React - counter

проектная работа

Тех задание к проекту (приложение счетчик)

Стек:

- JavaScript
- CSS
- React
- webpack/vite

UI: bootstrap библиотека (react-bootstrap)

Приложение должно иметь определенную архитектуру (указано далее)

Приложение должно иметь функционал переключения темы (светлая и темная)

В приложении должен быть закрепленная сверху шапка с названием проекта и кнопкой переключения темы

Приложение должно состоять из 15 счетчиков:

- 3 счетчика без связи реализованные с помощью useState хука, пользовательскокго хука и редьюсера
- 2 счетчика со связью состояние через props и реализацией через useState
- 2 счетчика со связью состояние через props и реализацией через пользовательский хук
- 2 счетчика со связью состояние через props и реализацией через редьюсер
- 2 счетчика со связью состояния через контекст и реализацией через useState
- 2 счетчика со связью состояния через контекст и реализацией через пользовательский хук
- 2 счетчика со связью состояния через контекст и реализацией через редьюсер

Инструкция по созданию react-приложения

Создать приложение react-counter-app

Для создания приложения использовать команду

npx create-react-app react-counter-app --template empty

Флаг ***--template empty*** создает пустой проект (без лишнего)

Альтернатива:

использовать сборщик **vite**

использовать онлайн-ресурс **stack-blitz** (имеется шаблон react-js)

Архитектура проекта

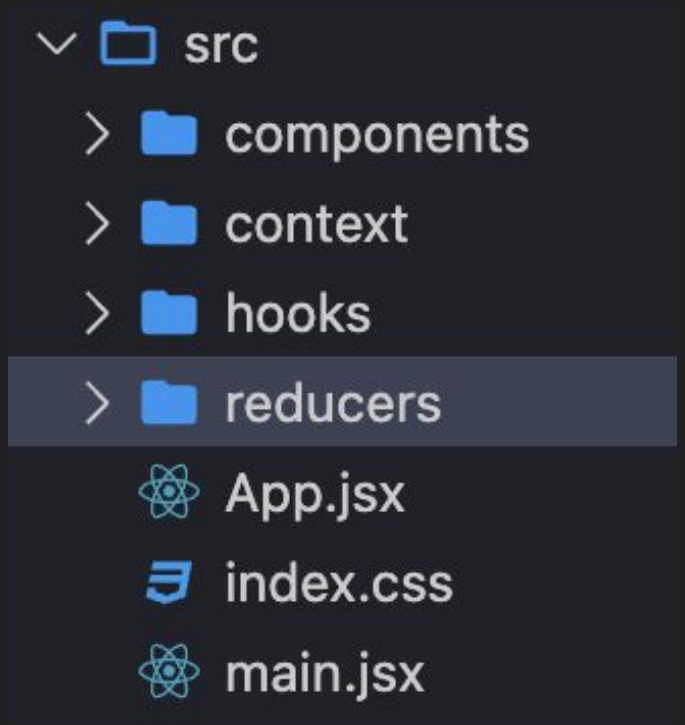
Архитектурно разделить файлы с кодом по директориям (папкам)

App.jsx, main.jsx и **index.css** должны быть в корне исходника (src)

Далее такая иерархия:

- **components**
- **context**
- **hooks**
- **reducer**

Данная архитектура является простой и удовлетворяет требования к проекту

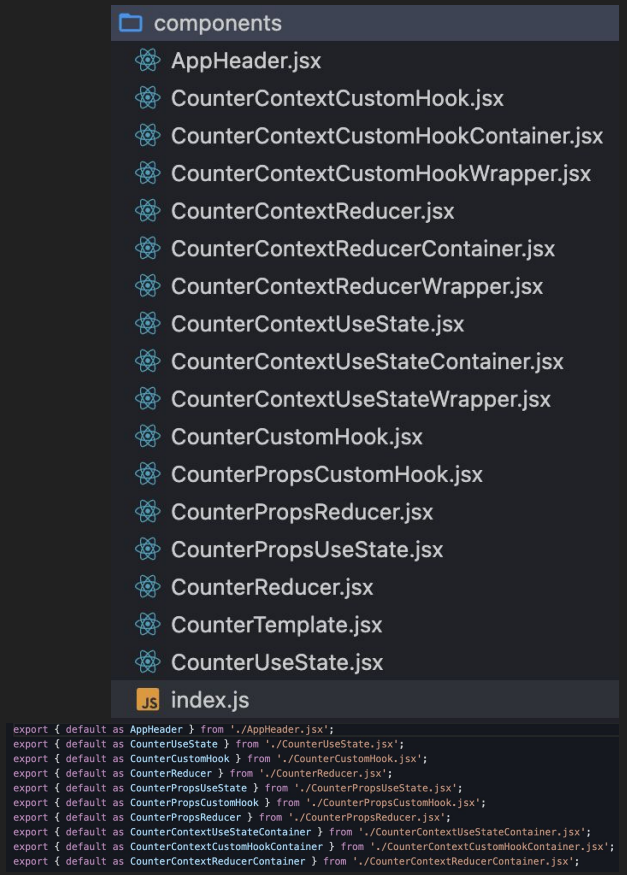


Директория components

Содержит в себе все компоненты используемые в проекте (то есть все файлы с разрешением **.jsx** за исключением **App.jsx** и **main.jsx**)

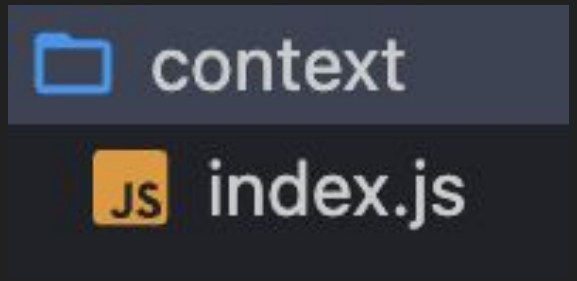
Названия компонента должно быть обязательно с большой буквы **CamelCase**, а так-же название должно быть осознанным и нести в себе некую информацию о себе

Создать файл “шлюз” для реэкспорта (**export/import**) чтобы в дальнейшем обращаться не к полноценному пути до компонента а деструктуризацией вытаскивать нужный компонент из импорта с папки **components**



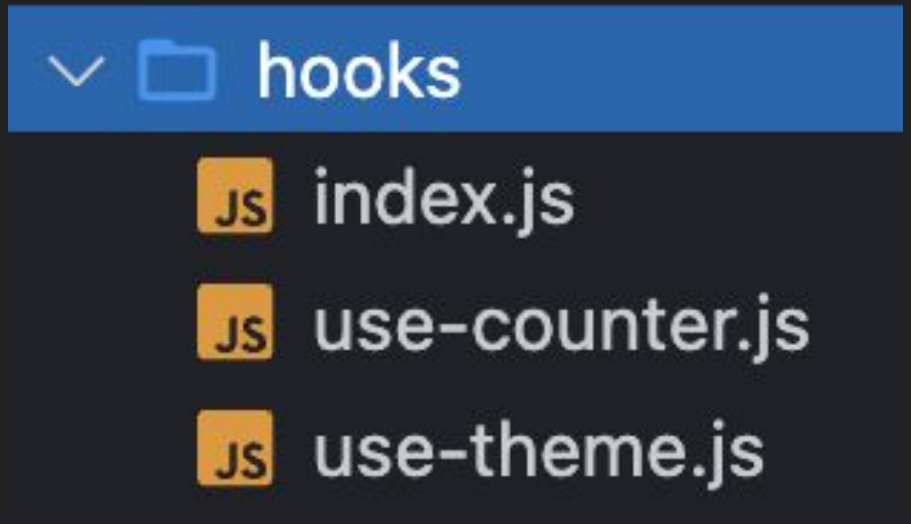
Директория context

Содержит в себе один файл index.js с описанным в нем созданием контекстов, так как проект достаточно простой и учебный разбиение создания контекстов по разным файлам не имеет смысла, при дальнейшем расширении функционала так-же можно продолжить наращивать файл index.js кодом, это не приведет к проблемам читаемости так как создание контекста производится в одну строку



Директория hooks

Директория содержит в себе файл “шлюз” index.js для реэкспорта хуков, а так-же файлы с реализацией хуков (каждый хук - отдельный файл) это требуется для более удобного ведения кода и легко-читаемости проекта

