**React**

#### 2019-12-08 – Lesson 32 - New Project, Components, Layout

# Setting new Project

1. לפתוח תיקייה
2. לפתוח CMD
3. לכתוב את הפקודה

create-react-app (nameOfProject) --typescript

1. לכתוב cd nameOfProject – להיכנס לתיקייה
2. להיכנס ל vs code – code .
3. ב terminal לפתוח את הפרויקט npm start
4. לפתוח ב src תיקיית compensates – בה נכתוב את כל ה components של הפרויקט

ללמוד generic -

* ב React אנחנו כותבים Components
* component היא יחידת UI
* component יכול להכיל component אחר – והם יכונו child component ו component Parent בהתאמה
* component נכתב כ class

# Layout – מבנה האתר שלנו

    display: grid; /\* Grid Layout \*/

    grid-template-rows: 15% 80% 5%;

    grid-template-columns: repeat (12, 1fr); /\* fr = fraction\*/

1. מגדירים על ה class layout הגדרת css מסוג display: grid
2. הפקודה     grid-template-row – כל שורה מוגדרת באחוזים
3. הפקודה     grid-template-column – מגדירה את רוחב העמודות (לרוב עובדים עם הנוהל של bootstrap שהוא 12 עמודות ((12, 1fr

#### Lesson 33 - Props, State, Interpolation, Property Binding, Event Binding

# Prop and State

* שני מושגים שמגדרים מידע (Data)

State - מידע אותו ה component מחזיק

* לדוגמה מודעה שמכילה 10% הנחה

Only now, 10 % discount on all products!

* ה State שלה יהיה 10.
* ה State יוכל להיכתב hardcoded (כמו בדוגמה) או על ידי משתנים

Prop – מידע שנשלח מ component parent אל component child.

לפי props ו state אנחנו יכולים ליצור ארבעה סוגים של components:

1. Component – ללא שום data
2. Component <prop, any> - מכיל prop
3. Component <any, State> - מכיל רק state
4. Component <prop, state.

## איך כותבים State?

1. כיוון שאנחנו עובדים בתצורה של class את ה state מגדירים על ידי שימוש ב interface (השם יהיה כשם ה class)

interface HomeState {

    currentDiscount: number;

    imageWidth: number;

    textColor: string;

}

* במקרה שלנו ה class Home לא מקבל Props ולכן האפיון שלו יהיה רק על פי state

export class Home extends Component<any, HomeState>

* המילה any מציינת default כאשר לא מגדירים ל component איזשהוprop או state

1. כיוון שאנחנו עובדים ב class אנחנו צריכים לאתחל את המשתנים ב constructor.

    public constructor (props: any) {

        super(props);

        this.state = {

            currentDiscount: 12,

            imageWidth: 300,

            textColor: "",

    }

* כל data שאנחנו רוצים להכניס ל class שלנו נכנס תחת אובייקט state.
* קריאה ל data תעשה על ידי הפקודה this.state.

 <p>Only now, {this.state.currentDiscount}% discount on all products/>

* בדוגמה ממוקדם, במקום להכניס את הספרה 10 הכנסנו משתנה אותו אנחנו יכולים לשנות בצורה יותר דינמית

# כתיבת פונקציות (this.setState)

* כאשר אנחנו רוצים להשפיע על ה state שכתבנו דרך פונקציה אנחנו לא יכולים לגשת אליו בצורה ישירה (בפועל אנחנו כן יכולים אבל react לא תעדכן את השינוי ב DOM).
* אנחנו צריכים להשתמש בפונקציה של react שנקראת this.setState

private reasetImage = () :void => {

        this.setState({imgWidth: 300})

    }

* כיוון שאנחנו משתמשים ב this הפונקציה חייבת להיכתב כ arrow function כדי ש this יוכר כ component שלנו ולא כאלמנט HTML שמפעיל את הפונקציה.

# Binding

* קישור בין שני מקורות מידע
* שינוי במקור גורם לשינוי משתנה ביעד

למדנו על ארבעה סוגים:

1. Interpolation – שימוש בסוגריים מסולסלות {} כדי להזריק משתנים ל DOM
2. Property Binding – שינוי property של HTML Object (כמו color, width..)
3. Event Binding - שינוי state כתוצאה מהפעלת Event
4. Two Ways Binding – שינוי state כתוצאה מ User info

**Event Binding**

* אירועים ב React מתחילים עם אות גדולה

<button onClick={this.decreaseImage}>&darr;</button>

* קריאה לפונקציה מצריכה שימוש ב this.

#### Lesson 34 - Two-Way Binding, Conditional Rendering, Collection Looping, SFC

# Two Ways Binding

* שינוי state כתוצאה ממידע שמתקבל מה DOM
* מידע מה DOM מתקבל על ידי שלושה אלמנטים – Input, select and textarea.

