בית הספר הרב תחומי
רשת תיכונים ומכללות מקבוצת עמל

פתח תקוה ב'
ע"ש ליידי דיוויס

פרויקט גמר הנדסאים יד מגמת הנדסת תוכנה

שם מגיש: דורון גולן

תעודת זהות: 322715517

שם מרצה: פרד ג'נין

רכזת מקצוע: טובי סטפ

תאריך הגשה: מאי 2020

תוכן עניינים

מבוא

שם הפרויקט הוא: Talking With Fred's.

הפרויקט הינו פלטפורמת שיחוח (צ'אטים) אינטרנטית מרובת משתתפים. בעת שימוש בפלטפורמה כל המשתמש יוצר לעצמו פרופיל, ויש בידיו את היכולת להתחיל שיחות אישיות עם כל אחד מבעלי הפרופיל במערכת וליצור קבוצות שיח מרובות משתתפים.

כמובן, שיש ביד המשתמש גם את היכולת לעדכן את פרטי הפרופיל שלו עצמו.

הפרויקט שלי מחולק לשני חלקים עיקריים:

- הממשק למשתמש (UI), ה-Front End.
- ה-API – ה-Back End.

<u>Front End</u>:

חלק זה של הפרויקט הינו האתר האינטרנטי עצמו, אשר אליו נחשף המשתמש. תפקידו של צד זה להיות נגיש ונוח לשימוש, להציג את כל הפרטים הרלוונטיים, וכמובן, להיות מעוצב בצורה מושכת ונעימה לעין.

לצורך חלק זה בחרתי להשתמש בספריית React.

<u>Back End</u>:

חלק זה של הפרויקט הינו החלק המנהל את המערכת מאחורי הקלעים. חלק זה תפקידו למלא את בקשות ה-Front End ולנהל את מסד הנתונים.

לצורך חלק זה בחרתי להשתמש בסביבת עבודה Node.js עם ספריית Express.js.

סביבות העבודה

Front End

השפה המרכזית שבה פיתחתי את הפרויקט היא TypeScript.

TypeScript היא שפה המבוססת ומתקמפלת לשפת JavaScript, אשר היא בין השפות המרכזיות המובילות בפיתוח אינטרנט. השיפור הקל של שפת TypeScript על פני שפת JavaScript היא ששפה זו מוסיפה אופציה להגדרת סוגי משתנים. תוספת זו מאפשרת ייעוד של משתנים ופעולות לסוגי משתנים שונים מה שמאפשר דרך לתיאור מבנה האובייקט, מספר תיעוד טוב יותר ומאפשר לקומפיילר לוודא שהקוד שלנו ירוץ בצורה נכונה וטובה.

ספריות רבות דורשות התאמה קטנה לשם עבודה טובה מול TypeScript, ולכן ספריות רבות יבקשו מאיתנו להוריד ספרייה נוספת אשר מתחילה ב - @types/ ולאחריו שם הספרייה.

הספרייה המרכזית שהשתמשתי בה לצורך פיתוח ה-Front End היא React.js.

React הינה ספריית פיתוח אפליקציות אינטרנט לאפליקציות SPA (Single Page Application). אפליקציות SPA הינן הסוג החדיש יותר של אפליקציות אינטרנטיות. לפני השימוש בהן נעשה שימוש באפליקציות מסוג SSR (Server Side Rendering). אפליקציות מסוג SSR הן אפליקציות אשר מפותחות בסביבות של PHP ו-NET.

בראשית פיתוח האינטרנט, מחשבי הלקוחות (PC) לא היו חזקים כמו השרתים שהחזיקו את אתרי האינטרנט. לכן על מנת להקל על הלקוחות ולהגדיל את מהירות עליית האתר, ננהגה שיטת ה SSR בה דף הHTML נבנה בשרת, ונשלח כמקשה אחת ללקוח על מנת שיצייר אותו על המסך. דבר זה דרש, כמתבקש, שרתים עם כוח מחשוב רב.

עם התפתחות הטכנולוגיה, המחשבים האישיים נהיו חזקים יותר ויותר, וכוח מחשוב חזק יותר, והספקים לקחו החלטה עסקית לניצול כוח המחשוב של הלקוחות והיכולת להפוך את השרתים שלהם לשרתי אחסון שאינם צריכים כוח מחשוב כלל לחיסכון כלכלי. זו בעצם התפתחות הSPA.

בשיטה זו, האתר נטען טעינה ראשונית חלקית בצד הלקוח, וכעבור מספר רגעים קצרים מתחיל שאר האתר להצטייר על המסך, כל פעם חלק אחר. אפליקציות אלא משתמשות בסוג מיוחד של בקשות אינטרנטיות (Requests) שנקראות AJAX (Asynchronous JavaScript and Xml). מטרתן של בקשות אלו הן לייבא HTML של חלק מסוים מאוד באתר ובכך, בכל פעם שקורה עדכון כלשהו בתצוגת האתר אין צורך לבקש ולצייר את **כל** דף ה- HTML, אלא שמתבצעים הבקשה והציור מחדש של חלק מסוים בדף המוצר וכך מהירות האתר גבוהה יותר, חווית המשתמש אינה נפגעת מטעינה מחודשת (היות וקיים רצף תצוגתי כללי ולא טעינה כללית מחודשת).

לשתי השיטות יש את היתרונות והחסרונות שלהן. על מנת לקרב את React לעולם הSSR, ניתן להשתמש בספריית Next.js אשר מתממשקת עם React והופכת את בסיס העבודה איתה לSSR ולא SPA.

