.getString(*protocol\_server\_login\_bool*) +  
 *resources*.getString(*protocol\_parameter\_separator*)))  
 return false;  
 String[] line3\_array;  
 line3\_array = lineArray[2].split(*resources*.getString(*protocol\_parameter\_separator*));  
 return line3\_array[1].equals(*resources*.getString(*protocol\_server\_login\_success*));  
 }  
  
 */\*\*  
 \* Turns the request packet from the server into a Task object.  
 \** ***@param*** *message The request packet that was received.  
 \** ***@return*** *The given Task object.  
 \*/* public static Task taskRequest(String message) throws BadInputException {  
 if (*isServerMessage*(message)) {  
 String[] lineArray = message.split(*line\_separator*, 5);  
 if (lineArray[1].equals(*resources*.getString(*protocol\_server\_task\_request\_announcement*))) {  
 String[] parameterLineArray = Arrays.*copyOfRange*(lineArray, 2, lineArray.length);  
 if (parameterLineArray.length == 3) {  
 return *taskCreator*(parameterLineArray);  
 }  
 }  
 }  
 throw new BadInputException("Something is wrong with the message");  
 }  
  
  
 */\*\*  
 \* Made for taskRequest.  
 \** ***@param*** *taskParamsArray The array of strings that describe the task object.  
 \** ***@return*** *A task object.  
 \*/* protected static Task taskCreator(String[] taskParamsArray) throws BadInputException {  
 String id = "";  
 String type = "";  
 String parameters = "";  
 for (String param : taskParamsArray) {  
 String[] paramDetailsCouple = param.split(*resources*.getString(*protocol\_parameter\_separator*));  
 if (paramDetailsCouple.length == 1)  
 {  
 String[] newParamDetailsCouple = new String[2];  
 newParamDetailsCouple[0] = paramDetailsCouple[0];  
 newParamDetailsCouple[1] = "";  
 paramDetailsCouple = newParamDetailsCouple;  
 }  
 switch (paramDetailsCouple[0]) {  
 case "id":  
 id = paramDetailsCouple[1];  
 break;  
 case "type":  
 type = paramDetailsCouple[1];  
 break;  
 case "parameters":  
 parameters = paramDetailsCouple[1];  
 break;  
 }  
 }  
 if (id.equals("") || type.equals("")) {  
 throw new BadInputException("Missing task parameters.");  
 }  
 return new Task(id, type, parameters);  
 }  
  
 public static String taskResultsMessage(String id, String output) {  
 String outputString = *resources*.getString(*protocol\_client\_header*) +  
 *line\_separator*;  
 outputString += *resources*.getString(*protocol\_client\_task\_results\_announcement*) +  
 *line\_separator*;  
 outputString += *resources*.getString(*protocol\_client\_task\_results\_id*) +  
 *resources*.getString(*protocol\_parameter\_separator*) +  
 id +  
 *line\_separator*;  
 outputString += *resources*.getString(*protocol\_client\_task\_results\_output*) +  
 *resources*.getString(*protocol\_parameter\_separator*) +  
 output +  
 *line\_separator*;  
 return outputString;  
 }  
  
 public static int getDownloadFileStartIndex(String message) {  
 if (*isServerMessage*(message)) {  
 String[] lineArray = message.split(*line\_separator*, 5);  
 if (lineArray[1].equals(*resources*.getString(*protocol\_server\_task\_request\_announcement*)) && lineArray[3].equals("type:SAVE\_FILE")) {  
 String[] parameterLineArray = Arrays.*copyOfRange*(lineArray, 2, lineArray.length);  
 if (parameterLineArray.length == 3) {  
 int startIndex = parameterLineArray[2].indexOf(",");  
 if (startIndex == -1) {  
 return -1;  
 }  
 else {  
 return startIndex + 1;  
 }  
 }  
 }  
 }  
 return -1; // -1 signals that the message is not a download task.  
 }  
  
 public static String getDownloadFileLocation(String message) {  
 if (*isServerMessage*(message)) {  
 String[] lineArray = message.split(*line\_separator*, 5);  
 if (lineArray[1].equals(*resources*.getString(*protocol\_server\_task\_request\_announcement*)) && lineArray[3].equals("type:SAVE\_FILE")) {  
 String[] parameterLineArray = Arrays.*copyOfRange*(lineArray, 2, lineArray.length);  
 if (parameterLineArray.length == 3) {  
 String parameters = parameterLineArray[2];  
 return parameters.substring(parameters.indexOf(":"), parameters.indexOf(","));  
 }  
 }  
 }  
 return null;  
 }  
  
 static class BadInputException extends Exception {  
 public BadInputException(String message) {  
 super(message);  
 }  
 }  
}

**Server.java:**

package com.yanoonigmail.rdroid.service;  
  
import android.content.Context;  
import android.content.SharedPreferences;  
import android.util.Base64OutputStream;  
import android.util.Log;  
import android.content.res.Resources;  
  
import com.yanoonigmail.rdroid.ApplicationContext;  
import com.yanoonigmail.rdroid.R;  
  
import java.io.BufferedInputStream;  
import java.io.BufferedOutputStream;  
import java.io.BufferedReader;  
import java.io.BufferedWriter;  
import java.io.ByteArrayOutputStream;  
import java.io.DataOutputStream;  
import java.io.File;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.OutputStreamWriter;  
import java.io.PrintWriter;  
import java.net.Socket;  
import java.net.InetSocketAddress;  
import java.lang.Thread;  
import java.util.Arrays;  
import java.util.concurrent.locks.ReentrantLock;  
import android.util.Base64;  
  
import static com.yanoonigmail.rdroid.R.string.*server\_address*;  
import static com.yanoonigmail.rdroid.R.string.*server\_port*;  
import static com.yanoonigmail.rdroid.R.string.*user\_data*;  
  
  
*/\*\*  
 \* Created by Yaniv Sharon on 17/02/2016.  
 \*/*public class Server {  
 private static Server *ourInstance* = new Server();  
 protected Context context = ApplicationContext.*getContext*();  
 protected boolean mInitialized = false;  
 protected Socket mServerSocket;  
 protected InetSocketAddress mServerAddress;  
 protected EncryptorFactory mEncryptorFactory;  
 protected Encryptor mEncryptor;  
 protected boolean mConnected = false;  
 protected boolean mLoggedIn = false;  
 protected Thread mInitThread;  
 protected ReentrantLock mConnectLock = new ReentrantLock();  
 protected ReentrantLock mLoginLock = new ReentrantLock();  
 protected String mEmail;  
 protected String mPassword;  
 protected static Resources *resources* = ApplicationContext.*getContext*().getResources();  
 protected boolean mFirstLoginTry = true;  
  
 public static Server getInstance() {  
 return *ourInstance*;  
 }  
  
 protected Server() {  
 mInitThread = new Thread(new Runnable() {  
 public void run() {  
 String ip = *resources*.getString(*server\_address*);  
 String port = *resources*.getString(*server\_port*);  
 mServerAddress = new InetSocketAddress(ip, Integer.*parseInt*(port));  
 Log.*d*("Server init", mServerAddress.toString());  
 mEncryptorFactory = new EncryptorFactory();  
 mInitialized = true;  
 connect();  
 }  
 });  
 mInitThread.start();  
 }  
  
 */\*\*  
 \* Connects to the server.  
 \*/* public void connect() {  
 Thread manageTasksThread = new Thread(new Runnable() {  
 public void run() {  
 if (mConnectLock.tryLock()) {  
 while (!mInitialized) {  
 try {  
 Thread.*sleep*(10);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 if (!mConnected) {  
 mLoggedIn = false;  
 }  
 while (!mConnected) {  
 try {  
 if (mServerSocket != null) {  
 mServerSocket.close();  
 }  
 } catch (IOException e2) {  
 e2.printStackTrace();  
 }  
 try {  
 mServerSocket = new Socket();  
 mServerSocket.connect(mServerAddress, 5000);  
 mConnected = true;  
 } catch (IOException e) {  
 e.printStackTrace();  
 try {  
 Thread.*sleep*(5000);  
 } catch (InterruptedException e2) {  
 e2.printStackTrace();  
 }  
 }  
 }  
 try {  
 mEncryptor = mEncryptorFactory.createEncryptor();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 mConnectLock.unlock();  
 passiveLogin();  
 }  
 }  
 });  
 manageTasksThread.start();  
 }  
  
 protected void passiveLogin() {  
 if (!isLoggedIn() && isConnected()) {  
 SharedPreferences preferences = context.getSharedPreferences(*resources*.getString(*user\_data*), Context.*MODE\_PRIVATE*);  
 if (preferences.contains("email") && preferences.contains("password")) {  
 mEmail = preferences.getString("email", "");  
 mPassword = preferences.getString("password", "");  
 tryLogin(mEmail, mPassword);  
 } else {  
 mFirstLoginTry = false;  
 SharedPreferences.OnSharedPreferenceChangeListener changeListener = new SharedPreferences.OnSharedPreferenceChangeListener() {  
 @Override  
 public void onSharedPreferenceChanged(SharedPreferences sharedPreferences, String key) {  
 passiveLogin();  
 }  
 };  
 preferences.registerOnSharedPreferenceChangeListener(changeListener);  
 }  
 }  
 }  
  
 public boolean send(String message) {  
 String encrypted\_message;  
 try {  
 encrypted\_message = mEncryptor.encrypt(message);  
 encrypted\_message = Protocol.*addMessageLen*(encrypted\_message);  
 } catch (Exception e) {  
 e.printStackTrace();  
 return false;  
 }  
 try {  
 rawSend(encrypted\_message);  
 } catch (IOException e) {  
 mConnected = false;  
 connect();  
 e.printStackTrace();  
 return false;  
 }  
 return true;  
 }  
  