**איך זה עובד ? – דוגמה בכיתה לשינוי צבע טקסט כתוצאה מהזנה בתיבת input**

1. מגדירים את ערך התחלתי ל property עליו אנחנו רוצים לעבוד (במקרה הזה color)

interface HomeState {

    pColor: string;

}

public constructor(props: any) {

        super(props)

        this.state = {

            pColor: "",

        }

    }

1. בתוך תיבת input מגדירים אירוע שיפעיל פונקציה (event binding)

 <input type="text" onChange={this.updateColor}></input>

1. הפונקציה מקבלת אובייקט שמכיל מידע לגבי האירוע (args).

    private updateColor = (args: SyntheticEvent) => {}

* האובייקט הוא מסוג Synthetic Event
* args.target – אלמנט ה HTML ממנו הופעלה הפונקציה (כפתור, תיבת Input וכו)
* args.target.value – משתנה שה Userהכניס

1. נשתמש ב setState כדי לקשר (binding) בין הערך החדש שמוזן (target.value) לערך שנמצא ב state שלנו

this.setState({pColor : (args.target as HTMLInputElement).value})

* צורת הכתיבה הזו אומרת – תזהה את האובייקט שמתקבל (args.target) כסוג Input ותביא את ה value שלו.

1. באלמנט שאנחנו רוצים לשנות הצבע שלו - נכתוב את המאפיין Style עם Property color, לתוך אותו Property נכניס את ה State שיצרנו

<section style={{color: this.state.pColor}}>

* את ה state צריך להכניס עם interpolation

# Conditional rendering

* לתנאים קיימת תכונה שנקראת short circuit – מצב בו התנאי לא נבדק עד הסוף

short circuit יכול להתקיים השני מצבים:

1. בתנאי && - כאשר נבחנים שני תנאי true והתנאי הראשון מתקבל כ false – התנאי לא ימשך להיבדק
2. בתנאי || - כאשר התנאי הראשון נענה כ true התנאי לא ימשיך להיבדק

**איך זה קשור אלינו ?**

* אנחנו יכולים להציג אלמנטים ב DOM כתלות בתנאי - Conditional rendering

דוגמה בכיתה – פונקציה שתציג טקסט רק אם קיץ.

1. הפונקציה – בודקת אם קיץ

    private isSummer(): boolean {

        const mounth = new Date().getMonth() +1;

        return mounth <= 9 && mounth >= 6

    }

1. התנאי שנמצא ב DOM

{ this.isSummer() && <p>Summer Sale!</p> }

* אם true תעדכן את ה DOM עם אלמנט p שמכיל טקסט

דוגמה נוספת - אם אנחנו רוצים להציג מידע בין אם true בין אםfalse התנאי צריך להיכתב

בסינטקס של תנאי טרינארי

<p> {this.isWinter() ? "Avocados": "No Avocados"}</p>

* אלמנט P תמיד יוצג ב DOM
* התוכן שלו ישתנה כתלות בתשובה של הפונקציה

# Loops over Collection

* עברנו על ארבעה פקודות נפוצות שיכולות לרוץ על מערך ולתת לנו מידע

find – מוצאת את הערך הראשון שעונה על תנאי

    // מציאת המספר הראשון הגדול מ-50

    const result1 = arr.find(n => n > 50);

    document.write("First value larger than 50: " + result1 + "<br>");

filter – מחזירה את כל הערכים שעונים על תנאי מסויים (תחזיר רק חלק מהמערך)

  // מציאת כל הזוגיים

    const result2 = arr.filter(n => n % 2 === 0);

    document.write("All even numbers: " + result2 + "<br>");

map – נקבל בחזרה את כל התאים שבמערך עם שינוי שאנחנו קובעים

// "Odd" או "Even" עבור כל מספר, להציג

    const result3 = arr.map(n => n % 2 === 0 ? "Even" : "Odd");

    document.write("All numbers - Even or Odd: " + result3 + "<br>");

reduce – מבצעת פעולה כלשהי שתפחית את סה"כ התאים במערך

  // הצגת סכום הפריטים

    const result4 = arr.reduce((sum, n) => sum + n, 0);

    document.write("Summary: " + result4 + "<br>");

* הפקודה צריכה לקבל שני משתנים – 1. משתנה אותו אנחנו מגדירים עם ערך ראשוני (sum) 2. משתנה שמגדיר את התאים במערך (n)

את אותם פקודות אפשר לבצע גם על מערך של אובייקטים

# Stateless Functional Component (SFC)

* לפעמים נרצה לבנות component כ פונקציה ולא ב class
* component כאלה לרוב יהיו פשוטים (לדוגמה Header) שאינם מכילים הרבה מידע
* ל components כאלה לא יהיה data פנימי (state) אבל הם כן יכולים לקבל props

צורת הכתיבה:

export const Header: React.SFC = () => {

  return (

    <div className="header">

      <h1>NorthWind Website</h1>

    </div>

  );

};

* במקום class נגדיר משתנה
* במקום JFX.Elment נקרא לפונקציה React.SFC

#### Lesson 35 - Routing, Props, Children

# Routing

* מנגנון המאפשר לנו להציג components כתלות ב route – נתיב פנימי באתר, כל מה שבא אחרי הכתובת הראשית של האתר (אחרי (/

