**מערך REACT JS**

**1. להתקנה מהירה יש לכתוב בספריית האב**

Npx create-react-app <my-app-name>

ואז להכנס לתקייה שבה הותקן האפליקציה:  
cd my-app-name  
  
מומלץ לצפות ב:  
[בעונה 10 סרטון 1 עד 3](https://www.youtube.com/playlist?list=PLP-UKVDk32JhrCHrkZe-wV_g2eAzwgP9T&pbjreload=10) בערוץ מאנקיס ביו טיוב להבנה איך JSX עובד ואיך כל REACT עובדת

**2. יצירת קומפ חדש ונראות של קומפ חדש**    
מומלץ מלבד ה APPS לייצר את כל הקומפס החדשים ב תקייה SRC/comps ואם נניח נייצר קובץ של קומפ בשם inputUser.js SRC/comps/ הוא יראה כך:

import React,{Component} from 'react';

class InputUser extends Component{

render(){

return(

<div>

</div>

)

}

}

export default InputUser;

3. כיצד לחבר בין קומפנינטות (בדוגמא קומפ APP) שקורא ל INPUT USER:

import React,{Component} from 'react';

import "./bootstrap.min.css"

import *InputUser* from './comps/inputUser';

class App extends Component{

render(){

return(

<div className="App container-fluid">

<*InputUser* />

</div>

)

}

}

export default App

\*\* שימו לב שכל מה שעלינו לעשות על מנת לחבר בין הקומפנינטות הוא לבצע ייבוא ב APP ופשוט לכתוב את הקומפ בתוך תגית במקום הרצוי

4. עבודה עם STATE ולמה צריך אותם  
אחד מהיתרונות הגדולים של REACT הוא עדכון הקומפנינטות ב LIVE במשתנים שקיימים בתוכם , ועל מנת שיהיה ניתן להפעיל את השינוי ב LIVE אנו עובדים עם המאפיין STATE שקיים BUILD IN בתוך סביבת העבודה REACT וכדי לעבוד עמו אנו יכולים להכריז (למשל באפלקציית השעות שנבצע בתוך ה APP על STATE בשם DEADLINE בצורה הבאה בתוך המחלקה של ה APP:

class App extends Component{

*state* = {deadline:"2021-05-05"}...

ועל מנת להשתמש ב STATE נוכל להציג אותו ב VIEW ב (JSX) בפונקציית ה RENDER בצורה הבאה:

render(){

return(

<div className="App container-fluid">

<h2> {this.*state*.deadline}</h2>

<InputUser />

</div>

)}

5. האזנה לאירועים ב REACT  
על מנת להאזין לאירוע בצורה החכמה ביותר אנו פשוט נייצר כפתור שיעדכן את ה DEADLINE אך קודם ניצור כפתור שיציג הודעת ALERT ב JSX נוסיף מתחת ל H2 את הכפתור הבא:

<button className="btn btn-danger" onClick={this.changeDeadLine}>change</button>

ובתוך המחלקה נכריז על הפונקציה בצורת ARROW FUNCTION על מנת להמנע משימוש בBIND:

changeDeadLine = () => {

alert("Work")

}

6. עדכון ה STATE ואם נרצה לעדכן את ה STATE כל מה שנצטרך לעשות בתוך הפונקציה זה לכתוב את הפקודה SETSTATE שגם אוספת את המידע לתוך מאפיינים של ה STATE וגם בסוף האיסוף מרפרשת את הקומפנינטה ואת האזור שמשתמש ב STATE:

changeDeadLine = () => {

this.setState({deadline:"2040-03-04"});

}

7. העברת משתנים ל קומפנינטה ילד ומה הוא ה PROPS ?  
נייצר קומפנינטה שתכיל את המידע של ה DEADLINE בשם TIMER בתוך תקיית   
src/comps/timer.js ונייצר בתוכה מחלקה ונבצע IMPORT ו EXPORT בדומה לסעיף 2  
בתוך קומפנינטה ה APP נקרא לטיימר בצורה הבאה (וכמובן נעשה לה IMPORT בדומה לסעיף 3):

render(){

return(

<div className="App container-fluid">

<Timer />

<InputUser />

</div>

)}

וכדי להעביר את המידע נניח של ה STATE של ה DEADLINE כל מה שנצטרך לעשות זה לכתוב על תגית הקומפ TIMER את המאפיין הבא (נוכל לתת לו כל שם אך אדיף את שם המאפיין של הSTATE כדי שיהיה קל להבין מה הוא מייצג בתוך הקומפ עצמו):

