**Создайте простое приложение Todo с помощью React**

Давайте создадим простое приложение Todo с помощью React, которое научит вас основным принципам CRUD (создание, чтение, обновление и удаление).

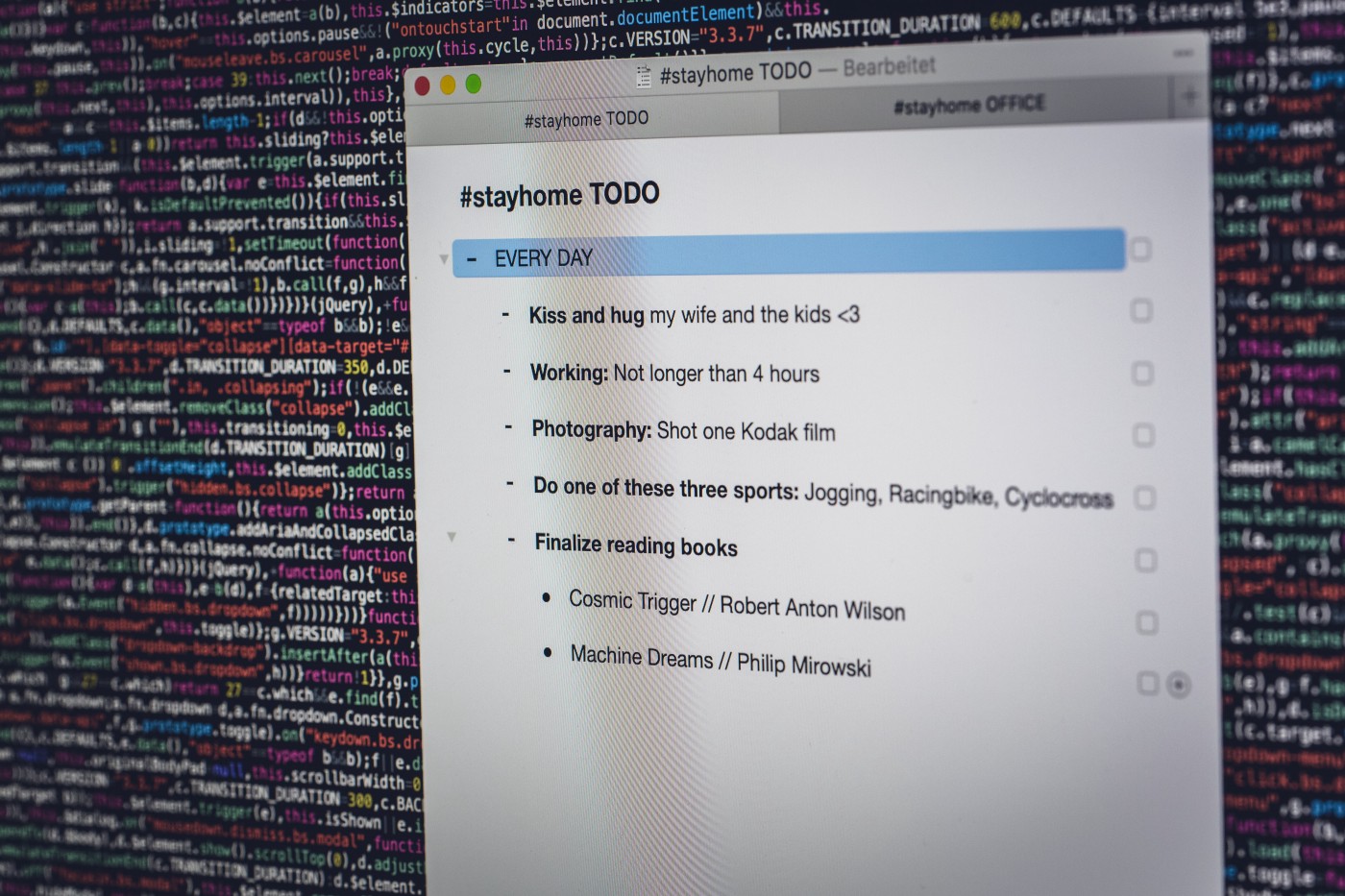


Фото [Маркуса Списке](https://unsplash.com/@markusspiske?utm_source=medium&utm_medium=referral) на [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral)

Здравствуйте, читатели! Я впервые пишу статью о создании чего-либо с помощью React. Итак, я также новичок в React и Frontend Frameworks. И лучший способ сделать свой первый проект React — это создать простое приложение Todo.

Создать приложение Todo легко и не займет много времени, но оно научит вас некоторым важным понятиям. Он учит вас принципам CRUD (создание, чтение, обновление и удаление), которые очень важно понимать любому разработчику.

Поскольку это наш первый проект в мире React, мы хотели бы упростить его. Мы не будем использовать Redux для управления состоянием и не будем использовать какой-либо сервер для управления им.

