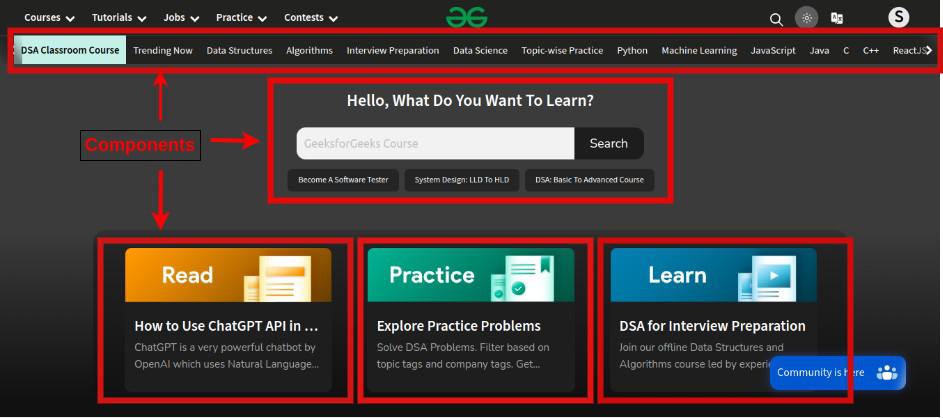
# **React**

ספרייה – Library: קובץ המכיל פקודות, פונקציות, מחלקות וכו'. ניתן להשתמש בו בפרויקט שאנו בונים לפי ראות עינינו. לדוגמה: jQuery זה ספרייה.

Framework: מערכת בתוכה אנו עובדים עם החוקים משלה והיא לוקחת את הקוד שלנו ומשנה אותו באיזשהו אופן, בכדי להביא לתוצר הסופי. לדוגמה: React זה Framework.

Component – רכיב או קומפוננטה: אבן הבניין של ריאקט. קומפוננטה זה יחידת UI נפרדת.



ניתן לבנות קומפוננטה ע"י מחלקה. זה נקרא Class Component או CC. מיושן ולא מומלץ.

ניתן לבנות קומפוננטה ע"י פונקציה. זה נקרא Functional Component או FC. זו השיטה המומלצת.

MPA: Multi-Page Application: זהו אתר אינטרנט הבנוי ממספר דפים. משתמש גולש לדף ע"י תפריט, הדפדפן גולש לשרת ומביא את הדף הדרוש. עקב כך האתר נטען מחדש. כל משתנה שיצרתי בזיכרון – נעלם.

SPA: Single-Page Application: זהו אתר אינטרנט הבנוי מדף index.html אחד. משתמש גולש לדף אחר ע"י תפריט, הדפדפן לא גולש לשרת! אלא משנה את ה-index.html בצורה דינאמית "מזריק HTML דינאמי ל-index.html). אין גלישות לשרת עבור קבלת דפים, אלא רק עבור קבלת מידע טהור.

React: Framework לבניית אתר SPA מבוסס קומפוננטות.

התקנות – מבוצעות ע"י npm:

1. התקנה של Node.js במחשב.
2. התקנת React פעם אחת ברמת המחשב: npm i -g create-react-app
3. יצירת פרויקט חדש: ב-Command-Line או טרמינל חיצוני, יש להגיע לתיקייה בה ברצוננו לבנות את הפרויקט ואז לכתוב:

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

CSS-Leaking: זליגת CSS מקומפוננטה אחת לאחרת.

שתי שיטות למנוע זאת:

1. עטיפת קוד הקומפוננטה ב-Class בעל שם הקומפוננטה, וביצוע עיצוב ב-CSS של הקומפוננטה הנמצא בתוך Class זה.
2. שימוש ב-CSS-Modules (נראה בהמשך מה זה)

