

# 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

- בכדי לגרום לריצה מהירה יותר של האפליקציה מתכנתים בSPA. כך ניטען רק דף אחד מלא וכל השאר אילו שינויים על הDOM ישירות.
- את האלמנטים של הHTML מייצרים ע"י JSX כך שהHTML נוצר ע"י קוד של JS. היצירה הזו היא רק עבור ה VIRTUAL DOM!
- המטרה של REACT היא לבחון את ההבדלים בין ה VIRTUAL DOM לבין ה DOM האמיתי ולרנדר רק את השוני למסך. את זה REACT עושה מאוד מהר ולכן מקבלים בסופו של דבר זמני תגובה מצויינים!

# ELEMENT

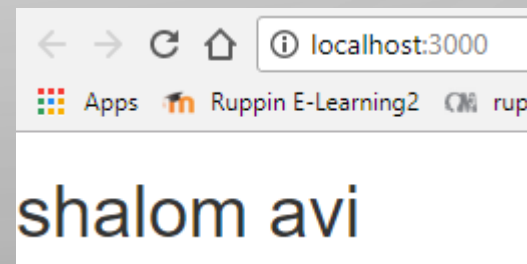
- היחידה הכי קטנה נקראת אלמנט – מציגה פיסת מידע להצגה על המסך

## App.jsx

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

## index.js

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

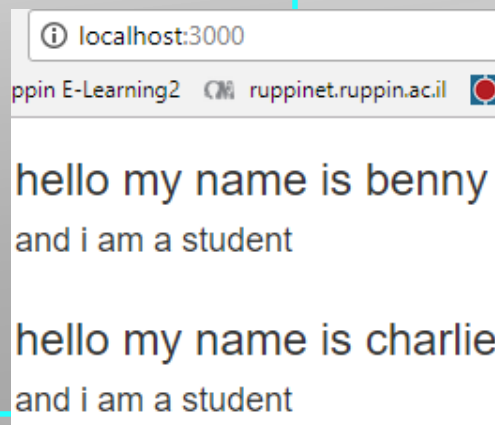


# FUNCTIONAL COMPONENT

## 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;
```



- Functional Component – זהו מרכיב של כל אפליקציה ב-REACT. כל קומפוננט כזה מכיל ערכים של קלט כאשר הפלט היא פקודת Return() שמכילה את מה שרוצים להראות על המסך.
- בצורה הפשוטה יותר זו יכולה להיות פונקציה של JS שמקבלת אובייקט JS בשם PROPS ומחזירה אלמנטים לרינדור על המסך. Functional component
- **PROPS הוא READ ONLY !!! לא ניתן לשנות את הערך שלו – IMMUTABLE ! קומפוננט לעולם לא יכול לשנות את הPROPS שלו!!!**
- הפונקציה RETURN חייבת להחזיר אלמנט אחד בלבד שיכול להכיל כמה אלמנטים שרוצים.

- נהוג להשתמש באות גדולה עבור קומפוננט
- ניתן לייצר קומפוננטים כיצירה של אלמנט HTML.
- ניתן להעביר מידע כ-ATTRIBUTE ל-PROPS של הקומפוננט.
- ניתן להשתמש ב{ } בכדי לכתוב קוד, או לגשת לPROPS

# 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

# CLASS COMPONENTS - STATE

• כאשר יש צורך בשינוי והחזקת משתנים בתוך הקומפוננט יש צורך לייצר class component המכיל STATE. הפונקציה שמחזירה את הפלט למשך נקראת .RENDER()

## • STATE

- יחידת מידע שניתן לשנות מתוך הקומפוננט.
- מעין שדה פרטי של מחלקה
- מוגדר בבנאי

### App.jsx

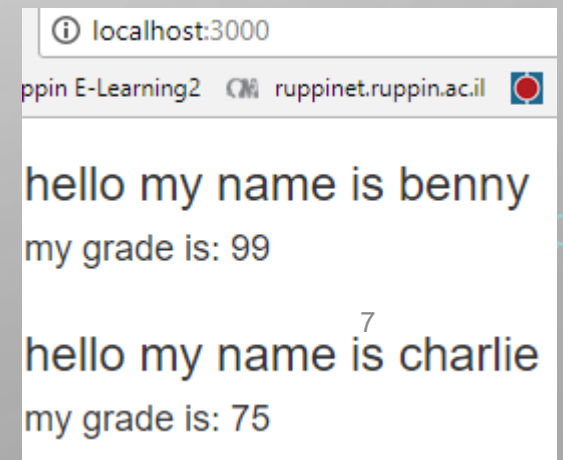
```
...
class Student extends React.Component {
  constructor(props) {
    super(props);
    this.state = { grade: Math.round(Math.random() * 40 + 60 )};
  }

