

# עבודת חקר במדעי המחשב

## Secure file sharing

יוני 2023

### פרטי העבודה:

- שם בית הספר: הכפר הירוק ע"ש לוי אשכול
- שם העבודה: אפליקציה מאובטחת לשיתוף קבצים בזמן אמת בין משתמשים.
- שם התלמיד: יואב קילשטיין
- תעודת זהות: 328234034
- שם המנחה: יהודה אור
- שם החלופה: הגנת סייבר ומערכות הפעלה
- תאריך הגשה: 10.6.2022



# YKFILESHARE

Secure file sharing

# תוכן העניינים

4	מבוא
5	יזום
5	תיאור ראשוני של המערכת:
5	קהל היעד:
5	יעדי הפרויקט:
5	בעיות, תועלות וחסכונות:
5	סקירה של פתרונות קיימים:
6	סקירת הטכנולוגיה של הפרויקט:
6	תיחום הפרויקט:
7	פירוט תיאור המערכת (אפיון):
7	יכולות משתמש:
8	בדיקות מתוכננות (קופסה שחורה):
9	סיכויי אבטחה ואמינות נתונים נוספים המטופלים ללא קשר לבדיקות:
9	ניהול סיכונים:
9	לוח זמנים להגשה:
11	תיאור תחום הידע - פרק מילולי
12	פירוט מעמיק של היכולות שהוצגו בשלב הקודם ומעבר:
15	מבנה/ארכיטקטורת הפרויקט
16	תיאור זרימת המידע במערכת בין הרכיבים (המחלקות) השונים:
16	תיאור החומרה – רכיבים שונים והקשרים ביניהם
18	תיאור בעיות מרכזיות ושיקולים בבניית הפרויקט:
18	אלגוריתם מרכזי בפרויקט:
19	השיקולים בבחירת: Firebase
22	סביבת העבודה - Visual studio code
23	ReactJS
24	Firebase
25	Hostinger
26	תקשורת שרת לקוח מבוססת Sockets ו-Socket.IO
28	רשתות
28	WWW-
29	HTTP/HTTPS:
30	WebSocket:
32	TCP/IP:
33	DNS:
34	TLS:
35	FTP:
36	תיאור מסכי המערכת:

36.....	1. עמוד הכניסה:
36.....	2. עמוד ההרשמה:
37.....	3. עמוד הבית/הצאט:
40.....	Screen Flow Diagram
41.....	מסד ומבני הנתונים-
41.....	מסד נתונים: Firebase/Firestore
41.....	"chats" - מאחסן את כל המידע הקיים על כל צאט בין משתמשים:
42.....	"userChats": מאחסן עבור כל משתמש את אוסף הצאטים שלו:
43.....	"users": מאחסן את פרטי המשתמש
44.....	מבני הנתונים בפרויקט:
46.....	סקירת חולשות והאיומים:
48.....	<b>מימוש הפרויקט</b>
49.....	חלק א' - סקירת המודולים והמחלקות
49.....	מודולים/מחלקות מיובאים:
49.....	react:
49.....	firebase:
50.....	socket.io:
50.....	node.js:
50.....	מודלים/מחלקות שפיתחתי:
50.....	Pages:
52.....	Context:
53.....	Components:
56.....	App.js:
56.....	Index.js:
56.....	style.scss:
56.....	server.js:
58.....	חלק ב' - פתרון הבעיה האלגוריתמית
61.....	חלק ג - מסך בדיקות מלא
61.....	בדיקות מתוכננות:
62.....	בדיקות נוספות:
64.....	עץ קבצים מרכזיים:
64.....	דרישות מקדימות לפני שימוש-
65.....	משתמשי המערכת:
72.....	תמונות בפרויקט:
74.....	קוד הפרויקט:

# מבוא

## **ייזום-**

### **תיאור ראשוני של המערכת:**

מטרת הפרויקט היא לפתח אפליקציה מבוססת web בשם "YKFileShare" המאפשרת למשתמשים לשתף קבצים בצורה מאובטחת. המוצר המוגמר מספר ממשק ידידותי למשתמש שבו אנשים יכולים ליצור חשבון, להיכנס ולבצע פעולות שונות כגון העלאת קבצים, גישה לקבצים המשותפים איתם ושהם שיתפו בעבר בזמן אמת עם משתמשים אחרים. הרציונל מאחורי פרויקט זה הוא לספק פלטפורמה חנימית לשיתוף או שמירת קבצים, המספקת מענה למשתמשים הדורשים חוויה אינטואיטיבית ונוחה. האתגרים שניתן לצפות בפרויקט זה כוללים את אבטחת המידע, הטמעת עדכונים ופעילות בזמן אמת וניהול אימות המשתמשים.

### **קהל היעד:**

המערכת מיועדת בעיקר לאנשים פרטיים אך גם לארגונים הזקוקים לפלטפורמה אמינה ומאובטחת לשיתוף קבצים ללא עלות. זה יכול להיות מנוצל על ידי אנשי מקצוע בתעשיות שונות, סטודנטים המשתפים פעולה בפרויקטים, או כל אדם אחד שרוצה לשתף קובץ או לשמור אותו.

### **יעדי הפרויקט:**

- לאפשר למשתמשים ליצור חשבונות ולהתחבר אליהם בצורה מאובטחת.
- לספק ממשק ידידותי להעלאת, שיתוף ושליחת קבצים.
- הטמעת עדכונים בזמן אמת לתקשורת חלקה ומהירה בין משתמשים.
- לודא אבטחת נתונים ופרטיות של קבצים משותפים ופרטי משתמשים.
- להקל על שיתוף פעולה יעיל ושיתוף של קבצים בין משתמשים.

### **בעיות, תועלות וחסכונות:**

הבעיה שהמערכת מטפלת בה היא הצורך בפלטפורמה מאובטחת, חנימית וברורה לשיתוף קבצים (השימוש בה ברור וחד משמעי).

### **היתרונות של המערכת כוללים:**

- פרודוקטיביות משופרת באמצעות שיתוף קבצים יעיל ותקשורת בזמן אמת.
- אמצעי אבטחה משופרים להגנה על קבצים משותפים ונתוני משתמשים.
- שיתוף פעולה פשוט יותר בין יחידים, הפחתת הצורך בשיטות שיתוף קבצים מורכבות ומפוצלות.
- חיסכון בזמן ובעלויות על ידי ביטול הצורך בהעברת קבצים פיזית או הסתמכות על פלטפורמות שיתוף קבצים של צד שלישי.
- הפחתת הסיכון לאובדן קבצים או אי מיקומם.

### **סקירה של פתרונות קיימים:**

ניתן להשוות את הפרויקט ליישומי שיתוף קבצים ותקשורת קיימים כמו Dropbox, Google Drive ו-Slack. עם זאת, המערכת שואפת לספק חוויה ממוקדת יותר המותאמת במיוחד לשיתוף קבצים בזמן אמת. הוא

משלב את הפונקציונליות של שיתוף קבצים לפלטפורמה אחת וברורה, מציע פתרון מגובש ונוח יותר למשתמשים.

### סקירת הטכנולוגיה של הפרויקט:

הטכנולוגיות המשותפות בפרויקט כוללת את React, Firebase (כולל, Firebase Authentication, Firestore ואחסון), וספריית ה-React Router לניווט. React היא ספריית JavaScript פופולרית לבניית ממשקי משתמש, בעוד ש-Firebase מספקת תשתית ושירותים שונים לאימות, אחסון וניהול מסדי נתונים. מעבר לזה השתמשתי ב-socket.io וכתבתי שרת למערכת שיאפשר למשתשים לדווח במידה ונשלח אליהם קובץ המכיל וירוסים. בעוד שטכנולוגיות אלו מבוססות היטב ונמצאות בשימוש נרחב, חשוב להבטיח הבנה נכונה וניצול של טכנולוגיות אלו כדי למקסם את הפוטנציאל שלהן ולהתגבר על כל מגבלה או אתגרים שהם עלולים להציג. בנוסף, השתמשתי ב-hostinger לקניית שם דומיין ואחסון האתר ברשת.

- מגבלה למערכת: מגבלות הנובעות מהתוכנית ללא עלות של firebase שבה אני משתמש.

[ראה פירוט.](#)

### תיחום הפרויקט:

הפרויקט מתמקד בעיקר בפיתוח אפליקציה מבוססת אינטרנט לשיתוף קבצים ותקשורת, בדגש על היבטי ממשק המשתמש, אבטחת המידע והפרטיות.

- **העברת נתונים מאובטחת:** הפרויקט משלב העברה מאובטחת של נתונים בין הלקוח (המכשיר של המשתמש) לבין השרת המארח את האפליקציה. זה כרוך בשימוש בפרוטוקולי תקשורת מאובטחים (כגון HTTPS) כדי להצפין את הנתונים במהלך המעבר ולהגן עליהם מפני גישה לא מורשית.
- **חיבור לרשת:** האפליקציה מסתמכת על החיבור לרשת של המשתמש כדי לגשת לאינטרנט וליצור חיבור עם השרת firebase. היא מנצלת את החיבור של המשתמש כדי להעלות ולהוריד קבצים בזמן אמת. הצגת היכולת ליצור חיבור מתמשך בין הלקוח לשרת. זה מאפשר חילופי נתונים יעילים ומיידיים ללא צורך בסקר מתמיד או רענון ידני.
- **תאימות:** הפרויקט פותח כדי להיות תואם למערכות הפעלה שונות ודפדפני אינטרנט. מטרתו היא לספק חווית משתמש עקבית על פני פלטפורמות מרובות, כגון Windows, macOS, Linux, Android ו-iOS.
- **שילוב מערכת קבצים:** המערכת מקיימת אינטראקציה עם מערכת הקבצים של מערכת ההפעלה כדי לאפשר למשתמשים לגלוש ולבחור קבצים מהמכשירים המקומיים שלהם להעלאה. הוא משתמש בפונקציונליות קלט הקבצים של הדפדפן כדי לגשת לקבצים המאוחסנים במכשיר של המשתמש.
- **היענות בין פלטפורמות:** הפרויקט מבטיח היענות לגדלי מסך ורזולוציות שונות, בהתחשב בווריאציות בין מכשירים ומערכות הפעלה. הוא משתמש בטכניקות עיצוב רספונסיבי כדי להתאים את ממשק המשתמש למכשירים שונים, כגון מחשבים שולחניים, מחשבים ניידים, טאבלטים וסמארטפונים.
- **שימוש ב-threads:** הפרויקט עושה שימוש בתהליכונים כדי להשיג ביצוע במקביל בתוך האפליקציה. היבט זה מדגים את ההבנה והיישום של מושגי מערכת הפעלה, הקשורים ספציפית לניהול תהליכונים, תזמון וסנכרון.

- **ביצוע אסינכרוני:** הפרויקט ממנף טכניקות תכנות אסינכרוניות, כגון שימוש ב-`setTimeout` בהשהייה מינימלית של מספר אלפיות שניות, כדי לבצע משימות מסוימות במקביל. זה מאפשר ביצוע של פעולות רשת, כגון שאילתות `Firestore`, תוך כדי עיבוד משימות אחרות, מה שמספר את ההיענות והיעילות של המערכת. משתמש בתהליכים כדי לבצע פונקציות באופן אסינכרוני בשרשרים נפרדים. זה מאפשר ביצוע במקביל של משימות, כולל פעולות פוטנציאליות הקשורות לרשת.
- **שילוב `Socket.io`:** הפרויקט משתמש ב-`socket.io`, תקשורת נתונים לפי מודל שרת לקוח מבוסס סוקטים, כדי לאפשר למשתמשים לשלוח בקשות לשרת (כהודעות) על משתמשים שהעבירו להם תכנים חשודים/עם וירוסים.
- ניהול וסנכרון של מצבי רכיבים עם נתונים מרוחקים בזמן אמת, ומאפשר תקשורת רשת וחילופי נתונים בין האפליקציה בצד הלקוח לבין השרת בעזרת מסד הנתונים בזמן אמת של `firebase` אשר עושה שימוש ב-`web sockets`.

#### - אין מימוש עצמי של פרוטוקולים כמו `HTTP`, `TCP`, `DNS` או `DHCP`.

עם זאת, חשוב לציין כי פרוטוקולים כמו אלו הם מרכיבים בסיסיים של תקשורת רשת. הם ממלאים תפקיד מכריע בהפעלת הפונקציונליות והקישוריות של מערכות ושירותים שונים באינטרנט. אמנם הפרויקט לא עבד ישירות מול פרוטוקולים אלו, אך הוא מקיים איתם אינטראקציה עקיפה דרך המסגרות, הספריות או התשתית הבסיסית עליהן הוא רץ.

לדוגמה, בביצוע בקשות `HTTP` לשירותים חיצוניים, נעשה שימוש בפרוטוקול `HTTP` לבקשות כמו `GET`, `POST`. באופן דומה, כשאתר מתקשר עם שרת מרוחק ברשת, פרוטוקול `TCP` מטפל בהעברת נתונים אמינה בין הלקוח לשרת. פרוטוקול ה-`DNS` אחראי על תרגום שמות מתחם לכתובות `IP`, המאפשר את הרזולוציה של כתובות רשת, ופרוטוקול `DHCP` מסייע בהקצאת כתובות `IP` דינמית בתוך רשת. כלומר, ישנם מספר רב של פרוטוקולים ומודלים בתחום הרשתות שיש צורך בהבנה עמוקה שלהם בכדי לבנות את הפרויקט גם אם לא עובדים עליהם באופן ישיר ורק משתמשים בהם ככלים לפעילות.

## **פירוט תיאור המערכת (אפיון):**

### **יכולות משתמש:**

#### **משתמש רגיל-**

- רישום חשבון: משתמשים יכולים ליצור חשבון על ידי מתן מידע הכרחי כגון שם משתמש, דואר אלקטרוני וסיסמה.
- כניסה/יציאה: משתמשים יכולים להיכנס כדי לגשת לחשבון שלהם ולהתנתק בסיום.
- העלאת ושיתוף קבצים: משתמשים יכולים להעלות קבצים מהמכשירים המקומיים שלהם למערכת ולשתף אותם עם אחרים על ידי יצירת קישורים הניתנים לשליחה (נעשה באופן אוטומטי על ידי המערכת).
- שמירת קבצים: משתמשים יכולים לשמור קבצים לשימוש אישי ולהוריד אותם מתי שירצו, תחת המשתמש שלהם בצאט עם עצמם.
- פתיחת צאט חדש: משתמשים יכולים לחפש שמות משתמשים של משתמשים רשומים אחרים, למצוא אותם, ללחוץ על שמם ובכך לפתוח איתם צאט שבו יכולו לשתף את הקבצים אחד עם השני.

- גישה לשיחות ישנות ומעבר על הצאטים השונים שהם חלק מהם: כל משתמש יכול לגשת לכל שיחה שהייתה לו עם משתמש רשום אחר בלחיצה על שמו שיופיע לו בסרגל בצד, לאחר הבחירה הצאט בניהם יפתח לו והוא יוכל לגלול בשיחותיהם.
- דיווח על משתמש: משתמשים יכולים לדווח לשרת אם קיבלו קובץ חשוד/אחד שמכיל וירוסים והשרת בתגובה ישלח להם הודעת אישור שקיבל את הדיווח ואם אכן זה יהיה קובץ כזה המשתמש ימחק מהאתר לצמיתות.

### מנהל המערכת והשרת-

- צפייה בדיווחי משתמשים: משתמשים יכולים לדווח לשרת אם קיבלו קובץ חשוד/אחד שמכיל וירוסים והשרת בתגובה ישלח להם הודעת אישור ואם אכן זה יהיה קובץ כזה המשתמש ימחק מהאתר.
- ניהול מסד נתונים: למנהל המערכת יש גישה ל-Firebase והוא יכול לבצע משימות ניהוליות, כגון ניהול חשבונות משתמש, מחיקת צאטים, מעקב אחר השימוש במערכת ועוד (בצורה חיצונית, לא קשור ישירות לאתר).

### בדיקות מתוכננות (קופסה שחורה)-

- **בדיקת העלאת קבצים:** בודקים אם משתמשים יכולים להעלות בהצלחה קבצים מסוגים וגדלים שונים ולוודא שהקבצים מאוחסנים כהלכה ב-Firebase Storage - מנסים לפתוח משתמש ולהעלות איתו מספר רב של קבצים שונים בגדלים שונים.
- **בדיקת שיתוף קבצים:** בודק את פונקציונליות שיתוף הקבצים של המערכת ודא שהקבצים המשותפים נגישים בצאט ושהקישור שנוצר בהצלחה (לאחר שוודאנו זאת בבדיקה הקודמת) מתקבל אצל המשתמש השני בצאט.
- **בדיקת הורדת קבצים:** ודא ששני המשתמשים בצאט יכולים להוריד קבצים שנשלחו מאז תחילת שיחתם באמצעות קישורי ההורדה שסופקו, זאת כדי לוודא שהקבצים שהורידו הם שלמים וזהים לקבצים המקוריים.
- **בדיקת אימות משתמש:** בדוק את תהליך אימות המשתמש של המערכת על ידי בדיקה שמשתמשים יכולים להירשם, להיכנס ולהתנתק בהצלחה. במקביל, לבדוק ניסיון להכניס ערכים שגויים ומתקפות שונות על עמודי הכניסה/ההרשמה לדוגמת sql injection, brute force ועוד. נמנע עם חסימת גישה למשתמש אחרי מספר ניסיונות כניסה שגויים ברצף וניהול מסד נתונים שהוא noSql לא מאפשר sql injection.
- **בדיקת עדכונים בזמן אמת:** לבדוק שהכל מתעדכן בזמן אמת במגוון דרכים. לדוגמה: פתיחת צאט חדש בין שתי משתמשים כשהמחשבים סמוכים ולראות אם זה קורה בזמן אמת העדכון אצל המשתמש שפתחו איתו (לא האחד שפתח).
- **בדיקת שליחת הודעות לשרת והתגובה ללקוח:** לבדוק שהתקשורת עם השרת תקינה ושכפתור הדיווח אכן עובד ושהמשתמשים יכולים להרגיש בטוחים ולדעת במקרה הצורך יכולו לדווח על קבצים חשודים.



## סיכוני אבטחה ואמינות נתונים נוספים המטופלים ללא קשר לבדיקות:

- **גישה לא מורשית לנתוני המשתמש** - הצפנת נתוני משתמש רגישים כדי למנוע גישה בלתי מורשית דרך פונקציות להצפנת סיסמאות כאשר לבסוף נבחר לנהל את תהליך ההרשמה עם firebase.
- **צפייה של אדם לא רצוי באופן פיזי במידע רגיש** - לדוגמה, משתמש מדבר עם הבוס שלו בעבודה והבוס מעביר לו קובץ עם מידע מסווג בעמוד הראשון "מסווג.pdf". העובד פותח את הצאט כי ראה שקיבל הודעה חדשה אך מישהו עובר ליד וצופה בקובץ המסווג (כי לרוב מוצגת תמונה של העמוד הראשון של הקובץ) - חשבתי על מספר דרכים לפתור בעיה זאת כמו הוספת הודעת אזהרה לפתיחת הצאט כאשר אתה לבד אך לבסוף בחרתי בהמרת הקובץ לקישור לחיץ שמורד למחשב רק כאשר לוחצים עליו ולא ניתן לצפות בו לפני.
- **שליחת קבצים זדוניים להורדה** - ניתן לדווח על כל קובץ שנראה חשוד/זדוני דרך לחיצה על כפתור שמעביר בקשה לשרת לבדוק את השיחה.
- **אובדן נתונים:** השתמשתי במנגנון אחסון אמין המעביר את המידע על גבי פרוטוקול TCP המבטיח את העברת המידע בהצלחה ברשת כדי למזער את הסיכון לאובדן נתונים.
- **כשלים בשרת או באחסון המשפיעים על זמינות הקבצים:** כדי לפתור בעיה זאת הבנתי שאני חייב להשתמש במקור אמין שניתן לבטוח בביצועים שלו לכן השתמשתי בfirebase של גוגל.

### ניהול סיכונים:

- **תקציב** - לקנות שם דומיין ולאחסן את האתר על שרתים עולה כסף ויכול היה למנוע ממני לממש את הפרויקט בעולם האמיתי. כדי להקל על בעיית התקציב, מצאתי כלי במחיר סביר שיכול לדאוג לי לשם דומיין ואחסון אתר אמין - Hostinger. השתמשתי בסיומת זולה יותר מאשר com. שמתאימה לפרויקט ובכך שילמתי פחות על אחסון האתר "ykfileshare.tech".
- **עמידה בזמנים** - לקראת סוף השנה, הייתי בלחץ גדול מהלימודים ועיסוקים אחרים עד כדי כך שלרגע לא הייתי בטוח שאצליח לעמוד בדרישות ההגשה. מכיוון שזיהיתי את הבעיה הזאת בשלב יחסית מוקדם בניתי לעצמי תוכנית עבודה מסודרת ביומן בשלבים ברורים כאשר המטרה הייתה לסיים את הכל כמה שיותר מוקדם מבלי לפגוע במחויבויות האחרות שלי.
- **תיקוני באגים לאחר העלאת האתר** - חששתי שכאשר אעלה את האתר לאינטרנט יהיו משובים מלקוחות שיגרמו לי להבין שיש תיקונים שאני צריך לעשות בפרויקט. לשמחתי, עם כלי אחסון האתרים שבו השתמשתי, להעלות מחדש את האתר עם תיקונים ושינויים נדרשים זה תהליך פשוט שלוקח מספר דקות.

### לוח זמנים להגשה:

#### **לוח זמנים ראשוני:**

ספטמבר 2022 - למידת רקע תיאורטי על התחום הנבחר והתנסות עם מיני פרויקטים.  
 נובמבר 2022 - הבנת הדרישות ופיתוח רעיוני של המערכת הסופית על בסיס הדרישות.

דצמבר 2022: פיתוח

מרץ 2023: בדיקות והרצות, חידוד ממשק המשתמש ותיקונים אחרונים.

אפריל 2023: כתיבת תיק פרויקט

יוני 2023: הגשה

#### **לוח זמנים בפועל:**

אוקטובר 2022: למידת רקע תיאורטי על התחום הנבחר

נובמבר 2022: התנסות עם מיני פרויקטים ולימוד על reactjs.

דצמבר 2022: פיתוח ממשק משתמש וחיבור עם מסד הנתונים

מרץ 2023: בדיקות, הרצות ותיקונים אחרונים

אפריל 2023: כתיבת תיק פרויקט

יוני 2023: הגשה

# תיאור תחום הידע - פרק מילולי

## פירוט מעמיק של היכולות שהוצגו בשלב הקודם ומעבר:

### שם היכולת: הרשמה למערכת

- מהות: מאפשר למשתמשים חדשים ליצור חשבון חדש באמצעות הזנת פרטים נדרשים.
- אוסף יכולות נדרשות:
  1. ממשק משתמש - עמוד הרשמה
  2. קליטת נתונים
  3. בדיקת תקינות נתונים (עצמאית)
  4. בדיקת תקינות נתונים של firebase ויצירת רשומת משתמש חדשה במערכת האימות של Firebase
  5. אחסון מידע משתמש נוסף במסד הנתונים של Firebase Firestore.
  6. אם יש בעיה נכנס ל catch, אם לא מעדכן בזמן אמת את state של המשתמש הנוכחי ונותן גישה לעמוד הבית.
- רכיבים נחוצים: ממשק משתמש, שירות אימות ותקשורת עם Firebase לרישום משתמש, מסד נתונים לאחסון רשומות משתמשים.

