svcollege
ללמוד. לדעת. לעבוד.

# React

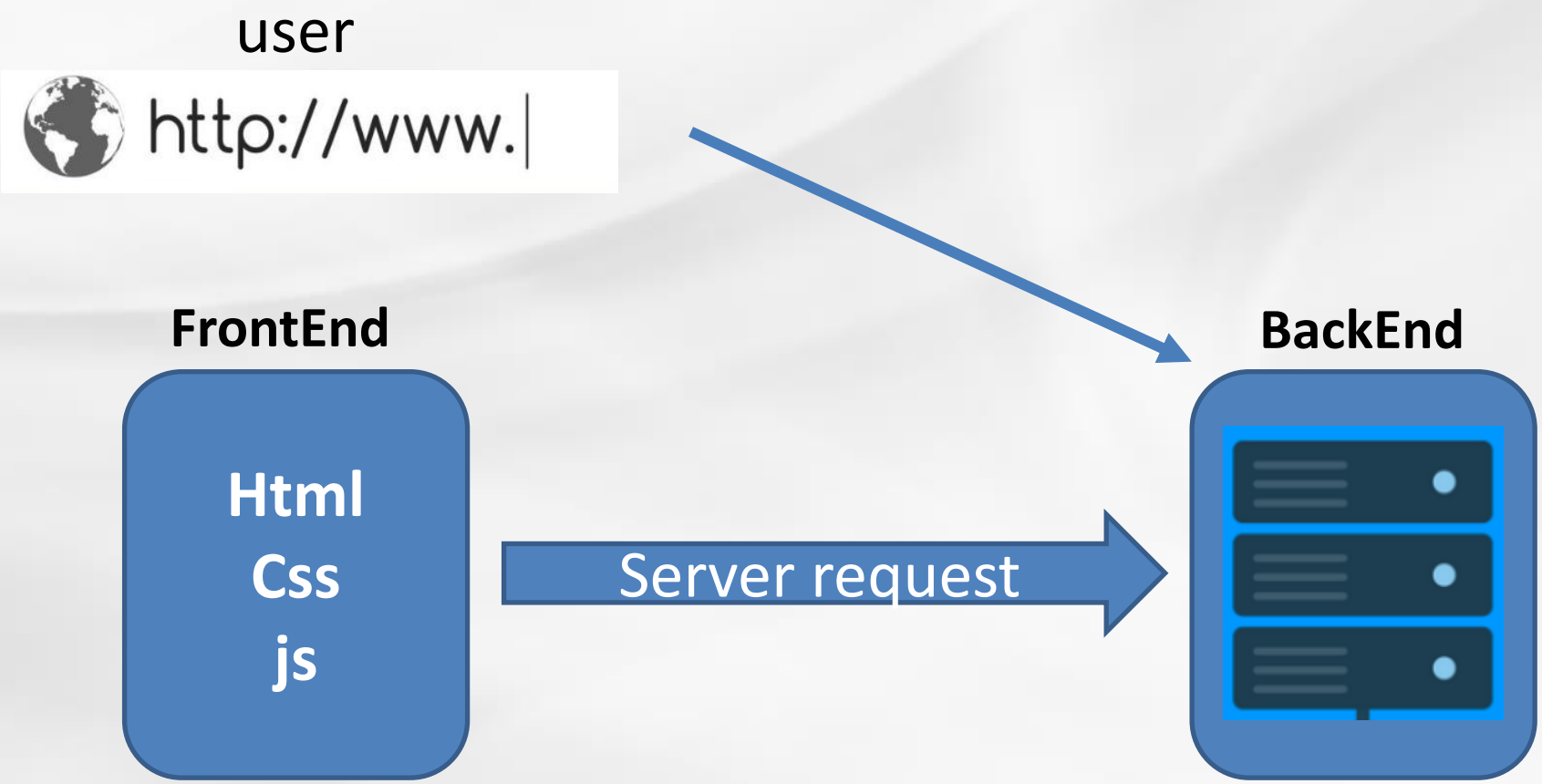
svcollege
ללמוד. לדעת. לעבוד.

# React? What is it?

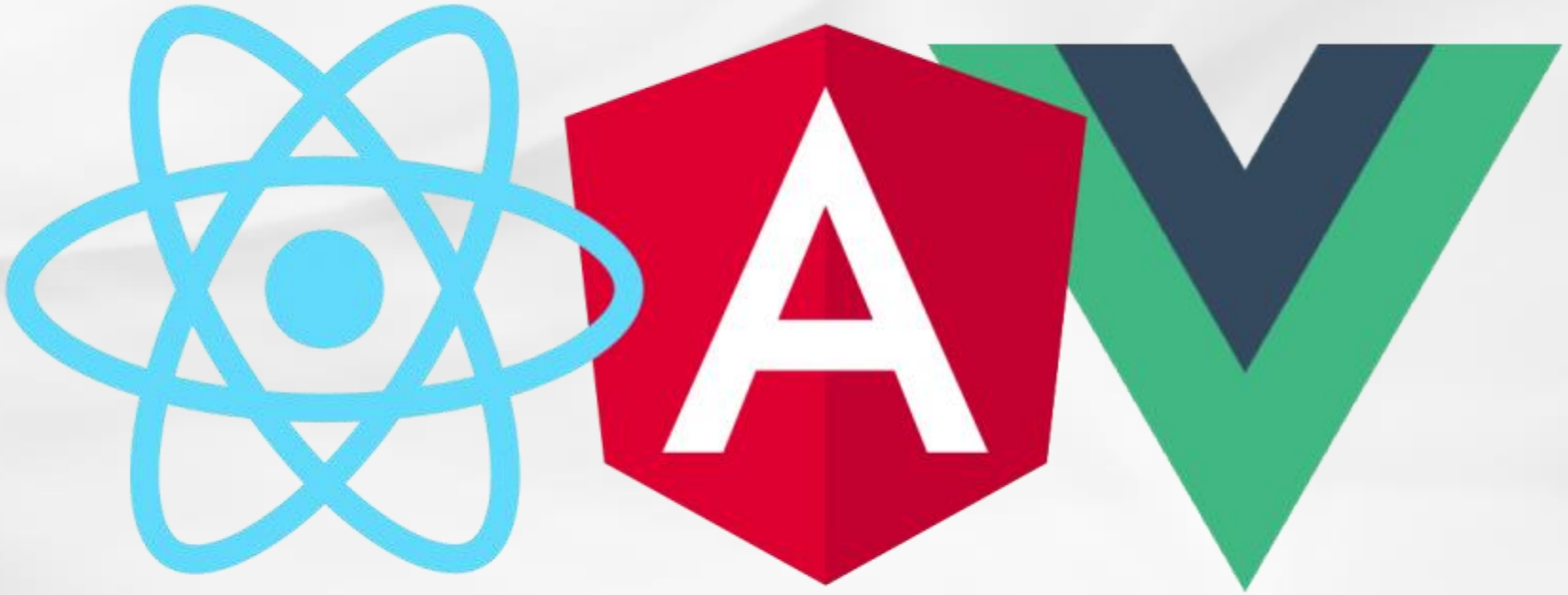
- ספריית צד לקוח של js מהנפוצות בעולם.
- נוסדה ע"י חברת Facebook בשנת 2013.
- נועדה לבנייה דינאמית של ממשק המשתמש בעזרת עיבוד "רכיבים" נפרדים.
- ספריית קוד פתוח בשפת js המשמשת לפיתוח ממשק משתמש (טפסים, גלריות תמונות...כל מה שהמשתמש רואה ב web).

svcollege
ללמוד. לדעת. לעבוד.

# הסבר...



svcollege
ללמוד. לדעת. לעבוד.



svcollege
ללמוד. לדעת. לעבוד.

# למה React?

1. קלות בפיתוח אפליקציות.
2. משמשת לאתרים דינאמיים.
3. מבוססת js.
4. נתמך בכל הדפדפנים (ללא הידור/פרשון)

- React Native

svcollege
ללמוד. לדעת. לעבוד.

