מדריך React.js

תוכן עניינים

התקנת תשתית React.js

עולם אתרי האינטרנט השתנה. אנו חיים במציאות חדשה בכל הנוגע לבניית אתרי אינטרנט.

מצד אחד, עלינו להעניק חווית משתמש טובה ומצד שני, לגרום לפיתוח שלנו להיות פשוט יותר ולא מורכב דיו.

בשנת 2003 מתכנתי פייסבוק המציאו את ReactJs , ספריית UI ופקדים, אשר נותנת מענה לשני תובנות אלו וגורמת לקוד שלנו להיות יעיל יותר. מדובר בframework שמתלבשת על javasript ובתוכה נוכל כבר לכתוב תגיות html, לשכפל אותן בקלות ולבנות את האתר שלנו בפשטות. זה הזמן לשבור את התכנות המסורתי בו ביצענו קובץ html נפרד, קובץ css נפרד וקובץ js נפרד.

על מנת להתחיל לעבוד עם React, אנא התקינו את סביבת הפיתוח של Visual Studio Code.

ובנוסף, צרו תיקייה חדשה בשולחן העבודה עבור הפרויקט שלכם.

לאחר מכן, כיתבו `cmd` בשורת החיפוש של התקייה שפתחתם ולפניכם תופיע חלונית שחורה הממתינה לפקודה מכם.

בתוך חלונית זו, אנא כיתבו את השורה הבאה:

```
C:UsersYour Name>npx create-react-app myfirstreact
```

כעת, יתחילו לרוץ ברקע שלל הקבצים הנלווים הדרושים להקמת אתר אינטרנט תומך React וכן גם שרת localhost שדרכו נוכל להריץ את כל הסקריפטים החדשים. אמרנו כבר שזהו עידן חדש ולא מדובר בjavascript רגיל כפי שהכרנו.

לסיום, כאשר התקנת שלל הקבצים הסתיימה, נוכל לראות כי נוצרה לנו תקייה חדשה ובה כלל הקבצים. ניגש לסביבת הפיתוח שלנו visual studio code , ונפתח את הפרויקט בכפתור `file`

ובתוכו על `open folder` .

תוך רגעים יופיעו לפניכם שלל קבצי הפרויקט ובראשם הקובץ `App.js` .

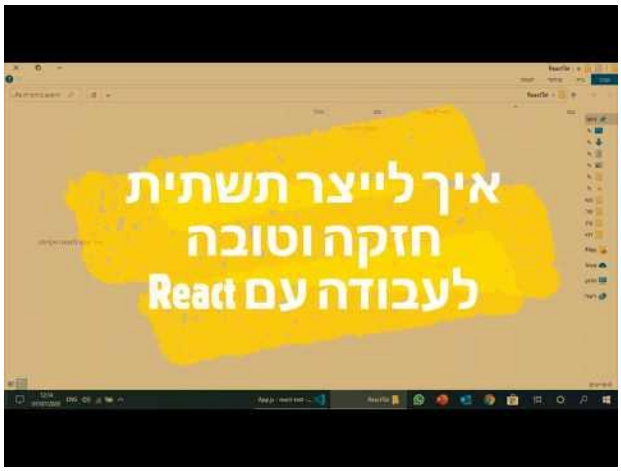
כעת, על מנת להריץ את האתר שנוצר לנו בשרת המקומי שנוצר במחשב, נצטרך ללחוץ על `ctrl + j`

ובתוך תוכנת הvsc שלנו ולכתוב בשורת הconsole:

```
npm start
```

האתר שלנו יופיע בדפדפן ונוכל לראות את קטע הReact הראשון שלנו. עתה נוצרה עבורינו האפשרות לעבוד עם React.

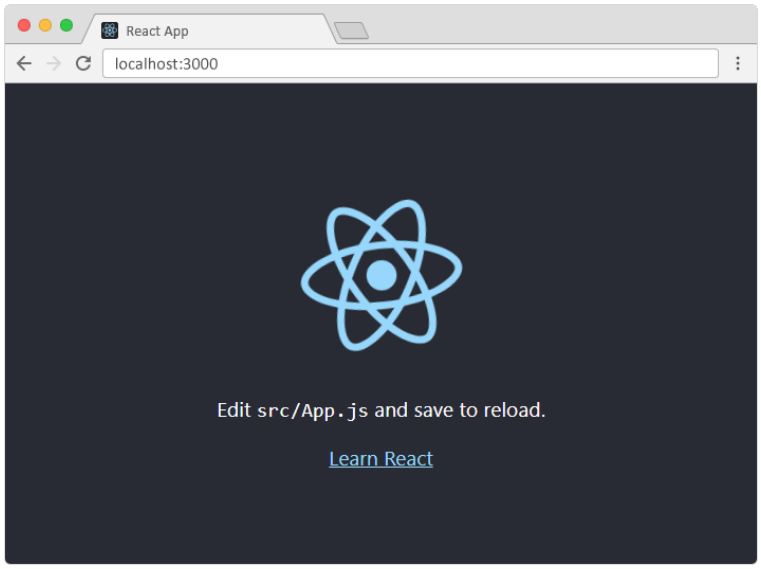
הצטרפו אליי גם לשיעורים הבאים במדריך וגלו איך משתמשים בReact.



משתנים בReact

בשיעור הקודם השארנו את פרוייקט הReact הראשון שלנו בטמפלט ברירת-המחדל כאשר הוא מציג Gif מסתובב בצורה ויזואלית והקוד עצמו מוכן לפעולה.