 public boolean unencryptedSend(byte[] message\_bytes){  
 try {  
 String message = Base64.*encodeToString*(message\_bytes, Base64.*DEFAULT*);  
 message = Protocol.*addMessageLen*(message);  
 rawSend(message);  
 } catch (Exception e) {  
 return false;  
 }  
 return true;  
 }  
  
 public void rawSend(String message) throws IOException{  
 while (!mServerSocket.isConnected()) {  
 if (!mConnectLock.isLocked()) {  
 connect();  
 }  
 else {  
 try {  
 Thread.*sleep*(100);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 PrintWriter output\_stream =  
 new PrintWriter(  
 new BufferedWriter(  
 new OutputStreamWriter(mServerSocket.getOutputStream())), true);  
 output\_stream.println(message);  
 }  
  
 public String recv() {  
 while (!isConnected()) {  
 if (!mConnectLock.isLocked()) {  
 connect();  
 } else {  
 try {  
 Thread.*sleep*(100);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 try {  
 InputStream sis = mServerSocket.getInputStream();  
 BufferedReader br = new BufferedReader(new InputStreamReader(sis));  
 long messageLen = 0;  
 boolean cont = false;  
 char[] oneCharBuffer = new char[1];  
 while (!cont) {  
 if (br.read(oneCharBuffer) != 1) {  
 return "";  
 }  
 char oneChar = oneCharBuffer[0];  
 if (oneChar == ':') {  
 cont = true;  
 } else {  
 double digit = (double) Character.*getNumericValue*(oneChar);  
 if (digit < 0 || digit > 9) {  
 return "";  
 }  
 messageLen = (long) (messageLen \* 10 + digit);  
 }  
 }  
 char[] streamBuffer = new char[(int) Math.*min*(8192, messageLen)];  
 int totalReadLen = 0;  
 int readLen = br.read(streamBuffer);  
 totalReadLen += readLen;  
 String messageStart = mEncryptor.decrypt(new String(streamBuffer));  
 String fileLocation = Protocol.*getDownloadFileLocation*(messageStart);  
 if (fileLocation != null) {  
 int startIndex = Protocol.*getDownloadFileStartIndex*(messageStart);  
 if (startIndex != -1) {  
 File file = new File(fileLocation);  
 FileOutputStream fos = new FileOutputStream(file);  
 fos.write(new String(streamBuffer).getBytes("UTF-8"), startIndex, readLen - startIndex);  
 fos.flush();  
 cont = false;  
 while (!cont) {  
 streamBuffer = new char[(int) Math.*min*(8192, messageLen - totalReadLen)];  
 readLen = br.read(streamBuffer);  
 totalReadLen += readLen;  
 if (readLen != -1) {  
 fos.write(mEncryptor.decrypt(new String(streamBuffer)).getBytes("UTF-8"));  
 fos.flush();  
 } else {  
 cont = true;  
 }  
 if (readLen != streamBuffer.length || totalReadLen == messageLen) {  
 cont = true;  
 }  
 }  
 fos.close();  
 return "";  
 }  
 }  
 StringBuilder stringBuilder = new StringBuilder(messageStart);  
 streamBuffer = new char[(int) Math.*min*(8192, messageLen - totalReadLen)];  
 cont = false;  
 while (!cont && totalReadLen < messageLen) {  
 readLen = br.read(streamBuffer);  
 totalReadLen += readLen;  
 if (readLen == -1) {  
 cont = true;  
 }  
 stringBuilder.append(mEncryptor.decrypt(new String(streamBuffer)));  
 }  
 return stringBuilder.toString();  
 } catch (Exception e) {  
 mConnected = false;  
 connect();  
 e.printStackTrace();  
 return "";  
 }  
 }  
  
 public String rawRecv() throws IOException {  
 BufferedReader input\_stream = new BufferedReader(new InputStreamReader(this.mServerSocket.getInputStream()));  
 return (input\_stream.readLine());  
 }  
  
 public byte[] unencryptedRecv() {  
 try {  
 BufferedReader input\_stream = new BufferedReader(new InputStreamReader(this.mServerSocket.getInputStream()));  
 String message = input\_stream.readLine();  
 String[] lenAndMessageArray = Protocol.*cutMessageLen*(message);  
 int len = Integer.*parseInt*(lenAndMessageArray[0]);  
 message = lenAndMessageArray[1];  
 while (message.length() < len) {  
 message += input\_stream.readLine();  
 }  
 return Base64.*decode*(message, Base64.*DEFAULT*);  
 } catch (IOException e) {  
 mConnected = false;  
 connect();  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 public void streamSend(InputStream stream, long streamLength, byte[] preStreamData) {  
 try {  
 BufferedOutputStream bos = new BufferedOutputStream(new DataOutputStream(mServerSocket.getOutputStream()));  
 Base64OutputStream b64os = new Base64OutputStream(new DataOutputStream(mServerSocket.getOutputStream()), Base64.*NO\_CLOSE* | Base64.*NO\_WRAP*);  
 BufferedInputStream bis = new BufferedInputStream(stream);  
 long totalStreamLength = streamLength + preStreamData.length; // pure  
 totalStreamLength = (long) (4 \* Math.*ceil*((double) (totalStreamLength / 3))); // after base64  
 bos.write((String.*valueOf*(totalStreamLength) + ":").getBytes("UTF-8"));  
 bos.flush();  
 b64os.write(preStreamData);  
 b64os.flush();  
 boolean again = true;  
 while (again) {  
 byte[] buffer = new byte[8192];  
 int readLen = bis.read(buffer);  
 if (readLen != -1) {  
 b64os.write(buffer, 0, readLen);  
 b64os.flush();  
 } else {  
 again = false;  
 }  
 }  
 b64os.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 mConnected = false;  
 connect();  
 }  
 }  
  
 public boolean tryLogin(String email, String password) {  
 mLoginLock.lock();  
 mEmail = email;  
 mPassword = password;  
 if (!isConnected()) {  
 mConnected = false;  
 connect();  
 }  
 if (isConnected() && !isLoggedIn() && !mEmail.equals("") && !mPassword.equals("")) {  
 String login\_request = Protocol.*loginRequest*(mEmail, mPassword);  
 boolean sent = send(login\_request);  
 if (!sent) {  
 mFirstLoginTry = false;  
 return false;  
 }  
 String login\_response = recv();  
 mLoggedIn = Protocol.*loginResponseBool*(login\_response);  
 }  
 mFirstLoginTry = false;  
 mLoginLock.unlock();  
 return mLoggedIn;  
 }  
  
 public boolean isConnected() {  
 return mConnected;  
 }  
  
 public boolean isLoggedIn() {  
 return mLoggedIn;  
 }  
  
 public boolean isLoggedInAfterFirstTry() {  
 waitForFirstLogin();  
 return isLoggedIn();  
 }  
  
 protected void waitForFirstLogin() {  
 while (mFirstLoginTry) {  
 try {  
 Thread.*sleep*(10);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public boolean disconnect() {  
 try {  
 mServerSocket.close();  
 mConnected = false;  
 mLoggedIn = false;  
 mFirstLoginTry = true;  
 connect();  
 return true;  
 } catch (IOException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
  
 protected byte[] appendData(byte[] firstObject,byte[] secondObject){  
 ByteArrayOutputStream outputStream = new ByteArrayOutputStream( );  
 try {  
 if (firstObject!=null && firstObject.length!=0)  
 outputStream.write(firstObject);  
 if (secondObject!=null && secondObject.length!=0)  
 outputStream.write(secondObject);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return outputStream.toByteArray();  
 }  
}

**StartupReciever.java:**

package com.yanoonigmail.rdroid.service;  
  
import android.content.BroadcastReceiver;  
import android.content.Context;  
import android.content.Intent;  
  
*/\*\*  
 \* Created by yanoo on 24-Feb-16.  
 \*/*public class StartupReceiver extends BroadcastReceiver {  
  
 public StartupReceiver() {  
 }  
  
 @Override  
 public void onReceive(Context context, Intent intent) {  
 Intent startIntent = new Intent(context, TaskManager.class);  
 context.startService(startIntent);  
 }  
}

**Task.java:**

package com.yanoonigmail.rdroid.service;  
  
import android.content.OperationApplicationException;  
import android.content.res.Resources;  
import android.os.Environment;  
import android.os.RemoteException;  
import android.util.Log;  
  
import com.yanoonigmail.rdroid.ApplicationContext;  
import com.yanoonigmail.rdroid.R;  
  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.net.Socket;  
import java.net.UnknownHostException;  
import java.util.Arrays;  
import java.util.logging.SocketHandler;  
  
import static com.yanoonigmail.rdroid.R.string.*protocol\_client\_task\_results\_output\_separator*;  
import static com.yanoonigmail.rdroid.R.string.*protocol\_parameter\_separator*;  
import static com.yanoonigmail.rdroid.R.string.*server\_address*;  
  