  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

### App.jsx

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

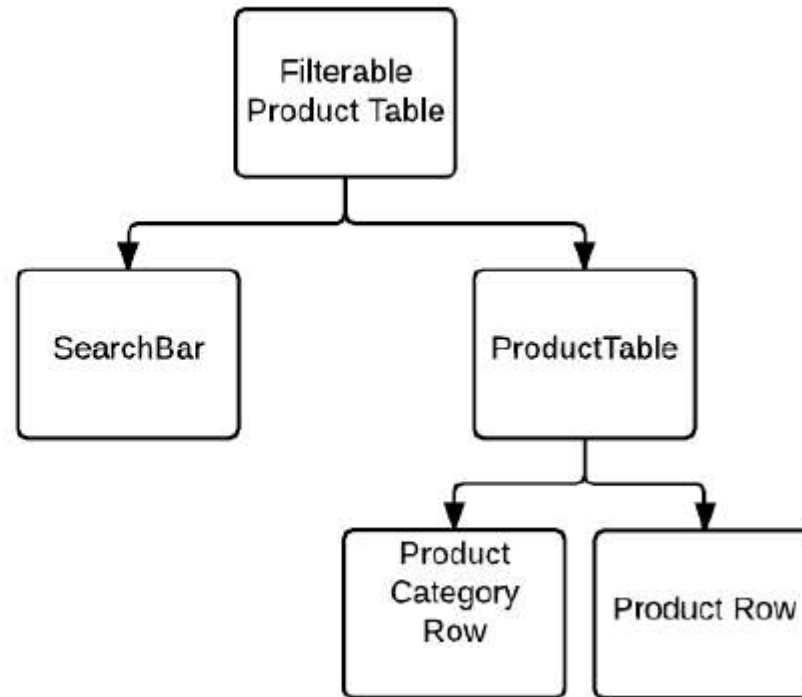


# COMPONENTS

Search...

☐ Only show products in stock

Name	Price
<b>Sporting Goods</b>	
Football	\$49.99
Baseball	\$9.99
Basketball	\$29.99
<b>Electronics</b>	
iPod Touch	\$99.99
iPhone 5	\$399.99
Nexus 7	\$199.99



- בריאקט הכל זה קומפוננטות
- חלקן "טיפשות" – ללא STATE רק functional components, חלקן "חכמות" – 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

# EVENT HANDLING – IN FUNCTIONAL COMPONENTS

...

```
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>);
```

hello my name is benny  
and i am a student

show number

hello my name is charlie  
and i am a student

show number

- בריאקט שמות האירועים הם

camelCase

- הקריאה לפונקציה נעשית ע"י

השימוש ב{} ובשם הפונקציה בתוך הסוגרים.

- הגדרת הפונקציה נעשית כרגיל

בJS 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;  
  }  
}
```

local

field

# 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);  
}
```