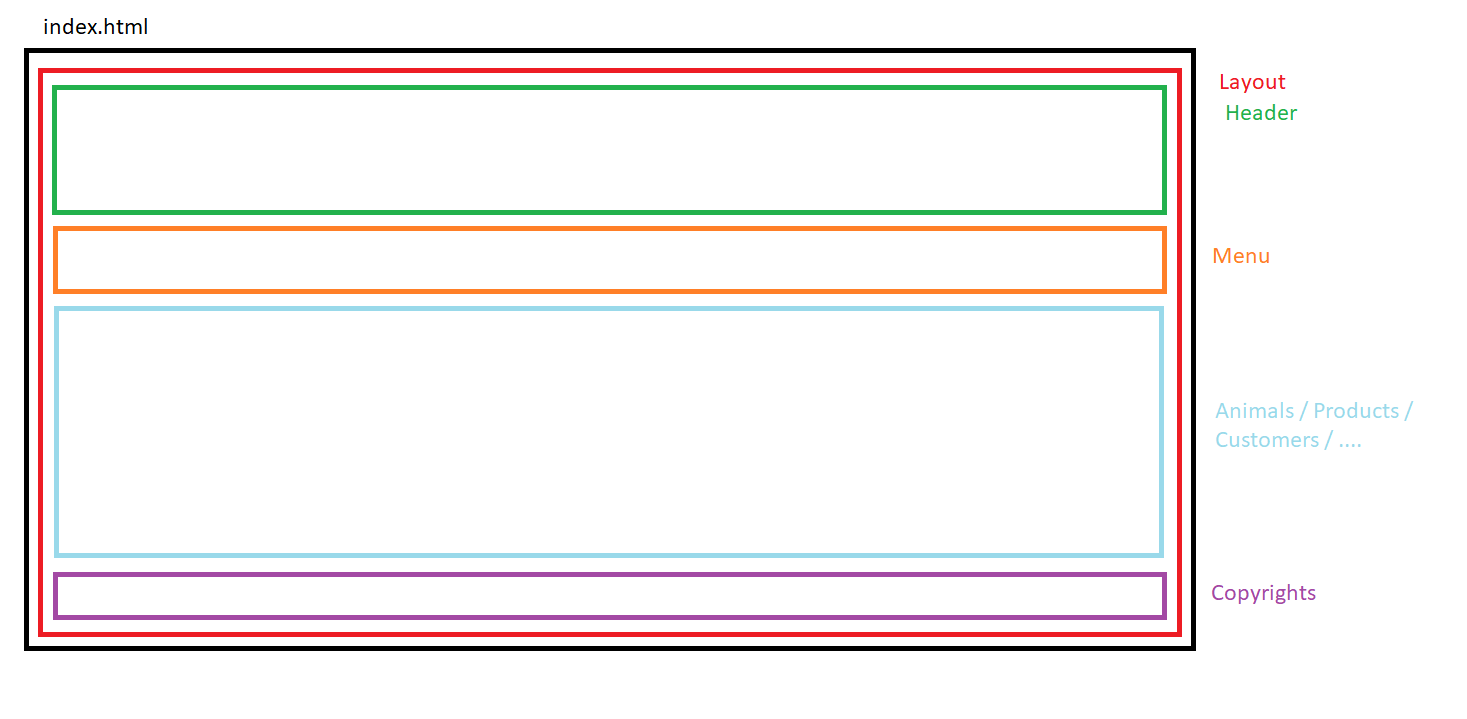
Props: זהו אובייקט ש-Parent Component יכול להעביר ל-Child Component (הוקמפוננטה שהוא מציג).

אובייקט זה נבנה ב-Child, לרוב ע"י type.

Render: הפעולה של תרגום הקוד ל-UI. הצגת ה-UI.

Conditional Rendering: הצגת UI כתלות בתנאי מסוים.

Layout: הסידור הכללי של רכיבי האתר. נבנה ע"י קומפוננטה ייעודית.



Directory Structure: זהו מבנה קבצים ותיקיות בפרויקט. בריאקט מקובל לבצע את המבנה הבא:

ProjectName

src

Assets – תמונות, סרטונים, גופנים

Images

Fonts

Components – כלל הקומפוננטות, מחולקות לאזורים

LayoutArea

Layout

Layout.css

Layout.tsx

Header

Header.css

Header.tsx

Footer

Footer.css

Footer.tsx

AnimalArea

Kitten

Kitten.css

Kitten.tsx

Puppy

Puppy.css

Puppy.tsx

ProductArea

CustomerArea

Models – מחלקות מידע טהור

KittenModel.ts

Services – מחלקות חישוביות או לוגיות

MonetaryService.ts

ApiService.ts

Utils

AppConfig.ts

CLI: Command-Line Interface: קבוצת פקודות עבור ה-Command-Line, לרוב כולן מתחילות באותה המילה, ומותקנות ע"י ספרייה גלובלית.

Snippet: קטע קוד מוכן מראש.

ספרייה ליצירת קומפוננטות (התקנה גלובלית ברמת המחשב): npm i -g react-cli-snippets

Route: ניתוב של הכתובת הפנימית המוצגת בדפדפן לקומפוננטה שברצוננו להציג.

כתובת פנימית:

[http://www.mysite.com**/news/sports/football**](http://www.mysite.com/news/sports/football)

מנגנון Routing נבנה ע"י קומפוננטה שיודעת להציג Component כתלות ב-Route.

דרושה התקנה של ספרייה פנימית בפרויקט של ריאקט (בטרמינל הפנימי של VS CODE):

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

אנו מגדירים איזה Route (נתיב פנימי ב-URL) גורם להצגה של איזו קומפוננטה.

React Hook: זו פונקציה שמכילה מספר חוקים:

1. חייבת להתחיל בתחילית use\_\_\_
2. ניתן לקרוא לה אך ורק ישירות מתוך FC (Functional Component) או מתוך Custom Hooks אחרים.
3. לא ניתן לקרוא לה מתוך תנאים או לולאות.
4. לא ניתן לקרוא לה מתוך פונקציות אחרות שנמצאות בתוך FC.

תפקידה לבצע פעולה מסוימת עבור FC.

State: מצב הזיכרון של משהו. לדוגמה, Component State זה המידע הדרוש עבור קומפוננטה. לדוגמה, Global State זה המידע הדרוש ברמת כלל האפליקציה.

אם קומפוננטה צריכה מידע, אנו יוצרים אותו ע"י משתנה. אך אם נשנה את המשתנה, ריאקט לא תרנדר מחדש את הקומפוננטה.

בכדי שריאקט כן תרנדר קומפוננטה כאשר משנים משתנה, על המשתנה להיווצר ע"י Hook בשם useState.

Two-Way Binding: קישור תיבת טקסט ל-State. שינוי התיבה גורם לשינוי משתנה ה-State. שינוי משתנה ה-State צריך לגרום לשינוי ערך התיבה.

Side-Effect: זהו קוד שקומפוננטה מריצה, שגורם לשינוי או השפעה על משהו מחוץ לגבולות הקומפוננטה. לדוגמה, יצירת טיימר – משפיע על ה-Window כי הטיימר שייך ל-Window ולא לקומפוננטה.

אין לבצע פעילות Side-Effect ישירות בתוך פונקציית הקומפוננטה כי פעילות זו תתבצע שוב כאשר הקומפוננטה מתרנדרת.

אם ברצוננו לבצע פעילות Side-Effect כאשר הקומפוננטה נוצרת, נבצע זאת ב-Hook מיוחד בשם useEffect.

Northwind Traders

חברה פיקטיבית לייצוא/ייבוא מוצרי מזון.

המידע שלה ניתן להשגה ע"י התקנת ספרייה גלובלית:

npm i -g northwind-rest-api

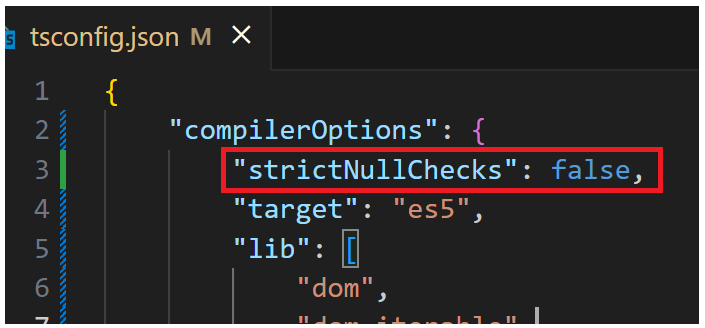
הרצת שרת חברת Northwind:

northwind

גלישה לשרת מרוחק, קבלת מידע והצגתו:

1. שמירת הלינק לשרת המרוחק במחלקה מיוחדת עבור קונפיגורציה ברמת הפרויקט. מחלקה זו נקראת AppConfig ונמצאת בתיקיית Utils.
2. בניית מחלקה המתארת את המידע המוחזר. מחלקה כזו נקראת מחלקת Model.

* בכדי להיפטר משגיאות של חוסר באתחול Properties, נגדיר ב-tsconfig.json את הגדרה הבאה:



1. בניית מחלקת Service לגלישה לשרת המרוחק. במחלקה אנו מבצעים את הגלישה לשרת המרוחק, נהוג ע"י Axios.

התקנת Axios:

npm i axios

1. בניית הקומפוננטה שמפעילה את ה-Service, מקבלת ממנו את המידע ומציגה אותו.

