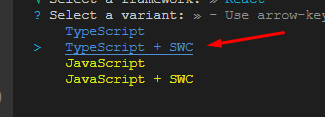
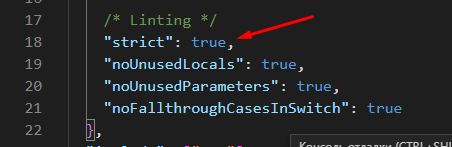
**npm create vite@latest**



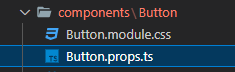


Должен быть включен strict режим:



**Типизация компонент**

Для типизации кнопки будем использовать интерфейс. Удобно наследовать.



import { ReactNode } from 'react';

// экспортируем интерфейс, который описывает те props, которые нужно передать в button

// ReactNode - это может быть строка, число, react элемент, JSX элемент конструктор, ReactPortal, undefined и т.д.

export interface ButtonProps {

  children: ReactNode;

}

Откуда берутся эти типы?

**devDependencies** – зависимости, которые используются при разработке.

Эти две зависимости обепечивают как раз типы для react / reactdom

    "@types/react": "^18.2.43",

    "@types/react-dom": "^18.2.17",

import { ButtonProps } from './Button.props';

function Button({ *children*, ...*props* }:ButtonProps) { *// типизируем пропсы, используя ButtonProps*

  return (

    <button *className*="button accent" {...props}>

      {*children*}

    </button>

  );

}

Теперь, если мы не пропишем внутрь кнопки children:

function App() {

  return (

    <>

      <Button/>

    </>

  )

}

То будет ошибка.

function App() {

  return (

    <>

      <Button>Кнопка</Button>

    </>

  )

}

Попробуем на кнопку навесить событие клика:

function App() {

  return (

    <>

      <Button onClick={()=>(console.log(1))}>Кнопка</Button>

    </>

  )

}

Мы не можем навесить ничего, кроме пропсов, так как это описали в интерфейсе.

Доработаем.

import { ButtonHTMLAttributes, ReactNode } from 'react';

// экспортируем интерфейс, который описывает те props, которые нужно передать в button

// ReactNode - это может быть строка, число, react элемент, JSX элемент конструктор, ReactPortal, undefined и т.д.

export interface ButtonProps extends ButtonHTMLAttributes<HTMLButtonElement> {

  // наш интерфейс наследуется от html атрибутов от HTMLButtonElement

  children: ReactNode;

}

import cn from 'classnames';

function Button({ children, className, ...props }: ButtonProps) {

  // типизируем пропсы, используя ButtonProps

  return (

    <button className={cn('button accent', className)} {...props}>

      {children}

    </button>

  );

}

// типизация функционального компонента - альтернатива нижнего

// тоесть мы вернули функциональный компонент вот с такими пропсами: <ButtonProps> - используем дженерик <>

export const ButtonAlt: FC<ButtonProps> = ({

  children,

  className,

  ...props

}) => {

  // FC - function component

  return (

    <button className={cn('button accent', className)} {...props}>

      {children}

    </button>

  );

};

**Типизация hoock и событий**

1) Типизируем useState:

function App() {

  const [counter, setCounter] = useState<number>(0);

  return (

    <>

      <Button className="btn" onClick={() => setCounter(1)}>

        Кнопка

      </Button>

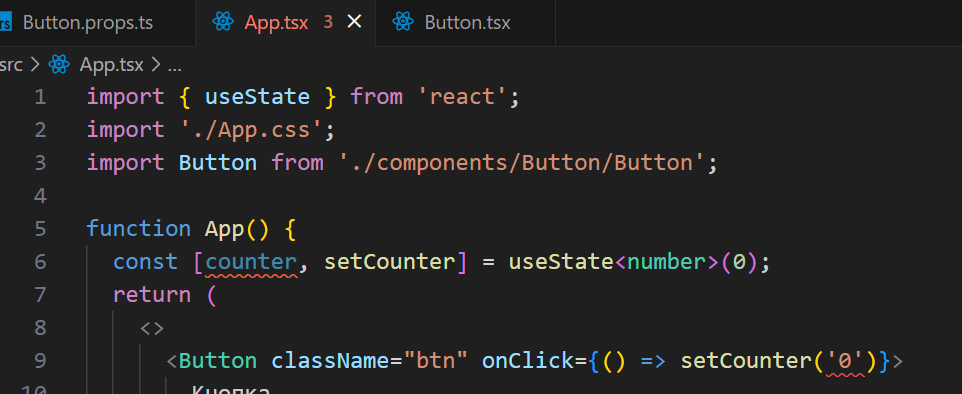
    </>

  );

}

Теперь мы можем передать только число.

Иначе ошибка:



useEffect не требует типизации.

2) Типизация event.

function App() {

  const [counter, setCounter] = useState<number>(0);

  const addCounter = (e: MouseEvent) => {

    // MouseEvent - синтетическое событие, и вся типизация этих событий лежит напрямую в реакт

    console.log(e);

  };

  return (

    <>

      <Button className="btn" onClick={addCounter} type="button">

        Кнопка

      </Button>

    </>

  );

}