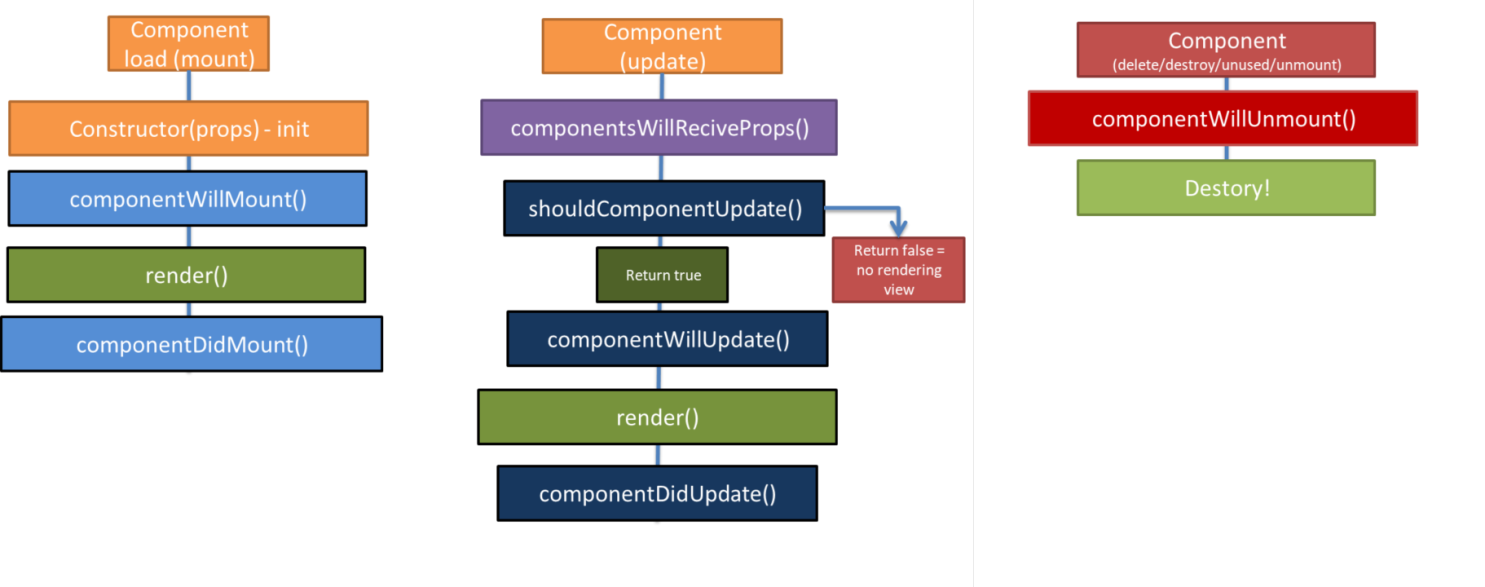
<Timer deadline={this.state.deadline} />

בתוך הקומפנינטה TIMER עצמה כדי לדבר עם המשתנה ,נצטרך לפנות למשתנה ה PROPS שנמצא בילד אין בתוך שפת ה REACT וכל מה שנצטרך לכתוב הוא:

<h2> {this.*props*.deadline}</h2>

44.ענייני זיכרון ועבודה חכמה ומועילה.

א. אם ניתן עדיף לייצר קופנינטה שהיא STATELESS   
ב. לעולם לא להשתמש במשתנים ב VIEW אלא רק דרך ה STATE וגם לעדכן אותם דרך ה setState  
ג. אם אין צורך לרנדר את הקומפנינטה למרות שה STATE שונה יש להעזר ב HOOK   
shouldComponentUpdate  
ד.

