עבודת חקר במדעי המחשב

פרטי המגיש:

- שם: יואב קילשטיין
- תעודת זהות: 328234034
- בית ספר: תיכון הכפר הירוק ע"ש לוי אשכול
- תאריך לידה: 11.10.2005
- כתובת: אליהו מפרארה 9
- מספר טלפון: 0549019103
- דוא"ל: yoavkil123@gmail.com

פרטי המנחה:

- שם: יהודה אור
- תעודת זהות: 32098007
- השכלה: מהנדס MA. מוסמך מטעם הטכניון בהנדסה ועבר הסבה אקדמית למחשבים. תעודת הסמכה מטעם Microsoft.
- מקום עבודה: תיכון הכפר הירוק, GSA, WAN TEC.
- תחום עבודה: מדעי המחשב
- מספר טלפון: 0507344457
- דוא"ל: Yoooda@gmail.com
- כתובת: הכרם 3, תל אביב

בית הספר:

- שם: הכפר הירוק ע"ש לוי אשכול
- סמל מוסד: 580019
- כתובת: הכפר הירוק 47800, רמת השרון
- טלפון: 036455621

העבודה:

- נושא העבודה: בניית סביבה המאפשרת אינטראקציה נוחה בטוחה בין אנשים.
- היקף העבודה: 5 יחידות לימוד
- תאריך הגשה: 28.2.2023

תוכן העניינים

חלק ראשון: מבוא

מבוא:

כמפתח צעיר עם תשוקה למדעי המחשב, אני חיפשתי ומחפש כל הזמן אתגרים והזדמנויות חדשות להרחיב את הידע והכישורים שלי. הזדמנות אחת כזו הגיעה בצורה של עבודת החקר. הפרויקט שלי, בניית אפליקציית צ'אט מבוססת ווב המאפשרת למשתמשים לשלוח הודעות בצורה מאובטחת ויעילה לא היה רק תרגיל קידוד פשוט עבורי, אלא הזדמנות לחקור את התחום המרגש והמורכב של קריפטוגרפיה, רשתות מחשבים, אבטחת מידע ותקשורת נתונים.

אחת הסיבות שנמשכתי לפרויקט הזה הייתה אהבתי לפתרון בעיות וללמוד על מדעי המחשב. ראיתי בפרויקט הזה הזדמנות להעמיד את הכישורים והידע שלי במבחן ולהעמיק בנבכי התחום. ידעתי שהפרויקט הזה יספק לי שפע של ניסיון מעשי והזדמנויות למידה מעשית.

המטרה הסופית של פרויקט זה הייתה ליצור אתר אינטרנט מוצפן, יעיל וידידותי למשתמשים.
התמסרתי להבטחת הפרטיות והסודיות של נתוני המשתמשים, וזו הסיבה שגיליתי עניין רב בבחינת טכניקות שונות, פרוטוקולי רשתות וכלי שמירת נתונים אמינים.

בתיק פרויקט זה, אלווה אתכם במסע האישי שלי בתהליך העיצוב והפיתוח של אפליקציית הצ'אט תחת שאלת המחקר והמטרה הכללית ההתחלתית שלי.
מקווה שאצליח ללמד אתכם משהו חדש ותהנו מהקריאה!

צעדים משוערים לפיתוח הפרויקט:

- פיתוח אתר צאט תאפשר תקשורת ואינטראקציה מהירה בין אנשים בעלי תחומי עניין דומים.
- יצירת סביבה בטוחה עם אימות מאובטח והצפנת נתונים תעודד יותר אנשים להשתתף ולהצטרף לפלטפורמה תוך תחושה ששומרים על הפרטיות שלהם.
- אירוח הפרויקט בשירות אחסון אתרים אמין וחזק יבטיח שהפלטפורמה תהיה נגישה לקהל רחב יותר ותוכל להתמודד עם תנועה מוגברת של משתמשים.
- שימוש בפלטפורמה רבת שירותים לייצרת אפליקציות כמו Firebase, שתאפשר שילוב חלק של תכונות כגון אימות משתמשים ועדכוני מסד נתונים בזמן אמת, מה שיוביל לתהליך פיתוח יעיל ויעיל יותר.

אלו הם מספר צעדים והשערות שהנחתי בתחילת הפרויקט שיהיו הכרחיים (מלבד כתיבת הקוד) למען פתרון הבעיה שהוצגה בשאלת המחקר.

חלק שני: הפרויקט

הפרויקט:

הפרויקט Trust Yoav הוא אפליקציית צאט מבוסס WEB ופלטפורמה מאובטחת ויעילה לתקשורת בזמן אמת. האתר מספק למשתמשים ממשק מהיר וידידותי, המאפשר לשלוח הודעות ותמונות בקלות.
הכלים החיצוניים בהם השתמשתי לבניית האתר ואפרט עליהם בהמשך הם Hostinger ו-FireBase.

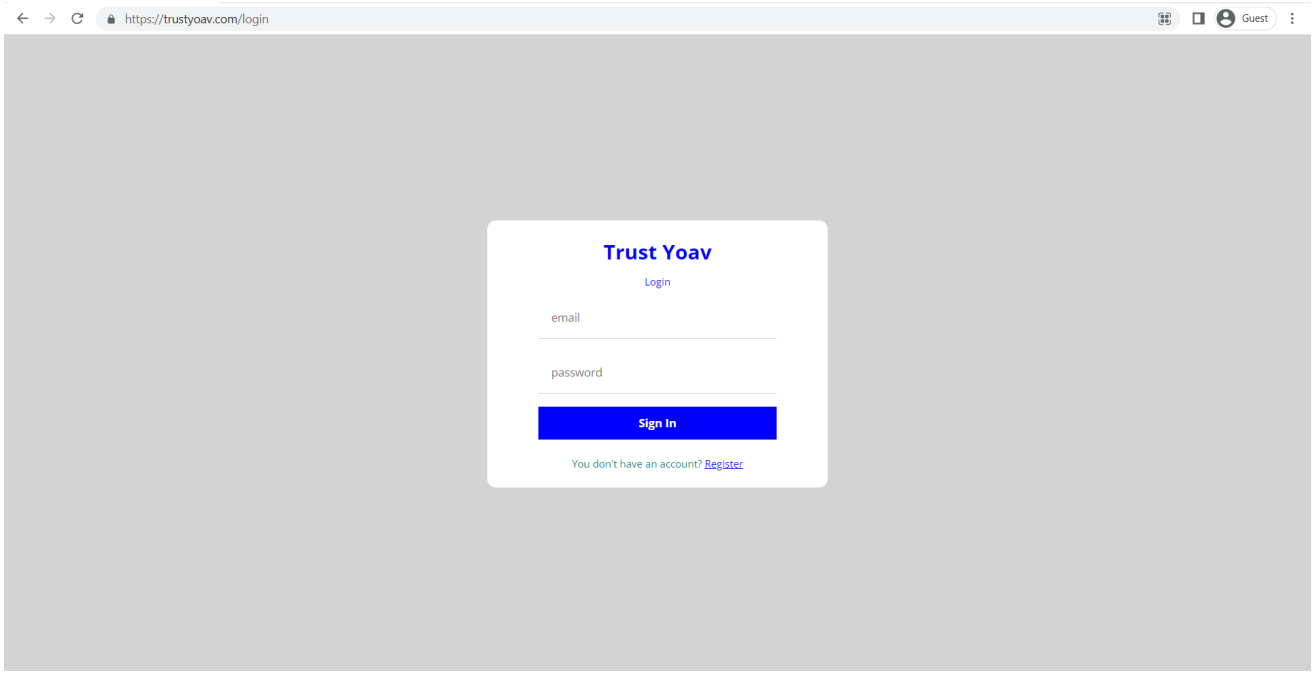
הוסטינגר סיפקה את הפתרון האמין לאחסון האתר, בעוד ש-Firebase את יכולות אחסון הנתונים והתקשורת בזמן אמת שהן חיוניות עבור אפליקציית צ'אט. השילוב של שני הכלים החזקים הללו אפשר יצירת אפליקציה מהירה, יעילה ומאובטחת המספקת למשתמשים חווית תקשורת חלקה. על ידי מינוף החוזקות של Hostinger ושל Firebase, הצלחתי להשיג את רמת הביצועים והאמינות הגבוהה שהייתי צריך והתחייבתי לעצמי שתהיה בתחילת העבודה.

הפרויקט נכתב בשפת התכנות Reactjs, הידועה בזכות הרב-גוניות והפשטות שלה. שפה זאת הייתה הבחירה המושלמת עבור הפרויקט הזה בגלל היכולת שלה לנהל ולעדכן בקלות את ממשק המשתמש, מה שהופך אותה להתאמה אידיאלית עבור אפליקציית צ'אט בזמן אמת.

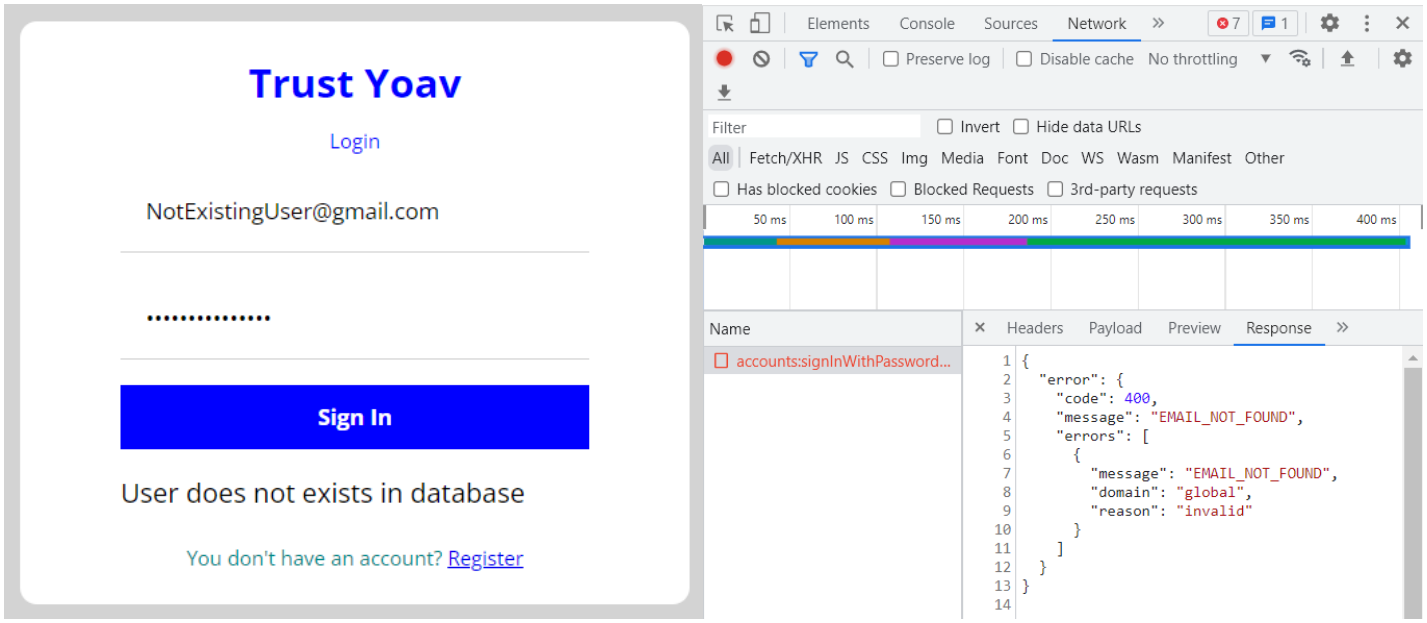
הפרויקט פותח בסביבת העבודה של Visual Studio Code, עורך קוד פופולרי ועשיר בתכונות שסיפק סביבת פיתוח מקיפה. סביבת עבודה זאת, אפשרה ניהול פשוט וקל של הפרויקט, והקל על מעקב אחר הרכיבים והמודולים השונים המעורבים בבניית הפרויקט.
פירוט עמודים והסבר זה יספק הבנה כללית וטובה של הפרויקט אך הדקויות על איך נעשו הדברים מאחורי הקלעים אפרט בחלק המעשי.

1. עמוד הכניסה:

זה העמוד אליו מגיעים כאשר מזינים באינטרנט את כתובת האתר. הדבר הראשון שעולה כמו שניתן לראות הוא שהאתר תחת פרוטוקול HTTPS שגם על כך אפרט בהמשך.
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 אכן קיים משתמש עם נתונים אלו באתר - הכניסה תתאפשר, אחרת תקבל הודעת שגיאה.
לאחר לחיצה על כפתור הSign In הנתונים מהלקוח מועברים לשרת אשר מאמת את פרטי המשתמש עם מסד הנתונים הקיים ב-FireBase.

