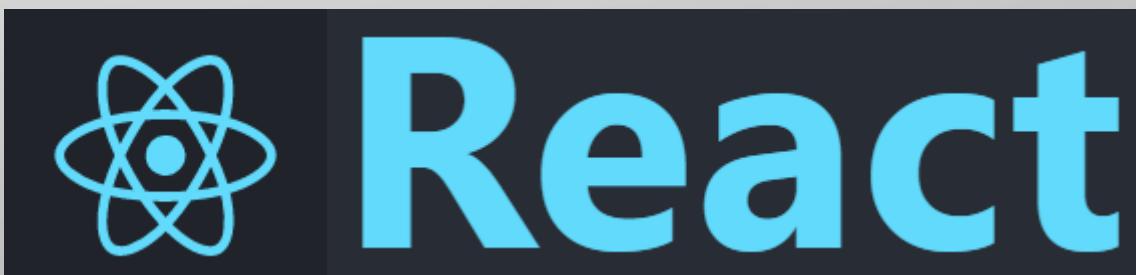


REACT

02 COMPONENTS AND PROPS

©NIR CHEN



SYLLABUS

- 01 Fundamentals And Installation
- **02 Components And Props**
- 03 State
- 04 Handling Events
- 05 SetState And Lifecycle
- 06 Lists And Keys
- 07 Forms
- 08 PropTypes
- 09 Refs
- 10 React Router v5

SPA AND JSX

- בצד ימין ליראה מהירה יותר של האפליקציה מתכנתים בSPA. כרגע ניתען רק דף אחד מלא וכל השאר אילו שינויים על ה-WDOM ישירות.
- את האלמנטים של ה-HTML מייצרים ע"י JSX כרגע שה-HTML נוצר ע"י קוד של SJ. הייצור הזה היא רק עברו ה-VIRTUAL DOM!
- המטרה של REACT היא לבחון את ההבדלים בין ה-VIRTUAL DOM לבין ה-WDOM האמתי ולרendar רק את השוני למסך. את זה REACT עושה מאוד מהר ולכן מקבלים בסופו של דבר זמן תגובה מצוין!

ELEMENT

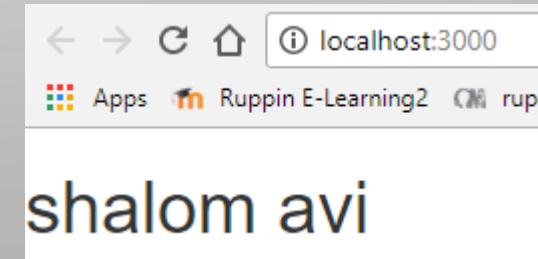
- היחידה הici קטנה נקראת אלמנט – מציגה פיסת מידע להציג על המסך

App.jsx

```
...  
const App = <h1>shalom avi</h1>;  
export default App;
```

index.js

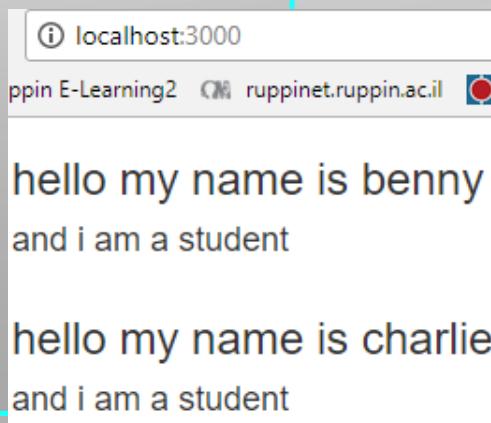
```
...  
ReactDOM.render(App,  
document.getElementById('root'));  
...
```



FUNCTIONAL COMPONENT

App.jsx

```
...  
function Student(props) {  
  //props.name="stam"; ERROR IMMUTABLE!  
  return (  
    <div className="container">  
      <h3>hello my name is {props.name} </h3>  
      <h4>and i am a student</h4>  
    </div>);  
}  
  
const App =  
  <div>  
    <Student name="benny" />  
    <Student name="charlie" />  
  </div>;  
export default App;
```



- Component – זהו המרכיב של כל אפליקציה בReact. כל קומפוננט מכיל עריכים של קלט כאשר הפלט היא פונקציה שנקראת `()` RENDER שמכילה את מה שרוצים להראות על המסך.
- בצורה הפשוטה יותר זו יכולה להיות פונקציה של JS שמקבלת אובייקט JS בשם PROPS ומחזירה אלמנטים לירידור על המסך. functional component
- **הוא ONLY READ !!! לא ניתן לשנות את הערך שלו –**
קומפוננט לעולם לא יכול לשנות את PROPS שלו!!! IMMUTABLE
- הפונקציה RETURN חייבת להחזיר אלמנט אחד בלבד שיכל להכיל כמה אלמנטים שרוצים.
- נהוג להשתמש באוט גודלה עבור קומפוננט נתן ליצור קומפוננטים כיצירה של אלמנט HTML.
- ניתן להעביר מידע כ-ATTRIBUTE ל-PROPS של הקומפוננט.
- ניתן להשתמש ב{} כדי לכתוב קוד, או לגשת ל-PROPS

SYLLABUS

- 01 Fundamentals And Installation
- 02 Components And Props
- **03 State**
- 04 Handling Events
- 05 SetState And Lifecycle
- 06 Lists And Keys
- 07 Forms
- 08 PropTypes
- 09 Refs
- 10 React Router v4

