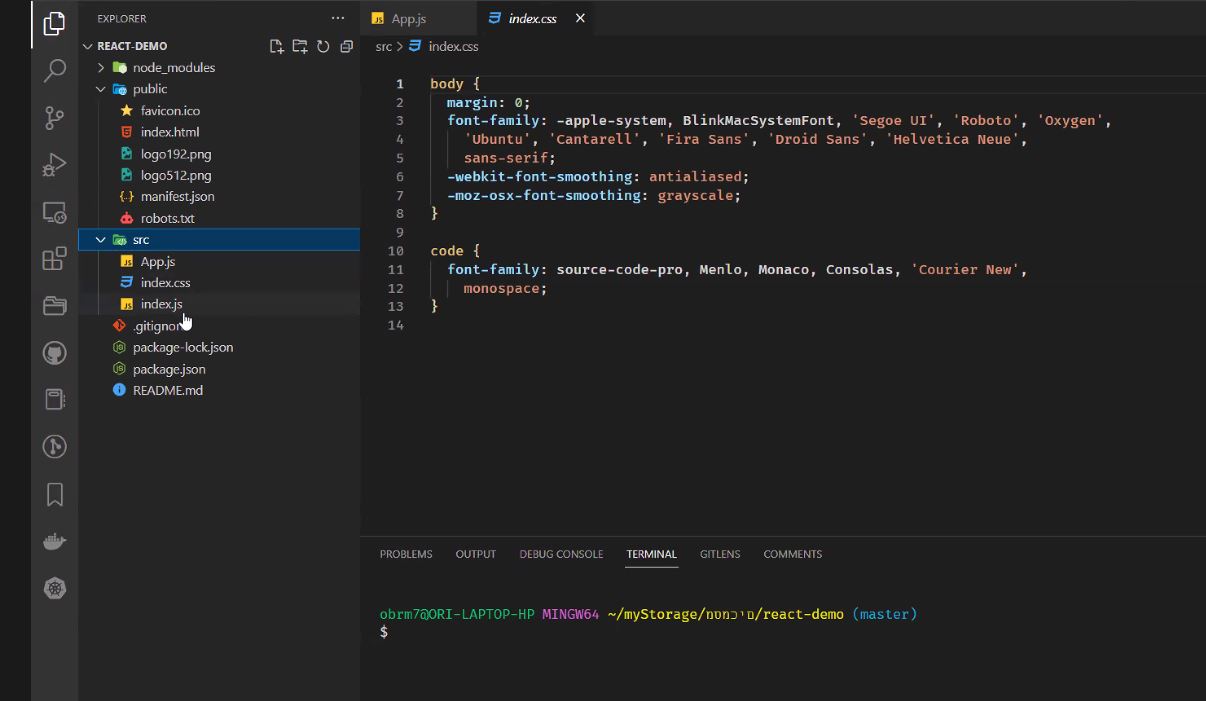
# React

## Setup

* **Create project**

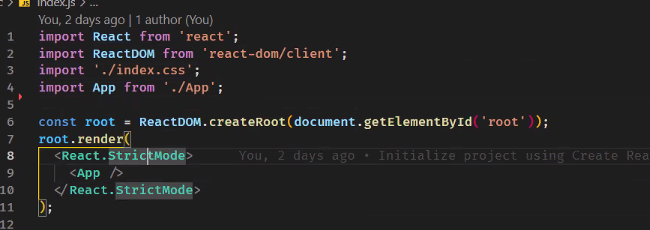
### 

* **Run project**: cd <project name>, Npm start
* **Delete some stuff**: watch lecture recording for detailed setup.

this is how it should look like afterwards: Text

Description automatically generated

* **Index:**



* **App:** empty function.
* **Create one template project**.
* For each project: Delete package lock, copy the src file and the and the packege json to the project file. When you save your stuff: save the packege json and the src file, not an entire project. If you want to continue working on a project (thus use the node modules again). When you’re in the file, you just do **npm i** (install the node modules).