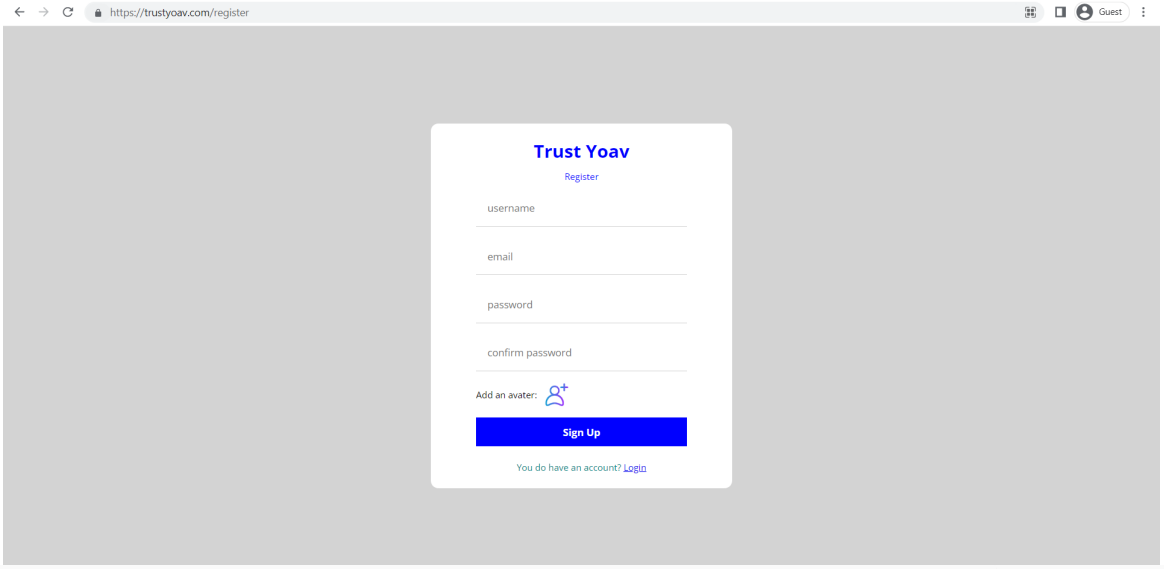


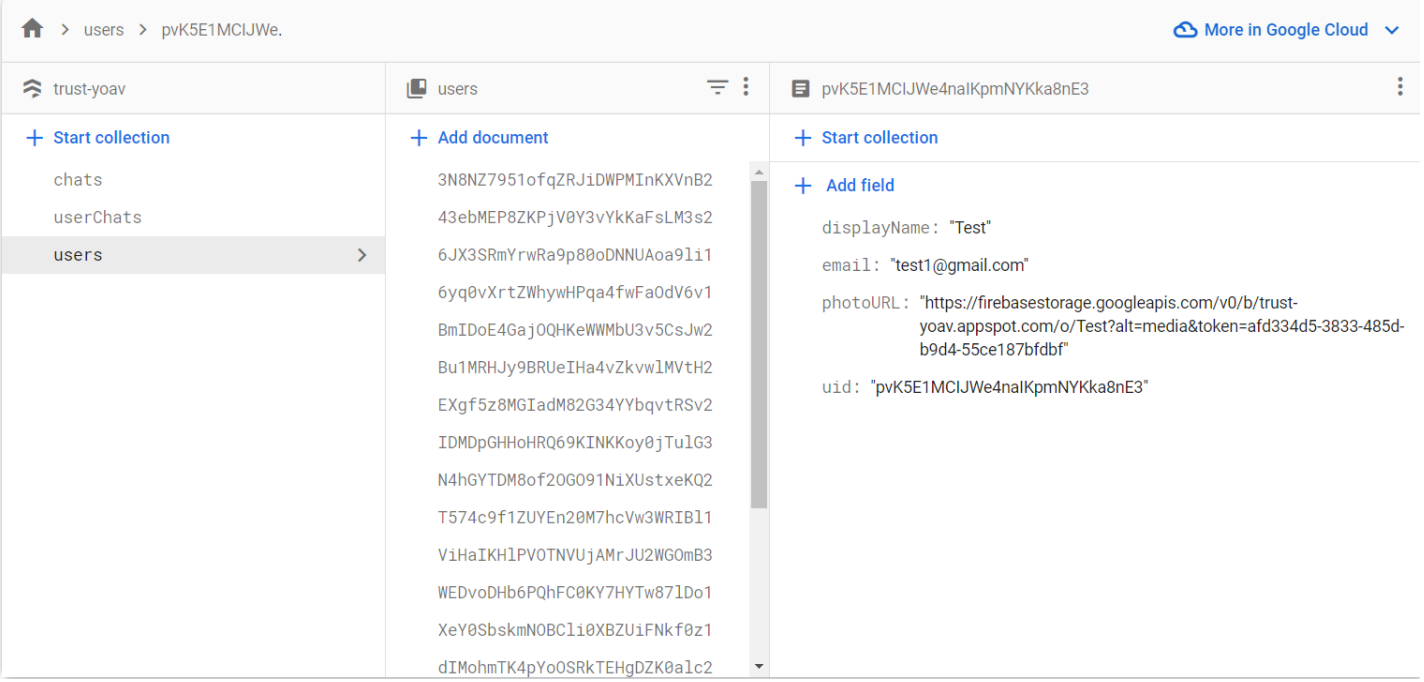
הודעת שגיאה:



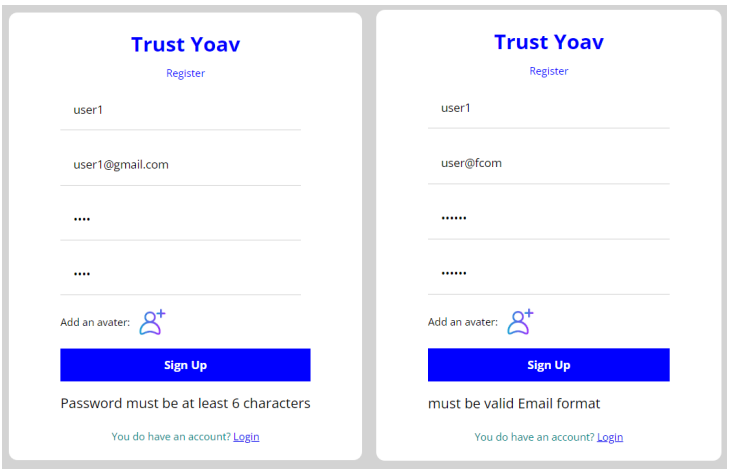
2. עמוד ההרשמה:

אל עמוד ההרשמה ניתן לגשת דרך גלישה לכתובת העמוד (https://trustyoav.com/register) 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 ישמר המשתמש החדש במסד הנתונים של האתר בFireBase באוסף המשתמשים והוא יקבל גישה לאתר.

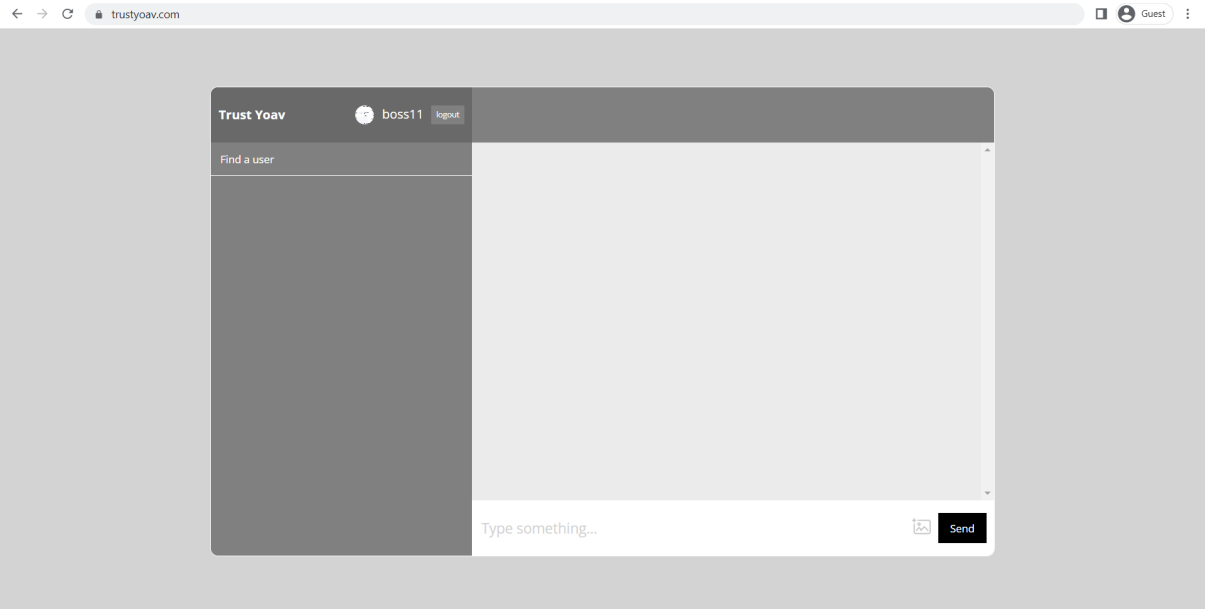




הודעות שגיאה:



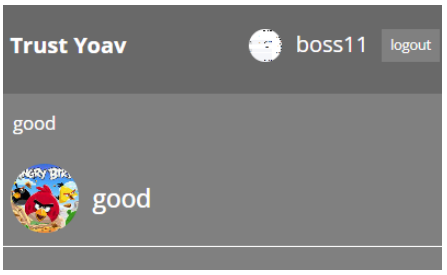
3. עמוד הבית/הצאט:

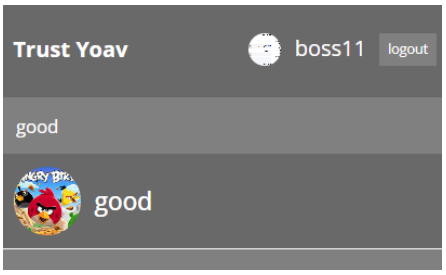


לאחר שנרשמים/מתחברים למשתמש, מגיעים לעמוד המרכזי באתר. כפי שניתן לראות, הוא תומן בתוכו מרכיבים רבים.

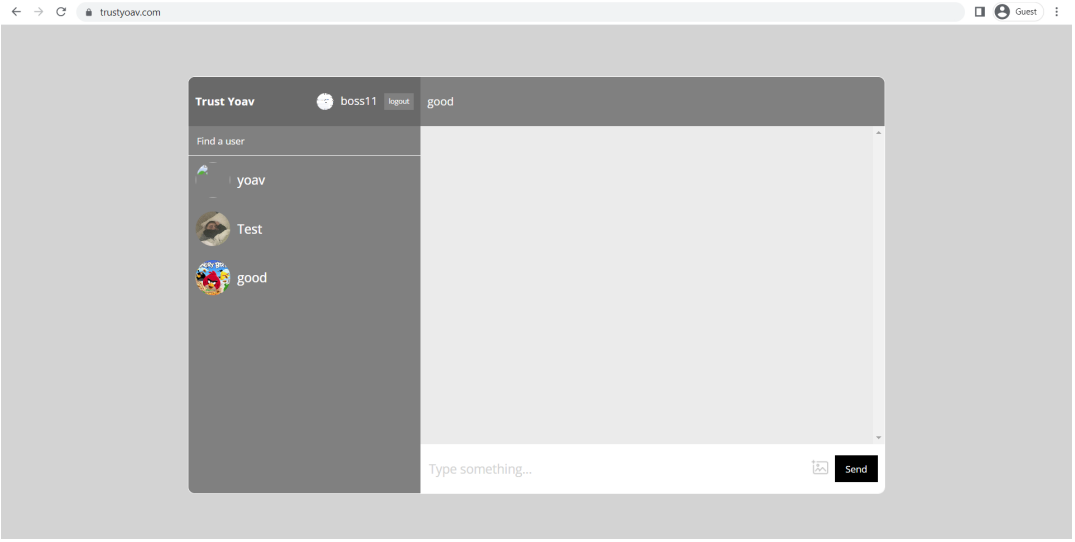
- בחלק העליון של העמוד, מוצג עבור המשתמש - שמו, תמונתו, כפתור התנתקות ושם האתר.
- בצד שמאל מופיע סרגל הניווט בין האנשים השונים שהמשתמש בשיחה איתם (יוצג בהמשך) ואפשרות לחפש משתמש לשיחה.

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





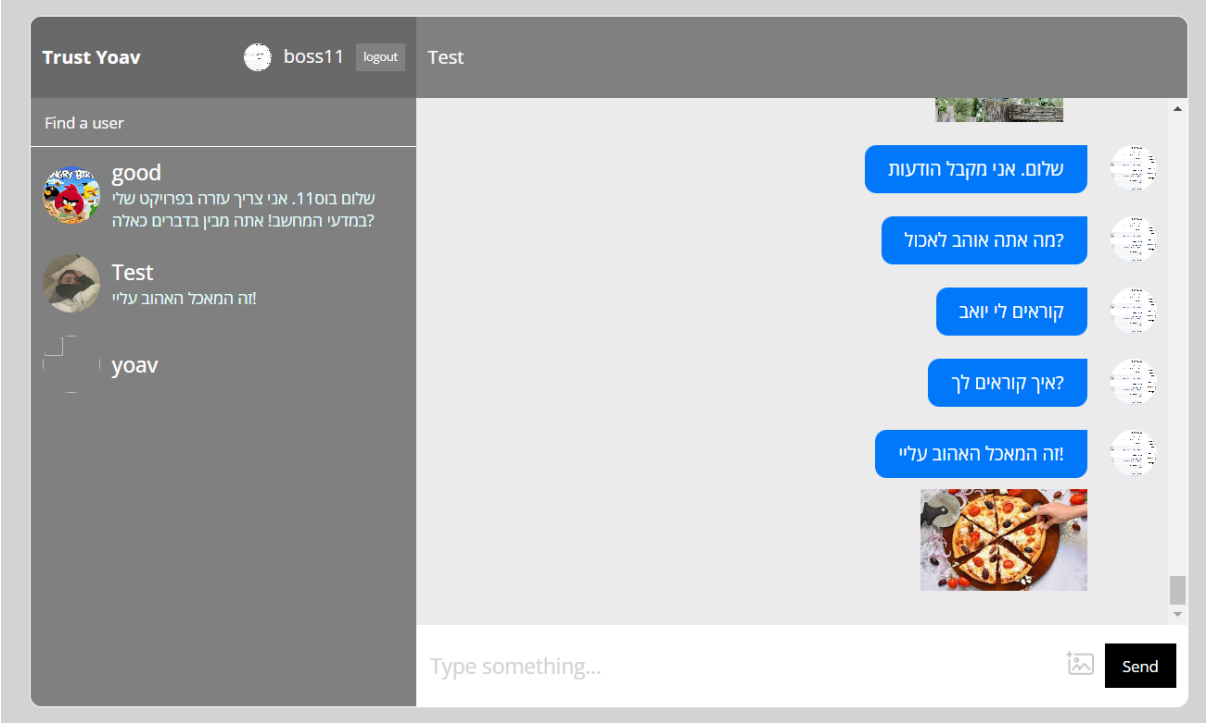
לאחר שלחצנו על המשתמש (ואחרים נוספים) הם יופיעו בסרגל הניווט שלנו, יתווספו לאוסף הצאטים השמורים במסד הנתונים ונוכל להיכנס לצאט איתם ולכתוב למשתמש מתי שנרצה. הנה המסך שאנו נמצאים בו לאחר פעולה זאת-