©NIR CHEN

CLASS COMPONENTS - STATE

App.jsx

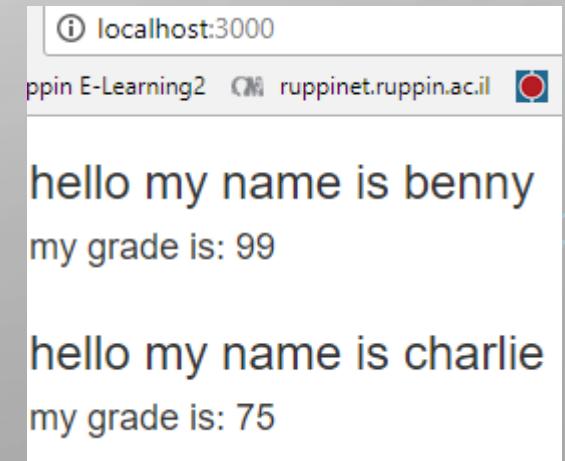
```
...
class Student extends React.Component {
  constructor(props) {
    super(props);
    this.state = { grade: Math.round(Math.random() * 40 + 60 )};
  }
  render() {
    return (
      <div className="container">
        <h3>hello my name is {this.props.name} </h3>
        <h4>my grade is: {this.state.grade}</h4>
      </div>
    );
  }
}
// ©NIR CHEN
```

כאשר יש צורך בשינוי והחזקת משתנים בתוך הקומponent יש צורך ליצור **component class** המכיל STATE.

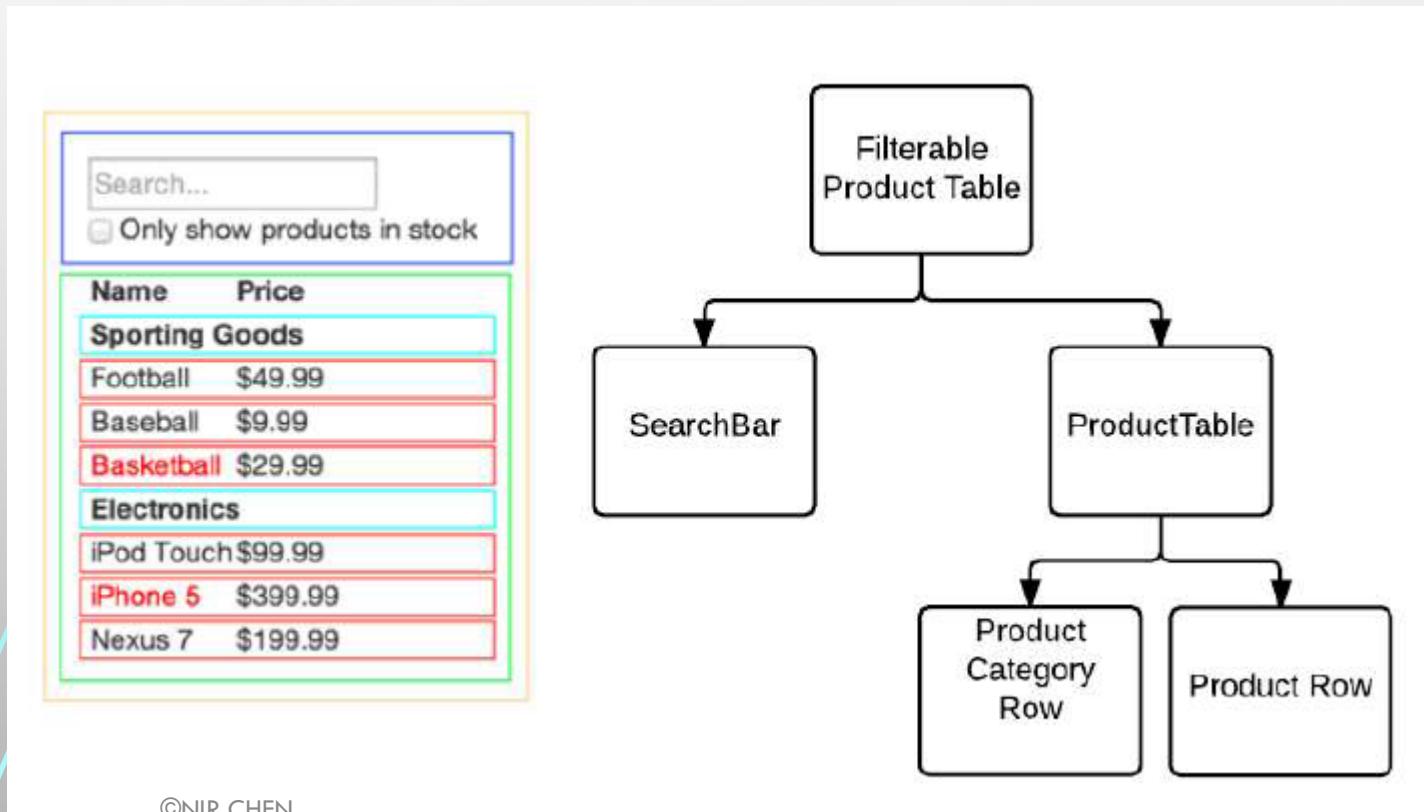
- ייחdet מידע שניית לשנות מתוך הקומponent.
- מעין שדה פרטי של מחלוקת
- מוגדר במבנה

App.jsx

```
המשר...
const App =
<div>
  <Student name="benny" />
  <Student name="charlie" />
</div>;
export default App;
```



COMPONENTS



- בראקט הכל זה קומפוננטות
- חלקו "טייפשות" – ללא STATE רק **functional components**
- "חכמת" – עם STATE **class components**