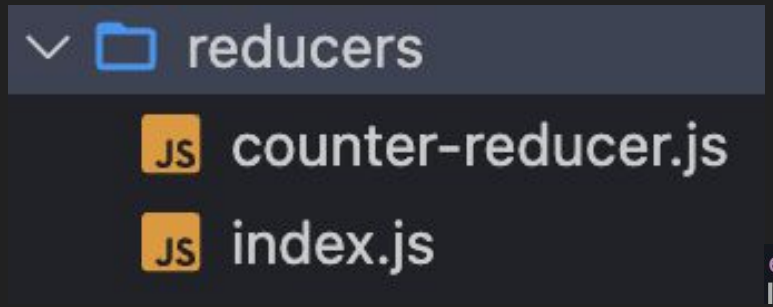



```
export { default as useTheme } from './use-theme.js';
export { default as useCounter } from './use-counter.js';
```

Директория reducer

Директория reducer должна содержать в себе файлы с реализацией reducer-ов и файлом “шлюзом” index.js для реэкспорта

Даже с учетом только одного reducer-а в проекте хорошим ведением кода будет изначальное разделение логики по файлам, для дальнейшего расширения функционала “без боли”, возможно будут добавляться другие reducer-ы с развитием проекта (задел на будущее)




```
export { default as counterReducer } from './counter-reducer.js';
```

Тема приложения и UI

Добавить на проект react-bootstrap командой

npm install react-bootstrap bootstrap

Использовать react-bootstrap UI компоненты

import { Button } from 'react-bootstrap';

Добавить пользовательский хук для контроля темы с использованием localStorage (если я выбрал темную тему, то при последующем посещении сайта с того-же браузере мой выбор запоминается)

const [theme, toggleTheme] = useTheme();

Добавить header в проект с кнопкой переключения темы приложения, вынести его в отдельный компонент ***AppHeader***

```jsx
import { Button, Container, Stack } from 'react-bootstrap';
import { useTheme } from '../hooks';

export default function () {
  const [theme, toggleTheme] = useTheme();

  return (
    <>
      <Container>
        <Stack direction="horizontal">
          <h1 className="">React counter app</h1>

          <Button
            className="ms-auto"
            variant={theme === 'light' ? 'dark' : 'light'}
            onClick={toggleTheme}
          >
            {theme}
          </Button>
        </Stack>
      </Container>
    </>
  );
}
```

```jsx
import { useEffect, useState } from 'react';

export default function () {
  const [theme, setTheme] = useState(localStorage.getItem('theme') || 'light');

  useEffect(() => {
    document.documentElement.setAttribute('data-bs-theme', theme);
    localStorage.setItem('theme', theme);
  }, []);

  const toggleTheme = () => {
    const nextTheme = theme === 'light' ? 'dark' : 'light';

    localStorage.setItem('theme', nextTheme);

    document.documentElement.setAttribute('data-bs-theme', nextTheme);

    setTheme(nextTheme);
  };

  return [theme, toggleTheme];
}
```

Counter - логика

Counter - счетчик, отображает число, дефолтное значение 0, имеет две кнопки: инкремент и декремент

Кнопка инкремент - при нажатии увеличивает значение счетчика на единицу

Кнопка декремент - при нажатии уменьшает значение счетчика на единицу

Реализовать компонент шаблон который в дальнейшем будет переиспользоваться в проекте

имеет пропс (props) counter и title, так-же два колбека (callback) increment и decrement которые подвязываются на UI составляющую компонента

```jsx
import {
  Badge,
  Button,
  Card,
  CardBody,
  CardHeader,
  CardTitle,
  Stack,
} from 'react-bootstrap';

export default function CounterTemplate({
  counter,
  increment,
  decrement,
  title,
}) {
  return (
    <>
      <Card>
        <CardHeader>
          <CardTitle>{title || 'Counter template'}</CardTitle>
        </CardHeader>
        <CardBody>
          <h3 className="text-center">
            <Badge bg="secondary">{counter}</Badge>
          </h3>
          <Stack direction="vertical" gap={2} className="align-content-center">
            <Button variant="success" onClick={increment}>
              increment + 1
            </Button>
            <Button variant="danger" onClick={decrement}>
              decrement - 1
            </Button>
          </Stack>
        </CardBody>
      </Card>
    </>
  );
}
```