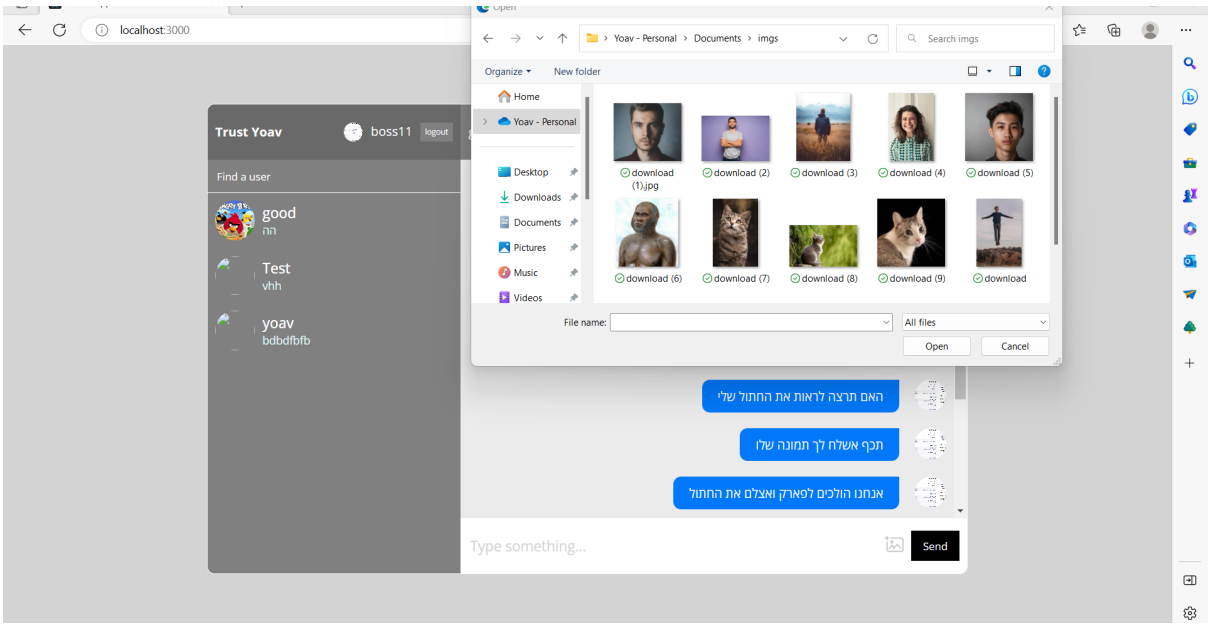
- **הצאט עצמו:**

בחלק של הצאט באתר מופיע שם המשתמש של מי שאנו בשיחה איתו כעת וכותבים ישירות אליו. בתבנית הType something... ניתן להקליד את הודעת הטקסט המבוקשת, ובלחיצה על אייקון התמונה ניתן לשלוח תמונות.

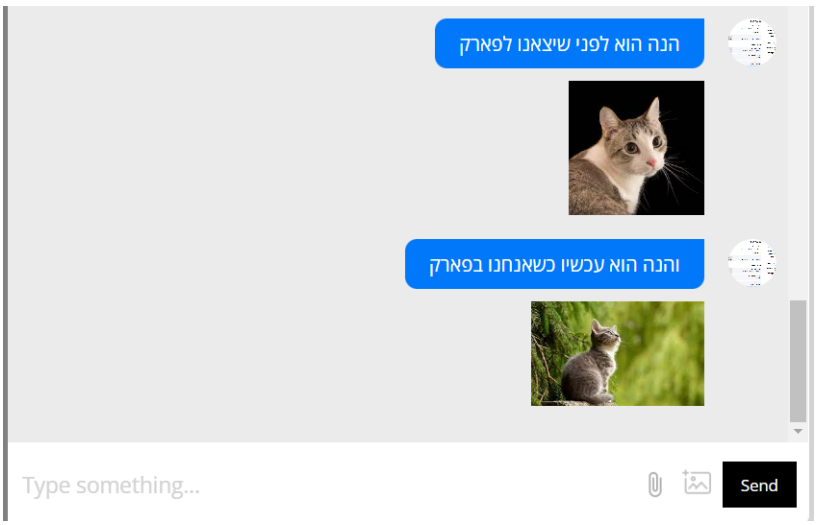
לאחר שהוספנו את מה שאנו מעוניינים לשלוח למשתמש, נלחץ על SEND. ההודעה תעבור ותשמר תחת אוסף ה"chats" במסד הנתונים ודרך כך תופיע למשתמשים.



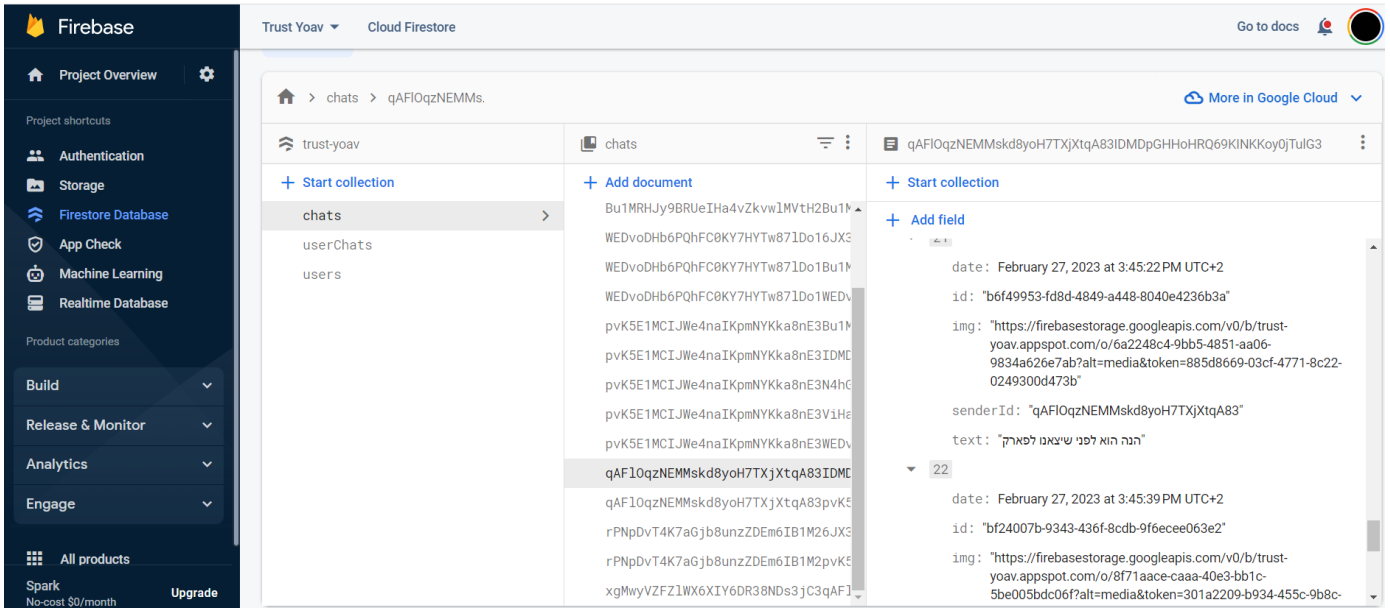
בשיחה לדוגמה אחרת, מוצג מה קורה לאחר שboss11 היה מעוניין לשלוח תמונה של החתול שלו לgood:



כפי שניתן לראות, boss11 לחץ על אייקון התמונה ונפתח לו תיקיית התמונות שלו- אשר מתוכן הוא בחר לשלוח את התמונה של החתול כפי שנראה כעת:

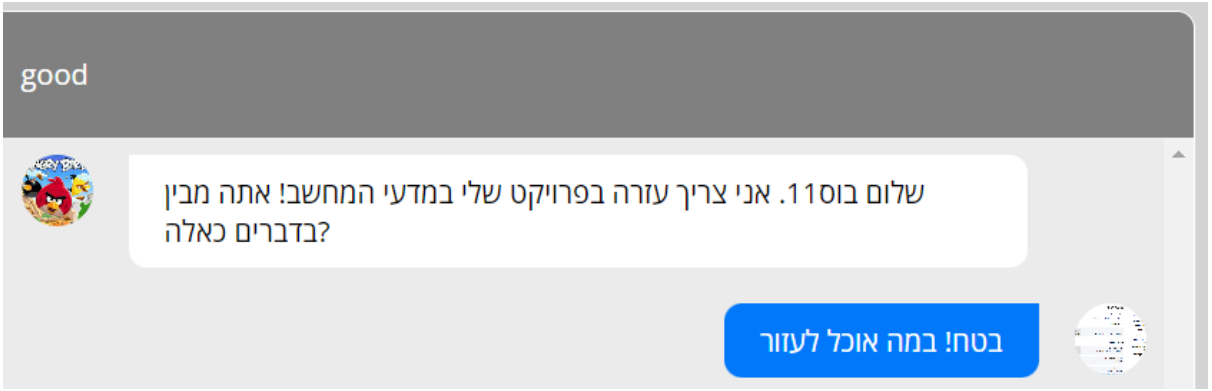


כך התמונות נשמרות במסד הנתונים באותו הרגע. נשמרות בצירוף עם ההודעה שנשלחה איתן.



כפי שניתן לראות ההודעות נשלחות ומוצגות בכחול וההודעה האחרונה שנשלחה מוצגת במקום הראשון בסרגל הניווט - לאחר ששלחנו את ההודעות ל"test" קיבלנו הודעה חדשה מ"good" (ראה תמונה ראשונה תחת "הצאט עצמו").

ניתן לענות להודעות בפשטות בהקלדת ההודעה ושליחתה חזרה עם כפתור הsend-



בזאת מסתכמת החוויה והאפשרויות הכלליות של המשתמש, על החיבורים עם השרת ועל הדברים שהצגתי אתעמק בהמשך.

חלק שלישי: רקע עיוני

ReactJS[1]

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.[2]

השפה מבוססת על מודל תכנות הצהרתי, המאפשר למפתחים לתאר מה הם רוצים שיקרה בקוד שלהם, מבלי לציין בהכרח איך זה צריך לקרות. מודל זה מקל על חשיבה לגבי קוד וניפוי באגים, ומקל על מפתחים לעבוד על פרויקטים גדולים עם משתפי פעולה רבים.

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.

React.js ידועה גם בתמיכה שלה ברכיבים הניתנים לשימוש חוזר, שיכולים לחסוך למפתחים כמויות משמעותיות של זמן ומאמץ. מפתחים יכולים ליצור רכיבים לשימוש חוזר שניתן להשתמש בהם בכל האפליקציות שלהם, מה שמפחית את כמות הקוד שצריך לכתוב ומקל על תחזוקת הקוד לאורך זמן.

בנוסף לתכונות הליבה הללו, React.js מספקת מגוון כלים וספריות המקלים על הבנייה והתחזוקה של ממשקי משתמש מורכבים. כלים אלו כוללים את React Router לניהול ניווט בין דפים, Redux לניהול מצב האפליקציה ו-React Native לבניית אפליקציות מובייל.

---

[1] *LEARN REACT*. React. (n.d.). Retrieved 2023, from https://react.dev/learn

[2] https://media.geeksforgeeks.org/wp-content/uploads/20210908120846/DOM.png *DOM*. (n.d.). Geeksforgeeks.

סביבת העבודה - Visual studio code

Visual studio code הוא עורך קוד פתוח וחינמי, שזכה לפופולריות נרחבת בקרב מפתחים הודות למגוון התכונות וקלות השימוש שלו. הפלטפורמה פותחה על ידי מיקרוסופט ותוכננה להתאמה אישית עם מגוון רחב של תוספים והרחבות שבהם ניתן להשתמש.

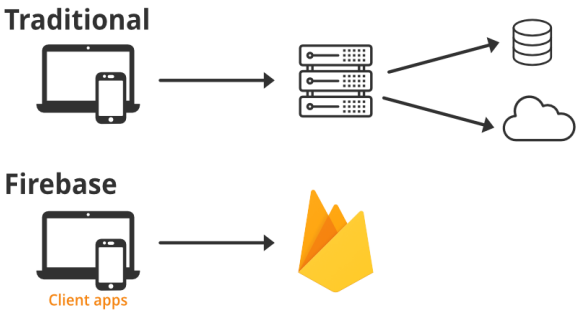
אחד היתרונות המרכזיים שלו הוא קלות השימוש. הוא מחזיק בממשק פשוט ואינטואיטיבי, המקל על מפתחים להתחיל ולעבוד על פרויקטים במהירות. העורך כולל תכונות רבות שעוזרות להאיץ את הפיתוח, כגון השלמת קוד חכמה, הדגשת תחביר וכלי ניפוי באגים.

