# **React**

התקנת React פעם אחת ברמת המחשב:

npm i -g create-react-app

יצירת פרויקט חדש:

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

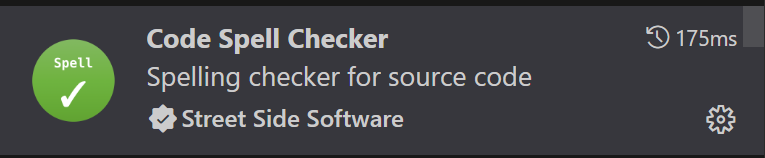
לדוגמה:

create-react-app bookstore --template typescript

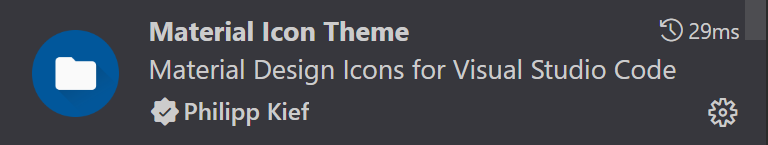
הרצת הפרויקט – בטרמינל הפנימי של VS CODE:

npm start

תוסף מגניב למציאת שגיאות הקלדה:



תוסף מגניב להצגת אייקונים יפים:



Single Page Application (SPA)

אתר המכיל דף index.html בלבד.

כלל ה-UI מוגדר ע"י JavaScript.

הלקוח גולש ומקבל אך ורק את ה-index.html. לאחר מכן ה-UI "מוזרק" לתוכו ע"י JS.

ריאקט עובדת בשיטה זו.

Component

קומפוננטה – רכיב UI אחד ויחיד. קומפוננטה מכילה HTML, CSS ו-JS או TS. היא יכולה גם להכיל קומפוננטות אחרות. ריאקט עובדת בשיטת קומפוננטות.

קומפוננטה מכילה:

1. tsx. קובץ המכיל את קוד ה-TypeScript של הקומפוננטה.
2. css. קובץ המכיל את קוד ה-CSS של הקומפוננטה.
3. test.tsx. קובץ המכיל tests עבור הקומפוננטה.

ניתן לבנות קומפוננטה ע"י Class – נקרא Class Component או CC. דיי יצא מהעולם.

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

Directory Structure

src

Components

BookArea

Book

Book.css

Book.tsx

AddBook

AddBook.css

AddBook.tsx

UserArea

User

User.css

User.tsx

Models

Services

Utils

…

CSS Leaking

זהו מצב בו עיצוב של קומפוננטה אחת משפיע על קומפוננטה אחרת.

יש שני פתרונות לבעיה הזו:

1. שימוש בהיררכיית CSS מדויקת.
2. שימוש ב-CSS Modules.

ספרייה מגניבה שמייצרת קומפוננטות דרך ה-Command-Line:

npm i -g react-cli-snippets

יצירת קומפוננטה:

create fc <AreaName>/<ComponentName>

לדוגמה:

create fc BookArea/StoreDescription

בניית קומפוננטה ע"י CSS Module:

create fc BookArea/StoreDescription --module

Interpolation

הכנסת ערך (משתנה או ערך מוחזר מפונקציה) לתוך ה-HTML

Props

זהו אובייקט שקומפוננטה יכולה לקבל מבחוץ.

Parent Component מכיל Child Component ויכול לשלוח אליו את ערכי אובייקט ה-Props.

Render: הפיכת הרכיב ל-UI המשולב בדף.

Conditional Rendering: בניית UI התלוי בתנאי כלשהו.

הצגת רשימת פריטים: מתבצעת ע"י קריאה ל-map והחזרת HTML מכל פריט.

יש לספק key ייחודי לכל פריט HTML ברשימה הזו – רק בתגית ה-HTML הראשונה של כל פריט.

Real DOM

DOM: Document Object Model: עץ אובייקטים שהדפדפן מייצר מתגיות ה-HTML.

JS יכולה לגשת לאובייקט כזה ולשנות אותו, מיד הדפדפן יציג את השינוי. גישה ל-DOM נחשבת איטית.

Virtual DOM

זהו העתק של ה-DOM האמיתי שריאקט מנהלת. כל שינוי מתבצע ב-Virtual DOM. בסיום סייקל של ריאקט (פרק זמן) – ריאקט מעדכנת את ה-Virtual DOM וזה יותר מהיר.

Local State

זהו המידע המנוהל ע"י קומפוננטה, כך שאם הוא משתנה אנו רוצים להציג את המידע העדכני בקומפוננטה.

לצורך כך על ריאקט לרנדר את הקומפוננטה מחדש.

אם יש לנו משתנה רגיל, הוא לא נחשב Local State. שינוי שלו לא יגרום לרינדור הקומפוננטה מחדש.

Two-Way Binding

זהו קישור בין תיבת קלט למשתנה בקוד.

כל שינוי של ערך בתיבה, המידע נשמר במשתנה.

אם המשתנה מקבל ערך אחר ללא קשר לתיבת הקלט – התיבה צריכה להציג את המידע החדש.

Side-Effect

זהו מצב בו קומפוננטה משפיעה על מידע הנמצא מחוץ לגבולות שלה.

לדוגמה, קומפוננטה משנה עצמאית את ה-DOM.

לדוגמה, קומפוננטה פותחת טיימר (שייך ל-window).

אסור לבצע פעולת Side-Effect ישירות בתוך קוד הקומפוננטה, כי כל פעם שריאקט תרנדר את הקומפוננטה, הקוד יתבצע מחדש! מוצר לבצע Side-Effect בתוך אירועים.

אם ברצוננו לבצע Side-Effect שיתבצע בטעינה הראשונית של הקומפוננטה, עלינו להשתמש ב-useEffect.

Routing

הצגת Component כתלות בכתובת הדפדפן.

בניית Routing דורש התקנת ספרייה פנימית בפרויקט:

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

התקנת REST API של Northwind:

npm i -g northwind-rest-api

הרצת השרת:

northwind

Axios

ספרייה לגלישה ל-REST APIs

התקנה:

npm i axios

React Hooks

אלו פונקציות מיוחדות שניתנות לשימוש אך ורק ישירות בתוך Functional Component.

הן נוצרו בהתחלה לתת מענה לפעילויות שהיו ניתנות לביצוע אך ורק ב-Class Component, לדוגמה ניהול State.

חוקים:

1. חייבות להתחיל בתחילית use\_\_\_
2. אפשר לקרוא להן אך ורק מתוך FC או מתוך Hooks אחרים.
3. אפשר לקרוא להן אך ורק ישירות מה-FC ולא מפונקציות פנימיות.
4. אי אפשר לקרוא להן מתוך בלוקים אחרים כמו תנאים או לולאות.

דוגמאות:

useState

useEffect

useParams

useNavigate