SYLLABUS

- 01 Fundamentals And Installation
- 02 Components And Props
- 03 State
- **04 Handling Events**
- 05 SetState And Lifecycle
- 06 Lists And Keys
- 07 Forms
- 08 PropTypes
- 09 Refs
- 10 React Router v4

©NIR CHEN

App.jsx

```
...  
function Student(props) {  
  //props.name="stam"; ERROR IMMUTABLE!  
  var number;  
  function txtChenged(e) {  
    number = e.target.value;  
  }  
  
  function btnClicked() {  
    alert(number);  
  }  
  
  return (  
    <div className="container">  
      <h3>hello my name is {props.name} </h3>  
      <h4>and i am a student</h4>  
      <button  
        onClick={btnClicked} className="btn btn-default">  
        show number</button>  
      <input type="text"  
        placeholder="insert your number"  
        onChange={txtChenged} />  
    </div>);  
}
```



EVENT HANDLING – IN FUNCTIONAL COMPONENTS

- בראקט שמות האירועים הם camelCase
- הקריאה לפונקציה נעשית ע"י השימוש ב{} ובשם הפונקציה בתוך הסוגרים.
- הגדרת הפונקציה נעשית כרגע בסע ע"י המילה function

```
const App =  
  <div>  
    <Student name="benny" />  
    <Student name="charlie" />  
  </div>;  
export default App;
```

App.jsx

```
...  
class Student extends React.Component {  
  constructor(props) {  
    super(props);  
    var rnd = Math.round(Math.random() * 40 + 60);  
    this.state = {  
      grade: rnd,  
      orgGrade: rnd,  
    };  
    //alert('constructor ' + this.props.name);  
    this.number = 8; ←  
    this.txtChanged = this.txtChanged.bind(this); ←  
    this.btnClicked2 = this.btnClicked2.bind(this); ←  
  }  
  txtChanged(e) {  
    this.number = e.target.value;  
  }  
}
```

EVENT HANDLING – IN CLASS COMPONENTS

```
btnClicked1() {  
  alert('your number is: ' + this.number);  
  //alert('btn ' + this.props.name);  
}  
}
```

```
btnClicked2() {  
  alert('your number is: ' + this.number);  
  //alert('btn ' + this.props.name);  
}  
}
```

```
btnClicked3 = () => { ←  
  alert('your number is: ' + this.number);  
  //alert('btn ' + this.props.name);  
}  
}
```

```
btnClicked4() {  
  alert('your number is: ' + this.number);  
  //alert('btn ' + this.props.name);  
}  
}
```

- ישן 4 אופציות לחבר אירועים ב CLASS COMP
- אופציה 3 היא המומלצת ביותר!

המשך בעמוד הבא...

EVENT HANDLING – IN CLASS COMPONENTS CONT'

```
<input type="button"  
value="show number"  
onClick={this.btnClicked1.bind(this)}  
className="btn btn-default" />
```

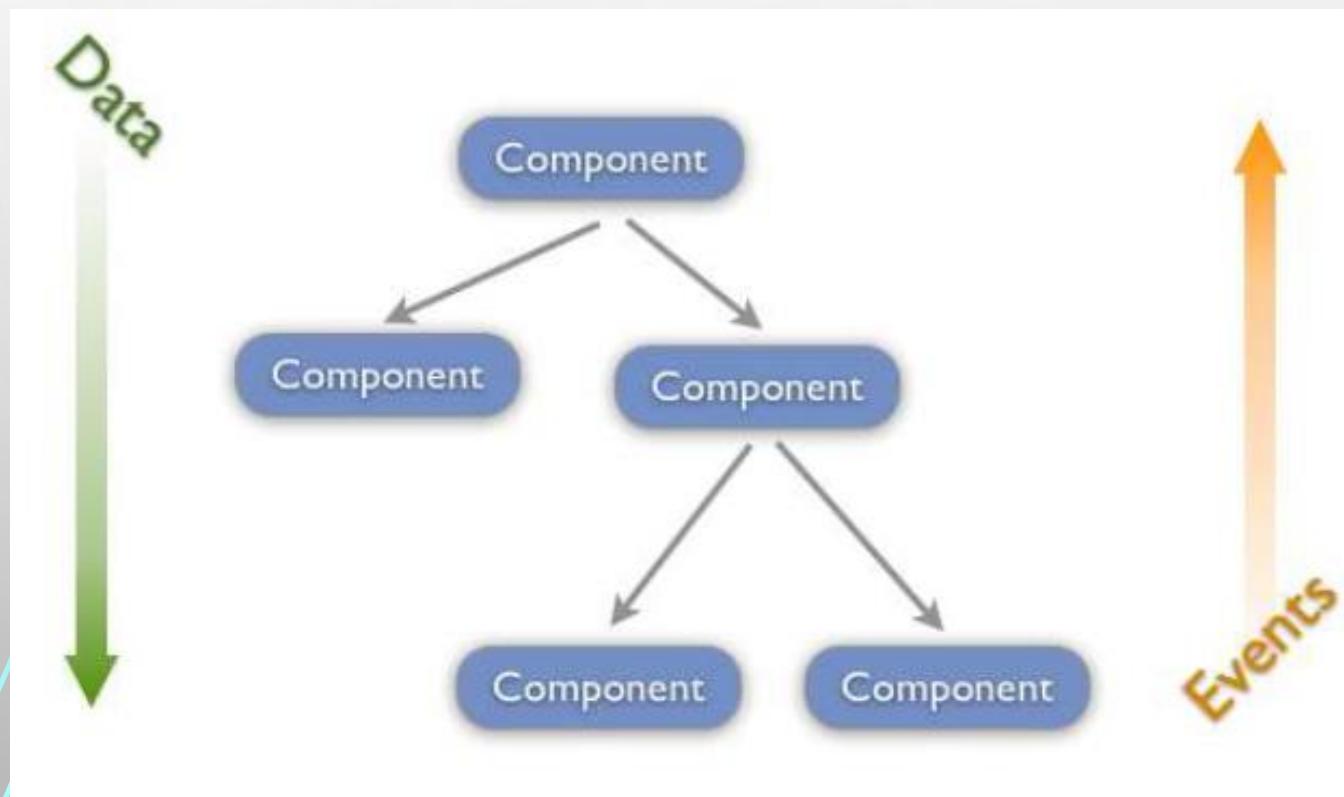
```
<input type="button"  
value="show number2"  
onClick={this.btnClicked2}  
className="btn btn-default" />
```

```
<input type="button"  
value="show number3"  
onClick={this.btnClicked3}  
className="btn btn-default" />
```

```
<input type="button"  
value="show number4"  
onClick={()=> this.btnClicked4()}  
className="btn btn-default" />
```

```
<input type="text"  
placeholder="insert your number"  
onChange={this.txtChanged} />  
</div>
```