Простой counter на основе useState

Используя встроенный хук React-а **useState** реализовать логику счетчика и далее соединить с помощью props & callbacks с компонентом шаблоном счетчика

инкремент и декремент вызывает функцию сеттер из хука **useState** что приводит к изменению состояния переменной counter и rerender (перерисовка) компонента

```jsx
import CounterTemplate from './CounterTemplate.jsx';
import { useState } from 'react';

const title = 'Counter useState';

export default function () {
  const [counter, setCounter] = useState(0);

  const increment = () => setCounter((prevVal) => prevVal + 1);

  const decrement = () => setCounter((prevVal) => prevVal - 1);

  return (
    <>
      <CounterTemplate
        title={title}
        counter={counter}
        increment={increment}
        decrement={decrement}
      />
    </>
  );
}
```

Counter с вынесенной логикой в пользовательский хук

Используя методологию пользовательских хуков вынести логику описанную с помощью встроенного хука React **useState** в отдельный пользовательский хук **useCounter**, далее переиспользовать ее в компоненте обертке в связке с шаблонным компонентом счетчика

Так как под капотом пользовательский хук **useCounter** работает на встроенном хуке React-а **useState** - он так-же будет вызывать rerender (перерисовку)

```jsx
import { useState } from 'react';

export default function (initialState = 0) {
  const [counter, setCounter] = useState(initialState);

  const increment = () => setCounter((prevVal) => prevVal + 1);

  const decrement = () => setCounter((prevVal) => prevVal - 1);

  return [counter, increment, decrement];
}
```

```jsx
import CounterTemplate from './CounterTemplate.jsx';
import { useCounter } from '../hooks';

const title = 'Counter hook';

export default function () {
  const [counter, increment, decrement] = useCounter(0);

  return (
    <>
      <CounterTemplate
        title={title}
        counter={counter}
        increment={increment}
        decrement={decrement}
      />
    </>
  );
}
```

Counter с реализацией в через reducer

Описать функцию reducer с логикой счетчика, далее используя встроенный хук React-а **useReducer** внедрить редьюсер в компонент обертку со связкой шаблонного компонента счетчика. Для реализации action-ов (действий) использовать **dispatch** из хука **useReducer**

Использование **dispatch** из хука **useReducer** так же как и сеттер из хука **useState** вызывает rerender (перерисовку)

```
export default function (state, action) {
  switch (action.type) {
    case 'increment':
      return { counter: state.counter + 1 };
    case 'decrement':
      return { counter: state.counter - 1 };
    default:
      return { counter: 0 };
  }
}
```

```
import CounterTemplate from './CounterTemplate.jsx';
import { counterReducer } from '../reducers';
import { useReducer } from 'react';

const title = 'Counter reducer';

export default function () {
  const [state, dispatch] = useReducer(counterReducer, { counter: 0 });

  const increment = () => dispatch({ type: 'increment' });

  const decrement = () => dispatch({ type: 'decrement' });

  return (
    <>
      <CounterTemplate
        title={title}
        counter={state.counter}
        increment={increment}
        decrement={decrement}
      />
    </>
  );
}
```

Передача состояния через props

Создать три компонента с вложенными двумя счетчиками использующими одно состояние (связанные состоянием), реализовать связку через передачу одного и того же состояние в пропсы (props) и одних и тех-же колбеков (callback), использовать шаблонные компоненты счетчика

Итого должно быть реализовано три связи через разные реализации:

- связь через props + реализация через **useState**
- связь через props + реализация через **useCounter** (пользовательский хук)
- связь через props + реализация через **useReduce** (редьюсер)

Преимущество такой связи - она легко реализуемая и явно видна в коде

Недостатки такой связи - при многоуровневой (вложенность компонентов) связки необходимо дублировать props & callbacks что повлечет собой жесткую связь и большое количество излишнего кода

Пример кода (useState, useCounter, useReducer)

```jsx
import CounterTemplate from './CounterTemplate.jsx';
import { Col, Row } from 'react-bootstrap';
import { useState } from 'react';

const title = 'Counter props + useState';

export default function () {
  const [counter, setCounter] = useState(0);

  const increment = () => setCounter((prevVal) => prevVal + 1);

  const decrement = () => setCounter((prevVal) => prevVal - 1);

  return (
    <>
      <Row className="g-4">
        <Col lg={4} md={4} sm={6} xs={12}>
          <CounterTemplate
            title={title}
            counter={counter}
            increment={increment}
            decrement={decrement}
          />
        </Col>
        <Col lg={4} md={4} sm={6} xs={12}>
          <CounterTemplate
            title={title}
            counter={counter}
            increment={increment}
            decrement={decrement}
          />
        </Col>
      </Row>
    </>
  );
}
```

```jsx
import CounterTemplate from './CounterTemplate.jsx';
import { Col, Row } from 'react-bootstrap';
import { useCounter } from '../hooks';

const title = 'Counter props + hook';

export default function () {
  const [counter, increment, decrement] = useCounter(0);

  return (
    <>
      <Row className="g-4">
        <Col lg={4} md={4} sm={6} xs={12}>
          <CounterTemplate
            title={title}
            counter={counter}
            increment={increment}
            decrement={decrement}
          />
        </Col>
        <Col lg={4} md={4} sm={6} xs={12}>
          <CounterTemplate
            title={title}
            counter={counter}
            increment={increment}
            decrement={decrement}
          />
        </Col>
      </Row>
    </>
  );
}
```

```jsx
import CounterTemplate from './CounterTemplate.jsx';
import { counterReducer } from '../reducers';
import { useReducer } from 'react';
import { Col, Row } from 'react-bootstrap';

const title = 'Counter props + reducer';

export default function () {
  const [state, dispatch] = useReducer(counterReducer, { counter: 0 });

  const increment = () => dispatch({ type: 'increment' });

  const decrement = () => dispatch({ type: 'decrement' });

  return (
    <>
      <Row className="g-4">
        <Col lg={4} md={4} sm={6} xs={12}>
          <CounterTemplate
            title={title}
            counter={state.counter}
            increment={increment}
            decrement={decrement}
          />
        </Col>
        <Col lg={4} md={4} sm={6} xs={12}>
          <CounterTemplate
            title={title}
            counter={state.counter}
            increment={increment}
            decrement={decrement}
          />
        </Col>
      </Row>
    </>
  );
}
```

Синхронизация состояния через контекст

Используя контекст через встроенный хук React-а **useContext** можно оборачивать компонент (или все приложение) неким контекстом, который будет предоставлять возможность обращения к нему в любом компоненте который входит во вложенность компонента с контекстом (можно воспринимать как общую шину над компонентом/приложением) и состояние будет общее (состояние контекста)

В компоненте где мы реализуем контекст передаем в компонент контекста функционал (создается общее состояние через контекст) - [родитель]

В компоненте где мы используем контекст мы отлавливаем через вложенность функционал который мы предоставили в родителе - [дочерка]

Преимущество - облегчает работу с вложенностью более чем один компонент, дает возможность глобального (в рамках контекста) контроля того или иного функционала

Недостаток - не явный функционал, не всегда сразу по коду понятно есть ли контекст или нет, так-же достаточно нагружено с точки зрения количества кода для тривиальных задач

Инструкция

Создаем контексты для каждой связки через встроенный в React функционал **createContext**

Создаем обертку которой будем давать данный контекст и в нем же реализуем логику счетчика тремя способами (**useState, useCounter, useReducer**).
Используем контекст как компонент с уточнением **.Provider**, функционал передаем через props **value** у данного компонента

Создаем дочерние компоненты, которые обращаются к тому или иному контексту, они не несут в себе реализацию логики счетчика, только визуал (UI).
Для отлова контекста используем встроенный хук React-а **useContext** с уточнением какой контекст мы отлавливаем

Так-же создаем максимально глупые компоненты обертки для демонстрации вложенности компонентов более чем в один компонент

Итого должно быть три вида связи:

- context + **useState**
- context + **useCounter**
- context + **useReducer** (через reducer)

Реализация контекстов createContext

```
import { createContext } from 'react';

const ContextCounterUseState = createContext(undefined);

const ContextCounterCustomHook = createContext(undefined);

const ContextCounterReducer = createContext(undefined);

export {
  ContextCounterUseState,
  ContextCounterCustomHook,
  ContextCounterReducer,
};
```