A. HOOKS LIFE CYCLE בריאקט  


**B. כיצד מבצעים פעולות ROUTE בריאקט:**  
כפי שהבנו אחד מהיתרונות הגדולים של ריאקט הינו שריאקט אפליקציה/אתר של דף אחד שאזורים מסויימים בו (או כולו) נטענים מחדש, על מנת שנוכל ליצור מצב שנוכל ללחוץ על לינק ושהדף לא יתרפרש אלא רק יציג את הקומפנינטה החדשה באזור הרצוי אנו ניהיה חייבים לעבוד עם ROUTER והROUTER שאנו נעבוד עמו נקרא:  
react-router-dom ולכן קודם כל עלינו להתקין את המודול שלו:  
npm I react-router-dom

ונבצע לו IMPORT בקומפנינטה שנרצה שתטען קומפנינטות לפי הURL שאנו נמצאים בו (בדרך כלל זה יהיה קומפ ה APP):

import { BrowserRouter as Router, Switch, Route ,Link } from 'react-router-dom';

הצעד הבא שנעשה הוא ליצור 3 קופנינטות שהVIEW שלהם שונה ונעשה להם IMPORT ב APP : page1,page2,page3  
ואז בתוך פונקצית ה RENDER של קומפ ה APP נכתוב את הקוד הבא:

render(){

return(

<Router>

<nav>

<Link to="/">Page 1</Link>

<Link to="/page2">Page 2</Link>

</nav>

<Switch>

<Route exact path={`/`} component={Page1} />

<Route exact path={`/page2`} component={Page2} />

</Switch>

</Router>

)

}

\*\* שימו לב שכאשר נעטוף את כל אזור שינוי ה ROUTER בתגית הקומפ ROUTER וכדי שריאקט ידע להחליט איזה קומפנינטה עליו לטעון לפי ה ROUTE URL נעטוף את האזור הפנימי בתגית הקומפ SWITCH ואז כל אפשרות בתגית הקומפ ROUTE כרגע אם נכנס ל URL localhost:3000 יטען קומפנינטה PAGE1 אך אם נכנס ל localhost:3000/page2 תטען קומפנינטה page2   
  
בנוסף ניתן לראות את תגית NAV שבתוכה במקום לינקים רגילים (A) יש תגית הקומפ LINK ובמקום המאפיין HREF הוחלף המאפיין ל to , ועכשיו שנלחץ על לינק במקום שהוא ישגר אותנו לדף וירפרש את הדפדפן הוא בעצם יעשה סוג של פעולת "AJAX" שרק מעדכנת חלק מהעמוד מה שיגורם לאתר לפעול הרבה יותר מהר !  
  
  
**C. כיצד לאוסף פרמטר מה ROUTER בואו נוסיף בתוך תגית ה SWITCH את השורה הבאה:**

<Route path={`/page3/:id`} component={Page3} />

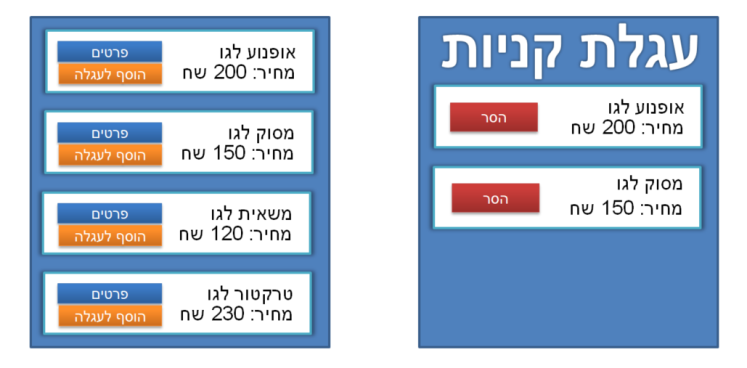
שימו לב שהוספנו את /:id הוא יתנהג כפרמטר שניתן לאסוף בתוך הקומפ PAGE3 , ואם ננסה עכשיו להכנס ל  
localhost:3000/page3/22  
ונוסיף בתוך ה RENDER של PAGE3 את הפקודה הבאה:

render(){

console.log(this.props.match.params.id) ...