“TOP-DOWN” OR “UNIDIRECTIONAL” DATA FLOW



- קומפוננט "הורה" יכול להעביר את המידע שבו לקומפוננט "ילד" ע"י PROPS.
- כל קומפוננט בהיררכיה יכול להיות רק CLASS או FUNCTIONAL.
- ע"י PROPS המידע זורם רק בכיוון אחד – למטה.

FROM CHILD TO PARENT DATA FLOW

ChildComponent.jsx

```
export default class ChildComponent extends Component {  
  btnClicked = ()=> {  
    this.props.sendData('from child ' +new Date().getSeconds());  
  }  
  
  render() {  
    return (  
      <div style={{border: "solid 2px black", width:"300px", padding:10}}>  
        CHILD<br/>  
        <button onClick={this.btnClicked}>PUSH DATA TO PARENT</button>  
      </div>  
    );  
  }  
}
```

- כאשר רוצים להעביר מידע מהבן לאבא עושים זאת ע"י אירועים

App.jsx

```
getData = (data)=>{  
  alert("alert from parent with child data: "+ data);  
}  
...  
<br/>  
<ChildComponent sendData={this.getData}>  
</div>
```

SYLLABUS

- 01 Fundamentals And Installation
- 02 Components And Props
- 03 State
- 04 Handling Events
- **05 setState And Lifecycle**
- 06 Lists And Keys
- 07 Forms
- 08 PropTypes
- 09 Refs
- 10 React Router v4

©NIR CHEN

STATE CHANGING

```
btnClicked3 = () => {
  alert('your number is: ' + this.number);
  //this.state += this.number; //ERROR try to change the state directly
  //opt1
  this.setState({grade: this.state.grade + parseInt(this.props.bonus)});
  //op2 -better because deals with async - and preferred
  this.setState((prevState, props) =>
    ({ grade: prevState.grade + parseInt(this.props.bonus)}));
  //opt3 - better because deals with async
  this.setState(function (prevState, props) {
    return {
      grade: prevState.grade + parseInt(props.bonus)
    };
  });
}
```

©NIR CHEN

- כאשר רוצים לשנות את ה-STATE חייבים לעשות זאת ע"י שימוש בפונקציה `setState` אחרת (אם מנסים לשנות את ה-STATE ישירות) לא יקרה רינדור של ה-WDOM חדש ו开会 לא נוכל לראות את השינוי. `SetState` קוראת `l()` ל-`RERENDER` לרוץ ואז נעשית בדיקה מה הת:redesh WDOM.
- אופציה 1: לשנות עבור מקרים בהם לא מעדכנים את הערך כ תלות מהערך הקודם. (פה בדוגמה זה כן תלות מהערך הקודם אבל... סעיף הבא!)
- יש שתי אופציות עבור מקרים של שינוי כ תלות מהערך הקודם. אופציה 2-3 חובה כי יכול להיות asynchronously. גם REACT יכול להריץ כמה שינויים בבאת אחת.

LIFE CYCLE

- - Constructor – רץ פעם אחת. מכיל את האיתחול של ה STATE. אסור לשנות את ה STATE ע"י SETSTATE! לא קיים עדין ב DOM.
- Render – מציירת על המסר את הקומפוננט, קיים ב DOM
- componentDidMount – פה אפשר לשנות את ה STATE ע"י SETSTATE כי הקומפוננט כבר צויר למסר וקיים ב DOM ו- SETSTATE גורם לrintce של RENDER שוב لكن צריך להיות קיים כבר ב DOM. פה גם נלך לקחת מידע מהשרת אם צריך ...

COMPONENTDIDMOUNT

- אירוע שקורה רק פעם אחת לאחר שהкомпонент מרים את הפונקציה RENDER בפעם הראשונה ומיצר את ה-WoM.
- למשל אם רוצים לשנות בהתחלה את ה-STATE ע"י קוד. לא ניתן לעשות זאת בבנאי כי אחרת תהיה לו לא אינסופית. אז ניתן לעשות זאת בcomponentDidMount.

App.jsx

```
...
componentDidMount() {
  this.setState((prevState, props) =>
    ({ grade: prevState.grade + parseInt(this.props.bonus) }));
}
```

COMPONENTWILLUNMOUNT

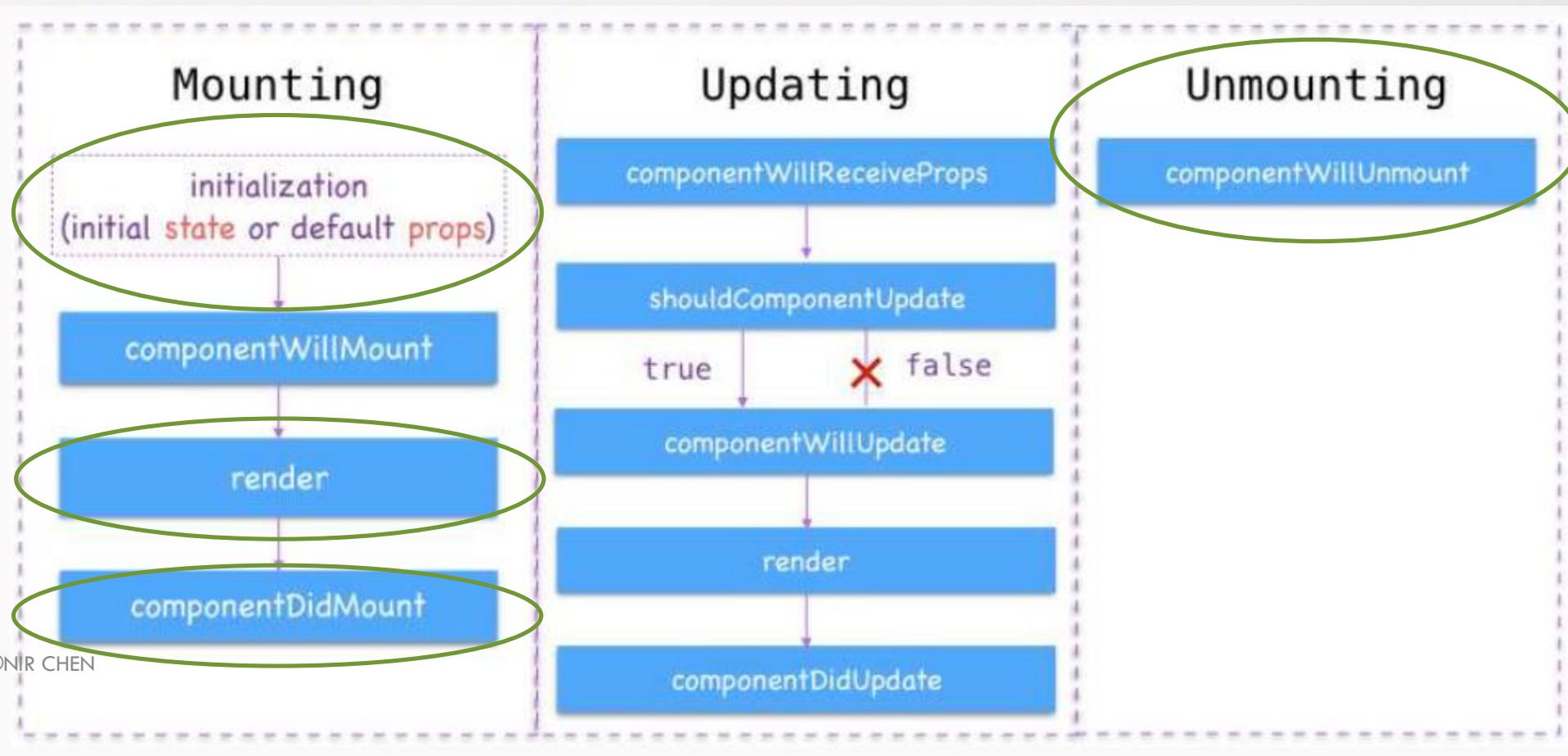
- אירוע שקוֹרֶה רָק פָּעֵם אַחֲת לִפְנֵי שהkomponent מפונה מהזיכרון.
- למשל אם לנוקות\לפנות משאבי אז ניתן לעשות זאת בcomponentDidMount

App.jsx

```
...
componentWillUnmount() {
...
}
```

©NIR CHEN

LIFECYCLE



SYLLABUS

- 01 Fundamentals And Installation
- 02 Components And Props
- 03 State
- 04 Handling Events
- 05 SetState And Lifecycle
- **06 Lists And Keys**
- 07 Forms
- 08 PropTypes
- 09 Refs
- 10 React Router v4

©NIR CHEN

JS MAP FUNCTION

- **Map** עוברת על מערך ופועלת על כל אחד מהאיברים ע"פ פונקציה מボוקשת.

```
render() {  
...  
var numbers = [1, 4, 9];  
var roots = numbers.map(Math.sqrt); ←  
console.log(roots);  
  
var powers = numbers.map(function (num) { return num * num; }); ←  
console.log(powers);  
  
var names = ["avi", "benny", "charlie"];  
var helloNames = names.map((name) => { ←  
  alert(name);  
  return name + "!"  
});  
console.log(helloNames);
```

- ניתנת לשימוש במספר צורות, לדוגמה

```
▶ (3) [1, 2, 3]  
▶ (3) [1, 16, 81]  
▶ (3) ["avi!", "benny!", "charlie!"]
```

LIST AND KEY

```
constructor(props) {  
...  
this.numbers = [1, 2, 3, 4, 5];  
this.listNumbers = this.numbers.map((number) =>  
  <a href="#" className="list-group-item" key={number}>{number * 2}</a>  
);  
  
this.list = ["avi", "benny", "charlie"];  
this.listAsLi = this.list.map((name, index) =>  
  <a href="#" className="list-group-item" key={index}>{index +": hello " + name + "!"</a>  
);  
...  
render(){...  
<p style={{ fontWeight: "bold", margin: 10 }}>names list:</p>  
<div className="list-group" style={{ width: "20%" }}>{this.listAsLi}</div>  
<p style={{ fontWeight: "bold", margin: 10 }}>numbers list:</p>  
<div className="list-group" style={{ width: "20%" }}>{this.listNumbers}</div>
```

כasher yozrim reshima yis zoruk letat lcel aibar
mapach cmachzot yichodit bcdi shriakut yidu
lezhot otto meshar haibrim ubor hosofo,
udcon vmechika.

