

# REACT

02 COMPONENTS AND PROPS

©NIR CHEN



# SYLLABUS

- 01 Fundamentals And Installation
- **02 Elements, Functional Components And Props**
- 03 Handling Events in FC
- 04 Class Component, State and SetState
- 04.5 Uplifting
- 05 Lifecycle
- 06 Lists And Keys
- 07 Forms
- 08 PropTypes
- 09 Refs

- Need to teach HOOKS HERE for navigation
- 10 React Router v7-1 **W HOOKS**
- 10.5 data-attribute
- 10.7 Sync setState

# SPA AND JSX

- ב כדי לגרום ליראה מהירה יותר של האפליקציה מתכנתים בSPA. כך ניתן 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 SJ.  
**היצירה הזאת היא רק מעבר ה VIRTUAL DOM !**
- המטרה של REACT היא לבדוק את ההבדלים בין ה VIRTUAL DOM לבין ה DOM האמתי ולרNDER רק את השוני למסך. את זה REACT עושה מאוד מהר ולכן מקבלים בסופו של דבר זמן תגובה מצוין!

# ELEMENT

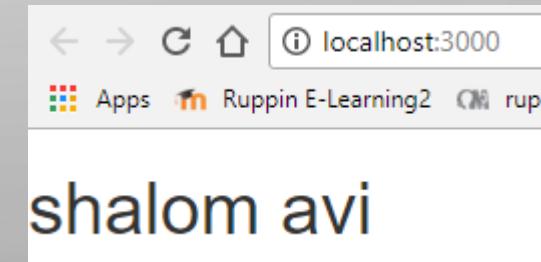
- היחידה הici קטנה נקראת אלמנט – מציגה פיסת מידע להציג על המסך

## App.jsx

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

## index.js

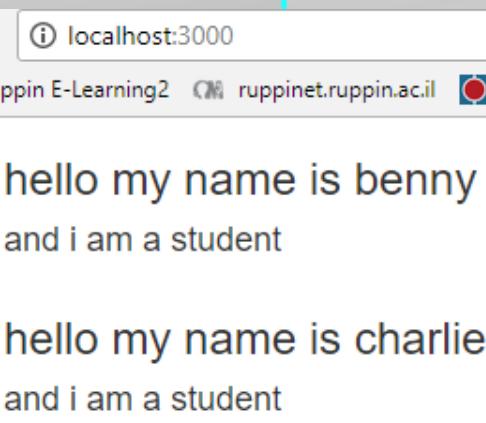
```
...  
ReactDOM.render(App,  
document.getElementById('root'));  
...
```