נוכל לראות שבקונסול נקבל את הספרה 22 כי זה השם שנתנו ל פרמטר !  
  
**D. כיצד להעביר מאפיינים לקומפנינטה שנקראת דרך ה ROUTE**   
כדי להעביר מאפיינים לקפנינטה ל PROPS שלה כל מה שנצטרך לעשות הוא לכתוב את השורה באופן הבא:

<Route path={`/page4/:id`} render={() => <Page4 myProps={props} />} />

**E. מה זה GENERIC COMPS ולמה צריכים אותם?**  
בואו נניח שנרצה לייצר את 2 הרשימות הבאות:  
  
כרגע ממה שאנחנו יודעים מכיוון שלכל רשימה בפריטים שלה יש כפתור שונה נשמע שאנו נצטרך לייצר לפחות 2 קומפנינטות רשימות. אבל לא חבל? נזכיר את חוק ה DRY של "אל תחזור על עצמך" וGENERIC COMPS בא לפתור את הבעיה הזאת , חשוב כבר להבין שלא חובה להשתמש ביכולת הזאת ונדיר המקרים שאכן השימוש בה יקל עלינו אך חשוב להכיר אותה בשביל המקרים כמו הנל שיכולים לכווץ לנו את מספר הקבצים של האפליקציה/אתר  
  
נדביק את הקוד לקומפ של הPRODUCTS :

import React,{Component} from 'react';

class ListProduct extends Component{

state = {};

componentWillMount(){

this.setState({prods:this.props.products});

}

render(){

let prods = this.state.prods;

let **customCompBtnData** = this.props.**compData**;

return(

<div className="row">

{prods.map(item => (

<div key={item.id} className="col-6 border">

<h3>{item.name} Lego</h3>

<div>Price: {item.price}</div>

{**customCompBtnData**(item)}

</div>

))}

</div>

)}}

נתחיל מהסוף – שימו לב שאנו משתמשים ב "פוקנציה בשם customCompBtnData אבל מהיכן היא הגיעה? ובכן כמו שניחשתם היא הגיעה מהקומפ אב APP שנראה כך:

import React,{Component} from 'react';

import "./bootstrap.min.css";

// import Product from './comps/product';

import ListProduct from './comps/list\_product';

class App extends Component{

products\_ar = [

{name:"Truck",price:"50 nis",id:1},

{name:"Fireman",price:"20 nis",id:2},

{name:"Police",price:"30 nis",id:3},

{name:"Ambulance",price:"55 nis",id:4},

]

removeProd = (e) =>{

let elem = e.currentTarget;

let prod = (this.products\_ar.find(o => o.id == elem.dataset.name));

this.products\_ar.splice(this.products\_ar.indexOf(prod),1);

console.log(this.products\_ar)

this.forceUpdate()

}

addProd = () => {

this.products\_ar.push(

{name:"koko",price:"100 nis", id: this.products\_ar.length+1}

)

this.forceUpdate()

}

render(){

return(

<div className="container">

<h2>generic components</h2>

<ListProduct products={this.products\_ar} ***compData***={

(item) => (

<button data-name={item.id} onClick={ this.removeProd}>{item.name} remove</button>

)

} />

<h2>Add Prod</h2>

<ListProduct products={this.products\_ar} ***compData***={

(item) => (

<button data-name={item.id} onClick={ this.addProd}>{item.name} add</button>

)

} />

</div>

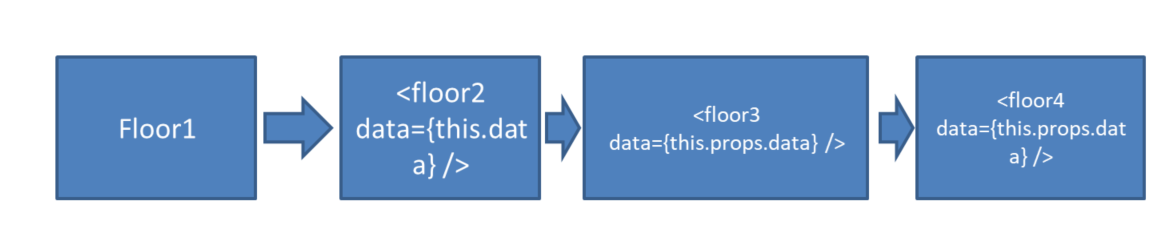
)