# Install

1. הורדת node.JS ← NPM (cmd node –v)
2. הורדת git (cmd git).
3. תוספת לדפדפן: React developer tools
4. תוספות לסביבת VSC :
   - ES7 React/Redux…
   - Prettier

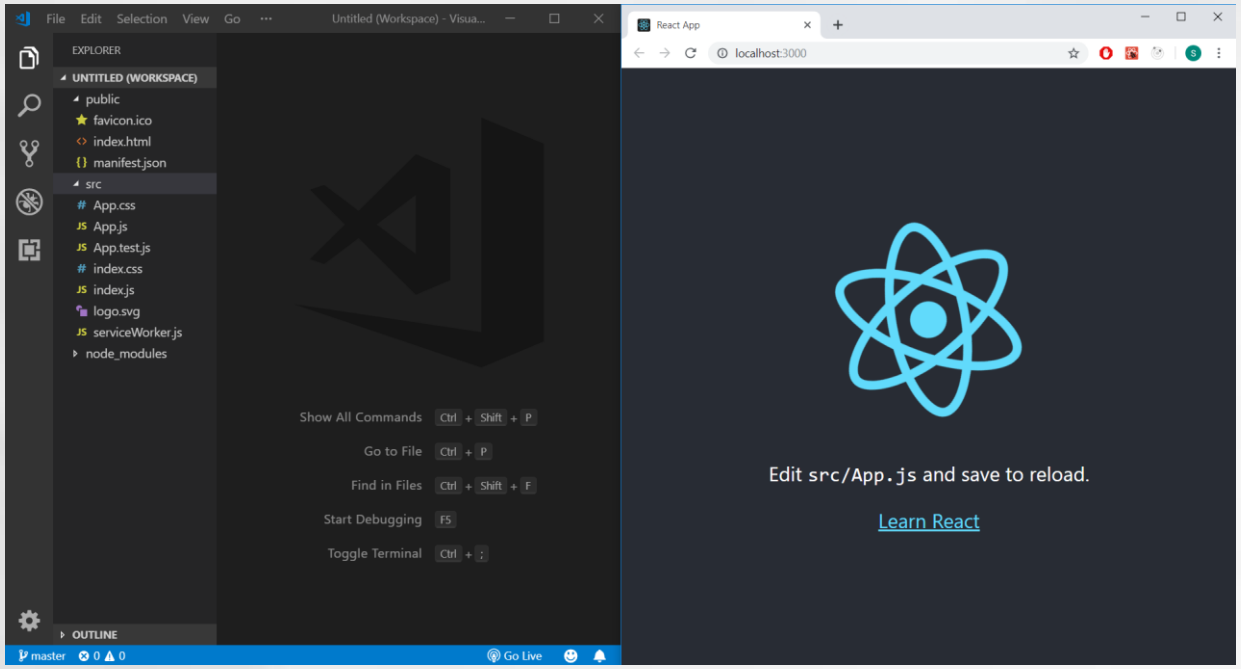
CMD (or Terminal) ← npm install -g create-react-app

5. פותחים תיקייה חדשה ב VSC
6. כניסה ל Terminal (view)
   - create-react-app appname (שם אפליקציה באותיות קטנות בלבד)
   - cd appname
   - npm start

svcollege
ללמוד. לדעת. לעבוד.

# Create your first app

1. יצירת תיקייה ← להגיע לנתיב שלה בעזרת CLI ← כתיבת הפקודה:
   npx create-react-app myapp
2. פתיחת הקבצים בסביבת הפיתוח.
3. יצירת סביבת פיתוח ← npm start



# Files in React

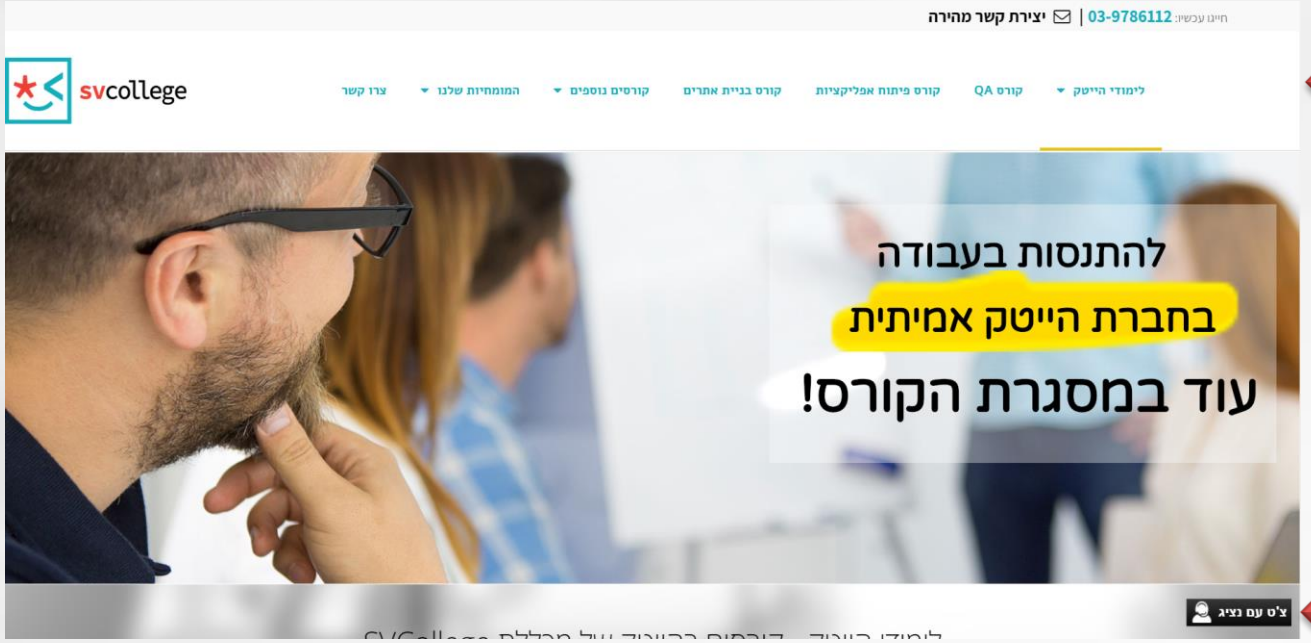
svcollege
ללמוד. לדעת. לעבוד.

1. **index.html:** מסמך html ראשי, דרכו מועבר ה root.
2. **Index.js:** מגדיר את ה root הראשי.
3. **App.js:** הקומפוננטה הראשית של האפליקציה.
   (אחראית לקריאת כל הקומפוננטות האחרות).
4. **App.css:** אחראית לעיצוב הקומפוננטה.

# Components

רכיב – הינו חלק מהאתר/אפליקציה אשר בנוי מ html/css/js.
כל אתר בנוי מכמה רכיבים שונים אשר מקבלים ומעבירים מידע מאחד לשני ומאפשרים דינמיות רבה ביצירת וחלוקת האתר.

Header component




Body component

Contact component

svcollege
ללמוד. לדעת. לעבוד.

# App.js

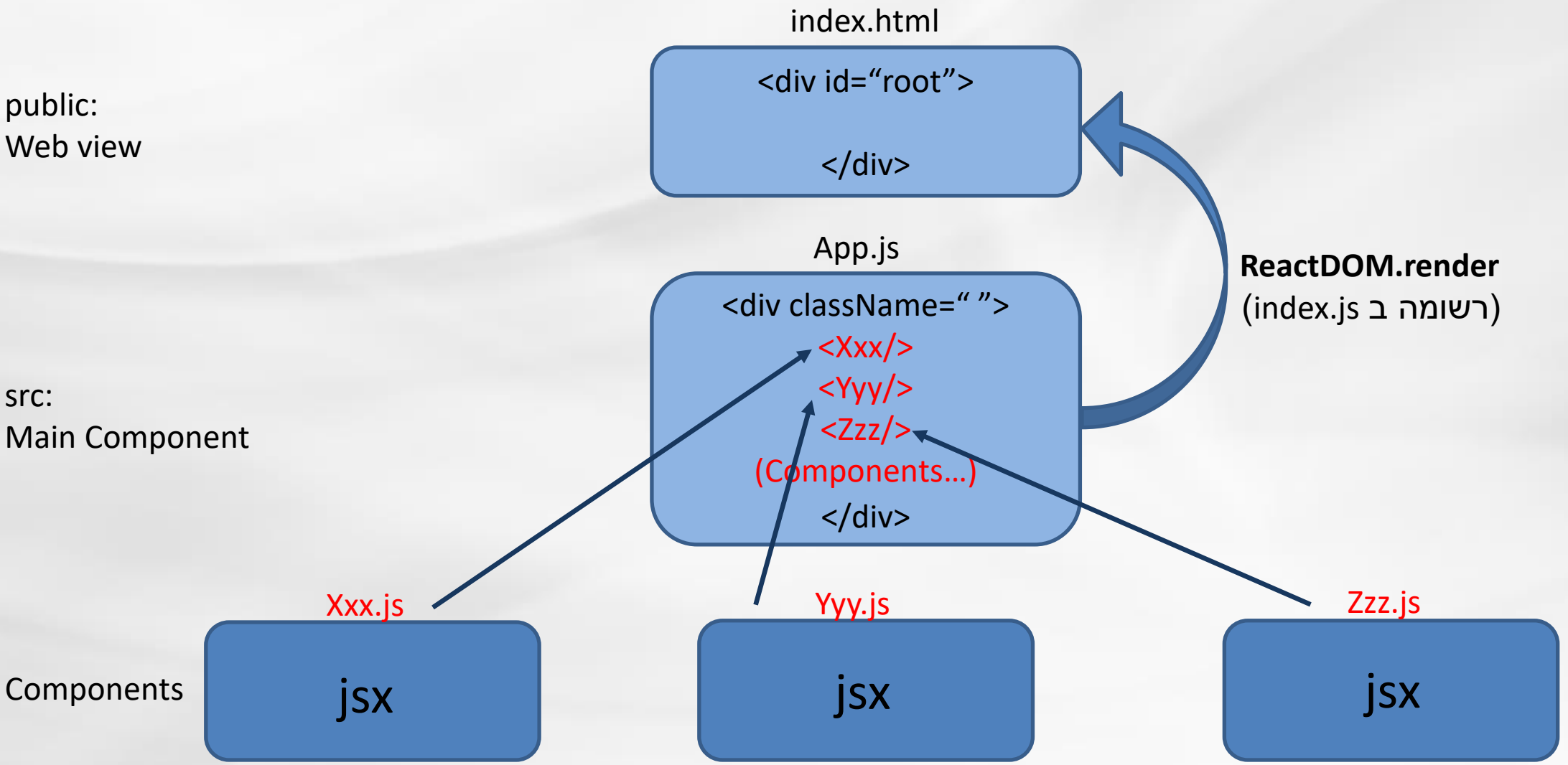
## רכיב ראשי – אחראי לקריאה של רכיבים נוספים.

```
import React from 'react’; //ייבוא ספריית ריאקט

import './App.css'; //ייבוא קובץ העיצוב

function App() { //פונקציה ראשית
        return (// חייבת להחזיר אלמנט אחד בלבד
        <div className="App">

        //כל הרכיבים יקראו כאן

        </div>
        );
}
export default App; //מאפשר קריאה לרכיב זה
```