### שם היכולת: כניסה למשתמש קיים/יציאה

- מהות: מאפשר למשתמשים להיכנס לחשבונות שלהם ולהתנתק בסיום.
- אוסף יכולות נדרשות:
  1. ממשק משתמש - עמוד כניסה
  2. קליטת נתונים
  3. בדיקת תקינות נתונים (עצמאית) - אם משהו ריק
  4. אימות התאמת מייל לסיסמה במסד הנתונים מנתוני הקלט.
  5. אם יש בעיה נכנס ל catch, אם לא מעדכן בזמן אמת את state של המשתמש הנוכחי ונותן גישה לעמוד הבית.
  6. בממשק משתמש בעמוד הבית - כפתור התנתקות
  7. מפעיל פונקציית התנתקות, שם את המצב של חיבור המשתמש ב false ומעביר לעמוד ההרשמה.
- רכיבים נחוצים: ממשק משתמש, שירות אימות ותקשורת עם Firebase, הגדרת מצב נוכחי שניתן לשנות בזמן אמת הנוגע לחיבור משתמש.

### שם היכולת: העלאת ושליחת קבצים

- מהות: מאפשר למשתמשים להעלות קבצים ולשתף אותם עם אחרים.
- אוסף יכולות נדרשות:
  1. ממשק משתמש - רכיב להזנת הקובץ בעמוד הבית המציג את שם הקובץ וגודלו למשתמש לפני השליחה וגם כפתור send.
  2. אפשרות לבחירת קבצים מהמכשיר המקומי של המשתמש.

3. עדכון מצב נוכחי של קובץ (אם העלו קובץ או לא).
  4. טיפול בתהליך העלאת קבצים.
  5. יצירת קישורים או כתובות URL שניתנים לשליחה והורדה עבור הקבצים שהועלו.
  6. אחסון נתוני הקבצים (שם, גודל, כתובת אתר) במסד הנתונים.
  7. אפשרות הורדה כדי לקישור שנשלח.
- רכיבים נחוצים: ממשק משתמש, רכיב קובץ יחיד, רכיב רשימת קבצים, רכיב הזנת קובץ, שירות אחסון לאחסון קבצים שהועלו.

### **שם היכולת: פתיחת שיחה חדשה**

- מהות: מאפשר למשתמשים לחפש משתמשים רשומים אחרים וליזום צ'אט.
- אוסף יכולות נדרשות:
  1. ממשק משתמש - תיבת חיפוש והזנת טקסט.
  2. הטמעת פונקציית חיפוש בלחיצה על כפתור אנטר - בודק עם מסד הנתונים אם קיים כזה משתמש. תהליך זה נעשה בthread נפרד.
  3. הצגת תוצאת חיפוש (אם יש) למשתמש.
  4. עדכון מצב נוכחי של קובץ (אם העלו קובץ או לא).
  5. אפשרות למשתמשים ללחוץ על שם משתמש כדי לפתוח צאט חדש בדאטה בייס.
  6. עדכונים בזמן אמת למצב רשימת הצאטים של המשתמש עם מאזין חי בפירבייס.
- רכיבים נחוצים: ממשק משתמש, רכיב חיפוש משתמש, רכיב ממש צאט להצגת הצאט הנפתח, תקשורת עם מסד הנתונים.

### **שם היכולת: גישה לצאטים שונים שהמשתמש חלק מהם:**

- מהות: מאפשר למשתמשים לגשת ולעבור על צאטים שהם חלק מהם
- אוסף יכולות נדרשות:
  1. עדכון במצב אמת באמצעות מאזין חי לאוסף הצאטים שהמשתמש הנוכחי חלק מהם.
  2. ממשק משתמש להצגת רשימת הצאטים.
  3. תקשורת עם מסד הנתונים ועם מאזין חי שפועל כאשר המשתמש לוחץ על שיחה אחרת - לעדכון כל הצאט שיש להם השמור במסד הנתונים במקום הצגת הצאט הקודם.
- רכיבים נחוצים: ממשק משתמש, רכיב הודעות, רכיב סרגל צד, מסד נתונים.

### **שם היכולת: דיווח על קבצים לשרת**

- מהות: מאפשר למשתמשים לדווח על קבצים חשודים או על משתמשים אשר מעבירים קבצים זדוניים.
- אוסף יכולות נדרשות:
  1. ממשק משתמש להצגת כפתור הדיווח.

2. תקשורת שרת לקוח מבוססת סוקטים בעבור שליחת הדיווחים.
3. קבלת המידע באשר למשתמש המדווח והמשתמש המדווח.
4. גישה למסד הנתונים (firebase) כדי לצפות בקבצים שנשלחו בשיחה.
5. ניהול המשתמשים באפליקציית שיתוף הקבצים (כדי למחוק משתמש במקרה הצורך).
- רכיבים נחוצים: ממשק משתמש, רכיב צאט, מסד נתונים, שרת.

כך נראה בשרת:

```
Received message from client: cooluser is reporting: try
```

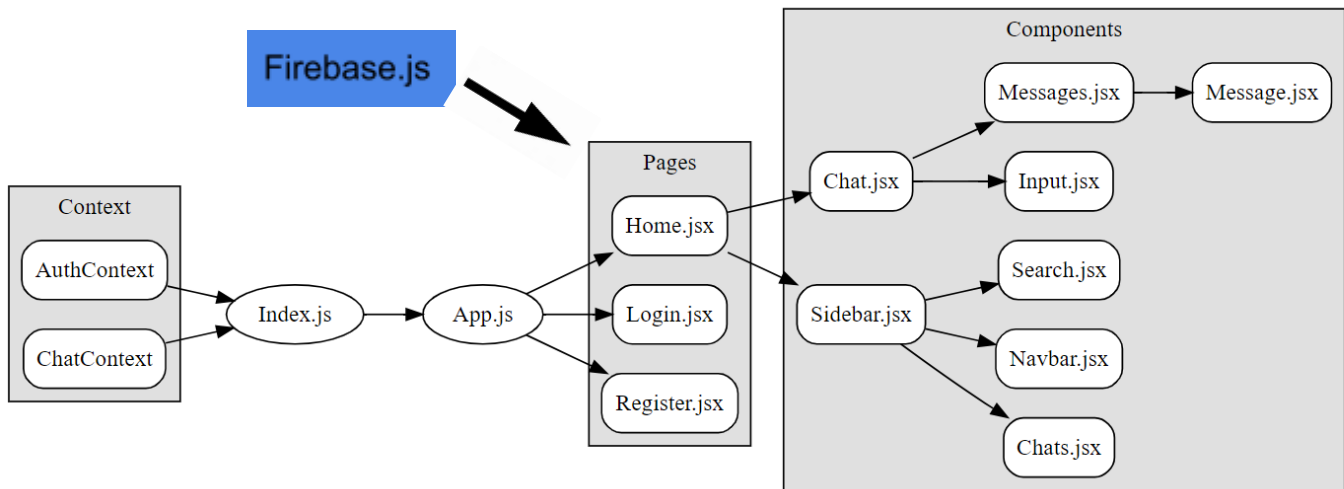
כך נראה אצל הלקוח:

The report was successfully received. If the user did upload a malicious file, he will be kicked out of the site!

[Chat.jsx:29](#)

# מבנה/ארכיטקטורת הפרויקט

## תיאור זרימת המידע במערכת בין הרכיבים (המחלקות) השונים:



## תיאור החומרה – רכיבים שונים והקשרים ביניהם

- **שרתים לאחסון האתר:** שרת אינטרנט כדי לארח את האתר להעלות לאינטרנט את הקבצים והדפים למשתמשים.
- **אחסון קבצים ומסד נתונים:** מערכת אחסון קבצים כדי לאחסן את הקבצים המשותפים למשתמשים. מעבר לכך מערכת אחסון של firebase לשמירת נתוני המשתמשים.
- **ציוד רשת פיזי:** כגון נתבים ומתגים, כדי לחבר בין המחשב הביתי לשרת האינטרנט, שרת מסד הנתונים ומערכת אחסון הקבצים שלך ואותם לאינטרנט ולהקל על התקשורת ביניהם.
- **מכשירי לקוח:** משתמשים באתר שיתוף הקבצים שלך היו ניגשים אליו באמצעות המכשירים של הלקוחות השונים, כגון מחשבים שולחניים, מחשבים ניידים, סמארטפונים או טאבלטים.

מכשירי לקוח אלו משתמשים בדברים שבהם משתמשת כל מערכת - ה-CPU (יחידת עיבוד מרכזית), GPU (יחידת עיבוד גרפית) ו-RAM (זיכרון גישה אקראית) הם כולם רכיבי חומרה חיוניים להרצת הפרויקט. להלן סקירה קצרה של רכיבים אלה ותפקידיהם:

- **המעבד:** המעבד מכונה לעתים קרובות "המוח" של המחשב. מבצע את רוב החישובים ומשימות כמו עיבוד נתונים, פעולות לוגיות ושליטה על הפעולה הכוללת של המערכת.
- **המעבד של הלקוח אחראי לביצוע הוראות הקוד של אתר שיתוף הקבצים.** הוא מטפל במשימות שונות הנדרשות להפעלת האתר. המהירות והיכולות של המעבד משפיעות על תגובתיות האתר ועל מידת היעילות שבה הוא יכול להתמודד עם פעולות כמו העלאות קבצים, הורדות ועיבוד נתונים.



- המעבד הגרפי: ה-GPU אחראי על רינדור והצגת תמונות, סרטונים וגרפיקה על מסך מחשב. הוא מתמחה בעיבוד במקביל וחשוב במיוחד למשימות עתירות גרפיקה כגון משחקים, עריכת וידאו ועיבוד תלת מימד.

- למרות שאתר שיתוף קבצים עשוי שלא להסתמך במידה רבה על עיבוד גרפי, היבטים מסוימים של ממשק המשתמש של האתר עשויים עדיין להפיק תועלת מה-GPU של הלקוחות. לדוגמה, אלמנטים עשירים מבחינה ויזואלית או תצוגה של הקבצים שנשלחו של תמונה או וידאו, GPU איכותי יכול לשפר את העיבוד של אלה, ולספק חווית משתמש חלקה ומושכת יותר.

- זיכרון RAM (זיכרון גישה אקראית): זיכרון RAM הוא סוג של זיכרון נדיף המספק אחסון זמני לנתונים שה-CPU צריך לגשת אליהם במהירות. הוא מכיל נתונים והוראות שנמצאים בשימוש פעיל על ידי ה-CPU במהלך הפעלת התוכנית. ככל שלמחשב יש יותר זיכרון RAM, כך הוא יכול לאחסן יותר נתונים באופן זמני, וכתוצאה מכך ביצועים טובים ויכולות ריבוי משימות.
- כאשר לקוחות ניגשים לאתר שיתוף הקבצים, המכשירים שלהם מאחסנים באופן זמני נתונים והוראות ב-RAM כדי להקל על פעולת האתר. כמות זיכרון ה-RAM הזמין קובעת את הקיבולת לטפל בפעולות במקביל, כגון העלאה או הורדה של מספר קבצים, ניהול הפעלות של משתמשים ושמירה במטמון של נתונים שניגשים אליהם לעתים קרובות. זיכרון RAM מספיק מאפשר אינטראקציות חלקות ומהירות יותר עם האתר, הפחתת עיכובים ושיפור הביצועים הכוללים.

בהקשר של האתר שבנית, רכיבי החומרה הללו אינם מעורבים ישירות ביישום הקוד. עם זאת, יכולות החומרה של הלקוחות משפיעות באופן משמעותי על יכולתם ליצור אינטראקציה עם האתר ולנצל אותו ביעילות. המעבד, ה-GPU וה-RAM של מכשירי הלקוחות תורמים לביצועים הכוללים, להיענות ולחווית המשתמש בעת השימוש באתר שיתוף הקבצים.

## תיאור בעיות מרכזיות ושיקולים בבניית הפרויקט:

הבעיה בפרויקט זה היא לבנות אפליקציית צ'אט עם אימות משתמש, אחזור והצגה של הודעות בזמן אמת, קבצים מצורפים להודעות, חיפוש משתמשים וניהול מפגשי צ'אט. המטרה היא ליצור חווית צ'אט יעילה וידידותית למשתמש. בכדי להבטיח לעשות זאת באמינות ישנם מספר רב של כלים שהייתי צריך לבחון כדי לבסוף להגיע להכרעה.

## אלגוריתם מרכזי בפרויקט:

לבצע עדכונים בזמן אמת לשינויים שנעשו על ידי משתמשים פעילים באתר ולהראות אותם לאחרים. צריך לקבוע מתי לבדוק בכלל אם יש שינוי (שלא תרוץ סתם פונקציה כל הזמן) - מתי לחכות שהשינוי יקרה, מתי עשות את השינוי, איך להעביר את המידע בין הרכיבים, איך להאזין לקובץ במסד הנתונים ומה הכלי הטוב ביותר להשתמש בו. הייתי צריך לפתח דרך שתעבוד באופן שיטתי וקבוע לביצוע הסנכרונים. אלגוריתמים קיימים לפתרון עדכונים בזמן אמת ובעיות סנכרון כרוכים לרוב בשימוש בארכיטקטורות מונעות אירועים ובפרוטוקולי תקשורת בזמן אמת. כמה גישות נפוצות כוללות:

1. תשאול (polling): גישה זו כוללת בדיקה תקופתית של עדכונים מהשרת במרווחי זמן קבועים. עם זאת, הוא לא יתאים לעדכונים בזמן אמת מכיוון שהוא עושה בדיקה לא תמידים לנתונים.<sup>1</sup> הדגמתי גישה זאת בפרויקט שלי כדי להדגיש את היכולת והאפשרות לעשות זאת ועם זאת את חוסר הכדאיות.
  2. תשאול ארוך: גישה זו כוללת הגשת בקשה לשרת ושמירה על החיבור פתוח עד שלשרת יהיו נתונים חדשים לשלוח. זה מקטין את זמן ההשהיה בהשוואה לסקר, אבל דורש הרבה משאבים.<sup>2</sup>
  3. WebSockets: WebSockets מספקים ערוץ תקשורת דו-כיווני בין הלקוח לשרת, המאפשר העברת נתונים בזמן אמת. הם מתוכננים להיות יעילים עם זמן אחזור נמוך, מה שהופך אותם לבחירה פופולרית עבור יישומים בזמן אמת.
  4. אירועים שנשלחו על ידי שרת (SSE): פרוטוקול תקשורת חד כיווני המאפשר לשרת לשלוח עדכונים ללקוח באמצעות חיבור HTTP בודד. זה מתאים לתרחישים שבהם השרת צריך לדחוף עדכונים ללקוח.
- עבור הפתרון הנבחר, בחרתי להשתמש ב-Firebase Realtime Database עם הפונקציה onSnapshot. גישה זו מאפשרת להאזין לעדכונים בזמן אמת למסמך הרצוי ולעדכן את הרכיבים הרלוונטיים בהתאם בעזרת שימוש ב-WebSockets.<sup>3</sup> בנוסף לכך, עשיתי שימוש ב-useEffect אשר מבטיח שהמאזינים הדרושים יוגדרו כאשר הרכיב נטען ומבוטלים בעת ביטול הטעינה כדי למנוע דליפות זיכרון.

<sup>1</sup> <https://javascript.info/long-polling> (Long Polling, 2022)

<sup>2</sup> <https://javascript.info/long-polling> (Long Polling, 2022)

<sup>3</sup> Introducing Firebase Realtime Database: <https://firebase.google.com/docs/database> (Realtime Database, n.d.)

האלטרנטיבות ל-Firebase לעדכונים בזמן אמת כוללות שימוש במסדי נתונים אחרים בזמן אמת או בניית פתרון מותאם אישית עם טכנולוגיות צד שרת כמו Socket.IO (שבה עשיתי שימוש בעבור יכולת אחרת). עם זאת, Firebase מספקת פתרון נוח ויעיל עם מסד הנתונים בזמן אמת ו-Firebase, יחד עם תכונות נוספות כמו אימות ואחסון, שיכולים לפשט את התהליך גם לשאר הליכי הפיתוח.

## השיקולים בבחירת Firebase:

### אימות משתמש:

- **אימות Firebase:** אימות Firebase מספק פתרון מאובטח וקל לשימוש עבור אימות משתמשים. הוא מציע שיטות אימות שונות, כגון דואר אלקטרוני/סיסמה, מספר טלפון וכניסות למדיה חברתית (למשל, גוגל, פייסבוק, טוויטר). אימות Firebase מטפל בפעולות רישום משתמש, התחברות, איפוס סיסמה וניהול משתמשים.<sup>4</sup>
- **פתרון חלופי:** הטמעת מערכת אימות מותאמת אישית באמצעות מסגרת מסורתית בצד השרת כמו Django או Ruby on Rails.
- **חסרון החלופות:** מערכות אימות מותאמות אישית דורשות מאמץ פיתוח נוסף כדי לטפל בבעיות אבטחה כמו גיבוב סיסמאות, הצפנה והגנה מפני פגיעויות נפוצות. הם חסרים את האינטגרציות המובנות עם ספקי אימות של צד שלישי ש-Firebase Authentication מציע, מה שעלול לגרום לחוויה פחות ידידותית למשתמש.

### אחזור והצגת הודעות צ'אט:

- **מאזין בזמן אמת של firebase firestore:** מציע סנכרון נתונים בזמן אמת באמצעות המאזין בזמן אמת (onSnapshot). אלגוריתם זה מאפשר לאפליקציה לקבל עדכונים בזמן אמת בכל פעם שמתווספות הודעות צ'אט חדשות או שינוי או מחיקת הודעות קיימות. המאזין ממזער בקשות רשת מיותרות ומעדכן ביעילות את ממשק הצ'אט.
- **פתרון חלופי:** סקר את השרת במרווחי זמן קבועים כדי להביא הודעות צ'אט חדשות.
- **חסרון החלופות:** סקר כרוך בהגשת בקשות חוזרות ונשנות לשרת, גם כאשר ייתכן שאין הודעות חדשות. גישה זו פחות יעילה בהשוואה למאזין בזמן אמת של Firebase Firestore, אשר מאחזר נתונים מעודכנים רק כאשר מתרחשים שינויים. סקר יכול להוביל לעומס מוגבר בשרת, תעבורת רשת מיותרת וצריכת סוללה גבוהה יותר במכשירים ניידים.

### שליחת הודעות עם קבצים מצורפים:

---

<sup>4</sup> [Introducing Firebase Authentication](#) (Introducing Firebase Authentication, n.d.)

- **Firestore: Firebase Storage:** מספקת פתרון ייעודי לאחסון והגשה של קבצים מצורפים. הוא מציע תכונות כגון העלאת קבצים מאובטחת, אחסון ניתן להרחבה ושילוב CDN. האלגוריתם כולל העלאת הקובץ המצורף ל-Firebase Storage ואחסון ההפניה המתאימה במסמך Firestore המייצג את הודעת הצ'אט.
- **פתרון חלופי:** אחסון קבצים מצורפים ישירות במסד הנתונים.
- **חסרון החלופות:** אחסון קבצים מצורפים במסד הנתונים יכול לגרום להגדלת גודל מסד הנתונים ולהשפיע על הביצועים הכוללים, במיוחד כאשר עוסקים בקבצים גדולים. Firebase Storage מספק פתרון ייעודי להעלאה, אירוח והגשה של קבצים ניתנים להרחבה, המבטיח אחסון יעיל ואחזור נתונים אופטימלי. זה גם מאפשר מינוף יכולות CDN לאספקה מהירה יותר של קבצים מצורפים למשתמשים.

### חיפוש משתמשים:

- **שאלות ב-Firestore:** מאפשרת לבצע שאלות לאוסף המשתמשים על סמך קריטריונים ספציפיים, כגון שם או שם משתמש. הוא תומך בסינון, מיון ועימוד לחיפוש יעיל של משתמשים. על ידי שימוש ביכולות השאלות של Firestore, האלגוריתם יכול לאחזר משתמשים רלוונטיים על סמך שאלת החיפוש.<sup>5</sup>
- **פתרון חלופי:** הטמעת אלגוריתם חיפוש ידני באמצעות שאלות מסד נתונים עם התאמת טקסט.
- **חסרון החלופות:** יישום אלגוריתם חיפוש ידני יכול להיות מאתגר להשגת תוצאות יעילות ומדויקות, במיוחד כאשר עוסקים במערכי נתונים גדולים של משתמשים. יכולות השאלות של Firestore, יחד עם מערכת האינדקס שלה, מספקות ביצועי חיפוש אופטימליים מחוץ לקופסה. יישום אלגוריתם חיפוש ידני ידרוש מאמץ נוסף לטיפול באינדקס, התאמת טקסט ודירוג משתמשים על סמך רלוונטיות החיפוש.

### ניהול מסד הנתונים:

- **מבנה Firestore:** המבנה המסמכים של Firestore מפשט את ניהול הצ'אטים. כל ששן צ'אט יכול להיות מיוצג כמסמך המכיל הודעות ומטא נתונים.
- **פתרון חלופי:** שימוש במסד נתונים מסורתי של SQL עם עיצוב סכימה יחסי מותאמת אישית.
- **חסרון החלופות:** בעוד שמסד נתונים של SQL יכול להתמודד עם ניהול מפגשי צ'אט, הוא דורש הגדרת סכמות יחסים מורכבות, ניהול קשרי טבלה וכתובת שאלות SQL מותאמות אישית. המבנה מונחה המסמכים של Firestore מפשט את ניהול הפגישות בצ'אט על ידי מתן גישה גמישה ללא סכימה. העדכונים בזמן אמת ופעולות הכתיבה האטומית של Firestore מבטיחים נתונים עקביים בין משתמשים ומפשטים את היישום של תכונות צ'אט בזמן אמת.

<sup>5</sup> [Perform simple and compound queries in Cloud Firestore | Firebase](#) (Perform Simple and Compound Queries in Cloud Firestore | Firebase, n.d.)

**סקירת הבחירה:**

הפתרונות שנבחרו ממנפים שירותי Firebase, המספקים פונקציונליות חזקה, מדרגיות וקלות אינטגרציה. אימות Firebase, מאזין Firestore בזמן אמת, Firebase Storage ושאליות ב-Firebase מציעים פתרונות אמינים ויעילים לאימות משתמשים, אחזור הודעות בזמן אמת, קבצים מצורפים להודעות, חיפוש משתמשים וניהול מפגשי צ'אט. פתרונות אלה נבחרים מהסיבות הבאות:

Firestore Authentication מספקת מערכת אימות מאובטחת וידידותית למשתמש עם תמיכה מובנית בשיטות אימות שונות.

מאזין Firestore בזמן אמת מבטיח עדכונים בזמן אמת להודעות צ'אט ללא בקשות רשת מיותרות, מייעל ביצועים וחווית משתמש.

Firestore Storage מציע אחסון ויעיל עבור קבצים מצורפים להודעות, מה שמבטיח מסירה מהירה ואמינה. שאליות Firestore מאפשרת חיפוש יעיל של משתמשים בהתבסס על קריטריונים ספציפיים, ומספקת תוצאות חיפוש רלוונטיות.

מבנה המסמכים של Firestore ופעולות הכתיבה האטומיות מפשטות את ניהול הפעולות הצ'אט, ומבטיחות עדכונים עקביים וללא קונפליקטים בין המשתמשים.

**הפתרונות שנבחרו מספקים פיתוח יעיל יותר, ביצועים טובים יותר וחווית משתמש משופרת בהשוואה לפתרונות החלופיים שהוזכרו.**

האימפלמנטציה של העדכונים בזמן אמת בקוד עם הסבר מפורט: [לחץ כאן](#)

## סביבת העבודה - Visual studio code

Visual studio code היא מסביבות עבודה הפופולריות ביותר לכתיבת קוד בקרב מפתחים. אחד היתרונות המרכזיים שלה הוא קלות השימוש. הוא מחזיק בממשק פשוט ואינטואיטיבי, המקל על מפתחים להתחיל ולעבוד על פרויקטים במהירות.

