



11:00 04/03/2018

מדריך *React* למתחילים

ב: מדריכים (tags/tutorials/.)

נכתב ע"י: מיתר ברונר

כל מי שמתעסק היום בעולם פיתוח האתרים בוודאי שמע על React ועל היתרונות בפיתוח עם ספריה זו. במדריך זה אנסה לתת את הכלים הבסיסיים שיאפשרו לכל מפתח אשר אינו נגע מעולם בספריה לכתוב את אפליקציית ה-React הראשונה שלו.

יש לציין שמדריך זה מיועד למפתחים אשר כבר מכירים את עולם פיתוח האתרים לפחות ברמה בסיסית כיוון שעל מנת להתחיל לכתוב קוד ב-React לפי הסטנדרטים המקובלים, כדאי כי למי שקורא מדריך זה יהיה ידע בסיסי ב-Javascript וכמובן ב-HTML ו-CSS.

הערה חשובה: יתכן ובתור מפתחי web מדריך זה ירגיש לכם ארוך מאד, שרק על מנת לבנות משהו מאד פשוט. אני ממליץ לסיים את המדריך עד סופו על מנת להבין את היתרונות ש-React מספקת בבניית רכיבי web שהם reusable (ניתן להשתמש בהם שוב ושוב בין פרויקטים) ומודולריים, כך שהם בלתי תלויים לחלוטין אחד בשני.

מהי ספריית React?

React היא ספריית קוד Javascript אשר פותחה ומתוחזקת בין היתר ע"י Facebook על מנת ליצור ממשקי משתמש (User Interfaces) לדפדפנים בצורה קלה ויעילה.

איך React עובדת?

ראשית נסביר על אבן הפינה הבסיסית ביותר בריאקט - הקומפוננטה (Component). הספרייה תומכת ביצירת קטעי קוד ב-Javascript אשר מייצגים רכיבים באפליקציה בעלי התנהגות זהה.

קוד ה-Javascript של כל קומפוננטה הוא בלתי תלוי בשאר הקומפוננטות האחרות באפליקציה ויש לו "מחזור חיים". על מחזור החיים מיד נרחיב, אך ראשית "נקפוץ קפיצת ראש" לדוגמת קוד של רכיב כזה:

```
JS Box.js
1
2 import React, { Component } from 'react';
3
4 class Box extends Component {
```

```
5
6   constructor(props) {
7     super(props);
8     this.state = {
9       isOpen: false,
10    }
11  }
12
13  render() {
14    return (
15      <div className="Box">
16        <h1>I am a closed box</h1>
17      </div>
18    );
19  }
20 }
21
22 export default Box;
```

ייתכן ואתם, שקוראים מדריך זה, בטוחים שאתם יודעים ג'אווהסקריפט אך אינכם מבינים דבר ממה שכתוב כאן. אל דאגה, קוד זה הוא אינו Javascript שרוב הדפדפנים יודעים לקרוא והוא כתוב בכלל ב-node.js בשילוב עם סינטקס מיוחד הנקרא JSX ועל שניהם ארחיב מעט.

ראשית, node.js הינה שפה בסינטקס של ג'אווהסקריפט שפותחה בעיקר לשימוש כתיבת קוד צד שרת של אתרים ואפליקציות. שפה זו רצה במערכת ההפעלה שעל מחשבכם על יד מנוע V8 – אותו מנוע שמריץ את קוד הג'אווהסקריפט שלכם בדפדפן הפופולארי Chrome.

אז למה אנחנו בעצם משתמשים בה אם אינה מתאימה לדפדפנים?

ל-node.js יתרונות רבים בכתיבת קוד אותם נראה בהמשך, בעזרת כלים נוספים כמו ספריות webpack ו-babel, ניתן לנצל את יתרונות התכנות ב-node.js ולהפוך את קוד ה-node.js לקוד Javascript שהדפדפנים השונים יודעים לקרוא.

לגבי JSX - JSX הוא סינטקס המשלב בתוכו את אלמנטי ה-HTML המוכרים. שימו לב שבתמונה למעלה הפונקציה render מחזירה אלמנט HTML. דבר זה יקל עלינו כיוון שב-React כל קומפוננטה מחליטה בכל רגע נתון איזה אלמנטים של HTML להכניס ל-DOM.

כיוון שמדריך זה יתעסק בעיקר בכניסה מהירה לכתיבת קוד של React, נשתמש ב--create-react-app. זוהי "תוכנה מוכנה" שפייסבוק יצרו על מנת לעטוף את כל הכלים השונים שהזכרנו כמו babel - החלק האחראי להמרת הקוד לקוד שהדפדפן יודע לקרוא, ו-webpack - שיוזע לקחת קבצים שונים של קוד עם התלות ביניהם ולשלב אותם גם כן לקוד שהדפדפן יודע לקרוא.

ראשית, אם אין על מחשבכם node.js ביחד עם NPM היכנסו לכאן (<https://nodejs.org/en/download>). NPM היא תוכנה שיושבת על מחשבכם ועוזרת לנהל חבילות שונות של קוד הכתוב ב-node.js גם על מערכת ההפעלה שלכם וגם בכל פרויקט שתכתבו. לאחר ש-NPM מותקן על מחשבכם, פתחו במחשבכם את ה-cmd (או ה-terminal) והריצו את הפקודה:

```
npm install create-react-app -g
```

כעת NPM יתקין על מחשבכם את create-react-app.

לאחר שההתקנה תגמר, הריצו את הפקודה:

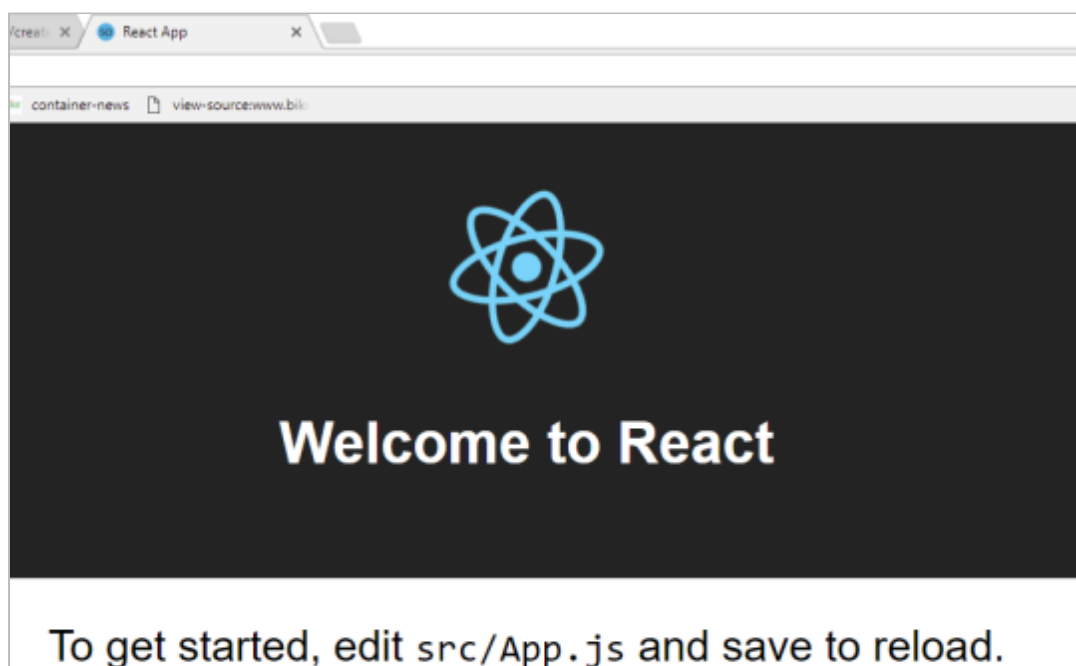
```
create-react-app my-first-app
```

כעת התוכנה תיצור לכם תיקיה בשם my-first-app ובתוכה תתקין את כל הדרוש על מנת לכתוב את האפליקציה הראשונה שלכם ב-React. כאשר ההתקנה תסתיים, הריצו את הפקודות:

```
cd my-first-app
```

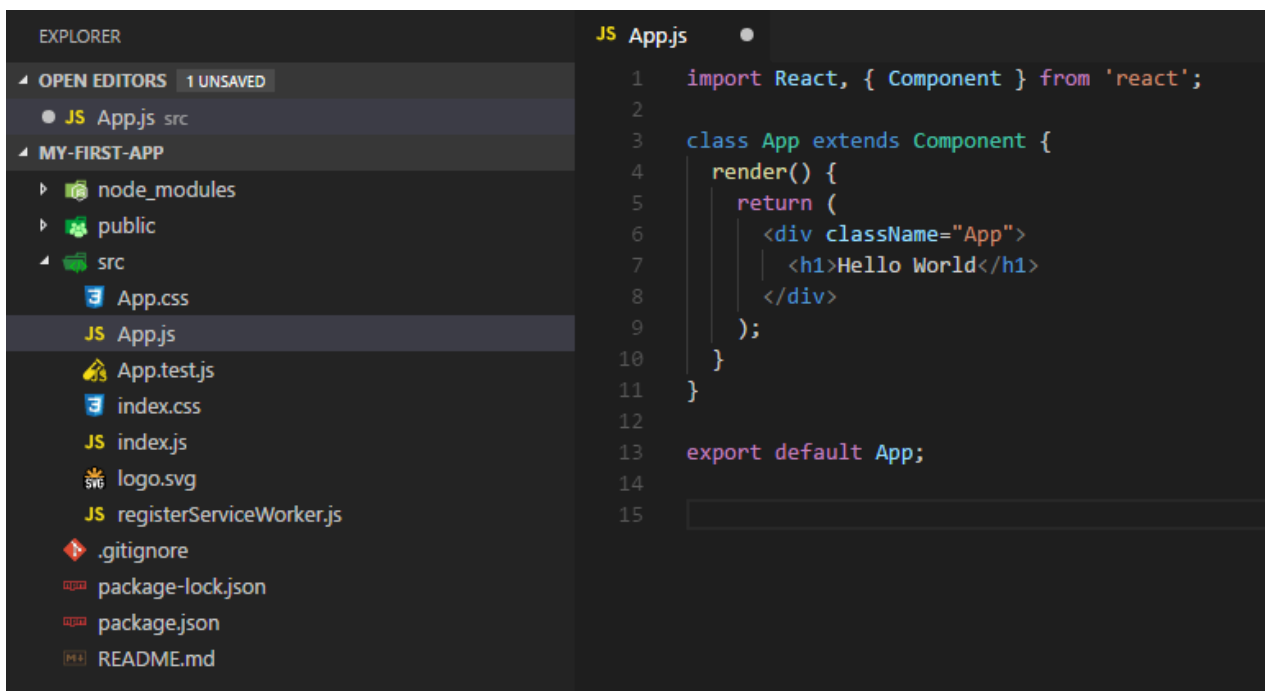
```
npm start
```

לאחר מספר שניות יפתח לכם דפדפן בכתובת localhost:3000 עליו נבצע את כל העבודה. מה שתראו על המסך הוא:



עכשיו, ניכנס לתיקיה ש-create-react-app יצרה לנו (my-first-app) ונערוך את הקובץ `src/App.js`.

נמחק את כל התוכן מה-`div` העליון ביותר בהיררכיה מהאלמנט שפונקציית `render` מחזירה ונכניס את השורה `<h1>Hello World</h1>` במקום. בנוסף נמחק את שורות ה-`import` מלבד השורה הראשונה בה אנחנו טוענים את React. נשמור את הקובץ. הקוד שלנו אמור להראות כך:



The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure for 'MY-FIRST-APP', with 'src/App.js' selected. The main editor area shows the content of 'App.js' with the following code:

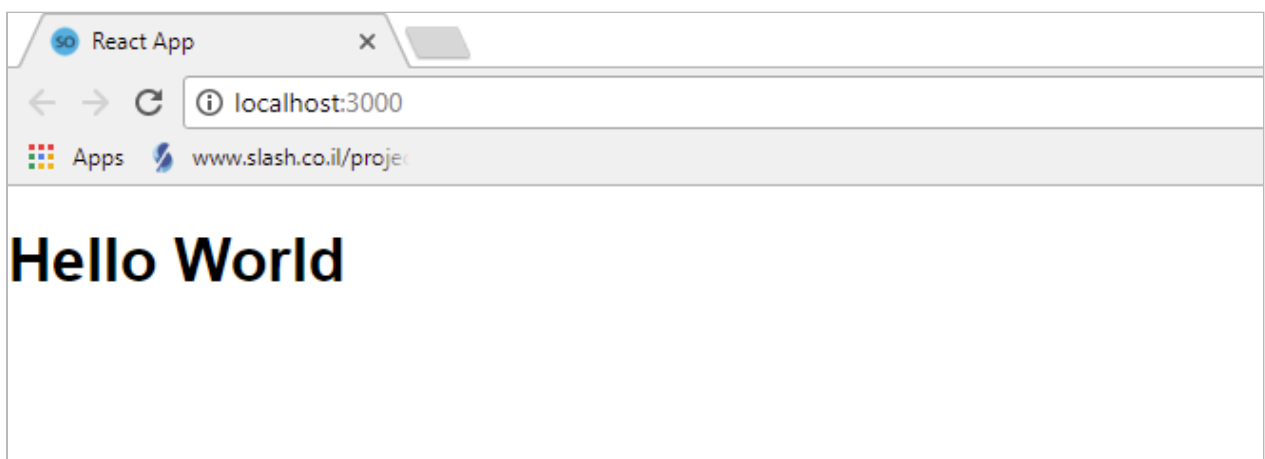
```

1  import React, { Component } from 'react';
2
3  class App extends Component {
4    render() {
5      return (
6        <div className="App">
7          <h1>Hello World</h1>
8        </div>
9      );
10   }
11 }
12
13 export default App;
14
15