RFC
React Function Component

קומפוננטה אשר מחזירה פונקציות בלבד.
כל פונקציה תחזיר אלמנט אחד בלבד.

# תרשים זרימה


index.html

public:
Web view
App.js

<Xxx/>
<Yyy/>
<Zzz/>
(Components...)

ReactDOM.render
(רשומה ב index.js)
src:
Main Component
Xxx.js
Yyy.js
Zzz.js
Components
jsx
jsx
jsx

svcollege
ללמוד. לדעת. לעבוד.

# JSX
# Java Script XML

בריאקט ניתן לשלב בתוך הקוד קטעי HTML.

- כל אלמנט ללא תגית סוגרת ייסגר עם /
  זה ייראה כך: <element/>
- ביטויים ב JS ייכתבו בתוך { }.
- כל פונקציה יכולה להחזיר אלמנט אחד בלבד.
- המילה Class נקראת כאן ClassName.

# Create new component

1. יצירת תיקייה לקומפוננטות (לא חובה אך מומלץ).
2. יצירת קומפוננטה (רכיב) עם סיומת js בתוך התיקייה.
3. יש לכתוב את הקוד הבא בקובץ הקומפוננטה:

```
import React from 'react'

export default function Person() {
   return (
        <div>
              קוד הקומפוננטה יירשם כאן
        </div>
  )
}
```

4. יש לייבא את הקומפוננטה הרצויה מ App:

```
import ComponentName from './Components/Person.js';
```

5. יש להציג את הקומפוננטה ב App (נרשום בתוך ה div של App <Person/> )

**ניתן להשתמש בפקודה rfc ליצירת התבנית

# Props

## שליחת ערכים לקומפוננטה

Props יודע לקבל ערכים (בהתאם למאפיינים) ולהחזיר אותם ל App.
בדוגמא זו הקומפוננטה Person יודעת לקבל שם וגיל ולהציג אותם דרך ה App.
**ניתן להציג קומפוננטה כמה פעמים שנרצה.

**Function Component**

```
import React from 'react'

export default function Person(props) {
   return (
        <div>
                <p> {props.name} </p>
                <p> {props.age} </p>
        </div>
  )
}
```

**App**

```
function App() {
  return (
        <div className="App">
              < Person name={'dor'} age={32} />
        </div>
  );
}
```

svcollege
ללמוד. לדעת. לעבוד.

# Props

## שליחת ערכים לקומפוננטה

ניצור מערך של אובייקטים כאשר כל אחד מהם יקבל name ו age
נשלח כל אובייקט במערך דרך App לפונקציית הקומפוננטה שיצרנו (Person).

JS App.js × JS Post.js

appp > src > JS App.js > ...

```
1  import React from 'react'
2  import './App.css';
3  import Post from './Components/Post';
4
5
6  export default function App() {
7
8    const list = [{name:'dor', age:'33'}, {name:'max', age:'64'}, {name:'shem', age:'30'}];
9
10   return (
11     <div className="App">
12       <Post name={list[0].name} age={list[0].age} />
13       <Post name={list[1].name} age={list[1].age} />
14       <Post name={list[2].name} age={list[2].age} />
15     </div>
16   )
17 }
```

# Add CSS

הוספת CSS לכל רכיב (קומפוננטה).

1. **הוספת style לאלמנט ספציפי**

<element style={{ color:'red', fontSize:'50px' }}

**שימו לב לשינוי של {{ }}, ו camelCase.

2. **יצירת קובץ css.**

הוספת הקובץ ע"י import './path';

3. **הוספת BOOTSTRAP**

א. ב Terminal: npm install bootstrap

ב. import 'bootstrap/dist/css/bootstrap.min.css'

# Style JSX

svcollege
ללמוד. לדעת. לעבוד.

```
<element style={{attribute:'value' , attribute:'value'}}
```

Bootstrap

1. CLI → npm install bootstrap
2. App.js → import 'bootstrap/dist/css/bootstrap.min.css';

# Map

לא תמיד נדע מראש כמה פעמים נרצה להציג את הקומפוננטה.
למשל בפייסבוק, הפוסטים יוצגו בהתאם לגלילת הדף כלפי מטה.
נרצה לדאוג שעם כל הוספה/הסרה של פוסטים למערך הם יוצגו בהתאם באופן אוטומטי.
לשם כך נשתמש בלולאת Map **שרצה בהכרח על כל המערך** ויכולה להחזיר ערך.

במקום לרשום:

```
<Post name={list[0].name} info={list[0].info}/>
<Post name={list[1].name} info={list[1].info}/>
<Post name={list[2].name} info={list[2].info}/>
```

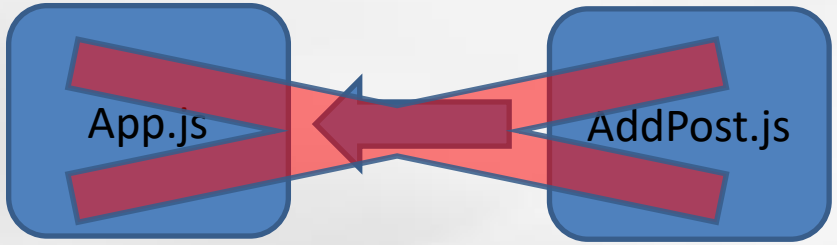
נרשום:

```
{list.map((element) => {
    return <Post name={element.name} info={element.info} />
})}
```

# Add Post

מטרה:
הוספת פוסט חדש (קומפוננטת AddPost), כאשר נלחץ על כפתור Add הפוסט החדש יתווסף למערך הפוסטים בApp (מהשרת) ויוצג (באמצעות לולאת map כפי שכבר ראינו).