- Zaricim lehiot yichodim rak bator reshima
spatzifit vla bcel haumod.
- La mouberim c'sops PROPS
- Nitn leusot zat l'mashl cr:

names list:
0: hello avi!
1: hello benny!
2: hello charlie!

numbers list:
2
4

SYLLABUS

- 01 Fundamentals And Installation
- 02 Components And Props
- 03 State
- 04 Handling Events
- 05 SetState And Lifecycle
- 06 Lists And Keys
- **07 Forms**
- 08 PropTypes
- 09 Refs
- 10 React Router v4

©NIR CHEN

FORM CONTROLS

- בטפסים נמצאים פקדים אשר דרכם יכול המשתמש לנוהל מיידע.
- אותם פקדים ניתנים לשינוי ע"י המשתמש
- בצד לעשوت שימוש במידע זהה בקוד אנחנו חייבים לחבר את הפקד ל-STATE. זאת נעשה ע"י למשל `value={this.state.userName}`
- כאשר חיברנו פקד ל-STATE לא נראה אותו משתנה אלה אם כן תרצו הפונקציה RENDER אשר רצתה רק כאשר נעשה שינוי בSTATE כפי שלמדנו בפרק 05 SetState And Lifecycle. בצד לעשות את אותו שינוי נוסף לפקד למשל את `onChanged={this.onTextChanged}` ובפונקציה נשימוש לمثال ב`this.setState({userName: event.target.value});`

המשך בעמוד
הבא...

```

constructor(props) {
  super(props)
  this.state = {
    userName: "insert your name"
  }
}

onTextChanged = (event) => {
  this.setState({userName: event.target.value});
}

btnHelloUser = () => {
  alert('hello ' + this.state.userName);
  alert('hello ' + this.state.stam); //undefined
}

render() {
...
<form>
  not connected to state input <input type="text" name="stam"/><br/>
  connected to state input WO onchange <input type="text" name="userName" value={this.state.userName}><br/>
  connected to state input With onchange <input type="text" name="userName" value={this.state.userName}>
    onChange={this.onTextChanged}

```

FORM CONTROLS – INPUT TEXT

not connected to state input avi

connected to state input WO onchange nir

connected to state input With onchange nir

hello user

Alert undefined

לא ניתן לשינוי
ע"י המשתמש

ניתן לשינוי ע"י
המשתמש
Alert nir

שליטה בהזנת נתונים

- מכיון שאנחנו מקבלים את שינוי המשתמש אבל מזינים בעצמו את ה STATE, ניתן לשנות אותו באופן שטוב לנו למשל ניתן לנקח את האותיות המוכנסות לטיבת הטקסט ולהפוך אותן לגודלות שירות בהקלדה.

```
onTextChanged = (event) => {
  this.setState({userName: event.target.value.toUpperCase()});
}
```

input	<input type="text"/>
put WO onchange	NIR
put With onchange	<input type="text" value="NIR"/>

<TEXTAREA>

- נעשה שימוש ב-**VALUE** בצד ה-**CSS** להציג מלל על המסר

```
ontxtAreaChenged = (event) => {  
  this.setState({ txtArea: event.target.value });  
}  
...  
<textarea name="txtArea" id=""  
cols="20" rows="5"  
placeholder="insert the story"  
onChange={this.ontxtAreaChenged}  
value={this.state.txtArea}></textarea><br />
```

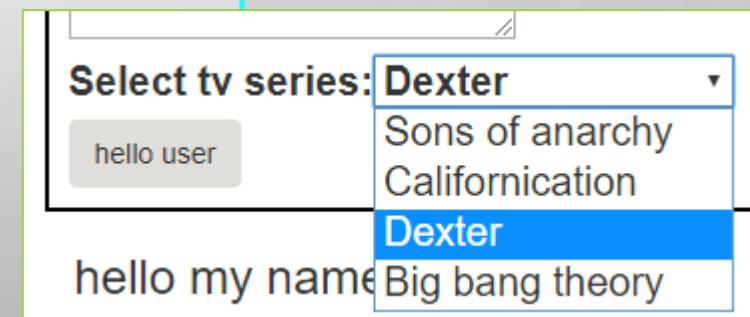
לא הגדרנו את TEXTAREA ב
CTOR בצד שמאלי לראות את ה
PLACEHOLDER

insert the story

<SELECT>

```
constructor(props) {  
    super(props)  
    this.state = {  
        userName: "insert your name",  
        tvSeries: "dexter"  
    }  
    ...  
    slctTvChange = (event)=>{  
        this.setState({tvSeries: event.target.value});  
    }  
    ...  
    <select value={this.state.tvSeries} onChange={this.slctTvChange}>  
        <option value="sons of anarchy">Sons of anarchy</option>  
        <option value="californication">Californication</option>  
        <option value="dexter">Dexter</option>  
        <option value="big bang theory">Big bang theory</option>  
    </select>
```

- שוב נעשה שימוש ב-**VALUE** כדי לבחור את האפשרות הרצויה מתוך תיבת הבחירה.