ממצב זה, נוכל לגשת לקוד שלנו ובעיקרו אצל הקבצים `index.js` , `index.html` ואל `App.js`.



כידוע לנו מכתיבה בHtml, תמיד יש לנו את קובץ הindex שבו נזין את כל הדף הראשי של האתר. בReact יש לנו את הindex הרגיל בו ניצור תחילה תגית div אחת בלבד עם `id='root'`.

לתוך div זה נרצה להזין את כל האתר שלנו שכתוב בReact.

איך נעשה זאת? באמצעות רפרנס בדף `index.js`.

ההקבלה הכי קרובה לכך מעולמות הjs המוכרים זה הפונקציה של innerHTML בה אנחנו מזינים סטרינג חדש (שיכול להכיל אפילו תגיות, ראו ערך בשיעורי הג'אווה סקריפט) ישירות אל תוך div ספציפי שבחרנו.

ההקבלה עלינו אנחנו מדברים שקיימת בדף `index.js` היא באמצעות 5 שורות הקוד הבאות:

תחילה, נבצע import לרכיבי React ואז לאחר מכן נייצר רפרנס לdiv שלנו ונשפוך אליו את קוד הReact באמצעות פונקציית `ReactDOM.render` –

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

const rootElement = document.getElementById('root');
ReactDOM.render(<App />, rootElement);
```

שימו לב, בשורה האחרונה בתוך הפונקציה של `ReactDOM.render` ישנם 2 פרמטרים.

האחד הוא קריאה לרכיב React שאותו נרצה לשפוך לתוך הdiv והשני הוא הdiv עצמו מתוך הhtml. בואו ניגש ליצירת הרכיב עצמו – `App.js`.

בתוך קובץ `App.js` נוכל לייצר לראשונה קוד ריאקטיבי שאותו נראה בתוך div הroot שלנו בדף html הראשי. תוכלו להבחין כי שורות הקוד בתוך קובץ זה, הן אותו הgif המסתובב והכיתוב שהופיע תחילה בתוך הדף שנפתח עבורכם. זו ההזדמנות לשנות את הטקסט הבא –

```
Edit <code>src/App.js</code> and save to reload.
```

לטקסט hello world ולראות שאכן טקסט זה משתנה לכם בlocalhost שנפתח.

על פניו, להגיע לנקודה זו היה די פשוט.

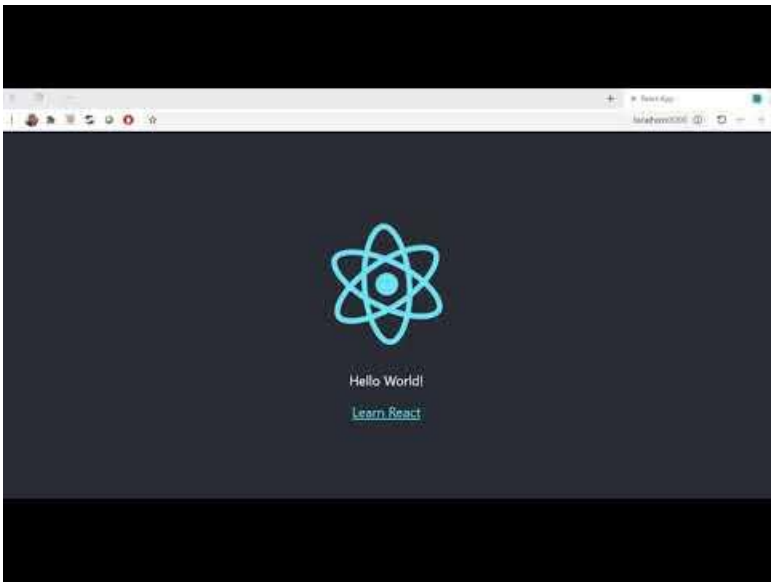
אז כעת, נמחק את כל התוכן שכתוב בתוך הreturn ונתחיל מחדש (ראו ערך בסרטון המצורף).

לאחר הצפייה בסרטון, נסו בעצמכם לענות על שלוש השאלות הבאות ולהציג אותם בתוך הdiv.

1. הציגו את הממוצע של המספרים 3,6,9
2. בידקו האם המודולו של 25 הוא זוגי או אי זוגי
3. הציגו את לוח הכפל

שימו לב שעליכם לכתוב את המשתנים מחוץ לreturn אבל בתוך הfunction!

תנו מבט בפתרון בתחילת סרטון מספר 3! בהצלחה!



יצירת קומפוננטה ראשונה

עולה השאלה למה כל כך הרבה מפתחי front-end בעולם הweb מעדיפים את הפיתוח בReact ולא באף ספרייה אחרת? בשיעור זה, נתחיל לצלול פנימה קצת יותר ולהבין מה כל כך מיוחד בReact. הפעם נילמד איך לייצר תבנית מסוימת של כרטיסייה (ניצור קובץ בשם Card.js) , נשכפל אותו מספר פעמים וניצור דף עם מספר כרטיסיות שנשענות על אותה תבנית שחוזרת על עצמה.

ראשית, עלינו לייבא בקובץ js חדש את ספריית הקומפננטה מתוך React –

```
import React, { Component } from 'react';
```

ולאחר מכן, נגדיר את שם הclass שלנו (במקרה זה Card) וכבר בשורה זו נייצא אותו מכאן (ראו ערך בשיעור הקודם, בו ייצאנו את הקומפוננטה בסוף הjs).

