React
שיעור שלישי

תנאים

- נשתמש בתנאים בריאקט בשני מקומות : בפונקציות Java Script לצורך בדיקות שונות כגון בדיקת תקינות שראינו בשיעור שעבר: תנאי ב JS נכתב רגיל כמו בשאר שפות התכנות.

```
const changeProductAmount = (newAmount, index) => {
    let updatedProducts = [...products];
    if (newAmount) {
        updatedProducts[index].amount = newAmount;
    }
    setProducts(updatedProducts);

}
```

בתצוגה – בתוך ה Return ניתן להציג תגיות html , תוכן מסוים , לעבור על לולאות... רק אם מתקיים תנאי מסוים ונכתב בצורה הבאה:

בתוך סוגריים מסולסלים נציב את התנאי, אם מחזיר true התגית תוצג, אם false – התגית לא תוצג (null)

```
let isProducts = true;
```

```
{isProducts? <h1>there are products</h1>: null}
```

useEffect

- useEffact – פונקציה המתרחשת מיד ברינדור הקומפוננט, על ידי שימוש בhook הזה, אומרים ל-React שהקומפוננט צריך לעשות משהו מיד כשנטען. שימושי מאוד בקריאות api שיעשו עם טעינת הקומפוננט וכך נוכל להשתמש במידע שהגיע.
דומה ל componentDidMount, componentDidUpdate ו-componentWillUnmount ב react classes. (לא hooks)
- הפעולות שנשים בתוך ה useEffect מתבצעות עם טעינת הקומפוננט וגם לאחר כל עידכון הנעשה בקומפוננט (יש דרך לשלוט על זמני הפעלת ה useEffect נראה בהמשך).
- כתיבת ה useEffect:
ייבוא הספריה מ react
צורת הכתיבה:

```
import { useState } from 'react';
import { useEffect } from 'react'
function First(props) {

    useEffect(() => {
        alert("You use useEffect hook! ")
    });
```

שליטה על ביצוע ה useEffect

- יש לנו אפשרות לשלוט על זמני ביצוע ה useEffect שלא יתבצע בכל שינוי הנעשה בכומפוננט – ע"י הוספת הוספת פסיק וסוגריים מרובעים במיקום זה בכתיבת ה useEffect –

```
useEffect(() => {
    alert("You use useEffect hook! ")
}, []);
```

- אם הסוגריים ריקים – ה useEffect יתבצע פעם אחת בלבד.
- ניתן להוסיף בתוך הסוגריים שם משתנה או פונקציה, זה יגרום ל useEffect להתבצע בכל פעם שמשתנה מה שנרשם בסוגריים.
- יתבצע בטעינת הכומפוננט וגם בכל שינוי במערך arr

```
useEffect(() => {
    alert("You use useEffect hook! ")
}, [arr]);
```

קריאות API

- את קריאות ה API שנצטרך בקומפוננט נבצע בד"כ ב useEffect hook. על מנת למשוך את הנתונים הנצרכים בקומפוננט מיד במהלך טעינת העמוד.
שלבים:
- צריך שיהיה לנו את ה url עם הנתונים אותם נרצה לקבל לדוג': https://api.github.com/search/commits?q=repo:facebook/react+css&page=1 ונשים אותו כמשתנה ב useEffect

```
useEffect(() => {
    const url = "https://api.github.com/search/commits?q=repo:facebook/react+css&page=1"
```

- יצירת פונקציה אסינכרונית כדי להביא את הנתונים שלנו. פונקציה אסינכרונית היא פונקציה שצריכה להמתין לאחר פתרון הבקשה לפני שתמשיך. במקרה שלנו, הפונקציה תצטרך להמתין לאחר שליפת הנתונים.

```
const fetchData = async () => {
    try {
        const response = await fetch(url);
        const json = await response.json();
        console.log(json);
    } catch (error) {
        console.log("error", error);
    }
};
```

React

קריאות API – המשך שלבים:

- נשים את הפונקציה האסינכרונית ב useEffect ונקרא לה כך:

```
useEffect(() => {
    const url = "https://api.github.com/search/commits?q=repo:facebook/react+css&page=1"
    const fetchData = async () => {
        try {
            const response = await fetch(url);
            const json = await response.json();
            console.log(json);
        } catch (error) {
            console.log("error", error);
        }
    };
    fetchData();
}, []);
```

נשים לב: הפונקציה שיצרנו עטופה בלולאה try...catch כך שהפונקציה תופסת את השגיאות ומדפיסה אותן ב console. זה מסייע באיתור באגים ומונע מהאפליקציה לקרוס באופן בלתי צפוי.

בתוך ה Try שמנו את המילה await ממש לפני הקריאה כדי לומר לפונקציה לחכות למשימת האחזור לפני הפעלת שורת הקוד הבאה.