יתרון מרכזי נוסף הוא אפשרויות ההתאמה האישית הנרחבות שלו. מפתחים יכולים להתאים אישית את מראה העורך, חיבורי המקשים והגדרות אחרות כך שיתאימו להעדפות ולזרימת העבודה שלהם. בין היתר, הוא תומך במגוון רחב של שפות תכנות, מה שהופך אותו לכלי רב תכליתי לעבודה על מגוון פרויקטים. הפלטפורמה מציעה גם תכונות שיתוף פעולה מצוינות, מה שמקל על מפתחים לעבוד על פרויקטים עם אחרים. העורך תומך בשיתוף חי, המאפשר למספר מפתחים לעבוד על אותו בסיס קוד בזמן אמת, וכולל שילוב Git מובנה עבור בקרת גרסאות ופיתוח שיתופי.

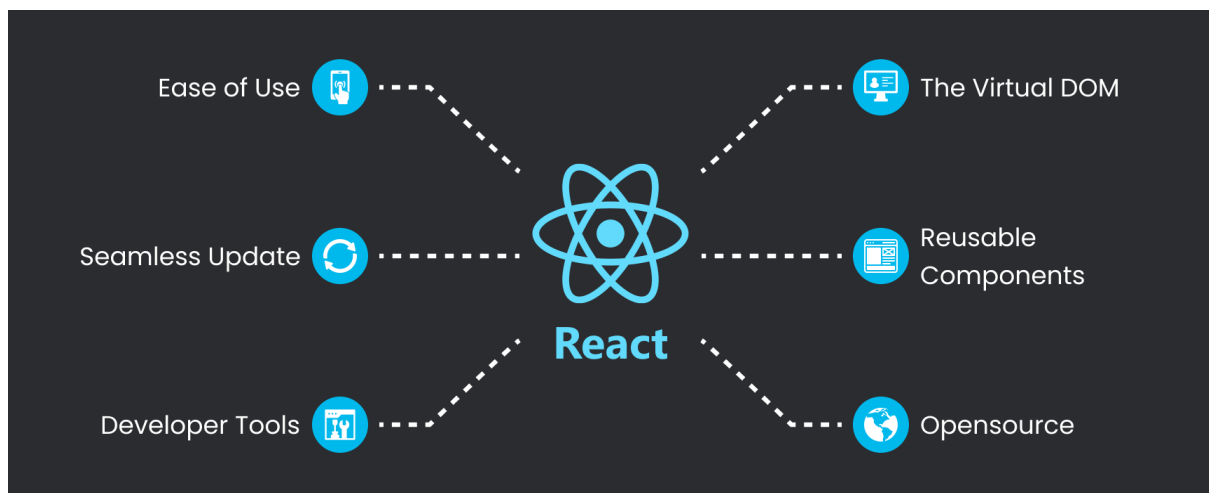


## <sup>6</sup>ReactJS

React.js היא ספריית JavaScript בקוד פתוח לבניית ממשקי משתמש. היא צברה פופולריות נרחבת בקרב מפתחים בשל הפשטות, הגמישות והביצועים היעילים שלה. בהמשך הפכה לאחת המסגרות הפופולריות ביותר לפיתוח חזיתי בשל יכולתה לנהל ממשקי משתמש מורכבים בקלות.

יכולת זאת, נובעת מהאפשרות שלה לבנות ממשקי משתמש כסדרה של רכיבים קטנים. React.js מספקת גם DOM וירטואלי (Document Object Model) המאפשר עדכונים יעילים לממשק המשתמש מבלי לדרוש טעינה מלאה מחדש של העמוד.<sup>7</sup>

אחד היתרונות המשמעותיים ביותר בשימוש בשפה הוא בזרימת נתונים חד-כיוונית. המשמעות היא שהנתונים באפליקציה זורמים בכיוון אחד, מהרכיב האב לרכיבי הילד. זה מקל על ההבנה והניהול של זרימת הנתונים בכל האפליקציה, ומפחית את הסבירות שהאפליקציה תסתבך וקשה לתחזוקה.



<sup>6</sup> LEARN REACT. React. (n.d.). Retrieved 2023, from <https://react.dev/learn> (Quick Start – React, n.d.)

<sup>7</sup> <https://media.geeksforgeeks.org/wp-content/uploads/20210908120846/DOM.png> DOM. (n.d.). Geeksforgeeks. (., 2023)

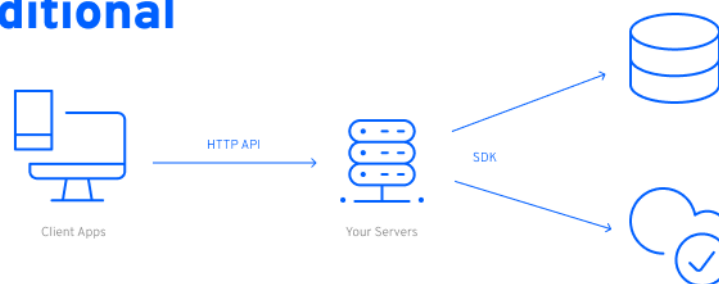
## Firestore

היא פלטפורמה ליצירת אפליקציות למובייל ולאינטרנט, המספקת מגוון כלים ושירותים לבניית באיכות גבוהה. Firestore מציעה תכונות שונות כגון אימות, אחסון בענן, מסדי נתונים בזמן אמת, אירוח והעברת הודעות.<sup>8</sup>

Firestore מספקת מסד נתונים בזמן אמת בענן NoSQL המאפשר למפתחים לאחסן ולסנכרן נתונים בין מספר לקוחות בזמן אמת. מסד הנתונים נועד לעבוד במצב לא מקוון, מה שאומר שניתן לגשת לנתונים גם כשהמכשיר אינו מחובר לאינטרנט.

בנוסף, מספקת שירותי אימות המאפשרים למשתמשים להיכנס עם הדוא"ל והסיסמה שלהם או חשבונות מדיה חברתית, כגון גוגל, פייסבוק, טוויטר ו-GitHub. שירות האימות משתלב עם שירותי Firestore אחרים, כגון מסד נתונים בזמן אמת, אחסון בענן ואירוח, ומאפשר למפתחים ליצור חוויות משתמש מותאמות אישית.

### Traditional



### Firestore



<sup>8</sup>Educative. (n.d.). What is Firestore? Educative: Interactive Courses for Software Developers. <https://www.educative.io/answers/what-is-firestore> (What Is Firestore?, n.d.)



## Hostinger

Hostinger היא ספקית אירוח אתרים שמציעה מגוון שירותי אירוח לאנשים פרטיים ועסקים. הוא מספק פתרונות אירוח ורישום דומיין.

- **אירוח משותף:** תוכניות האירוח המשותפות מיועדות לאנשים פרטיים ועסקים קטנים שמתחילים את הנוכחות המקוונת שלהם. הם מציעים תוכניות שונות עם הקצאות משאבים שונות, כולל שטח דיסק, רוחב פס, חשבונות דואר אלקטרוני ומסדי נתונים. אירוח משותף פירושו שאתרים מרובים מתארחים באותו שרת וחולקים את משאביו.
- **אירוח בענן:** המשתמשים ברשת של שרתים מחוברים כדי לפזר את העומס ולספק אמינות ומדרגיות גבוהות יותר. תוכניות אירוח בענן מספקות ביצועים וזמן פעולה טובים יותר בהשוואה לאירוח משותף, מכיוון שהמשאבים מוקצים באופן דינמי לפי דרישה.
- **רישום דומיין:** מאפשר למשתמשים לרשום שמות מתחם, מה שמאפשר להם לבסס את זהותם המקוונת. הם מציעים מגוון רחב של הרחבות דומיינים במחירים תחרותיים. משתמשים יכולים לנהל את הדומיינים שלהם דרך לוח הבקרה של Hostinger ולהגדיר הגדרות DNS.
- **תכונות וכלים:** מספקת מגוון תכונות וכלים לשיפור הניהול והביצועים של האתר. אלה כוללים לוח בקרה קל לשימוש, בוני אתרים, התקנה בלחיצה אחת של פלטפורמות CMS (מערכת ניהול תוכן) פופולריות כמו WordPress, אישורי SSL, שירותי דואר אלקטרוני, גישת FTP ומנגנוני שמירה מתקדמים במטמון.

**Dashboard** | 🏠 - Hosting - ykfileshare.tech

**ykfileshare.tech** 🌐

<b>Premium Web Hosting</b> Active <a href="#">See details</a>	<b>Domain</b> Active <a href="#">Manage</a>	<b>Free Email</b> Active <a href="#">Manage</a>	<b>Daily backups</b> Disabled <a href="#">Manage</a>
---	---	---	--

**Performance score**  
[Run speed test](#)

**File manager**

**Databases**

**Auto installer**

**Website is safe**  
 No malware found  
[See details](#)

**Your website is running smoothly**  
 No issues were found

## תקשורת שרת לקוח מבוססת Sockets ו-Socket.IO

תקשורת לקוח-שרת מבוססת שקעים (Socket-ים) היא מושג בסיסי ברשתות מחשבים. זה מאפשר חילופי נתונים והודעות בין לקוח לשרת דרך רשת. Socket.IO, היא ספריית JavaScript פופולרית המספקת תקשורת בזמן אמת, דו-כיוונית ומבוססת על "אירועים" בין לקוחות אינטרנט ושרתים. בהסבר מפורט זה, אתעמק ברקע התיאורטי של תקשורת לקוח-שרת וארחיב על התכונות והפונקציונליות הספציפיות של Socket.IO.

- Socket.IO היא ספריית JavaScript המאפשרת תקשורת דו-כיוונית בזמן אמת בין לקוחות אינטרנט (בדרך כלל דפדפנים) ושרתים. הוא מתבסס על פרוטוקול WebSocket הסטנדרטי, המספק ערוץ תקשורת מלא על בסיס חיבור על TCP.

### Socket.IO מציע מספר תכונות מפתח לתקשורת בזמן אמת:

1. תקשורת מונעת אירועים: מאפשר תקשורת מונעת אירועים, שבה לקוחות ושרתים יכולים לפלוט אירועים ולהאזין להם. זה מאפשר חילופי נתונים והודעות בצורה מאוד גמישה וא-סינכרונית.
2. תקשורת דו-כיוונית בזמן אמת: מאפשר תקשורת דו-כיוונית בזמן אמת, ומאפשרת ללקוח וגם לשרת לשלוח ולקבל נתונים בכל עת מבלי להסתמך על מנגנוני תגובה מסורתיים לבקשות. זה שימושי במיוחד עבור יישומים הדורשים עדכונים מיידיים, כגון יישומי צ'אט, כלים שיתופיים וניתוח בזמן אמת.
3. בחירת פרוטוקול אוטומטית: בוחר אוטומטית את פרוטוקול התקשורת המתאים ביותר בהתבסס על היכולות של הלקוח והשרת. זה מתחיל עם WebSocket וחוזר בחן לטכניקות אחרות, כגון סקר ארוך (long polling), אם WebSocket אינו זמין.

מימוש לשימוש ב-Socket.IO:

כדי להשתמש ב-Socket.IO ביישומי אינטרנט, השלבים הבאים כרוכים בדרך כלל:

- שילוב צד שרת: צד השרת של יישום זה עובד לרוב עם ספריות או מסגרות התומכות בשפת התכנות הרצויה. Socket.IO מספקת ספריות שרתים לשפות שונות, כגון Python, Node.js ו-Java. השרת מאתחל ומגדיר מופע Socket.IO לטיפול בחיבורי לקוח נכנסים ואירועים.

```
// Handle 'message' event
socket.on('message', (message) => {
  console.log(message);
})
```

- שילוב צד לקוח: בצד הלקוח, Socket.IO מספקת ספריית JavaScript שניתן לכלול בדפי אינטרנט. הלקוח יוצר חיבור עם השרת על ידי יצירת מופע. לאחר מכן הוא יכול לשדר אירועים לשרת או להאזין לאירועים מהשרת, מה שמאפשר תקשורת דו-כיוונית בזמן אמת.

```
// Create a socket instance
const socket = io('http://localhost:5000', {
  transports: ['websocket'],
});
console.log('Socket connected:', socket.connected);
// Function to send a message via the socket
```

## YKFileShare

```
const sendMessage = () => {  
  const report = currentUser.displayName;  
  const reported = data.user?.displayName;  
  const message = `${report} is reporting: ${reported}`;  
  socket.emit('message', message);  
};
```

# רשתות

## WWW-

המונח "www" מייצג World Wide Web, שהיא מערכת של מסמכי היפר-טקסט מקושרים הדדיים אליהם ניתן לגשת דרך האינטרנט. ה-World Wide Web הוא אוסף של דפי אינטרנט, תמונות, סרטונים ותוכן מולטימדיה אחר שניתן לגשת למשתמשים ברחבי העולם באמצעות דפדפן אינטרנט.<sup>9</sup>

ה-World Wide Web פועל על מודל שרת-לקוח, שבו דפי אינטרנט מאוחסנים בשרתי אינטרנט וניגשים אליהם לקוחות, כגון דפדפני אינטרנט, באמצעות פרוטוקול HTTP או HTTPS. דפדפני אינטרנט מפרשים HTML וטכנולוגיות אינטרנט אחרות לעיבוד דפי אינטרנט, שיכולים להכיל טקסט, תמונות, אודיו, וידאו ורכיבי מולטימדיה אחרים.

כיום, הרשת העולמית היא חלק חיוני מחיי היומיום עבור אנשים רבים ברחבי העולם. זה שינה את הדרך שבה אנו ניגשים ומשתפים מידע, מנהלים עסקים ומתקשרים זה עם זה. זה גם אפשר את הפיתוח של יישומים ושירותים מבוססי אינטרנט רבים, כולל פלטפורמות מדיה חברתית, קניות מקוונות ומחשוב ענן.



<sup>9</sup> *Untitled*. (2019, February 5). המרכז לחינוך סייבר. Retrieved February 19, 2023, from <https://data.cyber.org.il/networks/networks.pdf> (*Untitled*, 2019)

## HTTP/HTTPS:

HTTP (פרוטוקול היפרטקסט) הוא פרוטוקול המשמש להעברת נתונים דרך האינטרנט. זהו הבסיס לתקשורת נתונים עבור ה-World Wide Web ומשמש לשליחה וקבלה של נתונים בין דפדפני אינטרנט ושרתי אינטרנט. HTTP הוא פרוטוקול תגובה לבקשה.

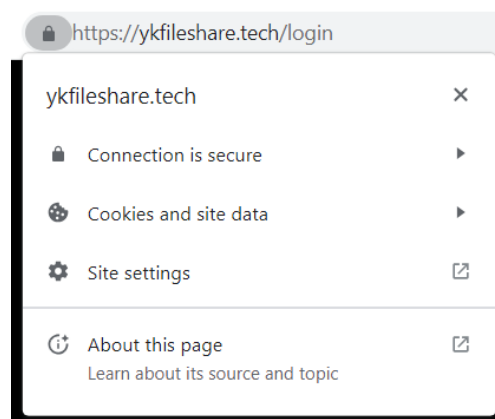
הפרוטוקול מתבסס על שימוש במודל שרת-לקוח, שבו הלקוח שולח בקשה לשרת והשרת מגיב עם הנתונים המבוקשים. בקשות HTTP נשלחות באמצעות כתובת URL, המציינת את מיקום המשאב המבוקש בשרת האינטרנט. הבקשה מכילה שיטה (כגון GET או POST), כותרות וגוף הודעה (אופציונלי). כאשר השרת מקבל בקשה, הוא מעבד את הבקשה ושולח תגובה חזרה ללקוח. התגובה מכילה קוד סטטוס, כותרות וגוף הודעה. קוד המצב מציין אם הבקשה הצליחה, וגוף ההודעה מכיל את הנתונים המבוקשים.<sup>10</sup>

×	Headers	Preview	Response	Initiator	Timing	Cookies
▼ General						
Request URL:	https://ykfileshare.tech/					
Request Method:	GET					
Status Code:	● 304					
Remote Address:	145.14.156.39:443					
Referrer Policy:	strict-origin-when-cross-origin					
▼ Response Headers						
Alt-Svc:	h3=":443"; ma=2592000, h3-29=":443"; mquic=":443"; ma=2592000; v="43,46"					
Content-Security-Policy:	upgrade-insecure-requests					
Date:	Mon, 05 Jun 2023 11:27:08 GMT					
Etag:	"28d-647db707-520e2c6a3b3b14eb;br"					
Platform:	hostinger					
Server:	LiteSpeed					

### דוגמה מהאתר שלי-

מייצג בקשת GET שנעשתה על ידי לקוח לאחר משאב מהכתובת `https://ykfileshare.tech`. השרת מגיב עם קוד סטטוס של 304, המציין שהמשאב המבוקש לא השתנה מאז הבקשה האחרונה של הלקוח, והלקוח יכול להשתמש בגרסת המטמון שלו.

HTTPS הוא הגרסה המאובטחת של HTTP המשתמשת במנגנוני הצפנה ואימות המסופקים על ידי פרוטוקולי SSL/TLS. זה מבטיח שהנתונים המועברים בין הלקוח לשרת מוגנים מפני האזנה ומוצפנים.<sup>11</sup> משתמשים בפרוטוקולים אלו לדוגמה לשליחת בקשת HTTP POST לשרת כדי לשמור קלט משתמש מטופס אינטרנט.



<sup>10</sup> (HTTPS) <https://he.wikipedia.org/wiki/HTTPS> – ויקיפדיה, n.d).

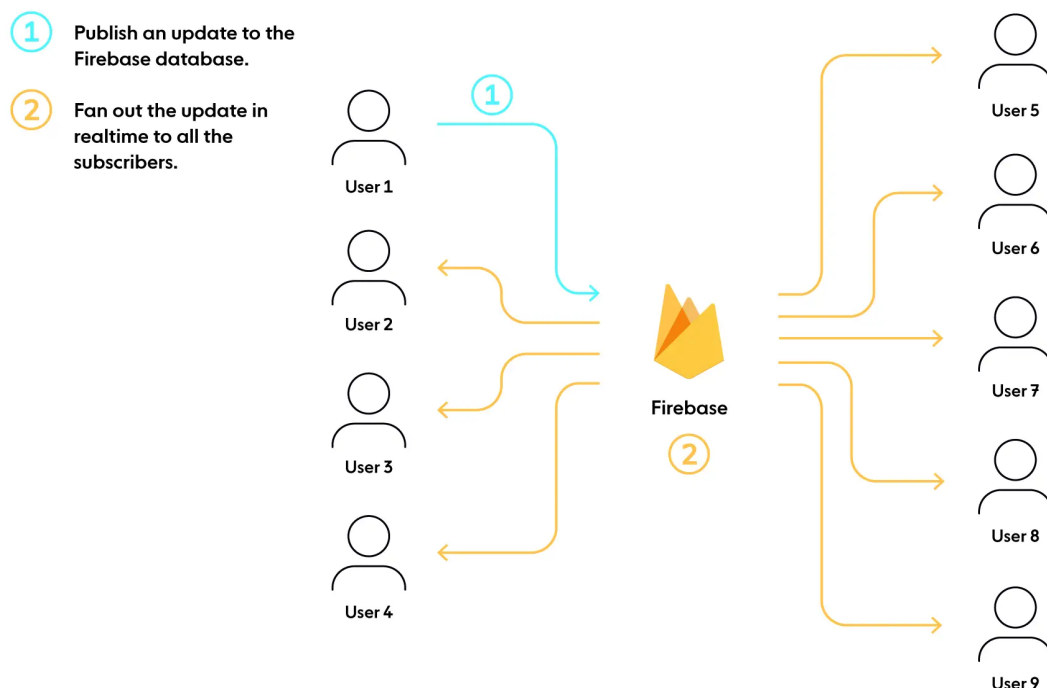
<sup>11</sup> <https://blog.logrocket.com/websockets-two-way-communication-react-app/> (Using WebSockets for Two-Way Communication in React Apps, 2022)

## WebSocket:

תיאור: WebSocket הוא פרוטוקול תקשורת המאפשר תקשורת מלאה בזמן אמת בין לקוח לשרת באמצעות חיבור TCP יחיד. הוא מאפשר העברת נתונים דו-כיוונית.

מבנה: WebSocket משתמש במנגנון [לחיצת יד תלת כיוונית](#) על גבי TCP כדי ליצור חיבור ולאחר מכן מאפשר חילופי הודעות בין הלקוח לשרת. הודעות יכולות להישלח לכל כיוון ויכולות להיות מסוגים שונים, כגון טקסט או בינארי.

בפרויקט שלי אני זקוק לעדכונים בזמן אמת ואינטראקטיביות לכן יש שימוש ב-WebSocket. לדוגמה, כדי שמשתמשים יוכלו לשלוח ולקבל הודעות בזמן אמת ללא צורך לרענן את הדף. השימוש הזה נעשה בין היתר ב-Realtime Database של Firebase (שבה השתמשתי)<sup>12</sup> בצורה הבאה:



דוגמה לתקשורת בין הלקוח לשרת ולחיבור ה-WebSocket שנוצר בעמוד הבא.

<sup>12</sup>

<https://ably.com/topic/firebase-vs-websocket#how-firebase-and-web-socket-work-together-realtime-database> (Firebase Vs WebSocket: Differences and How They Work Together, 2022)

▼ General	
Request URL:	ws://localhost:3001/socket.io/?EIO=4&transport=websocket
Request Method:	GET
Status Code:	● 101 Switching Protocols
▼ Response Headers <input type="checkbox"/> Raw	
Connection:	Upgrade
Sec-WebSocket-Accept:	S/DjQ/oTLhPjRWfRNSpK0CUHh3E=
Upgrade:	websocket
▼ Request Headers <input type="checkbox"/> Raw	
Accept-Encoding:	gzip, deflate, br
Accept-Language:	en,he;q=0.9,en-US;q=0.8
Cache-Control:	no-cache
Connection:	Upgrade
Host:	localhost:3001
Origin:	https://ykfileshare.tech
Pragma:	no-cache
Sec-WebSocket-Extensions:	permessage-deflate; client_max_window_bits
Sec-WebSocket-Key:	mhWnw7F5sQLvfaHljzqPjA==
Sec-WebSocket-Version:	13
Upgrade:	websocket

Data	Length	Time
0["sid":"1HK9IPXB-Si2Dr2WAAUh","upgrades":[],"pingInterval":25000,"pingTimeout":20000,"maxPayload":1...	107	14:27:08.559
40	2	14:27:08.560
40["sid":"CSSuep6uhOIC3HJQAAUi"]	32	14:27:08.568
2	1	14:27:33.244
3	1	14:27:33.244
2	1	14:27:58.266
3	1	14:27:58.266
42["message","cooluser is reporting: undefined"]	48	14:35:59.525
42["message","cooluser is reporting: undefined"]	48	14:35:59.526
2	1	14:36:18.491
3	1	14:36:18.491
2	1	14:36:43.499
3	1	14:36:43.499
2	1	14:37:08.501
3	1	14:37:08.501

```

▼ 42["message", "cooluser is reporting: undefined"]
  0: "message"
  1: "cooluser is reporting: undefined"

```

### ▼ Request initiator chain

▼ https://ykfileshare.tech/

▼ https://ykfileshare.tech/static/js/main.b67d504f.js

**ws://localhost:3001/socket.io/?EIO=4&transport=websocket**

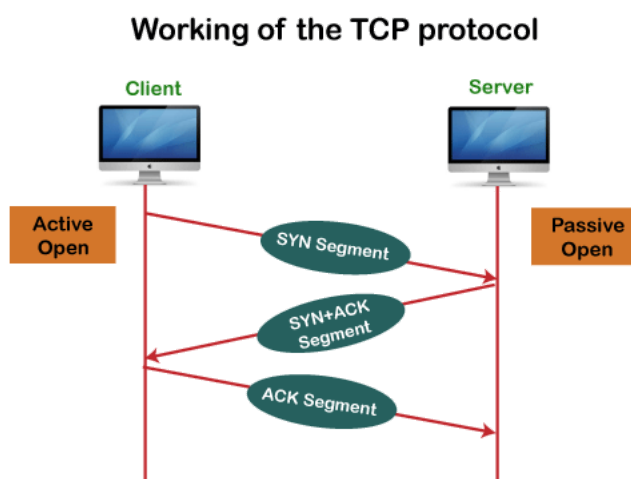
## TCP/IP:

### TCP/IP (פרוטוקול בקרת שידור/פרוטוקול אינטרנט):

היא חבילה של פרוטוקולי תקשורת המאפשרת העברת נתונים בין רשתות. הוא מספק תקשורת אמינה, מכוונת חיבור בין מכשירים ומהווה את הבסיס לתקשורת באינטרנט. הוא פועל בשכבת הרשת (IP) ושכבת התחבורה (TCP) ומפרק נתונים למנות, מקצה כתובות IP למכשירים, יוצר חיבורים ומבטיח אספקת נתונים אמינה.