Пример компонентов “родителей”

```jsx
import CounterContextUseStateWrapper from './CounterContextUseStateWrapper.jsx';
import { Col, Row } from 'react-bootstrap';
import { ContextCounterUseState } from '../context';
import { useState } from 'react';

export default function () {
  const [counter, setCounter] = useState(0);

  const increment = () => setCounter((prevVal) => prevVal + 1);

  const decrement = () => setCounter((prevVal) => prevVal - 1);

  return (
    <>
      <ContextCounterUseState.Provider
        value={{ counter, increment, decrement }}
      >
        <Row className="g-4">
          <Col lg={4} md={4} sm={6} xs={12}>
            <CounterContextUseStateWrapper />
          </Col>
          <Col lg={4} md={4} sm={6} xs={12}>
            <CounterContextUseStateWrapper />
          </Col>
        </Row>
      </ContextCounterUseState.Provider>
    </>
  );
}
```

```jsx
import { Col, Row } from 'react-bootstrap';
import { ContextCounterCustomHook } from '../context';
import { useCounter } from '../hooks';
import CounterContextCustomHookWrapper from './CounterContextCustomHookWrapper';

export default function () {
  const [counter, increment, decrement] = useCounter(0);

  return (
    <>
      <ContextCounterCustomHook.Provider
        value={{ counter, increment, decrement }}
      >
        <Row className="g-4">
          <Col lg={4} md={4} sm={6} xs={12}>
            <CounterContextCustomHookWrapper />
          </Col>
          <Col lg={4} md={4} sm={6} xs={12}>
            <CounterContextCustomHookWrapper />
          </Col>
        </Row>
      </ContextCounterCustomHook.Provider>
    </>
  );
}
```

```jsx
import CounterContextReducerWrapper from './CounterContextReducerWrapper.jsx';
import { Col, Row } from 'react-bootstrap';
import { ContextCounterReducer } from '../context';
import { counterReducer } from '../reducers';
import { useReducer } from 'react';

export default function () {
  const [state, dispatch] = useReducer(counterReducer, { counter: 0 });

  const increment = () => dispatch({ type: 'increment' });

  const decrement = () => dispatch({ type: 'decrement' });

  return (
    <>
      <ContextCounterReducer.Provider
        value={{ counter: state.counter, increment, decrement }}
      >
        <Row className="g-4">
          <Col lg={4} md={4} sm={6} xs={12}>
            <CounterContextReducerWrapper />
          </Col>
          <Col lg={4} md={4} sm={6} xs={12}>
            <CounterContextReducerWrapper />
          </Col>
        </Row>
      </ContextCounterReducer.Provider>
    </>
  );
}
```

Пример компонентов “дочерок”

```jsx
import CounterTemplate from './CounterTemplate.jsx';
import { useContext } from 'react';
import { ContextCounterUseState } from '../context';

const title = 'Counter context + useState';

export default function () {
  const { counter, increment, decrement } = useContext
  (ContextCounterUseState);

  return (
    <>
      <CounterTemplate
        title={title}
        counter={counter}
        increment={increment}
        decrement={decrement}
      />
    </>
  );
}
```

```jsx
import CounterTemplate from './CounterTemplate.jsx';
import { useContext } from 'react';
import { ContextCounterCustomHook } from '../context';

const title = 'Counter context + hook';

export default function () {
  const { counter, increment, decrement } = useContext(
    ContextCounterCustomHook
  );

  return (
    <>
      <CounterTemplate
        title={title}
        counter={counter}
        increment={increment}
        decrement={decrement}
      />
    </>
  );
}
```

```jsx
import CounterTemplate from './CounterTemplate.jsx';
import { useContext } from 'react';
import { ContextCounterReducer } from '../context';

const title = 'Counter context + reducer';

export default function () {
  const { counter, increment, decrement } = useContext
  (ContextCounterReducer);

  return (
    <>
      <CounterTemplate
        title={title}
        counter={counter}
        increment={increment}
        decrement={decrement}
      />
    </>
  );
}
```