Создание простого списка задач означает, что мы не сможем отслеживать задачи после обновления страницы. Таким образом, это не идеальное решение, но хорошее начало.

Мы научимся создавать расширенное приложение Todo, которое будет включать Redux и сервер, но пока мы хотели бы, чтобы все было просто. Мы бы создали простое работающее приложение React с некоторым стилем.

Итак, начнем: -

**Настройка проекта**

Итак, давайте начнем создавать наше первое приложение React Todo. Мы будем использовать ***приложение create-react-app,*** чтобы помочь нам загрузить приложение React для нас.

npm install create-react-app

Выполнение приведенной выше команды установит *приложение create-react-app* в наш проект. Теперь это будет готово к использованию.

Чтобы создать новый проект с именем « *todo»* , мы должны запустить команду следующим образом:

create-react-app todo

Теперь это создаст папку с именем *todo* в нашем текущем репозитории. Затем мы должны перейти в папку *todo* , выполнив следующие действия:

cd todo

Теперь мы установим две библиотеки с помощью npm, которые помогут нам использовать библиотеку Bootstrap в нашем приложении для применения стилей.

npm install react-bootstrap bootstrap

Выполнение приведенной выше команды установит в нашем приложении библиотеки *react-bootstrap* и *bootstrap .*

Теперь мы готовы создать приложение.

**App.cs**

Теперь давайте напишем пользовательский код CSS для создания стилей. Не беспокойтесь об этой части. Это очень простой стиль, чтобы наше приложение Todo выглядело немного лучше.

.app {  
padding: 30px;  
background-color: floralwhite;  
}  
  
.todo {  
align-items: center;  
display: flex;  
font-size: 18px;  
justify-content: space-between;  
}

Мы сделаем остальную часть стиля, используя компоненты *react-bootstrap .*

**App.js**

Далее мы начнем с основной части, файла *App.js.*Мы начнем с импорта необходимых вещей в наш файл App.js.

import React from "react";  
import "./App.css";  
import { Button, Card, Form } from 'react-bootstrap';  
import 'bootstrap/dist/css/bootstrap.min.css';

Мы будем использовать React Hooks, чтобы позволить нам заменить компоненты на основе классов функциональными компонентами и по-прежнему использовать все функции без каких-либо проблем.

Итак, начнем с основной функции приложения. Мы определим список *задач* javascript, который будет содержать все наши задачи, а также будет содержать статус каждой задачи, независимо от того, выполнены они или нет. Мы будем использовать *setTodos* и *React.useState* , который включен React Hooks.

function App() {   
 const [todos, setTodos] = React.useState([   
 {   
 text: "Это пример задачи",   
 isDone: false   
 }   
 ]);   
}

Далее мы переходим к части добавления todos. Мы определим функцию *addTodo* и определим *newTodos* , которая возьмет список *задач* и добавит текст новой задачи к списку, используя оператор распространения. Затем мы используем *setTodos* , чтобы установить *newTodos* как todos.

const addTodo = text => {  
const newTodos = [...todos, { text }];  
setTodos(newTodos);  
};

Далее мы переходим к части пометки Todos как выполненной. Мы определим функцию *markTodo* . Мы используем оператор распространения, чтобы скопировать все задачи в newTodos, а затем помечаем задачу *как* выполненную, используя ее индекс, а затем устанавливаем *новые задачи* как задачи .

const markTodo = index => {  
const newTodos = [...todos];  
newTodos[index].isDone = true;  
setTodos(newTodos);  
};

Далее мы перемещаем часть удаления задач. Точно так же на этот раз мы используем индекс для объединения списка и удаляем задачу, индекс которой совпадает, а затем устанавливаем новые задачи.

const removeTodo = index => {   
 const newTodos = [...todos];   
 newTodos.splice (индекс, 1);   
 установитьTodos(новыеTodos);   
 };

Затем мы завершаем функцию приложения, возвращая рендеринг JSX, который будет отображаться на веб-сайте. Мы используем компонент *FormTodo* , который мы определим позже. Он принимает addTodo в качестве параметра.