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.

Firebase

Firebase היא פלטפורמה ליצירת אפליקציות למובייל ולאינטרנט, המספקת מגוון כלים ושירותים לבנייתן באיכות גבוהה. Firebase מציעה תכונות שונות כגון אימות, אחסון בענן, מסדי נתונים בזמן אמת, אירוח והעברת הודעות.[3]



Firebase 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 Firebase אחרים, כגון מסד נתונים בזמן אמת, אחסון בענן ואירוח, ומאפשר למפתחים ליצור חוויות משתמש מותאמות אישית.

- אחסון הענן - מספק למפתחים דרך מאובטחת, ניתנת להרחבה וחסכונית לאחסן ולשמור תוכן שהגיע על ידי משתמשים, כגון תמונות, סרטונים וקבצים אחרים (ראה מתוך החלק השני של העבודה). ניתן להשתמש באחסון הענן עם השירותים האחרים של Firebase, כגון אירוח ומסד נתונים בזמן אמת, כדי לבנות את צד השרת של אפליקציה.

- שירות האירוח - מאפשר למפתחים לפרוס ולארח את יישומי האינטרנט שלהם בתשתית של גוגל, מה שמקל על ההשקה וההרחבה של האפליקציות שלהם. שירות האירוח משולב עם שירותי Firebase אחרים, כגון אימות ומסד נתונים בזמן אמת, המאפשר למפתחים ליצור חוויות משתמש מותאמות אישית ודינמיות.

- שירות ההודעות - מאפשר למפתחים לשלוח ולקבל הודעות בין לקוחות, מה שמקל על בניית אפליקציות צ'אט והתראות בזמן אמת. שירות ההודעות מותאם ליעילות הסוללה ופועל גם כשהמכשיר במצב לא מקוון.

[3]Educative. (n.d.). What is Firebase? Educative: Interactive Courses for Software Developers. https://www.educative.io/answers/what-is-firebase

HOSTINGER

Hostinger היא חברת אחסון אתרים המספקת מגוון שירותי אחסון, כולל אחסון משותף, אירוח VPS, אחסון בענן ועוד. תוכניות האחסון המשותפות של Hostinger מיועדות לאנשים פרטיים ועסקים קטנים.
התוכניות מגיעות עם מגוון אפשרויות אבטחה, כולל תעודת SSL בחינם, גיבויים יומיים והגנת DDoS. תכונות אלו עוזרות לשמור על אבטחת האתר שלך ולהגן עליו מפני איומים שונים ברשת (ראה פירוט הצפנות).

רשתות

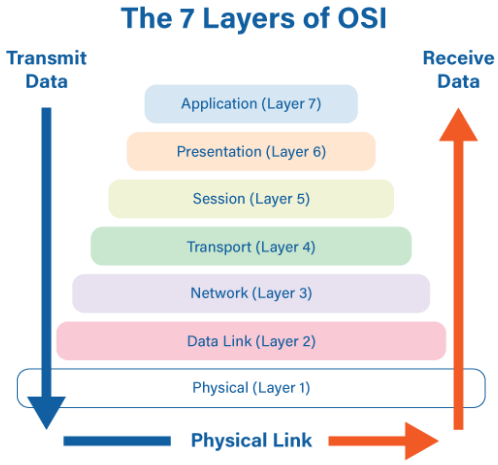
www -

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.[4]

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.

[4] *Untitled*. (2019, February 5). המרכז לחינוך סייבר. Retrieved February 19, 2023, from https://data.cyber.org.il/networks/networks.pdf



מודל הOSI-[5]

מודל OSI הוא מודל למערכות תקשורת המתאר מבנה של חמש (ניתן לתאר אותו גם עם שבע) שכבות אשר בהן עוברים בעת העברת נתונים ברשת. כל שכבה אחראית לסט ספציפי של פונקציות ושיטות, המאורגנות בצורה היררכית כדי לאפשר יישום והבנה קלים.

המודל, מהווה מסגרת שימושית לתכנון, הטמעה ופתרון בעיות של פרוטוקולי רשת, מאפשר הפרדה בין שכבות שונות ומקדם סטנדרטיזציה, מה שמקל על התקנים שונים לתקשר זה עם זה.

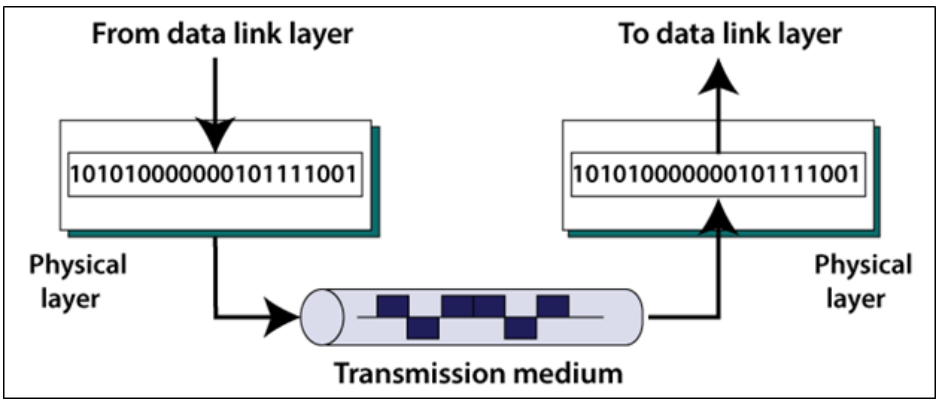
חמש שכבות הדגם, מהנמוכה לגבוהה, הן כדלקמן:

השכבה הפיזית:

השכבה הפיזית היא השכבה הראשונה של מודל הOSI. השכבה הפיזית אחראית על העברת ביטים גולמיים על פני ערוץ תקשורת, כגון חוט, כבל סיבים אופטיים או מדיום אלחוטי. תפקידו העיקרי הוא להמיר נתונים דיגיטליים לפורמט שניתן לשדר דרך ערוץ התקשורת על ידי קידודם לאותות חשמליים, אופטיים או רדיו.

השכבה מגדירה את המאפיינים של ערוץ התקשורת, לרבות המאפיינים הפיזיים והחשמליים של המדיה, סוגי המחברים והממשקים המשמשים לחיבור התקנים לרשת, ושיטות האיתות המשמשות להעברת נתונים על הערוץ. זה גם מציין את קצב הנתונים, או את כמות הנתונים המקסימלית שניתן להעביר על הערוץ בפרק זמן נתון, ואת רוחב הפס, או את טווח התדרים המשמשים להעברת האותות.

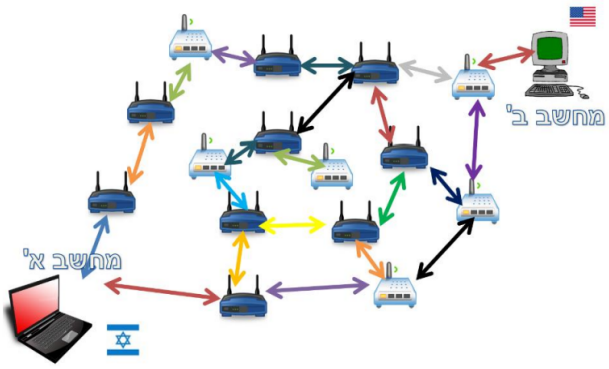
בנוסף, היא אחראית גם על איתור ותיקון שגיאות המתרחשות במהלך השידור, כגון רעש, הפרעות או הנחתה. הוא עושה זאת על ידי הטמעת טכניקות שונות כגון קודי זיהוי שגיאות, הגברת אותות או חידוש אותות.



הTransmission medium הוא שנושא את המידע בצורה של סיביות דרך LAN (רשת מקומית).

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

שכבת הקו:

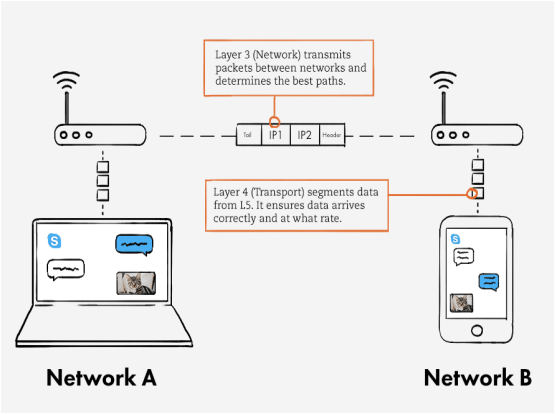


שכבת הקו היא השכבה השנייה במודל ומספקת העברה אמינה של נתונים על פני קישור או חיבור בודד בין שני צמתים סמוכים ברשת. השכבה מחלקת נתונים לפריימים ומוסיפה כותרת לכל frame. הכותרת כוללת מידע על צמתי המקור והיעד, כמו גם מידע בקרה אחר הדרוש לזיהוי ושחזור שגיאות.

בנוסף, היא מנהלת גם בקרת זרימה וגישה למדיה. בקרת זרימה מבטיחה שהעברת נתונים בקצב שהמקלט יכול להתמודד, בעוד בקרת גישה קובעת לאיזה מכשיר יש את הזכות להשתמש בערוץ התקשורת בכל זמן נתון.

שכבת הרשת:

שכבת הרשת מאפשרת תקשורת מקצה לקצה בין התקנים ברשתות שונות על ידי שימוש בכתובות לוגיות - העברת פקאטות (חבילות מידע) מישות אחת לאחרת.
שכבת הרשת משתמשת בכתובת לוגית כדי לזהות כל מכשיר ברשת, ללא קשר לכתובת או מיקומו הפיזית. סכימת הכתובות הלוגית הנפוצה ביותר בשימוש בשכבת הרשת היא כתובת פרוטוקול האינטרנט, שהיא מזהה מספרי ייחודי המוקצה לכל מכשיר ברשת IP.



בנוסף, היא אחראית גם על ניתוב נתונים בין רשתות, הכוללת בחירת הנתיב הטוב ביותר למעבר נתונים בין מכשירי המקור והיעד. זה נעשה על ידי שימוש באלגוריתמי ניתוב שלוקחים בחשבון גורמים כמו עלות כל נתיב, טופולוגיית הרשת והעומס בכל נתיב. הנתבים ברשת אחראים להעברת מנות הנתונים ליעד המתאים.

שכבת התעבורה:

שכבת התחבורה אחראית למתן שירותי תקשורת בין תהליכי יישומים הפועלים על מארחים שונים - העברת מידע בין תוכניות (תהליכים) שונים ברשת. אף על פי שתהליכים אלו אינם מחוברים פיזית, שכבת התעבורה מספקת ערוצי תקשורת לוגיים, המאפשרים חילופי מסרים ביניהם. פרוטוקולי שכבת התחבורה, כגון TCP ו-UDP, מיושמים במערכות קצה אך לא בנתבי רשת, מה שמבטיח שהתקשורת היא תהליך דו-כיווני.

אחת המטרות העיקריות של שכבת התעבורה היא לאפשר לאפשר למספר אפליקציות לתקשר עם אותו ישות באמצעות שירותים שונים, ובכך לאפשר מגוון אפשרויות תקשורת. המטרה השנייה של

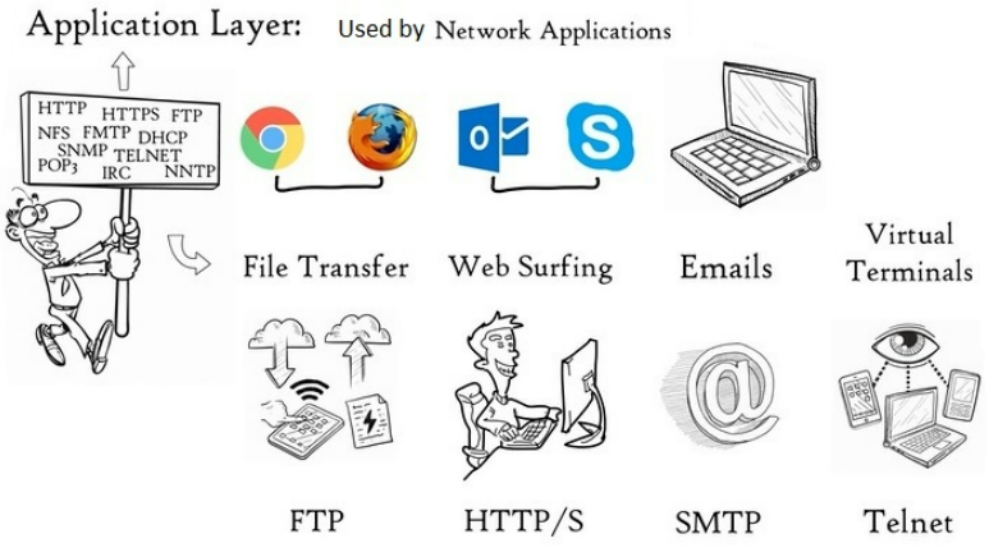
השכבה היא העברת מידע אמינה, אם כי זו מטרה אופציונלית ואינה קיימת בכל הפרוטוקולים של שכבת התעבורה.

שכבת האפליקציה/היישום:

שכבת היישום מספקת שירותים לתמיכה בתקשורת בין יישומים ברשת, והיא השכבה שמשתמשי הקצה מקיימים עמה אינטראקציה ישירה. שכבת האפליקציה אחראית לספק שירותי אפליקציה לתהליכי המשתמש ולאפשר להם לתקשר זה עם זה ברשת. הוא מציע פרוטוקולים ושירותים שונים ליישומים, כגון דואר אלקטרוני, העברת קבצים והתחברות מרחוק, ומאפשר ליישומים אלה להחליף נתונים זה עם זה.