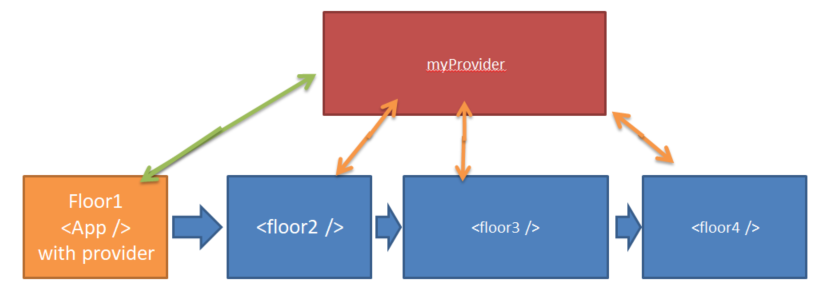
}

}

export default App

כשמסתכלים על זה השתלנו מאפיין על הקומפנינטה בשם compData שהינו STATELESS COMP  
  
וכפי שראיתם הפתרון הוא די פשוט – אך ה"קושי" היחידי שנתקל בו הוא הפונקציות שיפעלו דרך הכפתורים במקרה שלנו ששם מומלץ שיהיו אנונימיים או לצורך העניין יקראו לפונקציה מקומפ האב כמו בדוגמא

**F. איך עובדים עם CONTEXT בריאקט ולמה צריך אותו:**  
כפי ששמנו לב , כאשר האפליקציה גדלה ומתחיל להיות לנו קומפנינטות "נכדים" ו"נינים" שצריכים לאסוף את המידע הראשי מקומפ הסבא או מהסבא רבא אנו מתחילים להכנס לסוג של "לופ" של העברת פרופס מקומפ בן לקומפ נכד וכו'... (ראו את התרשים הבא קומה 1 מייצגת את קומפ APP:):  
  
  
אז על מנת לפתור את הבעיה CONTEXT עוזר לנו במקרה הנ"ל לדבר ישירות עם FLOOR4 כאשר בעצם יש לנו סוג של PROVIDER שיודע לדבר עם כל הקומפנינטות והתרשים יראה כך:



אך כדי שנוכל להשתמש ב PROVIDER עלינו קודם לייצר CONTEXT ואנו נייצר אותו בתיקייה בשם הקובץ:  
data/context.js והוא יראה כך:

import React from "react";

export const *MyContext* = React.createContext();

\* אחד הסיבות שאנו צריכים את ה CONTECT הוא זה שיכול לקבוע אם קומפנינטה תספק (provider) את ה STATE שלה (כמו FLOOR 1 בתרשים) או תצרוך ((consume (כמו FLOOR 2 עד 4) מידע של ה STATE מקומפנינטה אב שיספק את הSTATE שלו  
  
דבר ראשון בקובץ ה APP.js נוסיף ייבוא לCONTEXT שלנו בצורה הבאה:

import {***MyContext***} from './data/context'

ו  
ובנוסף בקובץ ה app.js נכתוב בתוכו את המחלקה (ניתן גם לייצר אותה בקובץ נפרד ב data/provider.js ולעשות לה MYPROVIDER ייבוא בקובץ ה App.js :

class **MyProvider** extends Component{

***state*** = {

name:"koko",

age:30

}

changeAge = () => {

this.setState({age:50})

}

render(){

return(

<*MyContext*.Provider value={{data:this.*state* , changeAge: this.changeAge}}>

{this.props.children}

</*MyContext*.Provider>

)

}

}

\*\* שימו לב שבתוך ה PROVIDER העברנו PROPS בשם ***VALUE*** שמכיל את כל המידע של ה STATE הגלובלי שיצרנו.  
  