Затем мы отображаем все задачи с помощью оператора карты. Для каждой задачи мы будем передавать ее компоненту *Todo* (определим позже). Мы отправляем функции index, todo, *markTodo* и *removeTodo* .

return (   
 <div className="app">   
 <div className="container">   
 <h1 className="text-center mb-4">Список задач</h1>   
 <FormTodo addTodo={addTodo} />   
 <div>   
 { todos.map((todo, index) => (   
 <Card>   
 <Card.Body>   
 <Todo   
 key={index}   
 index={index}   
 todo={todo}   
 markTodo={markTodo}   
 removeTodo={removeTodo}   
 />   
 < /Card.Body>   
 </Card>   
 ))}   
 </div>  
 </див>   
 </див>   
 );

Теперь мы определим компонент *Todo .*Он принимает четыре параметра, которые мы передали ранее при вызове компонента *Todo .*

Мы возвращаем какой-то JSX, который будет показывать каждый Todo. Также будут показаны две кнопки для отметки задач как выполненных и удаления задач соответственно.

function Todo({ todo, index, markTodo, removeTodo }) {   
 return (   
 <div   
 className="todo"   
   
 >   
 <span style={{ textDecoration: todo.isDone ? "line-through" : "" }}>{todo. text}</span>   
 <div>   
 <Button variant="outline-success" onClick={() => markTodo(index)}>✓</Button>{' '}   
 <Button variant="outline-danger" onClick ={() => removeTodo(index)}>✕</Button>   
 </div>   
 </div>   
 );   
}

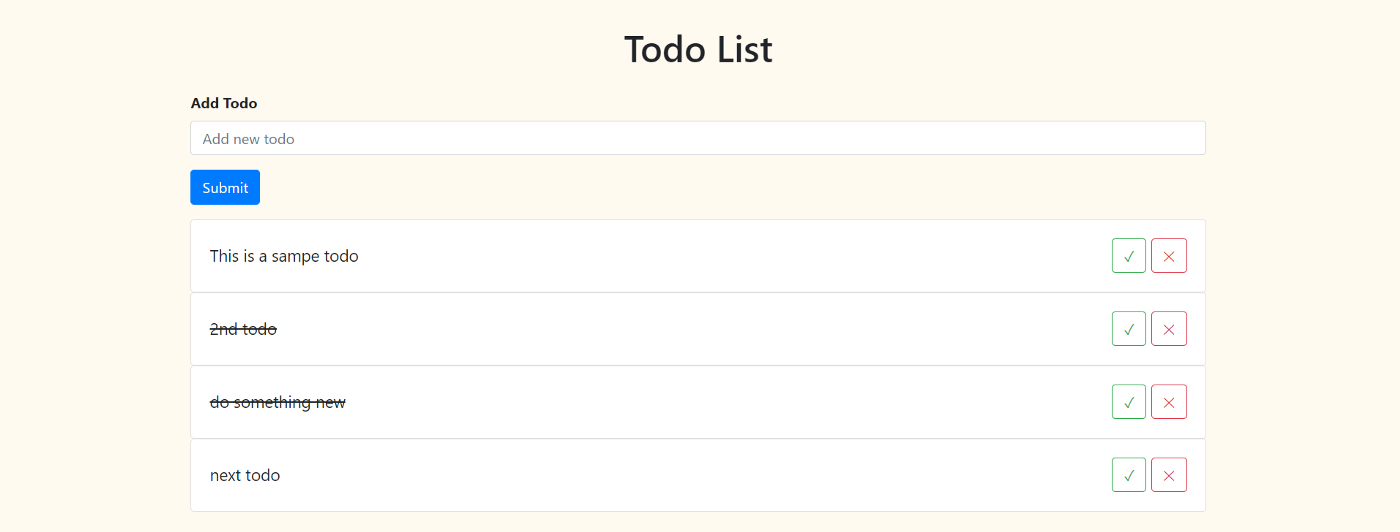
Далее мы должны определить функцию *FormTodo* . Он принимает *addTodo* в качестве параметра. Он обрабатывает отправку нового Todo. Если значение не пустое, то мы вызываем функцию *addTodo* для этого текста задачи, а затем снова устанавливаем пустое значение формы.

Мы возвращаем форму, которая принимает задачу и имеет кнопку «Отправить» для отправки задач. Нажатие на кнопку отправки добавит задачу в список задач.

function FormTodo({ addTodo }) {  
const [value, setValue] = React.useState("");  
  
const handleSubmit = e => {  
e.preventDefault();  
if (!value) return;  
addTodo(value);  
setValue("");  
};  
  
return (  
<Form onSubmit={handleSubmit}>   
<Form.Group>  
<Form.Label><b>Add Todo</b></Form.Label>  
<Form.Control type="text" className="input" value={value} onChange={e => setValue(e.target.value)} placeholder="Add new todo" />  
</Form.Group>  
<Button variant="primary mb-3" type="submit">  
Submit  
</Button>  
</Form>  
);  
}

Теперь давайте посмотрим на полный файл App.js:

Таким образом, мы создали наше простое приложение Todo. Вы можете попробовать этот простой React Todo, который мы создали [на этом веб-сайте, размещенном на страницах Github](https://shubham1710.github.io/React-Todo/) .



Список задач с React

Вы можете получить доступ к [репозиторию GitHub этой статьи](https://github.com/shubham1710/React-Todo) и увидеть полную структуру пакета и код.

Еще несколько статей для чтения после этой: