FULLSTACK

**תמר כהן**

2022

**הדרכה על REACT**

קורס FULLSTACK של יניב ארד

[קומפוננטות 1](#_Toc109996112)

[רינדור 2](#_Toc109996113)

[רינדור דינאמי 2](#_Toc109996114)

[הקמת פרויקט בריאקט 2](#_Toc109996115)

[Spa – Single Page Application 2](#_Toc109996116)

[Modules 3](#_Toc109996117)

[Axios 3](#_Toc109996118)

[React-router-dom 3](#_Toc109996119)

[Event Listener 3](#_Toc109996120)

[Hooks 4](#_Toc109996121)

[useState 5](#_Toc109996122)

[useEffect 5](#_Toc109996123)

[useNavigate 5](#_Toc109996124)

[היררכיה בין קומפוננטות והעברת מידע 6](#_Toc109996125)

[עבודה מול שרת ושליחת בקשות (resp API) 7](#_Toc109996126)

# קומפוננטות

אזור בדף שמציג html.

הן מקבלות קלט שרירותי (נקרא “props”) ומחזירות אלמנטים של React המתארים מה אמור להופיע על המסך.

**props הוא לקריאה בלבד**

**שינוי של props יגרום לרינדור של הקומפוננטה**

# רינדור

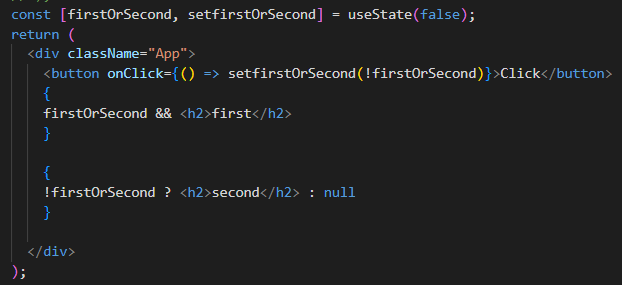
ייצור של קטע html

# רינדור דינאמי

שינוי שמבוצע בhtml בזמן שהקומפוננטה כבר רצה. למשל, שינוי עיצוב, מחיקה של מופע כלשהו וכו'

(שינוי מותנה, state בוליאני שמשתנה בהתאם ללחיצת כפתור, אם הוא true תבצע את הרקע באדום, אחרת – כחול)

דוגמא לרינדור דינאמי – נגדיר משתנה בוליאני, אם הוא true תופיע הכותרת הראשונה, אם הוא false תופיע הכותרת השנייה (אפשר לעשות הכל בתנאי אחד, זו גם דוגמא לצורות כתיבה של תנאים)



****

**Repeater**

תת תחום של רינדור דינאמי, ביצוע של פעולה מסוימת באופן רפטטיבי (למשל, הכנסת נתונים לטבלה, יצירת רשימה מתוך מערך נתון)

הערה – שימוש בריפיטר במקרה של מעבר על אוסף ערכים מצריכה מתן מפתח אישי לכל ערך

דוגמא – (המרה ממערך לכותרות , כל כותרת מקבל מפתח)

Arr.map( item , index => { return <h3 key={index} > {item} </h3> } )

# הקמת פרויקט בריאקט

בטרמינל נקליד npx create-react-app appName

ונתקין

לאחר מכן נקליד בטרמינל cd appName

להרצת הפרויקט נקליד בטרמינל npm start

(להפסקת הרצה נקליד בטרמינל ctrl+c פעמיים)

# SPA – Single Page Application

Spa הם יישומי רשת שמטרתם לתת חוויית משתמש יותר מהירה וזורמת. ביישומים אלו, כל הקוד הדרוש מגיע אל הדפדפן בטעינת דף אחת, ומשאבים נוספים נטענים בצורה דינמית, בדרך כלל כתגובה לפעולות המשתמש. דף האינטרנט לא מבצע טעינה מחדש בשום שלב, אולם כתובת האינטרנט עשויה להשתנות מעט, על מנת לתת למשתמש הבנה יותר טובה של הניווט בדף.

מעין קומפוננטה ראשית המציגה קומפוננטות שונות בהתאם לפעולות שהמשתמש מבצע

# Routing

ניווט בין "עמודים" בתוך יישום מסוג SPA.

