**ספר פרוייקט – צ'אט פשוט**

מגיש: רון שרמן

ת.ז: 217241264

**שם הפרויקט:** מערכת צ'אט פשוטה מרובת משתמשים עם יכולות שיתוף קבצים

**תקציר**

פרויקט זה מתאר תכנון, פיתוח והטמעה של מערכת צ'אט מבוססת שרת-לקוח, המאפשרת תקשורת בזמן אמת בין משתמשים מרובים. המערכת תומכת ביצירת קבוצות דינמיות, שליחת הודעות טקסט ושיתוף קבצים באופן יעיל. המסמך מציג את תהליך הפיתוח באופן כרונולוגי, המחולק לשלבים, החל מהקמת התשתית הבסיסית ועד להוספת תכונות מתקדמות ופתרון בעיות.

**הגדרת הבעיה**

הצורך בתקשורת מהירה ויעילה הוא קריטי בעולם המודרני. קיימות פלטפורמות רבות, אך ישנו ביקוש מתמיד לפתרונות מותאמים אישית, המאפשרים שליטה מלאה על הנתונים והפונקציונליות. הבעיה המרכזית שהפרויקט בא לפתור היא יצירת פלטפורמת תקשורת עצמאית, שאינה תלויה בשירותי צד שלישי, ומספקת למשתמשים כלים לניהול שיחות פרטיות וקבוצתיות, כולל היכולת להעביר קבצים בצורה פשוטה ואינטואיטיבית.

**הפתרון המוצע ומטרת הפרויקט**

הפתרון הוא יישום צ'אט המורכבת מצד שרת וצד לקוח. השרת מנהל את כל הלוגיקה המרכזית: אימות משתמשים, ניהול קבוצות, ניתוב הודעות ואחסון היסטוריית השיחות. הלקוח מספק ממשק משתמש נוח ואינטואיטיבי המאפשר למשתמשים להתחבר, לנהל שיחות ולהעביר קבצים.

**מטרתו העיקרית של הפרויקט** היא לספק מערכת תקשורת שלמה, יציבה ופונקציונלית, המדגימה את העקרונות של תכנות רשתות, ניהול תהליכונים (Threading), עבודה עם בסיסי נתונים ופיתוח ממשק משתמש.

**תפקידי הלקוח:**

הלקוח הוא התוכנה שרצה על מחשב המשתמש. הוא אחראי על כל חווית המשתמש ומספק את הכלים לביצוע הפעולות הבאות:

* **ממשק גרפי (GUI):** הצגת חלונות, כפתורים ושדות טקסט באמצעות ספריית **PyQt6**.
* **אימות:** שליחת כינוי לשרת לצורך התחברות.
* **ניהול שיחות:** שליחת הודעות טקסט, הצגת היסטוריית שיחות וקבלת הודעות חדשות חדשות בזמן אמת.
* **ניהול קבוצות:** שליחת בקשות לשרת ליצירת קבוצות חדשות, הוספת חברים ועזיבת קבוצות קיימות.
* **שיתוף קבצים:** בחירת קבצים מקומיים ושליחתם לשרת, וכן בקשת הורדה של קבצים משותפים.

**תפקידי השרת:**

השרת הוא המוח של המערכת, והוא פועל ברקע (לרוב על מחשב מרוחק) ללא ממשק משתמש. תפקידיו המרכזיים הם:

* **ניהול חיבורים:** האזנה וקבלת חיבורים מלקוחות מרובים במקביל, תוך הקצאת תהליכון נפרד לכל משתמש.
* **עיבוד וניתוב הודעות:** כאשר השרת מקבל הודעה, הוא מנתח אותה (בעזרת מודול הפרוטוקול) כדי להבין את סוגה (הודעת צ'אט, בקשה ליצירת קבוצה וכו') ואת יעדה. לאחר מכן, הוא מנתב אותה לכל המשתמשים הרלוונטיים באותה קבוצה.
* **לוגיקה עסקית:** טיפול בבקשות מהלקוחות, כגון אימות כינויים, יצירת קבוצות חדשות ושיוך משתמשים אליהן.
* **ניהול אחסון:** אינטראקציה עם בסיס הנתונים (SQLite) לשמירת הודעות, קבוצות ופרטי משתמשים. כמו כן, הוא מנהל את האחסון הפיזי של קבצים שהועלו על ידי משתמשים.

**ארכיטקטורת מערכת ומודלי יישום**

פרק זה מציג את המבנה הארכיטקטוני של המערכת, את סביבת הפיתוח והכלים ששימשו לבנייתה, ואת המודולים המרכזיים המרכיבים את היישום.