*/\*\*  
 \* Created by yanoo on 19-Mar-16.  
 \*/*public class Task {  
 private String id;  
 private String type;  
 private String parameters;  
 private Thread taskThread;  
 private final static Resources *resources* = ApplicationContext.*getContext*().getResources();  
 private final static Server *server* = Server.*getInstance*();  
  
 public Task(String id, String type, String parameters) {  
 this.id = id;  
 this.type = type;  
 this.parameters = parameters;  
 initTask();  
 }  
  
 public String getId() {  
 return id;  
 }  
  
 public String getType() {  
 return type;  
 }  
  
 public Thread getTaskThread() {  
 return taskThread;  
 }  
  
 private void initTask() {  
 taskThread = new Thread(new Runnable() {  
 public void run() {  
 String output = "";  
 boolean outputExists = true;  
 switch (type) {  
 case "GET\_FILES\_IN\_FOLDER":  
 output = getFilesInFolder(parameters);  
 break;  
 case "GET\_FILE":  
 getFile(parameters);  
 outputExists = false;  
 break;  
 case "SAVE\_FILE":  
 String[] saveFileInputArray = parameters.split(*resources*.getString(*protocol\_parameter\_separator*), 1);  
 output = saveFile(saveFileInputArray[0], saveFileInputArray[1]);  
 break;  
 case "GET\_CONTACTS":  
 output = getContacts();  
 break;  
 case "SAVE\_CONTACT":  
 String[] saveContactInputArray = parameters.split(*resources*.getString(*protocol\_parameter\_separator*));  
 if (saveContactInputArray.length == 3) {  
 output = saveContact(saveContactInputArray[0], saveContactInputArray[1], saveContactInputArray[2]);  
 } else {  
 output = "failure";  
 }  
 break;  
 }  
 if (outputExists) {  
 *server*.send(Protocol.*taskResultsMessage*(id, output));  
 }  
 }  
 });  
 taskThread.start();  
 }  
  
 private String getFilesInFolder(String folderLocation) {  
 String filesString = "";  
 File sdCardRoot = Environment.*getExternalStorageDirectory*();  
 File yourDir = new File(sdCardRoot, folderLocation);  
 try {  
 for (File f : yourDir.listFiles()) {  
 String name;  
 if (f.isFile()) {  
 name = f.getName();  
 Log.*i*("file name", name);  
 filesString += name + *resources*.getString(*protocol\_client\_task\_results\_output\_separator*);  
 } else {  
 if (f.isDirectory()) {  
 name = f.getName() + ":folder";  
 filesString += name + *resources*.getString(*protocol\_client\_task\_results\_output\_separator*);  
 }  
 }  
 }  
 if (!filesString.equals("")) {  
 filesString = filesString.substring(0, filesString.length() - 1);  
 }  
 } catch (NullPointerException e) {  
 e.printStackTrace();  
 filesString = "";  
 }  
 return filesString;  
 }  
  
 private void getFile(String location) {  
 try {  
 InputStream is;  
 String output;  
 File sdCardRoot = Environment.*getExternalStorageDirectory*();  
 File theFile = new File(sdCardRoot, location);  
 if (!theFile.isFile()) {  
 output = Protocol.*taskResultsMessage*(this.getId(), "failure" +  
 *resources*.getString(*protocol\_client\_task\_results\_output\_separator*) +  
 "File not found");  
 *server*.send(output);  
 } else {  
 if (!theFile.canRead()) {  
 output = Protocol.*taskResultsMessage*(this.getId(), "failure" +  
 *resources*.getString(*protocol\_client\_task\_results\_output\_separator*) +  
 "Cannot read the file");  
 *server*.send(output);  
 } else {  
 output = Protocol.*taskResultsMessage*(this.getId(), "success" +  
 *resources*.getString(*protocol\_client\_task\_results\_output\_separator*));  
 output = output.substring(0, output.length() - 1);  
 is = new FileInputStream(theFile);  
 long fileSize = theFile.length();  
 byte[] preStreamData = output.getBytes();  
 *server*.streamSend(is, fileSize, preStreamData);  
 is.close();  
 }  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 private String saveFile(String fileLocation, String fileData) {  
 String output;  
 try {  
 File sdCardRoot = Environment.*getExternalStorageDirectory*();  
 File theFile = new File(sdCardRoot, fileLocation);  
 OSInterface.*saveFile*(theFile, fileData);  
 output = "success";  
 } catch (Exception e) {  
 e.printStackTrace();  
 output = "failure";  
 }  
 return output;  
 }  
  
 private String getContacts() {  
 String output = "";  
 String[][] contactsInfo = OSInterface.*getContacts*();  
 for (String[] contactInfo: contactsInfo) {  
 if (!output.equals("")) {  
 output += *resources*.getString(*protocol\_client\_task\_results\_output\_separator*);  
 }  
 for (String info : contactInfo) {  
 if (output.charAt(output.length() - 1) == ',') {  
 output += ":";  
 }  
 output += info;  
 }  
 }  
 return output;  
 }  
  
 private String saveContact(String DisplayName, String MobileNumber, String emailID) {  
 try{  
 OSInterface.*saveContact*(DisplayName, MobileNumber, null, null, emailID, null, null);  
 return "success";  
 } catch (OperationApplicationException | RemoteException e) {  
 e.printStackTrace();  
 return "failure";  
 }  
 }  
  
}

**TaskManager.java:**

package com.yanoonigmail.rdroid.service;  
  
import android.app.Notification;  
import android.app.NotificationManager;  
import android.app.PendingIntent;  
import android.app.TaskStackBuilder;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.graphics.BitmapFactory;  
import android.os.Binder;  
import android.os.IBinder;  
import android.content.Context;  
import android.os.Parcel;  
import android.support.v4.app.NotificationCompat;  
import android.util.Log;  
  
import com.yanoonigmail.rdroid.ApplicationContext;  
import com.yanoonigmail.rdroid.R;  
import com.yanoonigmail.rdroid.app.LoadingActivity;  
  
import java.util.ArrayList;  
import java.util.Random;  
  
import static com.yanoonigmail.rdroid.R.string.*user\_data*;  
  
*/\*\*  
 \* Created by Yaniv on 24-Feb-16.  
 \*/*public class TaskManager extends android.app.Service {  
 public static final int *TRY\_LOGIN* = 1;  
 public static final int *IS\_CONNECTED* = 2;  
 public static final int *IS\_LOGGED\_IN* = 3;  
 public static final int *LOGOUT* = 4;  
 private Server mServer;  
 private Thread mManageTasksThread;  
 private ArrayList<String> mIDList = new ArrayList<>();  
 // Binder given to clients  
 private final LocalBinder mBinder = new LocalBinder();  
 // Random number generator  
 private int mId = 9000;  
  
 @Override  
 public IBinder onBind(Intent intent) {  
 return mBinder;  
 }  
  
 @Override  
 public void onCreate() {  
 super.onCreate();  
 Log.*d*("service create", "service is alive");  
 setForeground();  
 mServer = Server.*getInstance*();  
 manageTasks();  
 }  
  
 @Override  
 public int onStartCommand(Intent intent, int flags, int startId) {  
 super.onStartCommand(intent, flags, startId);  
 Log.*d*("service start", "service is alive");  
 mServer = Server.*getInstance*();  
 manageTasks();  
 return *START\_NOT\_STICKY*;  
 }  
  
 @Override  
 public void onDestroy() {  
 super.onDestroy();  
 }  
  
 public class LocalBinder extends Binder {  
 public TaskManager getService() {  
 // Return this instance of the service so clients can call public methods.  
 return TaskManager.this;  
 }  
 @Override  
 protected boolean onTransact(int code, Parcel data, Parcel reply,  
 int flags) {  
 boolean success = false;  
 switch (code) {  
 case *TRY\_LOGIN*:  
 String[] inputStrings = new String[2];  
 data.readStringArray(inputStrings);  
 boolean[] outputBoolean = new boolean[1];  
 outputBoolean[0] = mServer.tryLogin(inputStrings[0], inputStrings[1]);  
 reply.writeBooleanArray(outputBoolean);  
 success = true;  
 break;  
 case *IS\_CONNECTED*:  
 boolean[] connectedBool = new boolean[1];  
 connectedBool[0] = mServer.isConnected();  
 reply.writeBooleanArray(connectedBool);  
 success = true;  
 break;  
 case *IS\_LOGGED\_IN*:  
 boolean[] loggedInBool = new boolean[1];  
 loggedInBool[0] = mServer.isLoggedInAfterFirstTry();  
 reply.writeBooleanArray(loggedInBool);  
 success = true;  
 break;  
 case *LOGOUT*:  
 boolean disconnected = mServer.disconnect();  
 if (disconnected) {  
 SharedPreferences preferences = getSharedPreferences(getString(*user\_data*), Context.*MODE\_PRIVATE*);  
 SharedPreferences.Editor preferences\_editor = preferences.edit();  
 preferences\_editor.remove("email");  
 preferences\_editor.remove("password");  
 preferences\_editor.apply();  
 }  
 boolean[] output\_bool\_array = new boolean[1];  
 output\_bool\_array[0] = disconnected;  
 reply.writeBooleanArray(output\_bool\_array);  
 }  
 data.recycle();  
 return success;  
 }  
 }  
  
 private void manageTasks() {  
 if (mManageTasksThread == null) {  
 mManageTasksThread = new Thread(new Runnable() {  
 public void run() {  
 while (true) {  
 if (mServer.isLoggedIn()) {  
 try {  
 Task task = Protocol.*taskRequest*(mServer.recv());  
 mIDList.add(task.getId());  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 } else {  
 try {  
 Thread.*sleep*(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
 });  
 mManageTasksThread.start();  
 }  
 }  
  
 private void setForeground() {  
 Notification mNotification =  
 new Notification.Builder(ApplicationContext.*getContext*())  
 .setContentTitle("RDroid is running")  
 .setContentText("The RDroid service is running")  
 .setSmallIcon(R.mipmap.*ic\_launcher*)  
 .setLargeIcon(BitmapFactory.*decodeResource*(getResources(), R.mipmap.*ic\_launcher*))  
 .setOngoing(true)  
 .build();  
 startForeground(mId, mNotification);  
 }  
}

**שרת הפרוקסי:**

קבצי ההגדרות:

**default server settings.cfg:**

my\_ip=0.0.0.0  
my\_port=9000  
main\_server\_ip=127.0.0.1  
main\_server\_port=9001  
buffer\_size=8192  
RSA\_bits=2048  
new\_line=<LF>  
id\_notification\_message=Rdroid PROXY<LF>NOTIFY\_SESSION\_ID<LF>  
disconnect\_notification\_message=Rdroid PROXY<LF>NOTIFY\_SESSION\_DISCONNECT<LF>

קבצי הקוד:

**EncryptionKeyMaker.py:**

#region -----------------Info-----------------  
#Name: Encryption Key Maker  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**from** Settings **import** \*  
**from** Printer **import** \*  
**from** Singleton **import** \*  
**from** Encryptor **import** \*  
**from** socket **import** \*  
**from** Crypto.PublicKey **import** RSA  
**from** Crypto.Cipher **import** PKCS1\_v1\_5  
**from** Crypto.Cipher **import** AES  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
  
**class EncryptionKeyMaker**(object):  
 # This class is used for creating symmetric encryption keys for communicating with phones.  
 \_\_metaclass\_\_ = Singleton  
 printer = Printer()  
 settings = Settings()  
 \_\_pure\_encryption\_key = None # The asymmetric encryption key that is used for the process of creating the symmetric key.  
 \_\_encryption\_key = None  
  
 **def** \_\_init\_\_(self):  
 self.printer.printMessage(self.\_\_class\_\_.\_\_name\_\_, 'Generating RSA key.')  
 self.\_\_pure\_encryption\_key = RSA.generate(int(self.settings.getSetting('RSA\_bits')))  
 self.\_\_encryption\_key = PKCS1\_v1\_5.new(self.\_\_pure\_encryption\_key)  
 self.printer.printMessage(self.\_\_class\_\_.\_\_name\_\_, 'RSA key generated.')  
  
 **def createEncryptor**(self, phone):  
 # Communicates with the given socket and creates a symmetric key that is used by both sides.  
 errors\_param = None  
 public\_key = self.\_\_pure\_encryption\_key.publickey().exportKey(format='DER')  
 **print** 'public key: ' + public\_key  
 phone.raw\_send(public\_key)  
 encrypted\_key = phone.raw\_recv()  
 **print** 'key: ' + encrypted\_key  
 **print** 'key len: ' + str(len(encrypted\_key))  
 phone\_key = self.\_\_encryption\_key.decrypt(encrypted\_key, errors\_param)  
 **print** 'phone key: ' + phone\_key  
 **return** Encryptor(AES.new(phone\_key))  
  
  
#endregion -----------------Class-----------------

**Encryptor.py:**

#region -----------------Info-----------------  
#Name: Encryptor  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**from** Settings **import** \*  
**from** Printer **import** \*  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
  
**class Encryptor**(object):  
 # An object that simplifies the use of symmetric keys.  
 printer = Printer()  
 settings = Settings()  
 \_\_encryption\_key = None # The symmetric key that the object uses.  
  
 **def** \_\_init\_\_(self, encryption\_key):  
 self.\_\_encryption\_key = encryption\_key  
  
 **def encrypt**(self, message):  
 # Adds padding to the message and encrypts it.  
 padded\_message = self.\_\_add\_padding(message)  
 encrypted\_message = self.\_\_encryption\_key.encrypt(padded\_message)  
 **return** encrypted\_message  
  
 **def decrypt**(self, message):  
 # Decrypts the message and removes the padding.  
 **if** message **is** None:  
 **return** None  
 **try**:  
 padded\_message = self.\_\_encryption\_key.decrypt(message)  
 unpadded\_message = self.\_\_remove\_padding(padded\_message)  
 **except** (ValueError, TypeError, IndexError) **as** e:  
 **print** str(e)  
 unpadded\_message = message  
 **return** unpadded\_message  
  
 @staticmethod  
 **def \_\_add\_padding**(message):  
 # Adds padding to the message  
 pad\_len = 16 - len(message) % 16  
 padded\_message = message.ljust(len(message) + pad\_len, chr(pad\_len))  
 **return** padded\_message  
  
 @staticmethod  
 **def \_\_remove\_padding**(message):  
 # Removes padding from the message.  
 pad\_len = ord(message[-1])  
 **if** pad\_len >= 16:  
 **return** message  
 real\_message = message[:-pad\_len]  
 **return** real\_message  
  
#endregion -----------------Class-----------------

**Filter.py:**

#region -----------------Info-----------------  
#Name: Filter  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**from** Settings **import** \*  
**from** Printer **import** \*  
**from** Singleton **import** \*  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
  
**class Filter**(object):  
 # An object that is used for filtering the messages that do not fit the protocol.  
 \_\_metaclass\_\_ = Singleton  
  
 **def filter**(self, message):  
 # Returns whether the given message fits the protocol or does not.  
 **if** message **is** None:  
 **return** False  
 **if** message.startswith('Rdroid CLIENT\n'):  
 **return** True  
 **return** False  
  
  
#endregion -----------------Class-----------------

**IDGenerator.py:**

#region -----------------Info-----------------  
#Name: ID Generator  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**from** Singleton **import** \*  
**from** Printer **import** \*  
**from** Settings **import** \*  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
  
**class IDGenerator**(object):  
 # An object that generates different ids for the connected sessions.  
 \_\_metaclass\_\_ = Singleton  
 \_\_last\_id = 0 # Saves the last id that was used.  
  
 **def generateId**(self):  
 # Generates and returns a new id by taking the number of the last id and increasing it by 1.  
 **if** self.outOfIds():  
 self.resetIds()  
 self.\_\_last\_id += 1  
 **return** str(self.\_\_last\_id)  
  
 **def resetIds**(self):  
 # Sets \_\_last\_id to 0.  
 self.\_\_last\_id = 0  
  
 **def outOfIds**(self):  
 # Returns whether this object is out of ids (which is very unlikely).  
 **if** self.\_\_last\_id == 2147483647:  
 **return** True  
 **return** False  
  
  
#endregion -----------------Class-----------------

**Main.py:**

#region -----------------Info-----------------  
#Name: Main  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**import** Manager  
#endregion -----------------Imports-----------------  
  
#region -----------------Main-----------------  
# Runs manager  
manager = Manager.Manager()  
manager.run()  
#endregion -----------------Main-----------------

**Manager.py:**

#region -----------------Info-----------------  
#Name: Manager  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**from** Singleton **import** \*  
**from** Printer **import** \*  
**from** Settings **import** \*  
**from** Server **import** \*  
**from** PhoneListener **import** \*  
**from** PhoneManager **import** \*  
**from** threading **import** Thread  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
  
**class Manager**(object):  
 \_\_metaclass\_\_ = Singleton  
 settings = Settings()  
 printer = Printer()  
 server = Server()  
 phone\_listener = PhoneListener()  
 phone\_manager = PhoneManager()  
 \_\_close = False # Specifies whether the thread that this object runs should close.  
  
 **def run**(self):  
 # Runs the main code of the object.  
 phone\_listener\_thread = Thread(name='phone\_listener\_thread', target=self.phone\_listener.runThread)  
 phone\_listener\_thread.setDaemon(True)  
 phone\_listener\_thread.start()  
 phone\_manager\_thread = Thread(name='phone\_manager\_thread', target=self.phone\_manager.runThread)  
 phone\_manager\_thread.setDaemon(True)  
 phone\_manager\_thread.start()  
 **while not** self.\_\_close:  
 user\_input = raw\_input()  
 self.\_\_inputResponse(user\_input)  
  
 **def closeThreads**(self):  
 # Tells the open threads to close.  
 self.\_\_closeThread()  
 self.phone\_listener.closeThread()  
 self.phone\_manager.closeThread()  
 self.server.closeThreads()  
 **return  
  
 def \_\_inputResponse**(self, user\_input):  
 # Analizes the given input and calls the matching method.  
 **if** user\_input == 'stop':  
 self.closeThreads()  
  
 **def \_\_closeThread**(self):  
 # Tells the thread of this object to close.  
 self.\_\_close = True  
  
#endregion -----------------Class-----------------

**Phone.py:**

#region -----------------Info-----------------  
#Name: Phone  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**from** Settings **import** \*  
**from** Printer **import** \*  
**from** EncryptionKeyMaker **import** \*  
**from** Encryptor **import** \*  
**from** Filter **import** \*  
**from** Server **import** \*  
**import** socket  
**import** base64  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
  
**class Phone**(object):  
 server = Server()  
 settings = Settings()  
 printer = Printer()  
 filter = Filter()  
 encryption\_key\_maker = EncryptionKeyMaker()  
 \_\_ready = False # Tells if the phone is ready for communication.  
 \_\_socket = None # The socket of the session with the phone.  
 \_\_ip\_address = '' # The ip address of the phone.  
 \_\_number = '' # The phone number of the connected phone (might be empty / useless).  
 \_\_encryptor = None # The encryptor that is used for communication with the phone.  
 \_\_close = False # Specifies whether the thread that this object runs should close.  
 buffer\_size = int(Settings().getSetting('buffer\_size'))  
  
 **def** \_\_init\_\_(self, phone\_socket, phone\_ip\_address, phone\_id, phone\_manager):  
 self.\_\_socket = phone\_socket  
 self.\_\_ip\_address = phone\_ip\_address  
 self.\_\_phone\_id = phone\_id  
 self.phone\_manager = phone\_manager  
  
 **def runThread**(self):  
 # The method that the thread runs.  
 self.\_\_notifyCreation()  
 **while not** self.\_\_close:  
 message = self.recv()  
 **if** self.filter.filter(message):  
 message = '%s:%s' % (self.getID(), message)  
 self.server.send(message)  
  
 **def raw\_send**(self, message):  
 # Sends a message to the phone.  
 message = base64.b64encode(message)  
 message = str(len(message)) + ':' + message + '\n'  
 **print** 'sent: ' + message  
 self.getSocket().sendall(message)  
  
 **def raw\_recv**(self):  
 # Receives a message from the phone.  
 **try**:  
 stay = True  
 message\_len = 0  
 **while** stay:  
 one\_char = self.getSocket().recv(1)  
 **if** one\_char == '':  
 **print** 'got empty'  
 self.closeObject()  
 **return** None  
 **elif** one\_char == ':':  
 stay = False  
 **else**:  
 **try**:  
 message\_len = message\_len \* 10 + int(one\_char)  
 **except** ValueError:  
 message\_len = 0  
 message = ''  
 **while** len(message) < message\_len:  
 message += self.getSocket().recv(min(self.buffer\_size, message\_len - len(message)))  
 **except** socket.error **as** e:  
 **print** str(e)  
 self.closeObject()  
 **return** None  
 **except** (IndexError, ValueError) **as** e:  
 **print** str(e)  
 **return** None  
 **try**:  
 **if** message[:-1] == '\n':  
 **return** base64.b64decode(message[:-1])  
 **return** base64.b64decode(message)  
 **except** TypeError:  
 **print** 'bad padding. message:' + message  
  
 **def send**(self, message):  
 # encrypts and then sends a message to the phone.  
 message = message.replace(self.settings.getSetting('new\_line'), '\n')  
 self.printer.printMessage(self.\_\_class\_\_.\_\_name\_\_, 'after edit to phone: ' + message)  
 encryptor = self.getEncryptor()  
 encrypted\_message = encryptor.encrypt(message)  
 self.printer.printMessage(self.\_\_class\_\_.\_\_name\_\_, 'after encryption to phone: ' + encrypted\_message)  
 self.raw\_send(encrypted\_message)  
  
 **def recv**(self):  
 # Receives a message from the phone and decrypts it.  
 message = self.raw\_recv()  
 **if** message **is not** None:  
 **if not** message.startswith('Rdroid CLIENT'):  
 encryptor = self.getEncryptor()  
 message = encryptor.decrypt(message)  
 **return** message  
  
 **def establishConnection**(self):  
 # Handles important early communication with the phone (sets encryption).  
 self.\_\_setEncryptor()  
 self.setReady()  
  
 **def setReady**(self):  
 # Makes it so that the phone would count as ready.  
 self.\_\_ready = True  
  
 **def isReady**(self):  
 # Returns whether the phone is ready or not.  
 **return** self.\_\_ready  
  
 **def getNumber**(self):  
 # Returns the phone number.  
 **return** self.\_\_number  
  
 **def getSocket**(self):  
 # Returns the socket of the current session.  
 **return** self.\_\_socket  
  
 **def getIp**(self):  
 # Returns the ip of the phone.  
 **return** self.\_\_ip\_address  
  
 **def getID**(self):  
 # Returns the ID that was given to the phone.  
 **return** self.\_\_phone\_id  
  
 **def getEncryptor**(self):  
 # Returns the encryptor object that this object uses.  
 **return** self.\_\_encryptor  
  
 **def closeObject**(self):  
 # Closes the thread that this object runs and removes the object from the phone manager (removes the object from the memory completely).  
 self.\_\_closeThread()  
 self.phone\_manager.deletePhone(self.getID())  
  
 **def \_\_notifyCreation**(self):  
 # Send the server a message to notify it that this phone connected and it's ID.  
 new\_line = self.settings.getSetting('new\_line')  
 my\_id = self.getID()  
 notification\_message = self.settings.getSetting('id\_notification\_message')  
 notification\_message += 'session\_id:%s%s' % (my\_id, new\_line)  
 self.server.send(notification\_message)  
  
 **def \_\_closeThread**(self):  
 # Closes the thread that runs on this object.  
 self.\_\_close = True  
  
 **def \_\_setEncryptor**(self):  
 # Sets an encryptor object for this object.  
 self.\_\_encryptor = self.encryption\_key\_maker.createEncryptor(self)  
  
  
#endregion -----------------Class-----------------

**PhoneListener.py:**

#region -----------------Info-----------------  
#Name: Phone Listener  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**from** Singleton **import** \*  
**from** Settings **import** \*  
**from** Printer **import** \*  
**from** PhoneManager **import** \*  
**from** socket **import** \*  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
  
**class PhoneListener**(object):  
 # An object that simplifies the use of the listening socket of this proxy server.  
 \_\_metaclass\_\_ = Singleton  
 settings = Settings()  
 printer = Printer()  
 phone\_manager = PhoneManager()  
 \_\_close = False # Specifies whether the thread that this object runs should close.  
  
 **def** \_\_init\_\_(self):  
 self.my\_ip = self.settings.getSetting('my\_ip')  
 self.my\_port = int(self.settings.getSetting('my\_port'))  
 self.listen\_socket = socket()  
 self.listen\_socket.bind((self.my\_ip, self.my\_port))  
  
 **def runThread**(self):  
 # The main method that is supposed to run on a thread.  
 self.listen\_socket.listen(10)  
 **while not** self.\_\_close:  
 client\_socket, client\_address = self.listen\_socket.accept()  
 client\_ip = client\_address[0]  
 self.phone\_manager.addPhone(client\_socket, client\_ip)  
 self.listen\_socket.close()  
  
 **def closeThread**(self):  
 # Tells the thread of this object to close.  
 self.\_\_close = True  
 accept\_trigger\_socket = socket()  
 accept\_trigger\_socket.connect(('127.0.0.1', self.my\_port))  
 accept\_trigger\_socket.close()  
  
#endregion -----------------Class-----------------

**PhoneManager.py:**

#region -----------------Info-----------------  
#Name: Phone Manager  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**from** Singleton **import** \*  
**from** Settings **import** \*  
**from** Printer **import** \*  
**from** Server **import** \*  
**from** Phone **import** \*  
**from** IDGenerator **import** \*  
**from** threading **import** Thread  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
  
**class PhoneManager**(object):  
 \_\_metaclass\_\_ = Singleton  
 settings = Settings()  
 printer = Printer()  
 server = Server()  
 id\_generator = IDGenerator()  
 \_\_phone\_dict = {} # A dictionary that is used to remember the connected phones sessions by session IDs.  
 \_\_close = False # Specifies whether the thread that this object runs should close.  
  
 **def addPhone**(self, phone\_socket, phone\_ip\_address):  
 # Starts a thread that adds a phone to the phone dictionary.  
 phone = Phone(phone\_socket, phone\_ip\_address, self.id\_generator.generateId(), self)  
 add\_phone\_thread = Thread(name='add\_phone\_thread', target=self.\_\_addPhoneToDictThread, args=[phone])  
 add\_phone\_thread.setDaemon(True)  
 add\_phone\_thread.start()  
  
 **def deletePhone**(self, phone\_id):  
 # Removes a phone from the dictionary.  
 phone\_dict = self.\_\_phone\_dict  
 **if** phone\_dict.has\_key(phone\_id):  
 self.\_\_notifyDeletedPhone(phone\_id)  
 **del** phone\_dict[phone\_id]  
  
 **def getPhone**(self, phone\_id):  
 # Returns the phone that has the given ID.  
 phone\_dict = self.\_\_phone\_dict  
 self.printer.printMessage(self.\_\_class\_\_.\_\_name\_\_, 'phone\_dict: %s' % phone\_dict)  
 **if** phone\_dict.has\_key(phone\_id):  
 phone = phone\_dict[phone\_id]  
 **return** phone  
 **return** None  
  
 **def runThread**(self):  
 # The main method of this object that is supposed to run on a new thread.  
 **while not** self.\_\_close:  
 server\_message = self.server.recv()  
 **if** ':' **in** server\_message:  
 phone\_id, message = server\_message.split(':', 1)  
 phone = self.getPhone(phone\_id)  
 **if** phone **is not** None:  
 self.printer.printMessage(self.\_\_class\_\_.\_\_name\_\_, 'sending to phone: ' + message)  
 phone.send(message)  
 **else**:  
 self.\_\_notifyDeletedPhone(phone\_id)  
  
 **def closeThread**(self):  
 # Tells the thread of this object to close.  
 self.\_\_close = True  
  
 **def \_\_addPhoneToDictThread**(self, phone):  
 # A method that is supposed to run on a new thread that adds a phone to the dictionary and start it's thread.  
 phone.establishConnection()  
 phone\_ip\_address = phone.getIp()  
 phone\_id = phone.getID()  
 self.printer.printMessage(self.\_\_class\_\_.\_\_name\_\_, 'ip: %s, id: %s' % (phone\_ip\_address, phone\_id))  
 self.\_\_phone\_dict[phone\_id] = phone  
 phone\_thread = Thread(name=('phone\_thread-%s' % phone\_id), target=phone.runThread)  
 phone\_thread.setDaemon(True)  
 phone\_thread.start()  
  
 **def \_\_notifyDeletedPhone**(self, phone\_id):  
 # Notifies the server of phones that were deleted from the dictionary (most likely disconnected) and their IDs.  
 new\_line = self.settings.getSetting('new\_line')  
 notification\_message = self.settings.getSetting('disconnect\_notification\_message')  
 notification\_message += 'session\_id:%s%s' % (phone\_id, new\_line)  
 self.server.send(notification\_message)  
  
  
#endregion -----------------Class-----------------

**Printer.py:**

#region -----------------Info-----------------  
#Name: Printer  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**from** Singleton **import** \*  
**import** time  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
  
**class Printer**(object):  
 # An object that is used to print messages to the textual user interface.  
 \_\_metaclass\_\_ = Singleton  
  
 **def** \_\_init\_\_(self):  
 **print** 'Current time:%s' % (self.\_\_getFullTime())  
  
 **def printMessage**(self, class\_name, message):  
 # Prints a message.  
 **print** '%s-%s: %s' % (class\_name, self.\_\_getTime(), message)  
  
 **def \_\_getTime**(self):  
 # Returns the time in this format: Hour:Minute:Second.  
 **return** time.strftime("%H:%M:%S", time.strptime(time.ctime()))  
  
 **def \_\_getFullTime**(self):  
 # Returns the time in this format: Day(of the week) Month Day(of the month) Hour:Minute:Second.  
 **return** time.strftime("%a %b %d %H:%M:%S %Y", time.strptime(time.ctime()))  
  
  
#endregion -----------------Class-----------------

**Server.py:**

#region -----------------Info-----------------  
#Name: Server  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**from** Settings **import** \*  
**from** Printer **import** \*  
**from** Singleton **import** \*  
**from** Queue **import** \*  
**from** threading **import** \*  
**import** time  
**import** socket  
**import** base64  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
  
**class Server**(object):  
 # An object used as an interface for communicating with the server.  
 \_\_metaclass\_\_ = Singleton  
 settings = Settings()  
 printer = Printer()  
 \_\_server\_socket = socket.socket() # The socket that is connected to the server.  
 \_\_input\_queue = Queue() # A queue that holds the messages that were sent from the server.  
 \_\_output\_queue = Queue() # A queue that holds the messages that are going to be sent to the server.  
 \_\_connect\_lock = Lock() # A lock used to make sure that this object will not try to reconnect to the server twice at the same time.  
 \_\_close = False # Specifies whether the thread that this object runs should close.  
 buffer\_size = int(Settings().getSetting('buffer\_size'))  
  
 **def** \_\_init\_\_(self):  
 self.\_\_connectToServer()  
 self.runThreads()  
  
 **def runThreads**(self):  
 # A method that runs the thread that are supposed to run on this object.  
 server\_listener\_thread = Thread(name='server\_listener\_thread', target=self.\_\_serverReceiverThread)  
 server\_listener\_thread.setDaemon(True)  
 server\_listener\_thread.start()  
 output\_sender\_thread = Thread(name='output\_sender\_thread', target=self.\_\_outputSenderThread)  
 output\_sender\_thread.setDaemon(True)  
 output\_sender\_thread.start()  
  
 **def closeThreads**(self):  
 # Closes the thread that run on this object.  
 self.\_\_close = True  
 self.send('closing')  
 self.\_\_server\_socket.close()  
 self.\_\_addInput('closing')  
  
 **def send**(self, message):  
 # Adds a message to the output queue so it will be sent when possible.  
 message = message.replace(self.settings.getSetting('new\_line'), '\n')  
 self.\_\_output\_queue.put(message)  
  
 **def recv**(self):  
 # Returns the first message from the input queue. blocks until it has a value to return (if the queue is empty).  
 **while** self.\_\_input\_queue.empty():  
 time.sleep(0.1)  
 **return** self.\_\_input\_queue.get()  
  
 **def \_\_serverReceiverThread**(self):  
 # A method that is supposed to run on a thread that receives messages from the server and adds them to the input queue.  
 **while not** self.\_\_close:  
 **try**:  
 self.\_\_addInput(self.\_\_recv())  
 **except** socket.error **as** e:  
 **print** str(e)  
 self.\_\_connectToServer()  
  
 **def \_\_recv**(self):  
 message = self.\_\_server\_socket.recv(self.buffer\_size)  
 message\_len = int(message.split(':', 1)[0])  
 message = message.split(':', 1)[1]  
 **while** len(message) < message\_len:  
 message += self.\_\_server\_socket.recv(self.buffer\_size)  
 **return** base64.b64decode(message)  
  
 **def \_\_outputSenderThread**(self):  
 # A method that is supposed to run on a thread that sends messages from the output queue to the server.  
 **while not** self.\_\_close:  
 **try**:  
 message = self.\_\_output\_queue.get()  
 self.\_\_send(message)  
 **except** socket.error **as** e:  
 **print** str(e)  
 self.\_\_connectToServer()  
  
 **def \_\_addInput**(self, input):  
 # Adds a message to the input queue.  
 self.\_\_input\_queue.put(input)  
  
 **def \_\_send**(self, message):  
 # sends a message to the server.  
 **print** 'to server: ' + message  
 message = base64.b64encode(message)  
 message = str(len(message)) + ':' + message  
 **print** 'to server encoded: ' + message  
 self.\_\_server\_socket.sendall(message)  
  
 **def \_\_connectToServer**(self):  
 # connects to the server.  
 acquired = self.\_\_connect\_lock.acquire()  
 **if** acquired:  
 self.printer.printMessage(self.\_\_class\_\_.\_\_name\_\_, 'Connecting to server')  
 self.\_\_server\_socket = socket.socket()  
 ip = self.settings.getSetting('main\_server\_ip')  
 port = int(self.settings.getSetting('main\_server\_port'))  
 connected = False  
 **while not** connected:  
 connected = True  
 **try**:  
 self.\_\_server\_socket.connect((ip, port))  
 **except** socket.error:  
 connected = False  
 self.printer.printMessage(self.\_\_class\_\_.\_\_name\_\_, 'Connected to server')  
 self.\_\_connect\_lock.release()  
  
  
#endregion -----------------Class-----------------

**Settings.py:**

#region -----------------Info-----------------  
#Name: Settings  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**import** os  
**from** Singleton **import** \*  
**from** Printer **import** \*  
**from** threading **import** \*  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
  
**class Settings**(object):  
 # An object that is used as an interface to the settings files.  
 \_\_metaclass\_\_ = Singleton  
 \_\_printer = Printer()  
 \_\_settings\_location = os.path.dirname(os.path.abspath(\_\_file\_\_)) + '\\Files\\server settings.cfg' # A string of the location of the settings file.  
 \_\_default\_settings\_location = os.path.dirname(os.path.abspath(\_\_file\_\_)) + '\\Files\\default server settings.cfg' # A string of the location of the default settings file.  
 \_\_settings\_dict = {} # A dictionary that remembers the settings from the settings file.  
 \_\_read\_lock = Semaphore(10) # A semaphore used to make sure that there are no reading threads running when a writing thread is running.  
 \_\_write\_lock = Lock() # A lock used to make sure that no writing threads are running at the same time.  
  
 **def** \_\_init\_\_(self):  
 **if** os.path.isfile(self.\_\_settings\_location):  
 self.\_\_loadSettings(self.\_\_settings\_location)  
 **elif** os.path.isfile(self.\_\_default\_settings\_location):  
 self.\_\_loadSettings(self.\_\_default\_settings\_location)  
 self.\_\_updateSettingsFile()  
 **else**:  
 self.\_\_printer.printMessage(self.\_\_class\_\_.\_\_name\_\_, 'No settings files found. The program will likely crush soon.')  
  
 **def getSetting**(self, key):  
 # Returns the requested settings.  
 **if** self.\_\_settings\_dict.has\_key(key):  
 **return** self.\_\_settings\_dict[key]  
 **else**:  
 **return** None  
  
 **def setSetting**(self, key, value):  
 # Sets the requested setting to given value.  
 **if** self.\_\_settings\_dict.has\_key(key):  
 self.\_\_settings\_dict[key] = value  
 self.\_\_updateSettingsFile()  
 **return** True  
 **else**:  
 **return** False  
  
 **def \_\_loadSettings**(self, location=\_\_settings\_location):  
 # Loads the settings from the settings file to the settings dictionary.  
 self.\_\_write\_lock.acquire()  
 self.\_\_read\_lock.acquire()  
 self.\_\_write\_lock.release()  
 settings\_file = open(location, 'r')  
 settings\_list = settings\_file.read()  
 settings\_file.close()  
 self.\_\_read\_lock.release()  
 settings\_list = settings\_list.split('\n')  
 **for** setting **in** settings\_list:  
 **if** setting != '':  
 **if** setting[0] != '#':  
 setting = setting.split('=', 1)  
 setting[1] = setting[1].replace('<CRLF>', '\r\n')  
 self.\_\_settings\_dict[setting[0]] = setting[1]  
  