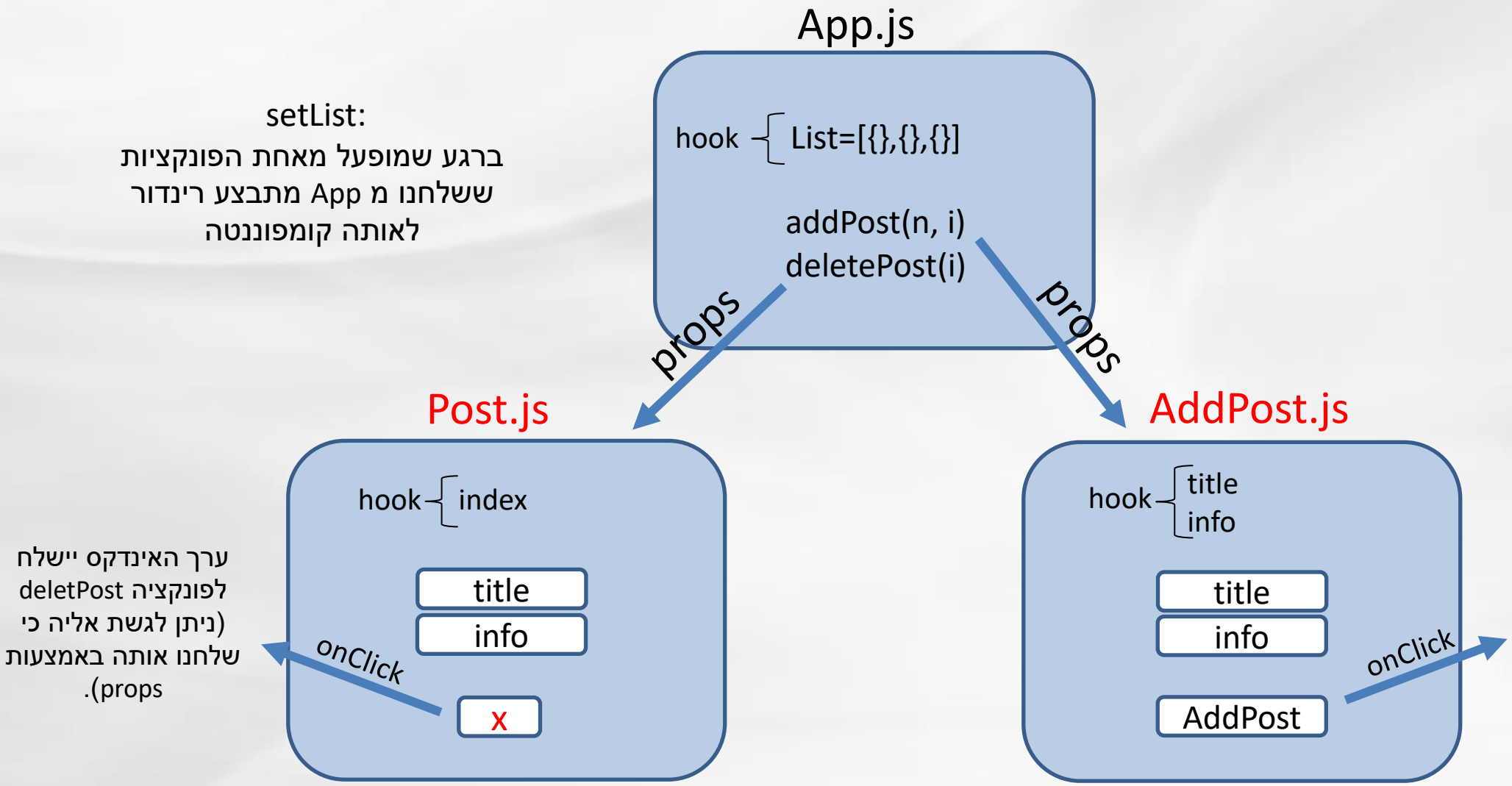
בעיה:
לא ניתן להעביר ערכים לקומפוננטה הראשית (מ AddPost אל App) אלא רק מ App אל AddPost.




פתרון:
ניצור פונקציה בקומפוננטה הראשית ונשלח אותה לקומפוננטה AddPost בעזרת props.

בואו נראה איך זה עובד...

# אז מה בעצם ראינו כאן?



# Hooks

Hooks (התווסף מגרסה 16.8 של react) מאפשר שימוש ב state ובפיצ'רים נוספים של מחלקות, ללא שימוש בקומפוננטת מחלקה.

<u>ייבוא useState לשימוש ב Hooks:</u>

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

**הערך הדיפולטיבי שאנו מייבאים הוא react, את השאר יש לציין (useState).

<u>יצירת hook:</u>

**const [state, setState] = useState(initialState);**

(מילה שמורה)

שם ה state

הפונקציה שאחראית לעדכון ה state (במקום setState)

איתחול ה state

יירשם ב rfc במיקום שבו היינו רושמים את ה state במחלקה

# Hooks

svcollege
ללמוד. לדעת. לעבוד.

דוגמא – יצרנו כפתור שכל פעם שלוחצים עליו הוא משנה את ה state.

App.js

app4 > src > App.js > ...

```
1  import React,{useState} from 'react';
2  import './App.css';
3
4  function App() {
5
6  const [flag, setflag] = useState(false);
7
8    return (
9      <div className="App">
10       <button onClick={()=>{setflag(!flag)}}>Change Flag</button><br/>
11       {flag.toString()}
12
13     </div>
14   );
15 }
16
17 export default App;
18
```

# Routers

Routers מאפשרים לנו לשנות את כתובת ה URL בהתאם לעמוד שבו אנו נמצאים.

כאשר עוברים בין עמודים באתר מתבצע מעבר בין ערוצים.
ניתן לראות שכתובת ה URL משתנה בהתאם לערוצים.

<u>דוגמא</u>

הערוץ הראשי של המכללה:

https://svcollege.co.il

ערוץ הקורסים של המכללה (לחיצה על "כל הקורסים"):

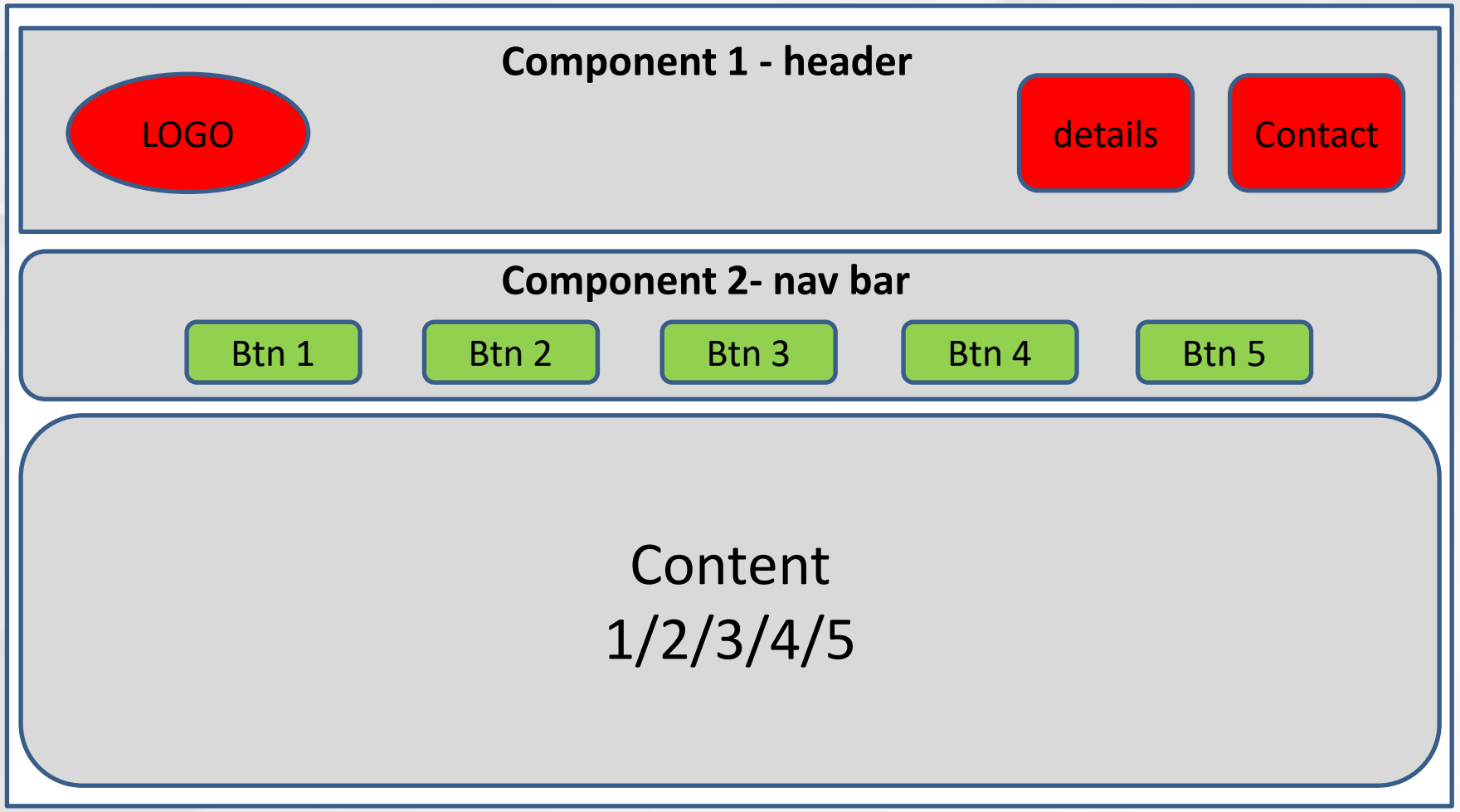
https://svcollege.co.il/courses/

ערוץ קורס פיתוח:

https://svcollege.co.il/courses/web-development/

# Routers

## (תכנון אתרים נפוץ)




לא קשור
לשינוי הערוצים

קשור לשינוי הערוצים
אבל לא מתחלף

קשור לשינוי הערוצים
וגם מתחלף בעצמו

# Routers

svcollege
ללמוד. לדעת. לעבוד.