**סביבת העבודה – Eco System**

סביבת הפיתוח של הפרויקט התבססה על כלים וספריות קוד פתוח, שנבחרו במטרה ליצור מערכת יציבה, ניתנת להרחבה וקלה לתחזוקה.

* **שפת תכנות: Python 3** פייתון נבחרה כשפת התכנות הראשית בזכות הפשטות שלה, קהילת המפתחים הענפה, והזמינות של ספריות חזקות בתחום הרשתות, מסדי הנתונים ופיתוח ממשקי משתמש.
* **ממשק משתמש גרפי (GUI): PyQt6** ספריית PyQt6 נבחרה לבניית ממשק המשתמש בצד הלקוח. זוהי ספרייה ותיקה ויציבה המספקת סט עשיר של רכיבי UI (ווידג'טים) ומאפשרת ליצור אפליקציית דסקטופ בעלות מראה מקצועי הפועלות על מערכות הפעלה שונות.
* **מסד נתונים: SQLite** לצורך שמירת נתונים בצד השרת, נעשה שימוש ב-SQLite. זהו מנוע מסד נתונים יחסי קל משקל, מבוסס קבצים, שאינו דורש התקנה או הגדרות מורכבות. הוא מתאים במיוחד לפרויקטים בסדר גודל זה ומספק את כל היכולות הנדרשות לשמירת מידע על משתמשים, קבוצות והיסטוריית הודעות.
* **תקשורת רשת: Socket & Threading** התקשורת בין השרת ללקוחות מבוססת על מודול ה-socket המובנה בפייתון, המאפשר יצירת תקשורת TCP/IP. כדי לטפל במספר משתמשים בו-זמנית, השרת משתמש במודול ה-threading ליצירת תהליכון נפרד עבור כל לקוח מתחבר.
* **סביבת פיתוח משולבת (IDE):** הפיתוח בוצע בסביבת pycharm, המספקת כלים מתקדמים לניפוי שגיאות (debugging), השלמה אוטומטית של קוד וניהול קבצים נוח.

**ארכיטקטורת המערכת**

המערכת בנויה במודל **שרת-לקוח (Client-Server)**. בארכיטקטורה זו, השרת מהווה את הרכיב המרכזי המנהל את כל המידע והלוגיקה, בעוד שהלקוחות הם יישומים הפועלים על מחשבי הקצה ומתקשרים עם השרת כדי לקבל שירותים.

השרת הוא יישום מרובה-תהליכונים (Multi-threaded), מה שמאפשר לו לשרת מספר לקוחות במקביל וביעילות. התהליכון הראשי של השרת מאזין באופן קבוע לחיבורים נכנסים. כאשר לקוח חדש מתחבר, השרת יוצר עבורו תהליכון ייעודי, שתפקידו לטפל בכל התקשורת עם אותו לקוח באופן בלעדי. גישה זו מונעת מלקוח אחד להפריע לפעולתם של אחרים ומבטיחה שהמערכת תישאר רספונסיבית.

התקשורת בין השרת ללקוח מתבצעת על גבי פרוטוקול TCP/IP ומבוססת על הודעות בפורמט **JSON (JavaScript Object Notation)** . פורמט זה נבחר בזכות היותו קריא, קל משקל ונתמך באופן מובנה בשפת פייתון, מה שמקל על אריזת ופריסת נתונים מובנים בין חלקי המערכת.

**דיאגרמת ארכיטקטורת המערכת**

TCP/IP

TCP/IP

TCP/IP

לקוח 2

N לקוח

לקוח 1

שרת

יוצר תהליכון חדש עבור כל לקוח שמתחבר

תהליכון האזנה ראשי

תהליכון לקוח N

תהליכון לקוח 1

לוגיקה מרכזית וניהול

מערכת קבצים

מודול בסיס נתונים

**דיאגרמת זרימת נתונים ראשית**

להלן תיאור של זרימת הנתונים:

**הסבר על זרימת תהליכי העבודה בשרת**

תהליך העבודה של השרת מתחיל עם הפעלתו וממשיך בלולאה אינסופית, הממתינה לפעולות מהלקוחות.