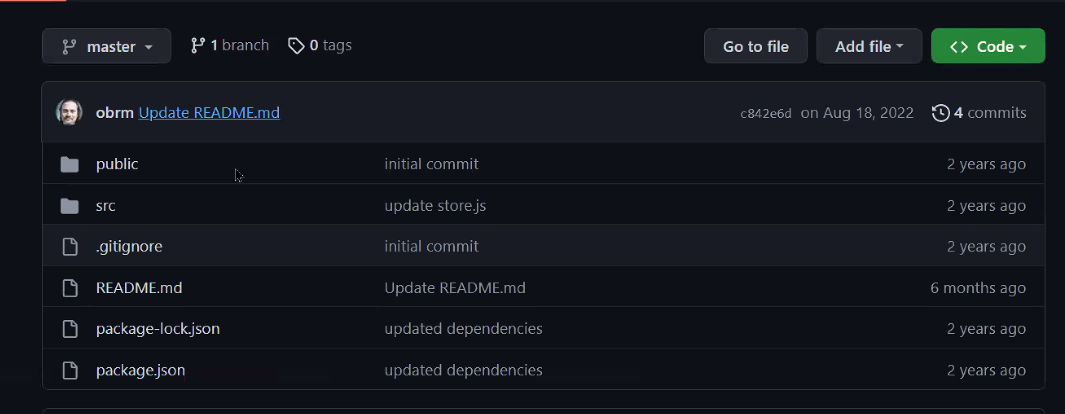
### Project structure

* **At the bottom of each component code:**

export default app

### uploading to github

the files you need to upload to github:



## Useful packages

### Helmet

A package that changes the name of the page to the name of the current component(השם בלמעלה בלשונית של האינטרנט)

Helmet.title = sth sth

## Basic terms

### Linter

Reads your code and checks for errors, bad code etc.

### JSX

JavaScript XML

### Imperative and \_\_ approach

* Impeartive: giving clear, step by step „instructions“, telling the DOM what to do (e.g. create new element, element.innerText, append).
* Declaratig : defining the desired end state using syntactic sugar. React translates your input to valid browser code in the background.

## React functions

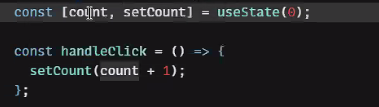
### Hooks

### 

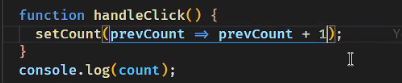
### useState

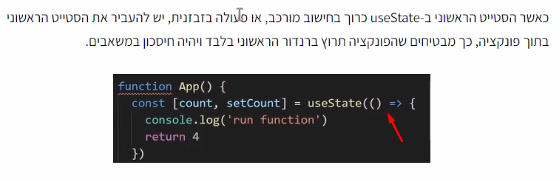
Set: renders a component

Case 1: change the value to sth new (if you update a value this syntax is wrong, for the right one see case 2.



Case 2: using the old value to get a new one: use prevCount.



Case 3: render only in the first run: 

### useEffect

When to use: whenever there’s an action that should be executed in response to another action.

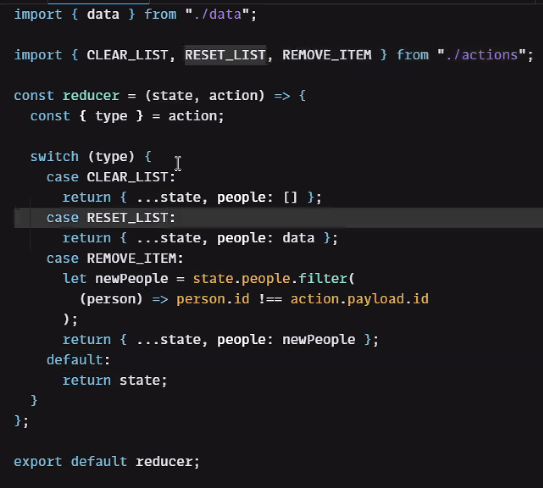
Cleanup function: runs before every time the effect function runs, and after it’s being demounted from the DOM.

usecases: **event listener scroll**, **setInterval**, window.title (manipulating the website title), **updating local storage**, **fetching sth by id** (when you choose an id or sth) – fetch accordingly, **debounce**: set timeout for the useState – so after the input is done being typed it waits for half a second and sends a request for the server.

### useReducer

when to use: when you update a state, that is dependant on another state.

Why: The state could in some cases, rely on a state that is not yet updated.



### useContext

When to use: when components always have the same props (e.g. input in a form always gets different infomation, so when you do an Input component, you should use props), and get a state from somewhere else in the app.

### useParams

Refering to the url adress: if you want to use the adress (e.g. based on the key of the product), in order to find, filter, in a page that is separated from where the state is.

### useImperativeHandle

if you want to give an input a ref, and it gives you an error, check it out.