# 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;
```



- קומפוננט – זה מרכיב של כל אפליקציה בReact. כל קומפוננט כזה מכיל ערכים של קלט כאשר הפלט היא פקודה (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

- **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.

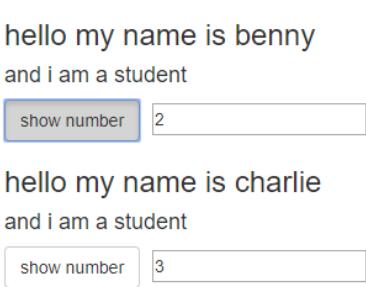
- נוהג להשתמש באות גדולה עבור קומפוננט.
- ניתן ליצור קומפוננטים כיצירה של אלמנט HTML.
- ניתן להעביר מידע כ-ATTRIBUTE ל-PROPS של הקומפוננט.

- ניתן להשתמש ב{} כדי לכתוב קוד, או לגשת ל-PROPS

## 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 נועשית ע"י השימוש ב{} ובשם הפונקציה בתוך הסוגרים.

- הגדרת הפונקציה נועשית כרגע

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

בגע ע"י המילה function

# SYLLABUS

- 01 Fundamentals And Installation
- 02 Elements, Functional Components And Props
- 03 Handling Events in FC
- **04 Class Component, State and SetState**
- 04.5 Uplifting
- 05 Lifecycle
- 06 Lists And Keys
- 07 Forms
- 08 PropTypes
- 09 Refs
- 10 React Router v4

©NIR CHEN

# CLASS COMPONENTS - STATE

## App.jsx

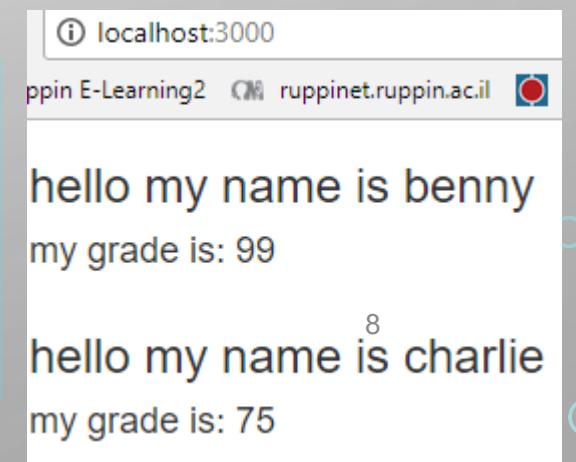
```
...
class Student extends React.Component {
  constructor(props) { ←
    super(props); ←
    this.state = { grade: Math.round(Math.random() * 40 + 60 )}; ←
  }
  render() { ←
    return (
      <div className="container">
        <h3>hello my name is {this.props.name} </h3>
        <h4>my grade is: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

## App.jsx

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

כאשר יש צורך בשינוי והחזקת משתנים בתוך הקומפוננט יש צורך ליצור `class component` המכיל STATE. הפונקציה `STATE` שמחזירה את הפלט למשר נקראת `RENDER()`.

- ייחdet מידע שנייתן לשנות מtower הקומפוננט.
- מעין שדה פרטי של מחלוקת
- מוגדר במבנה



## 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; ← field  
    this.txtChanged = this.txtChanged.bind(this);  
    this.btnClicked2 = this.btnClicked2.bind(this);  
  }  
  txtChanged(e) {  
    this.number = e.target.value;  
  }  
}
```

local

# 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>
```

