# **React**

**Library**

ספרייה – אוסף של פונקציות/מחלקות/פקודות וכדומה, שניתן להשתמש בה בפרויקט שלנו. היא לא מחליטה איך משתמשים ולא מבצעת שינויים על הקוד שלנו. דוגמה: jQuery, Bootstrap...

**Framework**

מערכת בתוכה אנו עובדים עם הפקודות שלה, ע"י חוקים משלה, והיא כן מבצעת שינויים על הקוד שאנו כותבים.

**Component**

יחידת UI קטנה. מכילה HTML, CSS וקוד כלשהו ב-JS או TS.

ניתן לבצע Reuse לקומפוננטה ולהציב אותה מספר פעמים על הדף או בתוך קומפוננטות אחרות.

ניתן לבנות קומפוננטה ע"י Class (נקרא Class Component או CC) או ע"י פונקציה (נקרא Functional Component או FC). רוב העולם בונה ע"י FC.

**MPA**

Multi Page Application

אתר הבנוי ממספר דפים. בהצגת דף, הדפדפן גולש לשרת להביא את הדף.

**SPA**

Single Page Application

אתר המכיל דף HTML אחד ויחיד והוא ריק מתוכן. הוא מכיל קישור ל-JS ול-CSS. כל המידע/UI וכדומה, נמצא ב-JS. ה-JS מזריק את ה-UI לאזור הריק של הדף. לרוב הדף נקרא index.html.

**React**

Framework מבית Facebook לפיתוח אתר Frontend מבוסס קומפוננטות הבנוי כ-SPA. ניתן לפתח ריאקט ע"י JavaScript ואפשר ע"י TypeScript.

**TypeScript**

שפת תכנות Strongly-Typed שנוצרה ע"י מייקרוסופט ב-2013.

JavaScript/TypeScript

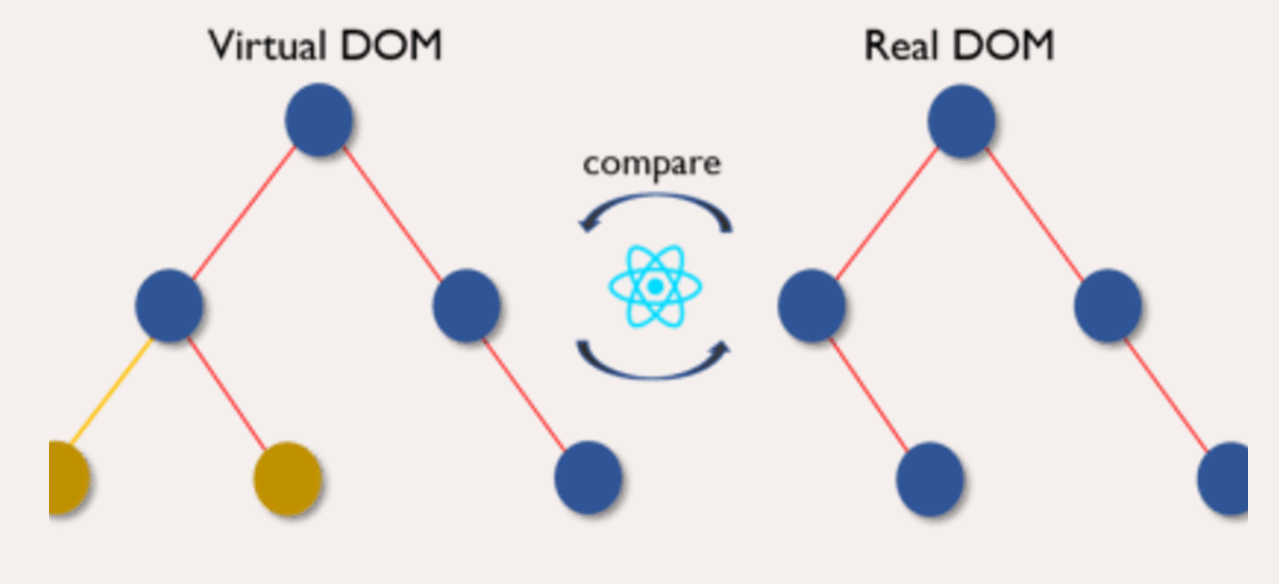
1. ב-JS אין קומפיילר. ב-TS יש קומפיילר. הקומפיילר מגלה שגיאות קומפילציה ואם אין – הופך את הקוד ל-JavaScript. זה נקרא Transpiler.
2. ב-JS אין Strongly Types. TypeScript היא Strongly Typed.
3. ב-JS ה-IntelliSense חצי כח. ב-TS יש IntelliSense מלא.
4. ב-JS ה-OOP חצי כח. ב-TS יש הכל.
5. ועוד...

**Real DOM**

זהו ה-DOM הרגיל שהדפדפן מנהל.

**Virtual DOM**

זהו העתק של ה-Real DOM שריאקט מנהלת. כל שינוי שאנו מבצעים בקומפוננטה נכתב ל-Virtual Dom ולא ל-Real DOM. בסיום סייקל (זמן מסוים), ריאקט משווה בין ה-Real DOM ל-Virtual DOM ויודעת לעדכן את ה-Real DOM בצורה יעילה.



**התקנות**

1. פרויקט React בשלב הפיתוח, מבוסס על מערכת Node.js. לכן התקנה ראשונה הינה Node.js.