```

נשים לב שעצם שמירת הקובץ תרענן את העמוד ב-`localhost:3000` ונראה מיד את השינויים על מסך הדפדפן זאת הודות ל-`webpack` ש-create-react-app התקין לנו בסביבת העבודה.

אנו אמורים לראות דף לבן עם המילים Hello World כמו בתמונה:



ברכות! הצלחתם לכתוב את ה-Hello World הראשון שלכם ב-React.

קצת על מה שקורה בקוד זה:

import – פיצ'ר של node.js לטעינת קוד מקבצים אחרים, לא נרחיב על פונקציה זו אבל נציין שהיא קיימת רק בסינטקס מתקדם יותר של node.js, שנקרא EcmaScript6. בעזרת Import אנחנו טוענים את היכולת לכתוב קומפוננטה של Component של React. למי שמכיר את העקרונות מאחורי תכנות מונחה עצמים, Component הוא בעצם סוג של אובייקט ובכל פעם שנכתוב קוד נרחיב את Component ונוכל לממש את השיטות שלו.

class – "הכרזה" על קומפוננט חדש עם שמו. כרגע הקומפוננט הראשי שלנו הוא App, אך הוא יוכל להכיל בתוכו קומפוננטות נוספות אותן ניצור. נשים לב שהמילה extends נמצאת בהכרזה על הקומפוננט ובכך אנו מכריזים על קומפוננט חדש של React.

export default – כמו ש-import טוען קוד, בעזרת export אנו בעצם אומרים ל-node.js איזה קטע קוד לייצא, כאשר נשתמש ב-import בקובץ אחר. המילה default משמשת על מנת להגדיר כי אם לא יוכרז שם קטע הקוד הספציפי אותו נרצה, זה קטע הקוד שיבחר כברירת מחדל.

render – פה בעצם React נכנס. כל קומפוננט React חייב להכיל פונקציה זו והיא קובעת איזה html יוחזר למסך בכל רגע נתון שאותו קומפוננט "חי". נשתמש במילה חי כיוון שבקרב נדבר על מחזור החיים של כל קומפוננט. נשים לב שבפרויקט זה הקומפוננט הראשי עליו אנחנו עובדים App תמיד "מרונדר" (מלשון render) למסך ראשון. הוא הקומפוננט העליון ביותר בהיררכיה וכל שאר האפליקציה תבנה בתוכו.

בניית אפליקציה פשוטה – ניהול רשימת קניות

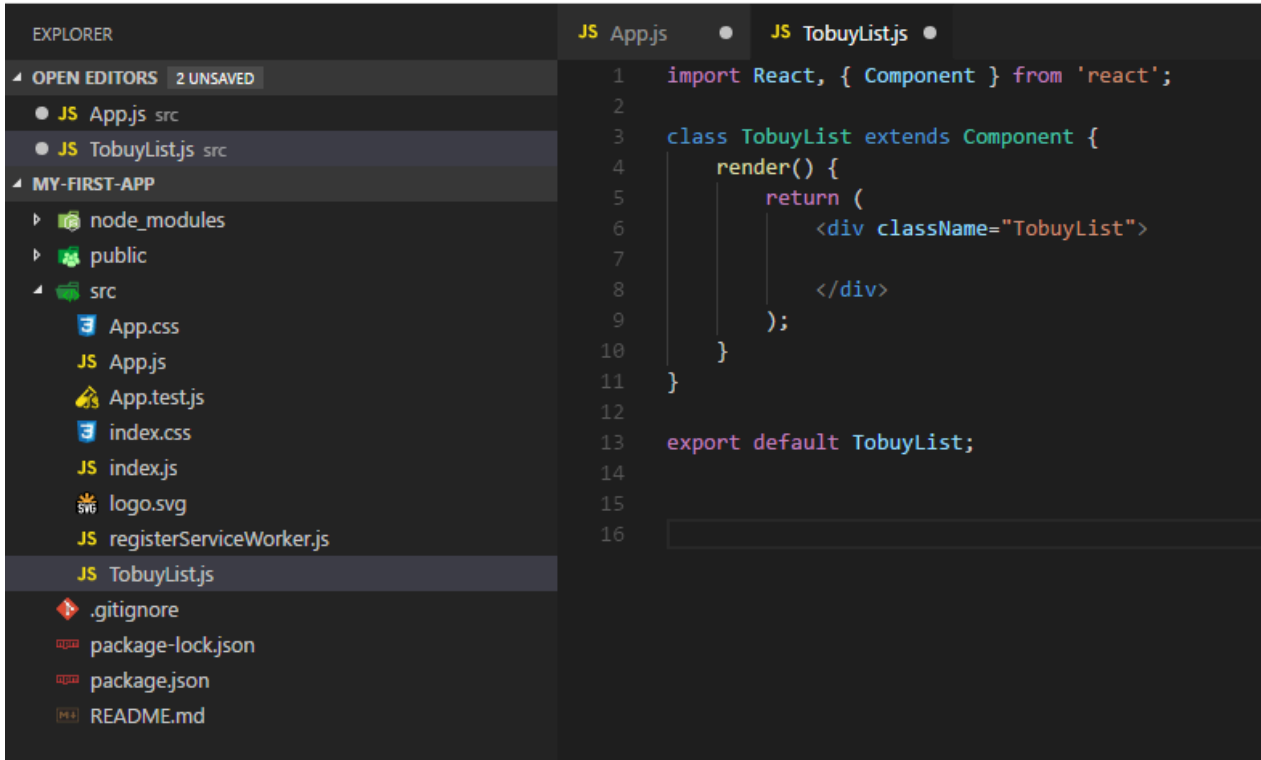
כעת נוכל להתחיל לכתוב את האפליקציה הראשונה שלנו.