 **def \_\_updateSettingsFile**(self):  
 # Writes the current settings to the settings file.  
 new\_settings\_text = ''  
 settings\_template = open(self.\_\_default\_settings\_location, 'r')  
 settings\_list = settings\_template.read().split('\n')  
 settings\_template.close()  
 **for** setting **in** settings\_list:  
 **if** setting != '':  
 **if** setting[0] == '#':  
 new\_settings\_text += setting  
 **else**:  
 setting = setting.split('=', 1)  
 new\_settings\_text += '%s=%s' % (setting[0], self.\_\_settings\_dict[setting[0]])  
 new\_settings\_text += '\n'  
 new\_settings\_text = new\_settings\_text[:-1]  
 self.\_\_write\_lock.acquire()  
 **for** i **in** xrange(10):  
 self.\_\_read\_lock.acquire()  
 settings\_file = open(self.\_\_settings\_location, 'w')  
 settings\_file.write(new\_settings\_text)  
 settings\_file.close()  
 **for** i **in** xrange(10):  
 self.\_\_read\_lock.release()  
 self.\_\_write\_lock.release()  
  
  
#endregion -----------------Class-----------------

**Singleton.py:**

#region -----------------Info-----------------  
#Name: Singleton  
#Version: 1.0  
#By: Yaniv Sharon  
#endregion -----------------Info-----------------  
  
#region -----------------Imports-----------------  
**from** threading **import** \*  
#endregion -----------------Imports-----------------  
  
#region -----------------Class-----------------  
  
**class Singleton**(type):  
 # A metaclass to all of the singleton classes that makes them be singletons.  
 \_instances = {}  
 \_lock = Lock()  
  
 **def** \_\_call\_\_(cls, \*args, \*\*kwargs):  
 **while** True:  
 acquired = cls.\_lock.acquire()  
 **if** acquired:  
 **if** cls **not in** cls.\_instances:  
 cls.\_instances[cls] = super(Singleton, cls).\_\_call\_\_(\*args, \*\*kwargs)  
 cls.\_lock.release()  
 **return** cls.\_instances[cls]  
  
  
#endregion -----------------Class-----------------

**המודולים שלי בשרת הראשי:**

**Phone.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

/// <summary>

/// The Phone object represents a phone that is connected to the server.

/// </summary>

public class Phone

{

private string Id;

private string User\_Email = "";

private int Task\_Id\_Generator\_Number = 0;

private Dictionary<string, string> Recieved\_Tasks\_Dict = new Dictionary<string, string>();

public Phone(string Id)

{

this.Id = Id;

}

public bool IsLoggedIn()

{

return User\_Email != "";

}

public string GetId()

{

return Id;

}

public string GetEmail()

{

return User\_Email;

}

public void SetEmail(string email)

{

User\_Email = email;

}

public void Send(string message)

{

ProxySocketInterface.Get\_Instance().Send(message);

}

public string GenerateTaskId()

{

Task\_Id\_Generator\_Number++;

return Task\_Id\_Generator\_Number.ToString();

}

public void AddRecievedTask(string Task\_Id, string Task\_Output)

{

Recieved\_Tasks\_Dict.Add(Task\_Id, Task\_Output);

}

public string GetRecievedTaskOutput(string Task\_Id)

{

string Recieved\_Task;

if (Recieved\_Tasks\_Dict.TryGetValue(Task\_Id, out Recieved\_Task))

{

Recieved\_Tasks\_Dict.Remove(Task\_Id);

return Recieved\_Task;

}

return null;

}

}

**Protocol.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

/// <summary>

/// A static class with methods used for analyzing and creating messages.

/// </summary>

public static class Protocol

{

public static string Cut\_Len\_From\_Message(string Message, out int Len)

{

Len = int.Parse(Message.Split(':')[0]);

return string.Join("", Message.Split(':').Skip(1));

}

public static string Add\_Len\_To\_Message(string Message)

{

return (Message.Length - 1).ToString() + ":" + Message;

}

public static bool Is\_Proxy\_Message(string Message)

{

return Message.StartsWith("Rdroid PROXY\n");

}

public static bool IsPhoneMessage(string Message)

{

try

{

return Message.Split(':')[1].StartsWith("Rdroid CLIENT\n");

}

catch (IndexOutOfRangeException e)

{

return false;

}

}

/\*\*

\* returns the id of the phone who sent the message.

\*\*/

public static string Get\_Sender\_Id(string Message)

{

return Message.Split(':')[0];

}

public static string Get\_Message\_Purpose(string Message)

{

string[] Message\_Lines = Message.Split('\n');

return Message\_Lines[1];

}

public static Dictionary<string, string> Get\_Message\_Parameters(string Message)

{

Dictionary<string, string> Parameter\_Dict = new Dictionary<string, string>();

string[] Parameter\_Lines = Message.Split('\n').Skip(2).ToArray<string>();

foreach (string Parameter in Parameter\_Lines)

{

string[] Split\_Parameter = Parameter.Split(new char[] { ':' }, 2);

if (Split\_Parameter.Length == 2)

{

Parameter\_Dict.Add(Split\_Parameter[0], Split\_Parameter[1]);

}

}

return Parameter\_Dict;

}

public static string Create\_Login\_Result\_Message(string Id, bool Successful)

{

string Result\_String;

if (Successful)

{

Result\_String = "success";

}

else

{

Result\_String = "failure";

}

string Message = Id + ":Rdroid SERVER\nLOGIN\n";

Message += "result:" + Result\_String + "\n";

return Message;

}

public static string Create\_Task\_Message(string Phone\_Id, string Task\_Id, string Type, string[] Parameters)

{

string Message = Phone\_Id + ":Rdroid SERVER\nTASK\n";

Message += "id:" + Task\_Id + "\n";

Message += "type:" + Type + "\n";

Message += "parameters:";

foreach (string Parameter in Parameters)

{

Message += Parameter + ",";

}

Message = Message.Substring(0, Message.Length - 1);

return Message;

}

}

**Proxy.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Threading;

using System.Data.OleDb;

using System.Data;

using localhostWebService;

/// <summary>

/// Summary description for PhoneManager

/// </summary>

public sealed class Proxy

{

private static readonly Proxy our\_instance = new Proxy();

private ProxySocketInterface Proxy\_Socket;

private Dictionary<string, Phone> Phone\_By\_Id\_Dict;

private Dictionary<string, Phone> Phone\_By\_Email\_Dict;

private Queue<string> Input\_Queue;

private OleDbConnection myConnection;

private Thread Proxy\_Manager\_Thread;

public static Proxy Get\_Instance()

{

return our\_instance;

}

private Proxy()

{

Proxy\_Socket = ProxySocketInterface.Get\_Instance();

Phone\_By\_Id\_Dict = new Dictionary<string, Phone>();

Phone\_By\_Email\_Dict = new Dictionary<string, Phone>();

Input\_Queue = new Queue<string>();

string connectionString = Connect.getConnectionString();

myConnection = new OleDbConnection(connectionString);

this.Proxy\_Manager\_Thread = new Thread(new ThreadStart(this.ProxyManagerThreadMethod));

Proxy\_Manager\_Thread.IsBackground = true;

Proxy\_Manager\_Thread.Start();

}

private void ProxyManagerThreadMethod()

{

while (true)

{

string Message = Proxy\_Socket.Recv();

if (Protocol.Is\_Proxy\_Message(Message))

{

string Purpose = Protocol.Get\_Message\_Purpose(Message);

switch (Purpose)

{

case "NOTIFY\_SESSION\_ID":

string Connected\_Id;

if (Protocol.Get\_Message\_Parameters(Message).TryGetValue("session\_id", out Connected\_Id))

{

AddPhoneById(Connected\_Id);

}

break;

case "NOTIFY\_SESSION\_DISCONNECT":

try

{

string Disconnected\_Id;

if (Protocol.Get\_Message\_Parameters(Message).TryGetValue("session\_id", out Disconnected\_Id))

{

Phone Removed\_Phone = GetPhoneById(Disconnected\_Id);

if (Removed\_Phone.IsLoggedIn())

{

Phone\_By\_Email\_Dict.Remove(Removed\_Phone.GetEmail());

}

Phone\_By\_Id\_Dict.Remove(Disconnected\_Id);

}

}

catch(NullReferenceException)

{ }

break;

}

}

else

{

if (Protocol.IsPhoneMessage(Message))

{

//todo do stuff with the phone message

string Phone\_Id = Protocol.Get\_Sender\_Id(Message);

string Purpose = Protocol.Get\_Message\_Purpose(Message);

if (Purpose == "LOGIN")

{

string Email;

string Password;

if (Protocol.Get\_Message\_Parameters(Message).TryGetValue("email", out Email) && Protocol.Get\_Message\_Parameters(Message).TryGetValue("password", out Password))

{

if (IsValidLogin(Email, Password))

{

AddPhoneByEmail(Phone\_Id, Email);

Proxy\_Socket.Send(Protocol.Create\_Login\_Result\_Message(Phone\_Id, true));

}

else

{

Proxy\_Socket.Send(Protocol.Create\_Login\_Result\_Message(Phone\_Id, false));

}

}

}

else

{

if (Purpose == "TASK\_RESULTS")

{

Dictionary<string, string> Task\_Parameters\_Dict = Protocol.Get\_Message\_Parameters(Message);

string Task\_Id;

string Task\_Output;

if (Task\_Parameters\_Dict.TryGetValue("id", out Task\_Id) && Task\_Parameters\_Dict.TryGetValue("output", out Task\_Output))

{

GetPhoneById(Phone\_Id).AddRecievedTask(Task\_Id, Task\_Output);

}

}

}

}

}

}

}

private bool IsValidLogin(string Email, string Password)

{

OleDbDataReader reader;

UserDetails userDetails = new UserDetails();

userDetails.email = Email;

userDetails.password = Password;

OleDbCommand myCmd = new OleDbCommand("CheckIfUserExistsByEmailAndPassword", myConnection);

myCmd.CommandType = CommandType.StoredProcedure;

OleDbDataAdapter adapter = new OleDbDataAdapter();

adapter.SelectCommand = myCmd;

DataSet dataset = new DataSet();

OleDbParameter objParam;

UserDetails x = new UserDetails();

objParam = myCmd.Parameters.Add("@Email", OleDbType.BSTR);

objParam.Direction = ParameterDirection.Input;

objParam.Value = userDetails.email;

objParam = myCmd.Parameters.Add("@Password", OleDbType.BSTR);

objParam.Direction = ParameterDirection.Input;

objParam.Value = userDetails.password;

try

{

myConnection.Open();

reader = myCmd.ExecuteReader();

while (reader.Read())

{

x.phoneNumber = reader["PhoneNumber"].ToString();

x.firstName = reader["FirstName"].ToString();

x.lastName = reader["LastName"].ToString();

x.email = reader["Email"].ToString();

}

}

catch (Exception ex)

{

throw ex;

}

finally

{

myConnection.Close();

}

if (x == null)

{

return false;

}

return true;

}

private void AddPhoneById(string Id)

{

try

{

Phone\_By\_Id\_Dict.Add(Id, new Phone(Id));

}

catch (Exception e)

{ }

}

private void AddPhoneByEmail(string Id, string Email) // needs the phone to be in the Phone\_By\_Id\_Dict.