בפריקט של, TCP/IP הוא הפרוטוקול הבסיסי המאפשר לאפליקציית שיתוף הקבצים לתקשר עם שרתים והתקנים אחרים דרך האינטרנט. לדוגמה, כאשר היישום שלך שולח בקשת HTTP או WebSocket הפרוטוקול מטפל בשידור ובמשלוח של מנות הנתונים.

TCP בפרט הוא פרוטוקול המספק ווידוא הצפנה בשכבת התעבורה עם חיבור המספק מסירה אמינה, מסודרת ונבדקת שגיאות של נתונים בין יישומים הפועלים על מארחים שונים. כאשר שני יישומים רוצים לתקשר אחד עם השני באמצעות TCP, הם יוצרים חיבור על ידי ביצוע **לחיצת יד תלת כיוונית** (TCP 3-Way Handshake).<sup>13</sup>



<sup>13</sup> <https://www.javatpoint.com/tcp> (What Is Transmission Control Protocol (TCP)? Header, Definition - Javatpoint, n.d.)



**DNS:**

DNS היא מערכת שמות היררכית ומפוזרת המשמשת לתרגום שמות דומיין לכתובות IP (פרוטוקול אינטרנט). שמות מתחם הם שמות הניתנים לקריאה על ידי אדם המשמשים לזיהוי אתרים ומשאבים אחרים באינטרנט, בעוד שכתובות IP הן מזהים מספריים המשמשים לאיתור משאבים אלה ולתקשורת איתם.<sup>14</sup>

כאשר משתמש מקליד שם דומיין בדפדפן האינטרנט שלו, הדפדפן שולח בקשה לפותר DNS "לתקן" את שם הדומיין לכתובת IP. פותר ה-DNS שולח שאילתה למערכת ה-DNS כדי למצוא את כתובת ה-IP המשויכת לשם הדומיין. מערכת ה-DNS מגיבה עם כתובת ה-IP, והמידע מידע זה לדפדפן האינטרנט של המשתמש. ניתן לחשוב על שרתי ה-DNS כספר טלפונים דיגיטלי - אנו יודעים למי אנחנו רוצים להתקשר למרות שאנחנו לא זוכרים את המספר שלו. במקרה כזה נוכל לחפש בספר הטלפונים הדיגיטלי את השם היעד כמו "משה" או "גוגל" ונקבל את הכתובת שלו. בספר טלפונים דיגיטלי כתובת היעד תהיה מספר טלפון, בעוד בשרת ה-DNS נקבל כתובת IP.

**Website Details**

Access your website at	<a href="http://ykfileshare.tech">http://ykfileshare.tech</a>
Access your website with www	<a href="http://www.ykfileshare.tech">http://www.ykfileshare.tech</a>
Website IP address	145.14.156.39

<sup>14</sup> *Untitled*. (2019, February 5). המרכז לחינוך סייבר. Retrieved February 19, 2023, from <https://data.cyber.org.il/networks/networks.pdf>

## TLS:

פרוטוקול אבטחה המספק תקשורת מאובטחת ברשתות. נועד כדי לאבטח את ערוצי התקשורת בין יישומים, בדרך כלל דרך האינטרנט. בדרך כלל מגן על מידע רגיש מפני האזנה, שיבוש וגניבה לא מורשית על ידי הצפנת הנתונים במהלך השידור.

### תהליך לחיצת היד של TLS:

לחיצת היד של TLS היא התהליך הראשוני שבו הלקוח והשרת מאמתים זה את זה, מנהלים משא ומתן על אלגוריתמי הצפנה ומקיים מפתחות הפעלה. זה כולל את השלבים הבאים:

- ClientHello: הלקוח יוזם את לחיצת היד על ידי שליחת רשימה של חבילות צופן נתמכות ופרמטרים אחרים לשרת.
- ServerHello: השרת מגיב עם חבילת הצפנים שבחרת, אישור דיגיטלי לאימות ופרמטרים נוספים.
- החלפת אישורים: השרת שולח את האישור הדיגיטלי שלו המכיל את המפתח הציבורי שלו ללקוח.
- ClientKeyExchange: הלקוח יוצר מפתח הפעלה, מצפין אותו במפתח הציבורי של השרת מהאישור, ושולח אותו לשרת.
- ChangeCipherSpec: גם הלקוח וגם השרת מודיעים זה לזה שהודעות הבאות יוצפנו באמצעות מפתח ההפעלה המוסכם.
- הסתיים: גם הלקוח וגם השרת שולחים הודעה המאמתת שתהליך לחיצת היד הושלם.

### הצפנה והעברת נתונים:

לאחר השלמת לחיצת היד של TLS, העברת הנתונים בפועל מתרחשת באמצעות אלגוריתמי ההצפנה המוסכמים. מקובל להשתמש בהצפנה סימטרית, כאשר אותו מפתח מצפין ומפענח את הנתונים. הצפנה אסימטרית משמשת במהלך לחיצת היד להחלפת מפתחות הפעלה בצורה מאובטחת.

רשויות אישורים (CA):

TLS מסתמך על ישויות מהימנות של צד שלישי. האישור חתום דיגיטלי על אישורי שרת כדי לאמת את האותנטיות שלהם. לקוחות יכולים לבדוק את תקפות האישורים על ידי אימות החתימה, להבטיח שהשרת איתו הם מתקשרים לגיטימי.



### התעודה של האתר שלי:

**FTP:**

File Transfer Protocol, הוא פרוטוקול רשת המשמש להעברת קבצים בין לקוח לשרת ברשת מחשבים. הוא מספק דרך אמינה ויעילה להחליף קבצים ברשתות מבוססות TCP/IP, כגון האינטרנט. פועל במודל שרת-לקוח, שבו הלקוח יוזם את החיבור ומבקש העברת קבצים, בעוד השרת מאזין לחיבורים נכנסים ומגיב לבקשות הלקוח.

**הקמת חיבור:**

FTP משתמש בעיקר בשני חיבורים נפרדים: חיבור עבור בקרה וחיבור עבור נתונים. חיבור הבקרה נוצר בפורט 21 TCP והוא אחראי על שליחת פקודות וקבלת תגובות בין הלקוח לשרת. חיבור הנתונים, לעומת זאת, נוצר באופן דינמי ומשמש להעברת נתוני הקובץ בפועל.

**מצבי העברת נתונים:**

FTP תומך בשני מצבים להעברת נתונים: קבצים: מצב ASCII ומצב בינארי. מצב בינארי, הידוע גם כמצב תמונה, מעביר קבצים כזרם בינארי, שומר על תוכן הקובץ המדויק ללא כל המרות תווים.

השימוש שלי ב-FTP:

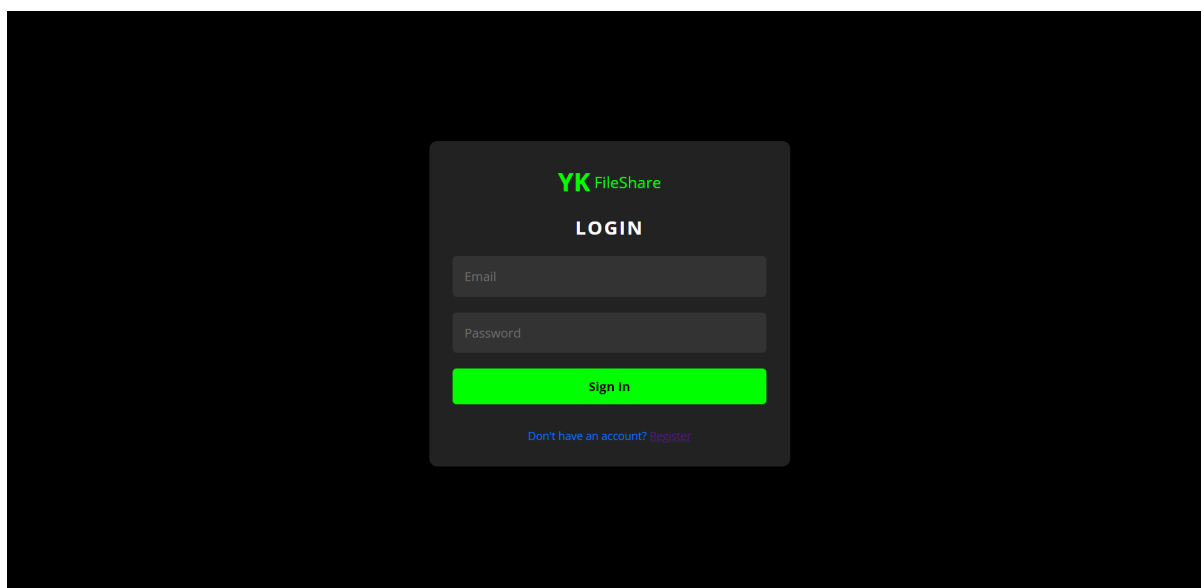
FTP Details	
FTP IP	ftp://145.14.156.39
FTP Hostname	ftp://ykfileshare.tech
FTP Username	u429970138.ykfileshare.tech
File Upload Path	public_html

הקובץ public\_html הינו הקובץ של הbuild של הפרויקט שלי. כלומר, כשהכנתי את הפרויקט שלי להעלאה לרשת, הייתי צריך לעשות לו פעולה שנקראת build. את התוצר שלה העלתי לתוך הקובץ הנתון והוא הקובץ הסופי שמועלה לאתר ומוצג למשתמשים.

## תיאור מסכי המערכת:

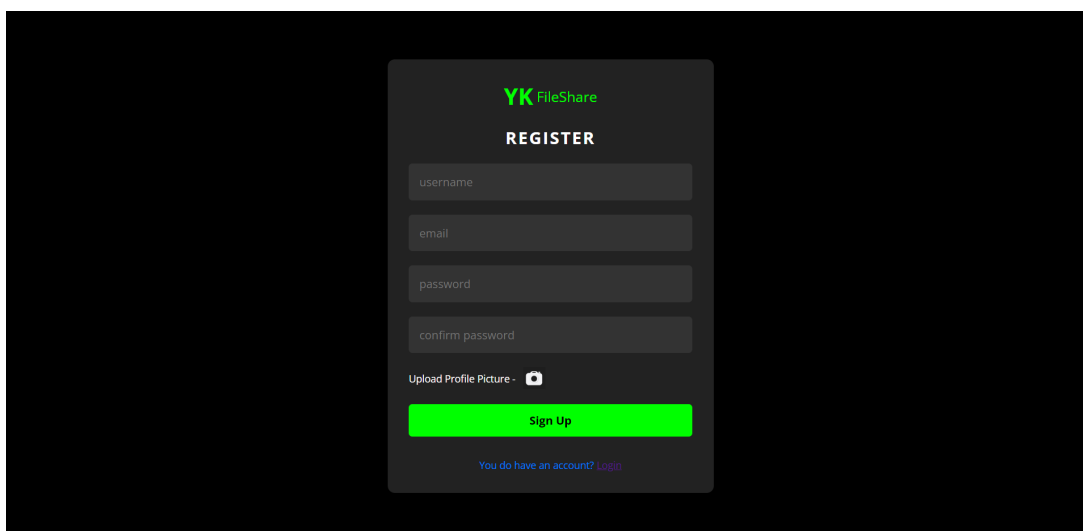
### 1. עמוד הכניסה:

זה העמוד אליו מגיעים כאשר מריצים את הפרויקט/מתחברים לכתובת האתר. עמוד הכניסה הוא עמוד הLogin לאתר, כמו רשתות חברתיות אחרות, גם אני בחרתי קודם להציג את עמוד הכניסה עם אפשרות לעבור לעמוד ההרשמה במקרה הצורך. בעמוד הכניסה עליך להזין את כתובת המייל והסיסמה של המשתמש שלך, ואם אין קיים משתמש עם נתונים אלו באתר - הכניסה תתאפשר, אחרת תקבל הודעת שגיאה. לאחר לחיצה על כפתור הSign In הנתונים מהלוקח מועברים לשרת אשר מאמת את פרטי המשתמש עם מסד הנתונים הקיים ב-Firebase.



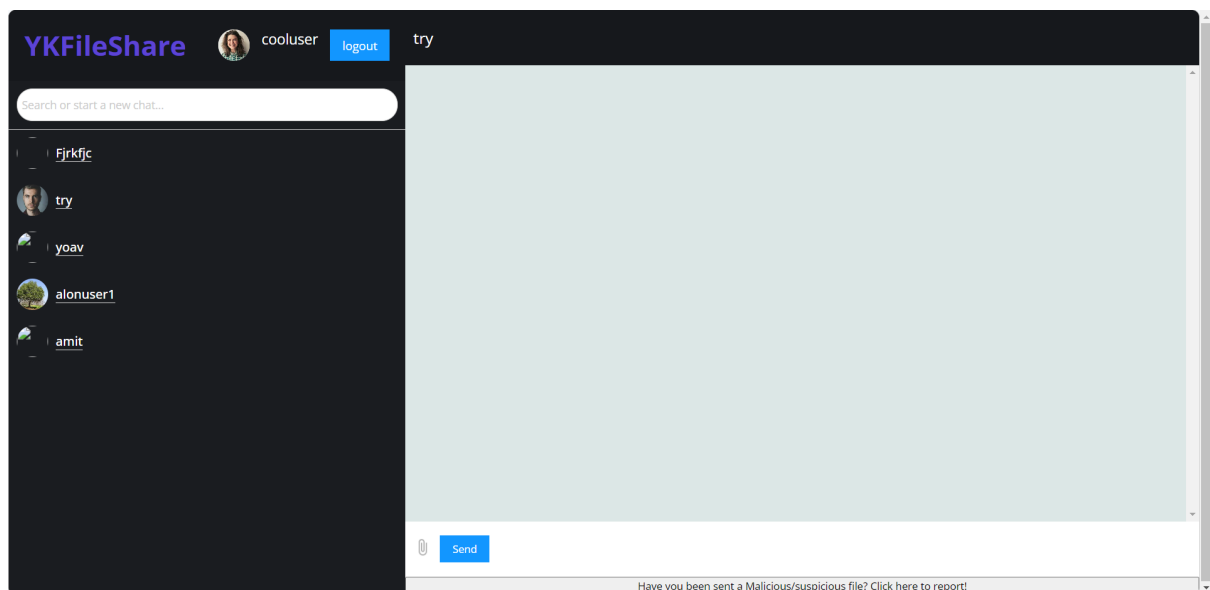
### 2. עמוד ההרשמה:

אל עמוד ההרשמה ניתן לגשת דרך גלישה לכתובת העמוד או דרך לחיצה על הקישור המופיע בעמוד ההרשמה. לאחר ההגעה לעמוד, המשתמש יצטרך למלא פרטים אודותיו העומדים בקריטריונים והדרישות של האתר (כמו לדוגמה סיסמה מעל 6 תווים) ולאחר מכן אם יעמוד בכל הדרישות, ישמר המשתמש החדש במסד הנתונים של האתר ב-Firebase באוסף המשתמשים ויקבל גישה לאתר.



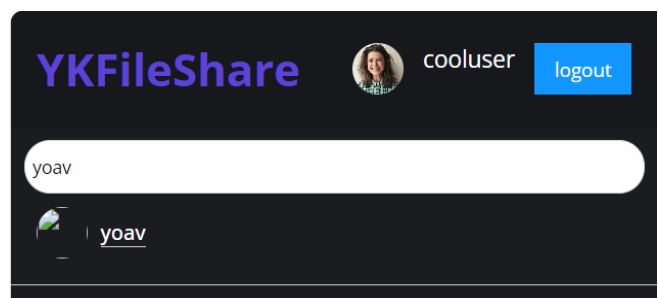
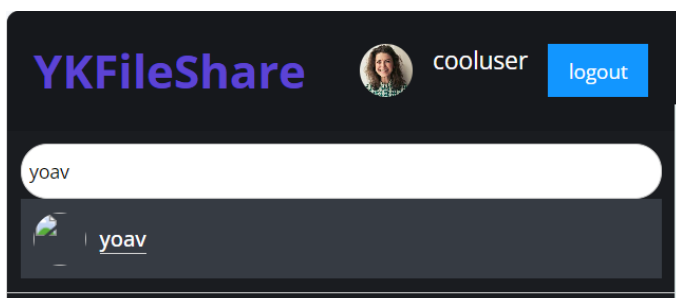
### 3. עמוד הבית/הצאט:

לאחר שנרשמים/מתחברים למשתמש, מגיעים לעמוד המרכזי באתר. כפי שניתן לראות, הוא תומן בתוכו מרכיבים רבים.



- בחלק העליון של העמוד, מוצג עבור המשתמש - שמו, תמונתו, כפתור התנתקות ושם האתר.
- בצד שמאל מופיע סרגל הניווט בין האנשים השונים שהמשתמש בשיחה איתם ואפשרות לחפש משתמש לשיחה.

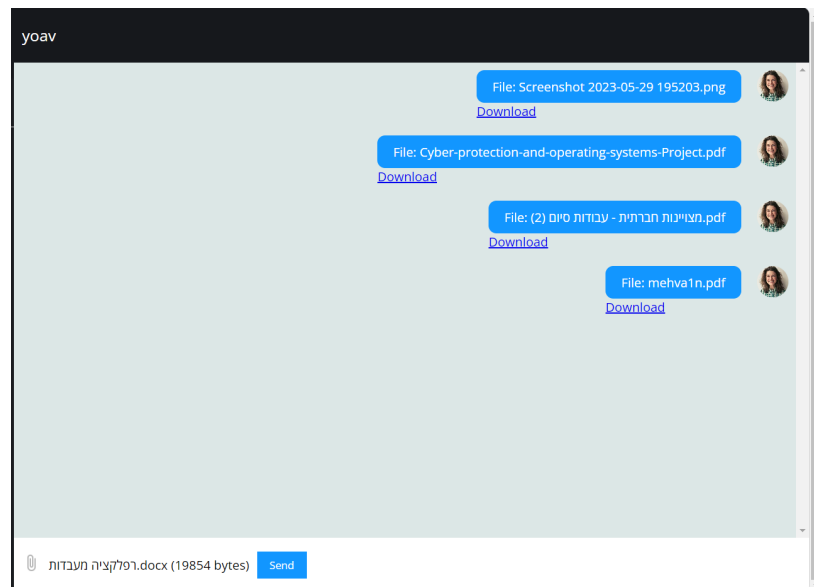
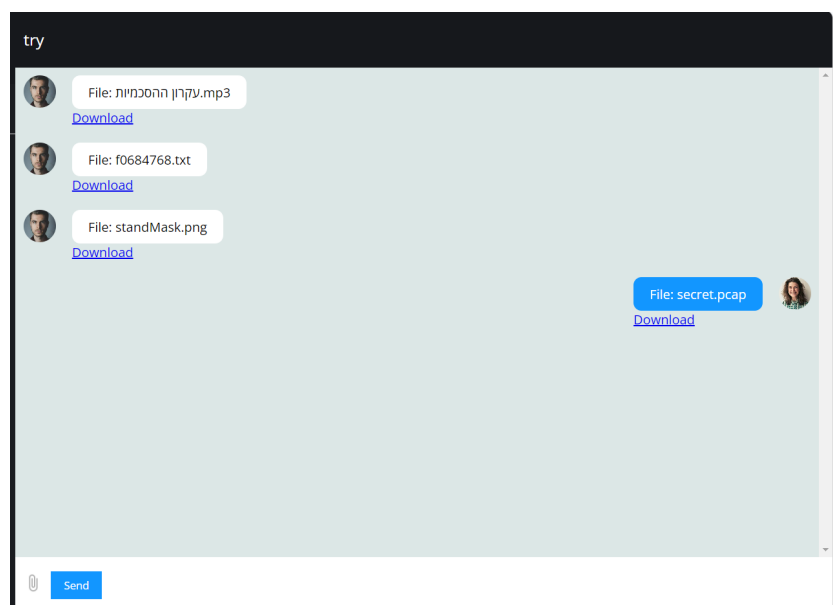
דוגמה: חיפשנו את שם חברינו - "yoav" שאיתו אנו רוצים לדבר ולחצנו על Enter. לאחר שמצביעים עליו עם העכבר לפני הלחיצה הוא מופיע באופן בהיר יותר.



לאחר שלחצנו על המשתמש (ומשתמשים אחרים נוספים) הם יופיעו בסרגל הניווט שלנו, יתווספו לאוסף הצאטים השמורים במסד הנתונים ונוכל להיכנס לצאט איתם ולשתף איתם קבצים מתי שנרצה. הנה המסך שאנו נמצאים בו לאחר פעולה זאת (כשהצאט האחרון שפתחנו הוא עם אלון, אבל בחרנו להיכנס לצאט עם יואב).


**הצאת עצמו:**

בחלק של הצאת באתר מופיע שם המשתמש של מי שאנו בשיחה איתו כעת. ניתן לצרף את הקובץ המבוקש בלחיצה על אייקון הקובץ. לאחר שהוספנו את מה שאנו מעוניינים לשלוח למשתמש, נלחץ על SEND. ההודעה תעבור ותשמר תחת אוסף ה"chats" במסד הנתונים ודרך כך תתעדכן תופיע למשתמשים. כאן ניתן לראות ששלחנו ליואב מספר קבצים ואנחנו מוכנים לשלוח לו אחד נוסף, שימו לב שהצאת עם יואב יקפוץ למעלה מצד שמאל (רשימת הצאטים מסודרת לפי הזמן האחרון ששותף קובץ בצאט):

**דוגמה למעבר של סוגי קבצים שונים בין משתמשים:**

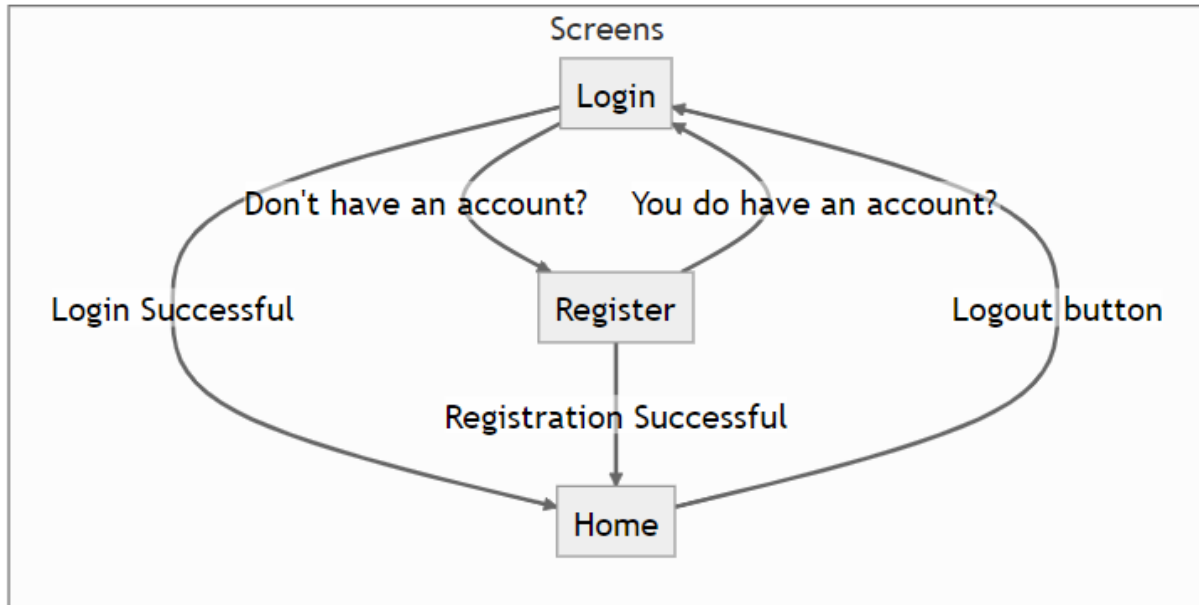
### כפתור הדיווח:

כדי לנסות לשמור כמה שיותר על המשתמשים באתר, ישנה אפשרות לכל משתמש לשלוח דיווח על משתמש אחר שהוא איתו בשיחה. הדיווח ישלח לשרת ובתגובה הלקוח (המדווח) יקבל הודעת אישור (-יוצג בConsole).

