**Begin   
Jak włączyć wsparcie?**$npx // używane do binarnych skryptów do inicjalizacji i do operacji bez projektu

$npx create-react-app first --template typescript

$npm start - creates developer server

\*\*public\*\*

index.html - <div id=”root”> <- miejsce instalacji aplikacji w reacie

\*\*src\*\*

index.tsx

root.render(

<React.ScriptMode> <App /> </>);

) /// uruchamia elementy

className = {value} /// atrybut jak class w html’u

file name & component name (e.x. App) should start with upperCaseLetters

‘

\*\*hooks\*\*

\*\*Example component\*\*

function Message() {  
 *// JSX: Will be compiled to js form a'like html* const *temp* = 'Abxc';  
 return <**h1**>**Hellow** {*temp*}</**h1**>;  
}  
  
export default Message;

import Message from "./Message";  
  
function App() {  
 return <**div**><**Message**/></**div**>;  
}  
  
export default App;

\*\*Wypisywanie tablicy jako listy

function ListGroup(){  
 const *items* = ["Hi", "There", "One"];  
 *items*.map(item => <**li**>{item}</**li**>)  
 return (  
 <><**h1**>**List**</**h1**>  
 <**ul** className="list-group">  
 {*items*.map(item => <**li** key={item} className="list-group-item">{item}</**li**>)}  
 </**ul**></>  
 );  
}

\*\*Hooks\*\*

używane do dostawania się do metod reacta

\*updating number\*

const arr[{nazwa zmiennej przechowywanej}, set{nazwa zmiennej}] = useState({value}) // [0] – varieble; [1] – updater function

const [*selectedIndex*, setSelectedIndex] = useState(-1);

onClick={() => {setSelectedIndex(index)}}

\*forms\*

do przechwytywania zmian inputa

interface ToDoFormProps{  
 *addElement*: (todo:Todo) => void;  
}  
  
export const ToDoForm: FC<ToDoFormProps> = ({addElement}) => {  
  
 const *titleRef* = useRef<HTMLInputElement>(null);  
 const handleSubmit = (e : FormEvent<HTMLFormElement>) => {  
 e.preventDefault();  
  
 const *title* = *titleRef*.*current*?.*value* ?? '';  
 const *todo*: Todo = {*title*: *title*, *id*:*title*, *isDone*: false};  
 addElement(*todo*);  
 }  
 return(  
 <>  
 <**form** onSubmit={(e) => handleSubmit(e)}>  
 <**input** ref={*titleRef*} type="text" name="" id="" required={true}/>  
 <**button** type="submit">**Send data!**</**button**>  
 </**form**>  
 </>  
 );  
};

function App() {  
 const [*todos*, todoesSet] = useState<Todo[]>([]);  
  
 const handleAt = (todo: Todo) => {  
 const *oldTodos* = [...*todos*, todo];  
 *// oldTodos.push(****todo);*** todoesSet(*oldTodos*);  
 }  
 return (  
 <**div** className="App">  
 <**ToDoList** todos={*todos*}/>  
 <**ToDoForm** addElement={handleAt}/>  
 </**div**>  
 );  
}

\*\*Additional \*\*

<></> - pomiędzy tym można zwracać parę obiektów na raz z componentu, dlatego bo bazowo tam jest używany <Fragment>

\*children\*

Do pozyskania dzieci z App.tsx jako argument

interface Prop{

children:text;

}

children: ReactNode; // Przesyła html content

**Formularze**

**\*\*NanoId\*\***

Generuje losowy string jako id

\*\*Układ katalogów\*\*

\*\*src\*\*

components  
\*features\* - wizualna reprezentacja   
toDoList – wszystko

src – features – {name e.x. ToDoList} – components/types