**פרויקט תקשורת נתונים**

לוחות זמנים: מסיימים ביום שני עד שלוש בצהריים – אז נעשה הצגה של הפרויקט. ביום רביעי בתשע בבוקר להגיש את הפרויקט הגמור גמור.

הפרויקט: לבנות אפליקציית צ'אט ארגוני. צ'אט שתמיד כל המשתתפים נמצאים באותה רשת. לא יוצאות הודעות מחוץ לגבולות הרשת של הארגון.

במסגרת האפליקציה: יהיה להקים סרבר, להקים הקליינט. מה הפונקציונליות?

צריך להירשם למערכת. צריכה להיות אופציה בהתחלה של REGISTRATION. הרשמה מצריכה סיסמא ושם משתמש. ברגע שנרשמנו, על פניו לא צריך להירשם שוב, זה יישמר במבנה נתונים כלשהו – כולל את הSOCKET של הקליינט. ההרשמה על פניו צריכה לקרות פעם אחת בלבד. אחרי ההרשמה, כל פעם שנכנסים אפשר לעשות LOGIN למערכת.

LOGIN – על ידי שם משתמש וסיסמא שהוגדרו כבר. רק אחרי שזה עובר, אפשר להיכנס למערכת ולעשות את הפעולות של המערכת.

חווית המשתמש: מפעילים את הסרבר וכלום לא קורה – לא אמורות להיות שם הדפסות. מבחינת הקליינט, כשזה עובד, מה שהוא יראה במסך: אופציה 1 של להירשם, אופציה 2 של להיכנס – LOGIN. אופציה 3 – לצאת – EXIT. זה המסך הראשון שהמשתמש ייראה. הוא יכול לבחור את אחת מהאופציות האלו. EXIT יוצא לגמרי מהקליינט.

בכניסה – לפני שרואים את המסך הזה, מבחינת הקליינט, הוא כבר הקים קשר עם הסרבר, אם הוא לא מצליח אז בכלל לא ייראו את המסך הזה. רק אחרי שעשו CONNECT והצלחנו אז מציגים את הדבר הזה.

צריך בדיקות בREGISTRATION – לקבוע כללים. אם אני נרשמת עם משהו שקיים, אז תהיה שגיאה. הKEY זה השם משתמש. סיסמא יכולה להיות אותו דבר. להחזיר שגיאה אם יש שגיאה עם השם משתמש או הסיסמא, אם הם לא קיימים / לא תקינים

לחדד: לא נבחנים פה על יכולות וויזואליות – לא להשקיע בזה. רק אם יהיה זמן בסוף.

אחרי שעושים LOGIN, יהיה מסך חדש:

במערכת, ניתן לעשות JOIN לקבוצה. אפשר להיכנס לקבוצה קיימת. אופציה נוספת – לעזוב קבוצה קיימת, להתנתק ממנה. אפשר גם ליצור קבוצה.

אופציה 1 – JOIN GROUP – יכול להיות תהליך מורכב, אפשר להציג שם את כל הקבוצות הקיימות כרגע, או שאפשר להוסיף אופציה נפרדת שתציג את הקבוצות הקיימות.

אופציה 2 – LEAVE GROUP

אופציה 3 – CREATE GROUP

אופציה 4 – LOG OUT (מהמערכת). זה מחזיר למסך של הREGISTRATION והLOGIN וכן הלאה – המסך הראשי

EXIT זה היחיד שמוציא לגמרי מהכל – רק שם אעשה CLOSE לכאורה. בEXIT יהיה את ההתנתקות מהסרבר.

LOGOUT – מוציא את הקליינט מכל הקבוצות שהוא נמצא בהם.

עד מתי הקבוצות קיימות? אין כאן אופציה של לסגור קבוצה באפליקציה – זה רק כי זה מסובך.

הסגירת קבוצה תהיה כשאין בה קליינטים בכלל. האחרון יסגור אותה.

מי שעושה CREATE GROUP – אוטומטית הוא כבר מצטרף לקבוצה.