```
export default class Card extends Component {

}
```

בתוך הclass שלנו, נוסיף את הפונקציה המרכזית דרכה נעבוד ונזין את כל הנתונים - render.

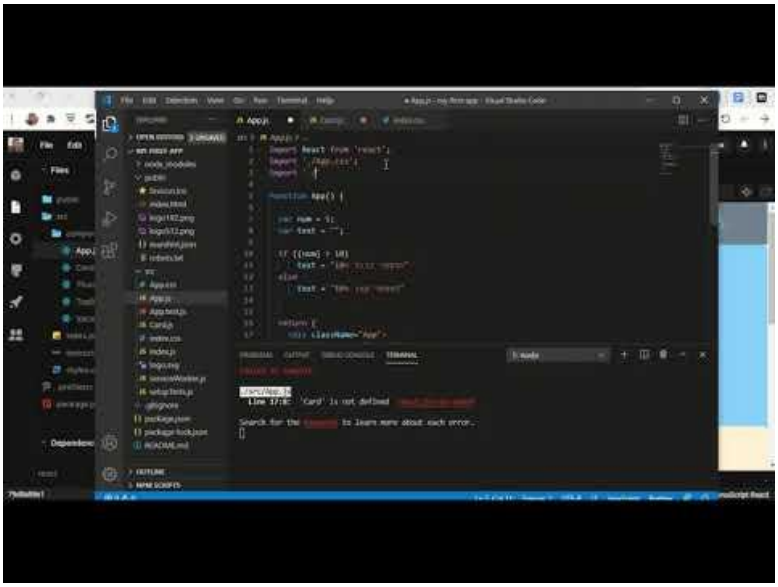
```
render() {

}
```

בתוך פונקציית זו אנו מזינים את כל התגיות והמידע שנרצה להציג בסופו של דבר. ראשית, ניצור שורה אחת עם כל המידע הקבוע שעתיד להיות (בסגנון של יצירת אובייקט וקונסטרקטור. ראו ערך, שיעורי לוגיקה).

```
const { country } = this.props;
```

לסיום, ניצור איזור חדש בשם return ובתוכו נחזיר את כל הdata שנרצה להציג בכל פעם שנקרא לCard. למשל, כמו בסרטון , נרצה להחזיר תגית div ובתוכה תגית h1 עם השם של המדינה המשתנה. בסרטון גם נעצב קלות את הCard. כאשר ניצור בבוא העת את הכרטיסיה שלנו, נצטרך להעניק לה attribute בשם country (כמו שמעניקים attribute רגיל בתגית Html) ונצטרך רק לשנות זאת. ראו בסרטון לדוגמא וצרו בעצמכם 5 כרטיסיות עם שמות של מדינות.



props

מה זה Props?

אנחנו מכירים את הattr שיש בעולם הhtml.

כמו src וalt בתגית img או href לa... כעת, בתוך שורות הקריאה לקומפוננטות, שאנחנו מייצרים בApp.js למשל, אנחנו יכולים להוסיף attr (תכונות למעשה) שנרצה לשלוח ישירות לתוך הקומפוננטה עצמה. זו השיטה הקלאסית והמוכרת לשלוח משתנים מהקריאה לקומפוננטה ישירות לתוך הטמפלט שבנינו ובכל פעם שנשכפל אותה כעת, נוכל להציג מידע חדש. זו גם הסיבה שריאקט כל כך נפוצה ופופולארית. היעילות שלה והשיטתיות עולה מעל לכל דמיון בהקשר חיסכון בכתיבת שורות הקוד (נסו להיזכר כיצד היינו כותבים את ארבעת הכרטיסיות המופיעות בסרטון בעידן הHTML המקורי... כמה שורות נחסכו מאיתנו למעשה בשיטה הזו?)

איך אנחנו שולחים מידע מהapp.js לקומפוננטה?

לאחר יצירת הקומפוננטה שלנו, למשל Card - אשר מכילה שני טקסטים (אחד H1 ומתחתיו H3) וכן גם עיצבנו את הCard שלנו (צבע רקע, מסגרת border וכו') נוכל להתקדם לשלב הבא. השלב הבא כאמור, ליצור בין 4-5 קומפוננטות שמכילות את מדינות העולם ואת מזג האוויר שלהן (ניתן להמציא). איך עושים זאת? מוציאים attribute בדיוק כמו בHTML(ראו ערך בסרטון) בתוך הרפרנס של הקריאה לקומפוננטה (למשל: `country='israel'`)

ובתוך הקומפוננטה עצמה, עלינו ליצור בתוך הrender שלנו, אבל מעל הreturn, ייבוא של הprops שלנו (של התכונות שכתבנו...)

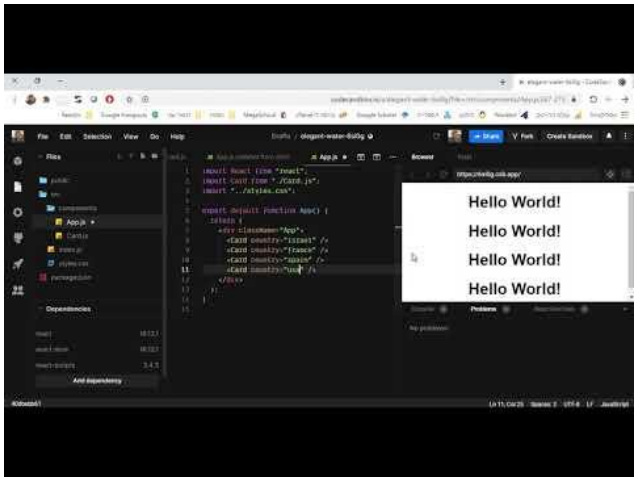
```
const {country} = this.props
```

הכוונה שבשורה זו היא שאנחנו מייצרים משתנה קבוע בשם country בסוגריים מסולסלים (דהיינו, פונקציה של Js. זהו לא משתנה רגיל.) אשר מכיל בתוכו את הprops שקיבל מהקריאה ממקודם. כמובן לא לשכוח בתוך הH1 לשנות את הטקסט שהיה כתוב קודם לכן ל{country}...

תרגול

זהו, סיימנו את ההסבר. עכשיו ברור ומובן מה עליכם לעשות, נכון? צרו רשימת כרטיסיות עם שמות מדינות העולם בהם טיילתם, מזג האוויר שלהן (ניתן להמציא) ותמונת דגל המדינה/תמונה המתארת את מקום בעולם.

בונוס: בנוסף למדינות העולם ומזג האוויר, ייבאו גם את תמונת המדינה.



state

מה זה State?

בשיעור זה נלמד על בשורה נוספת שמביאה עימה ReactJs והיא הState.

כמו שכל קומפוננטה יכולה להחזיק בprops, כך יש לה אפשרות גם לstate.

רק שהפעם מדובר במשתנים מקומיים של אותה הקומפוננטה אשר יכול להכיל המון מפתחות עם ערכים מכל הסוגים (משתנים בוליאניים, מספרים, מחרוזות ומערכים).

רגע, אבל מה החידוש פה? הרי למה אנחנו צריכים עוד משתנים אם יש לנו את הprops?

ממבט עיני המשתמש, יש בstate אפשרות מעבר ל-DOM אותו אנו רואים כמו בprops.

יש את עניין הרינדור האוטומטי. הכוונה היא שיש אובייקט משודרג בשם Virtual DOM אשר מכיל את כל מבנה ה-DOM שריאקט רינדר ובכל פעולת setState ריאקט יחפש בדיוק את האלמנטים של ה-html אשר תלויים ב-state ופשוט ישנה אותם אונליין בצורה אוטומטית מבלי שהמשתמש ירענן את הדף כל פעם.

זוהי הבשורה הנוספת שריאקט מביאה עימה!

איך אנחנו מעדכנים את המסך בעת שינוי בrender?

ניתן לעדכן אותו ע"י פונקציה מיוחדת של React הנקראת setState ובכל מקום בקוד הקומפוננט ניתן לפנות למשתנים ב-state ע'י this.state.

כעת, ניצור מעל פונקציית הrender שלנו אזור של state. הסינטקס הוא:

```
state = {
};
```

בסוף הסוגריים המסולסלים נוכל לכתוב את שם המשתנה שנרצה, למשל count.

הפעם נכתוב `count: 0` אין צורך בנקודה פסיק ; בהפרדת הערכים בתוך הstate, אלא להשתמש בפסיק. שימו לב גם לנקדתיים.

כעת, בתוך תגית הh1 שלנו נוכל לכתוב `{this.state.count}`

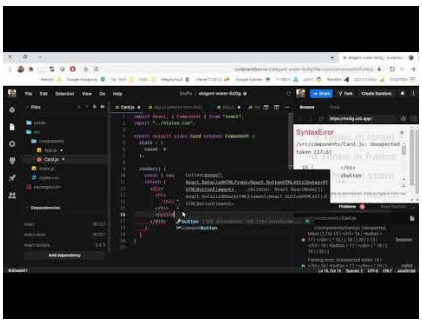
נזכיר שנית, על מנת לערוך את ערכי הstate בלייב על ידי המשתמש, נצטרך ליצור כפתור עם אפשרות לחיצה ובפונקציה פנימית נשתמש בפונקציה של `setState`

ראו ערך הדוגמא הבאה וכן גם בסרטון המצורף.

```
<button onClick={this.addCount}>
Add One
</button>
```

וכמובן לא לשכוח את הפונקציה עצמה...אותה נשים מתחת לstate ומעל הrender!