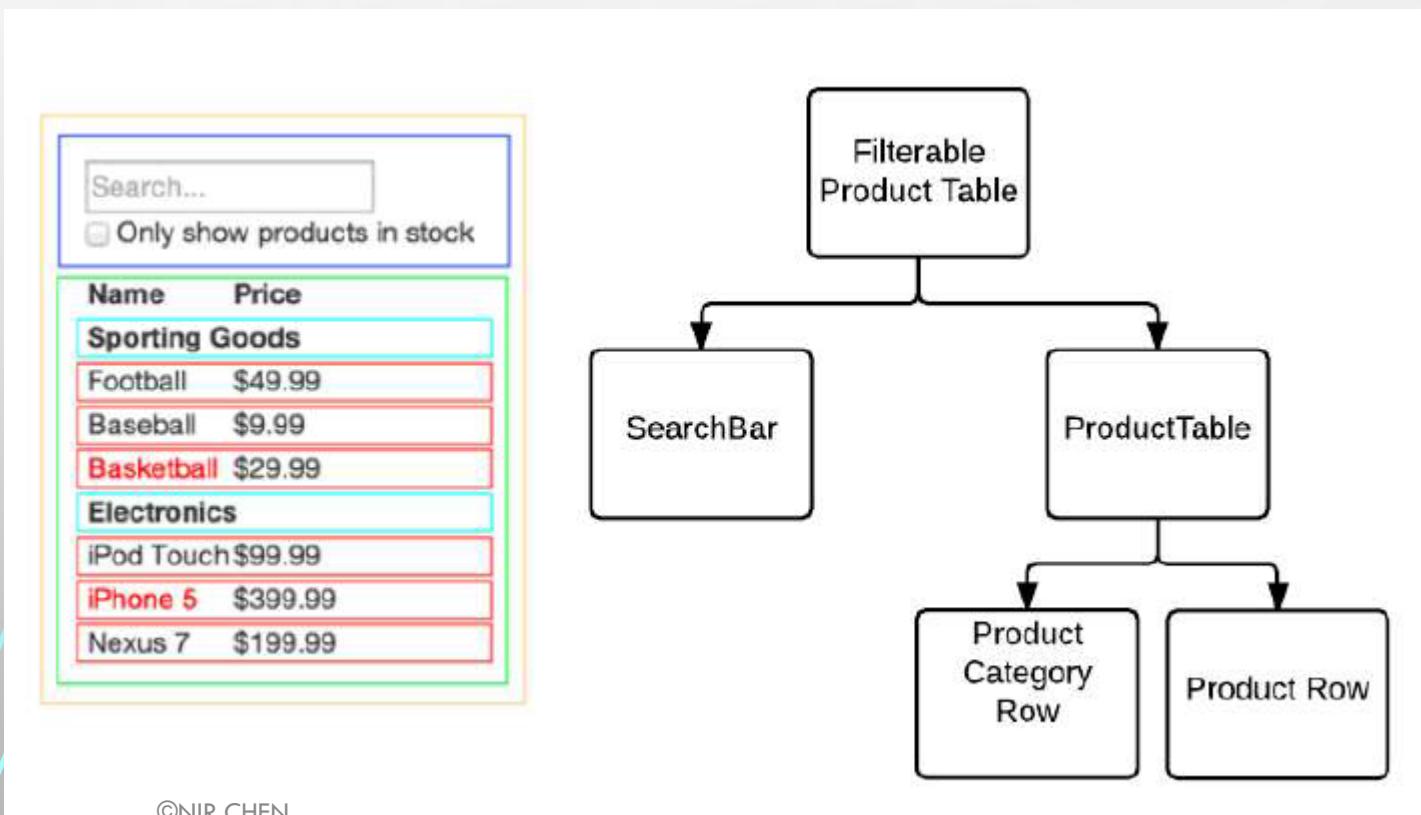
# STATE CHANGING

```
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 החדש וلن יוכל לראות את השינוי. `SetState` קוראת `l()` ל-`RENDER` לרווח ואז נעשית בדיקה מה התREDISDOM.
- 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 <sup>11</sup> להיות asynchronously. וגם REACT יכול להריץ כמה שינויים בבאת אחת.

# COMPONENTS



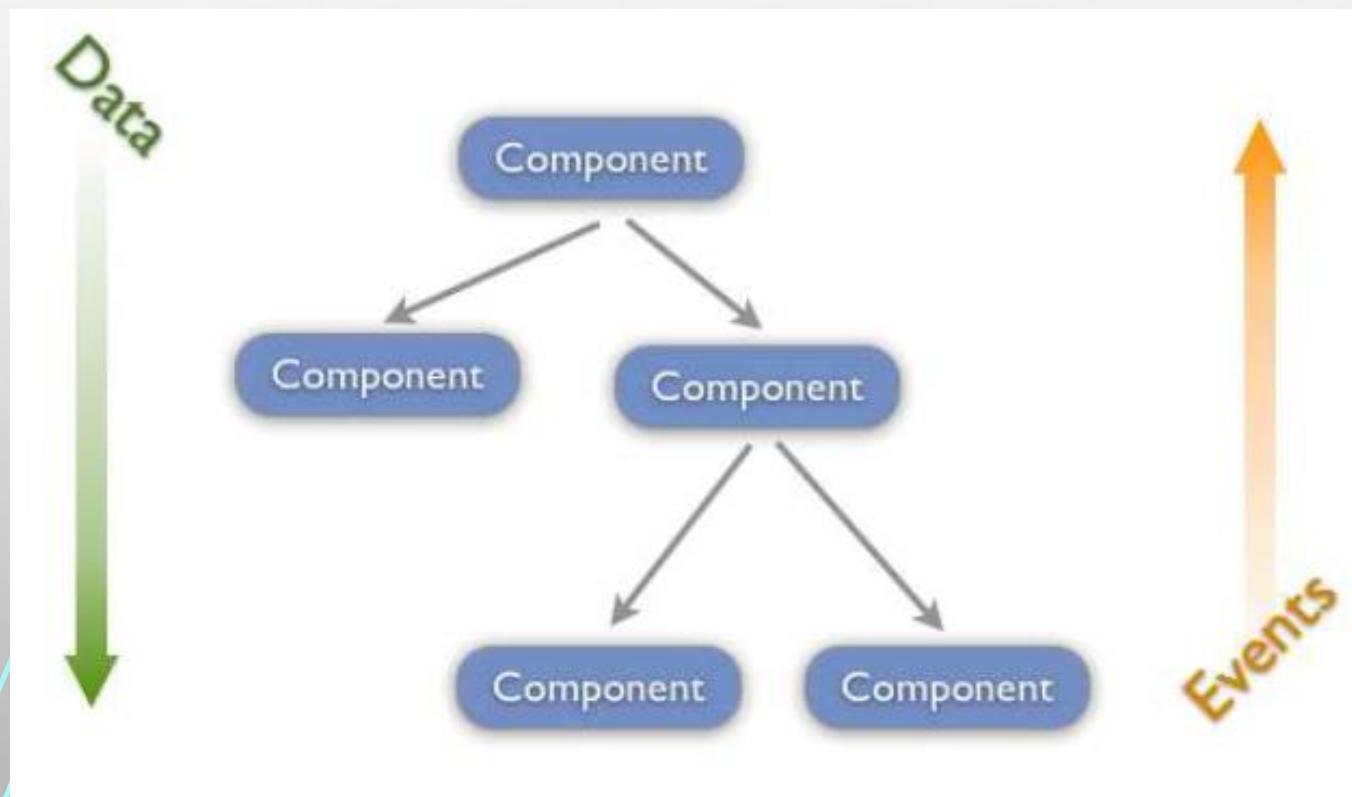
- בראקט הכל זה קומפוננטות
- חלקו "טייפשות" – ללא STATE רק **functional components**
- "חכמת" – עם STATE **components**

# SYLLABUS

- 01 Fundamentals And Installation
- 02 Elements, Functional Components And Props
- 03 Handling Events in FC
- 04 Class Component, State and SetState
