# **React**

מערכת מבית Facebook לבניית אתר – צד Front הבנוי בשיטת SPA והמבוסס קומפוננטות (Components).

**SPA – Single Page Application**

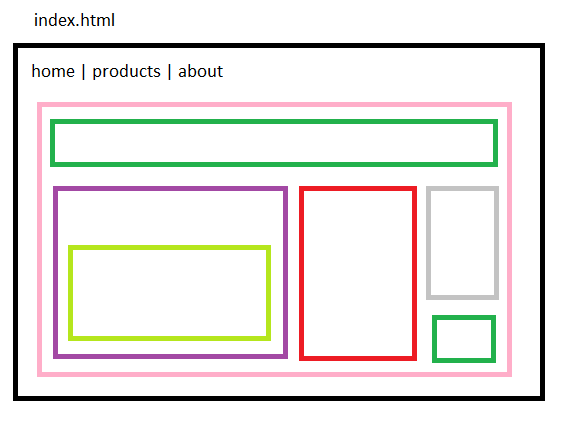
זהו אתר המכיל דף html אחד. אין מספר דפי html בשרת. בכדי להציג דפים שונים, JavaScript "מזריקה" HTML לתוך אזור ספציפי בדף האחד והיחיד.

**Component – קומפוננטה**

יחידת UI נפרדת. מכילה HTML משלה, CSS משלה, קוד JavaScript או TypeScript משלה.

קומפוננטה נבנית ע"י פונקציה (נקרא Functional Component – FC) או ע"י Class (נקרא Class Component – CC).

שם הפונקציה (במקרה של FC) או שם ה-Class (במקרה של CC) הופכים להיות שם של תגית שניתן להציב.



קומפוננטה יכולה להכיל בתוכה קומפוננטות אחרות.

התקנות:

1. התקנת Node.js
2. התקנת React פעם אחת ברמת המחשב:

npm i -g create-react-app

1. יצירת פרויקט חדש

* פתיחת Command Line / Terminal על תיקייה במחשב בה ברצוננו ליצור את הפרויקט.
* הרצת הפקודה:

create-react-app my-project-name --template typescript

לדוגמה:

create-react-app games-store --template typescript

הרצת פרויקט קיים: מתוך Terminal פנימי של VS Code, כתיבת הפקודה

npm start

שחזור תיקיית node\_modules אם לא קיימת:

npm i

**Directory Structure**

זהו מבנה תיקיות וקבצים עבור מערכת מסוימת. לדוגמה עבור בניית פרויקט צד שרת ב-#C יש מוסכמה איך לסדר את מערכת הקבצים והתיקיות של הפרויקט.

React Directory Structure:

project

src

Components

GamesArea

GameList

GameList.css

GameList.tsx

AddGame

AddGame.css

AddGame.tsx

EmployeesArea

EmployeeOfTheMonth

EmployeeOfTheMonth.css

EmployeeOfTheMonth.tsx

Models

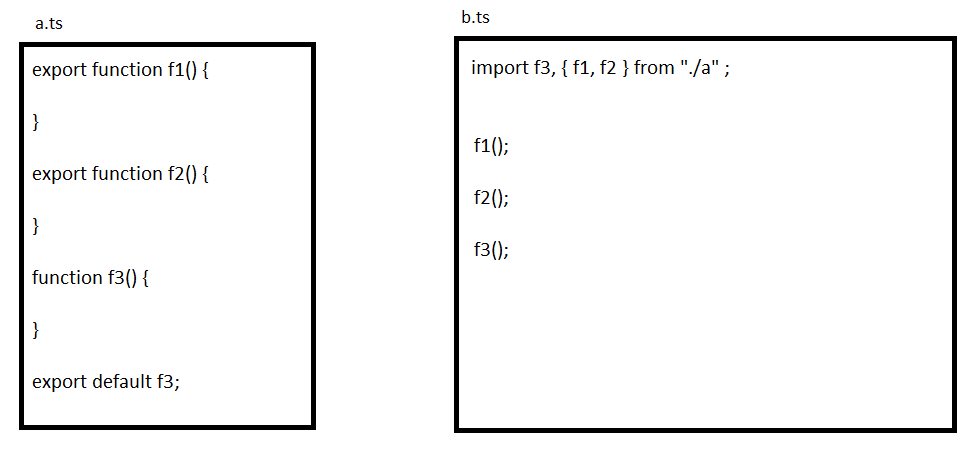
GameModel.ts

EmloyeeModel.ts

Services

GamesService.ts

**TypeScript export / import**



**CSS Leaking – דליפת CSS**

מצב בו אנו מעצבים תגיות של קומפוננטה א' אך העיצוב זולג לתגיות של קומפוננטה ב'.

אסור שזה יקרה!

מניעה – שתי אפשרויות:

1. הגדרת css-class בשם הקומפוננטה בתגית div הראשית, ושימוש ב-class זה בתוך ה-css לפני כל עיצוב תגית.
2. CSS Modules (נלמד בהמשך)

* אסור לתת id לשום דבר ב-React לצורך מניעת שכפול קומפוננטות ולכן שכפול ids.

**חברת Northwind Traders**

חברה אמיתית עם מידע אמיתי העוסקת בייבוא/ייצוא של מוצרי מזון ייחודיים.

המידע שלה כולל מוצרים, קטגוריות, עובדים, ספקים, הזמנות ועוד.

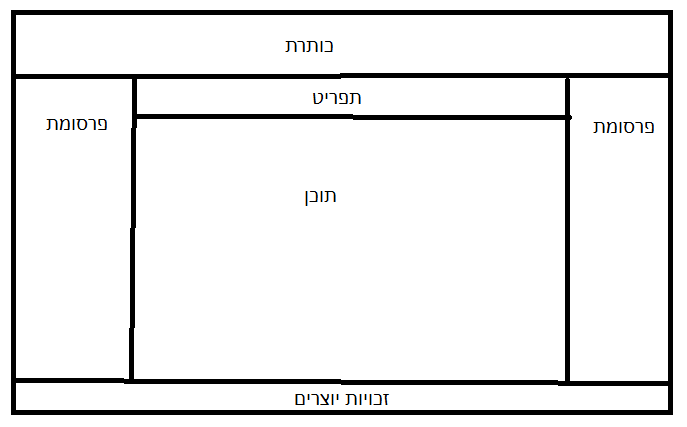
מייקרוסופט קנו את המידע שלה והפיצו אותו בעולם עבור מכללות, אוניברסיטאות וכו' לצורך לימודי.

אנו רוצים לבנות אתר לחברת Northwind ולעלות אותו ל-Cloud.

**Layout**

הסידור הכללי של דפי האתר. איך בגדול האתר בנוי.

לדוגמה:



בריאקט, ה-Layout נבנה ע"י קומפוננטה אחת התופסת את כל הדף (index.html) ובתוכה יש את החלקים השונים ושאר רכיבי האתר.

קומפוננטת ה-Layout מגדירה את הסידור הכללי.

קומפוננטה זו יכולה להיות ה-App, יכולה להיות קומפוננטה המוצבת בתוך ה-App או יכולה להיות קומפוננטה במקום ה-App.

ל-Layout יש גם מבנה וגם עיצוב CSS הקשור ל-Layout.

אך יש גם עיצוב כללי לכל האתר הנמצא ב-index.css.

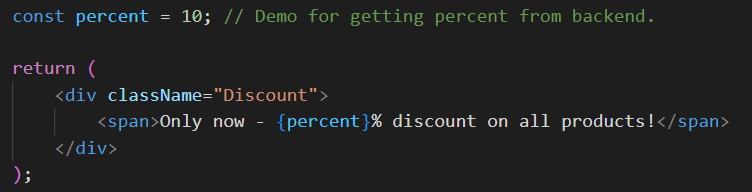
כלי המאפשר ליצור קומפוננטה ריקה בצורה אוטומטית:

npm i -g react-cli-snippets

**Interpolation**

הגדרה מילונית: שיבוץ של קטע או מילה בתוך טקסט.

בריאקט – שיבוץ של Value (משתנה / ערך מוחזר מפונקציה וכדומה) בתוך ה-HTML.



**Render**

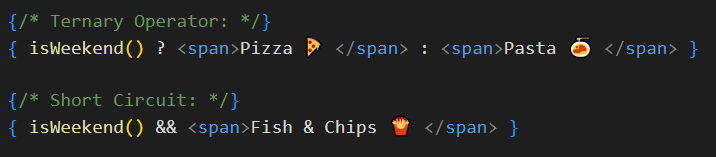
הפעולה של הצגת מידע / רכיב / תוכן כלשהו על הדף. ריאקט מבצעת Render עבור כל קומפוננטה.

**Conditional Rendering**

הצגת תוכן כתלות בתנאי מסוים.

ניתן לבצע במספר דרכים:

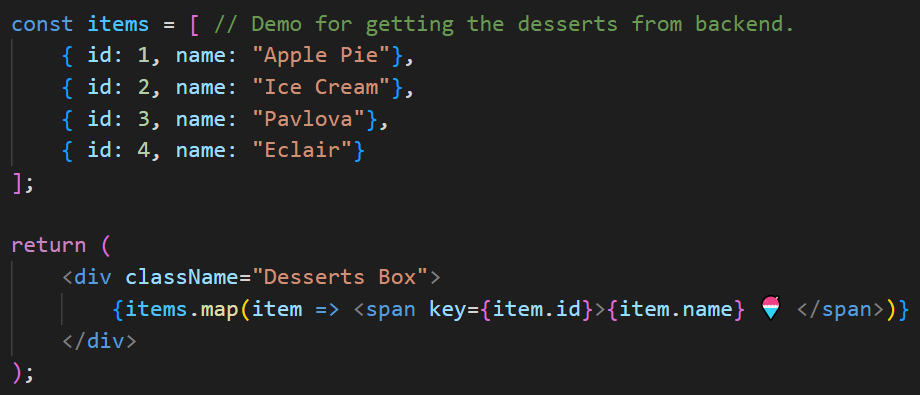
1. Interpolation המכיל בתוכו אופרטור טרינרי (משפט תנאי מקוצר) המציב אחד משני קטעי HTML.
2. Short Circuit (קצר חשמלי) – תנאי, אחריו && או || ואחריו קטע HTML אחד.
3. ביצוע return נוסף כתלות בתנאי, המחזיר HTML אחר.



**Displaying Lists**

קורה לא מעט שברצוננו להציג מידע הנמצא במערך או באוסף אחר כלשהו.

במצב כזה עלינו להמיר כל ערך במערך לקטע HTML ע"י פונקציית map של המערך:



**אירועים**

קוד ה-HTML יכול להעלות אירוע המפעיל פונקציה שאנו כותבים בתוך פונקציית ה-Component.

הפונקציה יכולה להיות ללא ארגומנטים או פונקציה המקבלת ארגומנט אחד מסוג SyntheticEvent.

זהו טיפוס המכיל מידע לגבי האירוע שהתרחש.



**הצגת תמונות**

כל התמונות צריכות להיות בתוך תיקיית Assets/Images (או בתתי תיקיות בתוך Assets/Images)

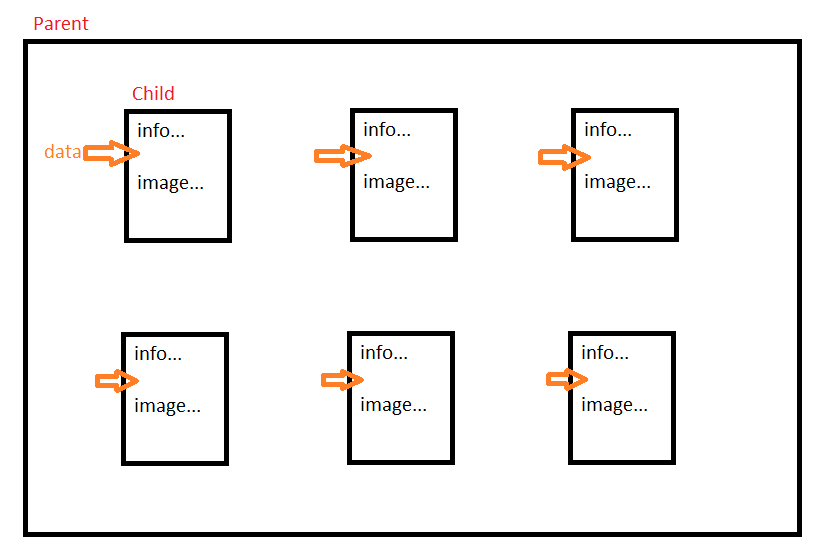
יש לבצע import לתמונה. לאחר מכן משתנה ה-import נכנס למאפיין src של תגית ה-img ע"י Interpolation.



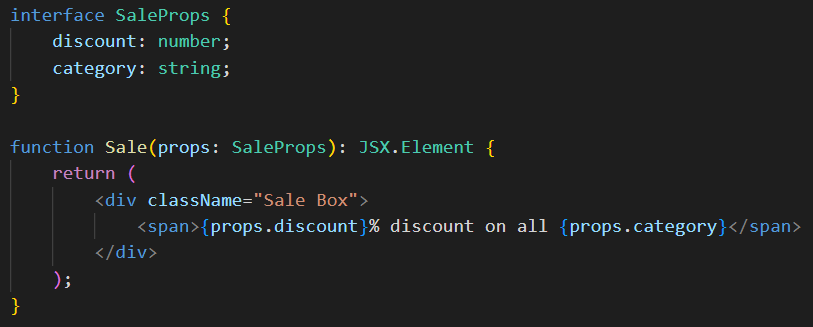
**Props**

זהו אובייקט שה-Child Component מבקש מה-Parent Component שלו.

על ה-Parent Component להביא את המידע הדרוש ל-Child.



ה-Child מגדיר את סוג ה-props שהוא מצפה לקבל:



ה-Parent צריך לשלוח לו את המידע:



**Component State**

State = "מצב" מבחינת מידע.

Component State – המידע השייך לרכיב והמנוהל על ידו.

אם לקומפוננטה יש מידע שמתעדכן לאחר זמן מסוים מהצגת הקומפוננטה, וברצוננו להציג מידע זה, עלינו לשייך את המידע ל-Component State. סתם משתנה שמוגדר בתוך הקומפוננטה אינו חלק מה-State. אם נשנה משתנה שאינו חלק מה-State, ריאקט לא תרנדר מחדש את הקומפוננטה. אם המידע הינו חלק מה-State – ברגע שנשנה אותו – ריאקט תרנדר מחדש את הקומפוננטה בכדי להציג את המידע העדכני.

הנוהל:

1. קריאה ל-useState – פונקציית Generic, כאשר אנו מציינים איזה טיפוס נתונים ברצוננו לנהל. פונקציה זו מחזירה מערך המכיל שני תאים.
2. התא הראשון מחזיר את המשתנה שברצוננו לנהל.
3. התא השני מחזיר פונקציה שכאשר נקרא לה – נוכל לעדכן את המשתנה וריאקט מיידית תציג את השינוי (תרנדר מחדש את הקומפוננטה)

**Hooks**

Hook זו פונקציית מערכת בעלת שם ספציפי (אנו לא ממציאים שמות אלו). אם אני כותב אותה אצלי בקוד היא מופעלת אוטומטית בזמנים מסוימים.

**Lifecycle Hooks**

אלו פונקציות Hooks המתבצעות בזמנים ספציפיים בחיי קומפוננטה מסוג Class Component (כאשר הקוממפוננטה נוצרת, כאשר יש בה שינוי, כאשר יש בה שגיאה, כאשר היא נהרסת וכדומה). קיימים רק ב-CC.

**React Hooks**

אלו פונקציות מיוחדות שניתנות לשימוש אך ורק ב-Functional Component.

ניתנות לשימוש רק ב-Top Level של הפונקציה (כלומר ישירות בתוך הפונקציה).

לא ניתנות לשימוש מחוץ לפונקציה, לא ניתנות לשימוש בתוך פונקציות פנימיות, לא ניתנות לשימוש בתוך תנאים או לולאות או switch או בלוקים וכדומה.

חייבות להתחיל בתחילית useXXX. לדוגמה useState הינה React Hook.

כל פונקציה מבצעת שירות מסוים עבור ה-FC. לדוגמה useState מנהלת State.

**Side-Effect**

זו פעילות שקומפוננטה מבצעת שיוצאת מגבולות הקומפוננטה.

אם קומפוננטה משנה משתנה הקיים בתוכה והוא מוצג, זה לא Side-Effect.

אם קומפוננטה משנה משתנה שנמצא בקומפוננטה אחרת, זה נחשב Side-Effect.

דוגמאות נוספות: גישה ל-Window של הדפדפן, גלישת AJAX ועוד.

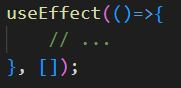
אסור לבצע פעילות Side-Effect ישירות בתוך ה-FC! זאת מפני שאנו לא שולטים על מתי ריאקט תרנדר מחדש את הקומפוננטה ולכן תבצע שוב בצורה מיותרת את ה-Side-Effect.

**useEffect**

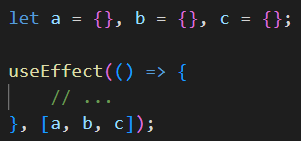
פונקציית React Hook המיועדת לבצע פעילות Side-Effect בתוך FC.

היא יכולה לבצע פעילות כזו בשלושה זמנים:

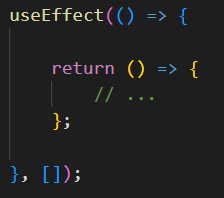
1. פעם אחת כאשר הקומפוננטה נוצרת ומוכנה לשימוש (שליחת מערך ריק):



1. בכל פעם שיש שינוי במשתנים ספציפיים שעלינו להגדיר (שליחת מערך עם משתנים ספציפיים):



1. פעם אחת ברגע שהקומפוננטה נהרסת – הפונקציה המוחזרת תרוץ פעם אחת ברגע שהקומפוננטה נהרסת:



**Routing**

זהו מנגנון המאפשר להציג קומפוננטה כתלות בכתובת הדפדפן.

Route זהו הניתוב הפנימי של האתר. לדוגמה אם בדפדפן כתובה הכתובת: <http://www.mysite.com/news/sports>

ה-Route (או ה-path) יהיו: /news/sports

אנו לא גולשים לדף אחר, אלא רק "מזריקים" קומפוננטה לאזור ספציפי ב-index.html.

בכדי לשלב מנגנון Routing עלינו להתקין ספרייה פנימית בתוך הפרויקט. אלו התקנות npm שמתקינות ספרייה בתוך הפרויקט שלנו. זה מבוצע ללא -g . יש לבצע התקנה כזו ע"י הטרמינל של סביבת הפיתוח המצביע על תיקיית הפרויקט.

ההתקנה הדרושה:

npm i react-router-dom @types/react-router-dom

**REST API**

קיצור של REpresentational State Transfer

נקרא גם RESTful Service זהו שירות שקיים ברשת האינטרנט המחצין מידע. לדוגמה, שירות המחצין מידע של ספרים.

ע"י שירות זה ניתן לבצע את ארבעת הפעולות הנפוצות על מידע הנקראות CRUD (Create Read Update Delete).

המידע כיום הינו בפורמט JSON.

**AJAX**

גלישה ל-Backend לצורך קבלת מידע, ללא ביצוע Refresh של הדף, ללא קבלת דף חדש, ללא גלישה לדף אחר.

ב-React אנו מבצעים גלישות AJAX ל-REST API ע"י ספרייה בשם Axios. התקנה: npm i axios.

לצורך ביצוע גלישת AJAX עלינו:

1. להגדיר את כתובת המידע באזור גלובלי של קונפיגורציה (AppConfig).
2. לבנות מחלקה המתארת את המידע שהשירות מחזיר. מחלקה כזו נקראת מחלקת Model.
3. לבנות מחלקה המבצעת בפועל את התקשורת עם השרת. מחלקה כזו נקראת מחלקת Service.
4. לבנות קומפוננטה המביאה את המידע ע"י ה-Service ומציגה אותו.

* התקנה המכילה REST API של חברת Northwind:

npm i -g northwind-back-end

* הרצה של השירות הזה:

northwind

**AppConfig**

מחלקה המכילה קונפיגורציה כללית בנוגע לאתר שלנו.

מכל מקום באתר אפשר לגשת לקונפיגורציה זו ולהשתמש בה.

**Model**

מחלקה המייצגת מידע טהור שלרוב מוחזר מהשרת. לדוגמה מוצר, לדוגמה לקוח, לדוגמה עובד חברה...

בכדי שניתן יהיה להגדיר משתני מחלקה ללא אתחולם, אפשר להגדיר בקובץ tsconfig.json את ההגדרה הבאה:



**Service**

זו מחלקה לביצוע פעילות לוגית טהורה. אין לה UI. היא לא קומפוננטה. היא אך ורק מבצעת חישובים.

גלישה לשרת מרוחק מבוצע במחלקה כזו כי קוד זה הינו קוד לוגי שיכול להיות בנוי ממספר שורות קוד.

**Route Parameter**

זהו מידע שניתן לשלוח ל-Route.

לדוגמה:

/toys/children/**dolls**

/toys/children/**trucks**

/toys/children/**chess**

**Forms**

מאפשרים לקלוט מידע מהמשתמש.

בריאקט קל לנהל טופס ע"י פונקציית Hook בשם useForm.

התקנה:

npm i react-hook-form

ספרייה זו מכילה מספר מאפיינים מרכזיים:

1. פונקציית register – מאפשרת לקשר בין תיבת קלט בטופס לבין מאפיין ב-Model.
2. פונקציית handleSubmit – מאפשרת לקשר בין אירוע onSubmit של הטופס לבין פונקציה שאנו בונים המקבלת את אובייקט המודל.
3. פונקציית setValue – מאפשרת לאתחל את שדות הטופס בערכים ראשוניים.

וולידציה – אימות קלט המשתמש

אפשר לבצע ע"י מאפייני HTML רגילים. יתרון – פשוט יותר ליישום. חסרון – ה-UI קבוע של המערכת וגם הודעות השגיאה.

אפשר לבצע ע"י ספריית useForm – יתרון – שליטה על ה-UI ועל ההודעות. חסרון – מורכב יותר.

**Redux**

זו ספרייה לניהול State ברמת האפליקציה (בנפרד מ-State ברמת Component).

כל רכיב יכול לפנות לספרייה ולהוסיף/לעדכן/למחוק, כל רכיב יכול לקרוא את המידע שיש בה, ניתן גם להירשם לשינויים. כל שינוי – נקבל עדכון.

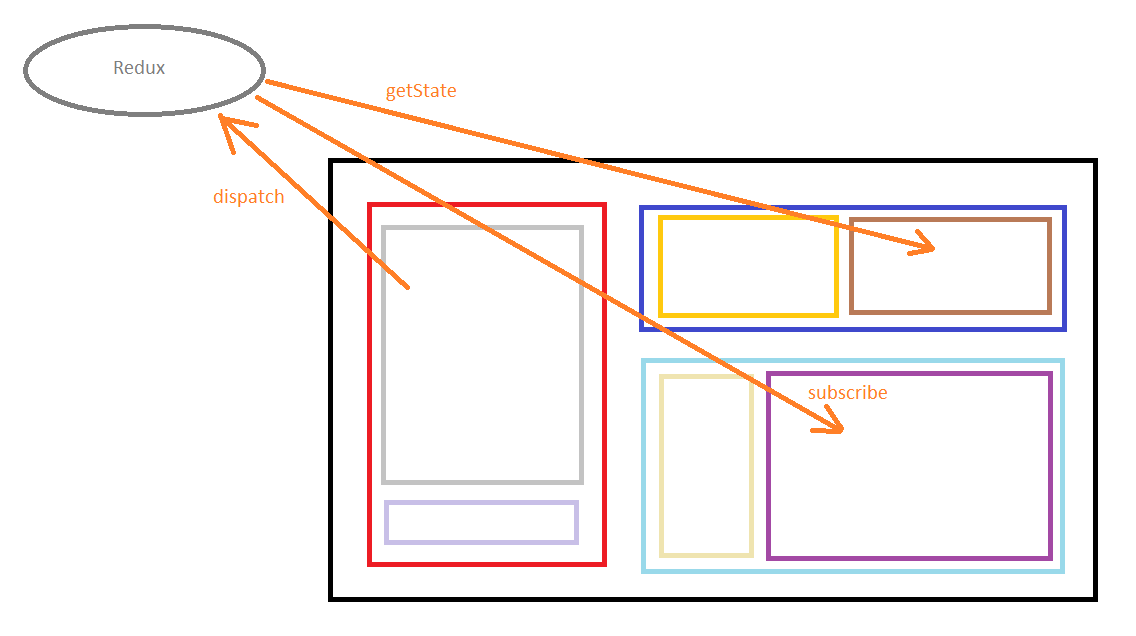
התקנה:

npm i redux

getState – קריאת המידע מה-Global State

dispatch – הוספה/עדכון/מחיקת מידע מה-Global State

subscribe – האזנה לשינויים



**מרכיבי Redux:**

1. AppState – המידע הקיים ברמת האפליקציה. מוגדר בתוך מחלקה. לדוגמה, מערך מוצרים יהיה בתוך מחלקה בשם ProductsState.
2. Action Type – אלו פעולות ניתן לבצע על ה-App State. לדוגמה, עבור מוצרים ניתן להוסיף, לעדכן, למחוק, להביא וכו'. זה נבנה ע"י enum המכיל את הרשימה הזו.
3. Action – זהו interface המתאר אובייקט לביצוע פעולה אחת על ה-AppState. לאוביקט זה יש type – איזו פעולה דרושה, ו-payload – מהו המידע המשויך לפעולה.
4. Reducer – פונקציה המבצעת את הפעולה הדרושה. כאשר אנו מבצעים פעולה על המידע (קריאה לפונקציה dispatch) אנו שולחים אובייקט Action. זה אוטומטית מפעיל את הפונקציה Reducer. לא אנו קוראים לפונקציה ישירות, אלא ספריית Redux קוראת לה כאשר אנו מצבעים dispatch. ספריית Redux שולחת לה שני דברים: (א) את אובייקט ה-AppState הנוכחי שהיא מנהלת, (ב) את אובייקט ה-Action ששלחו לה ב-dispatch. אסור לפונקציה לשנות את אובייקט ה-State שהיא קיבלה. לכן היא חייבת לשכפל את המידע, לבצע את השינוי על המידע המשוכפל ולהחזיר את המידע המשוכפל.
5. Store – האובייקט הראשי של ספריית רידקס המנהל את פעילות הספרייה. דרכו אפשר לבצע (א) getState, (ב) dispatch, (ג) subscribe.

**Auth**

Authentication & Authorization

Authentication – אימות – זיהוי המשתמש. מבוצע ע"י מנגנון הרשמה (Register) ומנגנון כניסה (Login).

Authorization – הרשאות – מה המשתמש יכול לבצע. מבוצע ע"י מנגנון תפקידים (Roles).

**JWT**

Json Web Token

זו שיטת Auth ש-REST API עובדים איתה.

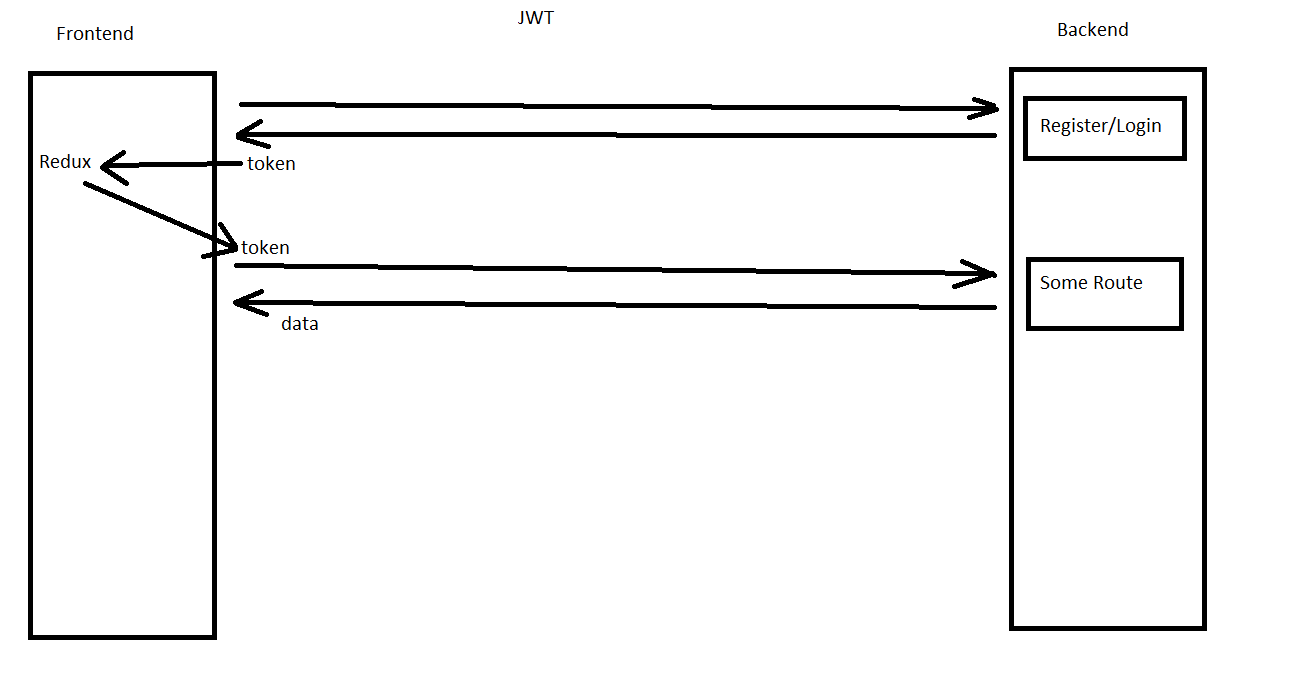
בשיטה זו, כאשר המשתמש מבצע הרשמה או כניסה, ה-Back מנפיק Token למשתמש.

Token זו מחרוזת מקודדת, המכילה בתוכה את אובייקט המשתמש, תאריך תפוגה ועוד דברים.

היא משמשת את ה-Front ואת ה-Back בכדי לדעת מי זה המשתמש ולדעת שהוא ביצע כניסה או הרשמה בהצלחה.

ספרייה שמחלצת את אובייקט ה-user מתוך ה-token:

npm i jwt-decode

****

**Interceptor**

זו פונקציה שמופעלת אוטומטית בכל Request או בכל Response.

פונקציה זו יכולה להתערב ב-Request, להוסיף מידע ל-Request ועוד.

דוגמה לשימוש: שליחת ה-token בכל Request.

**טיפול בהודעות שגיאה**

ראשית, אין להציג למשתמש הודעות מערכת לא ברורות, לדוגמה: Request failed with status code 401

במקום, יש להציג הודעות ברורות, לדוגמה: Incorrect username or password.

הודעות אלו חוזרות מהשרת, אך axios "מסתירה" אותן ועלינו לחלץ אותן מה-Response.

שנית, עדיף לא להציג alert-ים. הם תוקעים את זרימת הקוד, לא ניתנים לעיצוב, לא הכי ברורים...

יש מגוון ספריות להצגת הודעות בצורה אסתטית, לדוגמה npm i notyf

**Material UI (MUI)**

זו ספריית עיצוב המכילה רכיבים מעוצבים מראש.

נוצרה במקור ע"י Google.

קיימת כיום גם כספריית NPM עבור רכיבי React.

אתר רשמי:

<https://mui.com>

התקנה:

npm i @mui/material @emotion/react @emotion/styled

עבור הוספת אייקונים של MUI:

npm i @mui/icons-material