```
addCount = () => { this.setState({ count: this.state.count + 1 }); };
```



שליחת מידע בין קומפוננטות

שליחת המידע בין קומפוננטות מתבצעת כפונקציה כpropsוarrow function.

בשיעור זה נלמד כיצד לשלוח מידע בין קומפוננטות. נניח ויש לנו בApp.js שלנו גם קומפוננטה בשם Title וגם קופמוננטה רגילה, מהשיעור הקודם, בשם Card. את הCard שמציג את שם המדינה וcounter שבלחיצה קופץ ב1, ראינו בשיעור הקודם. אבל בTitle נרצה להציג את שם המדינה עליה לחצנו הרגע. כלומר, שהכפתור שלנו יבצע פעולה כפולה. (גם העלאה ב1 בתוך הCard וגם שינוי של הTitle).

שליחת מידע בין קומפוננטות (פונקציה כpropsוarrow function) בשיעור זה נלמד כיצד לשלוח מידע בין קומפוננטות. נניח ויש לנו בApp.js שלנו גם קומפוננטה בשם Title וגם קופמוננטה רגילה, מהשיעור הקודם, בשם Card. את הCard שמציג את שם המדינה וcounter שבלחיצה קופץ ב1, ראינו בשיעור הקודם. אבל בTitle נרצה להציג את שם המדינה עליה לחצנו הרגע. כלומר, שהכפתור שלנו יבצע פעולה כפולה. (גם העלאה ב1 בתוך הCard וגם שינוי של הTitle). על מנת לבצע תהליך זה, ראשית נרצה לשלוח כפרמטר props גם פונקציות, ולא רק משתנים פרימטיביים (כמו String שביצענו בשיעור הקודם). אנו מבינים כי כעת נרצה לשלוח משתנה מהstate של הapp (כמו שלמדנו בשיעור הקודם) אבל גם לשנותו בעת לחיצה על הכפתור בתוך הCard ולשלוח לTitle (אכן, העניינים יכולים להסתבך...) אז איך עושים זאת? ראשית, נזכיר משיעורי הג'אווה סקריפט על arrow function ואז נתקדם לכתיבת הקוד עצמו (ראו ערך בסרטון).

מה זה Arrow Function?

סכמת הכתיבה בשיטה זו היא, כשמה כן היא, שם הפונקציה, לאחריו צורת חץ (arrow) => ולציין את הפרמטרים שנרצה לשלוח. בנוסף, במקום לכתוב function וכן גם return בתוך הפונקציה. הקוד שלנו נהיה יעיל יותר, מקוצר יותר ונוח יותר. בואו נראה דוגמא: זוהי פונקציה רגילה לחלוטין שמקבלת שני פרמטרים ומחזירה את החיבור ביניהם...

```
var sum = function(num1, num2) {
return num1 + num2;
}
```

בשיטת הarrow function זה ייראה כך - (זיכרו, אמרנו ללא function וללא return. רק שם הפונקציה, הפרמטרים והחץ...)

```
var sum = (num1, num2) => num1 + num2;
```

שימוש בthis

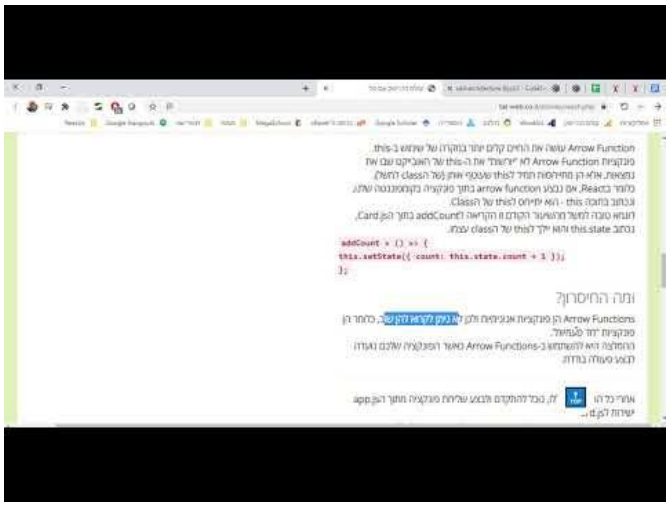
Arrow Function עושה את החיים קלים יותר במקרה של שימוש ב-this. פונקציות Arrow Function לא 'יורשות' את ה-this של האובייקט שבו את נמצאות, אלא הן מתייחסות תמיד לthis שעוטף אותן (של הclass למשל). כלומר בReact, אם נבצע arrow function בתוך פונקציה בקומפוננטה שלנו, ונכתוב בתוכה this - הוא יתייחס לthis של הClass. דוגמא טובה למשל מהשיעור הקודם זו הקריאה לaddCount בתוך הCard.js, נכתוב this.state והוא יילך לthis של הclass עצמו.