- **04.5 Uplifting**
- 05 Lifecycle
- 06 Lists And Keys
- 07 Forms
- 08 PropTypes
- 09 Refs
- 10 React Router v4

©NIR CHEN

## “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 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.

זה נקרא **uplifting**

## App.jsx

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

# SYLLABUS

- 01 Fundamentals And Installation
- 02 Elements, Functional Components And Props
- 03 Handling Events in FC
- 04 Class Component, State and SetState
- 04.5 Uplifting
- **05 Lifecycle**
- 06 Lists And Keys
- 07 Forms
- 08 PropTypes
- 09 Refs
- 10 React Router v4

©NIR CHEN

# 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 לריצה 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-`DOM`.
- למשל אם רוצים לשנות בהתאם את ה-`STATE` ע"י קוד. לא ניתן לעשות זאת בבנאי כי לא ניתן עדין להריץ `(Re)Render()` לפני שהיא רצתה בפעם הראשונה. אחרת תהיה לוולה אינסופית. ואז ניתן לעשות זאת ב`componentDidMount`.

## App.jsx

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

©NIR CHEN

18

# COMPONENTWILLUNMOUNT

- אירוע שקורה רק פעם אחת לפני שהkomponent מפונה מהזיכרון.
- למשל אם לנוקות\לפנות משאביים אז ניתן לעשות זאת בcomponentWillUnmount

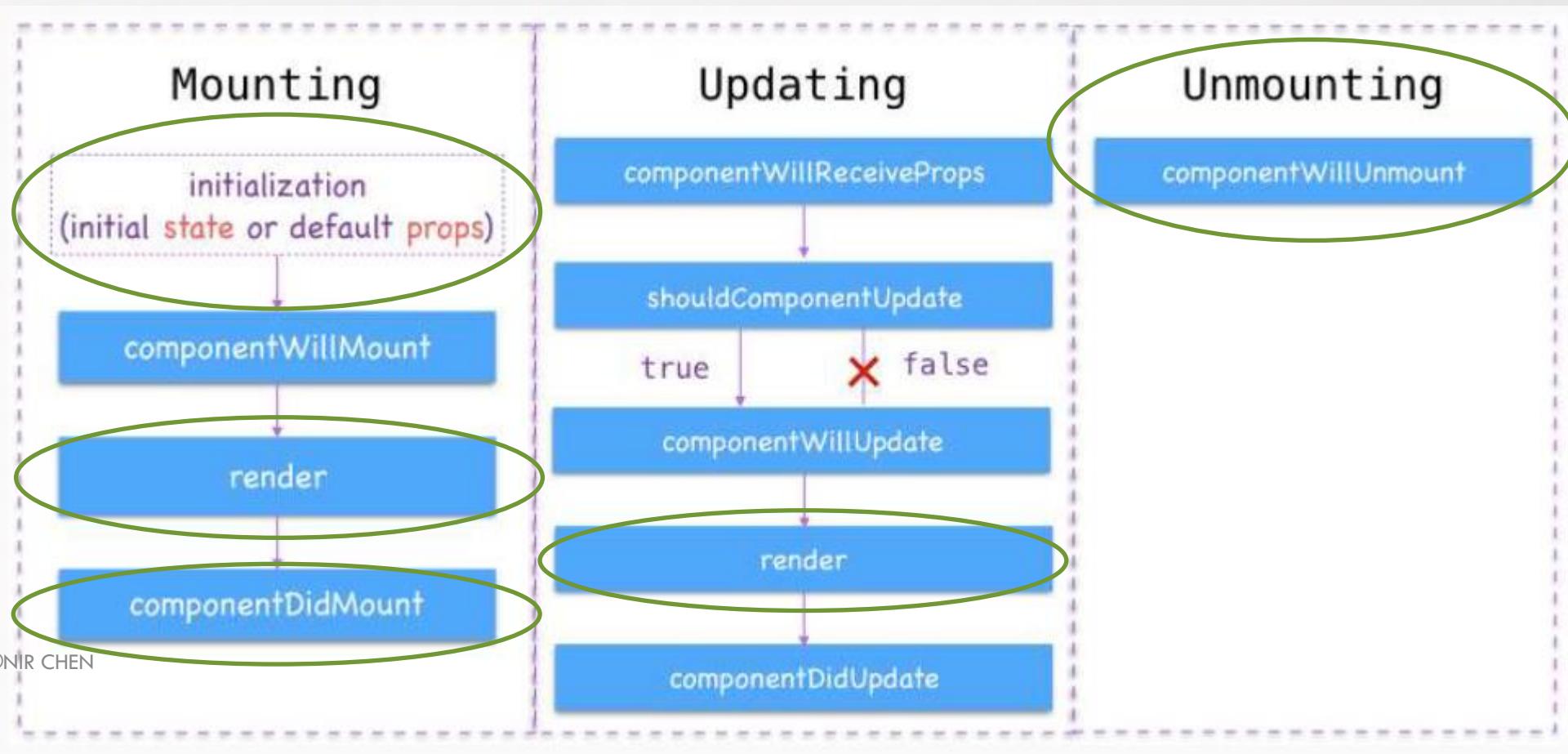
App.jsx

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

©NIR CHEN

19

# 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>
```

