# **React**

Framework מבית Facebook לבניית אתר Frontend מבוסס Components הבנוי כ-SPA (Single Page Application).

**Component**

יחידת UI.

מכיל תצוגה (HTML), עיצוב (CSS), לוגיקה (JavaScript או TypeScript).

התקנת React ברמת המכונה:

npm i -g create-react-app

בדיקת גרסה:

create-react-app --version

יצירת פרויקט חדש:

create-react-app northwind --template typescript

הרצת הפרויקט:

npm start

React Directory Structure

src

Assets

Components

HomeArea

Home

Home.css

Home.tsx

Sale

Sale.css

Sale.tsx

ProductsArea

LayoutArea

Layout

Layut.css

Layout.tsx

SuppliersArea

SharedArea

Models

Services

Utils

ספרייה ליצירת קומפוננטות בריאקט:

npm i -g react-cli-snippets

**Interpolation**

שיבוץ Value בתוך קוד ה-HTML.

**Conditional Rendering**

הצגת חלק מה-UI כתלות בתנאי. נהוג לביצוע ע"י Short Circuit, לדוגמה:

{discount > 0 && <span>Only now - {discount}% discount on all products!</span>}

**Props**

זהו אובייקט ש-Parent Component יכול לשלוח ל-Child Component שהוא מציג.

**React Hooks**

Hook זו פונקציה שמופעלת אוטומטית כאשר מתרחש משהו במערכת.

כל React Hook מתחילה ב-use\_\_\_.

חובה לקרוא ל-Hooks רק בתוך FC – ישירות.

אסור לקרוא להם בתנאים או לולאות.

**State**

המידע המנוהל ע"י הקומפוננטה.

המידע חייב להיות שייך לאובייקט הנחשב State בכדי שכאשר הוא ישתנה, ריאקט תבצע Render מחדש לקומפוננטה בכדי להציג את המידע מחדש.

ב-FC משתמשים ב-React Hook בשם useState לצורך ניהול המידע הזה.

**Side-Effect**

זו פעילות ש-Component מבצע אך שיוצאת מגבולות ה-Component. לדוגמה גישה ל-Window. לדוגמה גלישת AJAX לשרת מרוחק.

אסור לקומפוננטה לבצע פעילות Side-Effect ישירות בתוך הפונקציה כי אנו לא שולטים במספר הקריאות לפונקציה ע"י ריאקט.

ב-FC יש לנו את useEffect לצורך ביצוע פעולות Side-Effect שברצוננו לבצע פעם אחת כאשר הקומפוננטה מוכנה לשימוש.

**Routing**

Route זהו הניתוב הפנימי באתר. לדוגמה:

http://mysite.com**/news/sport/football**

http://mysite.com**/**

מנגנון Routing מאפשר להציג Component כתלות ב-Route הנוכחי.

זה מאפשר חוויה של מספר דפים.

ראשית, יש להתקין ספרייה מקומית בפרויקט:

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

**REST**

Representational State Transfer

סוג של Web Service שמאפשר להחצין מידע ולבצע עליו פעולות כמו הוספה, עדכון, מחיקה וכדומה.

REST API Methods:

GET – החזרת מידע

POST – הוספת מידע חדש

PUT – עדכון מידע מלא

PATCH – עדכון מידע חלקי

DELETE – מחיקת מידע

OPTIONS - מידע נלווה, בדיקות בתמיכת השרת...

**גלישה לשרת מרוחק**

התקנת REST API מקומי עבור המידע של חברת Northwind:

npm i -g northwind-back-end

הרצת השרת:

northwind

Axios – ספרייה לגלישה לשרת מרוחק (לרוב ל-REST API):

npm i axios

**Route Parameter**

מידע שניתן לשלוח לנתיב ה-URL ולקבל מידע זה ב-Component.

**Eager Loading**

מצב בו אנו טוענים קומפוננטה בטעינה הראשונית של האתר.

**Lazy Loading**

מצב בו אנו טוענים קומפוננטה רק אם המשתמש גולש ל-Route שמציג אותה.

התקנה לביצוע:

npm i react-loadable @types/react-loadable

**Interceptor**

פונקציה שניתן לכתוב שמופעלת אוטומטית בכל Request או בכל Response.

**טופס קבלת פרטים**

ספרייה לניהול טופס (במקום התעסקות עם Two-Way Binding):

npm i react-hook-form

**HOC**

Higher Order Component

זהו Component המקבל כפרמטר Component אחר, מבצע עליו שינוי כלשהו ומחזיר את ה-Component שקיבל אחרי השינוי.

**Global State**

שמירת מידע ברמת כל האפליקציה. נדרש כאשר המידע משותף למספר קומפוננטות.

ישנן מגוון ספריות המאפשרות לנהל מידע זה.

הפופולרית ביותר הינה Redux.

**Redux**

ספרייה לניהול State גלובלי.

מאפשרת לבצע:

1. שינוי ה-State (הוספה/עדכון/מחיקה).
2. צפייה ב-State.
3. הרשמה לשינויים – כאשר מתרחשים אנו רוצים לדעת מזה.

התקנה:

npm i redux

טרמינולוגיית Redux:

Global State – המידע הקיים ברמת האפליקציה. נקרא גם AppState. זה נבנה ע"י class. לדוגמה ProductsState – מחלקה שמחזיקה את כל מה שקשור למוצרים שברצוננו להעלות לרמת האפליקציה.

ActionType – זו רשימה סגורה של מה אפשר לבצע על ה-AppState שלנו. זה נבנה ע"י enum.

Action – זהו אובייקט המתאר פעולה אחת לביצוע. לכל פעולה לביצוע יש שני מרכיבים: א. מה סוג הפעולה (ActionType). ב. מהו המידע המשויך לפעולה. Action נבנה ע"י interface.

Action Creators – אלו פונקציות שיוצרות אובייקטי Action עבור הפעולות הדרושות.

Reducer – זו פונקציה המבצעת את הפעולות בפועל. אנו רושמים אותה ב-Redux והספרייה קוראת לה. לכן היא חייבת לקבל שני ארגומנטים ולהחזיר ערך אחד. הארגומנטים הם: א. אובייקט ה-State הנוכחי. ב. אובייקט ה-action המתאר את הפעולה לביצוע. על הפונקציה לבצע את הפעולה הדרושה. היא צריכה להחזיר את ה-State העדכני לאחר ביצוע הפעולה. אבל אסור לה לבצע את הפעולה על אובייקט ה-State שנשלח אליה.

Store – זהו האובייקט המנהל את ה-State. דרכו ניתן לבצע שינויים, צפייה במידע, רישום לקבלת אירועים וכדומה.

getState – קבלת ה-State הנוכחי

dispatch – ביצוע שינוי על ה-State

subscribe – האזנה לשינויים.

**react-redux**

ספרייה המכילה כלים שירותים עבור Redux:

npm i react-redux @types/react-redux

**Middleware**

זו פונקציה המתבצעת ברגע שמבצעים dispatch.

מקבלת את המידע הקודם ואת המידע החדש.

יכולה לבטל את ה-dispatch או לאשר אותו.

יכולה לבצע בנוסף דברים אחרים.

**redux-logger**

Middleware המאפשר להציג Log על הקונסול בצבעים. התקנה:

npm i redux-logger

npm i @types/redux-logger

**Redux DevTools**

תוסף המאפשר להתבונן בתוך Redux Global State.

יש להתקין באפליקציה ספרייה ולחבר אותה ל-Store:

npm i redux-devtools-extension

Auth

Authentication & Authorization – אימות והרשאות

**JWT – Json Web Token**

מחרוזת המייצגת התחברות של משתמש הכוללת את אובייקט המשתמש, תאריך תפוגה וכדומה.

שרת מייצר בזמן הרשמה/כניסה מחרוזת כזו ומחזיר אותה ל-Client.

על צד ה-Client לשמור אותה אצלו ולשלוח אותה לשרת בכל פעם שהוא רוצה לקבל שירות הדורש Login.

השליחה צריכה להתבצע ב-Header הבא:

authorization: "Bearer my-token"

ספרייה לחילוץ אובייקט המשתמש הנמצא בתוך ה-token:

npm i jwt-decode