נרצה לבנות רשימת קניות שתתמוך בפעולות:

1. הכנסת רשומה חדשה לרשימה.

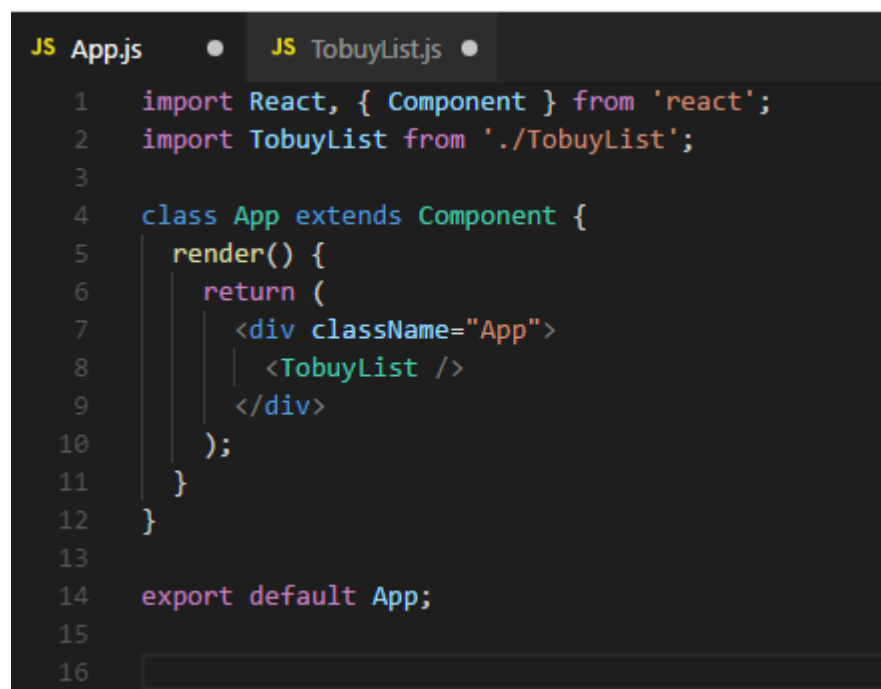
2. הסתרת הרשימה.

נפתח קובץ חדש בתיקיה שייקרא TobuyList.js ונעתיק אליו את כל הקוד שמופיע ב-App.js. נדביק בקובץ TobuyList.js ונשנה בכלך מקום שכתוב App את השם ל-TobuyList. בנוסף נמחק את אלמנט ה-h1. זה אמור להראות כך:



```
1 import React, { Component } from 'react';
2
3 class TobuyList extends Component {
4   render() {
5     return (
6       <div className="TobuyList">
7
8       </div>
9     );
10  }
11 }
12
13 export default TobuyList;
```

בקובץ ה-App שלנו נמחק גם כן את ה-h1 ונשתמש ב-import על מנת לטעון את הקומפוננט החדש שיצרנו נכניס ונרנדר אותו ל-DOM ע"י הכנסה שלו כאלמנט JSX כמו בתמונה:



```
1 import React, { Component } from 'react';
2 import TobuyList from './TobuyList';
3
4 class App extends Component {
5   render() {
6     return (
7       <div className="App">
8         <TobuyList />
9       </div>
10     );
11   }
12 }
13
14 export default App;
```

נחזור לקובץ `TobuyList.js`. נרצה להכניס ל-`div` הראשי שהוא מרנדר 2 כפתורים – אחד לצורך הסתרה / גילוי הרשימה ואחד לצורך הוספת רשומה לרשימה. בנוסף נרצה להכניס `input` שבו נרשום את שם הרשומה החדשה ורשימה `ul` שתכיל את איברי הרשימה. זה אמור להראות כך:

```

1  import React, { Component } from 'react';
2
3  class TobuyList extends Component {
4
5      render() {
6          return (
7              <div className="TobuyList">
8                  <input />
9                  <button>add</button>
10                 <button>hide/show</button>
11                 <ul>
12
13                 </ul>
14             </div>
15         );
16     }
17 }
18
19 export default TobuyList;
20
21
22

```

:props, state, virtual DOM

על מנת שנוכל להמשיך לכתוב בריאקט נעצור רגע וננסה קודם להבין קצת יותר טוב איך ריאקט עובדת:

1. כל קומפוננט יכול לקבל מהאב תכונות (props) ולגשת אליהם בקוד ע"י `this.props`. למשל אם היינו רוצים להחליט באיזה צבע תהיה הרשימה שלנו יכולנו בקומפוננט `App` לתת לקומפוננט `TobuyList` צבע בצורה:

```
<TobuyList listColor={'#17bc21'} />
```

2. ואז מתוך הקומפוננט לגשת לצבע ע"י `this.props.listColor`. נשים לב שאנחנו המצאנו את המפתח `listColor` ובכך אנחנו יכולים לתת לכל קומפוננט אילו תכונות שרק נרצה. בצורה זו ניתן למשל לרנדר מספר רשימות שלכל אחת צבע אחר והן מופרדות לחלוטין.

3. כל קומפוננט `React` יכול להחזיק `state`. ה-`state` הוא אובייקט שיכול להכיל המון מפתחות עם ערכים מכל הסוגים (משתנים בוליאניים, מספרים, מחרוזות ומערכים). וניתן לעדכן אותו ע"י פונקציה מיוחדת של `React` הנקראת `setState`. בכל מקום בקוד הקומפוננט ניתן לפנות למשתנים ב-`state` ע"י `this.state`.

4. מעבר ל-DOM אותו אנו רואים, ריאקט מחזיק ברקע הדפדפן אובייקט אותו אנו מכנים ה-Virtual DOM. אובייקט זה מכיל את כל מבנה ה-DOM שריאקט רינדד ובכל פעולת `setState` ריאקט יחפש בדיוק את האלמנטים של ה-html אשר תלויים ב-state ואותם יש לשנות. זו בעצם הגדולה של React מבחינת יעילות ו-Performance.