התצוגות מתחלפות בעמוד הראשי וכך גם משתנה מעט הכתובת, ולכן מבחינת המשתמש מתבצע מעבר בין דפים. בפועל לא מתבצעת טעינה של דפים חדשים אלא החלפת תצוגות.

# Nested Routing

מבנה היררכי של Routing. למשל בתוך Router יש תת Router נוסף שגם בו יש תצוגות מתחלפות ושינויים בכתובת

למשל Routing "רגיל" יכול "להעביר" מהכתובת localhost:3000 לכתובת localhost:3000/contact

וNested Routing יכול "להעביר" גם ל localhost:3000/contact/Email.

# Master–Detail Interface

תצוגה של רשימת ערכים (עם מידע) נתונה עם פרטים נוספים של הערך הנבחר

# Session Storage

דפדפנים מודרניים מצוידים בתכונה sessionStorage ששומרת את המידע בלשונית מסוימת של הדפדפן וכל המידע בה נמחק אוטומטית ברגע שהמשתמש סגר את אותה הלשונית.

ה sessionStorage הוא אובייקט בדפדפן שמקבל מפתח ומכניס אליו ערך. מאשרת שמירת מידע ומתן גישה אל המידע מתצוגות שונות.

למשל בקוד נבצע קריאה ל sessionStorage["key"]="value"

אז בכל הדפים המורצים על בדפדפן באותה הרצה שמור הערך הנ"ל וניתן לגשת אליו ע"י קריאה ל sessionStorage["key"].

**(!) ניתן לשמור רק מחרוזות**

# Modules

## Axios

התקנה – בטרמינל נכתוב npm install axios (אפשר גם npm i axios)

ייבוא – בראש העמוד נכתוב import axios from "axios"

מה זה ? ספריית http

למה זה עוזר ? מאפשרת לשלוח בקשות http ומחזירה promise (א-סינכרוני)

## React-router-dom

התקנה – בטרמינל נכתוב npm install react-router-dom

ייבוא - בראש העמוד של הרמה העליונה נכתוב

import {BrowserRouter} from "react-router-dom"

בראש העמוד של הרמות האחרות נייבא דברים אחרים, למשל {Routes, Route, Link}

Routes – אוסף של כל הנתיבים (מכילה בתוכה מספר של נתיבים)

Route – נתיב יחיד, מגדיר אם הסיומת של הדף היא XXX תציג את הקומפוננטה XXXComponent

Link – מאפשרת לשנות את כתובת האתר בלי טעינה מחדש

מה זה ? ניתוב לתצוגות שונות

שימוש – נעטוף את הקומפוננטה ברמה העליונה עם Browser

דוגמא לשימוש –

ברמה העליונה

<BrowserRouter>

<App/>

</BrowserRouter>

ברמות אחרות

<Link to="/XXX"> \*thelink\* <Link>

<Routes>

<Route path="/XXX" element={ <XXXComponent /> } />

</Routes>

* בהיררכיה עמוקה יותר, ניתן להגדיר רק Link (בהנחה שלאחד האבות הגדרנו Routes) והניתוב יזהה את השינוי ויבצע אותו. (מכל מקום ניתן לעבור לכל מקום)
* **שימוש בכפתורים** – יתרון – בשונה מלינק , כפתור מאפשר קריאה לפונ' ולכן ביצוע פעולות נוספות בעת הניתוב.

שימוש בכפתור יבוצע ע"י Hook שנקרא useNevigate שמיובא מreact-router-dom בתוך בonClick או בפונ' שנקראת מהonClick

* **הכנסת פרמטר לכתובת** (למשל id) תתבצע בעזרת : בpath. למשל path="/XXX/:id"

כך נשמר המידע שהלינק הוא מהצורה …/XXX/something וכדי לקבל את something ניתן להשתמש בuseParams().id

דוגמא לשימוש של Nested Routing –

<Routes>

<Route path="/XXX" element={ <XXXComponent /> } >

<Route path="ZZZ" element={ <ZZZComponent /> } />

</Route>

</Routes>

וכך בתוך הקומפוננטה XXXComponent ישנו ניתוב לקומפוננטה ZZZComponent אם נגדיר Link בתוך XXXComponent.