- כאשר יוצרים רשימה יש צורך לתת לכל איבר מפתח  
כמפורט יותר בצד שרייקט ידע לזהות אותו  
משאר האיברים עבור הוספה, עדון ומחיקה. לא  
כדי להשתמש בindex הפרמטר השני של MAP כי  
از למשל לאחר DELETE(index) יבלבל בין האיברים  
ולא יрендר נכון!

- 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

©NIR CHEN

# FORM CONTROLS

- בטפסים נמצאים פקדים אשר דרכם יכול המשתמש לנוהל מיידע.
- אותם פקדים ניתנים לשינוי ע"י המשתמש
- ב כדי לעשות 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 עשו שינוי בSTATE כפי שלמדנו בפרק 05 SetState And Lifecycle. ב כדי לעשות את אותו שינוי נוסף לפקד לדוגמה 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

The screenshot shows a browser window with three input fields and their respective alert outputs:

- not connected to state input**: An input field containing "avi". An alert above it says "Alert undefined".
- connected to state input WO onchange**: An input field containing "nir". An alert to its right says "לא ניתן לשינוי ע"י המשתמש".
- connected to state input With onchange**: An input field containing "nir". An alert below it says "ניתן לשינוי ע"י המשתמש Alert nir".

## שליטה בהזנת נתונים

- מכיון שאנו מקבלים את שינוי המשתמש אבל מזינים בעצמנו את ה STATE, ניתן לשנות אותו באופן שטוב לנו למשל ניתן לחת את האותיות המוכנסות לטיבת הטקסט ולהפוך אותן לגודלות ישרה 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 />
```

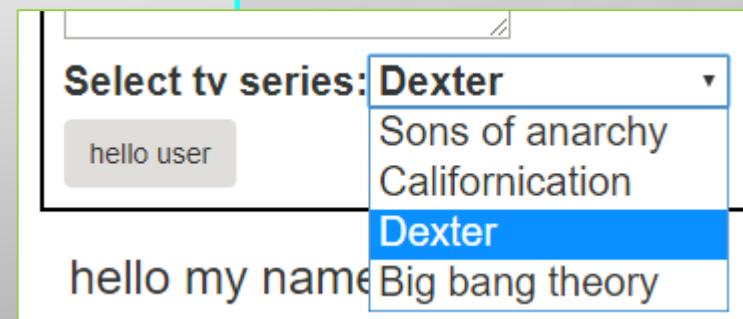
לא הגדרנו את TEXTAREA ב  
CTOR בצד שnochol לראות את ה  
PLACEHOLDER

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

30

# PROPTYPES

- ב כדי לאמת את הנתונים שmaguire לprops בזמן

ריצה ניתן להשתמש ב PropTypes

npm i prop-types

- לمثال STUDENT COMPONENT שלנו נוסיף בדיקת PROPS ע"י הוספה אחריו של הקוד הבא...  
לאחר שנפתח 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 `.propTypes` קטנה בהגדרות עצמן ב פ גדולה.

**.Student.propTypes**  
יש המונן 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

33

# 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 בזיהה כמה שפחות!!!
- אפשר לראות דוגמה לשימוש ב-REF ובלי REF
- פה בלוי 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 מודולרי
- 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 v6**

36

# HOOKS...

- Need to teach HOOKS here for navigation

# REACT ROUTER V7-1 – OPTION 1

- בACT מ עבר לעמוד לעמוד אחר לא נעשה ע"י החלפה של לעמוד ה HTML אלה כל הזמן נשאים באותו לעמוד INDEX.HTML ורק ה DOM מתחלף.
- קר חוסכים בזמן.
- כמובן שגמ החלפת חלק מעמוד במקרה למקורה למשל של NAVIGATION BAR שמחליף את החלק העיקרי של העמוד – גם אז עושים שימוש בROUTER שמחליף רק את ה DOM ולא את כל העמוד.
- נעשה שימוש בעיקר בארבעה **LINK**, **ROUTE**, **BROWERSERROUTER**, **ROUTES- COMPONENTS** –  
  - **BrowserRouter** – עותף את כל האפליקציה ודואג למעבר העמודים
  - – עותף את מקום ה ROUTE. איפה לשתול את הדפים עצם?
  - – הגדרת המסלול כ – URL שמתאים למעבר לCOMPONENT מסוים
  - **Link** – הceptors שעליו לוחצים בכך לעבור למסלול מסוים

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

# BROWSERROUTER

- בעמוד הראשי index.js נגדר את BROWSERROUTER

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

# ROUTE, ROUTES