נראה איך אנו משתמשים ב-state בקומפוננט שכתבתנו. על מנת לאתחל את ה-state, נקרא לפונקציה שנקראת `constructor`. פונקציה זו נקראת ראשונה כאשר נוצר מופע חדש של הקומפוננט. בתוך פונקציה זו נגדיר את המצבים התחיליים של קומפוננט חדש שנוצר. הקוד יראה כך:

```

JS App.js      JS TobuyList.js x
1  import React, { Component } from 'react';
2
3  class TobuyList extends Component {
4
5      constructor(props) {
6          super(props);
7          this.state = {
8              newItemToAdd: '',
9              isOpen: true,
10             items: ['tomatoes', 'cucumbers', 'carrots']
11         }
12     }
13
14     render() {
15         return (
16             <div className="TobuyList">
17                 <input />
18                 <button>add</button>
19                 <button>hide/show</button>
20                 <ul>
21
22                 </ul>
23             </div>
24         );
25     }
26 }
27
28 export default TobuyList;
29
30
31

```

נשים דגש על כמה דברים:

1. הפונקציה `constructor` מקבלת בארגומנט את `props`. אנו נכתוב זאת תמיד כברירת מחדל כיוון

שבדרך זו תיהיה לפונקציית הבנאי גישה ל-props אם והם קיימים.

2. קריאה לפונקציה:

super (props)

נדרשת על מנת לקרוא לבנאי של קומפוננט כללי של ריאקט ממנו אנו ירשנו. למי שמגיע משפה מונחת עצמים כמו #C או JAVA הדבר כנראה מוכר. אם הדבר אינו מובן לכם אל דאגה, אין צורך להבין זאת בשלב זה.

3. אנחנו מאתחלים את האובייקט state בשלושה ערכים: משתנה בוליאני isOpen שמתחיל כ-true, ערך השם החדש שיתווסף לרשימה כאשר נלחץ add (יעודכן בכל פעם שנשנה את ה-input) אשר יוחזק ב-newItemToAdd המאותחל כמחרוזת ריקה ורשימת הקניות שלנו שהיא בעצם מערך הנקרא items ומכיל את הערכים עגבניות, מלפפונים וגזרים.

הפונקציה map ושימושה בתוך JSX:

כעת דבר ראשון שנרצה לעשות הוא להדפיס את רשימת הקניות התחילית שלנו: מלפפון עגבנייה וגזר. על מנת לעשות זאת נלמד קודם על הפונקציה map אשר שייכת למערכים בג'אווהסקריפט ונמצאת בשימוש רב כאשר כותבים ב-React. פונקציה זו מקבלת "פונקציית הפעלה" אשר מקבלת כקלט כל איבר במערך ומחזירה מערך של הפלטים למשל עבור המערך:

```
var arr = [1, 2, 3, 4]
```

אם נכתוב:

```
arr.map(num => num**2) (*)
```

נקבל את המערך:

```
[1, 4, 9, 16]
```

כלומר ריבועי המספרים.

ייתכן וחלק מקוראי המאמר אינם יבינו את בסינטקס אותו רשמתי בשורה (*). פונקציה הנרשמת בצורה זו מכונה arrow function או lambda function. באותה צורה ניתן לרשום במקום השורה (*) את השורה:

```
arr.map(function(item){return item**2})
```

אך מעבר לכך שלטעמי הצורה הראשונה אלגנטית יותר ומובנת, היא גם מכניסה בצורה אוטומטית את ההפרדה ועל this לחזור ה-scope ועל הפונקציה הפנימית

אזרחי זה מובעת מהעבודה שעל מנת שר אקט עבודה כמו שבוך והוא בן קלמנט אנוקס זהו 2025
אלמנט שהוא מייצר ע"י לולאה תחת הערך key. נתקן זאת ע"י שנביא לכל אלמנט ברשימה פשוט את
האינדקס שלו בסדר המערך, ניתן לעשות זאת ע"י הארגומנט השני בפונקציית map:

```
<ul>
  {this.state.items.map((item, index) => <li key={index}>{item}</li>)}
</ul>
```

עדכון ה-state

כאשר אנו בונים רכיבים בריאקט, אחד הדברים שאנו מעוניינים בהם הוא לדעת את מצב האפליקציה בכל מצב נתון. לכן, נרצה בכל שינוי של ה-input לשמור את הערך הנוכחי שבתוכו ב-state. על מנת לעשות זאת, נוסיף לקומפוננט פונקציה חדשה שתקרא:

updateNewItemToAdd(newVal)

אשר תקבל כקלט את הערך החדש newVal שבתוך ה-input ותיקרא כל פעם שה-input משתנה. העדכון יתבצע ע"י פונקציה מיוחדת לעדכון ה-state של ריאקט שנקראת setState. בנוסף, נוסיף ל-input את הפונקציה באיוונט onChange. נשים לב שה-C גדולה, זאת בגלל JSX.

```
JS App.js JS TobuyList.js x
1 import React, { Component } from 'react';
2
3 class TobuyList extends Component {
4
5   constructor(props) {
6     super(props);
7     this.state = {
8       newItemToAdd: '',
9       isOpen: true,
10      items: ['tomatoes', 'cucumbers', 'carrots']
11    }
12  }
13
14  updateNewItemToAdd = (newVal) => {
15    this.setState({ newItemToAdd: newVal })
16  }
17
18  render() {
19    return (
20      <div className="TobuyList">
21        <input onChange={e => this.updateNewItemToAdd(e.target.value)} />
22        <button>add</button>
23        <button>hide/show</button>
24        <ul>
25          {this.state.items.map((item, index) => <li key={index}>{item}</li>)}
26        </ul>
27      </div>
28    );
29  }
30 }
31
32 export default TobuyList;
```

33

נשים לב שרשמנו:

```
<input onChange={e => this.updateNewItemToAdd(e.target.value)} />
```

כלומר, בכל שינוי נקראת פונקציה שמחזירה פונקציה ומכניסה אליה את ערך ה-input. זאת לטעמי דרך אלגנטית יותר.

בנוסף נשים לב ש-setState מקבל אובייקט. בעזרת אובייקט זה ניתן לעדכן כמה ערכים ב-state "במכה אחת".