על מנת להציג את הניתוב לZZZComponent נשתמש בתגית <Outlet/> אשר מיובאת מreact-router-dom (התגית הנ"ל מזהה את השינוי ומעבירה מידע לביצוע השינוי בהיררכיה)

(!) בpath המובילה לZZZ אין /

## Redux

התקנה – בטרמינל נכתוב npm install redux react-redux

redux זו ספריית JS שמאפשרת ניהול יעיל של state במקום אחד (ברמת כל האפליקציה).

איזה מידע נשמור ברידקס ? מידע שנרצה שיהיה נגיש **בכל** שלב באפליקציה או **ברוב** (למשל במספר רב של קומפוננטות)

פעולות ברידקס –

dispatch – שליחת מידע לסטייט (עדכון)

getState – קבלת המידע השמור

subscribe – האזנה לשינויים שמתבצעים במידע

unsubscribe – הפסקת האזנה לשינויים

ניתן להגדיר פעולות שכאשר המשתמש יבצע את הפעולות הנ"ל, המידע בstate ישתנה/יתעדכן.

**appReducer**

פונ' העדכון מקבלת את הstate (בד"כ Json) ואת הפעולה שקוראת לביצוע עדכון בstate

איך שולחים את הפעולה? הפעולה תישלח בJson שמכיל שני מפתחות , מפתח אחד מסוג type

(אנו מחויבים לספק מפתח זה) שמחזיק את סוג הפעולה, ומפתח שני payload שמחזיק את המידע

שהועבר עם הפעולה.

בתוך הפונ' נעבוד עם switch-case וכך נגדיר לכל סוג פעולה, את מה שצריך להתבצע.

**(!) על מנת לא לאבד מידע חשוב לעבוד עם spread operator כשמעדכנים את הstate**

בנוסף ניתן להגדיר default שמטפל במקרה בו סוג הפעולה לא מוכר לנו, ברוב המקרים בdefault נחזיר את הstate ללא שינוי.

שימוש –

בתיקיית הindex.js נייבא

import { createStore } from 'redux'

import { Provider } from 'react-redux'

import appStore from 'XXX'

const appStore = createStore(appReducer)

ובתוך הroot נכתוב

root.render(

<Provider store={ appStore}>

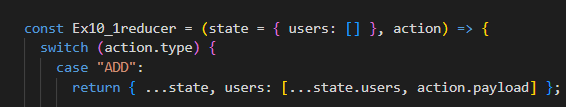
>App />

</Provider>

);

(כלומר, appStore היא בעצם הפונ' appReducer שהגדרנו)

דוגמא לשימוש –



אז הפונ' מקבלת את הstate ואת הaction שקרה. אם סוג הaction הוא "ADD" אז נבצע הכנסה של למערך הusers שבתוך הstate. מה נכניס? את המידע שהועבר עם הaction. כלומר, action.payload.

# Event Listener

* לחיצה על כפתור

<button onClick={XXX}> click </button>

<input type="button" value="click" onClick={XXX} />s

כאשר XXX זו פונקציה המתבצעת כאשר לוחצים על הכפתור או קריאה לפונ' שנרצה שתתבצע.

דוגמא –

onClick = { ( ) => console.log("Tamar")}

* הקלדה בתיבת טקסט

<input type="text" onChange={XXX} />

כאשר XXX זו פונקציה המתבצעת כאשר משנים את הטקסט בתיבה או קריאה לפונ' שנרצה שתתבצע

דוגמא –

onChange = { (e) => console.log(e.target.value)}

הערה – e זה המידע שמתקבל מתיבת הטקסט (כלומר, הטקסט שהוכנס לתיבה) וביצוע e.target.value ממיר את המידע לערך

* סימון בcheckbox

<input type="checkbox" onChange={XXX} />

כאשר XXX זו פונקציה המתבצעת כאשר משנים את הבחירה או קריאה לפונ' שנרצה שתתבצע

דוגמא –

onChange = { (e) => console.log(e.target.checked)}

הערה – e זה המידע שמתקבל מהבחירה (כלומר, באיזו אופציה בחרנו) וביצוע e.target.checked ממיר את המידע לערך (ערך בוליאני, אם סימנו או לא)

* שליחת טופס

<form onSubmit={XXX}>

כאשר XXX זו פונקציה המתבצעת כאשר שולחים את הטופס או קריאה לפונ' שנרצה שתתבצע

ברירת המחדל היא לרענן את הדף ולשלוח את המידע מהטופס לשרת

ע"מ לבטל את התנהגות ברירת המחדל וריענון הדף נגדיר onSubmit בטופס עם הפונ' XXX שמקבלת פרמטר e, ונבצע e.preventDefault(); ולאחר מכן ניתן להגדיר מה נרצה שיתבצע.