כדי שכל הילדים והנכדים והשושלת של קומפנינטות ה APP יקבלו את היכולת להשתמש ב STATE של ה PROVIDER עלינו לעטוף את ה VIEW של קומפנינטת ה APP בצורה הבאה:

class App extends Component{

render(){

return(

<**MyProvider**>

<div>

<Floor2 />

</div>

</**MyProvider**>

)

}

לאחר שביצענו את זה נייצר קובץ קומפ חדש בתקייה comps/floor2.js שיראה כך:

import React,{Component} from 'react';

import {*MyContext*} from '../data/context'

import Floor3 from '../comps/floor3'

class Floor2 extends Component{

render(){

return(

<*MyContext*.Consumer>

{ context => **(**

<div>

This is floor2 {context.data.name} {context.data.age}

<button onClick={ context.changeAge}>change age</button>

<Floor3 />

</div>

**)**

}

</*MyContext*.Consumer>

)

}

}

export default Floor2

\*\* שימו לב ששוב עשינו ייבוא לקובץ data/context ובנוסף עטפנו את הקופנינטה ב myContext.consumer (צרכן והפעלנו פונקציה (ARROW) שמקבלת פרמטר שקראנו לו CONTEXT שבתוכו קיימים כל הPROPS וה methods שהגדרנו בפרופס ***value*** של ה PROVIDER  
וניתן לייצר קומפנינטה ב FLOOR 3 שנעטפת בצורה זהה עם עם myContext.consumer   
אחד מחסרינותיו הגדולים של CONTEXT לעומת REDUX הינו אי האפשרות לעשות בדיקות עם ה STATE הגלובלי על הקומפנינטה בקלות

G.מה זה REDUX  
הרעיון של REDUX די דומה ל CONTEXT רק ש REDUX עובד חכם יותר מבחינת הזכרון  
להתקנה יש לכתוב דרך ה NPM:  
npm I redux react-redux

בואו נייצר בתור התחלה את הקוד הבא בתוך קומפנינטה COUNTER שתחובר ל APP:

import React,{Component} from 'react';

import "./bootstrap.min.css"

class Counter extends Component{

state = { counter:0}

add1 = () => {

this.setState({counter:this.state.counter + 1})

}

sub1 = () => {

this.setState({counter:this.state.counter - 1})

}

render(){

return(

<div className="container">

<h1>redux counter</h1>

<div class="col-5 border text-center p-3">

<h2>{this.state.counter}</h2>

<button onClick={this.add1}>+</button>

<button onClick={this.sub1}>-</button>

</div>

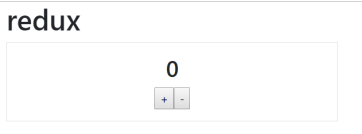
</div>

)

}

}

export default Counter

התוצאה תראה ככה כרגע עם כפתור פלוס ומינוס פשוטים שמוסיפים ומחסרים:  


עכשיו בואו נתחיל לנסות להכניס את הREDUX בואו נכתוב את הקומפנינטה COUNTER בצורה הבאה:

import React,{Component} from 'react';

import { **connect** } from "react-redux";

class Counter extends Component{

// state = { counter:0}

add1 = () => {

// this.setState({counter:this.state.counter + 1})

}

sub1 = () => {

// this.setState({counter:this.state.counter - 1})

}

render(){

return(

<div className="container">

<h1>redux counter</h1>

<div className="col-5 border text-center p-3">

<h2>{this.props.counter}</h2>

<button onClick={this.add1}>+</button>

<button onClick={this.sub1}>-</button>

</div>

</div>

)

}

}

const *mapState* = state => ({

counter:state.**momo**

})

export default **connect**(*mapState*)(Counter);

ובינתיים הקומפ APP יראה כך:

import React,{Component} from 'react';

import "./bootstrap.min.css"

import Counter from "./comps/counter";

import {createStore} from "redux"

import {***Provider***} from 'react-redux'

const ***store*** = createStore(reducer);

function reducer(){

return {

**momo**:60

}

}

class App extends Component{

render(){

return(

<*Provider* store={***store***}>

<Counter />

</*Provider*>

)

}

}

export default App

כרגע שנפעיל את האפליקציה, האפליקציה די דלה והיא רק יודעת להציג את המידע, ששמרנו, אז איך נגרום לאפליקציה לעבוד עם הפלוס והמינוס?  
  