- ישנן 4 אופציות לחבר אירועים ב CLASS COMPONENT
- 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-defalut" />
```

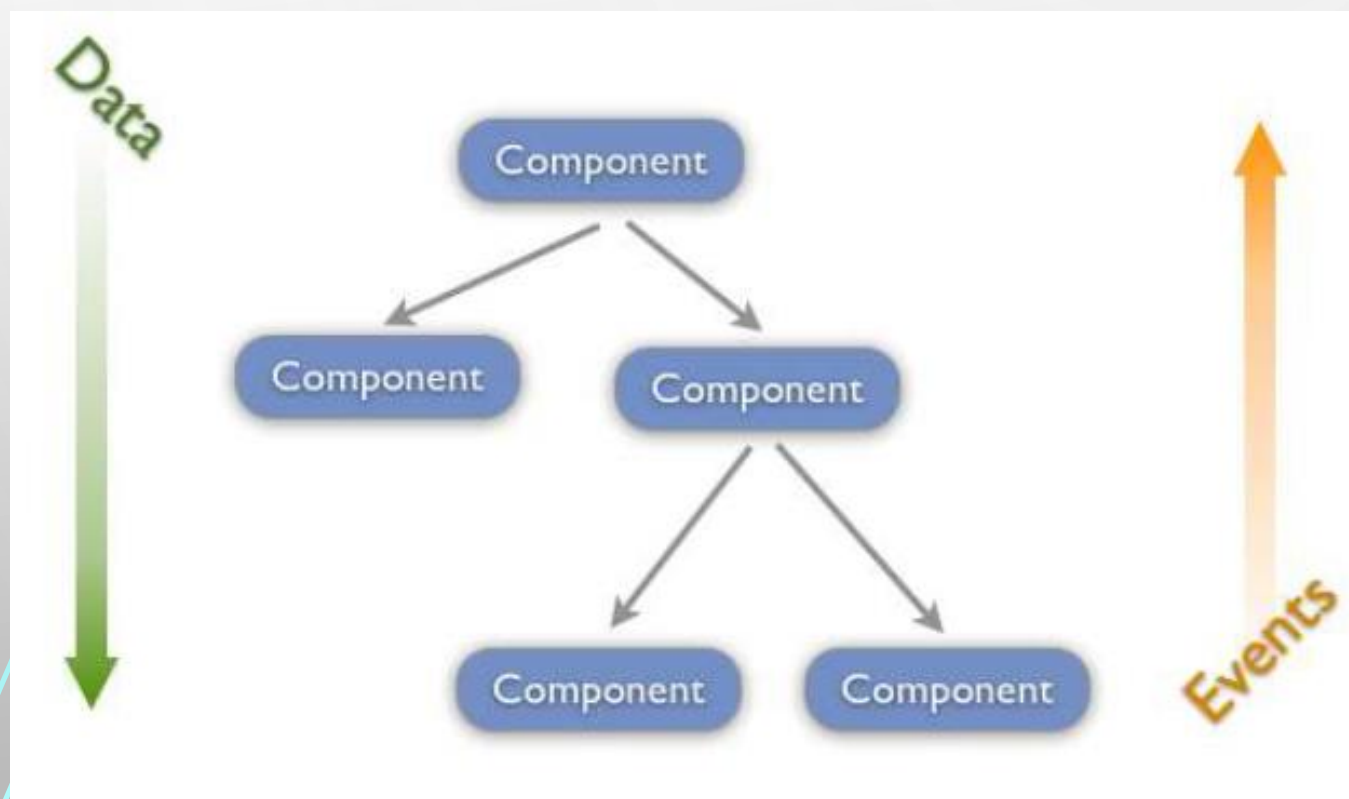
```
<input type="button"  
value="show number2"  
onClick={this.btnClicked2}  
className="btn btn-defalut" />
```

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

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

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

# “TOP-DOWN” OR “UNIDIRECTIONAL” DATA FLOW




©NIR CHEN

- קומפוננט "הורה" יכול להעביר את המידע שבו לקומפוננט "ילד" ע"י PROPS.
- כל קומפוננט בהיררכיה יכול להיות רק CLASS או FUNCTIONAL.
- ע"י PROPS ו-STATE 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/>  
<ChildCopmponent 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

# 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-DOM מחדש ולכן לא נוכל לראות את השינוי. `SetState` קוראת ל-`RENDER()` לרוץ ואז נעשית בדיקה מה התחדש ב-DOM.

- אופציה 1: לשנות עבור מקרים בהם לא מעדכנים את הערך כתלות מהערך הקודם. (פה בדוגמה זה כן תלות מהערך הקודם אבל...סעיף הבא!)