כדי לעבוד עם ערוצים יש להוסיף את ה module למערכת שלנו (ב- CLI):

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

בקומפוננטה App:

```
import {BrowserRouter as Router, Routes , Route} from 'react-router-dom';
```

# Routers

<Router/> – בתוכו יהיו הרכיבים הקשורים לערוץ (מתחלפים או לא מתחלפים)
<Routes/> – בתוכו יהיו הרכיבים הקשורים לערוץ שאמורים להתחלף בפועל.
<Route/> – הגדרת ערוץ עבור קומפוננטה ספציפית.

```
<h1>not related to the Routers</h1>

<Router>
    <Routes>
        <Route path='/routhPath’  element={<ComponentName/> } />
    </Routes>
</Router>
```

**במידה ונגדיר: path='/' הערוץ יהיה דף הנחיתה (הראשי)

# Routers

```
import {BrowserRouter as Router, Routes ,Route} from 'react-router-dom'
import First from './components/First'
import Second from './components/Second'
import HomePage from './components/HomePage'

function App() {

  let x = 10;

  return (
    <div className="App">
      <h1>Title</h1>
      <Router>
        <Routes>
          <Route  path='/' element={ <HomePage/>}/>
          <Route  path='/first' element={ <First val = {x}/>}/>
          <Route  path='/second' element={ <Second name='orgad'/>}/>
        </Routes>
      </Router>
    </div>
```

המשך בשקף הבא...

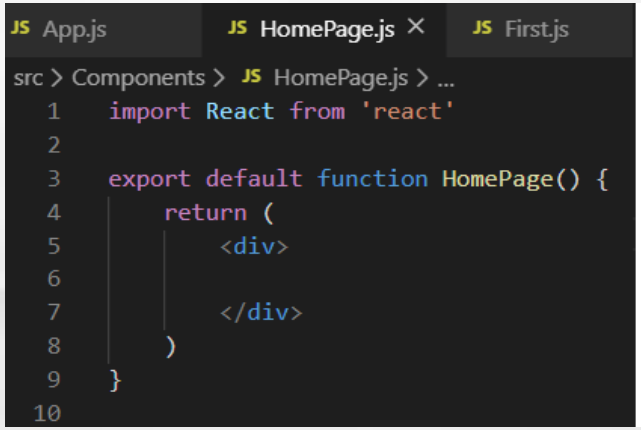
ה Title תמיד יופיע בחלקו העליון (לא קשור לשינוי הערוצים).

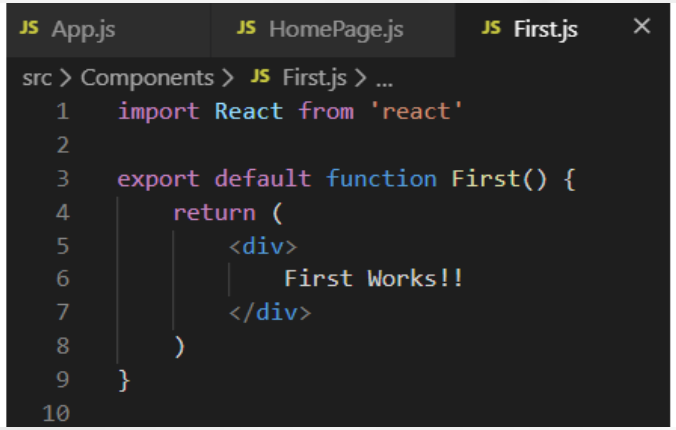
בעמוד http://localhost:3000 תתבצע קריאה לקומפוננטה HomePage (לא יוצג כלום).

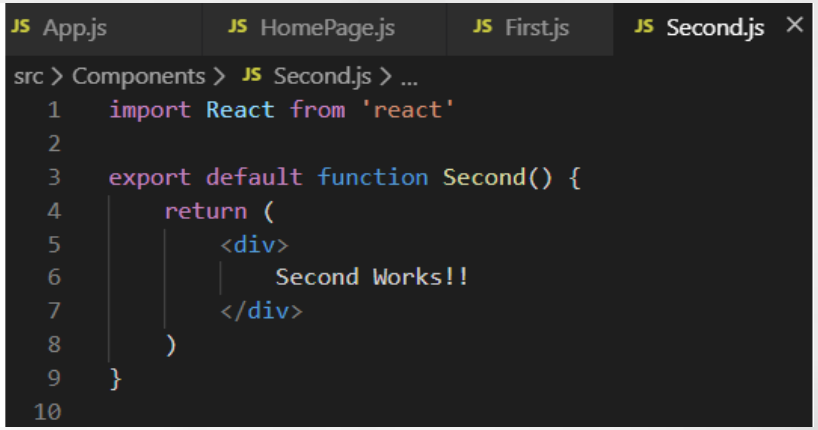
בעמוד http://localhost:3000/First תתבצע קריאה לקומפוננטה First (יוצג First Works!!).

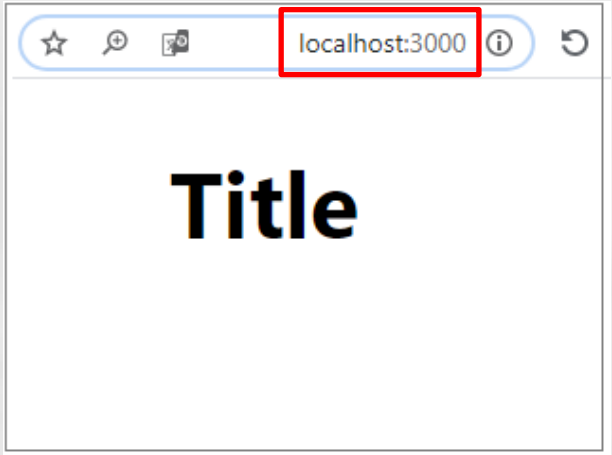
בעמוד http://localhost:3000/Second תתבצע קריאה לקומפוננטה Second (יוצג Second Works!!).

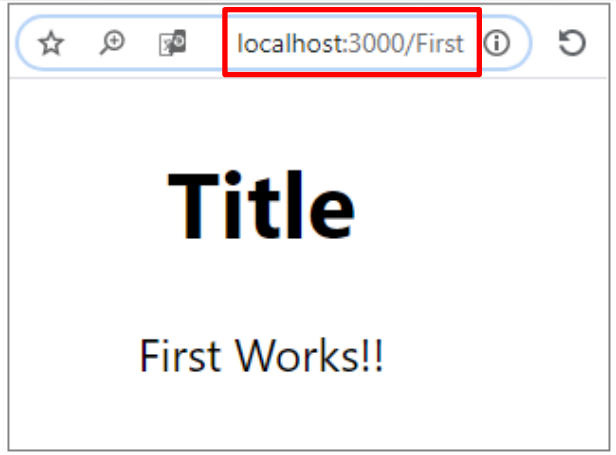
# Routers

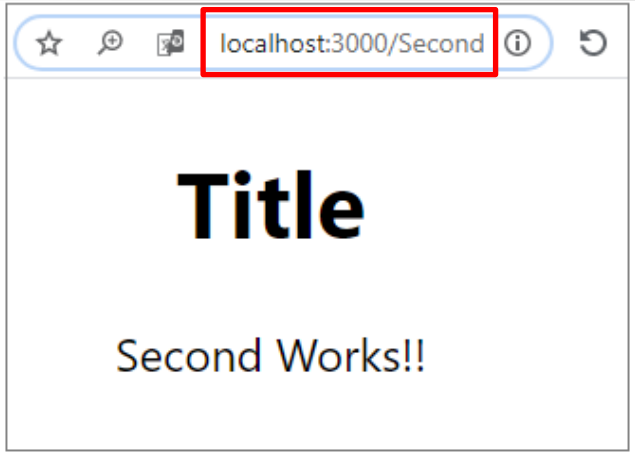



כעת, נרצה ליצור תפריט כדי לבצע את החלפת הערוצים ע"י כפתורי ניווט ולא ידנית...