עיקרון נוסף שעבדתי על פיו הוא עיקרון "מקור אחד של אמת" (Single Source of Truth). עיקרון זה אומר כי יש מקור אחד של משתנים, וכל שאר המחלקות והקומפוננטות (Components) יסתכלו על אותו סט אחד של משתנים, ישנו את ערכיו בהתאם ויגיבו לשינוי בו בהתאם. בסביבות העבודה בהן השתמשתי, עיקרון זה בא לידי ביטוי על ידי ניהול מצבים (State Management). אחת התכונות של React היא שימוש בסוג מיוחד של מאפיין בכל קומפוננטה, אשר נקרא State. השימוש המיוחד במאפיין הState של קומפוננטה, הוא שכאשר חל שינוי כלשהו במאפיין זה, הקומפוננטה בצורה אוטומטית ואוטונומית מבצעת ציור מחדש של הקומפוננטה על המסך (Re-Rendering).

על מנת לשמור על עיקרון זה השתמשתי בספריית Mobx.js.

ספריית Mobx.js היא ספרייה "מנוסת קרבות" אשר מפשטת את כל ההתעסקות בניהול מצבים. הפילוסופיה מאחורי הספרייה פשוטה מאוד : כל מה שאפשר להוציא מהקומפוננטות ולשים במקום אחד, נוציא. מנקודת מבט אישית, כל מה שמצאתי משותף לכלל הקומפוננטות, ובכלל, מה שהשימוש שלי בוא נעשה ברמת הפרויקט הגדול ולא ברמת קומפוננטה בודדת, יצא מהרמת הקומפוננטה ועלה ברמה.

החיבור של Mobx.js ו-React.js הוא חיבור חזק במיוחד מכיוון Mobx.js מאחד משתנים שמשפיעים על מספר רב של קומפוננטות, ו- React.js מצייר מחדש את הקומפוננטה כאשר משהו משתנה בState, וככה כאשר יש שוני במשתנה באחד הStore, יש מספר קומפוננטות שיגיבו לשינוי, וכל זה במינימום מאמץ בעזרת היכולות הגבוהות של React.js.

הדרך הפרקטית שבה היישמתי את העיקרון והשימוש ב-Mobx.js היא יישום ושימוש במה שנקרא **Stores**. הרעיון מאחורי ה-Store הוא שתפקידו של ה-Store הוא להיות המודול ששומר ומטפל במשתנים שקשורים ומנוהלים בכל רחבי הפרויקט. בעזרת Mobx.js, אני יכול לטפל במשתנים רק תחת האחריות של ה-Store, ועדיין לנהל ולהגיב לשינויים שקורים במשתנים הנמצאים באחריותו.

Mobx.js מספקת לי כמה עקרונות בסיסיים שעוזרים לי לנהל את כל העסק. בעזרת הגדרת משתנה כ-**Observable**, אני יכול להגדיר משתנה לכך שכאשר משהו בערכו משתנה, "תידלק נורה" אשר תדווח כי היה שינוי בערכו של המשתנה. המשתנה הנ"ל יכול להיות משתנה רגיל, אובייקט כלשהו וכו'. נוסף לכך, ישנה הגדרת ה-**Observer**. הגדרה זו מוגדרת ברמת המודול, ובעצם אומרת למודול "שים לב, יש בך שימוש במשתנה כלשהו מסוג observable, ועליך להיות קשוב ל"הידלקות הנורה" של אותו משתנה וכאשר ישנה "הידלקות" שכזו יש להגיב בהתאם.". הגדרה נוספת היא הגדרת **Computed**. הגדרה זו מעל פונקציה תגרום לתוכן הפעולה לקרות באופן אוטומטי כאשר ישנו שינוי כלשהו ב-observable המשומש בתוכה. שימוש בפעולה מסוג זה מחוץ ל-Store נראה כשימוש במאפיין רגיל של האובייקט. יחד עם הגדרה זו, אשר מתעדכנת לאחר שמתבצע עדכון כלשהו ב-Observable, ישנה הגדרת **Action.** הגדרה זו מציינת כי הפעולה עלולה לבצע שינוי כלשהו ב-observable ומעדכנת את Mobx.js כי היא צריכה לעדכן את כל מודולי ה-Observer ולבצע מחדש את פעולות ה-computed.

מעבר להגדרות אלו, ישנה האופציה לייעד פעולה מסוימת לתגובה לשינוי ב-observable. הגדרה זו נקראת **Reaction**. בעזרת הגדרה זו ניתן להקשיב לשינויים הקורים ב-observable של store אחד, מתוך store אחר, ולהגיב לשינויים אלו בצורות שונות. אופציה זו חזקה מאוד כיוון שניתן לשרשר שינויי observables בין stores שונים ובכך לדאוג לסוג של ספק אוטומציה ספק ניהול אירועים במערכת ה-stores של האתר ולהביא לניהול UI נוח ואחיד, ופעילות אתר דינמית מאוד.

Back End

הסביבה המרכזית בה השתמשתי לחלק זה של הפרויקט היא Express.js Frame Work + Node.js.

Node.js הינו מנוע אסינכרוני מבוסס JavaScript אשר מונע על ידי אירועים (Events). מנוע זה תוכנן על מנת לספק שירות למספר רב של חיבורים ללא מאמץ רב, ולעשות זאת בצורה מקבילית. על כל חיבור, מתבצעת פעולת callback אבל אם אין פעולה ממשית, אין עבודה, המנוע "ישן".

Express.js הינה סביבת עבודה מינימליסטית וגמישה המספקת סט יכולות חזק מאוד לאפליקציות רשת ומובייל. סביבת עבודה זו משתמשת במספר גדול של יכולות ושיטות שירות הHTTP ובנוסף יכולת ה-middleware שניתן להוסיף בצורה פשוטה וחופשית מה שגורם לה ליצור API מהיר, קל וחזק מאוד.

לצורך מסד הנתונים בפרויקט בחרתי להשתמש בMongoDB.

MongoDB הוא שירות מסד נתונים שלא בשיטת SQL (Non-SQL). בחרתי להשתמש בשירות זה מכיוון שלהיקף שימוש בפרויקט כמו שלי, אין צורך בהרמת מסד נתונים בשיטת SQL, היות והרמת שירות MongoDB הוא קל ונוח, גמיש, ופשוט מאוד לשימוש. בנוסף, על מנת לממשק את החיבור בין השירות לבין הAPI, השתמשתי בספריית mongoose אשר מספקת חיבור ושימוש נוח וקל אל מול השירות הנ"ל. על ספרייה זו ארחיב בהמשך.

את ה-API שלי כתבתי במודל RESTful API.

REST API הינו מודל אשר משתמש בשיטת הבקשות של HTTP על מנת לבצע פעולות מסוימות. מודל זה פשוט לעומת מודלים אחרים (כגון SOAP), אך הוא יעיל יותר בכך שמשתמש בפחות רוחב פס. אחד היתרונות ושוני מסוים של מודל זה לעומת מתחריו, הוא שמודל זה אינו שומר חיבורים לאחר שסיים לטפל בבקשה. צורה זו מאפשרת חיסכון במקום והפחתת עומס על המערכת ועל חיבור האינטרנט.

על מנת לספק חוויית ותחושת "זמן אמת" לצ'אט, השתמשתי בספרייה בשם Socket.io.
ספרייה זו מאפשרת לי לפתוח חיבור Socket ולשלוח דרך חיבור זה אירועים, ובקבלתם להגדיר כיצד להגיב. חיבור Socket דומה לחיבור צינור מידע בין שתי נקודות. הצינור הזה תמיד מחובר, ותמיד נוגע רק בשתי הנקודות שהוא מחובר אליהן. לא יותר, לא פחות.

בעזרת ספרייה זו פיתחתי רשת אירועים ותגובות בין הAPI והלקוח וככה יכולתי להצית שרשרת פעולות על ידי האזנה לאירועים מסוימים דרך החיבור הנ"ל.

ארכיטקטורת פרויקט

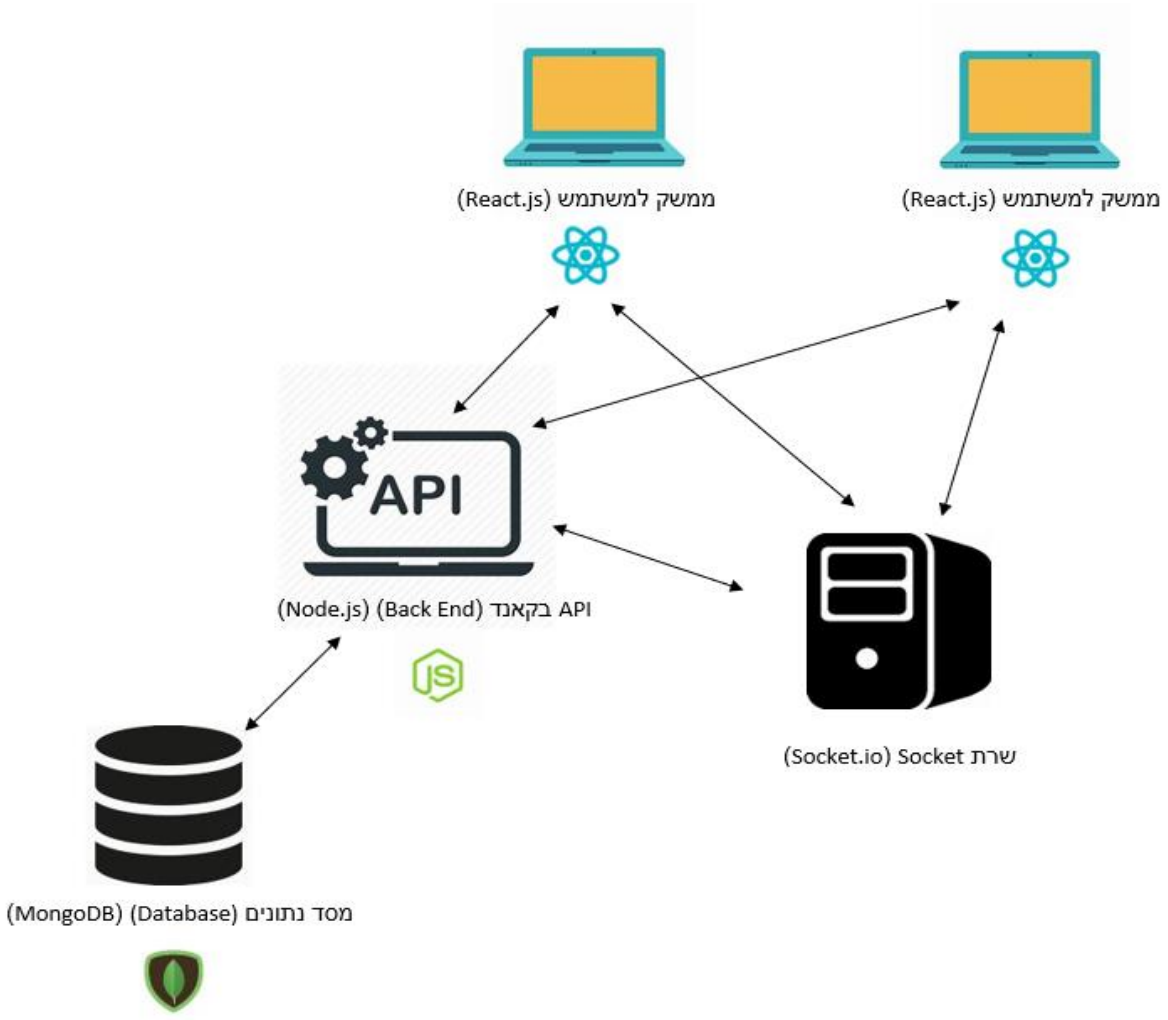
מבנה כללי

המבנה הכללי של המערכת הוא :

שני שרתים, אחד Node.js והשני Socket.io.

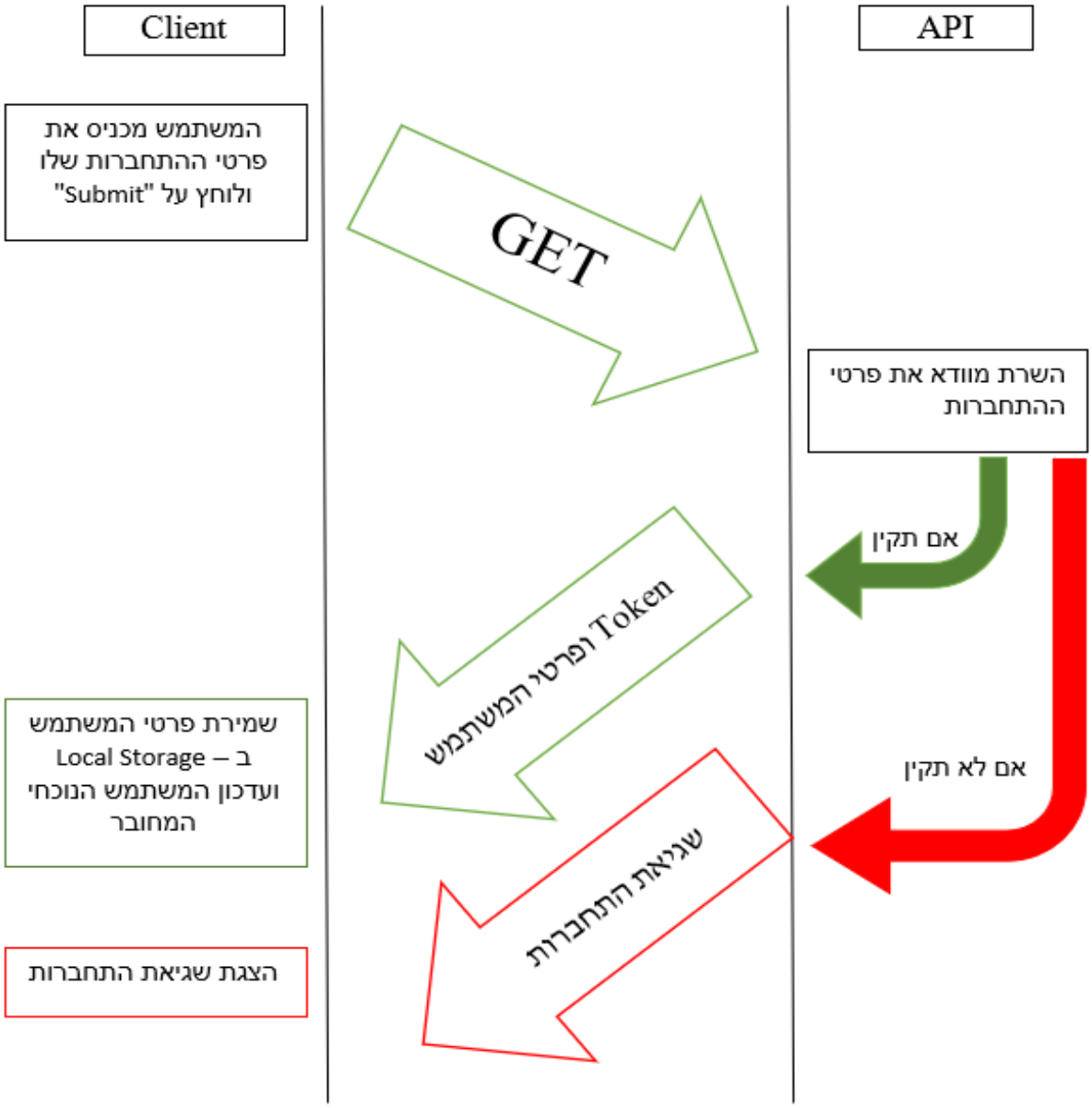
שרת ה-Node.js מחובר למסד הנתונים.

לקוח מתחבר לממשק למשתמש, אשר מחובר גם לNode.js API וגם לשרת Socket.

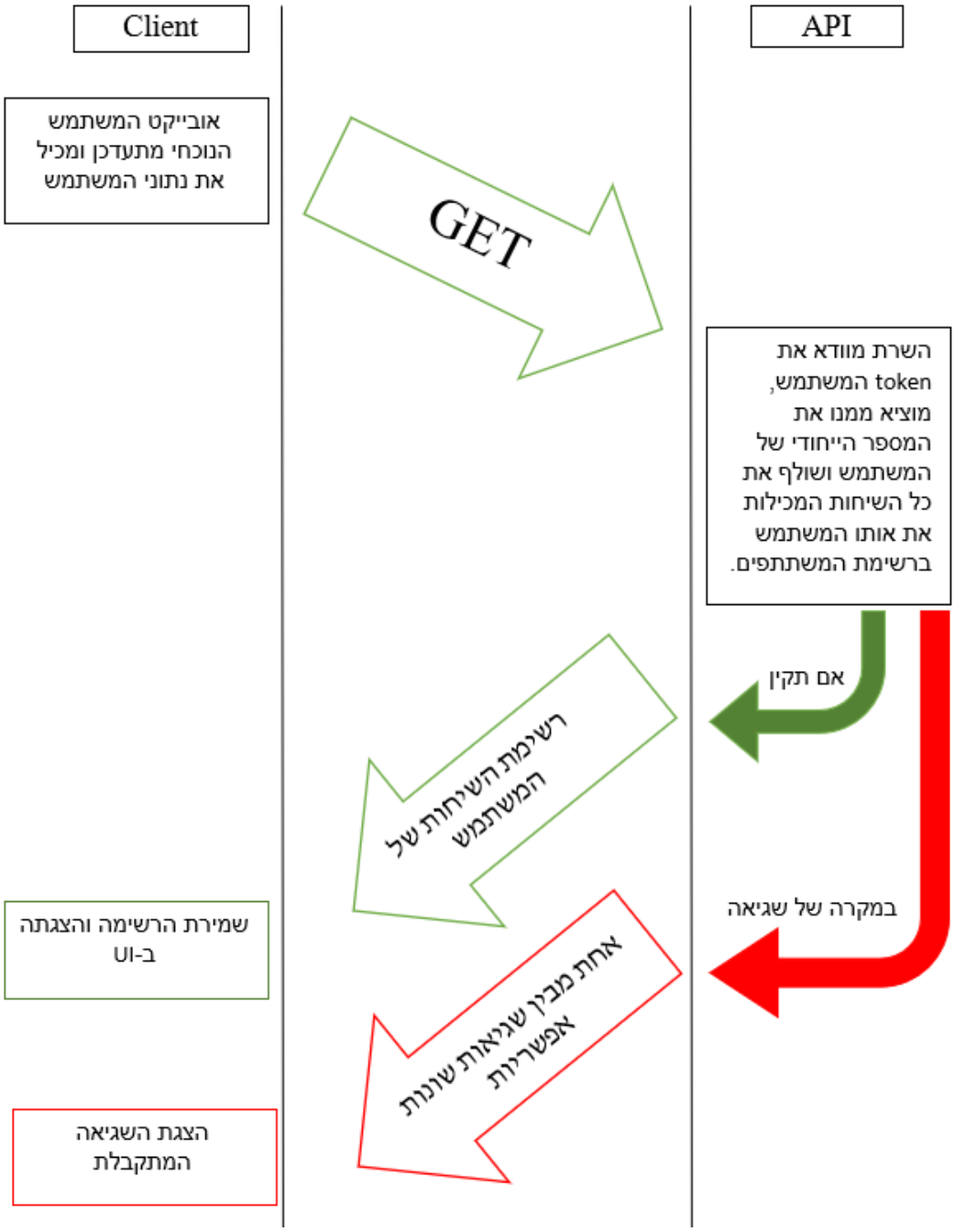


תהליכים מרכזיים בפרויקט

התחברות



הצגת כל שיחות המשתמש



תהליך שליחת הודעה

Client		API	Socket
המשתמש שולח הודעה בשיחה מסוימת	PUT		
שמירה של הודעת המשתמש בStore המתאים והצגתה בצ'אט		יצירת הודעה חדשה ושמירתה במסד הנתונים	
			מציאת כל המשתמשים המהווים חלק מהשיחה בה נשלחה השיחה
			לכל אחד מהמשתמשים, מלבד המשתמש ששלח את ההודעה, שליחת אירוע "הודעה חדשה" עם ההודעה שנשלחה
ברגע קבלת אירוע "הודעה חדשה" **אם ההודעה שייכת לשיחה הנוכחית שפתוחה:** הוספת ההודעה לרשימת ההודעות של השיחה הנוכחית. הקפצת ההודעה בUI בשיחה. **אם לא:** מציאת השיחה המתאימה והצגת הודעה זו כהודעה האחרונה שנשלחה + התראה על שיחה שלא נקראה			

היררכיה עסקית בפרויקט

בזמן פיתוח הפרויקט דאגתי לחלוקה ברורה היררכית של שני החלקים. חלוקה זו נעשתה ממספר סיבות.

ראשית, סדר. כאשר הקבצים מחולקים למחיצות, שמסווגות לתפקידים עסקיים, ומחולקים ומסודרים בהתאמה, ניתן בקלות לגשת לקבצים הנכונים ובמידת הצורך לראות דוגמה בקבצים בעלי תפקיד עסקי דומה.

שנית, סיווג זה לתפקידים עסקיים מהווה תיעוד בפני עצמו, וייעוד מודולים מסוימים לתפקידים מסוימים. כאשר יקרא הפרויקט בדרכו של מתכנת אחר, יהיה לו קל מאוד להבין היכן נמצאים קבצים מסוג מסוים, ומעבר לכך ידע היכן עליו להוסיף קוד במידת הצורך.

מעבר לכך, סיווג היררכי זה מאפשר שמירה על "פרוטוקול" / רצף פעולות לפי סדר ומסוים ותחומי אחריות מסוימים, ובכך יש מניעה לכפל קוד, גישה ממקומות למקומות בצורה לא רצויה ופעולות מסורבלות מאוד.

בשני החלקים ישנה היררכיה, שבחלקה משותפת ובחלקה שונה בתכלית, אבל זאת בהתאם לייעודם ותפקודם של הקבצים בכוח.

Front End

המבנה ההיררכי העסקי בחלק זה הוא:

- **BL – Business Logic** – חלק זה הינו החלק שאחראי, כפי שניתן לראות, על הלוגיקה האמיתית של מודלים כאלה ואחרים. תחת תפקיד עסקי זה ניתן יהיה למצוא את ה- Stores שדובר עליהם כאשר דובר על Mobx.js.

- **Common** – כשמו הוא, מחיצה זו מכילה את המודולים שנשתמש בהם בחלקים רבים ברחבי הפרויקט. תחת תפקיד עסקי זה נוכל למצוא את המודלים (מחלקות שמהוות תבנית לאובייקט מסוים), dtos (מחלקות המהוות תבנית לאובייקטים החוזרים כתשובה מן הAPI ואינם מתאימים למודל קיים) ואת הממירים, האמונים על המרת הDTO המתקבל למודל.

- **Infrastructure** – התשתית - מחיצה זו מיועדת למודולים אשר מהווים חלק בסיסי מאוד במערכת. מודולים אשר אחראיים על פעולות בסיסיות ומתעסקים, בין היתר, בדברים שנמצאים מחוץ לאתר שלנו. בתוך מחיצה זו ניתן יהיה למצוא את ה – Fetchers אשר אמונים על התקשורת הממשית מול הAPI ואת הUtils, אשר אמונים על מספר דברים שונים הכוללים בתוכם את התקשורת וההתנהלות אל מול האחסון המקומי של הדפדפן (Local Storage) ואת מודול זיהוי השפה.

- **Static** – חלק זה אמון על החלקים הסטטיים כגון תמונות שונות, אייקונים וכו'.

- **UI** – חלק זה מוגדר כאחראי על הממשק למשתמש. בתוך המחיצה ישנה חלוקה לקומפוננטות (Components) ולקונטיינרים (Containers). חלוקה זו באה לשם הפרדה בין קומפוננטות "טיפשות" אשר אינן עושות יותר מידי, ואינן מכילות פעולות רבות, ובין הקומפוננטות המורכבות יותר אשר מכילות בתוכן את הקומפוננטות "הטיפשות". בקומפוננטות אלו נבצע פעולות מורכבות יותר, ובמידת הצורך נעביר לקומפוננטות "הטיפשות" פעולות לבצע ברגע המתאים.

Back End

המבנה ההיררכי העסקי בחלק זה הוא:

- **BL – Business Logic** – חלק זה הינו החלק האחראי על תקשורת מול מסד הנתונים, קבלת נתונים מהבקשה, לעבד אותם ולבצע את הבקשה עצמה ולאחר מכן להחזיר את התוצאה.

- **Common** – כשמו הוא, מחיצה זו מכילה את המודולים שנשתמש בהם בחלקים רבים ברחבי הפרויקט. בחלק זה ניתן יהיה למצוא את המודלים אשר מתאים את מבנה הDocument בטבלאות השונות במסד הנתונים.

- **Infrastructure** – התשתית - מחיצה זו מיועדת למודולים אשר מהווים חלק בסיסי מאוד במערכת. תחת תפקיד זה ניתן יהיה למצוא את סיווג הUtils, המכיל מודולים עצמאיים של הצפנה ועבודה מול הtokens.

- **Static** – תחת חלק זה נמצאים הקבצים אותם הגדרנו כציבוריים. כלומר, בעת גלישה לכתובת הAPI, והוספת ה – path הנכון לתמונות או קבצים שונים נוכל לגשת ולראותם בדפדפן.

- **Routes** – תחת חלק זה מסווגים מודולים המטפלים בפניות לכתובות שונות. כלומר, בסקריפט הראשי הגדרנו routes מסוימים אליהם ניתן לפנות ולקבל תשובה. באותו סקריפט אנו גם מנתבים את הroutes השונים למודולים שיטפלו בבקשות בצורה מסודרת, בקשות הקשורות לאותו מודל יטופלו על ידי אותו מודל.

- **Socket** – תחת חלק זה מסווגים החלקים האחראיים על תקשורת הSocket של הAPI.

ספריות נוספות בפרויקט

Front End

Semantic-ui-react
ספרייה זו הינה ספרייה המכילה מספר רב מאוד של קומפוננטות מוכנות מראש ועיצוב מוגדר מראש. דבר זה מקל מאוד על מלאכת העיצוב, ועוזר לשמור על עיצוב דומה בכל האתר. נוסף על כך, העיצובים והקומפוננטות הבאים עם הספרייה הזו נוחים מאוד לעבודה ולכן בחרתי להשתמש בה.

Axios
ספרייה זו הינה ספרייה המספקת HTTP client מבוסס Promises (דרך עבודה בשפת JavaScript אשר מאפשרת עבודה אסינכרונית) בחרתי להשתמש בספרייה זו מכיוון שהיא נותנת לי את האפשרות להגדיר את התקשורת שלי בקלות יתרה, ויותר מכך, ניתנת לי האופציה לנצל כוח גדול הנקרא "Interceptor" – כלומר, לעשות דברים ממש לפני שהבקשה יוצאת וממש רגע אחרי שהתשובה חזרה, לפני שהיא עוברת למי ששלח את הבקשה. נעשה שימוש ביכולת זו על מנת "להזריק" את הtoken של המשתמש לתוך ה – headers, ועל מנת להחזיר לשולח רק את חלק ה – Data של התשובה.

React-router-dom
ספרייה זו מאפשרת לי גישה למאפייני הדפדפן, ויותר מכך למאפייני ה – url וההיסטוריה. דבר זה מאפשר לי להוסיף שימוש ב routes בתוך האתר וככה לנווט בין דפים. בעזרת פעולת ה – withRouter() אשר מעבירה לקומפוננטה שלי מאפיינים מיוחדים וגישה להיסטוריה, אני יכול לבצע את השינוי ומעבר בין דפים וקומפוננטות ללא מאמץ כמעט.

Socket-io.client
ספרייה זו הינה הצד השני של ספריית ה Socket.io אשר השתמשתי בה לשרת הSocket בAPI. היא נותנת לי את היכולת להתחבר, להאזין ולהגיב לאירועים העוברים בSocket.

sweetAlert2
ספרייה זו נותנת לי את היכולת לשפר את הופעת האזהרות הבסיסית של הדפדפן. במקום להשתמש במערכת הalerts הבסיסית וברירת המחדל של הדפדפנים (כשבכל דפדפן זה נראה שונה), אני יכול לפתוח Alert מעוצב, אחיד לכל הדפדפנים, בעיצוב חופשי שלי, נעים לעין ופשוט לתפעול ולתגובה על לחיצת כפתורים. בתוך חלון הAlert שנפתח אני יכול להכניס על תוכן שארצה, טקסט, תמונה, דף HTML שלם – הכל מתאים.

node-sass
ספרייה זו מאפשרת לי להמיר קבצי scss (אשר מאפשרים לי כתיבה נוחה יותר של היררכיה בעיצוב) לקבצי css במהירות ובקלות ובצורה אוטומטית בעזרת middleware.

Back End

Body-parser
ספרייה זו מאפשרת לי לנתח ולקרוא גוף (body) של בקשות HTTP שונות, בפורמטים שונים. אני השתמשתי בפורמט של JSON, היות ופורמט זה נוח מאוד לשימוש בתוך Node.js וReact.

Cors
ספרייה זו מאפשרת לי לשלוט במי יוכל להגיע אל תוך האתר שלי. בעיה גדולה שמתכנתי WEB נתקלים בה היא בעיית cors. ספרייה זו מאפשרת לי את היכולת להחליט על cors בהתאם לרצוני בכל מקום והשימוש פשוט מאוד.