```
addCount = () => {
this.setState({ count: this.state.count + 1 });
};
```

ומה החיסרון?

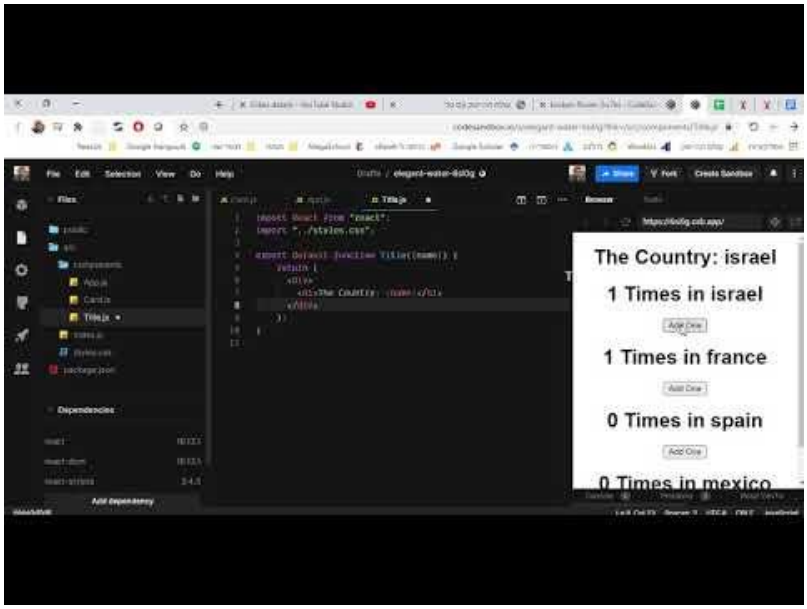
Arrow Functions הן פונקציות אנונימיות ולכן לא ניתן לקרוא להן שוב, כלומר הן פונקציות 'חד פעמיות'. ההמלצה היא להשתמש ב-Arrow Functions כאשר הפונקציה שלכם נועדה לבצע פעולה בודדת.

אחרי כל החידודים הללו, נוכל להתקדם ולבצע שליחת פונקציה מתוך הapp.js ישירות לcard.js. נסו בעצמכם וצפו בסרטון שלנו עם הפתרון המלא - איך ניתן לשלוח פונקציה מהapp.js ישירות ללחיצת הכפתור בתוך הcard.js לאחר מכן, כשלחצנו על הכפתור נרצה לשנות את שם המדינה בstate של הapp.js ולשלוח אותה לtitle.js. בהצלחה!!



קבלת props בשם הclass (destructuring)

עד כה קיבלנו את הpropsשלנו מתוך קריאות שונות שביצענו (בין אם בתוך הפונקציות כמו בפונקציית הלחיצה בCard.js או בתוך הreturn של הrender לתוך משתנה const) ריבוי הקריאות לpropsיוצרות סימני שאלה... האם בכל פעם נצטרך לקרוא לprops? אי אפשר להגדיר כמשתנה גלובלי ראשי ופשוט לשלוף משם? בשיעור זה נלמד על שיטת הdestructuring אשר כבר בתוך שם הclass שלנו נוכל לייצר את ייבוא הprops. כמו כן, בסרטון נעבור שלב אחרי שלב איך אנחנו מגיעים למצב של המימוש דרך arrow function (ראו ערך שיעור קודם). צפו בסרטון ונסו בעצמיכם לשנות את שם הclass של Card.js לשם עם arrow function המקבל את הpropsשלו.



לפני

```
export default class Title extends Component {
const {name} = this.props;
render(){
return (
<h1>The Country: {name}</h1>
)
};
};
```

אחרי

```
export default ({ name }) => {
return (
<h1>The Country: {name}</h1>
);
};
```

מערכים ופונקציות map

הרבה שואלים מהי פונקציית callback והתשובה בפשטות היא שcallback זהו פשוט שם לפונקציה שמועברת כפרמטר לפונקציה אחרת ומופעלת בתוכה.

למשל, כאשר אנו רוצים ליישם פונקציית sum שמכילה חישוב של שני מספרים. רק שנרצה שהיא תתרחש בשלב מאוחר יותר. ראו ערך בדוגמא הבאה:

```
function greeting(name) {
alert('Hello ' + name);
}

function processUserInput(callback) {
var name = prompt('Please enter your name.');
callback(name);
}

processUserInput(greeting);
```

(שוב, לרוב נשתמש בקריאה לפונקציה כלשהי כפרמטר בפונקציה אחרת - בגלל שהאירוע עליו אנחנו רוצים לקבל את המידע עדיין לא התרחש - כמו ששלחנו פונקציה כprops בשיעור הקודם.)

אחרי שהבנו את הרעיון של שליחת פונקציה כפרמטר לפונקציה אחרת, נוכל להתקדם הלאה ולהיזכר בכל הנוגע למערכים (ראו ערך בשיעור מערך במדריך הלוגיקה) הרעיון שמוביל אותנו הוא יעילות כתיבת הקוד שלנו בReactJs. אנחנו מודעים לכך שעלינו לייצר קומפוננטה אחת, לקרוא לה בApp.js כמספר הפעמים שנרצה להציגה ופשוט לשנות בכל קריאה את הpropsשלה.

לעיתים, כאשר מדובר במספר רב של אייטמים, עלינו יהיה לייצר משתנה אשר יאגד מערך של אובייקטים. שיטה זו תהיה אפקטיבית יותר משום שברעיון הסופי, נוכל כעת לקרוא לקומפוננטה פעם אחת ובכל פעם (כמו במעין לולאת for עם אינדקס) לייצר את הקומפוננטה עם הערכים לפי מספר האייטם במערך. ראו ערך בדוגמא הבאה:

```
const cards = [
{ country: 'israel', click: this.click },
{ country: 'france', click: this.click },
{ country: 'spain', click: this.click },
{ country: 'mexico', click: this.click }
];
```

שוב - מצב זה מייצר לנו 4 אובייקטים עם key בשם הpropוvalue עם הערך הרלוונטי ניצור מערך זה בתוך הrender שלנו, אבל לפני הreturn. כל שנותר עבורינו כעת יהיה לקרוא לפונקציית map, הקיימת גם בvanilla js, המדמה לנו ריצה כמו מעין לולאת for על כל המערך שלנו. כמו פונקציות רבות המשתייכות למערך (unshift, pop, remove ועוד) כך map זוהי פונקציה שרצה על כל המערך וממתינה לקבל פונקציית callback שאותה היא תממש. פונקציית ה-callback בהקשר

זה, מחזירה את המידע שבו אנחנו מעוניינים, ומוסיפה אותו למערך שנוצר בתהליך. בואו נראה דוגמא ובכך נבין טוב יותר –

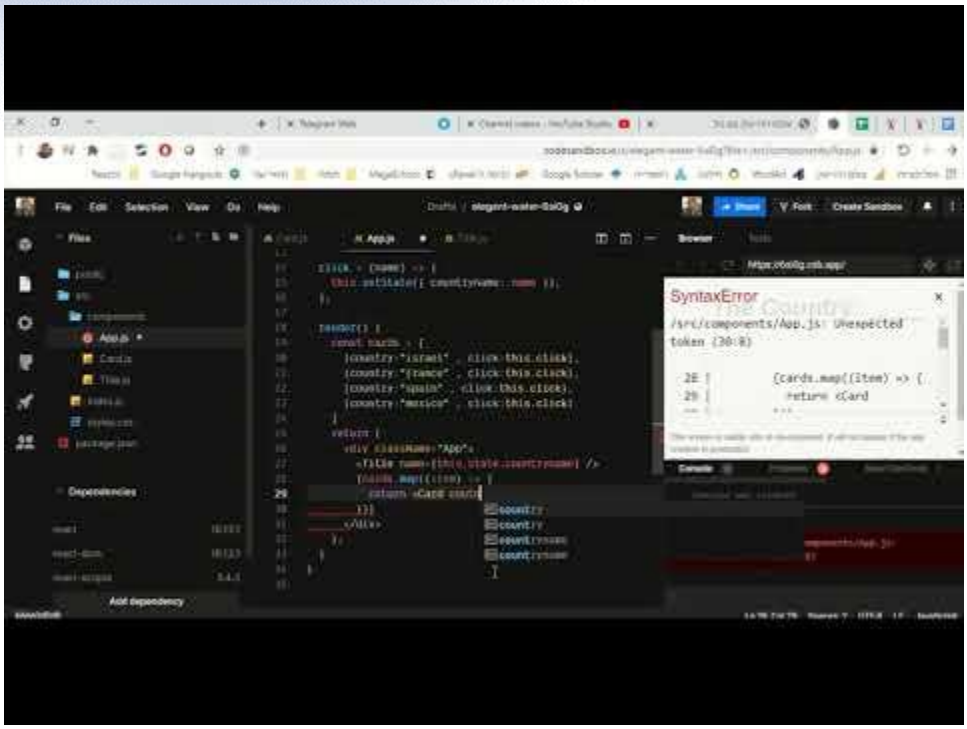
```
let cards_map = cards.map(function(item, index) {
return <Card country={item.country} click={this.click} />;
});
```

חידוד: item - הוא הפריט של המערך באיטרציה הנוכחית. מקביל ל i בלולאת for. הפרמטר הנוסף של index, אשר מחזיר לנו את המספר הסודר, לעיתים לא יהיה רלוונטי וניתן לוותר עליו... אנו משתמשים פה בarrow function על מנת לקבל את אותה התוצאה בצורה קומפקטית (ללא function וללא יצירת משתנה משום שזו פונקציה אנונימית).

```
{cards.map((item) => {
return <Card country={item.country} click={this.click} />;
})}
```

לסיום, חידה

ניתן יהיה לוותר גם על `click={this.click}` חישבו למה... תשובה בסרטון בעמוד הבא.



Routing

לפניכם אחד הנושאים המעניינים ביותר בעולם הריאקט אשר מציג לפנינו אתר ריאקטיבי עם דף אחד אשר ניתן לעבור בין מסכים שונים שבתוכו, מבלי באמת להפנות לקריאות שונות בדפדפן שלנו. כלומר ניתוב לכתובת אחרת וטעינת הדפים. בוורדפרס ובאתרים 'קלאסיים', הטעינה היא טעינה של כל הדף. באפליקצית ריאקט מדובר בניתוב שאינו דורש טעינה מחדש.

הדבר הראשון שנצטרך לבצע הוא ייבוא של ספריית הroutingמתוך התקנת הnpm –

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

כעת, נוכל ליצור ייבוא בחלק העליון של הקומפוננטה הראשית שלנו.

למשל, בתוך App.js ניצור

```
import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-dom'
```

וכעת נסביר על האובייקטים שייבאנו מתוך הreact-router-dom שלנו.

BrowserRouter

האובייקט הראשי שמכיל את כל הקריאות השונות לחלקי הראוטינג בדף שלנו.

Router

עלינו לעטוף את כל האזור הרלוונטי, בו נרצה לבצע את התחלופות והקריאה לקומפוננטות השונות. למעשה Router, פותח - בפנים כל האובייקטים הבאים שאסביר עליהם מיד ולסיום כאמור Router סוגר.

Switch

מדובר בתגית פותחת וסוגרת בתוך ה domשלנו ומה שיהיה בה, יוכל לבצע switch(כשמה כן היא) ולבצע תחלופות של הצגה והסתרה בין הקומפוננטות שיוצגו בתוכה.

Route

אולם, לא נוכל לכתוב רק את השם של הקומפוננטה בתוך כל Switch, אלא נצטרך לייצר תגית של Route פותח ו Routeסוגר. רק בתוכו נוכל לייצר את הקומפוננטה שלנו. לכל Route יש props של path אשר מכיל את שם הקומפוננטה. וכעת לחלק האחרון...

Link

אז אם נשארתם עד פה - אתם מבינים שנצטרך לבצע קריאה בלחיצה כלשהי כדי להפעיל ולכבות את ה routingשלנו שנמצאים בתוך ה ...switchכבר לא נבצע זאת על ידי תגית a עם תכונה של href, אלא על ידי ...Link בדיוק כמו בדוגמא הבאה אשר מציגה תפריט עם לחיצות ומעבר בין קומפוננטות שונות מבלי לשנות דבר בטעינת דף אחר!

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

class MainApp extends React.Component {
render() {
return
<div>
<nav>
<ul>
<li><Link to='/'>Home</Link></li>
<li><Link to='/about'>About</Link></li>
</ul>
</nav>
<h1>This is my App
<main>
<Switch>
<li><Route exact path='/'><Home /></Route>
<li> <Route path='/about'><About /></Route>
</Switch>
</main>
</div>
}
}

export default MainApp;
```