```
import React from 'react';
import { Routes, Route} from 'react-router-dom';
import Home from './Home';
import About from './About';
import MenuComponent from './MenuComponent';

...
return (
  <div>
    <Routes> ←
      <Route path="/" element={<Home />} /> ←
      <Route path="/about" element={<About />} />
      <Route path="/menu" element={<MenuComponent />} />
      <Route path="/user" element={<CCUser />} />
    </Routes> ←
  </div>
...
export default App;
```

- בעמוד הראשי למשל APP.JS נגדייר את הROUT-ים המסוימים.
- ריאקט מנסה להתאים את המסלול המתאים ביותר גם ללא סדר הכתיבה.

# 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';
...
return (
  <div>
    <Link to="/">home</Link> |
    <Link to="/about">About</Link> |
    <Link to="/menu">Menu</Link>
  ...
  <p>home</p>
)
```

# 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" element={<MenuComponent/>} />  
  </div>  
</BrowserRouter>
```

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

# USEPARAMS

- באמצעות USEPARAMS נוכל לשלוף מידע על ה ROUTE בין היתר גם את השדה שמכיל פרמטרים שהועברו ב URL.

```
import { useParams } from 'react-router';  
  
export default function  
MenuUserComponent(props) {  
  const params = useParams();  
  return (  
    <div>  
      <h3>in MenuUserComponent with <br />  
        user id: {params.userId}</h3>  
    </div>  
  );  
}
```

©NIR CHEN

# CHANGE PAGE PROGRAMMATICALLY

```
function App(props) {  
  const navigate = useNavigate();  
  
  const btnAbout = () => {  
    navigate('about');  
  }  
  const btnUser = () => {  
    var userObj = {  
      userId: 1,  
      userName: "avi"  
    };  
  
    navigate('user',  
    { state : userObj });  
  }  
}
```

```
const { state } = useLocation();  
let userObj = state;  
console.log(userObj.userId);  
//
```

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

# RUPPIN'S SERVER

- ברגע שאתה מעלים לשרת של רופין (proj.ruppin.ac.il) את הקוד, אתה צריכים להחליף את הפקודה `BrowserRouterRouter` בפקודה `HashRouter`

# REACT ROUTER V7-1 – OPTION 2

- בACT מ עבר לעמוד לעמוד אחר לא נעשה ע"י החלפה של לעמוד ה HTML אלה כל הזמן נשאים באותו לעמוד INDEX.HTML ורק ה DOM מתחלף.
- קר חוסכים בזמן.
- כמובן שגמ החלפת חלק מעמוד במקרה למקורה למשל של NAVIGATION BAR שמחליף את החלק העיקרי של העמוד – גם אז עושים שימוש בROUTER שמחליף רק את ה DOM ולא את כל העמוד.
- נעשה שימוש בעיקר באربעה מרכיבים - `LINK`, `RouterProvider`, `createBrowserRouter`, `Outlet`-
  - `RouterProvider` – עוטף את כל האפליקציה וודאג למעבר העמודים
  - `createBrowserRouter` – הגדרת המסלול כ- URL שמתאים למעבר לCOMPONENT מסוים
  - `Outlet` – מגדר מקום עבור שתילת COMPONENT Placeholder. עברו קומפוננטה פנימית יותר.
  - `Link` – הכפתור שעליו לוחצים בכך לעבר למסלול מסוים

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

