# **React Definitions**

**Library – ספריה**

קבוצה של פונקציות / מחלקות שניתן להשתמש בהן במערכת שאנו בונים. אנו קובעים מתי וכיצד להשתמש.

לדוגמה: jQuery.

**Framework**

מערכת סגורה שבתוכה אנו עובדים. היא לאחר מכן לוקחת את הקוד שלנו והופכת אותו למוצר המוגמר.

לדוגמה: Angular

**MPA – Multi Page Application**

זהו אתר המכיל מספר דפים פיזיים. כל דף בכתובת אחרת. ה-Frontend גולש לכל דף בנפרד הנמצא ב-Backend.

כך היו בנויים אתרים עד אזור 2010.

**SPA – Single Page Application**

זהו אתר המכיל רק דף פיזי אחד, לרוב בשם index.html. כל שאר ה"דפים" קיימים בתוך JavaScript ומוזרקים לאזור בתוך ה-index.html. זה אומר שהדפדפן גולש רק ל-index.html ולא לשאר הדפים. הדפדפן יכול לגלוש לשירות להחזרת מידע, אך הוא לא גולש להחזרת דף נוסף.

מ-2010 זו השיטה הפופולרית יותר לבניית אתרים.

**Component**

יחידת UI קטנה הניתנת להצבה על הדף בכל מקום וניתנת להצגה מספר פעמים.

זו הטכניקה הנכונה כיום לבניית אתרים. חלוקת ה-UI לחלקים קטנים והצבתם במקומות שונים על הדף.

קומפוננטה ב-React בנויה מהקבצים הבאים (עבור TypeScript):

tsx – הלוגיקה וה-UI של הקומפוננטה (חובה)

css – זה העיצוב של הקומפוננטה (לא חובה)

test.tsx – קובץ בידקות עבור הקומפוננטה - Unit Testing (לא חובה)

ניתן להציג קומפוננטה בתוך index.html

ניתן להציג קומפוננטה בתוך קומפוננטה אחרת.

הקומפוננטה שמכילה נקראת Parent Component

הקומפוננטה המוכלת נקראת Child Component

**React**

זה Framework מבית פייסבוק המאפשרת לבנות אתר SPA מבוסס Components.

ניתן לכתוב React ב-JavaScript או ב-TypeScript (לא כותבים בשניהם יחד).

הדבר הנכון ביותר Ever בעולם כולו הוא – TypeScript.

התקנת פרויקט חדש:

npx create-react-app northwind --template typescript

הרצת הפרויקט: npm start בטרמיל

**Directory Structure**

מבנה הקבצים והתיקיות של מערכת מסוימת.

ב-React נוח לעבוד עם ה-Directory Structure הבא:

src

Components

ProductsArea

AddProduct

AddProduct.css

AddProduct.tsx

UpdateProduct

UpdateProduct.css

UpdateProduct.tsx

ProductList

SuppliersArea

EmployeesArea

Services

Utils

Models

ProductModel.ts

UserModel.ts

**Layout**

זהו הסידור הכללי של האתר.

לכל אתר הגיוני שיהיה סידור כזה. לדוגמה, כותרת בכל הדפים, זכויות יוצרים בתחתית כל דף וכו'...

באתר מבוסס קומפוננטות, ה-Layout הינו Component נפרד שמכיל כל מיני חלקים, שכל אחד מהם הינו Component.

זה אומר שיש אזור נפרד עבור ה-Layout.

קומפוננטת ה-Layout זו הקומפוננטה הראשונה שמוצגת על הדף. כל רכיב אחר מוצג בתוך ה-Layout בצורה ישירה או בתוך רכיבים אחרים.

**CSS Leaking**

זהו מצב בו עיצוב המתבצע בקומפוננטה א' משפיע על קומפוננטה ב'.

קומפוננטה א' מכילה עיצוב של פיסקה אדומה, קומפוננטה ב' מכילה פיסקה שנצבעת באדום.

זהו מצב לא רצוי.

**CLI – Command Line Interface**

אלו קבוצה של פקודות המוכרות ע"י ה-Command Line (Terminal)

**Snippet**

זהו קוד מוכן שניתן לשתול בפקודה אחת

npm i -g react-cli-snippets

זו ספרייה המכילה פקודות CLI עבור Snippets ליצירת Components.