{

Phone Added\_Phone = GetPhoneById(Id);

if (Added\_Phone != null)

{

if (GetPhoneByEmail(Email) != null)

{

Phone\_By\_Email\_Dict.Remove(Email);

}

Added\_Phone.SetEmail(Email);

try

{

Phone\_By\_Email\_Dict.Add(Email, Added\_Phone);

}

catch (Exception e)

{ }

}

}

public Phone GetPhoneById(string Id)

{

Phone Wanted\_Phone;

if (Phone\_By\_Id\_Dict.TryGetValue(Id, out Wanted\_Phone))

{

return Wanted\_Phone;

}

return null;

}

public Phone GetPhoneByEmail(string Email)

{

Phone Wanted\_Phone;

if (Phone\_By\_Email\_Dict.TryGetValue(Email, out Wanted\_Phone))

{

return Wanted\_Phone;

}

return null;

}

public string HandleTask(string Email, string Type, string[] Parameters)

{

try

{

Phone Tasked\_Phone = GetPhoneByEmail(Email);

string Phone\_Id = Tasked\_Phone.GetId();

string Task\_Id = Tasked\_Phone.GenerateTaskId();

string Task\_Message = Protocol.Create\_Task\_Message(Phone\_Id, Task\_Id, Type, Parameters);

Proxy\_Socket.Send(Task\_Message);

string Task\_Output = null;

bool Got\_Output = false;

while (!Got\_Output)

{

Task\_Output = Tasked\_Phone.GetRecievedTaskOutput(Task\_Id);

if (Task\_Output != null)

{

Got\_Output = true;

}

else

{

Thread.Sleep(100);

}

}

return Task\_Output;

}

catch

{

return "";

}

}

}

**ProxySocketInterface.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Net.Sockets;

using System.Net;

using System.Text;

using System.Threading;

/// <summary>

/// Summary description for ProxySocketInterface

/// </summary>

public sealed class ProxySocketInterface

{

private static readonly ProxySocketInterface our\_instance = new ProxySocketInterface();

private Socket clientsocket;

private int BufferSize = 8192; // The amount of data

private Queue<string> input\_queue = new Queue<string>();

private Queue<string> output\_queue = new Queue<string>();

private Thread Start\_Thread;

private Thread Recv\_Thread;

private Thread Send\_Thread;

private Mutex Connect\_Mutex = new Mutex();

public static ProxySocketInterface Get\_Instance()

{

return our\_instance;

}

private ProxySocketInterface()

{

this.Start\_Thread = new Thread(new ThreadStart(this.StartThreadMethod));

Start\_Thread.IsBackground = true;

Start\_Thread.Start();

}

private void StartThreadMethod()

{

Connect();

RunThreads();

}

private void Connect()

{

lock (this)

{

if (this.clientsocket == null || !this.clientsocket.Connected)

{

TcpListener serverSocket = new TcpListener(IPAddress.Parse("0.0.0.0"), 9001);

TcpClient clientSocket = default(TcpClient);

serverSocket.Start();

clientSocket = serverSocket.AcceptTcpClient();

serverSocket.Stop();

this.clientsocket = clientSocket.Client;

}

}

}

private void RunThreads()

{

this.Recv\_Thread = new Thread(new ThreadStart(this.RecvThreadMethod));

Recv\_Thread.IsBackground = true;

Recv\_Thread.Start();

this.Send\_Thread = new Thread(new ThreadStart(this.SendThreadMethod));

Send\_Thread.IsBackground = true;

Send\_Thread.Start();

}

private byte[] Encode(string message) // Turning the message from string to byte[] to send it through the socket.

{

byte[] encoded\_message = Encoding.UTF8.GetBytes(message); // Encoding the message

return encoded\_message; // Returns the byte array.

}

private void RecvThreadMethod() // Decodes the incoming message from byte[] to string.

{

byte[] OneLetterBuffer = new byte[1];

while (true)

{

try

{

int MessageLen = 0;

string MessageLenString = "";

while (!MessageLenString.EndsWith(":"))

{

clientsocket.Receive(OneLetterBuffer);

MessageLenString += Encoding.ASCII.GetString(OneLetterBuffer, 0, OneLetterBuffer.Length).TrimEnd('\0');

}

char[] MessageLenFlippedCharArray = MessageLenString.Trim(':').Reverse().ToArray();

for (int i = 0; i < MessageLenFlippedCharArray.Length; i++)

{

char ShouldBeNumberChar = MessageLenFlippedCharArray[i];

int RecievedNumber;

if (int.TryParse(ShouldBeNumberChar.ToString(), out RecievedNumber))

{

MessageLen += Convert.ToInt32(RecievedNumber \* Math.Pow(10, Convert.ToDouble(i)));

}

else

{

break;

}

}

string RecievedMessage = "";

while (RecievedMessage.Length < MessageLen)

{

int NextBufferSize = MessageLen - RecievedMessage.Length;

if (NextBufferSize > this.BufferSize)

{

NextBufferSize = this.BufferSize;

}

byte[] Buffer = new byte[NextBufferSize];

clientsocket.Receive(Buffer);

RecievedMessage += Encoding.ASCII.GetString(Buffer, 0, Buffer.Length).TrimEnd('\0');

}

this.input\_queue.Enqueue(Base64Decode(RecievedMessage.TrimEnd('\n'))); // Returns the string.

}

catch (SocketException e)

{

Connect();

}

}

}

private void SendThreadMethod() // Sends the message.

{

while (true)

{

try

{

if (this.output\_queue.Count > 0)

{

string Message = Base64Encode(this.output\_queue.First());

Message = Protocol.Add\_Len\_To\_Message(Message);

clientsocket.Send(Encode(Message)); // Sends an encoded message.

this.output\_queue.Dequeue();

}

else

{

Thread.Sleep(100);

}

}

catch (SocketException e)

{

Connect();

}

}

}

private static string Base64Encode(string plainText)

{

try

{

var plainTextBytes = System.Text.Encoding.UTF8.GetBytes(plainText);

return System.Convert.ToBase64String(plainTextBytes);

}

catch (Exception e)

{

return plainText;

}

}

private static string Base64Decode(string base64EncodedData)

{

try

{

var base64EncodedBytes = System.Convert.FromBase64String(base64EncodedData);

return System.Text.Encoding.UTF8.GetString(base64EncodedBytes);

}

catch (Exception e)

{

return base64EncodedData;

}

}

public string Recv()

{

while (true)

{

if (this.input\_queue.Count != 0)

{

return input\_queue.Dequeue();

}

else

{

Thread.Sleep(100);

}

}

}

public void Send(string message)

{

this.output\_queue.Enqueue(message);

}

}

**Task.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data;

using System.Data.OleDb;

/// <summary>

/// Summary description for Task

/// </summary>

public static class Task

{

public static Proxy proxy = Proxy.Get\_Instance();

public static string[] GetFilesInFolder(string email, string folder\_location)

{

string[] input = new string[1];

input[0] = folder\_location;

string output = proxy.HandleTask(email, "GET\_FILES\_IN\_FOLDER", input);

return output.Split(',');

}

public static bool GetFile(string email, string file\_location, out string Data)

{

Data = null;

string[] input = new string[1];

input[0] = file\_location;

string output = proxy.HandleTask(email, "GET\_FILE", input);

if (output.StartsWith("success,"))

{

Data = output.Substring("success,".Length);

return true;

}

else

{

if (output.StartsWith("failure,"))

{

Data = output.Substring("failure,".Length);

return false;

}

}

return false;

}

public static bool SaveFile(string email, string location, string file\_data)

{

string[] input = new string[2];

input[0] = location;

input[1] = file\_data;

string output = proxy.HandleTask(email, "SAVE\_FILE", input);

if (output == "success")

{

return true;

}

if (output == "failure")

{

return false;

}

throw new Exception();

}

public static bool SaveContact(string email, string DisplayName, string MobileNumber, string emailID)

{

string[] input = new string[3] {DisplayName, MobileNumber, emailID};

string output = proxy.HandleTask(email, "SAVE\_CONTACT", input);

if (output == "success")

{

return true;

}

if (output == "failure")

{

return false;

}

throw new Exception();

}

}