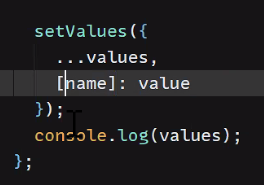
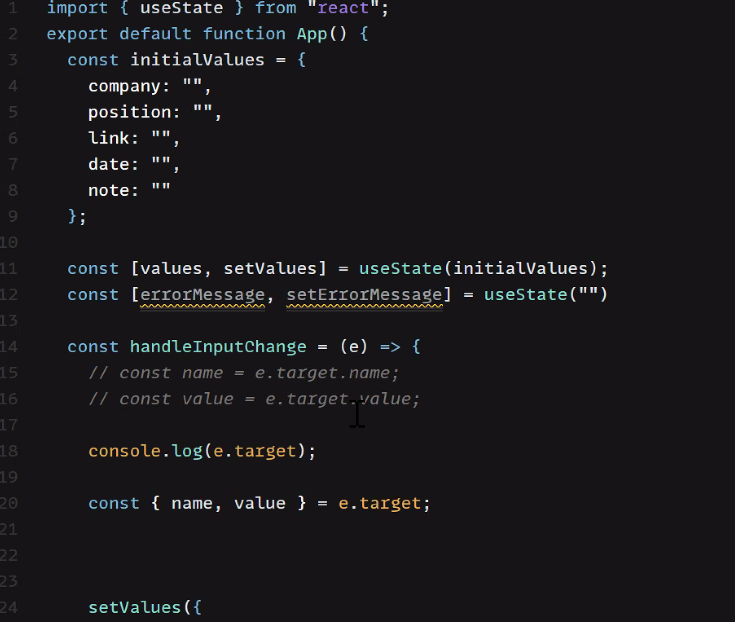
### Custom hooks

If you call a custom hook in a component, it is not a shared state throughout the app. The state is then tied to the component (only inside of it).

## Useful stuff

### Form – controlled (controlled component)

Dynamically getting every input, without setting a usestate per each input of the form.



Using the squared brackets here to updated the value of the chosen key. name can be different every time here.

### Form – uncontrolled (uncontrolled component)

Meaning of uncrontolled – the internal state (the value) is not controlled by react. There’s a direct manipulation of the DOM, and the virtual DOM and then rendering.

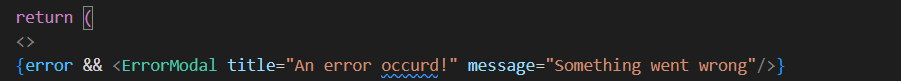
Use ref does not render the component.

Text

Description automatically generated

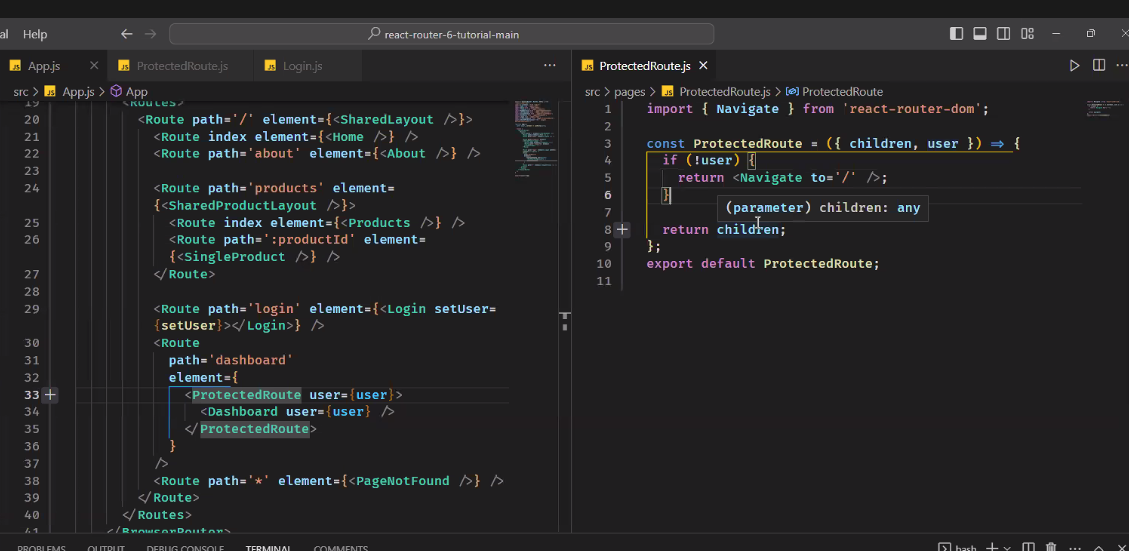
### If sth, then gimmie that jsx: &&

In this case, if error is definend (I initialized with no value).



## Route

### Protected Rout



If not user user navigate to home, else go to line 34 left – the user dashboard.

### Relative path

An absolute path goes always to the same address. A relative path, adds the path after the currently active path.

Setting a <Link relative:”path”> makes the url relative to the path. The default definition is “route”, and makes the url relative to the parent element. So if you go a step back, you wouldn’t remove the last segment of the path, you’d go back to it’s parent url (in the app).