הוספת item לרשימה

כעת נכתוב את היכולת של הרשימה להוסיף item. ראשית נוסיף פונקציה לקומפוננט שתקרא addItem שתעדכן את המערך items אשר נמצא ב-state של הקומפוננט. נוסיף לכפתור add האטריביוט onClick ונכניס אליו את הפונקציה. על מנת לעשות זאת בצורה נכונה מבחינת זיכרון, כיוון שאין אנו מעוניינים לשנות את המערך ב-state ישירות ולהפריע ל-React, "נשכפל" את המערך items שנמצא ב-state, נדחוף אליו את ה-item החדש ואז נעדכן את ה-state:

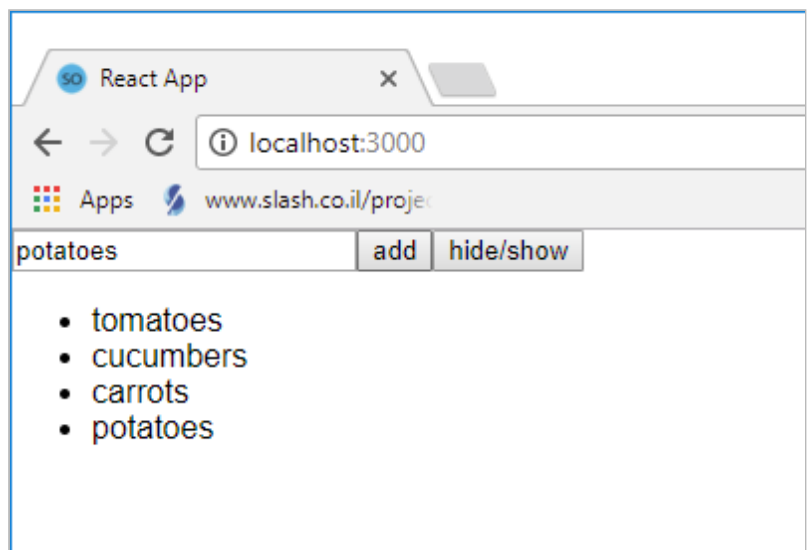
```
JS App.js JS TobuyList.js x
1 import React, { Component } from 'react';
2
3 class TobuyList extends Component {
4
5   constructor(props) {
6     super(props);
7     this.state = {
8       newItemToAdd: '',
9       isOpen: true,
10      items: ['tomatoes', 'cucumbers', 'carrots']
11    }
12  }
13
14  updateNewItemToAdd = (newVal) => {
15    this.setState({ newItemToAdd: newVal })
16  }
17
18  addItem = () => {
19    let clonedState = this.state.items.slice();
20    clonedState.push(this.state.newItemToAdd);
21    this.setState({ items: clonedState });
22  }
23
24  render() {
25    return (
26      <div className="TobuyList">
27        <input onChange={e => this.updateNewItemToAdd(e.target.value)} />
28        <button onClick={this.addItem}>add</button>
```

```

29       <button>hide/show</button>
30       <ul>
31         {this.state.items.map((item, index) => <li key={index}>{item}</li>)}
32       </ul>
33     </div>
34   );
35 }
36 }
37
38 export default TobuyList;
39

```

כעת נוכל לבדוק את האפליקציה שכתבנו. נכתוב המילה potatoes ב-input ונלחץ על add. ניתן לראות כי נוספה רשומה לרשימה עם שם המילה שכתבנו:



עדכון ה-DOM

נסביר מה בעצם קורה כאן: בכל פעם כאשר אנו מבצעים `setState`, ריאקט מחפש בדיוק איזה אלמנטים ב-DOM תלויים ב-state ומעדכן את האלמנטים הרלוונטיים. לכן, בכל פעם שאנו מעדכנים את המערך `items` בקומפוננט `TobuyList` פונקציית `render` נקראת ומעדכנת את הרשימה.

פונקציית הסתרת / הצגת הרשימה:

כעת נוסיף עוד פונקציונליות לרשימה שלנו. נרצה שבכל פעם שנלחץ על כפתור `hide/show`, הרשימה תוסתר, או לא תוצג לפי המצב הרלוונטי, ושהמילה הנכונה על הכפתור `hide / show` תופיע לפי המצב הרלוונטי.

נראה את הקוד ואז נסביר אותו:

```

1  import React, { Component } from 'react';
2
3  class TobuyList extends Component {
4
5      constructor(props) {
6          super(props);
7          this.state = {
8              newItemToAdd: '',
9              isOpen: true,
10             items: ['tomatoes', 'cucumbers', 'carrots']
11         }
12     }
13
14     toggleList = () => {
15         this.setState({ isOpen: !this.state.isOpen })
16     }
17
18     updateNewItemToAdd = (newVal) => {
19         this.setState({ newItemToAdd: newVal })
20     }
21
22     addItem = () => {
23         let clonedState = this.state.items.slice();
24         clonedState.push(this.state.newItemToAdd);
25         this.setState({ items: clonedState });
26     }
27
28     render() {
29         return (
30             <div className="TobuyList">
31                 <input onChange={e => this.updateNewItemToAdd(e.target.value)} />
32                 <button onClick={this.addItem}>add</button>
33                 <button onClick={this.toggleList}>{this.state.isOpen ? 'hide' : 'show'}</button>
34                 {this.state.isOpen ? <ul>
35                     {this.state.items.map((item, index) => <li key={index}>{item}</li>)}
36                 </ul> : ''}
37             </div>
38         );
39     }
40 }
41
42 export default TobuyList;
43

```

הסבר: הוספנו את הפונקציה toggleList אשר הדבר היחידי שהיא עושה, זה להפעיל את setState ולהחליף בין הערך הבוליאני של isOpen ב-state.

בנוסף, בפונקציית רנדר, את אלמנט הכפתור שמסתיר / מציג שינינו בצורה:

```
<button onClick={this.toggleList}>{this.state.isOpen ? 'hide' : 'show'}</button>
```

כלומר, לחיצה עליו מפעילה את הפונקציה toggleList והטקסט בתוכו הוא תנאי של המצב הנוכחי של isOpen ב-state. כלומר בכל פעם שה-state של isOpen משתנה, ריאקט ישנה את טקסט הכפתור לפי הערך הבוליאני הרלוונטי.

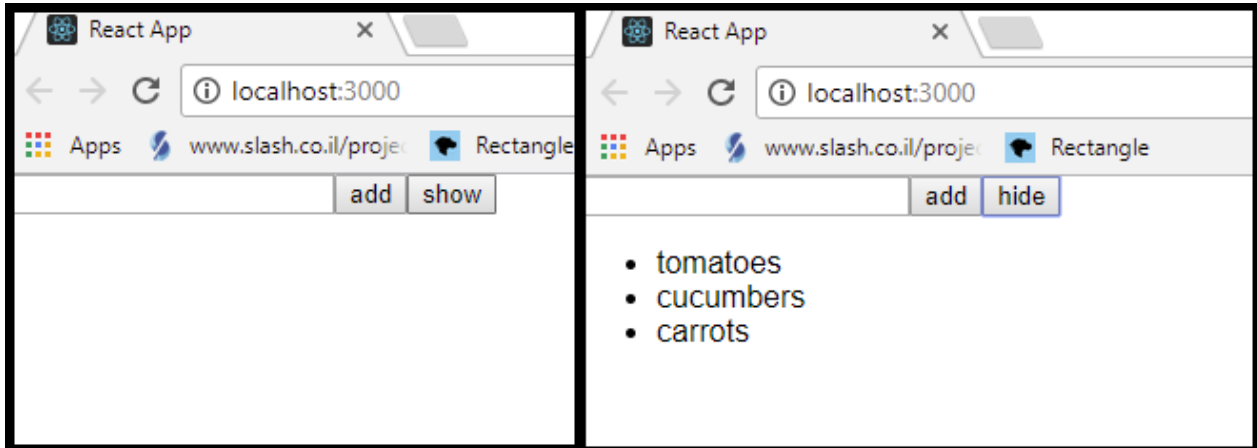
בנוסף, לרשימה ul שלנו, הוספנו את התנאי:

```
{this.state.isOpen ? <ul>
```

```
{this.state.isOpen ? <ul>
  {this.state.items.map((item, index) => <li key={index}>{item}</li>)}
</ul> : ''}
```

כלומר, בכל פעם שה-`state` של `isOpen` ישתנה, ריאקט יחליט האם להציג את הרשימה או להכניס מחזורת ריקה ל-DOM.