 [Send](#)

Have you been sent a Malicious/suspicious file? [Click here to report!](#)

## Screen Flow Diagram





## מסד ומבני הנתונים-

### מסד נתונים: [Firebase/Firestore](#)


#### "chats" - מאחסן את כל המידע הקיים על כל צאט בין משתמשים:

שם מסמך: חיבור הuid של שני המשתמשים בצאט.

שדות המידע בכל מסמך-

1. messages (מערך) - מערך של אובייקטי הודעה שכל אחד מכיל:

- date (חותם זמן): חותם זמן של ההודעה
- filename (מחרוזת): שם הקובץ המצורף
- fileSize (מספר): גודל הקובץ המצורף בבתים
- fileUrl (מחרוזת): כתובת להורדת הקובץ המצורף
- id (מחרוזת): מזהה ייחודי להודעה עצמה
- senderId (מחרוזת): המזהה ייחודי של שולח ההודעה (UID של השולח)

 qAFIOqzNEMMSkd8yoH7TXjXtqA8343ebMEP8ZKPjV0Y3vYkKaFsLM3s2

[+ Start collection](#)

[+ Add field](#)

▼ messages
 

▼ 0
 

date: May 26, 2023 at 6:06:40 PM UTC+3
   
 fileName: "נבחרצדשני.mp4"
   
 fileSize: 5459291
   
 fileUrl: "https://firebasestorage.googleapis.com/v0/b/trust-  
yoav.appspot.com/o/3328bf98-a57b-4d14-aed6-  
5545884d51ea?alt=media&token=b809e17c-3880-4f8d-  
99e7-b0a91388d7b3"
   
 id: "8984d4e3-0451-4862-b4b5-ba7182ae0b29"
   
 senderId: "qAFIOqzNEMMSkd8yoH7TXjXtqA83"

**"userChats": מאסן עביר כל משתמש את אוסף הצאטים שלו:****שם מסמך: המזהה הייחודי של המשתמש****שדות המידע בכל מסמך (בכל מסמך יופיעו כמספר הצאטים שהמשתמש חלק מהם):**

1. uid של הצאט (אובייקט): המזהה הייחודי של השולח + המזהה הייחודי של הנשלח.

- date (חותם זמן): הזמן של הפעם האחרונה שקובץ שותף בצאט

- last message (אובייקט):

• filename (מחרוזת): השם הקובץ המצורף האחרון שנשלח

• fileSize (מספר): גודל הקובץ המצורף האחרון שנשלח בבתים

• fileUrl (מחרוזת): כתובת להורדת הקובץ המצורף האחרון שנשלח

- userInfo (אובייקט) - של מי שאנחנו שולחים לו:

• displayName (מחרוזת): שם המשתמש

• photoURL (מחרוזת): הכתובת של תמונת המשתמש

• uid (מחרוזת): המזהה הייחודי של המשתמש

43ebMEP8ZKPjV0Y3vYkKaFsLM3s2

+ Start collection

+ Add field

▼ qAF10qzNEMMSkd8yoH7TXjXtqA8343ebMEP8ZKPjV0Y3vYkKaFsLM3s2

date: May 27, 2023 at 10:59:55 PM UTC+3

▼ lastMessage

fileName: "(3) עבודות חברתית - עבודות סיום.pdf"

fileSize: 7512786

fileUrl: "https://firebasestorage.googleapis.com/v0/b/trust-yoav.appspot.com/o/e6b5cb04-f0d1-4b6b-b57f-c1632b9818d3?alt=media&token=f9c5bfdc-035a-4e61-b434-613234d1b8b4"

text: ""

▼ userInfo

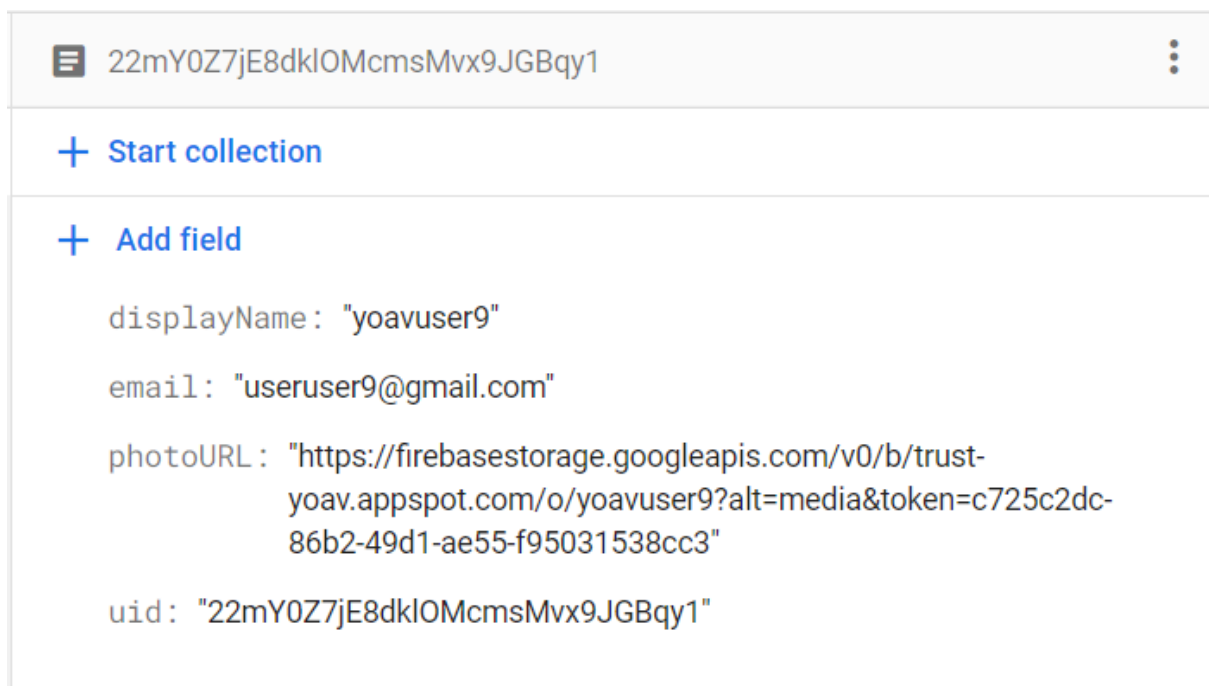
displayName: "boss11"

photoURL: "https://firebasestorage.googleapis.com/v0/b/trust-yoav.appspot.com/o/boss11?alt=media&token=b5c456d2-236c-4a4d-87b3-87e00979c170"

uid: "qAF10qzNEMMSkd8yoH7TXjXtqA83"

**"users": מאחסן את פרטי המשתמש****שם מסמך: המזהה הייחודי של המשתמש****שדות המידע בכל מסמך-**

1. displayName (מחזורת): שם המשתמש
2. email (מחזורת): מייל המשתמש
3. photoURL (מחזורת): הכתובת של תמונת המשתמש
4. uid (מחזורת): המזהה הייחודי של המשתמש



מבני הנתונים בפרויקט:

על כל מבנה נתונים מוצג - שם, היכן מופיע, מבנה ומה הוא מכיל.

1. `[chats, setChats]` - Chats
  - מבנה: state - מערך שמתעדכן עם פעולה.
  - מכיל: מילון עם נתוני צ'אט שאוחזרו מ-Firebase ופונקציה לעדכן אותם (בעדכון מצב נוכחי).
2. `chats` - Chats
  - מבנה: מילון של צאטים.
  - מכיל: מפתח - id של הצאט, ערך - מערך שכל איבר בו הוא חלק מנתוני הצאט.
3. `currentUser` - Chats, Input, Message, Navbar, Search, App, chat
  - מבנה: אובייקט המשתמש
  - מכיל: מידע על המשתמש הנוכחי
4. `dispatch/data` - Chats, Input, Chat, Message, Messages
  - מבנה: ערך אובייקט הצאט
  - מכיל: ערך שניתן להפעיל עליו פונקציה שתשפיע על `chatcontext`.
5. `doc` - Chats, Input, Messages, Register
  - מבנה: `DocumentReference`
  - מכיל: הפניה למסמך ספציפי (תלוי בשימוש)
6. `[file, setFile]` - Input
  - מבנה: state - קובץ ופונקציה לעדכון מצב.
  - מכיל: קובץ שהוזן, פונקציה לעדכון.
7. `[fileUrl, setFileUrl]` - Input
  - מבנה: state - קישור לקובץ ופונקציה לעדכון מצב.
  - מכיל: קישור לקובץ, פונקציה לעדכון.
8. `storageRef/displayNameRef` - Input, Register
  - מבנה: URL - `StorageReference`
  - מכיל: הפניה למיקום לקובץ באחסון (אם אין נוכחי, יוצר חדש) - עבור הקובץ/המשתמש החדש.
9. `uploadTask` - Input, Register
  - מבנה: URL
  - מכיל: "משימה" להעלאת הקובץ לתוך המיקום הרלוונטי.
10. `isCurrentUser` - Message
  - מבנה: משתנה בוליאני
  - מכיל: אמת או שקר - האם אתה שלחת את ההודעה? (המשתמש הנוכחי).
11. `fileName` - Message

- מבנה: string
  - מכיל: השם של ההודעה הנוכחית
12. `[messages, setMessages]` - Messages
- מבנה: state - מערך ופונקציה לעדכון
  - מכיל: נתוני ההודעות (הקבצים) שאוחזרו מ-Firebase לצאת הנוכחי ופונקציה לעדכן אותם (במקרה הצורך).
13. `messages` - Messages
- מבנה: מערך
  - מכיל: הקבצים ששותפו בצאת הנוכחי.
14. `[username, setUsername]` - Search
- מבנה: state - מחרוזת שמתעדכנת עם פעולה.
  - מכיל: מחרוזת של שם משתמש שהוזן, פעולה לעדכון לאחר הזנה.
15. `[user, setUser]` - Search
- מבנה: state - מידע על משתמש שמתעדכן עם פעולה.
  - מכיל: מידע על משתמש שנמצא בחיפוש, פעולה לעדכון לאחר מציאתו.
16. `[err, setErr]` - Search, Register, Login
- מבנה: state - משתנה בוליאני ופונקציה לעדכון מצבו.
  - מכיל: אמת/שקר למקרה שיש שגיאה בתהליך המוגדר (חיפוש/פתיחת משתמש).
17. `AuthContext` - AuthContext
- מבנה: `React.Context`
  - מכיל: קונטקסט שאותו רכיבים אחרים יוכלו "לצרוך".
18. `ChatContext` - ChatContext
- מבנה: `React.Context`
  - מכיל: קונטקסט שאותו רכיבים אחרים יוכלו "לצרוך".
19. `e` - Register, Login
- מחזיק במידע שהוכנס לטופס ההתחברות/ההרשמה.
20. `res` - Register
- מבנה: נתוני משתמש שהועלו לאחסון.
  - מכיל: משתמש שהועלה עכשיו.

## סקירת חולשות והאימים:

### שכבת האפליקציה:

- **הזרקת SQL:** בקוד, שכבת האפליקציה מקיימת אינטראקציה עם מסד נתונים Firestore באמצעות Firebase SDK שאינו רגיש להזרקות sql. עם Firebase, אתה לא בונה פקודת SQL (או כל פקודת מחרוזת המורכבת מחלקים שונים) כדי לבצע שאילתה. במקום זאת, אתה משתמש ב-API שמסופק על ידי ה-SDK ומעביר מחרוזות המנוהלות אוטומטית על ידי ה-API. זה אומר שהזרקת SQL אינה בעיה כאן (זה מסד נתונים שהוא no-sql).
- **עבודה עם אתרים:** הקוד נכתב עבור אפליקציית Web של React, העוקבת אחר שיטות עבודה מומלצות לפיתוח אתרים, כולל תקשורת מאובטחת באמצעות HTTPS. ריאקט עוזר למנוע התקפות XSS כברירת מחדל באמצעות מנגנון העיבוד שלו, כלומר ה-DOM הווירטואלי בעזרת כך שמשתמש בקבצי jsx אשר אוכפים הפרדה קפדנית בין js לhtml. הפרדה זו מקשה על התוקף להחדיר קוד זדוני לפלט המעובד. מעבר לכך, ריאקט מתייחסת לכל תוכן שמגיע מהמשתמש כטקסט ולא HTML או JS כך שקשה להזריק לה קוד.
- **תהליך הכניסה, ההרשמה והאימות:** מתבצע שימוש בשילוב בין אימות בעזרת JS ובאימות Firebase כדי לטפל בכניסה, הרשמה ואימות של משתמשים. Firebase Authentication מספק מנגנונים מאובטחים לאימות משתמשים - אימות זה אמין מונע פרצות אימות רבות ונפוצות.
- **מתקפת MITM (Man-in-the-Middle):** השימוש ב-HTTPS מסייע בהגנה מפני התקפות MITM על ידי הצפנת התקשורת בין הלקוח לשרת עם שימוש בתעודות SSL/TLS.
- **התקפות DDOS/DOS:** השתמשתי בcloudflare דרך הוסינגר (אחסון האתר) אשר משתמשים באמצעים ברמת הרשת, כגון הגבלת קצב, סינון תעבורה ואיזון עומסים כדי למנוע מתקפות אלו.
- **העלאת קבצים:** הקוד כולל תכונת העלאת קבצים שמשתמשת ב-Firebase Storage לאחסון הקבצים שהועלו. Firebase Storage מטפל באמצעי אבטחה, כולל אימות קבצים, כללי בקרת גישה ואחסון מאובטח של קבצים.

### שכבת התעבורה:

- כאשר המידע עובר מהאתר ל-Firebase, הוא מועבר באמצעות פרוטוקול TCP. פרוטוקול זה ונמצא בשימוש נרחב בעולם ומספק תקשורת אמינה ומוכונת חיבור בין יישומים דרך רשתות IP. זה מבטיח שהנתונים מועברים ללא שגיאות, בסדר הנכון, ועם מנגנוני בקרת כדי לייעל את ביצועי הרשת. כאשר האתר יוזם חיבור עם Firebase, פרוטוקול TCP מופעל כדי ליצור חיבור אמין. תהליך זה כולל לחיצת יד תלת כיוונית, שהיא סדרה של שלוש הודעות שנשלחות לפני החיבור כדי לסנכרן וליצור את החיבור.

### **להלן סקירה שלב אחר שלב של האופן שבו המידע זורם באמצעות TCP ולחיצת היד התלת-כיוונית:**

1. האתר שלך (הלקוח) שולח חבילת SYN (סנכרון) לשרת Firebase, המציין את כוונתו ליזום חיבור.
2. שרת Firebase מגיב עם חבילת SYN-ACK (Synchronize-Acknowledge), מאשר את בקשת הלקוח ומציין את מוכנותו ליצור את החיבור.
3. האתר שלך שולח חבילת ACK (אישור) לשרת Firebase, ומאשר את הקבלה של חבילת SYN-ACK. שלב אחרון זה משלים את לחיצת היד התלת כיוונית, והחיבור נוצר.

4. לאחר יצירת החיבור, האתר שלך ו-Firebase יכולים להחליף נתונים באמצעות TCP, מה שמבטיח אמינות, בקרת זרימה וזיהוי שגיאות לאורך כל התקשורת.

### הפעלת המערכת

- **חולשות כמו הזרקת קוד, הזרקת SQL וכו':** הקוד משתמש בשיטות פיתוח אינטרנט מודרניות ובספריות כגון React ו-Firebase, המספקות הגנה מובנית מפני נקודות תורפה נפוצות. React עוזר למנוע הזרקת קוד והתקפות XSS על ידי בריחה מקלט משתמש כברירת מחדל. Firebase Firestore מגן מפני הזרקת SQL ופגיעויות אחרות הקשורות למסד נתונים באמצעות כללי האבטחה ואימות השאילתות שלה.

# מימוש הפרויקט



## חלק א' - סקירת המודולים והמחלקות

### מודולים/מחלקות מיובאים:

- את השימוש של כל מודול/מחלקה יהיה ניתן לראות בפירוט הנוסף בתוך המחלקות בהמשך.

#### react:<sup>15</sup>

1. React: ספריית JavaScript לבניית ממשקי משתמש.
2. useContext: מאפשר לגשת לנתונים מאובייקט "context" דרך כל רכיב של היישום. ניתן לחשוב על הוק זה כמו צנצנת עוגיות שנשמרת על מדף גבוה. אם אתה משתמש ב-useContext, הוא מהווה כמו שרפרף שמאפשר להגיע לצנצנת מתי שנרצה. כלומר, דרך כך ניתן לגשת ישירות לנתונים המאוחסנים באובייקט ה"context", מבלי לעבור דרך מישור/משהו אחר כדי לקבל אותם - נקרא "צרכן".<sup>16</sup>
3. useEffect (מ-'react'): הוק של React המאפשר "לומר" ל-React להאזין ל"תופעות לוואי" על הרכיבים לאחר רינדור הרכיב.
4. useState (מ-'react'): מאפשר לרכיבים להוסיף state. המצב הוא מידע/נתונים שיכולים להשתנות לאורך זמן על ידי גורמים שונים.
5. useRef: מאוד דומה לuseState אבל מטרתו היא לשמור על ערך או נתונים ולא לעשות רינדור מחדש כאשר הערך מתעדכן.
6. useReducer: מאפשרת לנהל מצבים מורכבים ברכיבים שלנו. מקבל פונקציית reducer וערך מצב ראשוני כקלט, ומחזיר מערך עם הערך הנוכחי של המצב ופונקציית dispatch.

#### firebase:

1. db (מיובא מ-'firebase/..'): מייצג את אובייקט מסד הנתונים של Firebase Firestore ומשמש לאינטראקציה איתו, כגון לעשות שאילתות או עדכון מסמכים.
2. onSnapshot, doc (מיובא מ-'firebase/firestore'): פונקציות מספריית Firebase Firestore להאזנה לעדכונים בזמן אמת על מסמכי Firestore ולאחזר הפניות למסמכים.
3. arrayUnion, serverTimestamp, Timestamp, updateDoc (מיובא מ-'firebase/firestore'): לעדכון מסמכי Firestore, כגון הוספת הודעות חדשות או עדכון פרטי חדר צ'אט.
4. auth (מיובא מ-'firebase/..'): מייצג את אובייקט Firebase Storage ומשמש לאחסון ואחזור קבצים.
5. v4 as uid (מיובא מ-'uid'): ספרייה להפקת מזהים ייחודיים עבור משתמשים, קבצים וחדרי צ'אט.
6. getDownloadURL, ref, uploadBytesResumable, getMetadata: פונקציות מספריית Firebase Storage המשתמשות להעלאות והורדות של קבצים אל/מ מסד הנתונים.
7. signOut (מיובא מ-'firebase/auth'): פונקציה ליציאה מהמשתמש.

<sup>15</sup>Introducing hooks. React. (n.d.-a). <https://legacy.reactjs.org/docs/hooks-intro.html> (Introducing Hooks – React, n.d.)

<sup>16</sup>

<https://morioh.com/p/52eb73408b9a#:~:text=In%20this%20brief,in%20the%20tree.> what is a consumer in react (What Is A Consumer in React ?, n.d.)

8. auth (מיובא מ'..'firebase/'): מייצג את אובייקט האימות של Firebase, לניהול מצב אימות משתמש או אחזור מידע משתמש.

9. collection, query, where, getDocs, setDoc, getDoc (מיובא מ'..'firebase/firestore'): פונקציות מספריית Firebase Firestore לשאילתה, אחזור ועדכון מסמכים.

### socket.io:

1. io (מיובא מ'..'socket.io-client'): ספריה ליצירת חיבורי WebSocket, משמשת ליצירת תקשורת עם השרת.

2. socket.io: ספריה לאפשר תקשורת דו-כיוונית בזמן אמת בין לקוחות ושרתים באמצעות פרוטוקול WebSocket.

### node.js:

1. HTTP ב-Node.js נועדו לתמוך במאפיינים רבים של הפרוטוקול שהיו קשים לשימוש, במיוחד הודעות גדולות. הממשק מקפיד אף פעם לא לשמור במאגר בקשות או תגובות שלמות, כך שהמשתמש יכול להזרים נתונים.

2. express: מסגרת פופולרית של Node.js לבניית יישומי אינטרנט וממשקי API. הוא מספק קבוצה של תכונות ותוכנות ביניים כדי לפשט את תהליך הפיתוח. השתמשתי ליצירת והפעלת השרת.

## מודלים/מחלקות שפיתחתי:

### Pages:

#### Home.jsx:

העמוד הראשי לאחר ההרשמה.

מציג בתוכו את הבאים: `<Chat>` `</Sidebar>`

#### Login.jsx:

העמוד מורכב מטופס שמכיל מייל וסיסמה, וכפתור 'Sign In' המאפשר למשתמש להיכנס לחשבון שלו. אם למשתמש אין חשבון, הוא יכול ללחוץ על הקישור 'הירשם' כדי ליצור חשבון חדש. פירוט השיטות המשמשות ברכיב ההתחברות:

- `useState` - הוק המשמש לניהול מצב הרכיב. במקרה זה, הוא משמש כדי להגדיר את מצב הודעת השגיאה ל-`false` בתחילה, וכדי להגדיר את הודעת השגיאה למחרוזת אם מתרחשת שגיאה במהלך תהליך ההתחברות.
- `useNavigate` - הוא הוק מחבילת `react-router-dom` המאפשרת לרכיב לנווט לדפים שונים באפליקציה. הוא משמש כאן כדי להפנות את המשתמש לדף הבית לאחר כניסה מוצלחת.
- `handleSubmit` - היא פונקציה אסינכרונית הנקראת כאשר לוחצים על כפתור 'כניסה'. היא קולטת אובייקט אירוע ומחלץ את האימייל והסיסמה משדות הקלט. לאחר מכן, היא משתמשת בפונקציה `SignInWithEmailAndPassword` של `Firebase` כדי להיכנס למשתמש באמצעות הדוא"ל והסיסמה שלו. אם הכניסה מוצלחת, הפונקציה מפנה את המשתמש לדף הבית. אם מתרחשת

שגיאה במהלך תהליך הכניסה, הוא מגדיר את מצב הודעת השגיאה למחרוזת המתארת את השגיאה.

### **Register.jsx:**

מגדיר טופס רישום משתמש. כאשר הטופס נשלח, הרכיב מנסה ליצור חשבון משתמש חדש עם האימייל והסיסמה שסופקו. אם זה מצליח, שם התצוגה, האימייל ותמונת הפרופיל של המשתמש מאוחסנים ב-Firestore, והמשתמש מנותב לדף הבית של האפליקציה. הפונקציה המחזירה תבנית JSX. התבנית כוללת את שם הפרויקט, טופס עם שדות קלט לשם המשתמש, אימייל, סיסמה, סיסמה שוב (לאישור), תמונת פרופיל וקישור לעמוד הכניסה. הטופס כולל לחצן שלח והצגת הודעת שגיאה אם מתרחשת שגיאה במהלך תהליך ההרשמה. מגדיר משתנה מצב שנקרא err באמצעות ה-useState. משתנה זה משמש לאחסון כל שגיאה המתרחשת במהלך תהליך הרישום. אם מתרחשות שגיאות כלשהן במהלך תהליך הרישום, הודעת השגיאה מאוחסנת במשתנה מצב השגיאה ומוצגת בתצוגת הודעת השגיאה.