ב-LEAVE GROUP – לבדוק אם אני המשתמש האחרון, אם כן אז לסגור את הקבוצה.

אחרי שעשינו JOIN GROUP, נפתחים שני טרמינלים. באחד מהחלונות יהיה את האופציה להקיש הודעות לצ'אט, ובחלון השני יכיל את כל ההודעות שרצות בצ'אט.

יהיה צריך לחקור איך לפתוח חלונות טרמינל מתוך התוכנית שלנו, ולא סתם לפתוח את זה אלא להריץ שם תוכנית.

יהיו שם תוכניות קטנות: אחת מקבלת אינפוטים מהMAIN, מקשיבה לKEYBOARD עד שהיא מקבלת הודעה ולשלוח אותה לכתובת IP והPORT שקיבלה, והתוכנית השנייה מקשיבה לאותו ערוץ וברגע שהיא שומעת היא מדפיסה את זה.

כל קבוצה צריכה לקבל IP MULTICAST לפורט, כך שכולם יראו את כל ההודעות שנמצאות באותה קבוצה.

מי שיעשה את פתיחת החלונות יצטרך לבדוק איך לעשות MULTICAST בLAN. לא אמור לקחת יותר מחצי יום עבודה לאדם אחד??

עם החלונות האלה אפשר לשחק ברמה של מיקום, גודל, צבעים וכולי – מומלץ לא להשקיע בזה בינתיים.

תמיד יהיה את החלון הראשי של טרמינל, וכל קבוצה שאני מצטרפת קופצים לי 2 חלונות.

כל משתמש יכול להיות בכמה קבוצות בו זמנית.

כשאני עושה LEAVE GROUP, החלונות האלו אמורים גם להיסגר.

פונקציית KILL – סוגר חלון של טרמינל כנראה.

מבחינת חלוקת העבודה בין הקליינט לסרבר: הקליינט הוא מקיש מה הוא רוצה לעשות. הכל עובר לסרבר – הוא מנהל את כל האפליקציה. הסרבר צריך להכיל מבנה של כל הקבוצות, של כל היוזרים וכן הלאה. הכל עובר דרך הסרבר.

מבחינת התקשורת – בין הקליינטים לסרבר זה TCP. הצ'אטים עצמם ברמת הקבוצות זה UDP כי זה MULTICAST.

לגבי MULTICAST: זה פרוטוקולים מורכבים מאוד, איך קבוצה אחת מקבלת IP אחד. כשאני רוצה לעשות MULTICAST בתוך הLAN אני עוקבת את המורכבות הזו.

בסרבר יהיה צריך להחזיק כתובות של MULTICAST.

CLASS D מתחיל מ224.0.0.0 עד 239.255.255.255. נגיד 100 כתובות. ואז לנהל את הכתובות האלו. ברגע שנוצרת קבוצה, הסרבר ייתן IP מתוך הPULL שלו לקבוצה, וכל מי שמצטרף לקבוצה יקבל את הIP הזה. הסרבר אמור לתת גם IP וגם PORT. מאחר ויש IP שונה לכל קבוצה, אפשר לעבוד עם אותו פורט – הסרבר יכול לתת אותו פורט לכל קבוצה. צריך PULL של IP, לא של פורטים. הסרבר צריך לספק את שניהם לקבוצות.

להחליט באיזה מבנה לנהל את זה. קבוצה שנסגרת – הIP שלה חוזר לPULL של הכתובות.

כל התקשורת ברמה של הקבוצות בצ'אטים נעשה בUDP. זה נמצא בשני התהליכים שדיברנו עליהם. בקליינט יהיה 3 טרמינלים – אחד שזה הקליינט עצמו שמדבר מול הסרבר, ואז שני חלונות נפרדים – אחד ששולח הודעות ואחד שמקבל ומדפיס אותם (הצ'אט).

אם יהיה זמן בסוף – מומלץ להצפין את הDATA שעובר בצ'אט – אפשר בצורה פשוטה לעשות הצפנה כביכול. מומלץ גם לשפר את הנושא שייראה יותר יפה אם יהיה זמן.

עד עכשיו שפיתחנו דברים, פיתחנו אפליקציה ופונקציות שקוראות אחת לשנייה. כאן המצב הוא אחרת. הפעם, האפליקציה היא בשני חלקים – הקליינט והסרבר, ושניהם מדברים דרך תקשורת. זה אומר, שחייבים שהם ידעו איך לדבר בין אחד השני. אין כאן קריאה לפונקציה, צריך להעביר הודעה מצד אחד לשני ולתת תשובה מצד השני לראשון. איך לעשות את זה? צריך לבנות פרוטוקול. פרוטוקול של הודעות, שהקליינט מעביר לסרבר והסרבר נותן תשובה.

לדוגמא שעושים REGISTRATION, הקליינט קולט את המידע, מעביר את זה לסרבר, והסרבר בודק אם השם משתמש והסיסמא הם תקינים. הוא צריך להחזיר את זה בהודעה לקליינט שידע.

REGISTRATION REQUEST ו-REGISTRATION REPLY

LOGIN REQUEST -LOGIN REPLY

LOGOUT REQUEST

CREATE GROUP וכן הלאה.

יש המון סוגים של הודעות שצריכות לרוץ מצד לצד ולדעת איך לטפל בכל אחד.

איך בונים את ההודעות האלו?

צריך להיות פורמט להודעה כזו שכל צד יוכל לפענח אותה.

מה לא לעשות: אי אפשר להגיד שיש מבנה שמעבירים אותו כהודעה. בלתי אפשרי. זה כי בכל מחשב שונה, מבנה נראה אחרת.

אם יש לי מבנה ואני רוצה שהתוכן יעבור לצד השני, צריך לבנות מהמבנה הזה הודעה, ובצד השני – לבנות מההודעה מבנה.

פורמט של הודעה: מקובל מבנה שנקרא TLV – TAG LENGTH VALUE. ההודעה מורכבת מ3 חלקים. TAG – מה זו ההודעה הזו? האם זה REGISTRATION REQUEST, LOGIN REPLY וכן הלאה. הLENGTH – הגודל שלה. VALUE – התוכן, ההודעה עצמה.

נגיד, ב-REGISTARTION REQUEST, איך תראה הודעה כזו? בהתחלה יהיה TAG, זה יהיה ערך אחד – 01 לדוגמא. חובה – שכל הTAGS ייבנו בENUM. אז יהיה גם LENGTH – שהוא גם רק BYTE אחד.

הVALUE מכיל גם שם משתמש וגם סיסמא. צריך להגדיר כמה בתים זה השם משתמש, לדוגמא 4, ואחרי זה יהיה 4 תווים, ואז אחרי זה נגדיר גם 4 בתים, ולאחר זה יהיה את ה4 תווים עצמם. כך שאורך ההודעה יהיה 10 – זה כולל את הTAG, הLENGTH, והVALUE.

כשמעבירים הודעה בתקשורת, אפשר להעביר כל ערך שהוא. זה לא חייב להיות ערך ASCII.

TAG – BYTE אחד.

LENGTH – יכול להיות יותר מBYTE אחד. כדי שלא נצטרך כל הזמן לעשות המרות של HTONS כל הזמן, נגדיר שהLENGTH יהיה BYTE אחד. זה אומר שגוף ההודעה לא יכול להיות יותר מ255 בתים.

VALUE –

אם אני רוצה להכניס מספר שהוא גדול יותר מ255, זה לא INT או LONG, צריך לתת את ה-SIZEOF של הסוג כפול SIZEOF של CHAR. לא בטוח שיהיה צורך בזה בפרויקט.

הKEY זה השם משתמש. צריך לבדוק בLOGIN שהסיסמא היא חוקית, אבל בREGISTRATION אם קיים כבר שם משתמש צריך לבדוק את זה.

מה יש בתוך המודול של הפרוטוקול הזה בקוד עצמו?

יש צד של קליינט וצד של סרבר. שניהם צריכים להכיר את כל סוגי ההודעות, כי אם צד אחד שולח הודעה, השני שמקבל צריך לדעת איך לטפל בה. לכן יהיה מודול אחד של פרוטוקול, שגם הקליינט יקמפל וגם הסרבר יקמפל.

מה יהיה ב-PROTOCOL.C? על כל סוג של הודעה, יהיו שתי פונקציות. אחת שעושה PACKING – מקבלת את המידע, נגיד בצורת מבנה, בונה את ההודעה בפורמט הנכון, ושולחת את זה. ופונקציה שעושה UNPACKING – קיבלתי הודעה ועכשיו צריך לחלץ את המידע משם, נגיד במקרה של REGISTRATION אז לחלץ את המחרוזות של השם משתמש והסיסמא וכן הלאה.

בקובץ H של הפרוטוקול, יהיה ENUM של כל הTAGS – סוגי ההודעות. אולי גם עוד ENUM. אולי יהיה שם גם מבנים – נגיד מבנה של שם משתמש וסיסמא וכן הלאה. איך הולך התהליך – נגיד הקליינט רוצה לשלוח הודעה לסרבר. האפליקציה שולחת לפרוטוקול פעולת PACK לאותה הודעה. הפרוטוקול אורז את ההודעה בבאפר. האפליקציה צריכה להעביר את הבאפר לפרוטוקול. הPACK חייב תמיד להחזיר את האורך הכללי של ההודעה. רק הפרוטוקול מכיר את המבנה של מה שקורה בתוך הבאפר. הפרוטוקול בפעולת הPACK יחזיר את הכמות הכללית של הבתים של ההודעה, כדי שאני אעשה SEND אעשה את זה בדיוק עם גודל ההודעה ועם הבאפר שהוא ארז לי.

אחרי שהאפליקציה קיבלה הודעה שעשו לה PACKING, היא שולחת את זה לסרבר. בצד השני קורה התהליך ההפוך – ההודעה נקלטת מהקליינט. היא שולחת לפרוטוקול לבצע UNPACKING של ההודעה, ורק אחרי שבוצע הUNPACKING היא מתחילה לבצע את הפעולה של ההודעה.

בקליינט – יהיה CREATE, DESTROY, SEND, RECIEVE.

הסרבר נט לא מכיר את הפרוטוקול. הסרבר נט זה מה שכתבתי אתמול. יהיה עוד מודולים של הסרבר שהם יתנהלו מול הפרוטוקול ואז ידברו עם הסרבר נט. משהו כזה.

יש יותר קוד בצד של הסרבר.

כשעושים את החלוקת עבודה – שניים צריכים להיות על הסרבר ואחד על הקליינט.

בצד של הסרבר – צריך לנהל את כל המשתמשים, לדעת להוסיף להוריד לשנות סטטוס וכן הלאה, כנ"ל לגבי הקבוצות, כל יוזר באיזה קבוצה הוא נמצאת, וכן הלאה. הרבה דברים. צריך גם לנהל את הPULL של הכתובות IP. יש כאן עבודה.

יש מודולים בסיסיים: PROTOCOL, SERVERNET, CLIENTNET, MANAGESERVER, MANAGECLIENT.

יש MAIN – הוא עושה CREATE, RUN, וDESTROY. זהו. הוא לא חלק מהאפליקציה ממש. הוא רק מתניע את האפליקציה וסוגר בסוף. אחד לסרבר MANAGER ואחד לקליינט MANAGER.

בקליינט יש 3 MAIN – הראשי, והחלונות הקטנים של הקבוצות.

מבנה נתונים זה גם מודול. יהיה למשתמשים, לקבוצות, לכתובות IP.

מבנה כתובות IP – תור. בדרך כלל בעולם התקשורת, פחות מקובל שמשהו משתחרר ישר להשתמש בו.

מבנה לקבוצות – HASH MAP. חיפוש, הוספה והסרה. הגודל – הלימיט של הקבוצות. צריך להגדיר קובץ H שנקרא LIMITS, שמכיל את כל הLIMITATIONS – שיהיה מוגדר במקום אחד. גודל מקסימלי של שם משתמש, סיסמא, וכן הלאה.

מבנה למשתמשים – HASH MAP. חיפוש, הוספה והסרה. להגדיר לימיט של כמה משתמשים יכולים להיות בזמן נתון.

ערך הASCII סך הכל כפול המיקום של כל ערך במחרוזת. ועל כל זה לעשות מודול של הSIZE שלנו.

בקליינט – הUI במודול נפרד. יכול להיות שזה מספיק.

**מודולים**

מודול סרבר נט – מה שעשינו אתמול, מי שמדבר מולו זה רק ה-SERVER MANAGER

מודול של המשתמשים – יהיה פה את ה-HASH MAP של המשתמשים וכל מה שקשור לזה.

מודול של הקבוצות – יהיה פה את ה-HASH MAP של הקבוצות וכל מה שקשור אליהן.

מודול כתובות IP – מה שמנהל את התור של הIP.

קובץ H של מודול כתובות IP: פונקציה שיוצרת את התור - ליצור את התור ולהכניס 100 כתובות IP. פונקציה שמסירה איבר מהתור ומחזירה את הIP הזה. פונקציה שמוסיפה איבר לתור – IP שהוסר (נקרא לזה מתי שאנחנו סוגרים קבוצה אז הIP שלה חוזר לPULL). פונקציית DESTROY לתור הזה.

**מודול של הפרוטוקול** – מתרגם הודעה מצד אחד לשני.

REGISTRATION REQUEST ו-REGISTRATION REPLY.

LOGIN REQUEST ו-LOGIN REPLY

CREATE GROUP REQUEST ו-CREATE GROUP REPLY

JOIN GROUP REQUEST ו-JOIN GROUP REPLY

LEAVE GROUP REQUEST ו-LEAVE GROUP REPLY

EXIT REQUEST ו-EXIT REPLY?

LOG OUT REQUEST ו-LOG OUT REPLY

לכל TAG יש UNPACK וPACK. הLENGTH – הגודל הכולל של כל מה ששמור בבאפר.

int MsgReqPack(char \_s[], char \_buffer[])

int MsgReqUnpack(char \_buffer[], char \_s[])

int MsgReplyPack(char \_s[], char \_buffer[])

int MsgReplyUnpack(char \_buffer[], char \_s[])

\*להבין אם בENUM זה שמור כINT או כCHAR

**מודול סרבר MANAGER** – הוא היחיד שמדבר עם הסרבר נט, והוא היחיד שמדבר עם הפרוטוקול מהצד של הסרבר. הוא מדבר עם כל המודולים של הסרבר, יהיה בו פונקציות מעטפת לכל פעולה שאפשר לעשות מול הסרבר והוא יקרא לפונקציות הרלוונטיות מכל המודולים ויחזיר תשובה ל-CLIENT MANAGER.

ServerManagerCreate() – CREATE לSERVERNET. כנראה יהיה כאן גם CREATE לGROUPS ולUSERS. ליצור מבנה של סרבר.

הGOTMSG – שיהיה בSERVERMANAGER.

שיהיה פונקציה בפרוטוקול שמחזירה את הTAG. וברגע שיש אותו, לדעת לשייך את ההודעה למה היא.

ServerManagerRun() – קוראים לRUN של SERVERNET.

**מודול סרבר MAIN** – יוצא SERVER MANAGER, עושה לו RUN, ועושה לו DESTROY.

**מודול קליינט נט** – כאן יהיה CREATE, RUN, DESTROY לקליינט.

int CreateClientNet()

int CloseClientNet()

int ConnectClientNetWithServer(int \_clientSocket) – סטטי, בתוך הCREATE

int SendToServerNet(int \_listeningSocket, char \_data[], size\_t \_dataLen)

int RecieveFromServerNet(int \_listeningSocket, char \_data[], size\_t \_dataLen)

**מודול קליינט MANAGER** – זה שמנהל כל מה שקשור לקליינט. הוא היחיד שחשוף לקליינט נט ולפרוטוקול.

ClientManagerCreate – הוא עושה CREATE ל-CLIENTNET. זה מבצע SOCKET, CONNECT, ומחזיר את מספר הSOCKET שהוא קיבל.

ClientManagerRun – קוראת לפונקציית UI שמדפיסה את האופציות הרלוונטיות למסך הזה, וקולטת את מה שחזר ולפי מה שחזר מפעילה את אחת מהפונקציות האחרות.

ClientManagerDestroy – קורא לפונקציה CLOSE של CLIENTNET.

RegisterUser() – סטטי. הוא קורא לUI בשביל char \_userName[], char \_password[], וקורא לCREATECLIENTNET של CLIENTNET בשביל הסוקט. הוא יקרא לפונקציה של הפרוטוקול שעוטפת את ההודעה. וגם לפונקציה של SEND TO SERVER. וגם ל-RECEIVE FROM כדי לוודא שהכל עבר בסדר.

UserLogin() – סטטי. לקרוא לפונקציה UI שתביא לו את השם משתמש וסיסמא. אז פונקציה של הפרוטוקול שעוטפת את זה. ואז לשלוח לסרבר ולחכות לתשובה מהסרבר.

UserLogOut() – סטטי. יוצא מהקבוצות שהקליינט היה בהם.

Exit() – מנתק את הקשר עם הסרבר – CLOSECLIENTNET.

קליינט ברמת התקשורת – מתחבר לסרבר בעלייה, מתנתק מהסרבר ביציאה. ברמה האפליקטיבית

צריך לשמור אצלו בCLIENT – את הקבוצות שאליהן הוא מחובר – שם קבוצה וPID של שתי החלונות שנפתחו, ושם המשתמש.

מבנה של USER – שם משתמש וסיסמא, האם הוא אקטיבי או לא – רק בLOGIN הוא הופך להיות אקטיבי, בLOGOUT הוא הופך להיות פסיבי. צריך רשימה של כל שמות הקבוצות שהוא מחובר אליהם. אולי צריך גם USERID.

כשאני עושה EXIT, לא מגיעה הודעה לסרבר. הסרבר יסגור את עצמו בגלל שגיאה. לא בגלל קליינטים.

כשהסרבר עולה הוא קורא שם משתמש וסיסמא ומכניס אותם ל-HASH MAP של היוזר.

מודול אחד עם פונקציה שמאזינה לערוץ של הקבוצה וכל פעם שמתקבלת הודעה היא מעבירה את זה הלאה.

מודול אחד עם פונקציה שמאזינה לערוץ של הקבוצה וכל פעם שמקבלת הודעה היא מדפיסה אותה לטרמינל.

מודול קליינט MAIN – עושה CREATE, RUN, DESTROY לקליינט MANAGER.

מודול UI – כל דבר של הדפסה יעבור פה.

קובץ H עבור על הLIMITATIONS

צריך לתפעל את המצב שאם הסרבר ירד ועולה מחדש, כל מי שנרשם כבר הוא נשאר רשום. רק הפרטים של המשתמשים, לא קבצים.

CLIENT MANAGER – מנהל את האפליקציה של הקליינט. כל הממשק משתמש ברמת הMANAGEMENT יהיו במקום אחד – במודול של UI. יש CLIENT NET. יש שני חלונות שנפתחים לקבוצה. אולי יהיה גם צורך ברמת הקליינט להחזיק מבנה שיחזיק את כל הקבוצות שקליינט מחובר אליהם ברגע נתון.

מודול של הפרוטוקול שמשותף לסרבר ולקליינט MANAGEMENT.

בסרבר:

יש את הסרבר נט. יש את הMANAGER – הוא מנהל את האפליקציה של הסרבר. רק הוא מדבר מול הפרוטוקול מהצד של הסרבר. תחת הסרבר MANAGER, יש GROUP MANAGER ו-USER MANAGER, הMANAGER של הGROUP מטפל את כל מה שקשור לטיפול בקבוצות, וה-USER MANAGER מטפל בכל מה שקשור למשתמש. השמירה לקובץ וקריאה מקובץ נמצא ב-USER MANAGER.

תחת ה-GROUP MANAGER: הוא צריך לנהל מודול שאחראי על ייצוג קבוצה – המבנה של GROUP, וכל מה שקשור לשם. יש שם CREATE, DESTROY ועדכונים אליו – להוסיף יוזר, להסיר יוזר וכן הלאה.

HASH MAP של הGROUP – בVALUE של כל נוד יהיה את הGROUP מהמודול של הGROUP, ובKEY יהיה את השם של הקבוצה. התור יהיה גם תחת ה-GROUP MANAGER, והוא יכיל את הכתובות IP.

USER MANAGER – בוא יהיה מודול לUSER, ואת ה-HASH MAP שיש בו בVALUE את המבנה של הUSER, ובKEY את שם המשתמש.

מה GROUP צריך לשמור? הוא צריך COUNTER – לדעת שברגע שאין משתמשים, לסגור את הקבוצה.

מה USER צריך לשמור? צריך רשימה של כל הקבוצות שיוזר מחובר אליהם, כי כשעושים LOG OUT צריך להוציא אותו מכל הקבוצות שהוא מחובר אליהם.

ה-CLIENT צריך לשמור לעצמו אינפורמציות מסוימות. כשהוא מצטרף לקבוצה, נפתחים לו שתי חלונות, שלב מסוים, כדי לסגור אותם, כשהוא עושה LEAVE GROUP, הוא צריך לדעת מה הPID של כל אחד מהPROCESSים האלה, אז צריך לשמור את הPID של כל החלונות שנפתחו לדוגמא. צריך פנימית לשמור כל מה שרלוונטי לניהול בצד של הקליינט בלי קשר למה שיש בצד של הסרבר.

איך לגשת לדבר הזה? לעשות את מה שהתחלנו. HIGH LEVEL DESIGN, מה יש בכל קובץ H בייסיקלי. השלב הבא – להקים תשתית בסיסית. יש CLIENT MANAGER, SERVER MANAGER, SERVER NET ו-CLIENT NET, בסיס של הפרוטוקול ולראות שהדברים הכי בסיסיים עובדים.

השלב הבא – FLOW של הרשמה. הUI כבר משחק תפקיד. עדיין לא צריך GROUP – אבל צריך USER MANAGER ומה שקשור אליו.

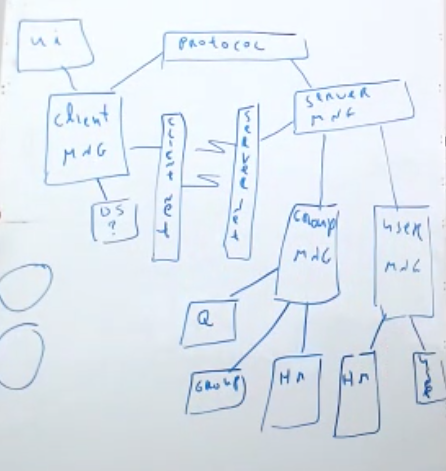
LOG OUT – להשאיר לסוף, זה הFLOW הכי מורכב. גם את ההצטרפות לקבוצה אפשר לחלק לשניים – כשיודעים לאיזה קבוצה רוצים להצטרף, ורק אחר כך להוסיף את הצורך של לקבל רשימה של קבוצות ולבחור מתוכם מה שרוצים וכן הלאה. מומלץ לא לכלול את יום שני בעבודה.

חובה לעשות אינטגרציות. כל הזמן לבדוק שזה עובד.

כשמתחילים לעבוד על קבוצות, אז אחד צריך להתחיל לטפל בעניין של השתי חלונות שנפתחות.

היעד של היום – תשתית בסיסית שעובדת. רצוי גם להתחיל את הREGISTRATION. לחשוב על ה-HIGH LEVEL DESIGN של מה שאני אשתמש בו היום.

לשתף קבצים בדרייב בגוגל.



לשאול על ה-SERVER MANAGER:

איפה לשים את המספר של הפורט והמספר IP של הסרבר?