**Введение**

Typescript – это мощный препроцессор, который позволяет сократить код и сделать его безопаснее за счет типов, потому React начали использовать вместе с ним.

Кроме всего в разработке с ts появляется возможность пользоваться автопроверкой в редакторах кода.

Только чистый React его не поддерживает, но есть специальный шаблон CRA (create-react-app) для поддержки typescript:

npx create-react-app *название проекта* --template typescript

**Расширение .tsx**

Для того чтоб файл мог использовать typescript необходимо использовать расширение tsx. Это не означает, что весь проект нужно писать только на typescript: обычный javascript также можно использовать.

**Хуки**

Типизация хуков происходит с помощью дженериков:

interface ITodo {

id: number,

text: string,

complete: boolean

}

const [todos, setTodos] = useState<ITodo[]>([])

В случае хука useRef его следует типизировать так:

const ref = React.useRef<HTMLCanvasElement | null>(null)

Теперь он может содержать canvas или null. Это делает код безопаснее.

**События**

function Scriber() {

function submit(e: React.FormEvent) {

e.preventDefault()

}

return <form

onSubmit={e => submit(e)}

>

<button

type='submit'

className='Submit'

>

Submit

</button>

</form>

}

**Пропсы**

При помощи интерфейсов можно сделать проверку пропсов легче:

interface Props {

todos: ITodo[],

toggleComplete: Function,

count ?: number

}

function TodoList({ todos, toggleComplete, count=todos.length }: Props) {

return <div className='TodoList'>

{todos.map(el =>

<TodoItem

toggleComplete={toggleComplete}

todo={el}

key={el.id}

/>

)}

{count}

</div>

}

**Расширение стандартных html атрибутов**

Часто нужно расширить какой-то стандартный html атрибут. В ts это можно сделать, если расширить интерфейс пропсов:

interface Props extends React.HTMLAttributes<HTMLButtonElement> {

icon: string

}

function IconButton({ children, icon, ...other }: Props) {

return <button {...other}>

<div>{icon}</div>

{children}

</button>

}

<IconButton icon='home' onClick={() => console.log('Click')} />

**Перечисления**

enum AlertFlag {

SUCCESS = '#009933',

FAIL = '#ff6666',

LOAD = '#6699ff'

}

interface Props {

flag: AlertFlag,

title: string

}

function Alert({ flag, title}: Props) {

return <h1 style={{ color: flag }}>{title}</h1>

}

<Alert flag={AlertFlag.SUCCESS} title='New Song Linkin Park'/>

**Получение children**

Если компонент не расширяет html атрибут, то чтоб получить children, нужно напрямую расширить функцию от React.FunctionComponent (сокращение React.FC)

interface Props {

flag: AlertFlag

}

const Alert: React.FC<Props> = ({ flag, children }) => {

return <h1 style={{ color: flag }}>

{children}

</h1>

}

<Alert flag={AlertFlag.SUCCESS}>

Text

</Alert>

**Подключение библиотек, не написанных на TypeScript**

Если библиотека не написана на TypeScript, то он просто его не поймет, поэтому выдастся ошибка. В таких случаях есть уже готовые интерфейсы этих библиотек на репозитории @types.

npm i @types\react-router-dom

npm i “@types\react-router-dom”