**Rendering**

הפעולה של תרגום הקוד שלנו ל-UI הסופי

**Interpolation**

שיבוץ משתנה או ערך בתוך קוד ה-HTML של הקומפוננטה.

**Conditional Rendering**

בדיקת תנאי ולפיו אנו מציגים חלק מה-UI או לא מציגים אותו, או מציגים משהו אחר.

**Displaying Lists**

מתבצעת ע"י קריאה לפונקציה map של מערך או אוסף כלשהו כאשר הפונקציה ממפה כל פריט למשהו אחר. והמשהו האחר שאנו רוצים למפות זהו קטע ה-HTML שברצוננו להציג עבור כל פריט.

כאשר אנו מציגים רשימה, חובה לכל פריט להגדיר מזהה ייחודי ע"י המאפיין key. זאת לצורך Performance.

**Events**

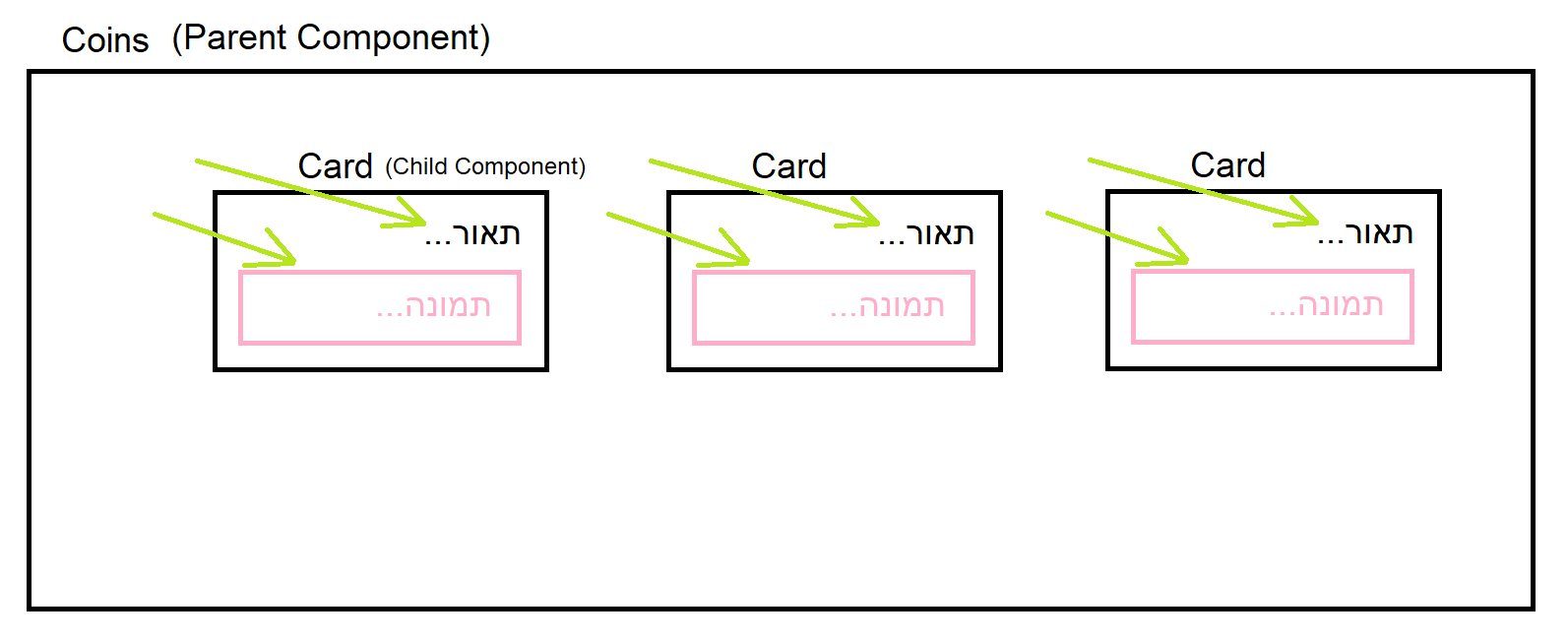
ארוע מתבצע בדומה ל-HTML. שהם הארוע שונה מבחינת Casing (אותיות קטנות / גדולות) מארועי HTML. ארוע נרשם לפונקציה.

**Props**

אובייקט אחד שה-Parent Component יכול לשלוח ל-Child Component.