Mongoose
ספרייה זו מאפשרת לי ליצור מודלים לשימוש בMongoDB, ויותר מכך, שימוש במודלים הללו כגישה לטבלאות בהתאמה ובכך להריץ שאילתות לא SQL (non-sql) על הטבלאות.

אבטחת מידע

בפרויקט הנ״ל אבטחת מידע תופסת מקום רב. כאשר מדובר בפלטפורמות צ׳אט בעת המודרנית, במיוחד באינטרנט, כל הנוגע לאבטחת מידע נהיה חלק אינטגרלי מהותי בכל פרויקט, שכן ללא זה למוצר לא יהיו קונים והוא יהיה מסוכן מאוד לשימוש.

על מנת לשמור על אבטחת המידע, החלטתי לפתח את הפרויקט שלי תוך אינטגרציה ושימוש של מספר שיטות ומערכות שונות.

אבטוח סיסמאות

על מנת לבצע בדיקה ואישור לפרטי ההתחברות של המשתמש עליי לשמור את הסיסמא שלו במסד הנתונים שלי. אך, אם אשמור את הסיסמאות בצורה הפשוטה שלהן, ללא כל עיבוד מצד המערכת, כל אחד שיש לו גישה למסד הנתונים יוכל להוציא את כל פרטי במשתמש, כולל סיסמתו, ולהתחבר ללא כל משים או מפריע ולעשות כרצונו.

על מנת למנוע זאת, השתמשתי בשיטת "Salt Hashing". שיטה זו אומרת כי לפני שאני מבצע פעולת Hashing על מחרוזת כלשהי, אוסיף לה עוד מחרוזת, שנקראת salt, ואבצע את פעולת הHashing על מחרוזת החיבור בין המקור וה-Salt.

על מנת לאבטח את הסיסמא ברמה מספקת, השתמשתי באלגוריתם SHA256 להצפנה של הסיסמה, ואת ה-salt ייצאתי כמחרוזת באורך של כ-256 ביט על בסיס 36.

לצורך כך השתמשתי בספרייה בשם ׳csprng׳, אשר עושה חישוב מתמטי מסוים ומפיקה לי מספר רנדומלי, באורך של מספר ביטים לבקשתי ועל פי בסיס לבקשתי. בנוסף, השתמשתי בספרייה מובנת בתוך Node.js אשר נקראת ׳crypto׳, ספרייה אשר מתעסקת הכל מה שנוגע להצפנות. בספרייה זו יש פעולות המתאימות להצפנה, לפיענוח, לביצוע hashing ועוד פעולות מהסוג הזה.

שימוש ב-JSON Web Token

לאחר ההתחברות הראשונית עם שם המשתמש והסיסמא, אני מייצר token המכיל את מידע המשתמש ושולח אותו בצורה מוצפנת בתור תשובה. שימוש בtoken מאפשר להעביר מידע בצורה מוצפנת, אשר ניתן לפענח אותה רק בעזרת מפתח מיוחד מוגדר מראש, מפתח סודי ומפתח ציבורי.

אני בחרתי לשלוח את פרטי המשתמש, רק את שם המשתמש, המספר הייחודי ואת תמונת הפרופיל) ולא להשתמש גם במפתח ציבורי.

את הtoken הזה אני שולח בכל בקשה של המשתמש מהלקוח לAPI, ובצורה כזו אני מוודא בתור שלב ראשון את קיומו ואת תקיפותו של הtoken, עוד לפני שאני מגיע לפעולה המבוקשת. לשם כך הגדרתי middleware שיבדוק את קיומו של ה Token בבקשה ובמידה ואינו קיים או תקף לא ייתן לבקשה להגיע בכלל ליעדה אלא יחזיר ישר תשובת 403 (לא מאושר – Unauthorized). לאחר מכן, במידה וה-token אכן מאומת ותקף, אני יכול להוציא ממנו את המספר הייחודי של המשתמש אשר שלח את הבקשה ובהתאם לבקשה לבצע את הפקודות המתאימות ולהחזיר תשובה.

לצורך זה השתמשתי בספרייה בשם jsonwebtoken, אשר נותנת לי מספר פעולות המאפשרות לי ליצור, לפענח ולוודא tokens בצורה אינטגרלית חלקה ופשוטה.

את הtoken בפרויקט שלי אני הגדרתי עם אמצעי הגנה נוסף – תוקף. לאחר שהכנסתי לtoken שדה של תוקף, שהצבתי על מספר שעות, אני מבטיח לעצמי שלאחר אותו מספר שעות מסוים הtoken כבר אינו תקף ומשום כך לא ניתן להשתמש בו ויש להירשם מחדש. בצורה הזו, במידה ומשתמש שכח את המשתמש שלו פתוח במחשב כלשהו, לא תהיה סכנה שמישהו יוכל להתחבר למשתמש שלו בלא ידיעתו (בהנחה שהפורץ אינו יודע את שם המשתמש והסיסמא של המשתמש.). נוסף לכך, הוספתי טיימר לחוסר פעילות של המשתמש באתר. לאחר מספר דקות ללא פעילות באתר כאשר הוא פתוח, המשתמש מנותק מהאתר ועליו יהיה להתחבר שנית.

~~**הצפנת ההודעות ופיענוחן רק אצל מקבל ההודעה - עוד לא בוצע בפועל**~~

~~**הצפנת תמונות - עוד לא בוצע בפועל**~~

~~**פתיחת חיבור SSL - עוד לא בוצע בפועל**~~

מבט אישי על העבודה ועל תהליך הפיתוח

מבחינתי הפרויקט הזה היה פרויקט מיוחד במינו ממספר סיבות.