אחת התפקידים העיקריים של שכבת האפליקציה היא לספק ממשק סטנדרטי בין האפליקציה לשכבות התחתונות של הרשת. ממשק זה מאפשר ליישומים לתקשר עם הרשת ללא צורך לדעת את הפרטים של פרוטוקול הרשת הבסיסי. זה מאפשר ליישומים להיות ברשת על פני פלטפורמות ומערכות הפעלה שונות.

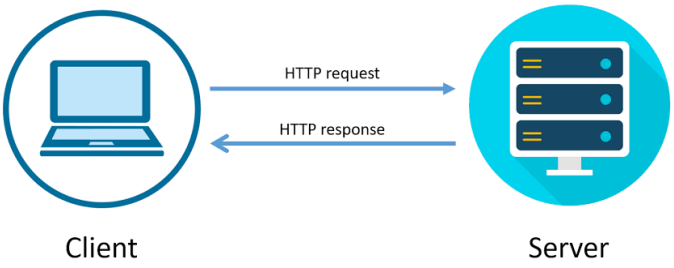
שכבת היישום יכולה גם לספק תמיכה לאימות משתמשים, הרשאות וחשבונאות. לדוגמה, כאשר משתמש נכנס למערכת מרוחקת, שכבת האפליקציה יכולה לאמת את זהות המשתמש ולהעניק גישה למשאבים על סמך ההרשאות שלו. זה יכול גם לעקוב אחר השימוש ולהפיק דוחות על שימוש במשאבים ומדדים אחרים. בנוסף, היא יכולה גם לספק תמיכה בהצפנת ופענוח נתונים, המשמשת כדי להבטיח את הסודיות והשלמות של הנתונים המוחלפים בין יישומים. זה חשוב ליישומים המחליפים נתונים רגישים, כגון מספרי כרטיסי אשראי או מידע אישי.



פרוטוקול HTTP/HTTPS:

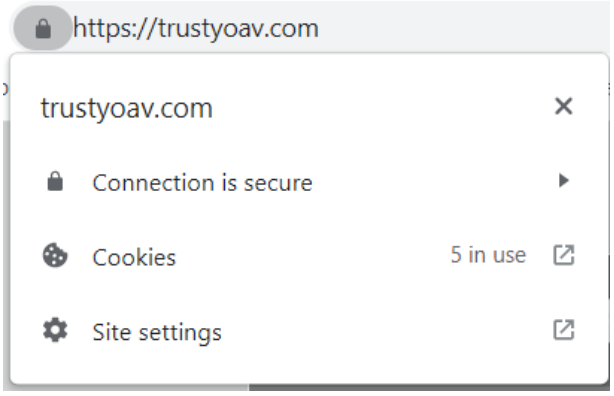
HTTP (פרוטוקול היפרטקסט) הוא פרוטוקול המשמש להעברת נתונים דרך האינטרנט. זהו הבסיס לתקשורת נתונים עבור ה-World Wide Web ומשמש לשליחה וקבלה של נתונים בין דפדפני אינטרנט ושרתי אינטרנט. HTTP הוא פרוטוקול תגובה לבקשה, שבו לקוח שולח בקשה לשרת והשרת מגיב עם הנתונים המבוקשים.

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.[6]



HTTPS, לעומת זאת, היא גרסה מאובטחת של HTTP המשתמשת בהצפנת SSL/TLS כדי לספק ערוץ תקשורת מאובטח בין הלקוח לשרת. כאשר משתמש שולח נתונים באמצעות HTTPS, הנתונים מוצפנים לפני שליחתם, וניתן לפענח אותם רק על ידי הנמען המיועד. זה מספק רמת אבטחה גבוהה יותר ומגן מפני ציתות, שיבוש נתונים וסוגים אחרים של התקפות.

דוגמה טובה לכך היא האתר שלי, המשתמש בפרוטוקול מוגן זה-

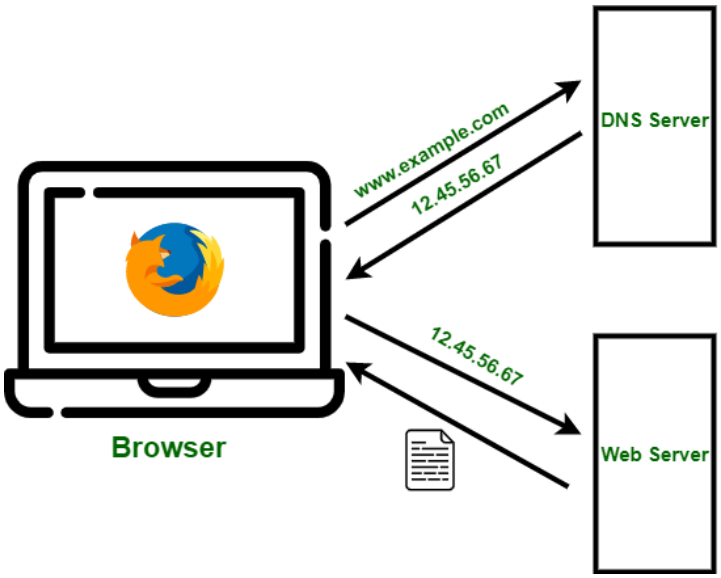


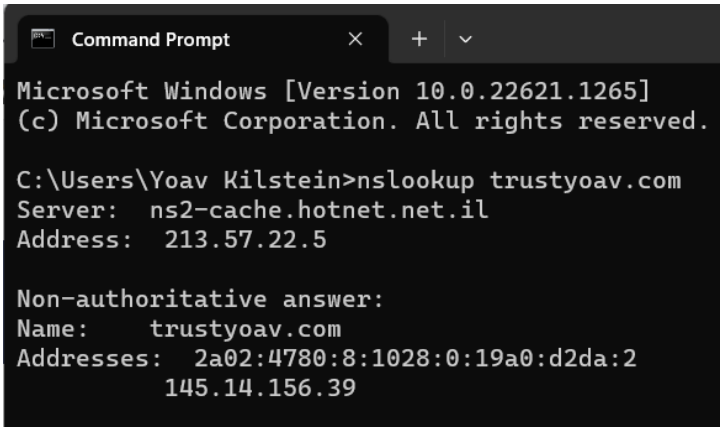
[6] https://he.wikipedia.org/wiki/HTTPS

פרוטוקול DNS:

DNS (מערכת שמות דומיין) 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.[7]

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 הDNS כספר טלפונים דיגטלי - 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 דיגטלי כתובת היעד תהיה מספר טלפון, בעוד בשרת DNS נקבל כתובת IP.





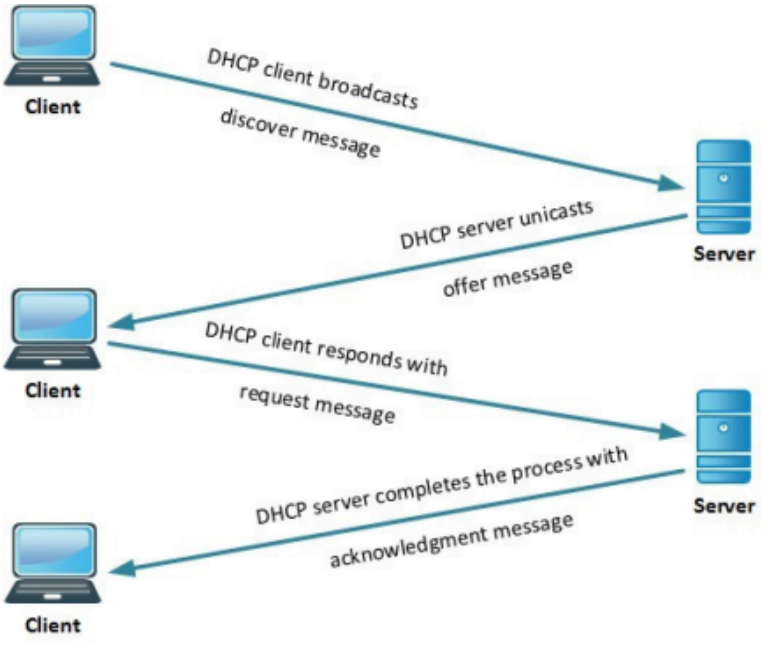
כלומר במקרה של האתר שלי, הכתובת שאנחנו מחפשים - trustyoav.com בעצם מגיעה מכתובת הIP (של שירות אחסון האתר שלי - hostinger):
145.14.156.39

[7] *Untitled*. (2019, February 5). המרכז לחינוך סייבר. Retrieved February 19, 2023, from https://data.cyber.org.il/networks/networks.pdf

פרוטוקול DHCP:

DHCP הוא פרוטוקול רשת המשמש להקצאה אוטומטית של כתובות IP (אינטרנט פרוטוקול) והגדרות תצורת רשת אחרות להתקנים ברשת. היא מאפשרת למנהלי רשת לנהל ולהקצות באופן מרכזי כתובות IP, ובכך לפשט את ניהול הרשת ולהפחית את הפוטנציאל לשגיאות.

- אופן הפעולה של הפרוטוקול:



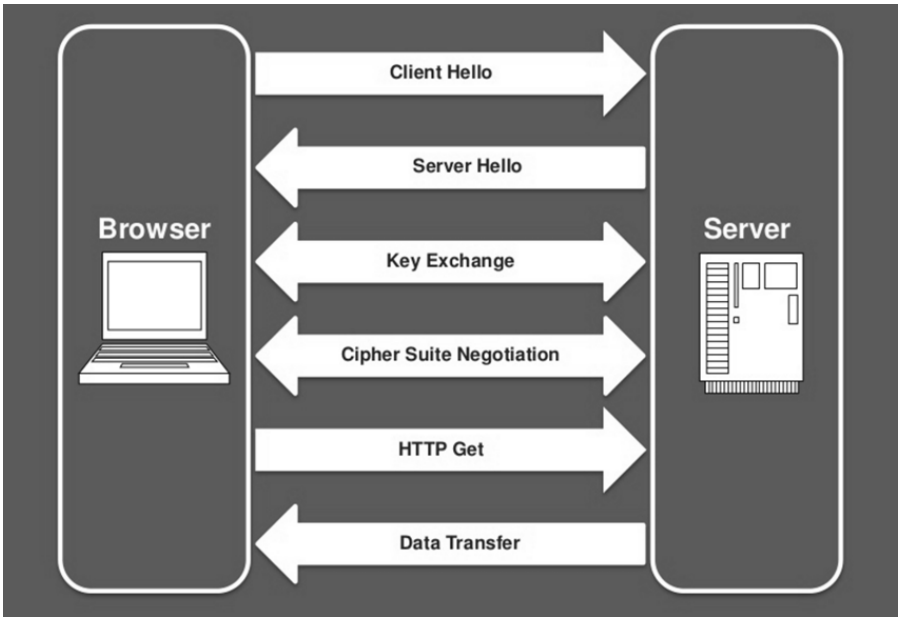
פרוטוקול SSL/TLS:

TLS/SSL הוא פרוטוקול אבטחה המספק תקשורת מאובטחת דרך האינטרנט. הוא משמש להצפנת נתונים המועברים בין שרת אינטרנט לדפדפן אינטרנט, ומבטיח שמידע רגיש, כגון סיסמאות, מספרי כרטיסי אשראי ומידע אישי, מוגן מפני גישה בלתי מורשית.

פועל על ידי יצירת חיבור מאובטח בין שרת אינטרנט לדפדפן אינטרנט, תוך שימוש בשילוב של הצפנה ואימות. כאשר משתמש מתחבר לאתר שמשתמש ב-SSL/TLS, דפדפן האינטרנט שלו מבקש משרת האינטרנט לזהות את עצמו. לאחר מכן שרת האינטרנט שולח לדפדפן האינטרנט תעודה דיגיטלית, המכילה מידע על זהות שרת האינטרנט, ואת המפתח הציבורי המשמש להצפנת הנתונים המועברים בין שרת האינטרנט לדפדפן האינטרנט.

לאחר מכן, דפדפן האינטרנט מאמת את האישור הדיגיטלי כדי לוודא שהוא תקף והונפק על ידי רשות אישורים מהימנה. אם האישור תקף, דפדפן האינטרנט משתמש במפתח הציבורי הכלול בתעודה כדי להצפין נתונים שנשלחים לשרת האינטרנט. שרת האינטרנט משתמש במפתח הפרטי שלו כדי לפענח את הנתונים המוצפנים ולהצפין נתונים שנשלחים חזרה לדפדפן האינטרנט.[8]

TLS/SSL מספק מספר יתרונות אבטחה, כולל פרטיות, שלמות ואימות. על ידי הצפנת נתונים המועברים דרך האינטרנט, מבטיח שלא ניתן ליירט מידע רגיש ולקרוא אותו על ידי גורמים לא מורשים. על ידי שימוש בתעודות דיגיטליות לאימות שרת האינטרנט, הוא מבטיח שהמשתמשים מתקשרים עם האתר המיועד ולא עם מתחזה.



