**1. Пропсы (Props)**

Компоненты в React принимают пропсы. Они должны быть типизированы, чтобы избежать ошибок.

Для функциональных компонентов:

type MyComponentProps = {

title: string;

count?: number; // необязательное свойство

};

const MyComponent: FC< MyComponentProps > = ({ title, count = 0 }) => {

return {title} - {count};

};

**2. Состояние (State)**

Для компонентов с состоянием нужно типизировать state.

С использованием хука useState:

const [count, setCount] = useState<number>(0);

**3. Контекст (Context)**

React.Context также требует типизации.

Создание контекста:

type Theme = 'light' | 'dark';

const ThemeContext = createContext<Theme>('light');

**4. Ссылки (Refs)**

Ссылки (ref) используются для доступа к DOM-элементам или компонентам.

Типизация рефов:

const inputRef = React.useRef<HTMLInputElement | null>(null);

**5. Обработчики событий**

Примеры:

const handleChange = (event: ChangeEvent<HTMLInputElement>) => {

console.log(event.target.value);

};

const handleClick = (event: MouseEvent<HTMLButtonElement>) => {

console.log('Button clicked');

};

**6. Хуки (Hooks)**

Для useReducer:

type State = { count: number };

type Action = { type: 'increment' | 'decrement' };

const reducer = (state: State, action: Action): State => {

switch (action.type) {

case 'increment':

return { count: state.count + 1 };

case 'decrement':

return { count: state.count - 1 };

default:

return state;

}

};

**7. Высшего порядка компоненты (HOC)**

HOC (Higher-Order Components) требуют типизации пропсов, передаваемых как исходному компоненту, так и обертке.

Пример:

function withLoading(

Component: React.ComponentType

): React.FC {

return ({ loading, ...props }: { loading: boolean } & T) => {

return loading ? Loading... : ;

};

}

const MyComponent = (props: { name: string }) => {props.name};

const MyComponentWithLoading = withLoading(MyComponent);

**8. Асинхронные данные (Async data или API responses)**

При работе с асинхронными данными (например, запросы через fetch), важно типизировать, что возвращается из API.

Пример:

type User = {

id: number;

name: string;

};

const fetchUser = async (): Promise<User> => {

const response = await fetch('/api/user');

return response.json();

};

**9. Ключи в списках (Keys in Lists)**

Типизация создаваемых массивов JSX-элементов в списках.

Пример:

type Item = { id: string; value: string };

const items: Item[] = [{ id: '1', value: 'A' }, { id: '2', value: 'B' }];

const MyComponent = () => (

{items.map((item) => (

<div key={item.id}>{item.value}</div>

))}

);

**10. Роуты с react-router**

При использовании маршрутизации важно типизировать параметры пути.

type Params = { id: string };

const MyComponent = () => {

const { id } = useParams<Params>();

return <div>ID: {id}</div>;

};

**11. Дефолтные и children-пропсы**

Часто нужно задавать children (или любые стандартные пропсы):

Пример с children:

type PropsWithChildren = {

children: ReactNode,

title: string;

};

const MyComponent: FC< PropsWithChildren > = ({ children, title }) => {

return (

{title}

{children}

);

};

**12. Типизация стилей**

При использовании объектов стилей их также стоит типизировать.

Пример:

const style: React.CSSProperties = {

color: 'red',

fontSize: '16px',

};

const MyComponent = () => {

return (

<div style={style}>Hello</div>

)

}