**Context:****ChatContext.js-**

מגדיר רכיב "ChatContextProvider" המספק "ChatContext" לכל ילדיו/הצרכנים שלו בשיטת "createContext" מ-React. ניתן לגשת להקשר הזה על ידי כל רכיב צאצא שהוא 'consumer' של "ChatContext".

על מנת לספק מזהה ייחודי לכל צ'אט, הפונקציה "chatReducer" מוגדרת לניהול עדכוני מצב. ה-"chatReducer" לוקח את ה-"state" הנוכחי ו-"action" שנשלחת כדי לעדכן את המצב הנוכחי. הפעולה ששינוי במצב יכול לעדכן היא "CHANGE\_USER" שמכילה בתוכה את אובייקט המשתמש.

כאשר פעולת "CHANGE\_USER" נשלחת, "chatReducer" מחזיר אובייקט מצב חדש עם ערכי המשתמש וה- chatId המעודכנים. ה- chatId נוצר על ידי ה-UID של המשתמש הנוכחי והמשתמש האחר בצ'אט.

**AuthContext.js-**

מייצא שני רכיבים: AuthContext ו-AuthContextProvider ואחראי לניהול מצב האימות של המשתמש באמצעות אימות Firebase.

- ה-AuthContext נוצר באמצעות שיטת createContext מספריית React. זה יוצר אובייקט הקשר שניתן להשתמש בו כדי לשתף נתונים בין רכיבים ללא צורך בהעברת אביזרים ידנית בכל רמה.
- הרכיב AuthContextProvider מגדיר מאזין אימות באמצעות הפונקציה onAuthStateChanged מ-Firebase. זה מעדכן את מצב המשתמש הנוכחי בכל פעם שמצב האימות משתנה.
- מגדיר את המצב ההתחלתי של אובייקט ה-currentUser לאובייקט ריק באמצעות ה-useState מ-React.
- "נרשם" לאירוע פיירבייס - onAuthStateChanged, המופעל בכל פעם שמצב האימות של המשתמש משתנה.
- כאשר המצב משתנה, הפונקציה setCurrentUser מעדכנת את הערך של currentUser עם מצב המשתמש החדש.
- ה-useEffect משמש כדי להירשם לאירוע onAuthStateChanged כאשר הרכיב מותקן.
- פונקציית החזרה בתוך ה-useEffect משמשת לביטול הרשמה לאירוע כאשר הרכיב אינו מותקן.

משמש כדי לספק גישה למשתמש המאומת הנוכחי בכל האפליקציה. הוא מספק מיקום מרכזי לגישה ולעדכון מצב האימות של המשתמש.

## Components:

### Chat.jsx-

רכיב ה-Chat מייצג את ממשק השיחה הראשי, כולל כותרת המציגה את שם המשתמש, הקבצים המשותפים ושדה קלט להוספה ושליחה של קובץ חדש.

רכיב ה"צ'אט" אחראי לעיבוד ממשק צ'אט שבו משתמשים יכולים להחליף הודעות. הוא משתמש ב-Socket.IO לתקשורת בזמן אמת עם שרת ומסתמך על ההקשרים "ChatContext" ו-"AuthContext" כדי לגשת לנתונים הקשורים לצ'אט ולנתונים הקשורים למשתמש הנוכחי.

השימוש ב-useEffect: מגדיר מאזין אירועים לאירוע "הודעה" הנפלט על ידי השרת בשיטת "socket.on". כאשר אירוע ה"הודעה" מתקבל, הוא רושם הודעת הצלחה המציינת שדוח התקבל בהצלחה. פונקציית הניקוי, המוחזרת על ידי ה-useEffect, אחראית להסרת המאזין לאירועים בשיטת "socket.off" כאשר הרכיב אינו פעיל.

הרכיב מייבא לתוכו שני רכיבים אחרים עליהם נדון בהמשך ("הודעות" - קבצים, וקלט).

פעולה במחלקה - `sendMessage`:

- `sendMessage` מופעלת כאשר המשתמש לוחץ על כפתור שליחת הדיווח באתר.
- לאחר הלחיצה הפונקציה בונה מחרוזת הודעה המבוססת על השם של המשתמש הנוכחי (מאוחר מה-AuthContext) ושם המשתמש המדווח (מאוחר מה-ChatContext).
- הפונקציה משדרת את ההודעה שנבנתה לשרת באמצעות שיטת `socket.emit`, ושולחת אותה כאירוע 'הודעה'. הפונקציה לא מחזירה שום ערך.
- הפונקציה `socket.emit` היא חלק מספריית Socket.IO ומשמשת לשליחת אירוע מותאם אישית ומוגדר מהלקוח לשרת.

### Chats.jsx-

רכיב ה-Chats אחראי לעיבוד רשימת כל הצ'אטים שהמשתמש הנוכחי הוא חלק מהם - מערך של צאטים (ראה [chats במוד הנדונים](#)). כל צ'אט מיוצג על ידי רכיב UserChat נפרד, המכיל את תמונת פרופיל המשתמש של הצ'אט, שם המשתמש והקובץ האחרון שנשלח לו/ממנו.

- הרכיב משתמש ב-`useState` כדי לשמור על המצב של מערך הצ'אטים, שהוא בתחילה מערך ריק. כאשר הרכיב נטען, הוא משתמש ב-`useEffect` כדי להביא את נתוני הצ'אטים מ-Firebase באמצעות שיטת `onSnapshot` מחבילת `firebase/firestore`.
- שיטה זו מגדירה "מאזין חי" [בממך userChats](#) עבור ה-UID של המשתמש הנוכחי, כך שכל שינוי במסמך יפעיל עיבוד מחדש של הרכיב עם הנתונים המעודכנים. הפונקציה `unsubscribe` המוחזרת על ידי `onSnapshot` משמשת לביטול הרישום מהמאזין כאשר הרכיב מתבטל.
- כאשר נתוני הצ'אטים מובאים, הרכיב ממפה את מערך הצ'אטים כדי להציג את רכיב ה-UserChat עבור כל צ'אט. רכיב [userChat](#) ממוינים בסדר יורד על סמך זמן שליחת ההודעה האחרונה בהם.

- הרכיב משתמש גם ב-`useContext` כדי לגשת לערכי המשתמש הנוכחיים ומצב הצאט מה-`AuthContext` ו-`ChatContext`.
- `handleSelect`: פונקציה זו משמשת לשליחת פעולה ל-`ChatContext` כאשר משתמש בוחר צ'אט מהרשימה. פעולה זו מעדכנת את מצב הצ'אט הנוכחי כדי להציג את ההודעות עבור הצ'אט שנבחר.

### Input.jsx-

- הרכיב אחראי על שדה הקלט וכפתור לשליחת הקבצים.
- `handleSend()`: פועלת בלחיצה על כפתור השליחה. הוא בודק אם קובץ הקלט תקין. לאחר מכן, הוא מעלה את הקובץ לאחסון Firebase כמו שמתואר בפירוט על מסד הנתונים.
- `setFile()`: שיטה זו נקראת כאשר המשתמש בוחר קובץ מהמערכת המקומית שלו. זה מגדיר את התמונה שנבחרה למצב של הרכיב.
- `setFileUrl()`: נקראת כאשר המשתמש בוחר קובץ מהמערכת המקומית שלו. מגדיר את ה-`downloadURL` שנוצר לקובץ לערך הרכיב.
- הרכיב משתמש גם ב-`useContext` כדי לגשת לערכי המשתמש הנוכחיים ומצב הצאט מה-`AuthContext` ו-`ChatContext`.

### Message.jsx-

- רכיב ה"הודעה" אחראי על עיבוד קובץ בודד בתוך ממשק הצ'אט. הוא מקבל את נתוני הקובץ כעזר ומנצל את ה-`contexts` ב-`AuthContext` ו-`ChatContext` כדי לגשת לנתונים הקשורים לאימות ולנתונים הקשורים לצ'אט, בהתאמה.
- מייבא הוקים של `useRef` ו-`useEffect` מ-`React`, כמו גם את ה-`context` של `AuthContext` ו-`ChatContext`.
- מקבל את הקובץ לעיבוד ומשתמש ב-`currentUser` ובאובייקט `data` מה-`ChatContext` כדי לקבוע את פרטי שולח ההודעה ואת המשתמש שאיתו המשתמש הנוכחי משוחח.
- משתמש ב-`useRef` כדי לקבל הפניה לרכיב שמכיל את הקובץ, וב-`useEffect` כדי לגלול את השיחה לתחתית המסך בכל פעם שמתקבלת או נשלחת הודעה חדשה.
- לאחר מכן, נעשה עיבוד של הקובץ, כולל של תמונת הפרופיל של השולח. אם המשתמש הנוכחי שלח את ההודעה, ההודעה מוצגת בצד ימין של המסך, אחרת היא מוצגת בצד שמאל של המסך.

### Messages.jsx-

- מייבא את הקבצים ממסד הנתונים של `Firestore` ומציג אותן באמצעות רכיב ההודעה.
- ה-`useEffect` מגדיר "מאזין" לתמונת המצב באוסף ההודעות של הצ'אט הנוכחי. כאשר הרכיב נטען, הוא קורא לשיטת `onSnapshot` באובייקט ה-`doc`, אשר מחזירה פונקציית `callback` שתקרא בכל פעם שהנתונים משתנים.
- בפונקציית ה-`callback`, שיטת `exists` משמשת כדי לבדוק אם המסמך קיים (ייתכן שהוא לא קיים אם הצ'אט נמחק), ואם כן, מערך ההודעות מוגדר לערך של שדה ההודעות של המסמך.
- רכיב ההודעות ממין את רכיבי ההודעה עבור כל הודעה במערך ההודעות בעזרת `map`.

**Navbar.jsx-**

רכיב "Navbar" אחראי על רינדור סרגל הניווט בחלק העליון של ממשק האפליקציה. הוא מציג את הלוגו, תמונת הדמות של המשתמש הנוכחי, שם התצוגה שלו וכפתור יציאה. הוא משתמש בהקשר "AuthContext" כדי לגשת לנתונים הקשורים למשתמש הנוכחי.

- ה-AuthContext משמש כדי לקבל את אובייקט ה-currentUser שמועבר לרכיב.
- פונקציית ה-signOut מיובאת ממודול Firebase/Auth ונקראת כאשר המשתמש לוחץ על כפתור היציאה.

**Search.jsx-**

**רכיב החיפוש המאפשר למשתמשים למצוא משתמשים אחרים לפי שמות המשתמש שלהם ולהתחיל איתם שיחה בצאט חדש משלהם.**

- הרכיב מייבא מספר פונקציות מספריית Firebase Firestore כדי לבצע שאילתות ולעדכן את מסד הנתונים.
- מייבא את AuthContext כדי לקבל את פרטי המשתמש הנוכחיים.
- הקוד משתמש בפונקציית io מהמודול socket.io-client כדי ליצור חיבור WebSocket עם השרת.
  - "מטפל האירועים" socket.on מאזין להודעות נכנסות מהשרת.
  - הפונקציה socket.emit שולחת הודעה לשרת.
  - המטפל והפונקציה באירוע משמשים ב-useEffect Hook כדי לטפל בתקשורת עם השרת.
  - מגדיר שם משתמש של משתנה מצב (state variable) לאחסון שאילתת החיפוש, ומשתמש במשתנה מצב לאחסון אובייקט המשתמש שהוחזר על ידי החיפוש.
- מגדיר פונקציה handleSearch שמחפשת את אוסף המשתמשים במסד הנתונים של Firestore עבור משתמשים עם שם התצוגה שצוין בשם המשתמש של משתנה המצב. הפונקציה handleSearch מבצעת את שאילתת Firestore באופן אסינכרוני באמצעות שימוש בתהליכון נפרד כדי למנוע חסימה של התהליכון הראשי בזמן ההמתנה לתוצאות השאילתה. זה מאפשר חווית משתמש חלקה יותר, שכן ממשק המשתמש נשאר פעיל במהלך פעולת החיפוש.
  - אם נמצא משתמש, אובייקט המשתמש מאוחסן במשתנה המצב של המשתמש.
  - אם לא נמצא משתמש, משתנה מצב השגיאה מוגדר כ-true.
- מגדיר פונקציה handleKey שמפעילה את פונקציית handleSearch כאשר מקש Enter נלחץ.
- מגדיר פונקציה handleSelect שיוצרת צ'אט בין המשתמש הנוכחי למשתמש הנבחר אם הצ'אט לא קיים כבר.

- הצ'אט נוצר על ידי הוספת מסמך לאוסף הצ'אטים עם מזהה המבוסס על "תעודת הזהות" של המשתמש הנוכחי והמשתמש הנבחר.
- מסמך הצ'אט מכיל מערך הודעות שבהתחלה ריק.
- לאחר מכן, אוסף ה-userChats של שני המשתמשים מתעדכן עם [הנתונים המפורטים](#).

### Sidebar.jsx-

רכיב הסרגל הצידי מכיל את `<Chats />` `<Search />` `</Navbar />`, ואחראי על רינדורם. הוא כולל את סרגל הניווט, פונקציונליות החיפוש ורשימת צ'אטים.

### App.js:

נקודת הכניסה הראשית של הפרויקט. הוא מגדיר את הניתוב עבור דפים שונים, מטפל בהליך כניסה לאחר אימות ומעבד את הרכיבים המתאימים על סמך המסלול הנוכחי. הוא מייבא וכולל את הרכיבים הבאים: "בית", "כניסה" ו"הרשמה".

- הגדרת מסלולים לדפי הבית, התחברות והרשמה, כאשר עמוד הבית הוא מסלול מוגן.
- הגדרת ProtectedRoute שבדוק אם המשתמש מאומת, ואם לא, הוא מנווט אותו לדף הכניסה.
- המסלולים לדפים השונים דרך ה"Route" הרגיל וה-"protectedRoute" נעשה על ידי ייבוא של "react-router-dom".

### Index.js:

הצגת רכיב ה-App.

- הפונקציה createRoot מחבילת ReactDOM משמשת ליצירת שורש לאפליקציה.
- הפונקציה root.render משמשת לעיבוד רכיב ה-App בתוך אלמנט ה-'root'.

### style.scss:

עיצוב האתר.

### server.js:

מגדיר שרת WebSocket באמצעות ספריית Socket.IO. להלן פירוט של הקוד והפונקציונליות שלו:

- הקוד מייבא את המודולים הדרושים: http, express ו-socket.io. מודולים אלו נדרשים כדי ליצור שרת HTTP, להגדיר יישום Express ולהשתמש בספריית Socket.IO לתקשורת WebSocket.
- אפליקציית Express נוצרת על ידי קריאה ל-express(), ושרת HTTP נוצר באמצעות אפליקציית Express על ידי קריאה ל-http.createServer(app). שרת זה יטפל בבקשות HTTP נכנסות.
- מופע Socket.IO נוצר על ידי העברת מופע השרת ל-socketIO(שרת). זה מאפשר תקשורת WebSocket בין השרת ללקוחות.
- מטפל האירועים ('connection', 'io.on', ...) אחראי לטיפול בחיבורי socket. כאשר לקוח מתחבר לשרת, הקוד רושם הודעה המציינת שלקוח התחבר.



- בתוך המטפל באירועי החיבור, קיים מטפל נוסף באירועים עבור אירוע 'הודעה'. מטפל באירועים זה מאזין לאירועי 'הודעות' נכנסות מלקוחות. כאשר מתקבל אירוע 'הודעה', תוכן ההודעה נרשם, וההודעה משודרת לכל הלקוחות המחוברים באמצעות io.emit('הודעה', הודעה). זה מאפשר תקשורת בזמן אמת בין השרת והלקוח (עבור הודעות הדיווח על משתמשים).
- בנוסף, קיים מטפל באירועים לאירוע 'ניתוק'. הוא רושם הודעה כאשר לקוח מתנתק מהשרת. לבסוף, השרת מתחיל להאזין ביציאה שצוינה, אשר מסופקת על ידי משתנה הסביבה process.env.PORT או ברירת המחדל היא יציאה 3000. מספר היציאה נרשם ברגע שהשרת מתחיל לפעול. לסיכום, רכיב זה מגדיר שרת WebSocket באמצעות Socket.IO ומאפשר תקשורת בזמן אמת בין לקוחות. הוא מטפל בחיבורי לקוחות, אירועי הודעות, ניתוקים ומשדר הודעות לכל הלקוחות המחוברים.

## חלק ב' - פתרון הבעיה האלגוריתמית

- עדכונים בזמן אמת:

כמו שציינתי, בחרתי לעשות שילוב בהבנה וביכולות של פונקציות של Firebase יחד עם הוק `useEffect` של ריאקט כדי ליצור את הפתרון האולטימטיבי בעניי לבעיה שעמדה מולי. השתמשתי בפתרון זה במספר רכיבים שאת הקוד שלהם אוסיף בהמשך, אך אסביר ואפרט לעומק עם דוגמה מרכזית מהקוד של Chats:

### פירוט הפתרון המוצג בשלבים:

- מגדיר את מערך הצאטים שהמשתמש חלק מהם כמערך ריק עם מצב - כלומר אפשרות לעדכן אותו.

```
const [chats, setChats] = useState([]); {/*state variable for storing chats*/}
```

- "צורך" את המשתמש הנוכחי מ `AuthContext` כדי לעדכן לפיו.

```
const { currentUser } = useContext(AuthContext); // get current user from AuthContext
```

```
const { dispatch } = useContext(ChatContext);
```

- מגדיר `useEffect`, הוק אשר מופעל פעם אחת בטעינת הרכיב ולאחר מכן כל פעם שמשתנה משהו שהגדרנו לו - במקרה זה: המזהה הייחודי של המשתמש הנוכחי.

```
useEffect(() => {
```

- מגדירים פעולה שהמטרה שלה זה לקבל את הצאטים שהמשתמש חלק מהם (באותו הרגע!)

```
const getChats = () => {
```

- בעזרת פונקציית `onSnapshot` אנחנו מגדירים מאזין בזמן אמת לקובץ ספציפי במסד הנתונים שלנו - כפי שניתן לראות ה `"userChats"` של המשתמש הנוכחי. התוצאה מועברת לפרמטר בשם `"doc"`.

```
const unsub = onSnapshot(doc(db, "userChats", currentUser.uid),  
(doc) =>
```

```
{
```

- משתנה `stater` של `chats` שהגדרנו בהתחלה ל `doc` (הפרמטר עם המידע העדכני). המידע.

```
setChats(doc.data());
```

```
});
```

```
return () => {
```

```
unsub();
```

```

    };

    };

    currentUser.uid && getChats(); /*get chats if currentUser exists
*/}

}, [currentUser.uid]);

```

### -Chats מתוך מסודר

```

const [chats, setChats] = useState([]); /*state variable for storing
chats*/}

const { currentUser } = useContext(AuthContext); // get current user
from AuthContext
const { dispatch } = useContext(ChatContext);

useEffect(() => {
    const getChats = () => {
        const unsub = onSnapshot(doc(db, "userChats", currentUser.uid),
(doc) =>
        {
            setChats(doc.data());
        });

        return () => {
            unsub();
        };
    };

    currentUser.uid && getChats(); /*get chats if currentUser exists
*/}

}, [currentUser.uid]);

```

### דוגמה נוספת שבה נעשה שימוש בלוגיקה דומה:

### -Messages מתוך

```

const [messages, setMessages] = useState([]) // Get the messages for
the current chat, if any, and set them in the state.
const {data} = useContext(ChatContext);

```

```
useEffect(() => {  
  const unSub = onSnapshot(doc(db, "chats", data.chatId), (doc) => {  
    doc.exists() && setMessages(doc.data().messages)  
  })  
  
  return () => {  
    unSub()  
  }  
}, [data.chatId])
```

## חלק ג - מסך בדיקות מלא

### בדיקות מתוכננות:

#### 1. בדיקת העלאת קבצים:

- מטרת הבדיקה: לוודא שכל המשתמשים יוכלו להעלות קבצים מכל הסוגים (גם קבצים גדולים) בכדי לא להגביל את המשתמשים לסוג קבצים מסוים.
- ביצוע בפועל: בדיקה באתר של Firebase בנוגע להגבלות שלהם, ניסיון לשלוח קבצי טקסט, שמע, סרטונים, מצגות, תוכניות ועוד.
- תוצאות הבדיקה: אין הגבלה לסוגי הקבצים כל הקבצים הועלו בהצלחה
- האם היו בעיות? אם כן, כיצד נפתרו: לא היו בעיות.

#### 2. בדיקת שיתוף קבצים:

- מטרת הבדיקה: לוודא שהקבצים המשותפים שנוצרו מוצגים בצאט של המשתמש השני (שאליו נשלח) כדי לראות שמשתמשים לא יתקלו בבעיה זאת.
- ביצוע בפועל: פתחתי מספר משתמשים שדרכם שלחתי עם כל אחד קבצים שונים לכל האחרים, בדקתי האם כל הקישורים אכן נשלחו לצד השני.
- תוצאות הבדיקה: כל הקבצים הגיעו.
- האם היו בעיות? אם כן, כיצד נפתרו: לא היו בעיות משמעותיות, אך שם הקובץ לעולם היה נראה אצל המשתמש לפני השליחה כשמו בתקיית הקבצים, ולאחר השליחה כשם הקישור מה שלא היה נראה טוב. פתרתי זאת בכך שיצרתי משתנה מצב ברכיב message וקראתי לו filename שלקח את שם הקובץ מהקובץ שהורד והציג אותו בצורה הזאת למשתמש בהודעה.

#### 3. בדיקת הורדת קבצים:

- מטרת הבדיקה: לבדוק שהקישור שנוצר ונשלח להורדת הקובץ לאחר שליחתו עובד כדי למנוע בעיות למשתמשים.
- ביצוע בפועל: שלחתי מספר קבצים שונים בין שני משתמשים וניסיתי ללחוץ על כפתור ההורדה בהודעות ישנות וחדשות כאחד (אצל שניהם) על כל הקבצים ובדקתי האם הקישור עבד.
- תוצאות הבדיקה: הקישורים אכן עבדו.
- האם היו בעיות? אם כן, כיצד נפתרו: לא נתקלתי בבעיה טכנית אך שמתי לב שמבחינה עיצובית עדיף להוסיף כפתור לחיץ בשם הורדה שצמוד ולהודעה ודרכו ניתן להוריד את הקובץ וכך עשיתי.

#### 4. אימות משתמש קיים:

- מטרת הבדיקה: לבדוק את תהליך הכניסה לאתר ואימות המשתמשים כדי למנוע פריצות או בעיות בהליך האימות.

- ביצוע בפועל: ניסיתי להיכנס למשתמש שידעתי שקיים עם ערכי סיסמה שגויים, להכניס ערכים לא קיימים, לנסות למצוא סיסמה של משתמש מספר רב של פעמים (דימוי למתקפת brute force)
- תוצאות הבדיקה: לאחר התיקונים שהצגתי בסעיף הבא אכן הבדיקה הצליחה - לא הצלחתי להתחבר למשתמש ללא ערכיו הנכונים, כשהכנסתי ערכים לא קיימים לא קיבלתי גישה גם כן (עמוד כניסה לא הרשמה!), וכשניסיתי מספר רב של פעמים להתחבר לאותו משתמש נחסמה לי האפשרות לכמות זמן מסוימת לנסות שוב.
- האם היו בעיות? אם כן, כיצד נפתרו: בהתחלה טעות הייתה גוגרת קריסה לא ברורה או סתם חוסר פעילות אך בעזרת try&catch הצלחתי לתפוס את השגיאות ולהדפיס את השגיאה המתאימה כל פעם.

