React Notes

TSX / JSX

→ allows you to return html tags with javascript in them

→ you can create variables as html:
const name = <h1> Name </h1>

Components

→ a javascript function that return some tsx/jsx

→ components can be called in other tsx files:
<Component/>

Props

→ every react component will take props;

→ you can pass any type of data in props;

→ it’s basically a parameter for components

→ example:
function MyComponent (props) {
    return (

             Name: {props.name}
             Age: {props.age}
        …
) }

→ now when you use the above component, you just pass in parameters:
<MyComponent name=”…” age=… /> and that’s about it

Prop Types

→ import PropTypes from ‘prop-types’

→ MyComponent.propTypes = {
name: PropTypes.string,
age: PropTypes.number,
}

Default Props

→ this are default values for props, if they are not passed from the parent component

→ MyComponent.defaultProps = {
name: “MyName”,
age: 0
}

CSS in React

→ you give your html elements in your TSX’s files
className=’my-class’ and if you import ‘../…/style.css’

and you access those className via style.css

→ similar to average html + css

→ you can pass it with style.module.css
import styles from “./ style.module.css”
<h1 className={styles.name} > instead of <h1 className = ‘name’>

Conditional Rendering

```
return (
    <div className='..'>
        {myVariable ≥ 18 ? <h1>Show this if myVariable is >= 18 !</h2> : <h1>This means myVariable is <= 18 !</h1> }
    </div>
)
```

Lists

- const names = [’Tudor’, ‘Rares’, ...]
- names.forEach - parse through all of the names
- names.filter
- names.map((name, [not mandatory key(basically the index) / can be the id]) ⇒{
  return <h2 key={key}> {name} </h2>
  });
- names.reduce
  you can do this even if your list has Objects and access the objects fields {obj.field}

useState Hook

- it is used for telling react to re-render the page when smth happens to that var:

const [varName, setVarName] = useState(initialValueOfTheVar);

so whenever setVarName is called ⇔ varName is changed, react re-renders

- HOW TO CHANGE CSS w useState:
  - <div style={{color: textColor}}>
    const [textColor, setTextColor] = useState(”black”);
    onClick = { () ⇒ {
    setTextColor = “red”}} or have a handleOnClick for it

Component lifecycle

- mounting - start appearing
- updating - changing
- unmounting - stopped appearing

useEffect Hook

- triggers for each lifecycle step
- useEffet(() ⇒ {
    //useEffect is called everytime the component state changes
    console.log(”Component mounted~!”);

    return () ⇒ {
      console.log(”This is called only when unmounted”);
  }
  }, [*here you can add the variable that changes or som shit*])

how to fetch data from an api

- you make a request, get the data and then display it to your website or whatever
- fetch(”api.url”) - uses to fetch data from API:
  - you grab the url from the api

- fetch() → json
- fetch().then((response) ⇛ response.json())
  .then((data) ⇛ {
  do smth with the data
  })

Axios library

- library to fetch data
- import Axios from “axios”
- Axios.get(”api.url”).then((response) ⇛ {

  response.data → manipulate it
  })
- BETTER WAY

---

useContext() hook

- allows to share values between multiple levels of components without passing props through each level
- Provider Component:
  - it is a component that provides the info to all others:
    - export const MyContext = createContext();
    - < MyContext.Provider value={myVar}>
      ...
      </MyContext.Provider>
- Consumer Component
  - you import {MyContext} from ‘.../.../ProviderComponent.tsx’
  - const user = useContext(MyContext);

    - then you get that myVar all the way to ConsumerComponent as user and u can use it for virtually anything you want
- Multiple Consumer Component:
    - you just do the same thing for any other component that needs that context

---