- יש שתי אופציות עבור מקרים של שינוי כתלות מהערך הקודם. אופציה 2 ו-3 חובה כי יכול להיות `asynchronously`. וגם REACT יכול להריץ כמה שינויים בבת 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 המסך את הקומפוננט , קיים ב DOM
- componentDidMount – פה אפשר לשנות את ה STATE ע"י SETSTATE כי הקומפוננט כבר צוייר למסך וקיים ב DOM ו- STATE גורם לריצה של RENDER שוב לכן צריך להיות קיים כבר ב DOM. פה גם נלך לקחת מידע מהשרת אם צריך
- ...

# COMPONENTDIDMOUNT

- אירוע שקורה רק פעם אחת לאחר שהקומפוננט מריץ את הפונקציה RENDER בפעם הראשונה ומייצר את הDOM.
- למשל אם רוצים לשנות בהתחלה את ה-STATE ע"י קוד. לא ניתן לעשות זאת בבנאי כי לא ניתן עדין להריץ (Re)Render() לפני שהיא רצה בפעם הראשונה. אחרת תהיה לולאה אינסופית. ואז ניתן לעשות זאת בcomponentDidMount

## App.jsx

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

©NIR CHEN

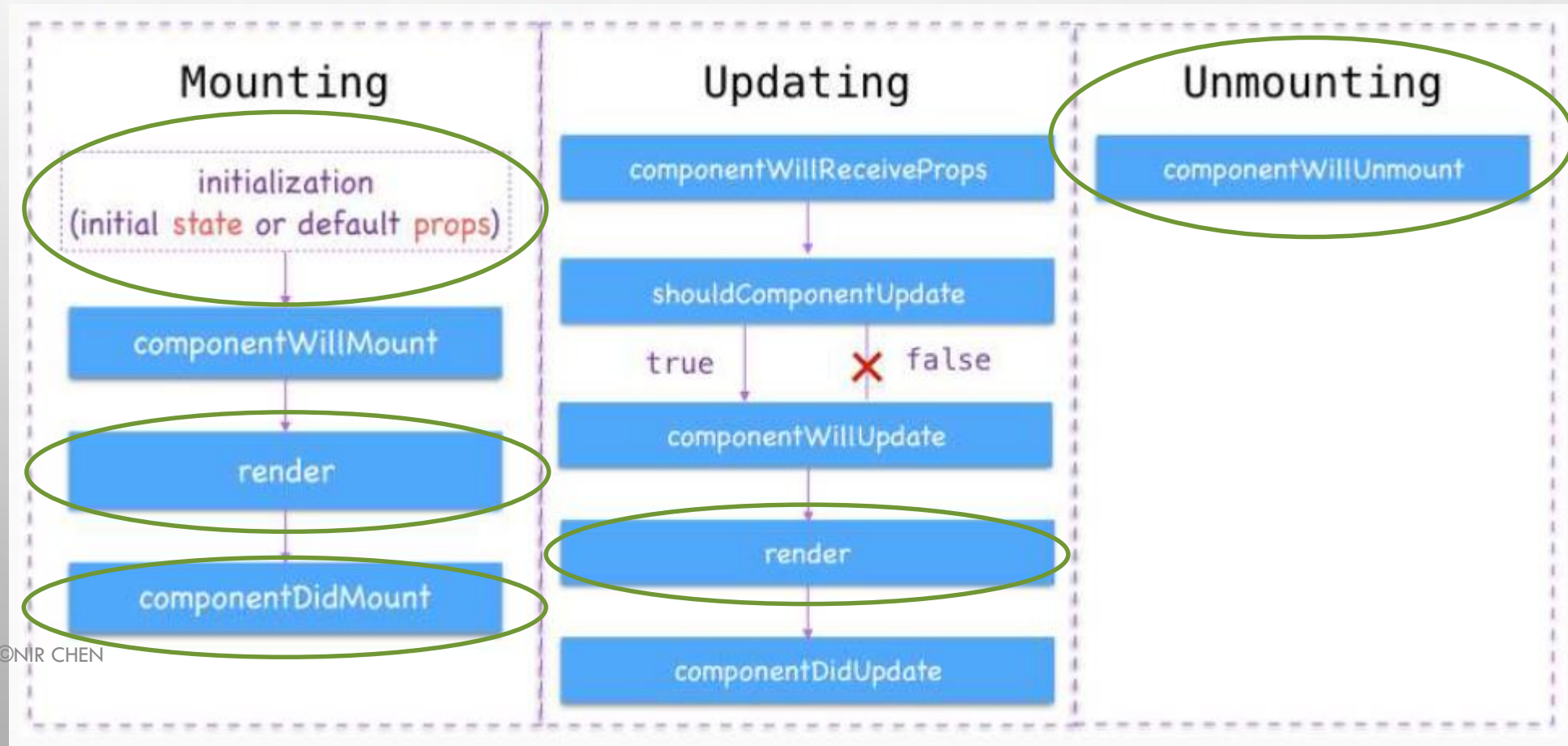
# COMPONENTWILLUNMOUNT

- אירוע שקורה רק פעם אחת לפני שהקומפוננט מפונה מהזכרון.
- למשל אם לנקות\לפנות משאבים אז ניתן לעשות זאת ב `componentWillUnmount`

## App.jsx

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

# 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

# JS MAP FUNCTION

- Map עוברת על מערך ופועלת על כל אחד מהאיברים ע"פ פונקציה מבוקשת.

- ניתנת לשימוש במספר צורות, לדוגמה

