**ממ״ן 15 –README**

מימוש השרת:

מימוש השרת מתחלק למספר חלקים:

* מחלקת Server – עוטפת את הפעלת השרת כולו. כוללת פתיחת listener, פתיחת database משותף וקבלת לקוחות. ממומשת כמחלקה שלמה כדי לאפשר שימוש פשוט מתוך קוד קיים, אם יווצר הצורך.
* מחלקת ClientSession – מהווה Thread – על מנת לאפשר הפעלה וטיפול במספר לקוחות במקביל. תפקידה העיקרי הוא להמתין לבקשות מהלקוח ולטפל בהן. מנגנון הטיפול הוא מנגנון Stateless – כלומר, אין זיכרון משותף בין בקשה לבקשה, שאינו הDatabase. מימוש בצורה זו מפשט את אופן הפעולה והטיפול בבקשות.  
  *טיפול בבקשה בודדת מתחיל בhandle\_single\_request.* פונקציה זו מאחזרת את הHeader מתוך החיבור ללקוח, מבצעת Parsing, ולפי הRequest Code מבצעת קריאה נוספת מהחיבור ללקוח וParsing של גוף הבקשה. בנוסף, כל בקשה מלבד Register מהווה ״התחברות״ של המשתמש – וישנו עדכון LastSeen בDB בהתאם. פונקציה זו משתמשת במיפוי של קודי בקשות למתודות של המחלקה (HANDLERS\_MAP) שמטפלות בבקשה מבחינה לוגית.
* מחלקת database מנהלת אחסון של נתוני לקוחות וקבצים בRAM ובSQLite Local DB. היא כוללת פונקציות שמבצעות מניפולציה על הנתונים (אחזור, עדכון, הוספה ומחיקה).
* המודול protocol כולל הגדרות של טיפוסי נתונים ופורמטים לpacking/unpacking שלהם אל ומתוך בתים. ביניהם – לכל חלק מבקשה (Header או סוג של תוכן) יש dataclass, ומיפוי לפורמט. באופן אנלוגי עבור תגובה. בנוסף, ישנן המרות אוטומטיות של טיפוסי נתונים (לדוגמה Enum שמהווה int או str שמהווה bytes). באופן זה, מונגשים טיפוסי Python טבעיים לנתונים, שמאפשרים טיפול בבקשות ובתגובות בצורה טבעית ופשוטה, שמומרת אוטומטית לבתים שיכולים להישלח ברשת. במודול זה ישנן פונקציות נוספות שמסייעות לבצע מעבר בין bytes שמגיעים מהרשת לבין בקשה, וכן בין תגובה לבין bytes שיוכלו להישלח ברשת.
* המודול utils כולל מספר פונקציות ומחלקות עזר פשוטות מאוד, אשר עומדות בפני עצמן – חישוב CRC, קבלת קובץ מוצפן, הצפנה ופענוח כנדרש. עקב הפשטות – לחלק מהפונקציות אין טעם במימוש מחלקה, ועדיין יש רצון לשמר הפרדה לוגית – שגורר הפרדה מודולרית בהתאם.

ארכיטקטורה זו של השרת מאפשרת שינוי פונקציונליות ולוגיקה בצורה נוחה מאוד (אחסון הנתונים במקום אחד, לוגיקה כללית במקום אחר, וטיפוסי נתונים במקום שונה), והוספת פונקציונליות בצורה פשוטה (הגדרת dataclass ומיפוי שלה לRequest/Response Code יחד עם Packing Format, והוספת מתודת handler שמבצעת לוגיקה).

מימוש הלקוח

הלקוח ממומש, גם הוא, כמספר מחלקות, שתפקיד כל אחת שונה. **על x86, עם C++17 ומעלה!**

* מחלקת Client היא המחלקה הראשית של הלקוח. היא מטפלת בעיבוד הנתונים, בניית הבקשות ופענוח התשובות, וביצוע הלוגיקה המקומית (לדוגמה, ניסיונות שליחה חוזרים). במחלקה מוגדרות מספר שיטות עזר שמאחזרות Header מתוך הSocket או מסייעות לבנות Request מסוגים שונים.
* מחלקת RSADecryptor היא מחלקה שמספקת ממשק לטעינה או יצירה של מפתח RSA פרטי, קבלת מפתח ציבורי מתאים לו, ופענוח בקשות שמוצפנות באמצעות המפתח הציבורי.
* מחלקת EncryptedFileSender היא מחלקה שמספקת ממשק להצפנה של קבצים ושליחתם. בפרט, היא מספקת שיטה אחת לחישוב גודל של קובץ לאחר שהוצפן, ושיטה נוספת שמצפינה קובץ ושולחת אותו דרך Socket.
* מחלקת MeInfo שמסייעת לעבד את הקובץ me.info – מפענוח שלו ועד כתיבתו.
* תת-התיקייה util מכילה מספר מחלקות סטטיות בעלות שימושים כלליים:

1. מחלקת SocketHelper, שמכילה מימושים קצרים לפונקציות אשר מפשטות שליחה או קבלה של Struct-ים דרך Socket. המחלקה כוללת שיטות לשליחה וקבלה לפי גודל שמוערך על ידי sizeof, או לפי גודל דינמי (במקרה ויש, לדוגמה, מערכים שאינם סטטיים במחלקה וגודלם נקבע בזמן ריצה).
2. מחלקת Uid, שמכילה שיטה לקריאה ושיטה לכתיבה של uuid כHex String – תחת format.
3. מחלקת Base64, שמכילה שיטה לקידוד ופענוח מחרוזות מבסיס 64 או לבסיס 64, גם תחת format.
4. מחלקת CRC – שמסייעת לחשב את cksum בהתאם למימוש בלינוקס

חשוב להעיר, כי רבות מהשגיאות שעלולות להיזרק (בין אם של boost ובין אם של הClient אינן נתפסות). הסיבה לכך היא שכולן מעבירות מידע חיוני אודות העובדה שחלה שגיאה קריטית – והקוד המשתמש בClient צריך לתפוס אותה ולפעול מול המשתמש בהתאם. מטרת מחלקת הClient אינה לממש ממשק משתמש – אלא ממשק תקשורת ולוגיקה שמממש את הפרוטוקול כנדרש. זו גם הסיבה שפענוח transfer.info נעשה בקובץ main.cpp – קובץ זה מספק הוראות להפעלת הClient ולא מהווה חלק ממנו.