SYLLABUS

- 01 Fundamentals And Installation
- 02 Components And Props
- 03 State
- 04 Handling Events
- 05 SetState And Lifecycle
- 06 Lists And Keys
- 07 Forms
- **08 PropTypes**
- 09 Refs
- 10 React Router v4

©NIR CHEN

PROPTYPES

- ב כדי לאמת את הנתונים שmaguiim ל PROPS בזמן ריצה ניתן להשתמש ב **PropTypes**
- למשל ל COMPONENT STUDENT שלנו נוסיף בדיקת PROPS ע"י הוספה אחריו של הקוד הבא...
- לאחר שניפתח F12 בכרום יוכל לראות את השגיאות הבאות:

```
import PropTypes from 'prop-types';
...
Student.propTypes = {
  name: PropTypes.string.isRequired,
  bonus:PropTypes.number.isRequired
};
...
const App =
<div>
  <FormDemo />
  <Student name="benny" />
  <Student name="charlie" bonus="3" />
  <Student name="dora" bonus={4} />
</div>;
```

✖ Warning: Failed prop type: The prop `bonus` is marked as required in `Student`, but its value is `undefined`. in Student (at App.jsx:178)

✖ Warning: Failed prop type: Invalid prop `bonus` of type `string` supplied to `Student`, expected `number`. in Student (at App.jsx:179)

PROPTYPES

```
optionalArray: PropTypes.array,  
optionalBool: PropTypes.bool,  
optionalFunc: PropTypes.func,  
optionalNumber: PropTypes.number,  
optionalObject: PropTypes.object,  
optionalString: PropTypes.string,  
optionalSymbol: PropTypes.symbol,  
...  
optionalArrayOf: PropTypes.arrayOf(PropTypes.number),  
...  
optionalObjectOf: PropTypes.objectOf(PropTypes.number),  
...  
// You can chain any of the above with `isRequired` to make sure a warning  
// is shown if the prop isn't provided.  
// A value of any data type  
requiredAny: PropTypes.any.isRequired,  
...
```

- חשוב להשתמש ב `PropTypes` כתנה ב גדרות עצמן ב גודלה.

`.Student.propTypes`

- יש מן הגדרות אפשרויות לסוגי הנתונים ומאפייניהם

<https://www.npmjs.com/package/prop-types>

SYLLABUS

- 01 Fundamentals And Installation
- 02 Components And Props
- 03 State
- 04 Handling Events
- 05 SetState And Lifecycle
- 06 Lists And Keys
- 07 Forms
- 08 PropTypes
- **09 Refs**
- 10 React Router v4

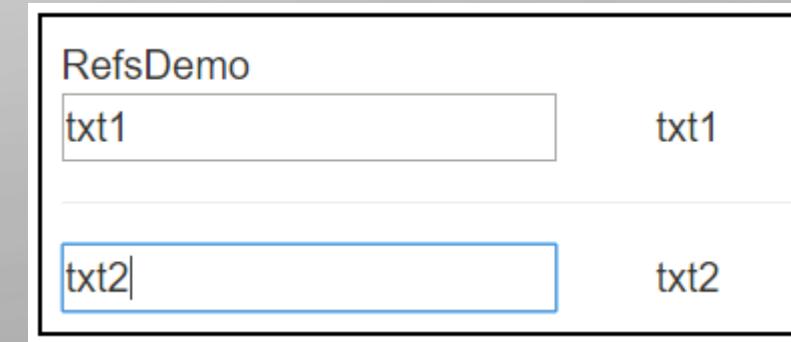
©NIR CHEN

REFS

```
TextChanged = (e) => {
  this.setState({
    txt1: e.target.value
  })
}
TextChanged2 = (e) => {
  this.setState({
    txt2: e.target.value
  })
}
```

```
...
<input type="text"
  onChange={this.TextChanged} />
<span style={{ marginRight: 50 }} />
{this.state.txt1}
<hr />
<input type="text"
  onChange={this.TextChanged2} />
<span style={{ marginRight: 50 }} />
{this.state.txt2}
```

- REF נותן לנו גישה ל-ELEMENT.DOM
- מומלץ להשתמש בזה כמה שפחות!!!
- אפשר לראות דוגמה לשימוש ב-REF ובלי REF
- פה בלי REF בעמוד הבא עם REF



```
// constructor(props) {  
//   ...  
//   this.txtInput1; ←  
//   this.txtInput2 = React.createRef(); ←  
// }  
  
TextChanged = () => {  
  this.setState({  
    txt1: this.txtInput1.value, ←  
    txt2: this.txtInput2.current.value ←  
  })  
}  
...  
<input type="text"  
  ref={(input) => { this.txtInput1 = input }} ←  
  onChange={this.TextChanged} />  
<span style={{ marginRight: 50 }} />  
{this.state.txt1}  
<hr />  
<input type="text"  
  ref={this.txtInput2} ←  
  onChange={this.TextChanged}/>  
<span style={{ marginRight: 50 }} />  
{this.state.txt2}
```

REFS- TWO VERSIONS

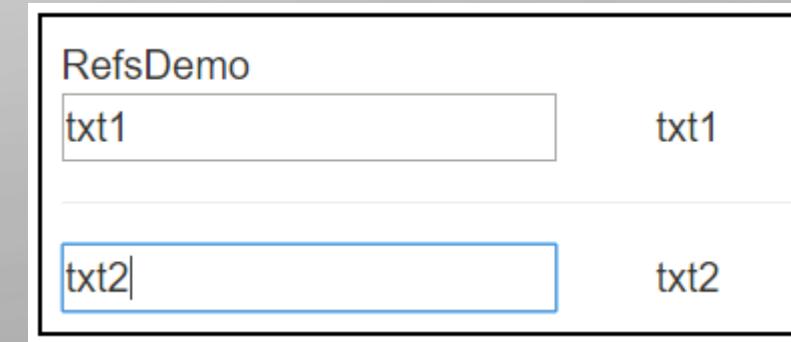
REF מען •

האתר של REACT •

There are a few good use cases for refs:

- Managing focus, text selection, or media playback.
- Triggering imperative animations.
- Integrating with third-party DOM libraries.