```
import { createBrowserRouter, RouterProvider } from "react-router-dom";

function App() {
  const router = createBrowserRouter([
    {
      path: "/",
      element: <Root/>,
      //loader: rootLoader,
      children: [
        {
          path: "/page1",
          element: <Page1 />,
        },
        {
          path: "/page2",
          element: <Page2 />,
        },
      ],
    },
    {
      path: "/page3/:userId",
      element: <Page3 />
    },
  ],
},
```

## ROUTERPROVIDER

- בעמוד הראשי נניח באז' App נגדיר את `createBrowserRouter` ואת `RouterProvider` המוסויים.
- בעמוד הראשי למשל SJ.APP נגדיר את הROUT-ים המוסויים.
- ריאקט מנסה להתאים את המסלול המתאים ביותר גם ללא סדר הכתיבה.

```
return (
  <>
  <RouterProvider router={router} />
)</>
```

# LINK

- עכשו נוכל להשתמש ב LINK בכל מקום באתר ב 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';
...
return (
  <div>
    <Link style={{margin:10}} to="page1">page1</Link> | ←
    <Link style={{margin:10}} to="page2">page2</Link> | ←
    <Link style={{margin:10}} to="page3/7">page3</Link> | ←
  <p>home</p>
```

# OUTLET

- עכשו נוכל להשתמש ב OUTLET כתופס מקום עבור הצגת קומפוננטות ילדיים.
- במקומות ה OUTLET ירונדרו ה children שהגדרכנו ב createBrowserRouter

```
import { Link, Outlet } from "react-router-dom";  
  
export default function Root() {  
  return (  
    <div>  
      <Link style={{margin:10}} to="page1">page1</Link> |  
      <Link style={{margin:10}} to="page2">page2</Link> |  
      <Link style={{margin:10}} to="page3/7">page3</Link> | <br />  
      START OUTLET  
      <Outlet/>  
      END OUTLET  
    </div>  
  );  
}
```

# URL PARAMETERS AND USEPARAMS

```
{  
  path: "/page3/:userId",  
  element: <Page3 />  
},
```

```
import { Link, useParams } from "react-router-dom"  
  
export default function Page3() {  
  const { userId } = useParams();  
  console.log(userId);
```

- ניתן להוסיף למסלול גם פרמטרים עם ערכים משתנים.
- באמצעות USEPARAMS נוכל לשלוּפּ מידע על ה ROUTE בין היתר גם את השדה שמכיל פרמטרים שהועברו ב URL.

# CHANGE PAGE PROGRAMMATICALLY

```
function App(props) {  
  const navigate = useNavigate();  
  
  const btnAbout = () => {  
    navigate('about');  
  }  
  const btnUser = () => {  
    var userObj = {  
      userId: 1,  
      userName: "avi"  
    };  
  
    navigate('/page7',  
    { state : userObj });  
  }  
}
```

```
const { state } = useLocation();  
let userObj = state;  
console.log(userObj.userId);  
//
```

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

# 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
- **10.5 data-attribute**

53

# DATA-ATTRIBUTE

- ניתן לזהות אלמנטים HTML ע"י הוספת מידע כ ATTRIBUTE באופן הבא:

```
<a href="#" data-userid={user.id} onClick={this.btnX}>X</a>
```

```
btnX = (e) => {  
  console.log(e.target.dataset.userid);  
}
```

- ניתן לקרוא את המידע ע"י ..

# SYNC SETSTATE

- ניתן לקבל חיווי ע"י כאשר ה-SETSTATE הסתiem לרצוי:

```
this.setState({  
    grade: this.state.grade + parseInt(this.props.bonus)  
, ()=>{  
    //code something that runs after the state was updated!!!  
});
```

# IMGAES

```
import img from "../assets/img.jpeg";
...
return (
  <div>
    from public: <br />
    from public: <br />
    from assets:<img src={"src/assets/img.jpeg"} /><br />
    from image:<br />
    from assets using import:<img src={img} /><br />
```