```
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);  
}
```

```
► (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>  
  }  
}
```

- כאשר יוצרים רשימה יש צורך לתת לכל איבר מפתח כמחרוזת ייחודית בכדי שריאקט ידע לזהות אותו משאר האיברים עבור הוספה, עדכון ומחיקה.

- צריכים להיות ייחודיים רק בתוך רשימה ספציפית ולא בכל העמוד.

- לא מועברים PROPS

- ניתן לעשות זאת למשל כך:

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



# FORM CONTROLS

- בטפסים נמצאים פקדים אשר דרכם יכול המשתמש לנהל מידע.
- אותם פקדים ניתנים לשינוי ע"י המשתמש
- בכדי לעשות שימוש במידע הזה בקוד אנחנו חייבים לחבר את הפקד לSTATE. זאת נעשה ע"י למשל  
`value={this.state.userName}`
- כאשר חיברנו פקד ל STATE- לא נראה אותו משתנה אלה אם כן תרוץ הפונקציה RENDER אשר רצה רק כאשר נעשה שינוי בSTATE כפי שלמדנו בפרק 05 SetState And Lifecycle. בכדי לעשות את אותו שינוי נוסיף לפקד למשל את `onChange={this.onTextChanged}` ובפונקציה נשתמש למשל ב  
`this.setState({userName: event.target.value});`

המשך בעמוד

הבא... 25



# FORM CONTROLS –INPUT TEXT

```
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.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}
```

The screenshot shows a web form with three text input fields and a button. The first input is labeled 'not connected to state input' and contains the text 'avi'. The second input is labeled 'connected to state input WO onchange' and contains the text 'nir'. The third input is labeled 'connected to state input With onchange' and also contains the text 'nir'. Below the inputs is a button labeled 'hello user'.

Alert  
undefined

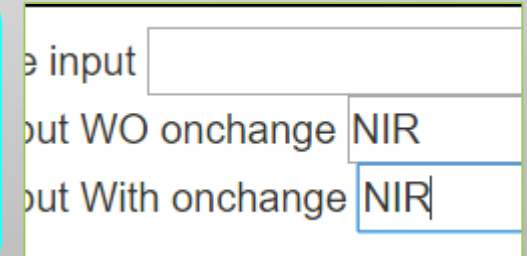
לא ניתן לשינוי  
ע"י המשתמש

ניתן לשינוי ע"י  
המשתמש  
Alert nir

## שליטה בהזנת נתונים

- מכיוון שאנחנו מקבלים את שינויי המשתמש אבל מזינים בעצמנו את ה STATE, ניתן לשנות אותו באופן שטוב לנו למשל ניתן לקחת את האותיות המוכנסות לתיבת הטקסט ולהפוך אותן לגדולות ישירות בהקלדה.

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



# <TEXTAREA>

- נעשה שימוש ב-VALUE בכדי להציג מלל על המסך