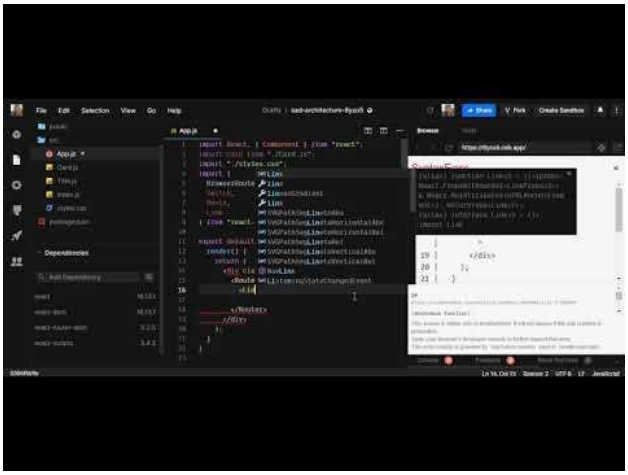
בונוס

בואו נדבר על מצב בו אנחנו רוצים להעביר פרמטרים (params) דרך הurl שלנו בעת יצירת routeו או בעת הלחיצה על הlink שלנו. הכוונה פה היא שנרצה ליצור מצב שבו מספר מסוים למשל מופיע למעלה (כמו המספר של המדריך הזה... נסו להחליף אותו ולראות מה קורה. כנל לגבי שם המדריך הזה) אז מה עלינו לעשות? עלינו לכתוב בתוך הpath של routeה שלנו הפעם –

```
path='country='/israel/:number'
```

ולמעשה בlink יהיה עלינו להמשיך ולכתוב באותה השיטה `to='/israel/3'` .

בשיטה זו, נוכל לשלוח פרמטרים כאשר בroute הם מוזנים כשם גנרי key ובlink הם כבר ממש valueה עצמו. והאתגר האחרון לגבי שליחת params דרך routing - על מנת לבצע בעת לחיצה על כפתור הcounter שיצרנו בסרטון ושישלח שינוי של הparams שקבענו, נוכל לשנות על ידי הקריאה הבאה – `this.props.history.push('/israel/' + counter);`



העלאת אתר ריאקטיבי לשרת הענן וירטואלי

אנו נמצאים כעת בלב הקורס, אחרי שאנחנו מכירים את מנגנון הstate שדרכו אנו מעדכנים כל דבר שיש לנו בדף (ממשתנים, בדיקות ופונקציות ועד לעדכון של טקסט בתוך input למשל או שמירה של נתונים כלשהם) וכן גם למדנו על הזרקת נתונים דרך props (התכונות מהhtml הקלאסי שהופכות כעת לאביזר משמעותי שדרכו ניתן לקבל מידע ולהזריקו חזרה לקומפוננטה אחרת).

כעת, נרצה ללמוד כיצד להעלות את האתר הריאקטיבי שיצרנו לשרת כלשהו. לא אתן לכם להמתין עד סוף המדריך מבלי להבין איך הממשקיות של ריאקט עובדת מול שרתים.

הפעם לא מדובר כאן בשרת רגיל המאחסן data כמו תמונות, טקסטים וכן גם קבצים כמו html וcss. ריאקט מופעל באמצעות שרת המעבד ומקפנג את כל קטעי הקוד השונים וממיר אותם לכדי js ונילה בסופו של דבר. כך גם לגבי scss או כל מיני ספריות כאלו ואחרות. עלינו יהיה לייצר פה קריאה בvisual studio שלנו ובקריאה זו לגרום להמרת הקוד לכדי תקייה אחת המכילה את כל הקבצים השונים. את אותה התקייה שנקבל נוכל לעטוף בקובץ zip ולהעלות לשרת שלנו. הפעם יהיה מדובר בשרת מסוג ענן וירטואלי.

על מנת לקבל את הקבצים עליכם להזין את שורת הקריאה הבאה בnpm שלנו –

```
npm run-script build
```

לאחריו, גשו לתקיית הקבצים של הפרויקט שלכם וראו בעצמיכם את התקייה החדשה שנוצרה. כל שעליכם יהיה לעשות הוא להמיר את התקייה לכדי קובץ zip ולאחר מכן להעלות אותה לשרת הוירטואלי שרכשתם. בשרתים וירטואלים אנו נתקלים בתופעה אחרת מאשר שרת אחסון רגיל והיא שהתשלום מתבצע כמעין קרדיטים על כל 'התאמצות' של הענן הוירטואלי. ההתאמצות מורכבת מכדי כמות הגולשים שנכנסים, הנפח, הכובד שמופעל על הענן, כמות השינויים שאתם מבצעים ועוד ועוד... שימו לב לכך, מדובר בשרת וירטואלי הפעם אשר מכיל בתוכו את כל ההמרות השונות ואת הרצות הnodejs הרלוונטיות לפרויקטים שלנו.