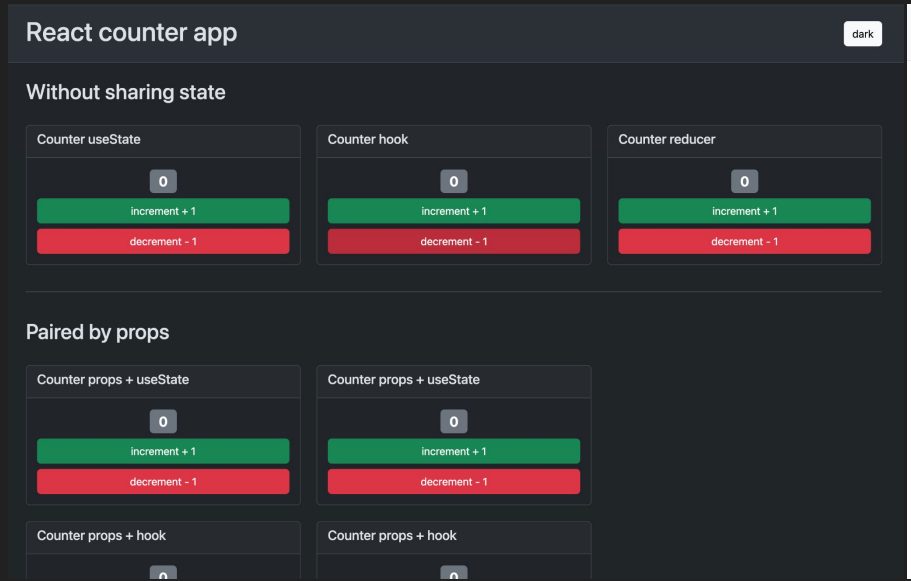
Промежуточные компоненты

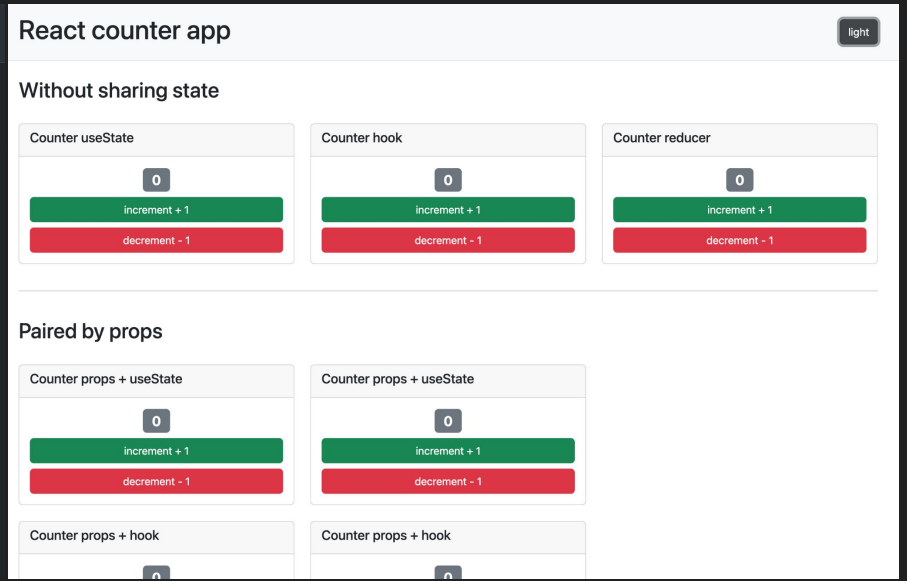
```jsx
import CounterContextUseState from './CounterContextUseState.jsx';

export default function () {
  return (
    <>
      <CounterContextUseState />
    </>
  );
}
```

```jsx
import CounterContextCustomHook from './CounterContextCustomHook.jsx';

export default function () {
  return (
    <>
      <CounterContextCustomHook />
    </>
  );
}
```

```jsx
import CounterContextReducer from './CounterContextReducer.jsx';

export default function () {
  return (
    <>
      <CounterContextReducer />
    </>
  );
}
```

Результат (примерный)





Ресурсы

Пример проекта - ТЫК

Пример проекта с TypeScript-ом - ТЫК (исходник), ТЫК (деплой)

react bootstrap - ТЫК

документация react - ТЫК

компоненты - ТЫК

useState - ТЫК

useReducer - ТЫК

useContext - ТЫК

useEffect - ТЫК

Кастомные хуки - ТЫК

vercel - ТЫК