לדוגמה במקרה שלנו הכתובת הראשית היא [http://localhost:3000](http://localhost:3000/home)

* הכתובת <http://localhost:3000/home> לדוגמה תהווה routing לדף הבית
* בכל פרויקט נצטרך להתקין תוסף שמאפשר לנו להתשמש במנגנון ה routing על ידי הפקודה npm i react-router-dom @types/react-router-dom
* התוסף נותן לנו תגים איתם אנחנו יוצרים את מנגנון ה routing ב Layout וב Menu.

|  |  |
| --- | --- |
| תגיות ב layout |  |
| * תגית שעוטפת את כל האלמנטים שיש לנו ב layout (נכתבת מעל ה header ומתחת ל footer) | *</BrowserRouter>*  *<BrowserRouter>* |
| * תגית שעוטפת את הcomponent אותם אנחנו רוצים לנתב * נכתבת ב main | *<Switch>*  *</Switch>* |
| תגית שמכילה את ה component אותו אנחנו רוצים לנתב  נכתבת כך: Route path="/home" component={**Home**} exact>> | *<Route>* |
| תגית שמפנה אותנו ל component שאנחנו רוצים שיופיע כאשר מגיעים לכתובת הראשית ((/  נכתבת כך: <Redirect from="/" to="/home" exact> | *<Redirect>* |
| תגיות ב Menu |  |
| נכתוב אותו בתוך התגית בצורה הבאה  <<NavLink activeClassName="active-route" to="/home" exact>**Home**</NavLink   * activeClassName – class שדרכו נשפיע על ה css של הלינק אחרי שלחצנו עליו | *NavLink* |

* לבסוף, אנחנו צריכים לקחת בחשבון מקרה בה המשתמש בטעות גלש לנתיב שאין לו תוכן.
* ניצור component מסוג PageNotFound ונפנה אותו אליו על ידי הניתוב

<Route component={**PageNotFound**} exact>

# Props

Props הוא data של component שאנחנו רוצים שיתקבל מ parent component.

* לדוגמה בכיתה – יצרנו component שנקרא thumbnail שמכיל תמונה ואנחנו רוצים להציג אותה בכמה component אחרים.
* את הערכים ל url, גובה והאורך ה thumbnail יקבל מאותם components.

**איך עושים את זה ?**

* יוצרים Interface מסוג Props

interface ThumbnailProps {

  imageSource: string;

  alt?: string; //optinal pramater

}

* כאשר נגדיר את ה component נגדיר את המשתנים שלו כ props

      <img

          src={this.props.imageSource}

          alt={this.props.alt || "picture"}

        ></img>

* כאשר נכתוב את ה component שייצרנו ב component אחר נכניס את הערכים שאנחנו רוצים.

<Thumbnail imageSource="assets/images/logo.jpg"  alt="Northwind\_logo"/>

אם נרצה להשאיר אפשרות לא להכניס משתנה מסויים נגדיר אותו כמשתנה אופציונאלי – נכתוב אחריו סימן שאלה

* הגדרה כזו לא מחייבת אותנו להכניס לו ערך כאשר נרצה להציג אותו.
* יכולת נוספת היא שאנחנו יכולים לתת יותר מערך אחד

imageHeight?: number; // Optional Parameter

height={this.props.imageHeight || 50} />

# children

* דרך נוספת להעביר data מ component אחד לשני.

children יעבוד רק בתוך תגיות סגורות (<h1></h1>)

**איך זה עובד?**

יוצרים component לדוגמה SubHeader

* בתוך אלמנט ה div הראשי יש תגית מכילה (h2)
* בתוכו כותבים {this.props.children}
* אין יצירה של Interface או constructor!

   <div className="subHeader">

        <h2>

          {this.props.children}

        </h2>

      </div>

בתוך components אחרים אנחנו כותבים את ה component שייצרנו עם תוכן

        <SubHeader>About The Aouther</SubHeader>

* במקרה הזה components שונים יכילו sub header עם אותו css אך עם תוכן שונה.

#### Lesson 36 - Create API Server, Fetch Products, Child to Parent Communication

# Lifecycle Hooks

* פונקציות שמופעלות בזמנים מוגדרים מראש

1. Mounting – לדוגמה: 1. didmount() – נקראת אחרי שה component נבנה (אחרי render)

2. willmount() – ה component עתיד להבנות

1. Unmounting – 1. willUnmount() – ה component עתיד להיהרס (אנחנו גולשים מה component הנוכחי ל component אחר)

# Modal

* קבצי ts שאנחנו יוצרים שמכילים אך ורק מידע
* בתוך תיקיית src יוצרים תיקייה שנקראת Modal בה ניצור ts שונים