נשמור ונראה כי כעת הפונקציות נוספה:



מעבר למספר רשימות, צביעתם, וקביעת פריטים התחלתיים

כעת, כאשר יש לנו קומפוננט שמייצג רשימת קניות, נוכל להרחיב את האפליקציה שלנו על מנת ליצור מספר רשימות קניות. לכל רשימה יהיה:

1. שם.

2. צבע.

3. פריטים התחלתיים (כך שנוכל לשנות את הערכים tomatoes, cucumbers, carrots).

ראשית, נחזור לקובץ `App.js`. נשכפל את `TobuyList` שלוש פעמים. לכל קומפוננט נכניס ב-`props` את הצבע שלו, את שמו, ומערך של ערכים תחיליים בצורה:

```
JS TobuyList.js JS App.js x
1 import React, { Component } from 'react';
2 import TobuyList from './TobuyList';
3
4 class App extends Component {
5   render() {
6     return (
7       <div className="App">
8         <TobuyList
```



```

9      color={"#aeda6d"}
10     title={"fruits"}
11     initialData={['apples', 'bananas']}
12   />
13   <TobuyList
14     color={"#8bcfe8"}
15     title={"vegetables"}
16     initialData={['tomatoes', 'cucumbers', 'carrots']}
17   />
18   <TobuyList
19     color={"#e293b1"}
20     title={"sweets"}
21     initialData={['ice cream']}
22   />
23 </div>
24 );
25 }
26 }
27
28 export default App;
29

```

כעת, נחזור נעבור שוב לקובץ TobuyList. נעדכן ב-constructor את ה-items לאלו המגיעים מה-props ונוסיף ל-input שלנו placeholder שיכלול את כותרת הרשימה שלנו:

```

JS TobuyList.js x JS App.js
1 import React, { Component } from 'react';
2
3 class TobuyList extends Component {
4
5   constructor(props) {
6     super(props);
7     this.state = {
8       newItemToAdd: '',
9       isOpen: true,
10      items: this.props.initialData
11    }
12  }
13
14   toggleList = () => {
15     this.setState({ isOpen: !this.state.isOpen })
16   }
17
18   updateNewItemToAdd = (newVal) => {
19     this.setState({ newItemToAdd: newVal })
20   }
21
22   addItem = () => {
23     let clonedState = this.state.items.slice();
24     clonedState.push(this.state.newItemToAdd);
25     this.setState({ items: clonedState });
26   }
27
28   render() {
29     return (
30       <div className="TobuyList">
31         <input placeholder="add " + this.props.title onChange={e => this.updateNewItemToAdd(e.target.value)} />
32         <button onClick={this.addItem}>add</button>
33         <button onClick={this.toggleList}>{this.state.isOpen ? 'hide' : 'show'}</button>
34         {this.state.isOpen ? <ul>
35           {this.state.items.map((item, index) => <li key={index}>{item}</li>)}
36         </ul> : ''}
37       </div>
38     );
39   }
40 }
41
42 export default TobuyList;

```

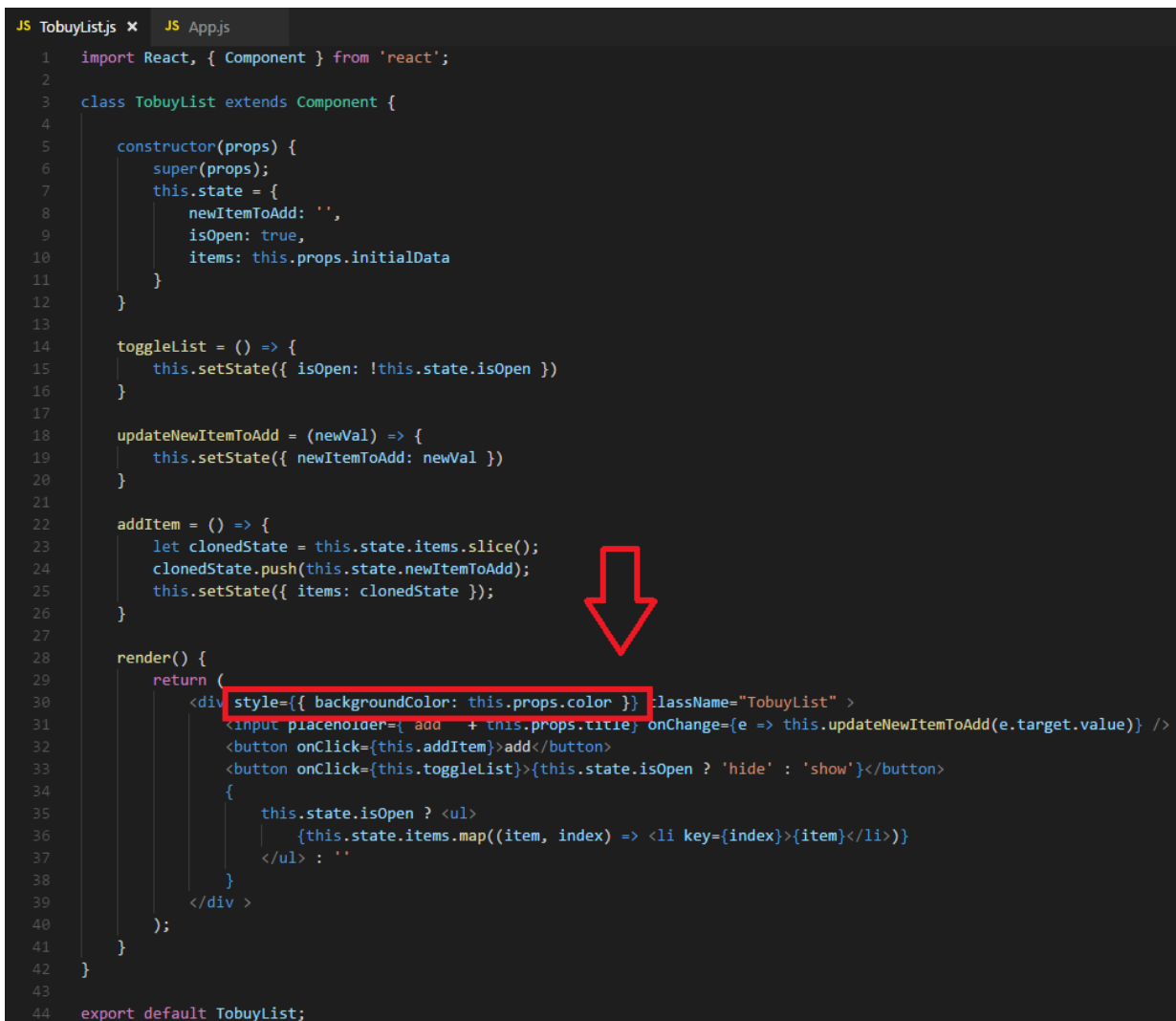
הוספת inline-style בריאקט:

דבר אחרון שנעשה הוא נלמד איך ניתן לתת ב-React ערכי css בתוך הקומפוננט. נרצה שכל רשימה תקבל את הצבע אותו נתנו לה ב-props.

הערה חשובה: יש לציין שדבר זה באופן בסיסי אינו מומלץ וב-99% מהמקרים רצוי ועדיף לכתוב את ה-css בצורה חיצונית שתלוית מחלקות css.

נוסיף ל-div הראשי שהקומפוננט TobuyList מרנדר, attribute בשם style. ה-attribute הזה מקבל אובייקט שמכיל בתוכו מפתחות וערכים הזזים ל-css. השינוי היחידי הוא שמפתחות עם יותר ממילה אחת אינם נרשמים בחיבור ע"י '-' ובמקום זה נרשמים בסגנון camelCase.

נכניס לאובייקט תחת המפתח backgroundColor את הערך מה-props:

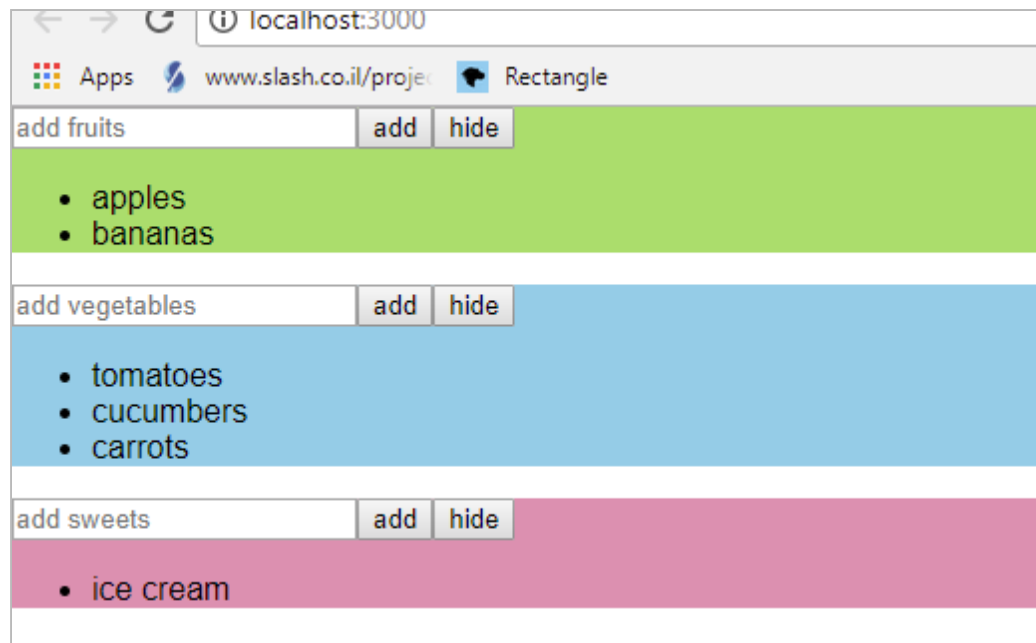


```

1  import React, { Component } from 'react';
2
3  class TobuyList extends Component {
4
5      constructor(props) {
6          super(props);
7          this.state = {
8              newItemToAdd: '',
9              isOpen: true,
10             items: this.props.initialData
11         }
12     }
13
14     toggleList = () => {
15         this.setState({ isOpen: !this.state.isOpen })
16     }
17
18     updateNewItemToAdd = (newVal) => {
19         this.setState({ newItemToAdd: newVal })
20     }
21
22     addItem = () => {
23         let clonedState = this.state.items.slice();
24         clonedState.push(this.state.newItemToAdd);
25         this.setState({ items: clonedState });
26     }
27
28     render() {
29         return (
30             <div style={{ backgroundColor: this.props.color }} className="TobuyList" >
31                 <input placeholder={ add + this.props.title } onChange={e => this.updateNewItemToAdd(e.target.value)} />
32                 <button onClick={this.addItem}>add</button>
33                 <button onClick={this.toggleList}>{this.state.isOpen ? 'hide' : 'show'}</button>
34                 {
35                     this.state.isOpen ? <ul>
36                         {this.state.items.map((item, index) => <li key={index}>{item}</li>)}
37                     </ul> : ''
38                 }
39             </div>
40         );
41     }
42 }
43
44 export default TobuyList;

```

כעת נשמור את שני הקבצים, ונראה את התוצאה:



לסיכום

במדריך זה למדנו לבנות את אבן הלגו הבסיסית של ריאקט - הקומפוננטה. למדנו איך ניתן לבנות רכיבים אשר אינם תלויים אחד בשני (מודולריים) שהופכים את התכנות של אפליקציות גדולות לקל ביותר.

במה לא נגענו?

ישנם דברים רבים שלא נגענו בהם אך בין הדברים החשובים עליהם לא הרחבנו הוא מחזור החיים של כל קומפוננטה, ישנם פונקציות מובנות בריאקט אותן ניתן לתת לכל קומפוננטה כמו `componentWillMount` או `componentWillUnmount` אשר קובעות איזה פונקציות יופעלו לפני העלייה או ירידה של קומפוננט מה-DOM.

ניתן ללמוד על כך בהרחבה בלינק זה (<https://reactjs.org/docs/state-and-lifecycle.html>).