[8] What happens in a TLS handshake? | SSL handshake | Cloudflare

פרוטוקולים TCP ו-UDP:

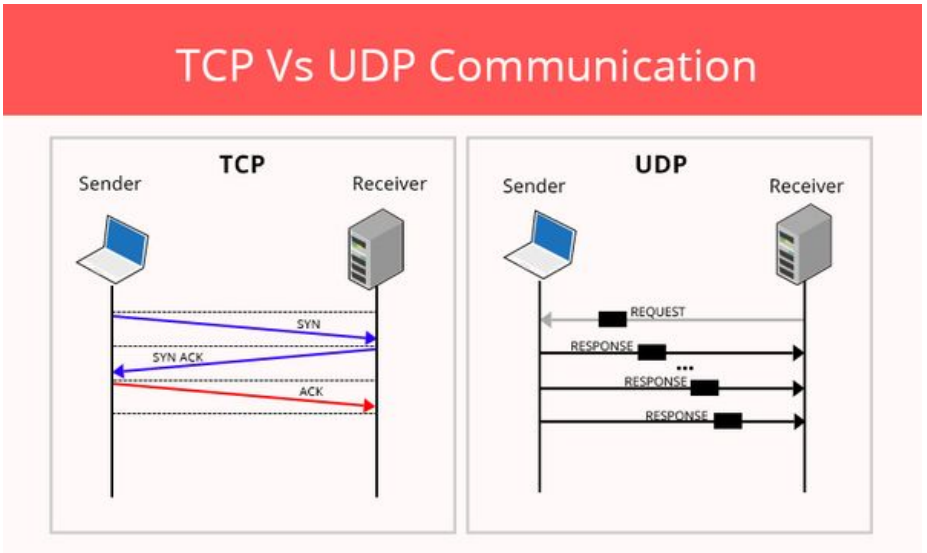
TCP ו-UDP הם שני פרוטוקולי שכבת תחבורה בשימוש נרחב ברשתות מחשבים.

TCP הוא פרוטוקול מכוון 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). לאחר יצירת החיבור, שני היישומים יכולים להחליף מנות נתונים, ו-TCP מבטיח שהמנות יסופקו בסדר הנכון, ללא אובדן או שכפול של מנות.

UDP, לעומת זאת, הוא פרוטוקול חסר חיבור שאינו מספק כל ערבויות לגבי מסירה, הזמנה או נכונות של מנות נתונים. יישומים המשתמשים ב-UDP פשוט שולחים מנות לכתובת יעד, ואין אימות שהמנות התקבלו על ידי היעד. זה הופך את UDP למהיר ויעיל יותר מ-TCP, אך גם פחות אמין.

TCP משמש בדרך כלל עבור יישומים הדורשים מסירה אמינה ומסודרת של נתונים, כגון דואר אלקטרוני, העברות קבצים וגלישה באינטרנט. UDP משמש בדרך כלל עבור יישומים הדורשים אספקה מהירה עם זמן אחזור של נתונים, כגון משחקים מקוונים, הזרמת וידאו וקול על גבי IP וכדומה.

בעוד TCP ו-UDP שונים במובנים רבים, לשניהם יש את מקומם בתקשורת רשת מודרנית. הבנת החוזקות והחולשות של כל פרוטוקול חשובה למנהלי רשת ולמפתחי יישומים בעת תכנון והטמעה של יישומי רשת.



חלק רביעי: רקע מעשי

תהליך העבודה

מהלך פרויקט זה התחיל בצלילה עמוקה לתוך שפת React.js. באמצעות סרטוני לימוד hands-on, קריאה, תרגול ובניית פרויקטים קטנים - פיתחתי הבנה חזקה ושליטה בשפת React.js. על מנת לעצב את ממשק המשתמש, הקדשתי זמן גם ללימוד CSS. CSS היא השפה העיקרית המשמשת לעיצוב דפי אינטרנט, ומצאתי שחיוני להבין כיצד היא עובדת ואת החלקים החשובים לדעת בעת שימוש בה. השתמשתי במיני-פרויקטים שבניתי עם ריאקט כדי להתנסות בעבודה עם משולבת עם CSS.

כדי לאחסן ולנהל נתוני משתמשים, החלטתי להשתמש ב-Firebase, פלטפורמה בבעלות גוגל המספקת שירותי מסד נתונים ואימות בזמן אמת. למדתי כיצד לשלב את Firebase עם React.js באמצעות סרטוני הדרכה, והצלחתי להגדיר את תהליך האימות ומסד נתונים עבור הפרויקט שלי דרך שם.
על מנת להעלות את הפרויקט לרשת, בחרתי בשירותי אחסון אתרים של Hostinger. אשר דרכם הצלחתי להעלות את הפרויקט בהצלחה לכתובת אינטרנט חיה.

תהליך כתיבת הקוד ופיתוח הפרויקט התחיל עם בניית ממשק בסיסי לאתר ולאחר מכן הוספתי בהדרגה תכונות שונות - כמו עדכוני מצב בזמן אמת, אימות משתמשים, שיתוף תמונות ועוד.
בנוסף למיומנויות הטכניות, פיתחתי גם ידע ברשתות ושיטות הצפנה. למדתי על החשיבות של אבטחת נתוני משתמש ועל השיטות השונות בהן נעשה שימוש כדי להשיג זאת, כגון הצפנת HTTPS ואמצעי אבטחה בצד השרת. במהלך התהליך רכשתי ניסיון רב ערך בפיתוח אתרים, פתרון בעיות ושיתוף פעולה עם קהילות מקוונות (כמו stackoverflow).

מבנה הפרויקט

בחלק זה של העבודה יופיעו מספר מושגים אשר לא בטוח יהיו בהירים לחלוטין לקורא אשר לא מגיע מהתחום.
על כן, אני מזמין אתכם לעיין במונחון המופיע בנספחים כדי להבין מושגים אלו יותר לעומק.

:Components

Chat.jsx-

רכיב ה-Chat מייצג את הקונטיינר הראשי של ממשק הצ'אט, הכולל כותרת המציגה את שם התצוגה של המשתמש, קבוצה של סמלי צ'אט, אזור תצוגת הודעות ושדה קלט עבור המשתמשים להזנת הודעות.
הרכיב משתמש ב-useContext כדי לגשת לנתוני המצב שמסופקים על ידי ChatContext - קשר המשמש לניהול מצב הצ'אט והנתונים הכוללים.
הרכיב מייבא לתוכו שני רכיבים אחרים עליהם נדון בהמשך (הודעות, וקלט).

Chats.jsx-

רכיב ה-Chats אחראי לעיבוד רשימת כל הצ'אטים שהמשתמש הנוכחי הוא חלק מהם. כל צ'אט מיוצג על ידי רכיב UserChat נפרד, המכיל את תמונת פרופיל המשתמש של הצ'אט, שם המשתמש וההודעה האחרונה שנשלחה 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" במסמך userChats עבור ה-UID של המשתמש הנוכחי, כך שכל שינוי במסמך יפעיל עיבוד מחדש של הרכיב עם הנתונים המעודכנים. הפונקציה unsub 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. רכיבי userChat ממוינים בסדר יורד על סמך זמן שליחת ההודעה האחרונה בהם.
- הרכיב משתמש גם ב-useContext כדי לגשת לערכי המשתמש הנוכחיים ולשליחה מה-AuthContext ו-ChatContext, בהתאמה. קשרים אלו משמשים לניהול מצב האימות ומצב הצ'אט של האפליקציה.
  - הפונקציה handleSelect 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.

להלן הסבר קצר על השיטות והמאפיינים העיקריים המשומשים ברכיב הצ'אטים:

1. **useState:** ההוק הזה משמש ליצירת צ'אטים של משתנה מצב מקומי לאחסון נתוני הצ'אט שנלקחו מ-Firebase.

2. **useEffect:** הוק זה משמש כדי להביא את נתוני הצ'אטים מ-Firebase באמצעות שיטת onSnapshot כאשר הרכיב נטען. זה גם מבטל את המנוי מהמאזין כאשר הרכיב מתבטל.
3. **onSnapshot:** שיטה זו מגדירה מאזין חי במסמך userChats עבור ה-UID של המשתמש הנוכחי ב-Firebase, כך שכל שינוי במסמך יפעיל עיבוד מחדש של הרכיב עם הנתונים המעודכנים.
4. **map:** שיטה זו משמשת לחזרה על מערך הצ'אטים ולעיבוד רכיב ה-userChat עבור כל צ'אט.
5. **sort:** שיטה זו משמשת למיון רכיבי userChat בסדר יורד על סמך תאריך ההודעה האחרונה שלהם.
6. **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-

רכיב המייצג את קטע הקלט של צ'אט שבו המשתמש יכול להזין את ההודעות שלו כדי לשלוח לצ'אט שנבחר. הרכיב מעבד שדה קלט וכפתור לשליחת הודעות, והוא גם מאפשר למשתמש לבחור תמונה מהקבצים השמורים שלו כדי לשלוח אותה כהודעה. הרכיב גם מייבא ומשתמש במספר שירותי Firebase כדי להעלות את התמונה שנבחרה לאחסון במסד הנתונים ולאחר מכן לשמור את כתובת הקובץ שלו להורדה במסד הנתונים של Firestore, יחד עם טקסט ההודעה, מספר זהות השולח והתאריך.

השיטות המשמשות ברכיב:

- **handleSend():** שיטה זו נקראת כאשר המשתמש לוחץ על כפתור השליחה. הוא בודק אם הודעת הקלט אינה ריקה והאם נבחרה תמונה או לא. לאחר מכן, הוא מעלה את התמונה לאחסון Firebase ושומר את כתובת האתר להורדה שלה, יחד עם טקסט ההודעה, מזהה השולח והתאריך במסד הנתונים של Firestore. זה גם מעדכן את ההודעה והתאריך האחרונים בצ'אטים של השולח והמקבל במסד הנתונים של Firestore. לבסוף, הוא מאפס את מצבי הקלט והתמונה.
- **setImg():** שיטה זו נקראת כאשר המשתמש בוחר תמונה מהמערכת המקומית שלו. זה מגדיר את התמונה שנבחרה למצב img של הרכיב.
- **setText():** שיטה זו נקראת כאשר המשתמש מקליד הודעה בשדה הקלט. הוא מגדיר את הטקסט המוקלד למצב הטקסט של הרכיב.

Message.jsx-

זהו רכיב שמייצג הודעה בודדת בשיחה.

- מייבא הוקים של useRef ו-useEffect מ-React, כמו גם ההקשרים של AuthContext ו-ChatContext שהוגדרו במקום אחר באפליקציה.

- מקבל את פרטי ההודעה לעיבוד ומשתמש ב- currentUser ובאובייקטי data מה-ChatContext כדי לקבוע את פרטי שולח ההודעה ואת המשתמש שאיתו המשתמש הנוכחי משוחח.
- משתמש ב-useRef כדי לקבל הפניה לרכיב ה-HTML שמכיל את ההודעה, וב-useEffect כדי לגלול את השיחה לתחתית המסך בכל פעם שמתקבלת או נשלחת הודעה חדשה.

לאחר מכן, נעשה עיבוד של ההודעה, כולל של תמונת הפרופיל של השולח, טקסט ההודעה וכל תמונה שנשלחה עם ההודעה יחדיו. אם המשתמש הנוכחי שלח את ההודעה, ההודעה מוצגת בצד ימין של המסך, אחרת היא מוצגת בצד שמאל של המסך.

Messages.jsx-

זהו רכיב שמייצג את ההודעות בצאט. הוא מביא את ההודעות ממסד הנתונים של Firestore ומציג אותן באמצעות רכיב ההודעה. בעזרת הוק של useEffect הוא מעודכן על אוסף ההודעות ומציג מחדש את הרכיב כאשר הנתונים משתנים.

- ה-useEffect מגדיר "מאזין" לתמונת המצב באוסף ההודעות של הצ'אט הנוכחי. כאשר הרכיב נטען, הוא קורא לשיטת onSnapshot באובייקט ה-doc, אשר מחזירה פונקציה שניתן להשתמש בה כדי לבטל את העדכונים.
- שיטת onSnapshot לוקחת אובייקט ה-doc המייצג את המסמך להאזנה לעדכונים, ופונקציית callback שתיקרא 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 מעבד את רכיבי ההודעה עבור כל הודעה במערך ההודעות בשיטת המפה. לכל רכיב הודעה מועבר המידע הרלוונטי אשר באמצעותו הוא מעבד את תוכן ההודעה.

Navbar.jsx-

זהו רכיב ניווט פשוט הכולל את הדברים הבאים:

- הצגת השם של האתר.
- הצגת השם והתמונה של המשתמש הנוכחי.
- מכיל כפתור יציאה שקורא לפונקציית היציאה מאובייקט האישור ב-Firebase SDK כדי להתנתק ולצאת מהמשתמש.

- 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.
- 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.
  - 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 המזהה של מסמך הצ'אט, יחד עם תאריך יצירת הצ'אט ופרטי המשתמש עבור שני המשתמשים.

Sidebar.jsx-

רכיב הסרגל הצידי הוא הפריסה הראשית של האפליקציה, והוא כולל את הרכיבים Navbar, חיפוש וצ'אטים.

:Pages

Home.jsx-

העמוד הראשי של אפליקציית הצ'אט. העמוד מכיל את רכיב הסרגל צד בצד שמאל של הדף הכולל סרגל Navbar, רכיב חיפוש ורכיב צ'אטים. בצד ימין של הדף, ישנו רכיב צ'אט המציג את ההודעות של שיחת צ'אט שנבחרה (עליו פירטתי קודם לכן).

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'. הוא קולט אובייקט אירוע ומחלץ את האימייל והסיסמה משדות הקלט. לאחר מכן, הוא משתמש 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.

Imports:

- React - לאפשר שימוש ברכיבי React.
- useState - לאפשר ניהול מצב בתוך הרכיב.
- Add - להתייחס לתמונה המשמשת כלחצן "הוסף אווטאר".
- createUserWithEmailAndPassword ממודול Firebase/Auth - ליצור חשבון משתמש חדש.
- updateProfile ממודול Firebase/Auth - לעדכן את פרטי הפרופיל של המשתמש, כגון שם התצוגה ותמונת הפרופיל.
- auth ממודול Firebase - לגשת לשירותי אימות Firebase.
- db ממודול Firebase - לגשת ל-Firestore.