# Hooks

תחילה, נבצע יבוא לHooks שנרצה להשתמש בהם

import {useXXX} from 'ZZZ'

## useState

ייבוא – import { useState } from 'react'

Hook אשר שומר ערך עכשווי ופונ' המאפשרת לשנות את הערך.

ניתן להשתמש בHook זה יותר מפעם אחת בקומפוננטה

דוגמא –

const [num , setNum] = useState(0)

שינוי של state יגרום לרינדור של הקומפוננטה.

(בעזרת virtual DOM , זה מתבצע בצורה מאוד מהירה גם אם הקומפוננטה מכילה קוד Html מאוד ארוך)

(?) מה קורה אם הstate מכיל ערך מטיפוס json ואנו רוצים לשנות רק שדה אחד בתוך הjson?

נשתמש בspread operator (אופרטור 3 נקודות ...)

דוגמא –

עבור הstate

Const [data , setData] = useState( { fullName : "Tamar Cohen" , city : "modiin" } )

על מנת לבצע שינוי בערך city בלבד, נבצע זאת בצורה הבאה

setData( { …data, city : "beer sheva"} )

## useEffect

ייבוא – import { useEffect } from 'react'

Hook אשר שומר מידע לפעולה שצריכה להתבצע לאחר הרינדור. ברירת המחדל היא שהHook הזה רץ לאחר **כל** רינדור.

בכל שלב חיים של הקומפוננטה נוצר אפקט , ניתן להשתלב בכל נקודת זמן, בשלב הכנסה לDOM, ברגע שיש שינוי כלשהו במידע או בשלב עזיבת הDOM.

mount – שלב הייצור הראשוני

update – השלב בו מתבצעים שינויים בקומפוננטה

unmount – שלב מחיקה מהזיכרון

דוגמא –

useEffect( () => {console.log("Tamar")});

הערה חשובה – ניתן להגביל את הHook הנ"ל להתבצע במצבים מסוימים ע"י הכנסת משתנה נוסף, כאשר המשתנה הנוסף משתנה – הHook ירוץ

למשל, אם נרצה שהHook ירוץ **רק** בטעינה הראשונית ניתן לרשום

useEffect( () => {console.log(Tamar)} , []);

וכך המערך שבכתום משתנה **רק** בטעינה הראשונית ולכן הHook ירוץ רק בטעינה.

ניתן גם להתנות את ריצת הHook בstate ע"י הכנסת הstate לתוך המערך הכתום.

(!) שינוי מוגדר להיות שינוי בכתובת בזיכרון. למשל אם התננו את ריצת הHook במערך והוספנו איבר למערך הHook לא ירוץ (כי הוספת איבר למערך לא משנה את הכתובת). לעומת זאת, שימוש בspread operator יפתור את בעיה זו כי spread operator מעתיק לכתובת חדשה בזיכרון.

## useNavigate

ייבוא – import {useNavigate} from 'react-router-dom';

Hook המאפשר לבצע שינוי בסיומת הדף (כתובת)

יוסיף לכתובת של הדף את הסיומת ./XXX

מה שיעזור בביצוע ניתובים

דוגמא – נגדיר משתנה navigate ע"י const navigate = useNavigate();

וקריאה תתבצע ע"י navigate("/XXX");

וכך קריאה זו מעבירה לכתובת עם הסיומת /XXX ותוצג קומפוננטה מתאימה.

## useParams

ייבוא – import {useParams} from 'react-router-dom';

Hook אשר מייבא את הפרמטרים מהכתובת

דוגמא –

const id = useParams();

למשל אם הכתובת בRoute מנתבת ל path="/XXX/:id" והLink שולח ל to="/XXX/1" אז הפונ' useParams().id; תחזיר לנו את הערך 1.