[כיצד להשפיע על STATE](https://redux.js.org/api/store)

// במקום להכניס לרידיוסר ... עם תנאי

const initState = { momo: 90}}

function reducer(state = initState, action){

//if(state === undefined)

console.log(action)

if(action.type === "add1"){

return { momo: state.momo + 1 }

}

else if(action.type === "sub1"){

return { momo: state.momo - 1 }

}

return state;

}

const ***store*** = createStore(reducer);  
***store***.dispatch({type:"add1"})

\*\* שימו לב : כאשר אנחנו בפעם הראשונה מפעילים אפליקציה עם REDUCER אוטומטית ריאקט תפעיל את פונקציית ה REDUCER העניין שבפעם הראשונה לא ישלח ACTION מוכר ולכן הפרמטר STATE נהוג שנשווה אותו ל STATEINIT עם כל המאפיינים החשובים.  
  
כדי להפעיל את ה REDUCER פעם נוספת יש להשתמש כבר ב store.dispatch עם ה TYPE שנרצה לבצע עליו את הפעולה.  
  
עכשיו נשאלת השאלה אז איך נעביר את ה DISPATCH של ה STORE ל תוך קומפנינטה ה COUNTER?  
  
ובכן התשובה קלה ממה שחשבתם מסתבר שREDUX בפונקצייה ה CONNECT כבר דואג להעביר ב PROPS של ה MAPSTATE את הפקודה של ה DISPATCH ולכן כל מה שנשאר לנו לעשות ב COUNTER.js הוא:

add1 = () => {

this.props.dispatch({type:"add1"})

}

sub1 = () => {

this.props.dispatch({type:"sub1"})

}

וזהו יש לנו REDUX עובד ועכשיו נוכל להכניס קומפנינטה בתוך קופנינטה ותמיד להוסיף לה PROVIDER

**H. רידקס 2019 (מה זה HOOKS)**  
בשביל להבין איך הרידקס החדש עובד עלינו קודם כל להבין איך HOOKS חדשים של ריאקט עובדים:  
קודם כל נייצר קומפנינטה שנראת כך:

import React, { Component } from 'react';

function ShowCounter() {

let counter = 20

return (

<div>

ShowCounter

<h2>

counter: {counter}

</h2>

<button>add</button>

</div>

)

}

export default ShowCounter

אם נרצה שה COUNTER יושפע על ידי ההוקס החדש (ללא קשר לREDUX הוקס) נגלה שנוסף לנו היכולת שנקראת   
useState שמכילה 2 מאפיינים – הראשון משתנה עם הSTATE, השני פונקציה שמשפיעה על השינוי של הSTATE ועושה בשבילנו את ה

import React, { Component ,useState } from 'react';

function ShowCounter() {

const [**counter**, **setCounter**] = useState(5);

return (

<div>

ShowCounter

<h2>

counter: {**counter**}

</h2>

<button onClick={

() => { **setCounter**(**counter**+1)}

}>add</button>

</div>

)

}

עכשיו בוא נראה איך נבצע את אותו צעד כאשר נהפוך את COUNTER למשתנה בSTATE גלובלי ב STORE של הרידקס  
  
I. שימוש ב REDUX HOOKS   
אז דבר ראשון שנבצע הוא בקובץ ה APP.JS יבוא של המודולים הבאים:

import {createStore} from 'redux';

import { Provider } from 'react-redux';

\*\* בינתיים נראה כמו רידקס רגיל .  
בנוסף נייצר את הSTORE ונעטוף עם PROVIDER את ה APP (לא לשכוח לעשות ייבוא לקומפ ולריאקט)

const store = createStore(***RootRed***)

class App extends Component{

render(){

return(

<Provider store={store}>

<div className="container">

<ShowCounter />

</div>

</Provider>

)