- Storage ממודול Firebase - לגשת ל-Firebase Storage.
- ref ממודול ה-firebase/storage - להתייחס למיקום האחסון של תמונת הפרופיל של המשתמש.
- uploadBytesResumable ממודול ה-firebase/storage - להעלות את תמונת הפרופיל של המשתמש.
- getDownloadURL ממודול ה-firebase/storage - לאחזר את כתובת האתר להורדה של התמונה שהועלתה.
- doc מהמודול Firebase/Firestore - להפנות למסמך ב-Firestore.
- setDoc מהמודול Firebase/Firestore - להגדיר מסמך ב-Firestore.
- useNavigate מהמודול react-router-dom - לאפשר ניווט בצד הלקוח בתוך האפליקציה.
- Link ממודול react-router-dom - לאפשר יצירת קישור בתוך האפליקציה.

העמוד מוגדר כפונקציה 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

AuthContext.js-

זהו מודול שמייצא שני רכיבים: AuthContext ו-AuthContextProvider. הוא אחראי 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.

ChatContext.js-

מגדיר רכיב "ChatContextProvider" המספק "ChatContext" לכל ילדיו 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.

App.js:

קובץ זה הוא נקודת הכניסה הראשית של הפרויקט והוא מגדיר את הניתוב עבור דפים שונים.

- ייבוא רכיבים וגיליונות סגנונות נחוצים מקובץ הscss שלי (עליו ארחיב בהמשך).
- הגדרת רכיב ה'App'.
- שימוש בהוק useContext כדי לקבל את המשתמש הנוכחי מה-AuthContext.
- הגדרת ProtectedRoute שבודק אם המשתמש מאומת, ואם לא, הוא מנווט אותו לדף הכניסה.
- הגדרת מסלולים לדפי הבית, התחברות והרשמה, כאשר עמוד הבית הוא מסלול מוגן.
- ייצוא רכיב האפליקציה כברירת מחדל.

המסלולים לדפים השונים דרך ה"Route" הרגיל וה-"protectedRoute" נעשה על ידי ייבוא של "react-router-dom".

firebase.js:

זהו מודול שמייצא את פונקציות ה-SDK של Firebase כדי לאתחל את אפליקציית Firebase ולקבל את שירותי האימות והאחסון שלה.

הפונקציות שמכילה:

1. **initializeApp:** מאתחל ומחזיר מופע של אפליקציית Firebase עם התצורה הנתונה. פונקציה זו לוקחת אובייקט תצורה כארגומנט היחיד שלה, המכיל מידע על פרויקט Firebase שאליו ברצונך להתחבר.
2. **getAuth:** מחזירה את מופע שירות Firebase Auth עבור אפליקציית ברירת המחדל או אפליקציה נתונה. ניתן להשתמש בפונקציה זו כדי לאמת משתמשים באפליקציה שלך ולבצע פעולות שונות הקשורות לאימות, כגון כניסה, הרשמה וניהול חשבונות משתמש.
3. **getStorage:** מחזירה את מופע שירות Firebase Storage עבור אפליקציית ברירת המחדל או אפליקציה נתונה. ניתן להשתמש בפונקציה זו כדי להעלות, להוריד ולנהל קבצים באפליקציה שלך.
4. **getFirestore:** מחזירה את מופע השירות Firebase Firestore עבור אפליקציית ברירת המחדל או אפליקציה נתונה. ניתן להשתמש בפונקציה זו כדי לגשת ולתפעל נתונים במסד הנתונים של Firestore, שהוא מסד נתונים של מסמכי NoSQL המסופק על ידי Firebase.

:index.js

בindex.js המטרה היא להציג את רכיב הApp ב-AuthContextProvider ו-ChatContextProvider.

- ה-AuthContextProvider מספק את הערך של ה-"currentUser" לרכיבי הצאצא שלו דרך ה-AuthContext.
- ChatContextProvider מספק אובייקט מצב ופונקציה לרכיבי הצאצא שלו באמצעות ChatContext.
- הפונקציה createRoot מחבילת ReactDOM משמשת ליצירת שורש לאפליקציה. הפונקציה root.render משמשת לעיבוד רכיב האפליקציה בתוך אלמנט ה'root'.
- הרכיב <React.StrictMode> משמש כדי לעטוף את רכיב ה'App' כדי לאפשר את המצב המחמיר של React. זה מאפשר אזהרות ובדיקות נוספות כדי לעזור למצוא ולתקן בעיות פוטנציאליות באפליקציה.

:style.scss

הקוד בקובץ זה מגדיר וקובע את עיצוב האתר. הקוד כתוב בשפת Sass, שהיא שפה שנועדה להוסיף מנגנונים של שימוש חוזר לקוד ב-CSS. מאפשר שימוש במגוון כלים ייחודיים שלא מתאפשרים לשימוש בעיצוב בשפת CSS רגילה.

:img

בתיקייה זאת שמורות כל התמונות אשר בהן השתמשתי בפרויקט ומשם ייבאתי אותן לעמודים הרלוונטים.

1. **"AvatarAdd.png"** - תמונה מעמוד ההרשמה שעליה ניתן ללחוץ כדי להוסיף תמונה למשתמש.
2. **"img.png" -** תמונה מעמוד הבית/הצאט שעליה ניתן ללחוץ כדי להוסיף ולשלוח קובץ תמונה למשתמש אחר.

סיכום אישי, דיון ומסקנות:

הפרויקט נועד לפתח אפליקציית צ'אט מבוססת web תוך שימוש בטכנולוגיות שונות לפיתוחו. עבור בניית הפרויקט היה צורך בהוספת תכונות כמו אימות משתמשים, שליחה וקבלה של הודעות.

כעת נדון בהיבטים השונים של הפרויקט, לרבות הבעיות שעמדו בפניי בכתיבתו, הפתרונות שיושמו ואפשרויות פיתוח עתידיות עבורו:

1. בתהליך כתיבת העבודה נתקלתי במספר אתגרים שדרשו פתרונות ייחודיים. אחת הבעיות המרכזיות הייתה יישום תהליך האימות, שדרש רישום וכניסה בטוחה למשתמשים. לאחר מחקר רב על האפשרויות השונות בחרתי להשתמש ב-API של Firebase לאימות ליצירת חשבונות משתמש חדשים ולאימות חשבונות קיימים. הפשטות של ה-API הקלה על שילוב תכונת האימות.
2. נושא משמעותי נוסף היה יצירת חדרי צ'אט. ה-ChatContext יושם כדי לנהל את מצב אפליקציית הצ'אט. השימוש בcontext זה, אפשר יצירת חדרי צ'אט ייחודיים, וניתן היה לשלוח ולקבל הודעות בזמן אמת. היישום של ChatContext דרש הבנה מעמיקה בReact Hooks וכיצד לעשות בהם שימוש.
3. הצפנה ואבטחת המידע של המשתמש -

- Hostinger מספקת אפשרויות אבטחה שונות, כולל שימוש ב-HTTPS ובתעודות SSL/TLS להצפנת נתונים. בנוסף, דרך hostinger קיבלתי גישה לשרתים מוגנים ב-Cloudflare להגנה מתקיפות DDOS.
- עשיתי שימוש גם במסד הנתונים בזמן אמת של Firebase כדי לאחסן את מידע המשתמשים וההודעות באופן מוצפן - מה שסיפק שכבת אבטחה נוספת.

- מגבלה שבה נתקלתי היא לנסות ליישם הצפנת RSA על ההודעות הנשלחות בצאט. עם זאת, חקרתי ולמדתי המון על ההצפנה ומאמין כי בעתיד כאשר אמשיך לפתח ולחקור את הפרויקט שלי אוסיף הצפנה זו.
- חשוב לי לציין אפשרות פיתוח נוספת לעתיד שהייתי רוצה ליישם באתר ושדנתי בה בהצעת המחקר - הטמעת אפשרות לשיחה מרובת משתתפים.

לסיכום, הצלחתי "לבנות סביבה המאפשרת אינטרקציה נוחה, בטוחה ויעילה בין אנשים בעלי תחומי עניין דומים" (שאלת החקר). הבנתי ששילוב בין כתיבת קוד איכותי וברור, שמירה על פרטיות המשתמשים בעזרת מסד נתונים אמין והסתמכות על שרתים חזקים ומאובטחים לאתר יאפשרו לי להתמודד בצורה טובה עם פיתוח הפרויקט - וכך היה.
פיתוח הפרויקט היה עבורי חוויה לימודית יקרת ערך. בזכות תהליך המחקר, התכנון והיישום של הפרויקט שלי, רכשתי הבנה עמוקה ב-React, CSS, Firebase וטכנולוגיות נוספות הנמצאות בשימוש נרחב בתחום פיתוח התוכנה. אני מאמין שהכישורים והידע שצברתי בתהליך העבודה יהיו בעלי משמעות רבה במאמציי העתידיים כמפתח תוכנה. על כן, אני מאוד מרוצה וגאה בתוצר הסופי שאליו הצלחתי להגיע.

ביבליוגרפיה

1. *LEARN REACT*. React. (n.d.). Retrieved 2023, from https://react.dev/learn
2. *Add data to Cloud Firestore | Firebase*. (n.d.). Firebase. Retrieved February 19, 2023, from https://firebase.google.com/docs/firestore/manage-data/add-data
3. *Authenticate with Firebase using Password-Based Accounts using Javascript*. (n.d.). Firebase. Retrieved February 19, 2023, from https://firebase.google.com/docs/auth/web/password-auth
4. Introducing hooks. React. (n.d.-a). https://legacy.reactjs.org/docs/hooks-intro.html
5. https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/#:~:text=What%20happens%20during,handshake%20is%20complete What happens in a TLS handshake? | SSL handshake | Cloudflare
6. *Feature Overview v6.8.1*. (n.d.). React Router. Retrieved February 19, from https://reactrouter.com/en/main/start/overview
7. https://media.geeksforgeeks.org/wp-content/uploads/20210908120846/DOM.png *DOM*. (n.d.). Geeksforgeeks.
8. ויקיפדיה https://he.wikipedia.org/wiki/HTTPS
9. *Get realtime updates with Cloud Firestore | Firebase*. (n.d.). Firebase. Retrieved February 19, 2023, from https://firebase.google.com/docs/firestore/query-data/listen
10. Educative. (n.d.). What is Firebase? Educative: Interactive Courses for Software Developers. https://www.educative.io/answers/what-is-firebase
11. *Perform simple and compound queries in Cloud Firestore | Firebase*. (n.d.). Firebase. Retrieved February 19, 2023, from https://firebase.google.com/docs/firestore/query-data/queries
12. . (2019, February 5). המרכז לחינוך סייבר., from https://data.cyber.org.il/networks/networks.pdf
13. https://morioh.com/p/52eb73408b9a#:~:text=In%20this%20brief,in%20the%20tree. what is a consumer in react
14. . (n.d.). Hostinger. Retrieved February 19, 2023, from https://www.hostinger.co.il/web-hosting

נספחים

מונחון

אלו הם מספר מושגים שבהם השתמשתי בעת הסברת חלקי הפרויקט ברקע המעשי אשר היה צורך לנמקם

- **Consumer - צרכן**

רכיב המקבל באופן פעיל עדכונים או הודעות על שינויים באובייקט "context". אובייקט זה הוא דרך עבור התוכנית להעביר מידע בקלות, מבלי שתצטרך לעשות זאת ידנית בכל פעם. אובייקט ה"context" נוצר באמצעות כלי מיוחד שנקרא "createContext", והוא יכול לאחסן מידע שכל חלק בתוכנית יכול לגשת אליו.
ה"consumer" הוא כלי מובנה המסייע לחלק מהתוכנית לגשת לנתוני ה"context". זה עובד על ידי קבלת פונקציה בתור הילד שלה, ומתן לפונקציה הזו את ערך ה"context" הנוכחי. אתה יכול להשתמש בconsumer בחלקים רבים ושונים של התוכנית, כך שחלקים שונים יכולים לגשת לסיביות שונות של נתוני ה-context.[9]

- **Firebase SDK**

ערכת פיתוח תוכנה (Software Development Kit) היא אוסף של רכיבי תוכנה מובנים מראש שמפתחים יכולים להשתמש בהם כדי לשלב שירותי Firebase ביישומי המובייל או האינטרנט שלהם. הוא מספק גישה לתכונות שונות כגון מסד נתונים בזמן אמת, אימות, אירוח, ניתוח ועוד - מה שמקל על מפתחים לבנות אפליקציות איכותיות ועתירות תכונות במאמץ מופחת.

- **Hook - הוק[10]**

פונקציה מיוחדת המאפשרת לך להשתמש בתכונות React כמו "state" בתוך רכיבים פונקציונליים. מבלי ההוק - שיטות שונות היו זמינות רק ברכיבי המחלקה.
הוקים אשר בהם עשיתי שימוש והסברתי עליהם בפרויקט:

1. useState: מאפשר לרכיבים להוסיף state. המצב הוא מידע/נתונים שיכולים להשתנות לאורך זמן על ידי גורמים שונים.
2. useEffect: מאפשר "לומר" ל-React לבצע מעין "תופעות לוואי" על הרכיבים לאחר רינדור הרכיב. תופעת לוואי היא כל קוד שמקיים אינטראקציה עם העולם החיצון, כמו שליפת נתונים מ-API, עדכון כותרת הדפדפן או הגדרת טיימר.
3. 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 מישהו/משהו אחר כדי לקבל אותם.
4. useReducer: מאפשרת לנהל מצבים מורכבים ברכיבים שלנו. מקבל פונקציית reducer וערך מצב ראשוני כקלט, ומחזיר מערך עם הערך הנוכחי של המצב ופונקציית dispatch. זה

[9] https://morioh.com/p/52eb73408b9a#:~:text=In%20this%20brief,in%20the%20tree. what is a consumer in react
[10] Introducing hooks. React. (n.d.-a). https://legacy.reactjs.org/docs/hooks-intro.html

שימושי כשצריך לנהל מצב שכולל מספר משתנים ודורש לוגיקה מורכבת יותר ממה ש-useState יכולה לטפל בו.