1. **אתחול והאזנה:** עם הפעלת השרת, הוא מאתחל את מסד הנתונים (init\_db), טוען את רשימת הקבוצות הקיימות לזיכרון ומתחיל להאזין על פורט מוגדר מראש (למשל, 12394) לחיבורי TCP נכנסים.
2. **קבלת לקוח חדש:** כאשר לקוח חדש מתחבר, התהליכון הראשי של השרת מקבל את החיבור ומיד יוצר ומפעיל **תהליכון חדש** (handle\_client) שיוקדש אך ורק לטיפול באותו לקוח. לאחר מכן, התהליכון הראשי חוזר מיד להאזין ללקוחות חדשים נוספים.
3. **תהליך אימות (Authentication):** התהליכון החדש שנוצר מצפה לקבל מהלקוח את ההודעה הראשונה, שהיא תמיד בקשת רישום כינוי ("type": "nickname"). השרת בודק אם הכינוי תקין ואינו תפוס. אם כן, הוא שולח ללקוח אישור ("status": "ok"), רושם את המשתמש במסד הנתונים ומוסיף אותו לקבוצת "General".
4. **לולאת קבלת הודעות:** לאחר האימות, התהליכון נכנס ללולאה אינסופית שבה הוא מאזין להודעות נוספות מאותו לקוח.
5. **ניתוח וניתוב:** כל הודעה שמגיעה מהלקוח מנותחת. השרת בודק את שדה ה-"type" בהודעה כדי להבין את כוונת המשתמש (למשל, "chat", "create\_group", "upload\_request"). בהתבסס על סוג ההודעה, השרת מפעיל את הפונקציה המתאימה (handle\_chat, handle\_group\_creation וכו').
6. **אינטראקציה עם רכיבים אחרים:**
   * **הודעת צ'אט:** השרת שומר את ההודעה במסד הנתונים ולאחר מכן מפיץ אותה לכל שאר הלקוחות המחוברים לאותה קבוצה.
   * **בקשה ליצירת קבוצה:** השרת מוסיף את הקבוצה והחברים בה למסד הנתונים ומעדכן את כל המעורבים.
   * **בקשת העלאת קובץ:** השרת פותח תהליכון נוסף ופורט זמני כדי לקבל את הקובץ במקביל, מבלי לעצור את התקשורת הראשית.
7. **סיום וניתוק:** אם החיבור עם הלקוח מתנתק (או אם מתקבלת הודעה ריקה), הלולאה נשברת. השרת מסיר את המשתמש מרשימת המחוברים הפעילים, סוגר את ה-socket, והתהליכון שהוקדש לאותו משתמש מסיים את חייו.

**פירוט רכיבים (מודולי יישום)**

היישום מחולק למספר מודולים (קבצי פייתון), כאשר לכל אחד אחריות ותפקיד מוגדרים. החלוקה למודולים בצד הלקוח ובצד השרת מאפשרת ארגון קוד יעיל ותחזוקה קלה.

**מודול צד לקוח**

אחראי על חווית המשתמש, הצגת המידע וניהול התקשורת מול השרת.

* **שמות הקבצים של הקוד:** main.py, window.py, dialogs.py, network\_client.py.
* **טכנולוגיה / מודולים:** PyQt6, socket, QThread (חלק מ-PyQt6), json.
* **פונקציונליות:**
  + **main.py:** נקודת הכניסה של האפליקציה. אחראי על יצירת האובייקטים הראשיים (חלון לוגין, חלון צ'אט, מנהל רשת), חיבור הסיגנלים והחריצים (signals & slots) של PyQt6, והפעלת לולאת האירועים הראשית.
  + **window.py:** מגדיר את כל רכיבי הממשק הגרפי (חלון התחברות, חלון צ'אט ראשי, כפתורים, תיבות טקסט). הוא לוכד את פעולות המשתמש (כמו לחיצה על כפתור "שלח") ופולט סיגנלים שמפעילים את הלוגיקה המתאימה ב-main.py.
  + **dialogs.py:** מגדיר חלונות דיאלוג משניים, כגון החלון ליצירת קבוצה חדשה, ומטפל בלוגיקה הפנימית שלהם (כמו חיפוש משתמשים להוספה).
  + **network\_client.py:** מנהל את כל התקשורת מול השרת. הוא פועל בתהליכון רקע נפרד (QThread) כדי לא להקפיא את ממשק המשתמש. הוא אחראי על שליחת הודעות JSON לשרת, קבלת הודעות מהשרת, וניהול תהליכי העלאה והורדה של קבצים בתהליכונים נפרדים.

**מודול צד שרת**

אחראי על הלוגיקה המרכזית של המערכת, ניהול המידע והתיאום בין כל המשתמשים.

* **שמות הקבצים של הקוד:** server.py, data\_base.py, models.py.
* **טכנולוגיה:** socket, threading, sqlite3, json.
* **פונקציונליות:**
  + **server.py:** לב המערכת. הוא מאזין לחיבורים, יוצר תהליכון (thread) לכל לקוח, מנתח את ההודעות הנכנסות בעזרת הפרוטוקול, ומפעיל את הלוגיקה המתאימה. הוא מנהל את רשימת המשתמשים והקבוצות הפעילות בזיכרון.
  + **data\_base.py:** שכבת הגישה לנתונים. מודול זה מכיל את כל הפונקציות האחראיות על ביצוע פעולות מול מסד הנתונים של SQLite, כגון יצירת טבלאות, הוספה, עדכון ומחיקה של רשומות (משתמשים, קבוצות, הודעות).
  + **models.py:** מגדיר מחלקות פשוטות (User, GroupChat) המשמשות לייצוג אובייקטים בזיכרון השרת. מחלקות אלו עוזרות לארגן את המידע על משתמשים מחוברים וחברי קבוצות בצורה נוחה.

**מודול משותף (לקוח ושרת)**

* **protocol.py:** משמש כ"מילון" משותף לשרת וללקוח. הוא מכיל פונקציונליות ליצירה וניתוח של הודעות ה-JSON, ומבטיח ששני הצדדים "מדברים באותה שפה" ומבינים את מבנה ההודעות.

**מבנה מסד הנתונים**

מסד הנתונים chat\_clients.db מורכב מארבע טבלאות מרכזיות, המאפשרות שמירה וניהול של כל המידע במערכת.

**שם הטבלה: users** טבלה זו מרכזת את כל המידע על המשתמשים שנרשמו למערכת. כל משתמש מזוהה על ידי הכינוי הייחודי שלו. הטבלה שומרת גם נתונים נוספים כמו מועד ההתחברות הראשון והאחרון וכתובת ה-IP, שיכולים לשמש לניתוח וניהול.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| nickname | TEXT | Primary Key, שם המשתמש הייחודי. |
| first\_seen | TEXT | חותמת זמן של ההתחברות הראשונה. |
| last\_seen | TEXT | חותמת זמן של ההתחברות האחרונה. |
| ip\_address | TEXT | כתובת ה-IP האחרונה של המשתמש. |

**שם הטבלה: groups** טבלה זו מגדירה את כל קבוצות השיחה הקיימות במערכת. כל קבוצה מקבלת מזהה ייחודי מספרי (group\_id) ושם ייחודי. שמירת הקבוצות בטבלה נפרדת מאפשרת ניהול קל והוספת תכונות עתידיות לקבוצות.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| group\_id | INTEGER | Primary Key, מזהה ייחודי אוטומטי לקבוצה. |
| group\_name | TEXT | שם הקבוצה הייחודי. |
| created\_at | TEXT | חותמת זמן של יצירת הקבוצה. |

**שם הטבלה: group\_members** זוהי טבלת קישור (linking table) שתפקידה לחבר בין משתמשים לקבוצות. היא מאפשרת לממש קשר של "רבים לרבים" (many-to-many), שכן כל משתמש יכול להיות חבר במספר קבוצות, ובכל קבוצה יכולים להיות מספר חברים. כל רשומה בטבלה מייצגת שיוך של משתמש אחד לקבוצה אחת.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| group\_id | INTEGER | Foreign Key המצביע לטבלת groups. |
| user\_nickname | TEXT | Foreign Key המצביע לטבלת users. |

**שם הטבלה: messages** הטבלה המרכזית ביותר, המכילה את היסטוריית כל ההודעות שנשלחו במערכת. כל הודעה משויכת לקבוצה שבה היא נשלחה ולמשתמש ששלח אותה. עמודת ה-content גמישה ויכולה להכיל טקסט רגיל או מחרוזת JSON במקרה של הודעות מערכת או שיתוף קבצים.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| message\_id | INTEGER | Primary Key, מזהה ייחודי אוטומטי להודעה. |
| group\_id | INTEGER | Foreign Key המצביע לקבוצה אליה נשלחה ההודעה. |
| user\_nickname | TEXT | Foreign Key המצביע למשתמש ששלח את ההודעה. |
| content | TEXT | תוכן ההודעה (טקסט רגיל או אובייקט JSON עבור קבצים). |
| timestamp | TEXT | חותמת זמן מדויקת של שליחת ההודעה. |

**רפלקציה**

במהלך הפרויקט נרכשו מיומנויות רבות בתחום תכנות הרשתות והיישומים המבוזרים. האתגר המרכזי היה ניהול נכון של תהליכונים (threads) כדי להבטיח שהמערכת תהיה יציבה, תגיב במהירות ולא תיכנס למצבי קיפאון (deadlock). כמו כן, תכנון פרוטוקול תקשורת ברור ומובנה התגלה כשלב קריטי להצלחת הפרויקט. העבודה עם ממשק משתמש גרפי (PyQt6) דרשה הבנה של מודל ה-Event-Driven והפרדה נכונה בין לוגיקת התצוגה ללוגיקת הרקע. חלוקת הפרויקט לשלבים ברורים איפשרה להתמודד עם המורכבות בצורה הדרגתית ויעילה.