# Create API server

* מורידים באופן חד פעמי ספרייה של שרתי npm כדי להרים שרת מהיר לבדיקות. -
* פותחים ב cmd את התיקייה שמכילה את קובץ Jason
* מכניסים את הפקודה - npm install -g json-server פותחת port חדש – json-server json.(nameOfFile) –port3002
* בזמן העבודה חלון ה cmd צריך להישאר פתוח

# fetch Products

* הבאת מידע משרת לתוך ה component שלנו על ידי הפונקציה fetch()

public componentDidMount(): void {

    fetch("http://localhost:3002/products")

      .then(response => response.json())

// the left products is the response, the right products is a class in the modal

      .then(products => this.setState({ products }))

      .catch(err => alert(err.message));

  }

* פונקציה fetch() מקבלת אליה url אליו אנחנו רוצים לגלוש
* הפונקציה מחזירה promise אותו אנחנו צריכים לעבד:

1. להמיר את ה Response ל json
2. להכניס את ה response לתוך class products שייצרנו

# Child to Parent Communication

Parent component

props

* העברת data מ child component ל parent component.

functions

Child component

* העברת data מתבצעת על ידי פונקציות
* העיקרון בו מעבירים את המידע – כותבים ב child פונקציה שמקבלת לתוכה את ה data אותו אנחנו רוצים להעביר. הפונקציה תעבוד רק אם היא קבלה פונקציה מה parent אליו אנחנו רוצים להעביר את ה data

לדוגמה בכיתה – העברת data מ Thumbnail אל Products –

* אנחנו רוצים להעביר את ה url (Image Source) שנמצא ב thumbnail ל Products.

**Thumbnail (child component)**

1. נגדיר property שמקבל לתוכו פונקציה שמקבלת את ה imageSource

  enter?(imageSource: string): void

* הפונקציה המתקבלת היא prop – היא data שמתקבל מה parent component.
* סימן שאלה מציין ש property מסוג enter לא חייב להיכתב כאשר אנחנו קוראים ל thumbnail

1. נכתוב פונקציה שמגדירה מתי הפונקציה שנכניס ל enter תעבוד:

  public userEntersMe = () => {

    if(this.props.enter) {

      this.props.enter(this.props.imageSource)

    }

  }

* הפונקציה הזו אומרת – רק אם ה parent component הכניס פונקציה לתוך property enter תשלח לאותה פונקציה את imageSource

1. נגדיר אירוע בו userEnterMe() תפעל

<div className="thumbnail" onMouseEnter={this.userEntersMe}

Products (parent component)

1. נגדיר state שיקבל לתוכו את ה data שנקבל מה child

  hoverImage: string;

  hoverImage: ""

1. נכתוב פונקציה שתבצע פעולה ב data שאנחנו מקבלים (binding)

 private disolayPewview = (imageSource: string) => {

    this.setState({hoverImage: imageSource});

}

* displayPreview() – מקבלת את המידע שנשלח מ ImageSource - child (prop) ומעבירה אותו למשתנה לוקאלי (state) שיצרנו ב parent (hoverImage)

1. בתוך Thumbnail שנמצא ב Product נכניס ל enterאת הפונקציה displayPreview()

 <Thumbnail

     imageSource={`assets/images/products/${product.id}.jpg`}

     alt="food\_logo"

     enter={this.disolayPewview}/>

* כאשר נעשה כניסה עם העכבר (OnmouseEnter) לתוך ה thumbnail הפונקציה userEntersMe ()  תופעל
* היא תבדוק האם יש בתוכה פונקציה – במקרה הזה כן – ותקרא לה
* הפונקציהdisplayPreview() תופעל, תקבל לתוכה את ImageSource ותעדכן אותו ב state של ה parent

1. לבסוף - כדי להציג את התמונה נפתח תגית img עם המשתנה hoverImage שמקבל אליו את ה url

<img className="preview" src={this.state.hoverImage}/>

#### React - Route Parameters, Forms

# Route Parameters

* האפשרות להציג דף לפי ניווט ספציפי ?

**איך אנחנו עושים את זה ?**

1. יוצרים component אליו נגיע (Details)
2. בדף ה Layout יוצרים route ל component שיצרנו

<Route path="/products/:prodID" component={Details} exact/>

1. בחלק של ה Path נוסיף אחרי הדף ממנו אנחנו רוצים לנווט (במקרה הזה products) שלושה דברים: 1. / - מצייןroute נוסף

2. : מציין ניווט לכל מה שיבוא אחריו

3. משתנה כללי (prodID) שיכיל את המידע של הדף אותו אנחנו רוצים לנווט

* מה שהפקודה הזו אומרת – נווט ל Details כאשר אחרי Products/ אתה מקבל משהו.