Route Parameter: זהו מידע שניתן לשלוח כחלק מה-Route. ה-Route הוא קבוע, המידע יכול להשתנות. לדוגמה:

[http://mysite.com**/articles/sports/83774**](http://mysite.com/articles/sports/83774)

[http://northwind.com**/product-details/17**](http://northwind.com/product-details/17)

?. Null Chaining Operator

אם קיים אובייקט מצד שמאל – כנס למאפיין או פונקציה שמימין. אם לא – החזר undefined.

לדוגמה const x = person.firstName אם קיים אובייקט ב-person, החזר את ערך ה-firstName. אם לא קיים אובייקט, כלומר person זה null או undefined – החזר undefined, אך מבלי לקרוס.

ניתן לשרשר: const x = person?.address?.city

הצגת הודעות: alert זה "רע". לא אלגנטי, תוקע זרימה, לא ניתן לעיצוב, לא ברור, מכיל מידע מיותר...

עדיף להשתמש בספרייה ייעודית להצגת הודעות קופצות. לדוגמה: npm i notyf

ניהול טופס

טופס מאפשר קבלת מידע מהמשתמש.

ניתן לקבל את המידע מכל תיבה ע"י Two-Way Binding בין תיבה למשתנה. זה דורש אירוע onChange לכל תיבה ומשתנה לכל תיבה. אך זה לא מעט קוד.

לכן יש מגוון ספריות שמאפשרות קליטת מידע מטפסים. הספרייה הפופולרית ביותר נקראת useForm

התקנה:

npm i react-hook-form

שליחת תמונה לשרת

ישנם מספר מאפיינים הקשורים לתמונות:

מחשב הלקוח – צד ה-Frontend:

1. קובץ התמונה הקיים במחשב הלקוח. זהו קובץ פיזי. טיפוס: File. לדוגמה קובץ תמונה של תפוח.
2. שם קובץ התמונה (כולל סיומת) במחשב הלקוח. טיפוס: string. לדוגמה שם קובץ בשם "apple.png".

מחשב השרת – צד ה-Backend:

1. קובץ התמונה שהגיעה לצד השרת לאחר ביצוע ה-Upload. זהו קובץ פיזי. טיפוס: UploadedFile. לדוגמה קובץ תמונה של תפוח שנשלח מהלקוח לשרת והגיע לשרת. הקובץ לא נשמר ב-DB אלא בתיקייה בצד השרת.
2. שם קובץ התמונה המקורי (כולל סיומת). טיפוס: string. לדוגמה apple.png שהיה השם בצד הלקוח והגיע לשרת.
3. שם קובץ חדש שהשרת מייצר עבור התמונה שהינו שם ייחודי. קיים אלגוריתם מיוחד המייצר מחרוזת בעלת 36 תווים שלא חוזרת על עצמה שוב אף פעם. זה נקרא UUID: Universal Unique IDentifier. טיפוס: string.

לדוגמה: xi5ca5fa-75fc-45c2-9cdc-6fccb681dddb.png

ב-DB אנו שומרים שם זה (כולל סיומת). לא את הקובץ.

1. לינק ספציפי לתמונה שנשמרה בצד השרת. טיפוס: string.

לדוגמה: <http://nortwhind.com/products/images/xi5ca5fa-75fc-45c2-9cdc-6fccb681dddb.png>

לינק זה אפשר לחבר לתגית <img src="…" />

HTTP Method

המילה הראשונה המופיעה בטקסט ה-HTTP Request שנשלח לשרת.

מילה זו מתארת את הייעוד הכללי של ה-Request.

להלן HTTP Methods הקיימים עבור REST API:

GET: הייעוד המרכזי של ה-Request הינו קבלת מידע מהשרת.

POST: הייעוד המרכזי של ה-Request הינו הוספת מידע חדש ל-DB בשרת.

PUT: הייעוד המרכזי של ה-Request הינו עדכון מידע מלא – כלל המאפיינים של האובייקט לעדכון נשלחים, בין אם הם שונו או לא שונו.

PATCH: הייעוד המרכזי של ה-Request הינו עדכון מידע חלקי – חלק מהמאפיינים של האובייקט לעדכון נשלחים, בין אם הם שונו או לא שונו.

DELETE: הייעוד המרכזי של ה-Request הינו מחיקת מידע קיים.

דיווח שגיאות:

לעולם אין להציג למשתמש שגיאות מערכת.

יש להציג שגיאה משמעותית.

השרת לרוב מדווח שגיאות וולידציה מדויקות. Axios עוטפת אותן באובייקט המדווח שגיאת מערכת. לכן יש לחלץ מהאובייקט את השגיאה המקורית שהשרת דיווח. ספרייה שמבצעת זאת: npm i error-extractor