<https://nodejs.org/en>

1. התקנת React פעם אחת ברמת המכונה:

פתיחת Command-Line (בג'ון ברייס וב-MAC חובה לפתוח כאדמין) וכתיבת הפקודה:

npm i -g create-react-app

npm: Node Package Manager – ה-Package Manager של Node.js

Package Manager – שרת התקנות. ב-.net יש את NuGet.

i: install

-g: Global – התקנה ברמת המכונה ולא ברמת פרויקט ספציפי.

create-react-app: ה-Framework שאנו מתקינים.

1. יצירת פרויקט חדש:

* פתיחת Command-Line בתיקייה ספציפית.

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

* פתיחת התיקייה שנוצרה ע"י VS Code
* הרצת הפרויקט: בטרמינל הפנימי של VS Code: npm start

**Directory Structure**

src

Components

ProductsArea

ProductList

ProductList.css

ProductList.tsx

AddProduct

AddProduct.css

AddProduct.tsx

SuppliersArea

PersonalSupplier

PersonalSupplier.css

PersonalSupplier.tsx

Models

ProductModel.ts

SupplierModel.ts

Services

ProductsService.ts

SuppliersService.ts

Utils

AppConfig.ts

// …

**CSS Leaking**

CSS בכל קומפוננטה תמיד יכול להשפיע על כל קומפוננטה אחרת. אם זה קורה זה נחשב "זליגת CSS".

**Interpolation**

הכנסת ערך המוגדר ב-TypeScript לתוך ה-HTML. מבוצע ע"י {} ובתוכם הערך.

**Conditional Rendering**

זהו מצב בו אנו מציגים UI כלשהו כתלות בתנאי.

שיטה ראשונה ע"י אופרטור טרינארי:

{ price <= 100 ? <span>🎲</span> : <span>🎮</span> }

שיטה שנייה ע"י Short Circuit:

{ price > 100 && <span>💳</span> }

**התקנה גלובלית עבור ספרייה ליצירת קומפוננטות:**

npm i -g react-cli-snippets

יצירת קומפוננטה – בטרמינל של VS CODE:

create fc AreaName/ComponentName

**הצגת Lists**

מתבצע ע"י map על משתנה המערך והחזרת HTML מכל פריט.

חובה לתת key ייחודי (ברמת הרשימה) לכל פריט המוצג ב-List עבור ניהול נכון של הפריטים ב-Virtual DOM.

**React Hook**

אלו פונקציות שמספקות יכולת מסוימת עבור Functional Component. חוקי React Hook:

1. מתחילות בתחילית useXXX
2. יכולות להיקרא אך ורק בתוך Functional Component או בתוך Custom Hooks (שאנו בונים).

לא ישירות ב-Script, לא בתוך Class Component.

1. יכולות להיקרא אך ורק ישירות בתוך FC. לא בתוך פונקציות פנימיות, לא בתוך תנאים או לולאות.

**State**

מידע ברמת הקומפוננטה שכאשר אנו משנים אותו, ריאקט גם מרנדרת מחדש את הקומפוננטה בכדי להציג את המידע העדכני.

הביצוע הינו ע"י useState

    const arr = useState<string>(""); // "" is the initial value.

    const gameToDisplay = arr[0]; // Value to display

    const setGameToDisplay = arr[1]; // Function for changing the value + rerender the component

או ע"י שימוש ב-Destructuring Assignment:

    const [gameToDisplay, setGameToDisplay] = useState<string>("");

Side-Effect

זהו מצב בו קומפוננטה מבצעת פעולה שיוצאת מהגבולות שלה וגורמת לשינויים כלשהם מיחוצה לה.

לדוגמה, גלישת AJAX לשרת, גישה ל-DOM ישירות ועוד.

אין לבצע Side-Effect ישירות בתוך פונקציית הקומפוננטה כי אנו לא שולטים על מתי ריאקט בוחרת לרנדר את הקומפוננטה.

useEffect

Hook מיוחד שיכול להריץ קוד בזמנים ספציפיים:

1. בהעלאת הקומפוננטה – פעם אחת.

useEffect(()=>{}, []);

1. בהעלאת הקומפוננטה + בכל שינוי של Dependency: a או b או c

useEffect(()=>{}, [a, b, c]);

1. בהריסת הקומפוננטה.

**מנגנון Routing**

Route זהו ניתוב בין כתובת ה-URL לבין קומפוננטה כך שאם ישנה כתובת URL סצפיפית, המנגון יציג קומפוננטה ספציפית.

כתובת ה-URL נמדדת רק פנימית (Endpoint או Route פנימי לאתר).

לדוגמה, אם המשתמש גולש ל: <http://mysite.com/sports/news> אז אפשר לומר שהכתובת הפנימית או ה-Route הפנימי הוא: /sports/news במקרה כזה, הצג קומפוננטה SportNews.

מנגנון Routing אנו בונים בקומפוננטת Routing.

מנגנון Routing לא קיים כברירת מחדל בפרויקט ריאקט, והוא נדרש להתקנה פנימית בפרויקט:

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

**HTTP Request-Response**

התקנת Northwind REST API (בטרמינל נפרד):

npm i -g northwind-rest-api

באותו הטרמינל – הרצת השרת:

northwind

**Model**

מחלקת מידע טהור המייצגת את המידע שמגיע מה-Backend.

**Service**

מחלקה המכילה פעילות לוגית.

**Axios**

זו ספרייה נפוצה לביצוע גלישות ל-REST APIs.

התקנה:

npm i axios

Props

זהו אובייקט שקומפוננטה יכולה לקבל מה-Parent שלה – הקומפוננטה שמשתמשת בה.