**(!) עובד בהנחה שהגדרנו פרמטרים בכתובת (**[**ראה הסבר**](#_React-router-dom) **בReact-router-dom)**

## useDispatch

ייבוא – import { useDispatch } from 'react-redux'

Hook אשר מאפשר לבצע dispatch לפעולה וכך לעדכן את הstate בRedux

דוגמא – נגדיר משתנה dispatch ע"י

const dispatch = useDispatch();

וקריאה תתבצע ע"י dispatch( { type : "XXX" , payload : "ZZZ" } ) ;

## useSelector

ייבוא – import { useSelector } from 'react-redux'

Hook אשר מאפשר גישה למידע שנשמר בstate של הRedux בזמן אמת (כלומר, גם אם התקיימו שינויים במידע)

דוגמא – נגדיר משתנה storeData ע"י

const storeData = useSelector(state => state);

(בעצם, אם יש מידע בredux שנקרא state, תחזיר אותו)

וכעת, storeData מכיל Json וניתן לגשת למידע שבו.

# היררכיה בין קומפוננטות והעברת מידע

* **העברת מידע מאבא לבן** – באמצעות props . נוכל לקרוא לקומפוננטת ילד ולהעביר דרך הקריאה מידע.

דוגמא –

<childComponent data={nameOfStateWeWantToSend} />

כאשר קומפוננטת הבן מקבל (props).

גישה למידע זה תתבצע בעזרת props.data

* **העברת מידע מבן לאבא** – באמצעות פונ' callback . האבא יעביר לבן כתובת לפונ' שנמצאת אצל האבא.

דוגמא –

בקריאה לקומפוננטת הבן מהאבא נבצע

<childComponent callback={XXX} />

כאשר XXX זו פונ' אנונימית ששומרת מידע בstate כלשהו.

ובקומפוננטת הבן נוכל לגשת לפונ' הנ"ל בעזרת props.callback וכך לשמור מידע בstate שנמצא אצל האבא

* העברת מידע בין קרובי משפחה רחוקים

sessionStorage

localStorage

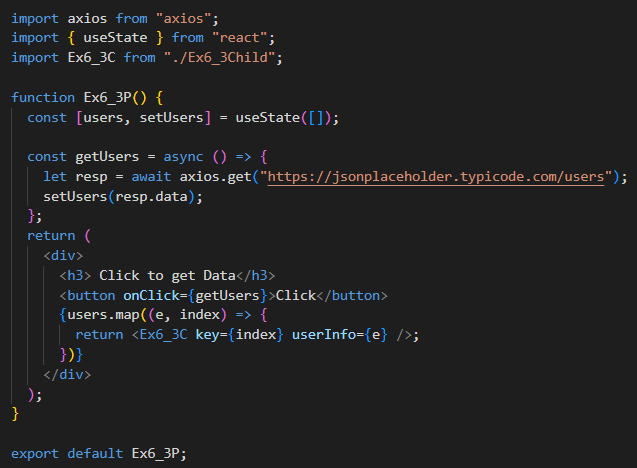
Redux

# עבודה מול שרת ושליחת בקשות (resp API)

1. עבודה ישירה מהקומפוננטה

פונ' בקומפוננטה אשר שולחת בקשה לכתובת מסוימת

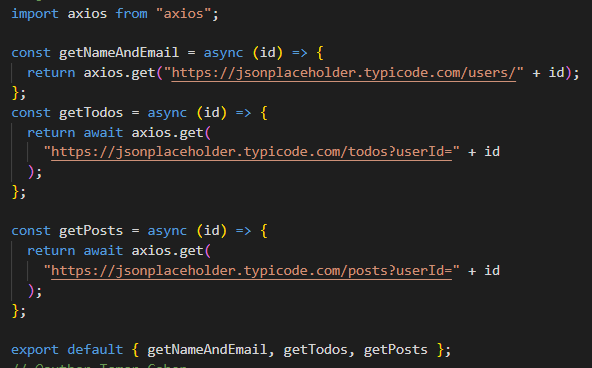
דוגמא –



1. עבודה דרך קובץ עזר – util

קובץ המכיל פונ' אשר שולחות בקשה לכתובות מסוימות. כל קומפוננטה מייבאת את הקובץ וכך מקבלת גישות לפונ' קיימות המבצעות את הבקשות

דוגמא –



* שדרוג לאופציה 2 :

נגדיר כמה שכבות util (למשל שכבה העובדת מול users ושכבה נפרדת העובדת מול product).

* שדרוג נוסף :

שכבת הutil תקבל כתובת URL כמשתנה. כך נוכל להגדיר util אחד שיבצע את אותה פעולה עבור כתובות שונות.