}

}

עכשיו נצטרך גם ליייצר את ה REDUCER שאנו רגילים מ ריאקט הרגיל בשם ROOTRED

const initState = {

counter: 30

};

function ***RootRed***(state = initState, action){

switch(action.type){

case "add":

return {counter:state.counter +1};

case "sub":

return {counter:state.counter - 1};

default:

return state;

}

}

export default RootRed;

עכשיו כל מה שנשאר לנו לעשות בתוך הקומפנינטה של הCOUNTER לכתוב את הקוד הבא (ושימו לב אין CONNECT או הגדרה של STATE גלובלי , ריאקט עם REDUX מוכנה כבר לעבודה !

import React, { Component } from 'react';

import { useSelector, useDispatch } from 'react-redux'

function ShowCounter() {

let counter = useSelector(state => state.counter);

let dispatch = useDispatch();

return (

<div>

ShowCounter

<h2>

counter: {counter}

</h2>

<button onClick={

() => dispatch({type:"add"})

}>add</button>

</div>

)

}

J.איך עושים TEST UNIT פשוט ב REACT עם JEST ו Enzyme  
בשביל לבדוק את ה APP יש קודם לכתוב בCLI:  
npm run test

בשביל לבדוק קומפ אחר (קודם נייצר את הקוד הכתוב מטה):

import React from 'react';

import React,{Component} from 'react';

class Mycomp extends Component{

constructor(props){

super(props);

this.state = {

isShown:true,

}

}

toggleIsShown = () => {

this.setState({isShown: !this.state.isShown})

}

render(){

const { isShown } = this.state;

return(

<div>

<button onClick={this.toggleIsShown}>toggle</button>

<div>Text go here</div>

</div>

)

}

}

export default Mycomp

בנוסף נייצר myComp.test.js באותה תיקייה, מומלץ למחוק את app.test.js והקוד שיהיה ב MYCOMP TEST

describe('My comp',() => {

it('should be true', () => {

const foo = true;

expect(foo).toBe(true);

});

it('should be false', () => {

const foo = true;

expect(foo).toBe(false);

})

})

כדי שנוכל לקשר ב TEST בין ה DOM לTEST נצטרך להתקין את ה MODULE הבא בשם Enzyme , JEST כבר מותקן בנוי בתוך ריאקט לכן אין צורך בהתקנה שלו:  
npm install --save-dev enzyme enzyme-adapter-react-16  
ונכתוב את הקוד הבא בתוך myComp.test.js :

import React from 'react';

import Enzyme , {shallow} from 'enzyme'

import Adapter from 'enzyme-adapter-react-16';

import MyComp from './myComp'

Enzyme.configure({adapter: new Adapter()})

describe('My comp',() => {

it('show show text' , () => {

const wrapper = shallow(<MyComp />);

const text = wrapper.find('div div');

expect(text.text()).toBe('Text goes here');

})

it('hide text ?' , () => {

const wrapper = shallow(<MyComp />);

const button = wrapper.find('button');

button.simulate('click');

const text = wrapper.find('div div');

expect(text.length).toBe(0);

})

})

פקודות חשובות ב TESTING של ה SHALLOW :

|  |  |
| --- | --- |
| toBe | שווה בדיוק למה שצפינו אם זה סטרינג או מספר |
| toBeGreaterThanOrEqual | שווה או גדול מהמספר |
| toBeLessThanOrEqual | שווה או קטן מהמספר |
| button.simulate('click(' | מדמה לחיצה על כפתור |
| shallow(<MyComp />); | מדמה בזכרון את הקומפנינטה שלנו |
| wrapper.find('div div'); | מחפש לפי סלקטור של CSS את האלמנט בקומפ שעשינו לה SHALLOW ומחזיר את המידע שלו |
| wrapper.state('days') | מתחבר ל STATE של אותו קומפ ומחזיר אותו על מנת שנוכל לבדוק אותו עם TOBE וכו… |
| wrap.setState({prop:"val"}) | קובע סטייט חדש לאותו קומפ |
| wrap.setProps({prop:"val"}) | קובע פרופס חדש לאותו קומפ (לפעמים על מנת לבדוק את הסטייט) |
|  |  |
|  |  |
|  |  |