הסיבה הראשונה היא שהפרויקט הזה נכתב בתקופת הקורונה. תקופה לא פשוטה, תקופה של חוסר ודאות, תקופה של סגר. היכולת הזו שלי לשבת לכתוב את הפרויקט הזה ולהעסיק את עצמי חלק נכבד מאוד מהזמן עזרה לי, אני מאמין, עם ההתמודדות הכללית שלי עם המצב.

בנוסף לכך, הפרויקט הזה נכתב לאחר 14 שעות לימוד במערכת החינוך, ולאחר 5 שנים בלימודי הנדסת תוכנה. אני הרגשתי שהפרויקט הזה נכתב ברמה גבוהה בהרבה מהרמה שהייתי ניגש בה לפני שנה, שנתיים או יותר.

זאת ועוד, זה פרויקט הWEB הראשון שפיתחתי במסגרת הלימודים ובתור פרויקט ראשון אני מרגיש שיש משהו מיוחד בפרויקט הזה שחשף אותי לדברים חדשים שלא ידעתי לפני.

במהלך הפיתוח נתקלתי במספר אתגרים לא פשוטים.

עיצוב, שאף פעם לא היה חלק חזק אצלי, נכנס מאוד חזק כשמדובר בפיתוח פרויקט WEB. כשאני צריך להתבונן על התוכן של הפרויקט לא רק ברמת הנראות והיופי של הקוד שלי, אלא גם ברמת היופי, הנראות והנגישות על ממשק המשתמש שלי, זה מצריך ממני לחשוב על דברים בצורה שונה, זה מצריך ממני למצוא דרכים יצירתיות אך הנכונות ביותר בנודע לדרך שבה צריך לבצע דברים בצורה פשוטה, נקייה ונוחה.

נוסף על כך, היה גם את תהליך הפיתוח שלא היה פשוט כלל. בדרך כלל כאשר מפתחים פרויקט, במיוחד למסגרת כזו, מתמקדים בסביבה אחת, בפרויקט אחד ויחיד. אך בפרויקט הזה היה עליי לפתח שני חלקים במקביל – הFront End וה-Back end. הייתי צריך ללמוד כיצד לחלק את הזמן, את העבודה וכיצד אני מבצע אינטגרציה בין שני החלקים תוך כדי פיתוח בניסיון לחסוך זמן ובעיות בהמשך.

אתגר נוסף היה, האם אפשר שלא, מחקר רב מאוד על ביצוע פקודות ותהליכים שונים בעולם האינטרנט, בAPI שלי או אפילו בחיבור בניהם. שעות ארוכות של לנסות להתנסח אל מול גוגל בניסיון למצוא את הפתרון שמתקרב לבעיה שיש לי.

אני חושב שמתכנתים כקהילה, במיוחד אלו שמאוד נהנים מתכנות ונכנסים לזה ברמה גבוהה ועמוקה, חולקים מכנה משותף של עמידות בסבל גבוה, מציאת האושר שבהצלחות הקטנות ועקשנות יוצאת דופן. הפרויקט הזה הסב לי רגעים רבים (אפילו רבים מידי) של סבל על בעיות, דברים שלא עובדים, דברים שכמעט ולא ניתן למצוא להם תשובות או את מקור הבעיה. אבל הרבה מתכנתים, וכמוני כמוהם, מתענגים על רגעי ההצלחות הקטנים ביותר, אפילו אלו שאדם מהשורה לא יוכל להבין למה ההצלחה הקטנה הזו מסבה אושר גדול כל כך. הפרויקט הזה גרם לי למצוא את עצמי בהרבה מצבי ייאוש, ויחד עם זה, נקראתי על ידי מצבים רבים של אושר גדול.

אני חושב שהפרויקט הזה עוד יכול ללכת דרך, ואני בטוח שיש לי עוד מקום לשפר אותו ולמצות יותר ויותר ממנו ולהביא אותו כמעט לידי שלמות, אולי אפילו במקומות שאיני מוצא ורואה כעת. מה שבטוח הוא שאני מהפרויקט הזה לוקח איתי הרבה ידע וניסיון. ואף אל פי שהפיתוח הזה היה בדם יזע ודמעות אני יודע שאני יכול לנצל את מה שלמדתי מהפיתוח הזה, הן בפן הטכני והתכנותי והן בפן האישי והתנהלותי שלי, ואני אקח איתי את הרגעים הטובים ואת הרגעים הרעים ואני אשתמש בהם ללמידה ולפיתוח עצמי כדי שאוכל להשתפר ולהוציא מעצמי יותר ומוצרים טובים יותר.

ביבליוגרפיה

קישור לספריית הפרויקט בGITHub

https://github.com/bamba4320/TalkingWithFreds

אתרים נוספים בהם נעזרתי על מנת לחפש פתרונות שונים, עצות או קצות חוטים לבעיות, פתרונות או פיתוחים רצויים.

- אתר הסבר על Express.js:
  http://expressjs.com/

- אתר הסבר על Node.js:
  https://nodejs.org/en/

- אתר הסבר על jwt.io:
  https://jwt.io/

- אתר הסבר על Mobx.js:
  https://mobx.js.org/README.html

- אתר פורומים למפתחים המכיל פורומים על מגוון רחב מאוד של נושאים:
  https://stackoverflow.com/

- אתר הסבר על csprng:
  https://www.npmjs.com/package/csprng

- אתר הסבר על semantic-ui-react:
  https://react.semantic-ui.com/

- אתר הסבר על axios:
  https://github.com/axios/axios

- אתר הסבר על react-router-dom:
  https://www.npmjs.com/package/react-router-dom

- אתר הסבר על crypto:
  https://www.w3schools.com/nodejs/ref_crypto.asp

- אתר הסבר על socket.io:
  https://socket.io/

- אתר הסבר על sweetalert2:
  https://sweetalert2.github.io/

- אתר הסבר על body-parser:
  https://www.npmjs.com/package/body-parser

- אתר הסבר על cors:
https://www.npmjs.com/package/cors

- אתר הסבר על mongoose:
https://mongoosejs.com/