ה-Child Component מגדיר האם יש לו אובייקט כזה או לא. רק אם הגדיר שיש – ה-Parent יכול לשלוח את ערכי האובייקט.

דוגמה:



**Interface**

מילה שמורה שיוצרת סוג נתונים חדש.

ניתן ב-TypeScript לבנות Literal Object (אובייקט הנבנה ע"י סוגריים מסולסלים {} ) מהסוג הזה.

Class יכולה לרשת interface ואז היא חייבת להגדיר בתוכה את כל מה שיש ב-interface.

**Component State**

מידע הקיים ברמת הקומפוננטה שמתעדכן במהלך חיי הקומפוננטה (בלחיצה על לחצן / בהבאת המידע מהשרת / בטיימר...) ואנו רוצים להציג אותו כחלק מה-UI של הקומפוננטה.

React לא מרנדרת את הקומפוננטה בהפעלת ארועים, בשינוי משתנים רגילים, בקריאה לפונקציות וכו'.

React מרנדרת את הקומפוננטה באחד משני מקרים:

1. מתי שהיא חושבת לנכון (לא ידוע)
2. מתי שאנו משנים State.

לכן אם קיים מידע (משתנה אחד או יותר) שהולך להשתנות במהלך חיי הקומפוננטה ואנו רוצים להציג אותו ברגע שמשתנה – המידע הזה חייב להיות מוגדר כ-State ולא כמשתנה רגיל.

נוהל יצירה ושימוש ב-State שונה בין CC לבין FC.

ב-CC אובייקט ה-State מגיע ממחלקת בסיס בשם Component אותה יורשת מחלקת ה-Component שלנו.

ב-FC ה-State מוגדר ע"י React Hook בשם useState.

**React Hooks**

בגרסאות הראשונות של React היה ניתן לבצע דברים מסוימים רק ע"י CC ולא ע"י FC.

לדוגמה ניהול State.

בגרסה מסוימת, הוסיפו פונקציות מיוחדות שתפקידן לספק עבור FC את אותן יכולות שהיו רק ב-CC.

פונקציות אלו נקראות React Hooks.

מהר מאוד נבנו עוד ועוד פונקציות כאלו שמספקות יכולות נוספות.

דגשים:

* ניתן לקרוא לפונקציה כזו רק ב-FC ולא ב-CC.
* כל פונקציה כזו מתחילה בתחילית use.
* חובה לקרוא לפונקציה כזו ישירות בתוך ה-FC. לא בתוך פונקציות פנימיות ולא בתוך תנאים או לולאות.

**Side-Effect**

אלו פעולות שה-Component מבצע שיוצאות מחוץ לגבולות ה-Component.

לדוגמה, Component גולש לשרת (קריאת AJAX).

לדוגמה, גישה ישירה לתגית DOM כלשהי שלא שייכת ל-Component (שאגב, זה לא מומלץ).

לדוגמה, שימוש ישיר ב-window (פתיחת טיימר)

אסור לבצע Side-Effects ישירות בתוך פונקציית ה-Component! זאת מפני שאנו לא יודעים מתי React בוחרת לרנדר את הקומפוננטה. כל רינדור כזה גורם לה להיות מופעלת מחדש. אם ביצענו Side-Effect ישירות בתוך הקומפוננטה הוא יתבצע מחדש בכל רינדור. לדוגמה – גלישת AJAX – כל רינדור גלישה מחדש לשרת!

ב-CC מבצעים Side-Effects בתוך פונקציות מיוחדות שקיימות במחלקה.

ב-FC מבצעים Side-Effects בתוך React Hook בשם useEffect.

פונקציה זו מסוגלת להריץ Callback כתלות בשינוי מידע.

הפעלת קוד פעם אחת בטעינת ה-FC:

useEffect( () => { … }, [] );

הפעלת קוד בכל פעם שהמשתנים a או b משתנים:

useEffect( () => { … }, [a, b] );

Component מופעל אוטומטית כאשר הוא מוצג.

Component נהרס אוטומטית כאשר הוא כבר לא מוצג יותר על הדף.

קורה שאנו רוצים להפעיל קוד רק כאשר ה-Component נהרס.

לדוגמה, סגירת טיימרים. אם אנו לא סוגרים טיימר, הוא לא יסגר לבד. פעולות סיום, פעולות סגירה...

הפעלת קוד כאשר קומפוננטה נהרסת:

useEffect( () => {

return () => { … };

}, [] );