#### 5. בדיקת עדכונים בזמן אמת:

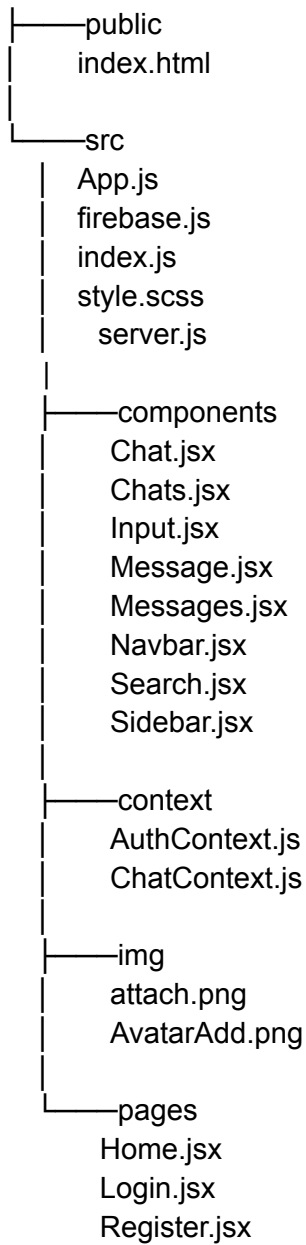
- מטרת הבדיקה: לוודא שמשתמשים מצליחים לעשות שיתוף קבצים אחד עם השני בזמן אמת ללא דיליי מסוים.
- ביצוע בפועל: פתחתי שני משתמשים במקביל ושלחתי קבצים אחד לשני במקביל ובדקתי אם הקבצים עודכנו בזמן אמת עם החתם זמן שלהם בfirebase ובמו עיניי בפעילות ברשת (בהנחה שהיה לי חיבור תקין לרשת).
- תוצאות הבדיקה: הקבצים אכן נשלחו במקביל בהצלחה (מלבד הבדל זעיר התלוי בהפרש הזמן בין הלחיצה שלי על כפתור השליחה במחשב).
- האם היו בעיות? אם כן, כיצד נפתרו: לא נתקלתי בבעיה לאחר [שפתרתי את הבעיה הזאת](#) מוקדם יותר בהליך פיתוח הפרויקט.

#### בדיקות נוספות:

#### 6. פתיחת משתמש חדש:

- מטרת הבדיקה: לבדוק שאפשר לפתוח משתמש חדש בהצלחה ושנתוניו מתעדכנים במסד הנתונים ובנוסף לוודא שבעת שגיאה אין קריסה ומוצגת הודעת השגיאה המתאימה.
- ביצוע בפועל: ניסיתי להירשם עם ערכים חסרים, ערכים לא מתאימים, ערכי סיסמה וסיסמה לווידוא לא תואמות וכדומה.
- תוצאות הבדיקה: לא הצלחתי להירשם כאשר הערכים לא עמדו בדרישות שהוגדרו בקוד.
- האם היו בעיות? אם כן, כיצד נפתרו: כמו בעמוד ההרשמה הייתי צריך להוסיף try&catch כדי לתפוס את השגיאות (אם היו) ולהדפיס אותן בהצלחה למשתמש.

# מדריך למשתמש

עץ קבצים מרכזיים:דרישות מקדימות לפני שימוש-

- כל מכשיר עם חיבור לרשת.



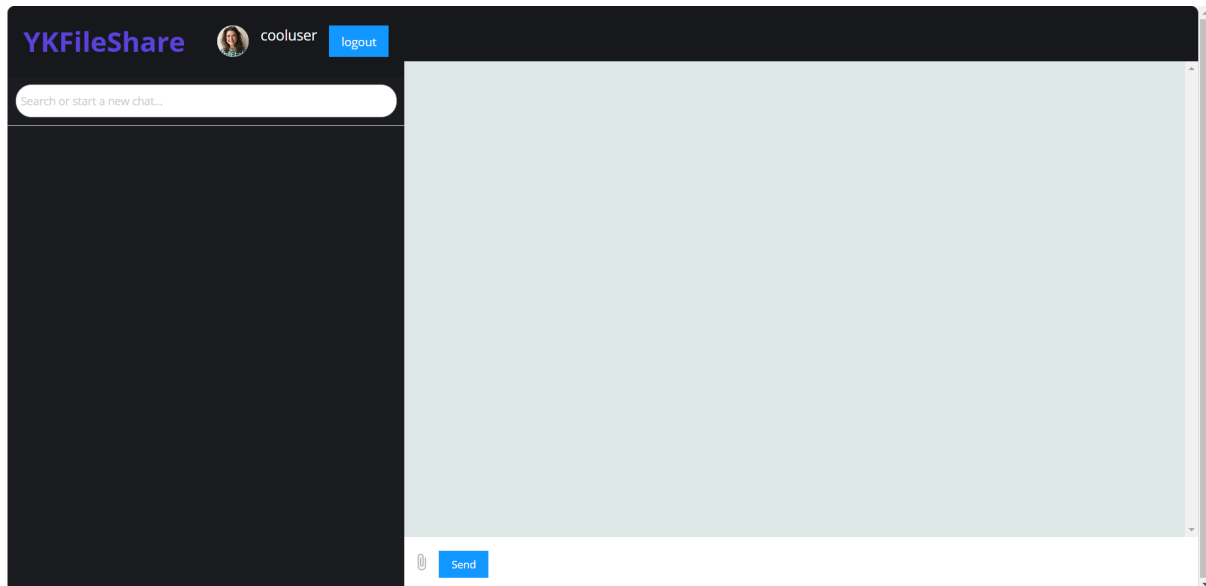
משתמשי המערכת:

- אופן הפעלת המערכת: התחברות לכתובת האתר.
  - מסכי הפרויקט:
1. **עמוד ההתחברות:** זה העמוד שתקבלו כשתכנסו לכתובת האתר בפעם הראשונה. אם כבר יש לכם משתמש קיים, תזינו את המייל והסיסמה שלכם ותלחצו על כפתור הכניסה, אם אין לכם תלחצו על כפתור ההרשמה!

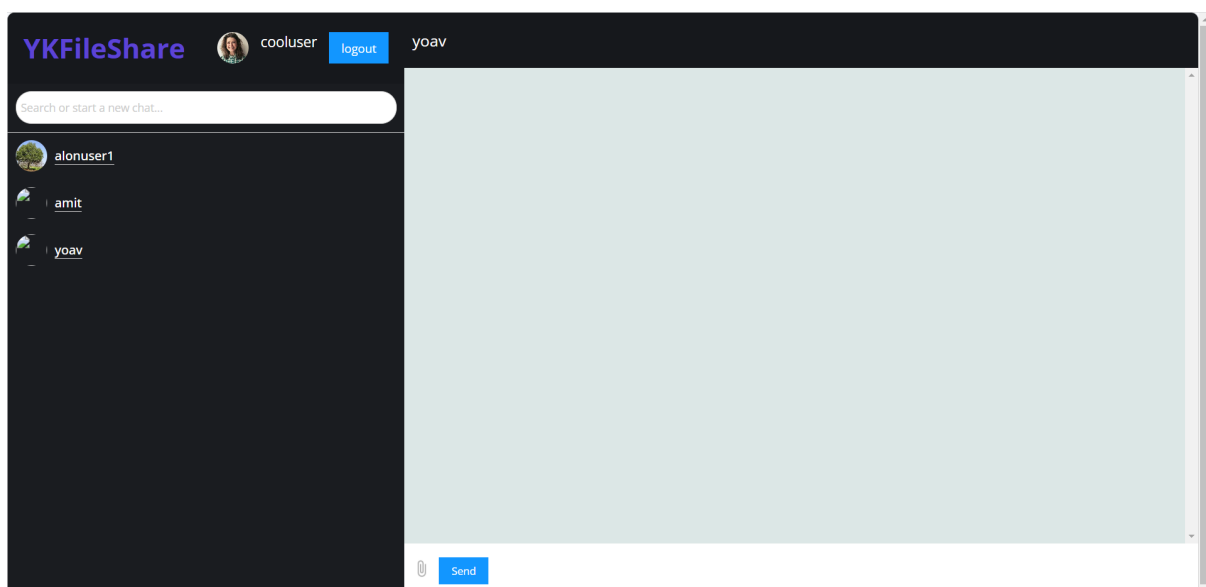
2. **עמוד ההרשמה:** זה העמוד שמגיעים אליו לאחר לחיצה על כפתור ההרשמה בעמוד הכניסה. בעמוד הזה יש לכם את האופציה לפתוח משתמש חדש עם הפרטים הבאים לבחירתכם: שם משתמש, מייל, סיסמה ותמונת אוטאר (לא חובה). לאחר שמילאתם את הפרטים ועמדתם בדרישות (אף שדה לא ריק, סיסמה והאימות שלה זהות, סיסמה מעל 6 תווים, מייל בפורמט תקין וכו'). יפתח לכם המשתמש החדש!

3. בין אם עשיתם התחברות או הרשמה, אם עשיתם זאת בהצלחה תגיעו לעמוד הבא (עם שם המשתמש ותמונת האוואטר שלכם!) -

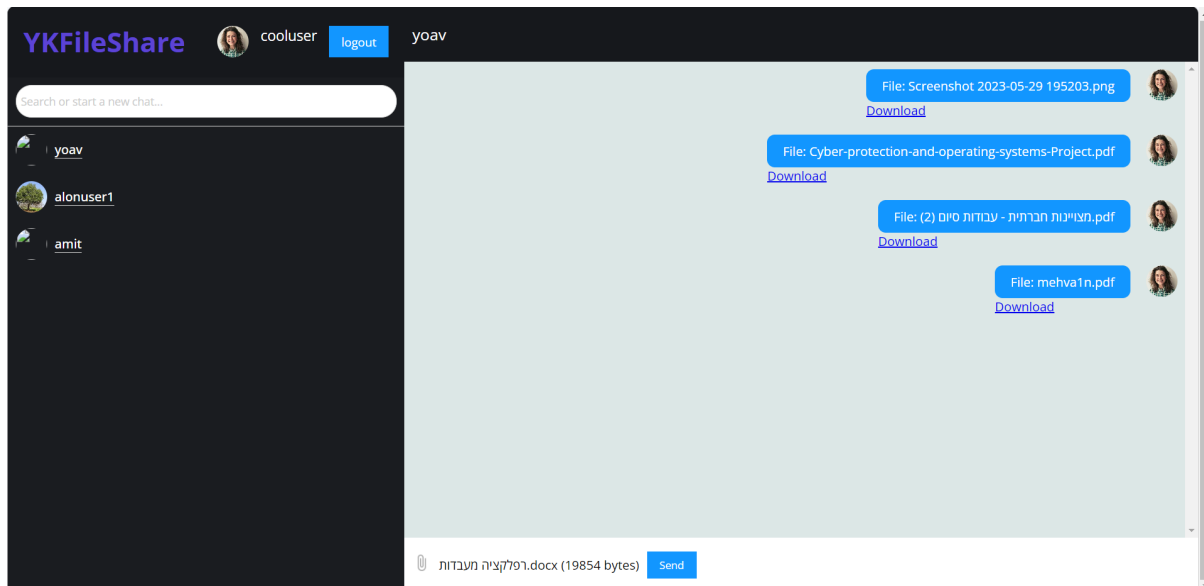
- בתיבת החיפוש מצד שמאל למעלה תוכלו לחפש שמות משתמשים אחרים באתר כדי לפתוח איתם צאט ששם תוכלו לשתף איתם קבצים.
- בלחיצה על כפתור הlogout תתנתקו מהמשתמש שלכם!



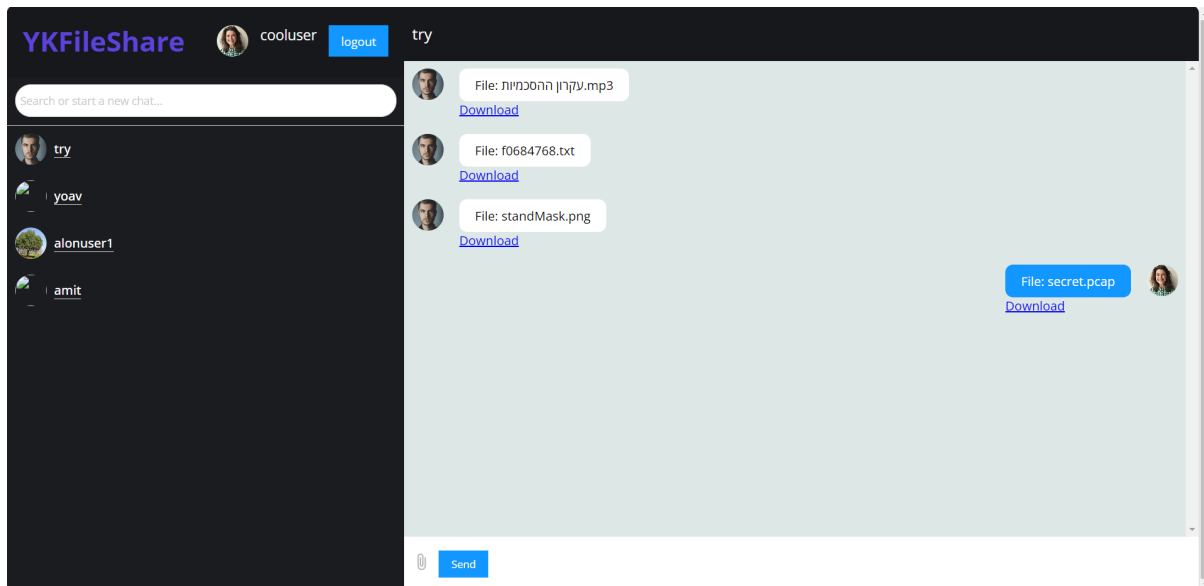
לאחר חיפוש, מציאת והוספת מספר צאטים חדשים המסך שלכם יראה כך:



כעת, יש לכם אפשרות לשתף קבצים עם המשתמשים שאיתם יש לכם צאט קיים!  
 בלחיצה על כפתור הקובץ שנמצא למטה ליד כפתור הsend, יפתח לכם תיקיית הקבצים שלכם, משם תוכלו  
 לבחור קובץ לשליחה!  
 יוצג לכם שם הקובץ וגודלו:



שיחה אחרת לדוגמה שבה שלחתם ונשלחו אליכם קבצים תראה כך:



יש לכם אפשרות ללחוץ על כפתור הDownload שנמצא ליד כל הודעה כדי להוריד את הקובץ שנשלח.  
 בנוסף, תוכלו ללחוץ על כל משתמש שנמצא מצד שמאל כדי לפתוח את הצאט איתו (תדעו שזה קרה  
 בהצלחה כאשר השם למעלה ישתנה!).

# סיכום אישי / רפלקציה

• תיאור תהליך העבודה על הפרויקט, ההצלחות, האתגרים, הקשיים ודרכי הפתרון

בניתי אפליקציית שיתוף קבצים מבוססת web. תהליך העבודה כלל מספר שלבים, בניהם למידת הרקע התיאורטי, התנסות עם מיני פרויקטים בשפת ריאקט, פיתוח ממשק המשתמש וחיבור עם מסד הנתונים.

אחת ההצלחות המרכזיות של הפרויקט הייתה יישום מוצלח של פונקציונליות שיתוף הקבצים. משתמשים יכולים להעלות קבצים בידיעה שהם יאוחסנו במסד נתונים וישלחו בהצלחה. זה דרש ממנו הבנה עמוקה במסד הנתונים של firebase וביכולותיה של שפת ריאקט.

עם זאת, הפרויקט הציב גם כמה אתגרים. אחד הקשיים היה מימוש עצמי של תקשורת מבוססת סוקטים, שכן לא היה לי צורך משמעותי בכך לפרויקט שלי. כדי להתגבר על האתגר הזה, הקדשתי זמן ללימוד עצמי וחקרתי רבות כדי להבין את המושגים ביסודיות ולאחר מכן השתמשתי במימוש זה עבור שני צרכים עיקריים:

1. צורך מחקרי: להתנסות ולהראות מדוע שיטות שונות כמו polling הנוגעת לתכנות בסוקטים פחות עדיפות עבורי (שולחת כל הזמן בקשות עדכון במרווח זמן קבוע מראש - ניתן לראות בפרויקט הרץ).
2. ניסיון לימודי: חקירה והטמעה של Socket.IO בפרויקט, גם אם לא נעשה בו שימוש נרחב, סיפקה לי ניסיון וידע רב ערך בעבודה עם טכנולוגיות תקשורת בזמן אמת. אי לכך שגם Firebase בהעברת הנתונים שלהם עושים שימוש ב web sockets זה היה הכרחי עבורי לעשות מימוש לכך ולהבין את הנושא לעומק, שכן זה עזר לי אחרי בהבנת פיתוח תהליכי העברת המידע.

אתגר נוסף היה הבטחת אבטחת מידע ופרטיות נתונים. זה היה חיוני ליישם אמצעי אבטחה נאותים כדי להגן על נתוני המשתמש ולמנוע גישה לא מורשית. זה הצריך שיקול זהיר של מנגנוני אימות והרשאה, כמו גם טכניקות הצפנה. לאחר שהתייעצתי עם המנחה שלי, חברים קרובים שמבינים בתחום ועם אמא שלי אשר עובדת בתחום הטכנולוגיה, החלטתי לעבוד עם Firebase להליך האימות.

לאורך כל כתיבת הפרויקט הייתי בתהליך למידה מתמשך. למדתי באופן עצמאי מושגים חדשים הקשורים לבניית אפליקציות אינטרנט, כמו עבודה עם מסדי נתונים והבנת רשתות תקשורת. התהליך האינטנסיבי כלל לימוד מסוגים שונים: עם המנחה שלי, עצמאי, לימוד מתוך משאבים מקוונים ותרגול מעשי. זו הייתה ניסיון רב ערך שהרחיב את הידע והכישורים שלי בתחום.

קיבלתי הדרכה ותמיכה מהמנחה שלי, שמילא תפקיד משמעותי בפרויקט בקבלת החלטות בנוגע להחלטות שונות שהייתי צריך לקבל בנושאי הפיתוח. המומחיות והתובנות שלו סייעו לי בקבלת החלטות מושכלות. בנוסף, ביקשתי עזרה מחברי שעבדו על פרויקטים אחרים, מה שאפשר לנו לחלוק מידע וללמוד אחד מהניסיון של השני.

במבט לאחור, יש כמה היבטים של הפרויקט שהייתי מיישם אחרת. אחד מההיבטים היה להוסיף אפשרות לערוך את שם הקובץ לפני השליחה וגם להוסיף כפתור ברור לשינוי קובץ אם בחרת באחד לא נכון (כרגע פשוט צריך ללחוץ שוב על כפתור העלאת הקובץ). בנוסף, הייתי עורך בדיקות משתמשים רחבות יותר ממה שעשיתי (עשיתי עם משפחה וחברים) כדי לאסוף משוב ולשפר את חוויות המשתמש עוד יותר.

אם היו משאבים נוספים זמינים, הייתי קונה שם דומיין עם סיומת אשר נראת יותר אמינה לאתר - כרגע tech. אשר עלה לי מספר שקלים בלבד. יתר על כן, הייתי עושה מימוש/משתמש בAPI חיצוני בכדי לוודא את ביטחות הקבצים באופן אוטומטי - כלומר שכל קובץ לפני שליחתו והעלאתו למסד הנתונים יעבור בדיקה. לסיום, ברצוני להביע את תודתי למנחה שלי, הממונה על התמיכה והליווי לאורך תהליך פיתוח הפרויקט ולמשפחתי וחבריי שהקדישו לי מזמנם בכדי להתנסות בפרויקט ולתת לי חוות דעת לשיפורו.

# ביבליוגרפיה

1. (2023, January 5). geeksforgeeks. Retrieved May 31, 2023, from <https://media.geeksforgeeks.org/wp-content/uploads/20210908120846/DOM.png>
2. *Firebase vs WebSocket: Differences and how they work together*. (2022, November 22). Ably Realtime. Retrieved May 31, 2023, from <https://ably.com/topic/firebase-vs-websocket#how-firebase-and-web-socket-work-together-realtime-database>
3. *HTTPS – ויקיפדיה*. (n.d.). ויקיפדיה. Retrieved May 31, 2023, from <https://he.wikipedia.org/wiki/HTTPS>
4. *Introducing Firebase Authentication*. (n.d.). Firebase. Retrieved May 31, 2023, from <https://firebase.google.com/docs/auth>
5. *Introducing Firebase Realtime Database*. (n.d.). Firebase. Retrieved May 31, 2023, from <https://firebase.google.com/docs/database>
6. *Introducing Hooks – React*. (n.d.). React. Retrieved May 31, 2023, from <https://legacy.reactjs.org/docs/hooks-intro.html>
7. *Long polling*. (2022, December 12). The Modern JavaScript Tutorial. Retrieved May 31, 2023, from <https://javascript.info/long-polling>
8. *Perform simple and compound queries in Cloud Firestore | Firebase*. (n.d.). Firebase. Retrieved May 31, 2023, from <https://firebase.google.com/docs/firestore/query-data/queries>
9. *Quick Start – React*. (n.d.). React. Retrieved May 31, 2023, from <https://react.dev/learn>
10. *Untitled*. (2019, February 5). data | המרכז לחינוך סייבר. Retrieved May 31, 2023, from <https://data.cyber.org.il/networks/networks.pdf>

11. *Using WebSockets for two-way communication in React apps.* (2022, April 10). LogRocket Blog. Retrieved May 31, 2023, from <https://blog.logrocket.com/websockets-two-way-communication-react-app/>
12. *What Is A Consumer in React ?* (n.d.). Morioh. Retrieved May 31, 2023, from <https://morioh.com/p/52eb73408b9a>
13. *What is Firebase?* (n.d.). Educative.io. Retrieved May 31, 2023, from <https://www.educative.io/answers/what-is-firebase>
14. *What is Transmission Control Protocol (TCP)? Header, Definition - javatpoint.* (n.d.). Javatpoint. Retrieved May 31, 2023, from <https://www.javatpoint.com/tcp>

# נספחים

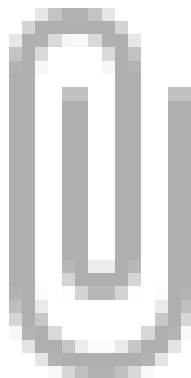


## **תמונות בפרויקט:**

1. הוספת תמונת משתמש - מופיע בעמוד ההרשמה:



2. העלאת קובץ לשליחה - מופיע בעמוד הבית:



```
.formContainer{
  background-color: #000;
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;

  .formWrapper{
    background-color: #222;
    padding: 30px;
    border-radius: 10px;
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 20px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);
    max-width: 400px;
    width: 100%;
    margin: 0 20px;

    .logoContainer {
      display: flex;
      align-items: center;
      gap: 5px;
    }

    .logo {
      color: #00ff00;
      font-size: 32px;
      font-weight: bold;
    }

    .subLogo {
      color: #00ff00;
      font-size: 20px;
    }
  }
}
```

```
.title {
  color: #fff;
  font-size: 24px;
  font-weight: bold;
  text-transform: uppercase;
  letter-spacing: 2px;
}

form{
  display: flex;
  flex-direction: column;
  gap: 20px;
  width: 100%;

  input {
    padding: 15px;
    border: none;
    border-radius: 5px;
    background-color: #333;
    color: #fff;
    outline: none;
    font-size: 16px;
  }

  button {
    background-color: #00ff00;
    color: #000;
    padding: 12px 20px;
    font-size: 16px;
    font-weight: bold;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
  }

  button:hover {
    background-color: #0f0;
  }
}
```

```
.error-message {  
    color: red;  
    font-size: 14px;  
    text-align: center;  
}  
  
.registerLink {  
    color: #fff;  
    font-size: 14px;  
    margin-top: 10px;  
    text-align: center;  
}  
  
.registerLink a {  
    color: #00ff00;  
    text-decoration: none;  
}  
  
.registerLink a:hover {  
    text-decoration: underline;  
}  
  
label{  
    display: flex;  
    align-items: center;  
    gap: 10px;  
    color: #333;  
    font-size: 14px;  
}  
}  
p{  
    color: #007bff;  
    font-size: 14px;  
    margin-top: 10px;  
}  
}  
}
```

```
.home {
  background-color: #f7f9fc;
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;

  .container {
    border: 1px solid #d3d3d3;
    border-radius: 10px;
    width: 100%;
    height: 100%;
    display: flex;
    overflow: auto;

    .sidebar {
      flex: 1;
      background-color: #1a1c20;
      position: relative;
      overflow: auto;

      .navbar {
        display: flex;
        align-items: center;
        background-color: #16181c;
        height: 50px;
        padding: 20px;
        justify-content: space-between;
        color: #fff;

        .logo {
          font-weight: bold;
          font-size: 35px;
          color: #5c43d5;
          display: block;
        }

        .user {
```

```
display: flex;
gap: 15px;
font-size: 18px;

img {
  background-color: #fff;
  height: 40px;
  width: 40px;
  border-radius: 50%;
  object-fit: cover;
}

button {
  background-color: #1296ff;
  color: #fff;
  font-size: 15px;
  border: none;
  cursor: pointer;
  padding: 8px 16px;
}
}

.search {
  border-bottom: 1px solid #d3d3d3;
  padding: 10px;

  .searchForm {
    display: flex;
    align-items: center;
    border: 1px solid #d3d3d3;
    border-radius: 20px;
    padding: 4px;
    background-color: #fff;

    input {
      background-color: transparent;
      border: none;
      width: 100%;
    }
  }
}
```

```
        height: 30px;
        font-size: 14px;
        color: #000;
        outline: none;

        &::placeholder {
            color: #ccc;
        }
    }
}

.userChat {
    padding: 10px;
    display: flex;
    align-items: center;
    gap: 10px;
    color: #fff;
    cursor: pointer;

    &:hover {
        background-color: #363b43;
    }

    img {
        width: 40px;
        height: 40px;
        border-radius: 50%;
        object-fit: cover;
    }

    .userChatInfo {
        span {
            font-size: 16px;
            font-weight: 500;
        }
        border-bottom: 1px solid #d3d3d3;
    }
}
```

```
}

.chat {
  flex: 2;
  background-color: #fff;
  border-radius: 10px;
  display: flex;
  flex-direction: column;

  .chatInfo {
    height: 50px;
    background-color: #16181c;
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 10px;
    font-size: 20px;
    color: #fff;
  }

  .chatIcons {
    display: flex;
    gap: 10px;

    img {
      height: 24px;
      cursor: pointer;
    }
  }

  .messages {
    background-color: #dce7e6;
    padding: 10px;
    flex: 1;
    overflow-y: scroll;

    .message {
      display: flex;
      gap: 20px;
    }
  }
}
```



```
margin-bottom: 20px;

.messageInfo {
  display: flex;
  flex-direction: column;
  color: #000;
  font-weight: 300;

  img {
    width: 40px;
    height: 40px;
    border-radius: 50%;
    object-fit: cover;
  }
}

.messageContent {
  max-width: 80%;
  display: flex;
  flex-direction: column;
  gap: 10px;

  p {
    background-color: #fff;
    padding: 10px 20px;
    border-radius: 10px;
    max-width: max-content;
  }

  img {
    width: 50%;
  }
}

&.owner {
  flex-direction: row-reverse;

  .messageContent {
    align-items: flex-end;

    p {
```

```
        background-color: #1296ff;
        color: #fff;
    }
}
}
}
```

```
.input {
    height: 50px;
    background-color: #fff;
    padding: 10px;
    display: flex;
    align-items: center;
    justify-content: space-between;
```

```
    input {
        width: 100%;
        border: none;
        outline: none;
        color: #000;

        &::placeholder {
            color: #ccc;
        }
    }
}
```

```
.send {
    display: flex;
    align-items: center;
    gap: 10px;
```

```
    img {
        height: 24px;
        cursor: pointer;
    }
```

```
    button {
        border: none;
```

```
padding: 8px 16px;
color: #fff;
background-color: #1296ff;
cursor: pointer;
    }
  }
}
}
}
```

## YKFileShare

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { AuthContextProvider } from './context/AuthContext';
import { ChatContextProvider } from './context/ChatContext';

//Used to "root" the app. The root.render function is used to render
the app element inside the 'root' element.
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <AuthContextProvider>
    <ChatContextProvider>
      <React.StrictMode>
        <App />
      </React.StrictMode>
    </ChatContextProvider>
  </AuthContextProvider>
);
```

```
import Home from "../pages/Home";
import Login from "../pages/Login";
import Register from "../pages/Register";
import "../style.scss"
import { BrowserRouter, Routes, Route, Navigate } from
"react-router-dom";
import { useContext } from "react";
import { AuthContext } from "../context/AuthContext";

function App() {

  const {currentUser} = useContext(AuthContext)

  //Checks if the user is authenticated, and if not, it navigates them
  to the login page.
  const ProtectedRoute = ({children}) =>{
    if(!currentUser){
      return <Navigate to="/login" />;
    }
    return children
  };

  return (
    <BrowserRouter>
      <Routes>
        <Route path="/">
          <Route index element={
            <ProtectedRoute>
              <Home />
            </ProtectedRoute>
          } />
          <Route path="login" element={<Login />} />
          <Route path="register" element={<Register />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
};
```

## YKFileShare

```
}

export default App;

import React from 'react'
import Navbar from '../components/Navbar'
import Search from '../components/Search'
import Chats from '../components/Chats'

const Sidebar = () => {
  return (
    <div className='sidebar'>
      <Navbar />
      <Search />
      <Chats/>
    </div>
  );
};

export default Sidebar
```

```
import React, { useContext, useEffect, useState } from 'react';
import Messages from '../Messages';
import Input from '../Input';
import { ChatContext } from '../context/ChatContext';
import io from 'socket.io-client';
import { AuthContext } from '../context/AuthContext';

// Create a socket instance
const socket = io('http://localhost:3001', {
  transports: ['websocket'],
});

console.log('Socket connected:', socket.connected);

// Listen for socket connection errors
{/*socket.on('connect_error', (error) => {
  console.log('Socket connection error:', error);
}};*/}

const Chat = () => {
  const { data } = useContext(ChatContext); // Get chat data from
  ChatContext
  const {currentUser} = useContext(AuthContext); // Get the current
  user from Firebase Auth

  useEffect(() => {
    // Listen for 'message' event from the socket
    socket.on('message', () => {
      console.log('The report was successfully received. If the user
  did upload a malicious file, he will be kicked out of the site!');
    });

    return () => {
      // Clean up the socket event listener when the component unmounts
      socket.off('message');
    };
  }, []);

  // Function to send a message via the socket
  const sendMessage = () => {
```

```
const report = currentUser.displayName;
const reported = data.user?.displayName;
const message = `${report} is reporting: ${reported}`;
socket.emit('message', message);
};

return (
  <div className='chat'>
    <div className='chatInfo'>
      <span>{data.user?.displayName}</span>
      <div className='chatIcons'>{/* Placeholder for chat icons
*/}</div>
    </div>
    <Messages />
    <Input />
    <button onClick={sendMessage}>Have you been sent a
Malicious/suspicious file? Click here to report!</button>
  </div>
);
};

export default Chat;
```



```
import { doc, onSnapshot } from "firebase/firestore";
import React, { useContext, useEffect, useState } from "react";
import { AuthContext } from "../context/AuthContext";
import { ChatContext } from "../context/ChatContext";
import { db } from "../firebase";

const Chats = () => {
  const [chats, setChats] = useState([]); /*state variable for storing chats*/

  const { currentUser } = useContext(AuthContext); // get current user from AuthContext
  const { dispatch } = useContext(ChatContext);

  useEffect(() => {
    const getChats = () => {
      const unsub = onSnapshot(doc(db, "userChats", currentUser.uid), (doc) => {
        {
          setChats(doc.data()); // set chats state variable with the data from the document
        }
      });

      return () => {
        unsub();
      };
    };

    currentUser.uid && getChats(); /*get chats if currentUser exists */
  }, [currentUser.uid]);

  const handleSelect = (u) => {
    dispatch({ type: "CHANGE_USER", payload: u }); /*dispatch action to change user*/
  };

  return (
    <div className="chats">
```

## YKFileShare

```
      {Object.entries(chats)?.sort((a,b)=>b[1].date -
a[1].date).map((chat) => (
        <div
          className="userChat"
          key={chat[0]}
          onClick={() => handleSelect(chat[1].userInfo)} // handle
click on chat to change user

        >
        <img src={chat[1].userInfo.photoURL} alt="" />
        <div className="userChatInfo">
          <span>{chat[1].userInfo.displayName}</span>
        </div>
      </div>
    ) ) }
  </div>
);
};

export default Chats;
```

```
import React, { useContext, useState, useEffect } from "react";
import {
  collection,
  query,
  where,
  getDocs,
  setDoc,
  doc,
  updateDoc,
  serverTimestamp,
  getDoc,
} from "firebase/firestore";
import { db } from "../firebase";
import { AuthContext } from "../context/AuthContext";
import io from "socket.io-client";

const Search = () => {
  const [username, setUsername] = useState("");
  const [user, setUser] = useState(null);
  const [err, setErr] = useState(false);
  const { currentUser } = useContext(AuthContext);

  const handleSearch = async () => {
    // Execute the Firestore query asynchronously using a separate
    thread
    await new Promise((resolve) => setTimeout(resolve, 1));
    const q = query(
      collection(db, "users"),
      where("displayName", "==", username)
    ); // Searches the users collection in the Firestore database for
    users with the display name specified in the state variable username.

    try {
      const querySnapshot = await getDocs(q);
      querySnapshot.forEach((doc) => {
        setUser(doc.data());
      });
    }
  };
};
```

```

    } catch (err) {
      setErr(true);
    }
  };

const handleKey = (e) => {
  e.code === "Enter" && handleSearch();
  // Send a message to the server
  socket.emit("message", "Hello, server!");
}; // Runs the handleSearch function when the Enter key is pressed.

const handleSelect = async () => {
  //check whether the chat (chats in firestore) exists, if not -
  create
  const combinedId =
    currentUser.uid > user.uid
      ? currentUser.uid + user.uid
      : user.uid + currentUser.uid;
  try {
    const res = await getDoc(doc(db, "chats", combinedId));

    if (!res.exists()) {
      //create a chat in chats collection
      await setDoc(doc(db, "chats", combinedId), { messages: [] });

      //create user chats
      await updateDoc(doc(db, "userChats", currentUser.uid), {
        [combinedId + ".userInfo"]: {
          uid: user.uid,
          displayName: user.displayName,
          photoURL: user.photoURL,
        },
        [combinedId + ".date"]: serverTimestamp(),
      });
    }
  }
};

```

```

    await updateDoc(doc(db, "userChats", user.uid), {
      [combinedId + ".userInfo"]: {
        uid: currentUser.uid,
        displayName: currentUser.displayName,
        photoURL: currentUser.photoURL,
      },
      [combinedId + ".date"]: serverTimestamp(),
    });
  }
} catch (err) {}

setUser(null);
setUsername("")
};

return (
  <div className="search">
    <div className="searchForm">
      <input
        type="text"
        placeholder="Search or start a new chat..."
        onKeyDown={handleKey}
        onChange={(e) => setUsername(e.target.value)}
        value={username}
      />
    </div>
    {err && <span>User not found!</span>}
    {user && (
      <div className="userChat" onClick={handleSelect}>
        <img src={user.photoURL} alt="" />
        <div className="userChatInfo">
          <span>{user.displayName}</span>
        </div>
      </div>
    )}
  </div>
);
};

```

```
export default Search;
```

## YKFileShare

```
import React, { useContext, useEffect, useRef, useState } from 'react';
import { AuthContext } from '../context/AuthContext';
import { ChatContext } from '../context/ChatContext';

const Message = ({ message }) => {
  const { currentUser } = useContext(AuthContext);
  const { data } = useContext(ChatContext);
  const ref = useRef();
  const [fileName, setFileName] = useState(message.fileName);
  useEffect(() => {
    ref.current?.scrollIntoView({ behavior: "smooth" });
  }, [message]);

  /*Get the sender's profile picture, if any, else the default one.*/
  const isCurrentUser = message.senderId === currentUser.uid;

  const renderMessageContent = () => {
    if (message.fileUrl) {
      return (
        <div>
          <p>File: {fileName}</p>
          <a href={message.fileUrl} target="_blank" rel="noopener
noreferrer">Download</a>
        </div>
      );
    }
    return null;
  };

  return (
    <div ref={ref} className={`message ${isCurrentUser ? "owner" :
""}`}>
      <div className="messageInfo">
        <img
          src={isCurrentUser ? currentUser.photoURL :
data.user.photoURL}
          alt="Profile"
        />
        <span></span>
      </div>
    </div>
  );
};
```

## YKFileShare

```
        </div>
        <div className="messageContent">
            {renderMessageContent()}
        </div>
    </div>
    );
};

export default Message;
```



## YKFileShare

```
import { doc, onSnapshot } from "firebase/firestore";
import React, { useContext, useEffect, useState } from 'react'
import { ChatContext } from '../context/ChatContext';
import { db } from '../firebase';
import Message from "../Message"

const Messages = () => { // array of files
  const [messages, setMessages] = useState([]) // Get the files for the
  current chat, if any, and set them in the state.
  const {data} = useContext(ChatContext);

  useEffect(()=>{
    const unsub = onSnapshot(doc(db, "chats", data.chatId), (doc)=>{
      doc.exists() && setMessages(doc.data().messages)
    })

    return ()=> {
      unsub()
    }
  }, [data.chatId])

  console.log(messages)

  return (
    <div className='messages'>
      {messages.map(m=> (
        <Message message={m} key={m.id}/> // Render the files list if it
is not empty.
      ))}
    </div>
  );
};

export default Messages;
```

## YKFileShare

```
import React, { useContext } from 'react'
import { signOut } from "firebase/auth"
import { auth } from '../firebase'
import { AuthContext } from '../context/AuthContext'

const Navbar = () => {
  const { currentUser } = useContext(AuthContext)
  return (
    <div className='navbar'>
      <span className="logo">YKFileShare</span>
      <div className="user">
        <img src={currentUser.photoURL} alt=""/>
        <span>{currentUser.displayName}</span> {/*displaying the user
avatar photo and name */}
        <button onClick={() => signOut(auth)}>logout</button> {/*Sign
out and redirect to login page*/}
      </div>
    </div>
  )
}

export default Navbar
```

```
import React, { useContext, useState } from "react";
import attach from "../img/attach.png";
import { AuthContext } from "../context/AuthContext";
import { ChatContext } from "../context/ChatContext";
import {
  arrayUnion,
  doc,
  serverTimestamp,
  Timestamp,
  updateDoc,
} from "firebase/firestore";
import { db, storage } from "../firebase";
import { v4 as uuid } from "uuid";
import {
  getDownloadURL,
  ref,
  uploadBytesResumable,
  getMetadata,
} from "firebase/storage";

const Input = () => {
  // Set up state for the file
  const [text, setText] = useState("");
  const [file, setFile] = useState(null);
  const [fileUrl, setFileUrl] = useState(null);

  // Get the current user and chat data from the appropriate contexts
  const { currentUser } = useContext(AuthContext);
  const { data } = useContext(ChatContext);

  // Handle sending files
  const handleSend = async () => {
    // Check if there is no files before sending
    if (!file) return;

    try {
      const storageRef = ref(storage, uuid());
      const uploadTask = uploadBytesResumable(storageRef, file);
```

```

uploadTask.on(
  "state_changed",
  null,
  (error) => {
    console.error(error);
  },
  async () => {
    try {
      const downloadURL = await
getDownloadURL(uploadTask.snapshot.ref);
      const metadata = await
getMetadata(uploadTask.snapshot.ref);
      const fileName = file.name;
      const fileSize = metadata.size;

      await updateDoc(doc(db, "chats", data.chatId), {
        messages: arrayUnion({
          id: uuid(),
          text,
          senderId: currentUser.uid,
          date: Timestamp.now(),
          fileUrl: downloadURL,
          fileName,
          fileSize,
        }),
      });

      // Update the last file and date in the userChats
document for both the current user and the chat partner
      await updateDoc(doc(db, "userChats", currentUser.uid), {
        [data.chatId + ".lastMessage"]: {
          text,
          fileUrl: downloadURL,
          fileName,
          fileSize,
        },
        [data.chatId + ".date"]: serverTimestamp(),
      });

```

```

        await updateDoc(doc(db, "userChats", data.user.uid), {
          [data.chatId + ".lastMessage"]: {
            text,
            fileUrl: downloadURL,
            fileName,
            fileSize,
          },
          [data.chatId + ".date"]: serverTimestamp(),
        });
        // Clear the input fields after sending
        setText("");
        setFile(null);
        setFileUrl(null);
      } catch (error) {
        console.error(error);
      }
    }
  );
} catch (error) {
  console.error(error);
}
};

const handleFileChange = (e) => {
  const selectedFile = e.target.files[0];
  if (selectedFile) {
    setFile(selectedFile);
    setFileUrl(URL.createObjectURL(selectedFile));
  }
};

return (
  <div className="input">
    <div className="send">
      <input
        type="file"
        style={{ display: "none" }}
        id="file"
        onChange={handleFileChange}

```

```
    />
    <label htmlFor="file">
      <img src={attach} alt="" /> { /* The "attach file" button */ }
    </label>
    {fileUrl && (
      <span>
        {file.name} ({file.size} bytes)
      </span>
    )}
    <button onClick={handleSend}>Send</button>
  </div>
</div>
);
};

export default Input;
```

```
import React, { useState } from 'react';
import { useNavigate, Link } from 'react-router-dom';
import { signInWithEmailAndPassword } from "firebase/auth";
import { auth } from '../firebase';

const Login = () => {

  const [err, setErr] = useState(false); // Used to set the error
message state to false initially,--
                                     //--and to set the error
message to a string if an error occurs during the login process.

  const navigate = useNavigate()

  /*Called when the 'login' button is clicked. It takes an event
object and extracts the email and password from the input fields.*/

  const handleSubmit = async (e)=>{
    e.preventDefault();
    const email = e.target[0].value;
    const password = e.target[1].value;

    if (!email || !password) {
      setErr({ message: 'All fields are required.' });
    } else {
      try{
        await signInWithEmailAndPassword(auth, email, password);
        navigate("/")
      } catch (err) {
        setErr({ message: err.message.slice(9) });
      }
    }
  };

  return (
    <div className='formContainer'>
      <div className='formWrapper'>
```

```
    <div className="logoContainer">
      <span className="logo">YK</span>
      <span className="subLogo">FileShare</span>
    </div>
    <h1 className="title">Login</h1>
    <form onSubmit={handleSubmit}>
      <input type="email" placeholder="Email" />
      <input type="password" placeholder="Password" />
      <button>Sign In</button>
      {err && <span className="error-message">{err.message}</span>}
    </form>
    <p className="registerLink">Don't have an account? <Link
to="/register">Register</Link></p>
  </div>
</div>
);
};

export default Login
```



```

import React, { useState } from 'react';
import Add from "../img/AvatarAdd.png"
import { createUserWithEmailAndPassword, updateProfile } from
"firebase/auth";
import { auth, db, storage } from "../firebase";
import { ref, uploadBytesResumable, getDownloadURL } from
"firebase/storage";
import { doc, setDoc } from "firebase/firestore";
import { useNavigate, Link } from 'react-router-dom';

const Register = () => {
  const [err, setErr] = useState(null);
  const navigate = useNavigate()

  const handleSubmit = async (e) => {

    // Get the input values from the sign up form.
    const displayName = e.target[0].value;
    const email = e.target[1].value;
    const password = e.target[2].value;
    const confirmPassword = e.target[3].value;
    const file = e.target[4].files[0];

    // Check if the email is valid and if it is a gmail email
    address, if not, return undefined.
    const emailRegex =
/^(((^[^<>()\\[\]\\\.,;:\s@"]+(\.[^<>()\\[\]\\\.,;:\s@"]+)*|(".+"))@((\[0-9
]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\)|((([a-zA-Z\-0-9]+\.)+[a-zA
-Z]{2,})))$)/;

    const isValidEmail = emailRegex.test(email);
    const isValidDomain = /@gmail\.com$/i.test(email);
    let errorMessage = '';

    if (!displayName || !email || !password || !confirmPassword) {
      errorMessage = 'All fields are required.';
    } else if (!isValidEmail) {
      errorMessage = 'must be valid Email format';
    }
  }

```

```

    } else if (!isValidDomain){
      errorMessage = 'Email must end with "@gmail.com"';
    } else if (password !== confirmPassword){
      errorMessage = 'Passwords do not match';
    }
    if (errorMessage !== '') {
      setErr({ message: errorMessage });
      return; // Exit the function if there is an error.
    }
    try {
      // Create a storage reference for the user
      const res = await createUserWithEmailAndPassword(auth, email,
password);

      const displayNameRef = ref(storage, displayName);

      // Check if a file was selected
      if (file) {
        const uploadTask = uploadBytesResumable(displayNameRef,
file);

        uploadTask.on(
          (error) => {
            setErr({ message: error.message });
          },
          async () => {
            try {
              // Get the download URL of the uploaded file
              const downloadURL = await
getDownloadURL(uploadTask.snapshot.ref);

              // Update the user's name and photo URL in the
database.

              await updateProfile(res.user, {
                displayName,
                photoURL: downloadURL,
              });
              await setDoc(doc(db, "users", res.user.uid), {
                uid: res.user.uid,

```

```

        displayName,
        email,
        photoURL: downloadURL,
    });
    // Navigate to the home page after the user has been
added to "userChats".
    await setDoc(doc(db, "userChats", res.user.uid), {});

    navigate("/");
  } catch (error) {
  }
}
);
} else {
  try {

    // If no file was selected, update the user's name without
a photoURL
    await updateProfile(res.user, {
      displayName,
    });
    await setDoc(doc(db, "users", res.user.uid), {
      uid: res.user.uid,
      displayName,
      email,
    });
    await setDoc(doc(db, "userChats", res.user.uid), {});
    navigate("/");
  } catch (error) {
  }
}
} catch (err) {
  if (err.code === "auth/email-already-in-use") {
    setErr({ message: "Email is already in use" });
  } else {
    setErr({ message: err.message.slice(9) });
  }
}
};

```

```

return (
  <div className='formContainer'>
    <div className='formWrapper'>
      <div className="logoContainer">
        <span className="logo">YK</span>
        <span className="subLogo">FileShare</span>
      </div>
      <span className="title">Register</span>
      <form onSubmit={handleSubmit}>
        <input type="text" placeholder="username"/>
        <input type="email" placeholder="email"/>
        <input type="password" placeholder="password"/>
        <input type="password" placeholder="confirm
password"/>
        <input style={{display:"none"}} type="file"
id="file"/>
        <label htmlFor="file" >
          <span style={{ color: "#fff" }}>Upload Profile
Picture -</span>
          <img style={{ width: 32, cursor: 'pointer'}}
src={Add} alt="" />
        </label>
        <button>Sign Up</button>
        {err && <span>{err.message}</span>}
      </form>
      <p>You do have an account? <Link
to="/login">Login</Link></p>
    </div>
  </div>
);
};

export default Register;

```

## YKFileShare

```
import React from 'react'
import Chat from '../components/Chat'
import Sidebar from '../components/Sidebar'

const Home = () => {
  return (
    <div className='home'>
      <div className="container">
        <Sidebar/>
        <Chat/>
      </div>
    </div>
  )
}

export default Home
```