קריאות API – המשך שלבים

- ברגע שהתקבל תגובה, אנחנו הופכים את המידע שהתקבל לנתוני JSON שאנו יכולים לקרוא בקלות. לאחר מכן, אנו מדפיסים את נתוני ה- JSON בconsole

```
const json = await response.json();
console.log(json);
```

- אפשר לשחק עם הנתונים שהתקבלו ולהציג נתונים ספציפיים לדוגמא:

```
console.log(json.items);
```

- כעת ניתן לשמור את המידע שהתקבל ב state .
- ניתן להדפיס את המידע ולהשתמש בו. כפי הצורך, בטבלה, רשימה או בתגית רגילה. כמובן לפי סוג המידע שהתקבל.

```
{apiRequest ? apiRequest.map((c, index) => (
    <table>
        {c.commit && (
            <>
                <tr>
                    <td>
                        <h4><u>
                            {c.commit.committer.name}
                        </u></h4>
                        <p>{c.commit.message}</p>
                    </td>
                </tr>
                <hr />
            </>
        )}
    </table>
)) : <h1>null</h1>}
```

React

Higher Order component (HOC) - קומפוננטות מסדר גבוה יותר

- קומפוננטה מסדר גבוה יותר (HOC) היא טכניקה מתקדמת של React שעוזרת למחזר קוד קומפוננטות. היא פונקציה שמקבלת קומפוננטה ומחזירה קומפוננטה אחרת עם פונקציונליות נוספת שמספקת לה. – במקרים בהם נרצה את אותה פעולה בכמה קומפוננטות, במקום לכתוב בכמה מקומות, נכתוב פעם אחת ונשתמש באותה הקומפוננטה כשנצטרך.
- שלבים: ניצור קומפוננט הכוללת פונקציה המקבלת קומפוננט שתהיה עטופה בפונקציונליות שהפונקציה תספק לה. דוגמא בסיסית:
- מקבלת קומפוננט
- מחזירה את אותה קומפוננט עם props או עם אפשרויות ביצוע פעולות מסוימות

```
import React from "react";
const UpdatedComponent = (OriginalComponent) => {
  function NewComponent(props) {

    return <OriginalComponent />;
  }
  return NewComponent;
};
export default UpdatedComponent;
```

React

המשך - Higher Order component (HOC)

- שימוש בקומפוננט זה בקומפוננטות אחרות:

1. נייבא קומפוננטה זו:

```
import withCounter from "./withCounter";
```

2. בייצוא הקומפוננט – export נעטוף אותה בקומפוננט:

```
export default withCounter(ClickIncrease2);
```

שם הקובץ של הקומפוננט העוטף

המשך – Higher Order component (HOC)

- ניתן להעביר props בין הקומפוננטות. כך ניתן בקלות לשתף נתונים בין קומפוננטות.

```
import React from "react";
const UpdatedComponent = (OriginalComponent) => {
  function NewComponent(props) {

    return <OriginalComponent name="Efrat"/>;
  }
  return NewComponent;
};
export default UpdatedComponent;
```

- שימוש בקומפוננט:

```
<p>Value of 'name' in ClickIncrease: {props.name}</p>
```

שימוש – Higher Order component (HOC)

- כל פעולה שיש בקומפוננט העוטף – יתבצעו בכל קומפוננט העטוף בקומפוננט זה.
- שימוש נפוץ –שימוש ב useEffect עם פעולות משמעותיות כגון קריאות API וכד' שנרצה בכמה קומפוננטות – נכתוב בHOC ונשתמש בכל קומפוננט שצריך.

Reach Router – ניווט בין עמודים

- ניווט בין עמודים שונים בreact נעשה באמצעות React Router ובשימוש בספריה react-router-dom , מאפשר לשנות את תצוגת הרכיב, את ה-URL של האתר, שלבים:
- נתקין את הספרייה react-router-dom - נריץ פקודה זו בטרמינל או ב cmd:

```
npm i react-router-dom
```

כך הפרויקט שלנו יכיר את הספרייה ונוכל לייבא את המודולים הנצרכים מספרייה זו.

- בניווט עמודים צריך שיהיה עמוד שבתוכו נוכל לנווט בין העמודים – יכיל לדוגמא כפתורים שכל כפתור ישלח לעמוד אחר. דוגמא נפוצה היא nav bar המכיל קישורים לעמודים שונים. בשביל יכולת זו נצטרך לייבא את הספריות הבאות:

```
import { BrowserRouter as Router, Routes, Route, Link } from "react-router-dom";
```

- ReactRouter – רכיב האב המשמש לאיכסון כל שאר הרכיבים.

Reach Router המשך:

- מתחלק לשניים - יש את הלינקים שזה מה שהמשתמש רואה, ולוחץ על הלינק שרוצה.

```
<Routes>
    <Route path="/home" element={<Home />} />
    <Route path="/about" element={<About />} />
    <Route path="/dashboard" element={<Dashboard />} />
</Routes>
```

```
<ul>
    <li>
        <Link to="/home">Home</Link>
    </li>
    <li>
        <Link to="/about">About</Link>
    </li>
    <li>
        <Link to="/dashboard">dashboard</Link>
    </li>
</ul>
```

- Routs – בו מגדירים את העמודים אליהם ינווטו העמודים.
- שם העמוד ב to ב Link חייב להיות כשם ה path ב Route – בלחיצה על הלינק ריאקט מעביר לעמוד ששמו כשם בלינק.
- הכל עטוף באובייקט <Routers>

שיעורי בית:

א- יצירת קומפוננט עוטף המכיל useEffect ובתוכה קריאת API המחזירה אבייקט JSON לתוך state.

ב- יצירת שני קומפוננטות, לעטוף אותן בקומפוננט מסעיף א. – (שימוש ב HOC.)

ג- באחת מקומפוננטות יש להציג את המידע שהתקבל מקריאת ה API בצורה של טבלה, ובקומפוננט השני להציג את המידע בצורה של רשימה.

בהצלחה!!

React