ב Details אנחנו רוצים להציג מידע של מוצר בודד

1. מבצעים קריאת ajax לקובץ Json שייצרנו עם כמה שינויים

  public componentDidMount(): void {

    const prodID = this.props.match.params.prodID;

    // the url is for a specifc product

    const url = `http://localhost:3002/products/${prodID}`;

    fetch(url)

      .then(response => response.json())

      .then(product => this.setState({ product }))

      .catch(err => alert(err.message));

  }

1. נכתוב את הפקודה הבאה -

const prodID = this.props.match.params.prodID;

* על ידי הפקודה הזו אנחנו מכניסים לתוך המשתנה הכללי שהגדרנו ב route של details (prodID) את המילה שנקליד אחרי ה product/

1. את הקריאה של הjson נבצע לפי אותו משתנה כללי (שמצביע על מוצר בודד)

const url = `http://localhost:3002/products/${prodID}`;

מקובץ ה Json אנחנו מקבלים מידע על מוצר בודד = אובייקט ואותו נרצה להציג בדף:

1. נפתח state מסוג class ה Product שיצרנו

  product: ProductModel;

1. ב constructor אנחנו רוצים לאתיל את האובייקט אבל ללא ערכים ראשונים ולכן נאתחל אותו כ null

export class Details extends Component<any, DetailsState> {

  public constructor(props: any) {

    super(props);

    this.state = {

      product: null

    };

  }

1. כדי לא לקבל שגיאה צריך לשנות את ההגדרות של הקובץ שלנו (tsconfig.json) כדי שלא ייראה שגיאה על null

"strictNullChecks": false

כדי להציג את המידע צריך לבצע כמה טריקים:

1. כיוון שהגדנו את האובייקט שלנו כ null כאשר המערכת תרצה להציג את המידע היא תביא לנו שגיאה.

* לכן אנחנו רוצים שמערכת תציג את המידע רק אחרי שהוא מתקבל מהשרת. כדי לעשות את זה נשתמש ב short circuit + conditional rendering – רק אם המידע קיים תציג אותו

{this.state.product && (

     <React.Fragment>

         <p> Name: {this.state.product.name}</p>

         <p> Price: {this.state.product.price}</p>

         <p> Stock: {this.state.product.stock}</p>

<img src={`/assets/images/products/${this.state.product.id}.jpg`}  width="100px" height="100px"></img>

            <br></br>

            <br></br>

            <NavLink to="/products" exact>Back To Products</NavLink>

          </React.Fragment>

        )}

1. בנוסף, conditional rendering עובד רק על תגיד בודדת. במקום לעטוף את כל תגי p ב div אנחנו עוטפים אותם בתגית ייעודית של react – React. Fragment

#### React - Forms, Validation, Debugging

# Forms - הוספת מידע חדש לקובץ json

1. ניצור component חדש (Insert)
2. בדף ה Layout יוצרים route ל component שיצרנו

<Route path="/products/new" component={Insert} exact/>

* אחרי products מוסיפים new שמציין את הדף שאנחנו רוצים
* חשוב מאוד – בסידור ה route ב Layout הניווט הזה צריך להיות ממוקם לפני הניווט

ל Detail – מנגנון ה route עובד מלמעלה למטה, הוא מחפש האם הניווט שהוזן לו מתאים ואז שולח אותנו לדף

* מאחר שבניווט ל details הגדרנו ניווט לכל דבר שמגיע אחרי Products (על ידי שימוש **:**( אם הניווט החדש יהיה ממוקם אחריו לא נוכל להגיע אליו.

1. ב component שיצרנו

* יוצרים form ובתוכו תיבות Input כדי לקבל את המידע החדש

<form>

input type=

input

<input

        <button type="button" onClick={this.forminfo}>Add</button>

</form>

* ניצור state לפי אובייקט ה Products שיש לנו אליו נכניס את ה data החדש.

interface InsertState {

  product: ProductModel;

}

* ב constructor אנחנו רוצים ליצור אובייקט חדש

  public constructor(props: any) {

    super(props)

    this.state = {

      product: new ProductModel(),

    }

  }

* כדי לא לקבל שגיאה שאנחנו צריכים להכניס ערכים לתוך ה constructor נגדיר אותם ב optional parameters על ידי הוספת סימן שאלה (?)

**שלב הבא – עדכון המידע מה form לתוך האובייקט שלנו על ידי two ways binding.**

* נזכיר – two way binding אנחנו קושרים הין מידע שמתקבל מתיבות קלט (Input, checkbox ו textarea) למשתנה בתוך ה class.

1. בתוך תיבות ה Input נגדיר משתנה מתוך אובייקט ה Product שיצרנו

<input value={this.state.product.name || ""}onChange={this.setName}></input>

* נגדיר גם פונקציה שתירה כל פעם שאנחנו משנים את הערך ב input (onChange)

1. נכתוב את הפונקציה – היא מקבלת אובייקט שמכיל מידע מהאירוע (SyntheticEvent)

 private setName = (args: SyntheticEvent) => {

    const name = (args.target as HTMLInputElement).value;

    const product = { ...this.state.product }; // make new object like an object we awant

    product.name = name;

    // setState put the local product in the global product

    this.setState({ product });

  };

* את המידע שמתקבל מהאירוע- (args.target as HTMLInputElement).value אנחנו רוצים להכניס לתוך אובייקט product של ה class
* אנחנו לא יכולים לבצע את זה ישירות – מה שאנחנו עושים זה ליצור אובייקט לוקאלי בפונקציה שזהה לאובייקט הגלובלי ב class (על ידי הפקודה …this.state.product) ואליו אנחנו מכניסים את המידע מהאירוע
* נשתמש בפונקציה setState כדי לעדכן את המידע מהאובייקט הלוקאלי (product) לתוך האובייקט הגלובלי (product) - כיוון שלשניהם אותו שם אפשר לכתוב רק את אחד מהם.