# Routers - Link

ניצור קומפוננטה Menu שבה יהיו כפתורי הניווט.
נרצה שכל כפתור יעביר אותנו לערוץ הרלוונטי ולכן נוסיף ב Menu את הפקודה:

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

נרצה שבעת לחיצה על כפתורי הניווט יבוצע מעבר לערוץ הרצוי באתר.
נעשה זאת באמצעות הפקודה הבאה:

```
<Link to='/'> LinkName </Link>
```

לאיזה נתיב לבצע את ההחלפה

שם הלינק (הכפתור)

ה Menu קשור לערוצים אך אינו מתחלף, ולכן הוא יהיה בתוך <Router> ומחוץ ל <Switch>

# Routers - Link

JS App.js | JS Menu.js | JS HomePage.js | JS First.js | JS Secon

src > Components > JS Menu.js > ...

```
1   import React from 'react';
2   import { Link } from 'react-router-dom';
3
4   export default function Menu() {
5       return (
6           <div className='row' style={{border:'2px solid gray'}}>
7               <div className='col-4'>
8                   <Link to='/'>Home</Link>
9               </div>
10              <div className='col-4'>
11                  <Link to='/First'>First</Link>
12              </div>
13              <div className='col-4'>
14                  <Link to='/Second'>Second</Link>
15              </div>
16          </div>
17      )
18  }
19
```

ב Menu יצרנו לינק לכל ערוץ.
כאשר המשתמש ילחץ על אחד הלינקים (Home, First או Second) הערוץ ישתנה בהתאם וכך יחזיר את הקומפוננטה שהוגדרה בו והיא תוצג על המסך.

svcollege
ללמוד. לדעת. לעבוד.

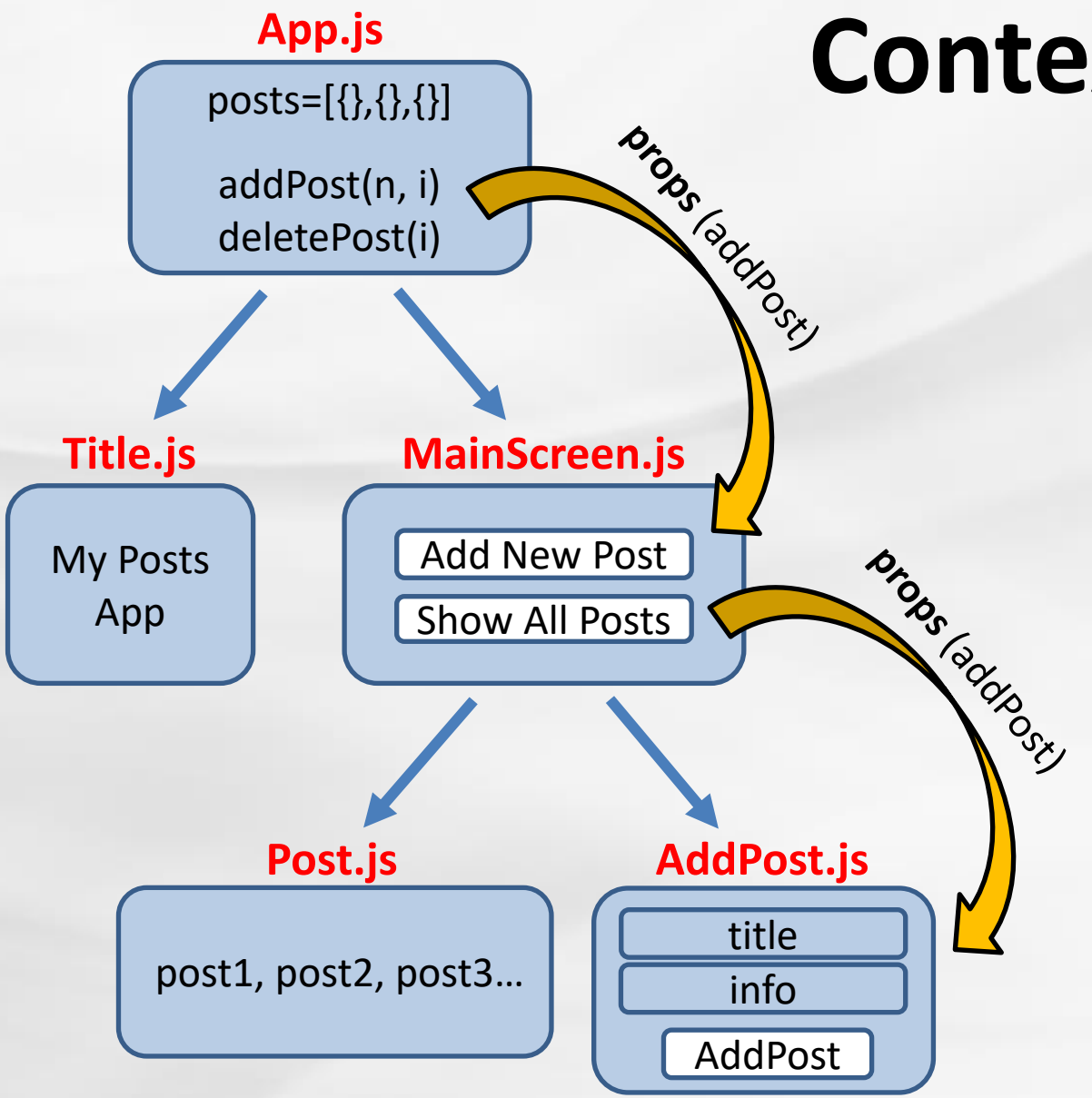
# אז מה בעצם ראינו כאן?

יצרנו nav bar שכל כפתור ניווט יפנה לערוץ המתאים באמצעות Link.

יצרנו ערוצים ב App שיודעים להחזיר קומפוננטות.

תכננו את מבנה האתר: מה לא קשור לערוצים, מה קשור ולא משתנה ומה כן משתנה.

# ContextAPI



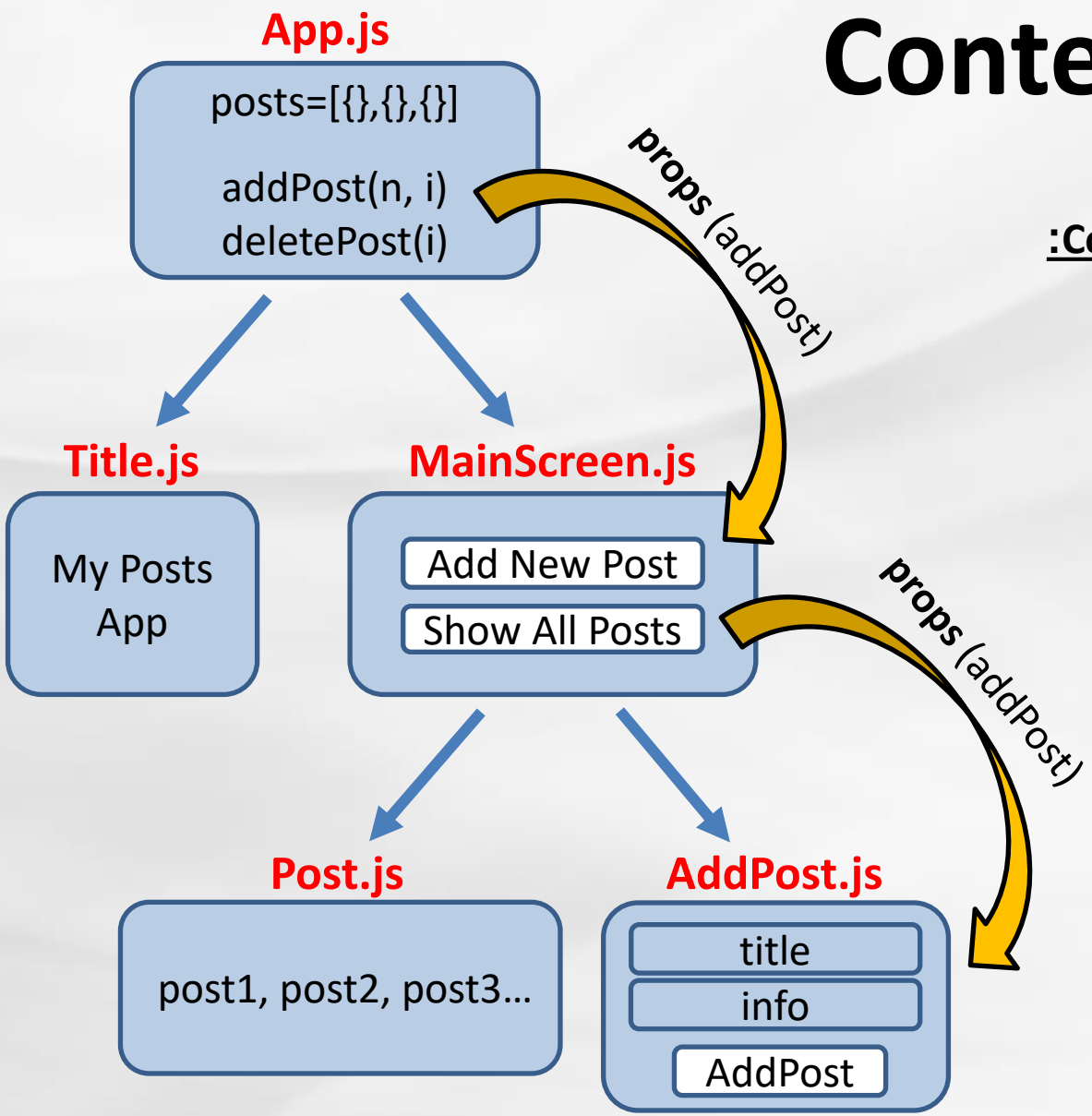

כאשר מעבירים props בין קומפוננטות הם יעברו גם דרך קומפוננטות ביניים שלעתים כלל לא ישתמשו במידע.

בדוגמא זו הפונקציה addPost נשלחת מ App ל AddPost באמצעות props ועוברת דרך MainScreen שלא באמת עושה בה שימוש.

באותו אופן זה קורה עם מערך הפוסטים והפונקציה deletePost שנשלחות מ App ל Post באמצעות (props.

נרצה להעביר מידע מקומפוננטה אחת לאחרת ללא מעבר דרך קומפוננטות ביניים.
נעשה זאת באמצעות ContextAPI.

# ContextAPI




**דוגמא להעברת props (addPost) לפני שימוש ב ContextAPI:**

**App:**

```
<MainScreen add={addPost} />
```

שליחת המאפיין add מ App ל MainScreen
(ערך המאפיין הוא הפונקציה addPost).

**MainScreen:**

```
<AddPost add={props.add}
```

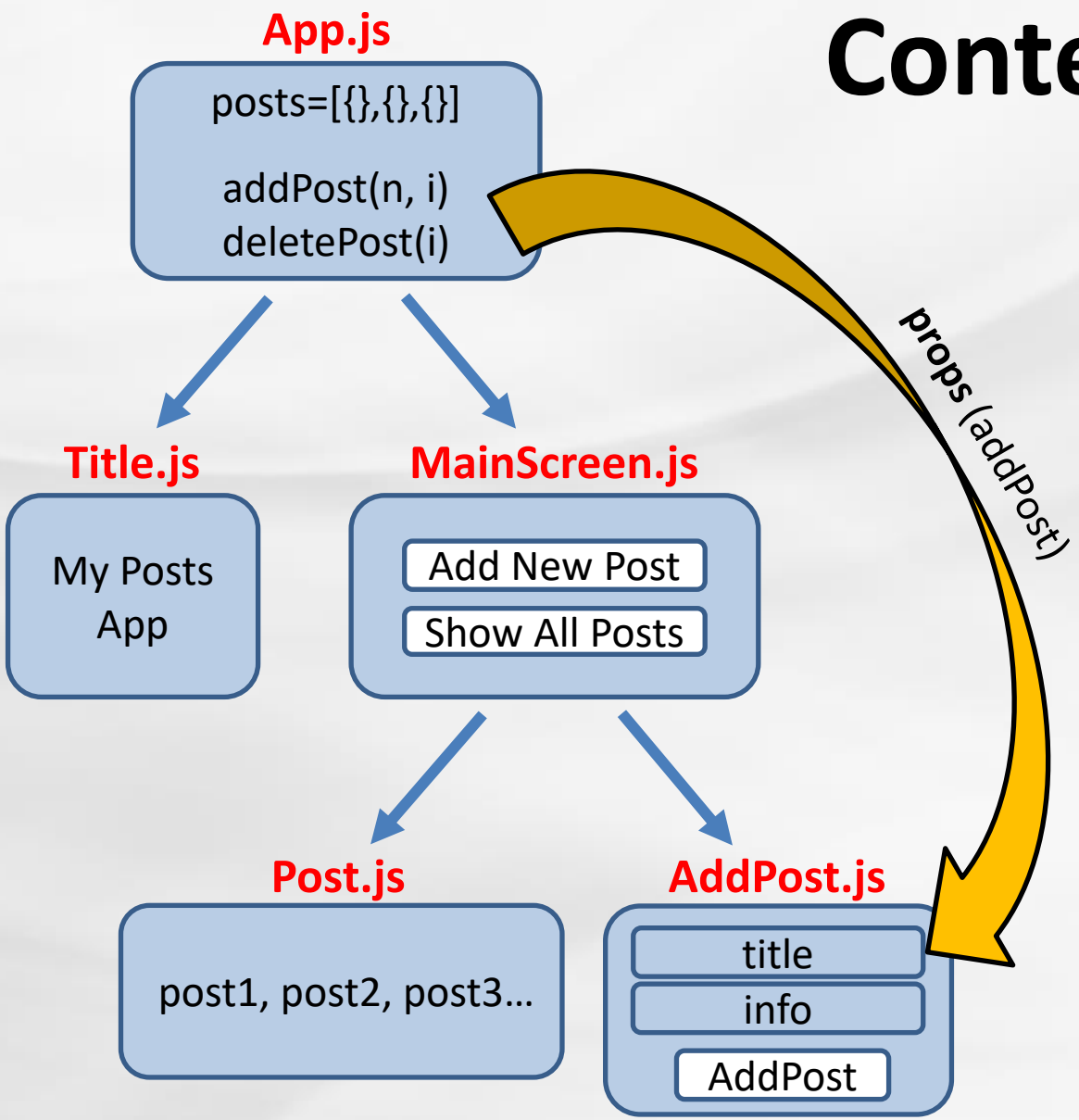
שליחת המאפיין Add מ MainScreen ל AddPost
(הערך הוא אותו add שנשלח באמצעות props מ App).

**AddPost:**

```
props.add(name, info);
```

שימוש במאפיין שנשלח מ App, שליחת ערכים לפונקציה.

# ContextAPI




באמצעות ContextAPI נוכל לדלג על השרשרת ולהעביר props באופן ישיר מקומפוננטה אחת לאחרת.

# ContextAPI

svcollege
ללמוד. לדעת. לעבוד.

ניצור קומפוננטה ContextAPI.js ונכתוב בה:

```
import React from 'react';

const context = React.createContext();
export const Provider = context.Provider;
export const Consumer = context.Consumer;
```

# ContextAPI

**נגדיר את App שיהיה ה Provider:**

```
import {Provider} from './ContextAPI'
```

נעטוף את MainScreen שנמצא ב App ב Provider (היות והוא מספק את כל ה props).

```
<Provider value={{add:addPost}}> // הגדרנו אובייקט לאפשר שליחה של כמה מאפיינים ביחד
    <MainScreen add={addPost} />
</Provider>
```

נמחק את המאפיין add ששלחנו, היות ודאגנו לשלוח אותו באמצעות Provider.

מהקומפוננטה MainScreen נמחק את המאפיין add היות והקומפוננטה אינה תחנת מעבר.

```
<AddPost add={props.add} />
```

**value הינה מילה שמורה.

# ContextAPI

**נגדיר את AddPost שיהיה ה Consumer:**

```
import {Consumer} from './ContextAPI'
```

נרצה לגשת מה Consumer למאפיינים שהעברנו מה Provider. ה Consumer הוא האבא אשר מעביר את המאפיינים לבן (כתבנו arrow function שמקבלת את המאפיינים באמצעות val). באמצעות val.propName נוכל לגשת למאפיינים אלה.

```
<Consumer>
    {
        (val)=>{
            return <button onClick={() => { val.add(name, info) }}>Add Post</button>
        }
    }
</Consumer>
```

# פרויקט בית חכם

svcollege
ללמוד. לדעת. לעבוד.

# Fetch

**משיכת מידע משרת**

https://jsonplaceholder.typicode.com/

**דוגמא לאתר המספק Json מובנה שמולו נרצה לעבוד.**
**באמצעות קריאות מול השרת נוכל לגשת ל Json ולהחזיר את הנתונים שנרצה.**

נרצה למשוך את ה Json, לשמור אותו ב Hook **ולאחר מכן** לרענן את הקומפוננטה.
האלגוריתם של ריאקט לא תמיד עובד בסדר שאנו כמתכנתים מצפים ולכן יצרו פונקציה שנקראת **useEffect** שתפעל לפני ריענון הקומפוננטה.

# Fetch

```
import React, { useEffect, useState } from 'react';

function App() {

  const [list, setList] = useState([]);

  // Everything will be done before the component is refreshed.
  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/users')
      // .then((result) => { console.log(result) })
      .then((result) => { return result.json() })
      .then((data) => { setList(data) })

  })
```

באמצעות fetch נשלח בקשת התחברות לשרת (request).

then היא פונקציה שמקבלת את התגובה מהשרת (response).

לאחר קבלת התגובה נרצה לשמור אותה במשתנה ולהציג אותה.

# Fetch

```
useEffect(() => {
  fetch('https://jsonplaceholder.typicode.com/users', {
    method: 'POST',
    headers: {
      'Content-type': 'application/json'
    },
    body: JSON.stringify({
      name: "Avi",
      username: "Cohen",
      email: "Avi123@gmail.com"
    })
  })
    .then((result) => { return result.json() })
    .then((data) => { setList(data) })
})
```

דוגמא לבקשה מסוג POST
באמצעותה ניתן להוסיף/לעדכן נתונים בשרת.