Avoid using refs for anything that can be done declaratively.



SYLLABUS

- 01 Fundamentals And Installation
- 02 Components And Props
- 03 State
- 04 Handling Events
- 05 SetState And Lifecycle
- 06 Lists And Keys
- 07 Forms
- 08 PropTypes
- 09 Refs
- **10 React Router v5**

REACT ROUTER

- ב-RER ממעבר לעמוד אחר לא נעשה ע"י החלפה של לעמוד ה HTML אלא כל הזמן נשארים באותו לעמוד INDEX.HTML וرك ה-WOM מתחלף.
- כך חוסכים בזמן.
- כמובן שגם החלפת חלק מעמוד במקרה לדוגמא של NAVIGATION BAR שמחליף את החלק העיקרי של העמוד – גם אז עושים שימוש ב-ROUTER שמחליף רק את ה-WOM ולא את כל העמוד.
- נעשה שימוש בעיקר בשלושה COMPONENTS - **LINK ,ROUTE ,BROWERSERROUTER**
 - Router - עוטף את כל האפליקציה ודואג למעבר העמודים INDEX.JS
 - withRouter - מייחצן את ה APP לפני ה-INDEX.JS
 - Switch - עוטף את מקומם ה ROUTE. איפה לשתול את הדפים עצמן?
 - Route - הגדרת המסלול כ - URL שמתאים למעבר לCOMPONENT מסוים
 - Link - הceptors שעלייו לוחצים בכדי לעבור למסלול מסוים

```
npm install react-router-dom --save
```

BROWSERROUTER

- בעמוד הראשי index.js נגדייר את BROWSERROUTER

```
import { BrowserRouter } from 'react-router-dom';  
  
ReactDOM.render(  
  <BrowserRouter> ←  
    <App />  
  </BrowserRouter>,  
  document.getElementById('root'));
```

```
import React, { Component } from 'react';
import { Switch, Route, Link, withRouter } from 'react-router-dom';
import Home from './Home';
import About from './About';
import MenuComponent from './MenuComponent';
```

```
class App extends Component {
  render() {
    return (
      <div>
        <Switch>
          <Route exact path="/" > ←
            <Home />
          </Route>
          <Route path="/about">
            <About />
          </Route>
          <Route path="/menu">
            <MenuComponent />
          </Route>
        </Switch> ←
      </div>
    ...
  }
}

export default withRouter(App); ←
```

ROUTER

- בעמוד הראשי למשל APP נגזר את הROUT-ים המסויימים.
- המסלול הראשון שמתאים הוא זה שנעבור עליו
- שימוש לב לשימוש במילה **exact** עבור המסלול של HOME כי אחרת זה יתאים לכל קרייה.
- בכל מקום בו נרצה לעבור ע"י קוד לעמוד אחר (ראה בהמשך) נחזיר את withRouter ע"י COMPONENT

LINK

- עכשו נוכל להשתמש ב LINK בכל מקום באתר כדי לעבור לעמוד המבוקש.
- אם היינו רוצים את התפריט בכל העמודים יכולנו לכתוב אותו פעם אחת ב APP.js. זה בד"כ מה שעושים לתפריט צד או עליון.

```
...
import {Link} from 'react-router-dom'; ←

export default class Home extends Component {
  render() {
    return (
      <div>
        <Link to="/">home</Link> | ←
        <Link to="/about">About</Link> | ←
        <Link to="/menu">Menu</Link>

        <p>home</p>
      </div>
    );
  }
}
```

URL PARAMETERS AND MATCH

- ניתן להוסיף למסלול גם פרמטרים עם ערכים משתנים.

```
<BrowserRouter>  
...  
  <Link to="/menu/1">user 1</Link><br />  
  <Link to="/menu/2">user 2</Link><br />  
  </div>  
  <div style={{ ...  
    <Route path="/menu/:userId" component={MenuUserComponent} />  
  </div>  
</BrowserRouter>
```

- בעמוד הבא ניתן לראות כיצד לחוץ את המידע מהפרמטרים החוצה.

URL PARAMETERS AND MATCH

- בכל פעם שקוראים לCOMPONENT מועבר בצורה אוטומטית לתוך הPROPS שלו פרמטר נוסף שנקרא MATCH שמכיל מידע על הROUTE בין היתר גם את השדה PARAMS שמכיל פרמטרים שהועברו בURL וכך ניתן שלוף אותם.

```
<div>
  <h3>in MenuUserComponent with <br/>
    user id: {this.props.match.params.userId}</h3>
</div>
```

©NIR CHEN

CHANGE PAGE PROGRAMMATICALLY

```
var userObj={  
    userId:this.props.userId,  
    userName: this.props.userName  
};  
  
this.props.history.push({ ←  
    pathname: '/userPage/' + this.props.userId,  
    search: '?query=abc',  
    state: {userObj : userObj }  
});
```

```
{this.props.location.state.userObj.userId}  
...  
export default withRouter(CCUser);
```

- ניתן לעבור לעמוד אחר בקוד ע"י :
- ניתן גם להעביר מידע בין העמודים ע"י ע"ו או ע"י במעבר המידע ב-state.
- ע"י state ניתן להעביר גם אובייקט שלם.