**שלב הבא - שליחת המידע ל json על יד פונקציית fetch**

1. את בקשת fetch נכניס תחת פונקציה אותה נחבר לאירוע click של ה button

<button  type="button" onClick={this.formInfo}>

1. נכתוב בקשת fetch.

 fetch("http://localhost:3002/products", options)

        .then(response => response.json())

        .then(product => {

          alert("Product has been added:" + product.id);

          this.props.history.push("/products");

        })

        .catch(err => alert(err.message));

* בנוסף ל url אנחנו מכניסים אובייקט שיכיל מידע לגבי הפעולות שאנחנו רוצים לבצע

    const options = {

      method: "POST",

      headers: {

        "Content-Type": "application/json",

        Accept: "application/json"

      },

      body: JSON.stringify(this.state.product)

    };

* method – איזה בקשה אנחנו רוצים לעשות (post, put, delete), הבקשה הבסיסית היא get.
* headers – הודעות שאנחנו רוצים שיוצגו?
* body – תשלח את המידע כstring?

בחזרה לבקשת ה fetch.

* then – עיבוד ה response שמתקבל לפורמט json()
* then – דחיפה (push) של המידע שמתקבל לתוך דף ה products?

## Validations – אימות המידע שמתקבל מהטופס

1. נגדיר אובייקט חדש שיכיל הודעות error עם ערך ראשוני

interface InsertState {

  errors: {

    nameError: string;

    priceError: string;

    stockError: string;

  };

}

export class Insert extends Component<any, InsertState> {

  public constructor(props: any) {

    super(props);

    this.state = {

      errors: {

        nameError: "\*",

        priceError: "\*",

        stockError: "\*"

      }

    };

  }

1. נשבץ את המשתנה בתוך הטופס שלנו בתוך תגית span

{this.state.errors.priceError !== "\*" && (<span>{this.state.errors.priceError}</span>

1. בתוך הפונקציה שמעבדת את המידע מהטופס נגדיר תנאים ל error Messages

* כמו עם product כדי לעדכן את האובייקט אנחנו צריכים ליצור אובייקט error לוקאלי זהה לאובייקט error הגלובלי

    const errors = { ...this.state.errors };

    if (name === "") {

      errors.nameError = "Missing name";

    } else if (name.length > 10) {

      errors.nameError = "Name is too long";

    } else {

      errors.nameError = "";

    }

1. נכניס את האובייקט הלוקאלי ל setState כדי שהמנגנון של React יוכל להציג אותו בדף ה HTML.

this.setState({ errors });

1. אימות נוסף – אנחנו לא רוצים שהמידע יישלח במקרה ויש שגיאה

* נכתוב פונקציה שבודקת אם אין טעויות (האם התוכן של אובייקט errors הוא string ריק)

  private isFormLegal = () => {

    return this.state.errors.nameError === "" &&

     this.state.errors.priceError === "" &&

     this.state.errors.stockError === "";

  };

* נכניס את הפונקציה כ attribute ל property מסוג disabled של כפתור שלחית הטופס

<button disabled={!this.isFormLegal()} type="button"

* הפונקציה isFormLegal מחזירה true/false – במקרה של true תתאפשר לחיצה ובמקרה של false לא תתאפשר לחיצה

1. אימות נוסף שאפשר לבצע (לא חייב) הוא להכניס את isFormLegal לתוך הפונקציה ששולחת את המידע

    if(!this.isFormLegal()) {

      alert("Please correct all errors!")

      return;

    }

* במקרה ויש טעות, הפונקציה isFormLegal תחזיר false, אנחנו נכנס לif והטופס לא יישלח

1. לבסוף אנחנו רוצים לגלוש בחזרה לדף המוצרים
2. this.props.history.push("/products");

* הפקודה הזו מחקה routing.

## Delete request – מחיקה של מוצר מתוך קובץ Jason

1. ניצור קישור שמחובר לפונקציה

 <a href="javascript:void" onClick={this.deleteProduct}> Delete </a>

1. יוצרים בקשת fetch()

    const options = {

      method: "DELETE"

    };

fetch(`http://localhost:3002/products/${this.state.product.id}`, options)

      .then(response => response.json())

      .then(() => {

        alert("Product has been deleted.");

        this.props.history.push("/products");

      })

      .catch(err => alert(err.message));

  };

* ה url שאנחנו מכניסים צריך להיות למוצר ספציפי ולכן נשתמש ב Id של המוצר

* בדומה לבקשת ה Post אנחנו צריכים להגדיר אובייקט עם הפעולות שאנחנו רוצים לבצע (במקרה הזה בקשת delete)

1. לבסוף – הודעה שמוודא האם המשתמש רוצה לבצע מחיקה

    const answer = window.confirm("Are you sure?");

    if (!answer) {

      return;

    }

#### React – Lesson 39 – Redux

Redux - ארכיטקטורה שמאפשרת לנו לנהל data ברמת האפליקציה

* Redux מאפשרת לנו לקשר data שנמצא במספר components – אם component מסויים יבצע שינוי באותו data יתעדכן גם ב components האחרים
* component level state - data שנמצא ברמת ה component (כמו שעון)
* AppState – data שנמצא בכמה components.