ה headers מגדיר את סוג הנתונים שנרצה לשלוח (json).

ה body מגדיר את הנתונים שנשלחו

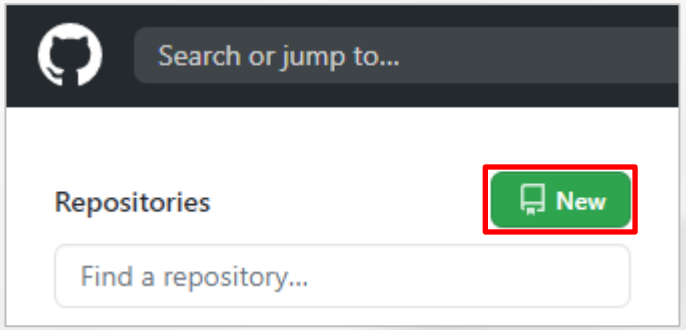
svcollege
ללמוד. לדעת. לעבוד.

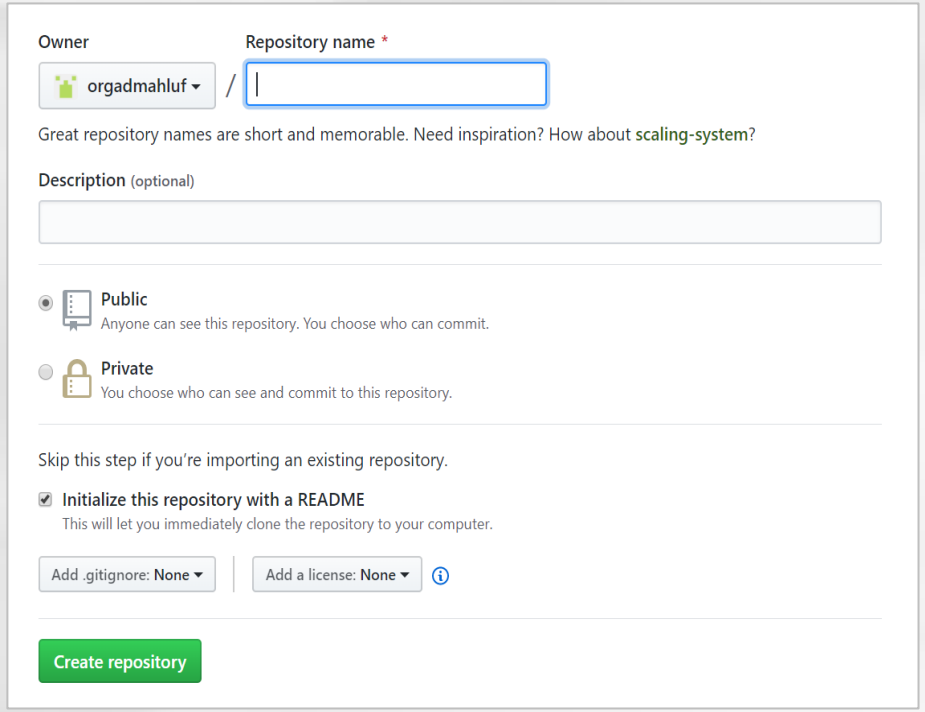
# GIT – מערכת לניהול בקרת תצורה

המערכת מהווה כלי לניהול גרסאות בחברה.

כל ענף מהווה Repository (מאגר מידע) מסוים, הרעיון הוא שכל מחלקה תוכל לעבוד באופן לוקאלי על הענף שלה, האחראי יוכל למזג את כל מה שהעובדים עשו באותו ענף ובסופו של דבר "לעדכן" את הגזע (Master) - זה בעצם הבילד הבא.

# GitHub




הרשמה ל GitHub:

- לזכור את שם המשתמש והסיסמא.
- יצירת repository (מאגר המידע באתר GitHub).

# GitHub

GitHub

יצירת מאגר גלובאלי:

יוצרים מאגר ב GitHub

clone

Desktop

יצירת מאגר לוקאלי:

יוצרים תיקייה על שולחן העבודה,
ב cmd בנתיב התיקייה מבצעים clone לקישור של המאגר בגלובלי

# GitHub

## סדר פעולות – העלאת קבצים

העבודה באמצעות ה cmd תתבצע בנתיב הלוקאלי של התיקיה שיצרנו במחשב.

**cd GithubRepName**

**git clone** path – מקשרת בין המאגר הלוקאלי (במחשב האישי) לבין המאגר הגלובלי (אתר github), ע"י קובץ מוסתר בשם git.

- פקודה זו מתבצעת פעם אחת בלבד בפרויקט.
- נדרש להעתיק את הקישור מהמאגר הגלובלי.

**cd GithubRepName** – היות ונוצרה תיקיה חדשה לאחר ביצוע clone

# GitHub

## סדר פעולות – העלאת קבצים

svcollege
ללמוד. לדעת. לעבוד.

**– git status**

- מציגה את הקבצים המופיעים בתוך המאגר הלוקאלי.
- הקובץ באדום מוגדר כ untracked file (קובץ לא מזוהה) משמע שאינו מעודכן במאגר הגלובאלי.

```
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        script.js
```

אז איך נעדכן קובץ חדש שהוספנו מהמאגר הלוקאלי למאגר הגלובאלי?

# GitHub

## סדר פעולות – העלאת קבצים

**git add -A** – מכין להעלאה את כל הקבצים בתיקייה.

**git add script.js** – מכין להעלאה קובץ ספציפי.

**לאחר מכן נוכל לרשום בשנית **git status** ונראה שהקבצים מוכנים להעלאה למאגר הגלובלי (צבועים בירוק).

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   script.js
```

# GitHub

## סדר פעולות – העלאת קבצים

**git commit -m”message” –**

- נקודת בדיקה/דגל (check point) – המצב האחרון שהיינו בו לפני השינוי, נוכל לחזור אחורה במידת הצורך (במקרה של תקלות – rollback)

- בתוך ה ” ” נרשום מלל חופשי שלנו המתאר את השינוי או התוספת.

**git push** – טעינת הקבצים למאגר הראשי.

Document

# GitHub

## סדר פעולות – הנגשת האפליקציה ל web (react)

יוצרים מאגר חדש ב github ותיקייה חדשה במחשב (מאגר לוקאלי).
מבצעים clone בין המאגרים (git clone path).
התיקייה הלוקאלית הפנימית קיבלה את שם המאגר מאתר github.
יוצרים בתיקייה נפרדת פרויקט ב React ומעתיקים את כל קבצי הפרויקט (מלבד התיקיה git.) לתיקיה הפנימית (שנוצרה ב clone).

**<u>התקנות בטרמינל (לאחר יצירת או העתקת הפרוייקט למאגר הלוקאלי):</u>**

1. npm I react-router-dom
2. npm I gh-pages – בשימוש בערוצים מוודא הפעלת ערוצים בצורה נכונה.
3. (חובה לעשות בכדי לקבל בסיום התהליך לינק ולא רק בהקשר של פרויקט עם ערוצים)

# GitHub

## סדר פעולות – העלאת פרויקט ב React

Import {HashRouter as Router,Switch,Route,Link} from ‘react-router-dom’;

- **<u>git ignore – אילו קבצים לא יעלו לסביבת ה production</u>**
  - **משאירים רק את ה node modules – מתוך פרויקט ריאקט בוחרים בקובץ זה ומוחקים את כל שאר הנתונים.**

- **<u>Package json – קובץ קונפיגורציה</u>**
  - <u>מתחת ל private נוסיף את הפקודה:</u>

"homepage" : "http://**nameOfGitHub**.github.io/**nameOfReposetory**",

  - <u>בתוך הבלוק של ה- script נוסיף את הפקודה:</u>

"deploy" : "npm run build&&gh-pages -d build",

# GitHub

## סדר פעולות – העלאת פרויקט ב React

```
"name": "orgadapp",
"version": "0.1.0",
"private": true,
"homepage" : "http://orgadmahluf.github.io/react",

"dependencies": {
  "@testing-library/jest-dom": "^4.2.4",
  "@testing-library/react": "^9.4.0",
  "@testing-library/user-event": "^7.2.1",
  "gh-pages": "^2.1.1",
  "react": "^16.12.0",
  "react-dom": "^16.12.0",
  "react-router-dom": "^5.1.2",
  "react-scripts": "3.3.0"
},
"scripts": {
  "react":"npm run build&&gh-pages -d build",
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
```

לשים לב לשנות את
שם ה github ושם ה repository

# GitHub

## סדר פעולות – העלאת פרויקט ב React

cd repName

<u>בטרמינל:</u>

git add -A

git commit –m”msg”

git push

npm run deploy

ב github בתוך settings יהיה קישור לפרויקט שלנו.