```
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 />
```

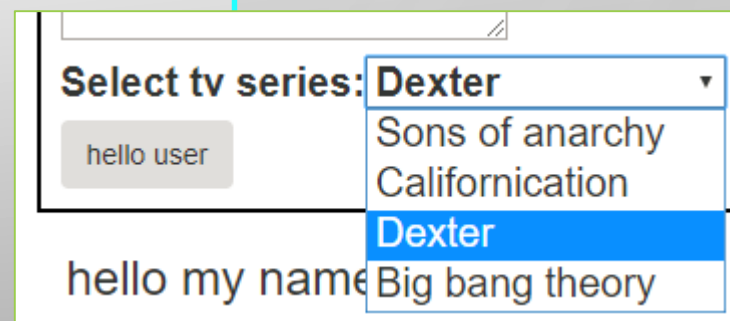
לא הגדרנו את TEXTAREA ב  
CTOR בכדי שנוכל לראות את ה  
PLACEHOLDER

insert the story

# <SELECT>

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

```
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>
```



# 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

# PROPTYPES

- בכדי לאמת את הנתונים שמגיעים לPROPS 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 נוכל 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 `Student.propTypes` קטנה בפ גדולה.  
בהגדרות עצמן בפ גדולה.

- יש המון הגדרות אפשריות לסוגי הנתונים ומאפייניהם

- חלק מההגדרות...

- <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

```

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


```

## REFS

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

RefsDemo

<input type="text" value="txt1"/>	txt1
<input type="text" value="txt2"/>	txt2

```

constructor(props) {
  ...
  this.txtInput1; ←
  this.textInput2 = React.createRef(); ←
}

TextChanged = () => {
  this.setState({
    txt1: this.txtInput1.value, ←
    txt2: this.textInput2.current.value ←
  })
}
...


```

## REFS- TWO VERSIONS

• עם REF

• מהאתר 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.

RefsDemo

txt1	txt1
txt2	txt2

35

# 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

- בREACT מעבר מעמוד לעמוד אחר לא נעשה ע"י החלפה של עמוד ה HTML אלה כל הזמן נשארים באותו עמוד INDEX.HTML ורק הDOM מתחלף.
- כך חוסכים בזמן.
- כמובן שגם החלפת חלק מעמוד במקרה למשל של NAVIGATION BAR שמחליף את החלק העיקרי של העמוד – גם אז עושים שימוש בREACT ROUTER שמחליף רק את הDOM ולא את כל העמוד.
- נעשה שימוש בעיקר בארבעה COMPONENTS - SWITCH, ROUTER, BROWSER, LINK
  - BrowserRouter - עוטף את כל האפליקציה ודואג למעבר העמודים
  - withRouter – מייחצן את ה APP כלפי ה-INDEX.JS
  - Switch – עוטף את מיקום ה ROUTE. איפה לשתול את הדפים עצמם?
  - Route – הגדרת המסלול כ - URL שמתאים למעבר לCOMPONENT מסוים
  - Link - הכפתור שעליו לוחצים בכדי לעבור למסלול מסוים

# BROWSERROUTER

- בעמוד הראשי index.js נגדיר את ה-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);
```

## ROUTE, WITHROUTER

- בעמוד הראשי למשל APP.JS נגדיר את הROUTים המסוימים.
- המסלול הראשון שמתאים הוא זה שנעבור עליו
- שימו לב לשימוש במילה **exact** עבור המסלול של הHOME כי אחרת זה יתאים לכל קריאה.
- בכל מקום בו נרצה לעבור ע"י קוד לעמוד אחר (ראה בהמשך) נחזיר את הCOMPONENT ע"י withRouter

# LINK

- עכשיו נוכל להשתמש ב LINK בכל מקום באתר בכדי 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>  
    )  
  }  
}
```

```
<p>home</p>
```





# URL PARAMETERS AND MATCH

- ניתן להוסיף למסלול גם פרמטרים עם ערכים משתנים.

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

```
<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>
```



# URL PARAMETERS AND MATCH

- בכל פעם שקוראים לCOMPONENT מועבר בצורה אוטומטית לתוך ה PROPS שלו פרמטר נוסף שנקרא MATCH שמכיל מידע על ה ROUTE בין היתר גם את השדה PARAMS שמכיל פרמטרים שהועברו ב URL 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 query string או ע"י העברת המידע בstate.
- ע"י state ניתן להעביר גם אובייקט שלם.