**בניית מערכת ה redux:**

* המונחים האלו מרכיבים את מנגנון ה redux
* אנחנו נשתמש בהם גם כפקודות וכשמות ל class ולדפי ts בהם נכתוב אותם

1. AppState : class בו אנחנו מגדירים ה data הקיים ברמת האפליקציה כולה.

* יוצרים קובץ ts עם אותו שם – app-state

import { ProductModel } from "../models/product-models";

export class AppState {

  public products: ProductModel [] = [];

}

* במקרה שלנו ה data הוא מערך של ProductModel
* אנחנו צריכים לאתחל את המערך כמערך ריק (דרישה של redux)

1. ActionType: רשימה של הפעולות הניתנות לביצוע על ה-AppState.

* יוצרים קובץ ts עם אותו שם – action-type
* את ActionType אנחנו כותבים כ enum – אוסף של שמות קבועים שאי אפשר לשנות מבחוץ

export enum ActionType {

  LoadAllProducts,

  AddProduct,

  UpdateProduct,

  DeleteProduct,

  ClearAllProducts

}

* איזה פעולות אנחנו רוצים לבצע? - טעינה של מוצרים, הוספת מוצר, עדכון מוצר, מחיקת מוצר, מחיקת כל המוצרים
* בפעול בקוד, כל פעולה צריכה להיכתב כפונקציה

1. payload: המידע שאנחנו שולחים ל-AppState.

* בפועל בקוד, payload תתפקד כ argument כאשר נכתוב פונקציה ל ActionType
* ה payload ישתנה בין ActinType שונים:
* בהוספת מוצר חדש, payload יהיה המוצר החדש.
* במחיקת מוצר קיים, payload יהיה ה-ID של המוצר.
* ישנם ActionType שלא יכילו payload - במחיקת כל המוצרים

1. Action: אובייקט המכיל את הפעולה לביצוע (ActionType) + המידע שלה (payload) במידה ויש

* יוצרים קובץ ts עם אותו שם – action
* Action נכתב כ Interface

import { ActionType } from "./action-type";

export interface Action {

  type: ActionType

  payload?: any

}

* אובייקט Actionמכיל שני מרכיבים:

1. type יכיל את class ActionType (רשמת הפעולות אותם אנחנו רוצים לבצע)
2. payload - ה data שנשלח ל ActionType

* payload יכול להיות כל דבר ולכן מוגדר כany .
* בנוסף כיוון לפעמים לא נשלח אותו נסמן אותו כמשתנה אופציונאלי על ידי סימן שאלה

1. dispatch: הפעולה של שליחת Action ל-Store.

* dispatch נכתבת בקוד כפונקציה שמקבלת לתוכה את ה action

store.dispatch(action)

1. reducer: פונקציה המבצעת בפועל את ה action השונים שהגדרנו

**איך בונים את ה reducer:**

* יוצרים קובץ ts עם אותו שם – reducer

1. כאמור reducer היא פונקציה

export function reducer(oldState: AppState, action: Action): AppState {

* היא מקבלת אליה שני arguments – את ה AppState (אותו היא צריכה לעדכן) ואת Action (אותו היא תגדיר)
* ל-reducer אסור לשנות את ה-AppState המקורי, כל פעולה שנבצע על ה AppState צריכה להחזיר AppState חדש המכיל את השינוי
* לכן נקרא ל AppState הראשוני oldState ובתוך reducer ניצור AppState חדש (newState) אותו נחזיר אחרי כל פעולה

  const newState = { ...oldState }; //sperad oprator

* שימוש בspread oprator - אובייקט newState מכיל בתוכו את כל מה שהיה ב oldState.

1. נבנה את ה Action שלנו

* אנחנו משתמשים במנגנון switch אליו נכניס את אובייקט action ונעבוד עלaction.type (אליו הכנסו את (class ActionType

  switch (action.type) {

* דרך מנגנון switch ניצור case שונים – כל case מייצג ActionType

    case ActionType.LoadAllProducts:

      newState.products = action.payload

      break

* case שאומר - אם קראת ל ActionType מסוג LoadAllProducts בצע את הפעולה הבאה: לתוך אובייקט newState תכניס למערך products את ה data שקבלת (ה data מיוצג על ידי המילה payload)
* בפועל כאשר נקרא לפונקציה, ה payload שנכניס הוא רשימת המוצרים מתוך שרת.

case ActionType.AddProduct:

      newState.products.push(action.payload)

      break;

* case שאומר – אם קראת ל ActionType מסוג AddProducts : לתוך newState תוסיף למערך products את ה data שקבלת
* בפועל כאשר נקרא לפונקציה, ה payload שנכניס הוא מוצר חדש

      return newState;

  }

* switch מחזירה newState – AppState מעודכן

1. subscribe: רישום ל-Store לקבלת מידע עדכני. ברגע שהמידע ישתנה - אנו מקבלים את המידע החדש.