5. useRef: מאוד דומה לuseState אבל מטרתו היא לשמור על ערך או נתונים ולא לעשות רינדור מחדש כאשר הערך מתעדכן.

- **JSX**

JSX הוא הרחבה של JS המשמשת להגדרת המבנה והפריסה של רכיבי ממשק משתמש. הרחבה זאת מאפשרת למפתחים לכתוב קוד כמו HTML לדוגמה בקבצי JavaScript שלהם.

- **state variable**

משתנה מצב מאחסן מידע על המצב הנוכחי של מערכת או תהליך, המשמש לעתים קרובות בפיתוח חזיתי לניהול מצב ממשק המשתמש. על ידי עדכון משתני מצב, מפתחים יכולים להפעיל עיבוד מחדש של רכיבי ממשק משתמש, ליצור ממשקי משתמש דינמיים ומגיבים המגיבים לקלט המשתמש ואירועי מערכת.

הקוד המלא

```
import React, { useContext } from 'react'
import Messages from "./Messages";
import Input from "./Input";
import { ChatContext } from '../context/ChatContext';

const Chat = () => {
  const{data} = useContext(ChatContext); // use ChatContext to
get data

  return (
    <div className='chat'>
      <div className="chatInfo">
        <span>{data.user?.displayName}</span> // display the
user's display name
        <div className="chatIcons">
          {} // placeholder for chat icons
        </div>
      </div>
      <Messages /> // display messages
      <Input/> // input field for new messages
    </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); // get dispatch
function from 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">
      {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="" /> //
display user's photo
            <div className="userChatInfo">
              <span>{chat[1].userInfo.displayName}</span> //
display user's display name
              <p>{chat[1].lastMessage?.text}</p> // display last
message's text
            </div>
          </div>
        ))}
      </div>
    );
};

export default Chats;
```

```
import React, { useContext, useState } from "react";
import Img from "../img/img.png";
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 } from
"firebase/storage";

const Input = () => {
  // Set up state for the text input and image file
  const [text, setText] = useState("");
  const [img, setImg] = useState(null);

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

  // Handle sending a message (with or without an image)
  const handleSend = async () => {
    // Check if there is no text or image before sending
    if (!text.trim() && !img) return;

    if (img) {
      // If an image is selected, upload it to Firebase Storage
and add the download URL to the message data
      const storageRef = ref(storage, uuid());

      const uploadTask = uploadBytesResumable(storageRef, img);

      uploadTask.on(
        (error) => {
          console.error(error);
        },
```

```
            () => {
              getDownloadURL(uploadTask.snapshot.ref).then(async
(downloadURL) => {
                await updateDoc(doc(db, "chats", data.chatId), {
                  messages: arrayUnion({
                    id: uuid(),
                    text,
                    senderId: currentUser.uid,
                    date: Timestamp.now(),
                    img: downloadURL,
                  }),
                });
              });
            }
          );
        } else {
          // If no image is selected, simply add the text message to
the chat data
          await updateDoc(doc(db, "chats", data.chatId), {
            messages: arrayUnion({
              id: uuid(),
              text,
              senderId: currentUser.uid,
              date: Timestamp.now(),
            }),
          });
        }

        // Update the last message 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,
          },
          [data.chatId + ".date"]: serverTimestamp(),
        });

        await updateDoc(doc(db, "userChats", data.user.uid), {
          [data.chatId + ".lastMessage"]: {
            text,
          },
          [data.chatId + ".date"]: serverTimestamp(),
        });
```

```
    // Clear the input fields after sending
    setText("");
    setImg(null);
  };

  return (
    <div className="input">
      {/* The text input */}
      <input
        type="text"
        placeholder="Type something..."
        onChange={(e) => setText(e.target.value)}
        value={text}
      />

      {/* The "attach file" button */}
      <div className="send">
        <input
          type="file"
          style={{ display: "none" }}
          id="file"
          onChange={(e) => setImg(e.target.files[0])}
        />
        <label htmlFor="file">
          <img src={Img} alt="" />
        </label>

        {/* The "send" button */}
        <button onClick={handleSend}>Send</button>
      </div>
    </div>
  );
};

export default Input;
```

```
import React, { useContext, useEffect, useRef } 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();

  useEffect(() => {
    ref.current?.scrollIntoView({behavior:"smooth"})
  }, [message]); // make scrolling smooth

  return (
    <div ref={ref}
    className={`message ${message.senderId === currentUser.uid &&
"owner"}`}> {/*Get the sender's profile picture, if any, else the
default one.*/}
      <div className="messageInfo">
        <img
          src={
            message.senderId === currentUser.uid
            ? currentUser.photoURL
            : data.user.photoURL
          } // User photo
          alt=""
        />
        <span></span>
      </div>
      <div className="messageContent">
        <p>{message.text}</p>
        {message.img && <img src={message.img} alt="" />}  {/*If
there is an image the user wants to send, save it*/}
      </div>
    </div>
  );
};

export default Message;
```

```
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 = () => {
  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])

  console.log(messages)

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

export default Messages;
```

```
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">Trust Yoav</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 {
  collection,
  query,
  where,
  getDocs,
  setDoc,
  doc,
  updateDoc,
  serverTimestamp,
  getDoc,
} from "firebase/firestore";
import { db } from "../firebase";
import { AuthContext } from "../context/AuthContext";

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

  const handleSearch = async () => {
    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();
  }; // 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="Find a user"
              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;
```

```
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 { createContext, useEffect, useState } from "react";
import { auth } from "../firebase";
import { onAuthStateChanged } from "firebase/auth";


export const AuthContext = createContext();

export const AuthContextProvider = ({children}) =>{
    const [currentUser, setCurrentUser] = useState({});

    useEffect(() =>{
        const unsub = onAuthStateChanged(auth, (user) =>  {
            setCurrentUser(user);
            console.log(user);
        });

        return () =>{
            unsub();
        };
    }, []);
    return(
        <AuthContext.Provider value={{ currentUser }}>
          {children}
        </AuthContext.Provider>
    );
};
```

```
import { createContext, useContext, useReducer } from "react";

import { AuthContext } from "./AuthContext";


export const ChatContext = createContext();


// Defines a "ChatContextProvider" component that provides a
"ChatContext" to all its children using the "createContext"
method from React.
// This context can be accessed by any child component that is a
'consumer' of "ChatContext".
export const ChatContextProvider = ({children}) =>{
    const {currentUser} = useContext(AuthContext);
   const INITIAL_STATE = {
    chatId:"null",
    user:{}
   };


//Takes the current "state" and the "action" sent to update the
current state.
// The action that a state change can update is "CHANGE_USER"
which contains the user object.
   const chatReducer = (state, action)=>{
        switch(action.type){
            case "CHANGE_USER":
                return{
                    user: action.payload,
                    chatId:
                        currentUser.uid > action.payload.uid
                            ? currentUser.uid +
action.payload.uid
                            : action.payload.uid +
currentUser.uid,
                };

            default:
                return state;
        }
    };
```

```
    const [state, dispatch] = useReducer(chatReducer,
INITIAL_STATE);

    return(
        <ChatContext.Provider value={{ data:state, dispatch }}>
          {children}
        </ChatContext.Provider>
    );
};
```

```
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
```

```
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'>
            <span className="logo">Trust Yoav</span>
            <span className="title">Login</span>
            <form onSubmit={handleSubmit}>
```

```
                <input type="email" placeholder="email"/>
                <input type="password" placeholder="password"/>
                <button>Sign In</button>
                {err && <span>{err.message}</span>}
            </form>
            <p>You 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) => {
    e.preventDefault();

    // 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,}))$/;
    const isValidEmail = emailRegex.test(email);
    const isValidDomain = /@gmail\.com$/.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 (err.code === 'auth/email-already-in-use') {
        errorMessage = 'Email is already in use';
      } 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 storageRef = ref(storage, displayName);
      const uploadTask = uploadBytesResumable(storageRef, file);

      uploadTask.on(
        (error) => {
          setErr({ message: error.message });
        },
        () => {
          // Update the user's name and photo url in the database.
          getDownloadURL(uploadTask.snapshot.ref).then(
async(downloadURL) => {
            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 (err) {
    setErr({ message: err.message.slice(9) });
  }
```

```
  };

  return (
    <div className='formContainer'>
        <div className='formWrapper'>
            <span className="logo">Trust Yoav</span>
            <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>Add an avater:</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;
```

```
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>
  );
}

export default App;
```

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAuth } from "firebase/auth";
import { getStorage } from "firebase/storage";
import { getFirestore } from "firebase/firestore";


// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyBf-UiUfKbrAOP8P1ZWi7_-Km-gQkCNo0w",
  authDomain: "trust-yoav.firebaseapp.com",
  projectId: "trust-yoav",
  storageBucket: "trust-yoav.appspot.com",
  messagingSenderId: "406200167098",
  appId: "1:406200167098:web:5e694a8449feb0cb1ab083"
};

// Initialize Firebase
export const app = initializeApp(firebaseConfig);
export const auth = getAuth()
export const storage = getStorage();
export const db = getFirestore();
```

```
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>
);
```

```
@mixin mobile{
    @media screen and (max-width: 480px) {
        @content;
    }
}

@mixin tablet{
    @media screen and (max-width: 770px) {
        @content;
    }
}

@mixin laptop{
    @media screen and (max-width: 1200px) {
        @content;
    }
}

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

    .formWrapper{
        background-color: white;
        padding: 20px 60px;
        border-radius: 10px;
        display: flex;
        flex-direction: column;
        gap: 10px;
        align-items: center;

        .logo{
            color: blue;
            font-weight: bold;
            font-size: 24px;
        }

        .title{
            color: blue;
```

```
            font-size: 12px;
        }

        form{
            display: flex;
            flex-direction: column;
            gap: 15px;

            input{
                padding: 15px;
                border: none;
                width: 250px;
                border-bottom: 1px solid lightgray;
            }

            button{
                background-color: blue;
                color: white;
                padding: 10px;
                font-weight: bold;
                border: none;
                cursor: pointer;
            }

            label{
                display: flex;
                align-items: center;
                gap: 10px;
                color: black;
                font-size: 12px;
            }
        }
        p{
            color: darkcyan;
            font-size: 12px;
            margin-top: 10px;
        }
    }
}

.home{
    background-color: lightgray;
    height: 100vh;
```

```
    display: flex;
    align-items: center;
    justify-content: center;

    .container{
        border: 1px solid white;
        border-radius: 10px;
        width: 65%;
        height: 80%;
        display: flex;
        overflow: hidden;
        @include tablet{
            width: 90%;
        }

        .sidebar{
            flex: 1;
            background-color: gray;
            position: relative;
            .navbar{
                display: flex;
                align-items: center;
                background-color: dimgray;
                height: 50px;
                padding: 10px;
                justify-content: space-between;
                color: white;

                .logo{
                    font-weight: bold;
                    @include tablet{
                        display: none;
                    }
                }

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

                    img{
                        background-color: white;
                        height: 24px;
                        width: 24px;
```

```
                    border-radius: 50%;
                    object-fit: cover;
                }

                button {
                    background-color: gray;
                    color: white;
                    font-size: 10px;
                    border: none;
                    cursor: pointer;
                    @include tablet{
                        position: absolute;
                        bottom: 10px;
                    }
                }
            }
        }

        .search{
            border-bottom:  1px solid white;

            .searchForm{
                padding: 10px;
                input{
                    background-color: transparent;
                    border: none;
                    color: white;
                    outline: none;
                    &::placeholder{
                        color: white;
                    }
                }

            }
        }

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

```
            &:hover{
                background-color: dimgray;
            }

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

            .userChatInfo{
                span{
                    font-size: 18px;
                    font-weight: 500;
                }
                p{
                    font-size: 14px;
                    color: lightcyan;
                }
            }
        }

    }
    .chat{
        flex: 2;

        .chatInfo{
            height: 50px;
            background-color: gray;
            display: flex;
            align-items: center;
            justify-content: space-between;
            padding: 10px;
            color: white;
        }

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

            img{
```

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

    .messages{
        background-color: rgb(235, 235, 235);
        padding: 10px;
        height: calc(100% - 160px);
        overflow-y: scroll;

        .message{
            display: flex;
            gap: 20px;
            margin-bottom: 20px;

            .messageInfo{
                display: flex;
                flex-direction: column;
                color: black;
                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: white;
                    padding: 10px 20px;
                    border-radius: 0px 10px 10px 10px;
                    max-width: max-content;
                }
                img{
                    width: 50%;
```

```
                }
            }

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

                .messageContent {
                    align-items: flex-end;

                    p{
                        background-color:rgb(0, 120, 249);
                        color: white;
                        border-radius: 10px 0px 10px 10px;
                    }
                }
            }

        }
    }

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

        input{
            width: 100%;
            border: none;
            outline: none;
            color: black;
            font-size: 18px;

            &::placeholder{
                color: lightgrey;
            }
        }

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

```
                    gap: 10px;

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

                    button{
                        border: none;
                        padding: 10px 15px;
                        color: white;
                        background-color: black;
                        cursor: pointer;
                    }
                }
            }
        }
    }/*vv*/
}
```

תיקיית img: התמונות בפרויקט

AvatarAdd.png -

img.png -