* כמו dispatch גם subscribe היא פונקציה

1. store: משתנה שמכיל את כל המנגנון שבנינו עד כה

* יוצרים קובץ ts עם אותו שם – store

import {createStore} from "redux";

import { reducer } from "./reducer";

import { AppState } from "./app-state";

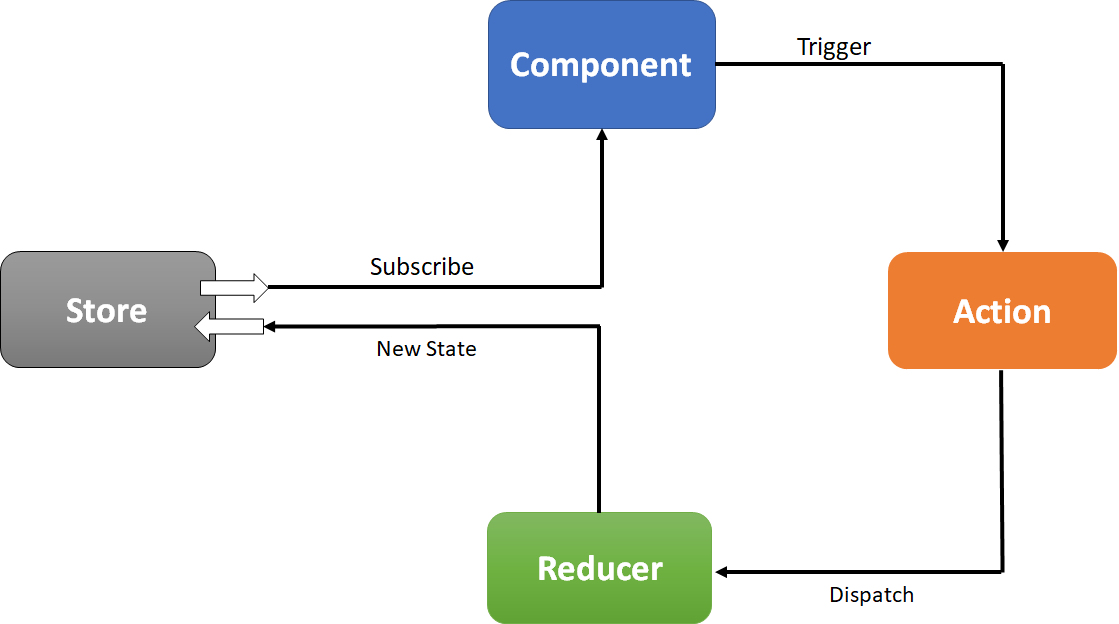
export const store = createStore(reducer, new AppState());

ממה store מורכב ?

* פונקציה - createStore() המקבלת לתוכו שני arguments:

1. פונקציית reducer – עם כל הפעולות שהגדרנו
2. אובייקט AppState חדש – ה data הכללי של האפליקציה

סכמה למנגנון redux.



component רוצה לבצע פעולה כלשהי על AppState - נשתמש ב Action.

בתוך Action נקרא ל ActionType (מאחרי הקלעים פועלת פונקציית reducer)

reducer יוצרת newState חדש שנשלח ל store ע"י הפקודה dispatch.

כדי שהמידע החדש מה store יתעדכן ב component צריך לבצע רישום ל store על ידי הפקודה subscribe

סיכום המונחים בטבלה

|  |  |  |  |
| --- | --- | --- | --- |
| **שם** | **הגדרה** | **בניית המערכת** | **שימוש בפועל** |
| AppState | המידע הקיים ברמת האפליקציה כולה. | דף ts  class |  |
| Store | המקום בו נשמר ה-AppState. | דף ts  משתנה | אובייקט |
| ActionType | מתאר את סוג הפעולה הניתנת לביצוע על ה-AppState. | דף ts  enum | Field של משתנה action |
| Payload | המידע עצמו שאנו שולחים ל-AppState. | argument | Field של משתנה action |
| Action | הפעולה לביצוע + המידע שלה | דף ts  interface | משתנה  Data Type |
| Dispatch | הפעולה של שליחת Action ל-Store |  | פונקציה |
| Reducer | פונקציה המבצעת בפועל את הפעולה הדרושה על ה- Store | דף ts  פונקציה |  |
| Subscribe | רישום ל-Store לקבלת מידע עדכני |  | פונקציה |