

Создайте приложение CRUD Todo с помощью Django и React / Redux

Учебник для начинающих



Коджи Мочизуки

29 июня 2019 г. · 10 минут чтения ★



Фото Анете Лусиня на Unsplash

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

Task

Add

<div><div></div><div>buy milk</div><div>06/12/2019 04:22:16</div></div>	<div>Delete</div>
<div><div></div><div>wash the car</div><div>06/12/2019 04:25:46</div></div>	<div>Delete</div>
<div><div></div><div>pay bills</div><div>06/12/2019 04:26:53</div></div>	<div>Delete</div>
<div><div></div><div>tidy my room</div><div>06/12/2019 04:27:59</div></div>	<div>Delete</div>
<div><div></div><div>call John</div><div>06/12/2019 04:30:28</div></div>	<div>Delete</div>

CRUD Todo App

Содержание

- Настройка Django
- Настройка React
- Получение данных из API и отображение списка
- Создание формы и добавление нового Todo
- Создание заголовка
- Удаление Todos
- Редактирование Todos

. . .

Настройка Django

Создание виртуальной среды с Django

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

Давайте создадим виртуальную среду с помощью этой команды:

```
$ pipenv --python 3
```

Если у вас еще не установлен Pipenv, установите его, выполнив следующую команду:

```
$ pip install pipenv
```

Мы установим нужные нам пакеты:

```
$ pipenv install django djangorestframework
```

Создание нового проекта и некоторых приложений

В этом уроке мы создадим проект под названием «todocrud». Мы можем опустить дополнительную папку, которая создается автоматически, добавив точку в конец команды и выполнив:


```
$ django-admin startproject todocrud.
```


Далее мы создадим два приложения. Один для внутреннего интерфейса, другой для внешнего интерфейса:

```
$ python manage.py startapp todos  
$ python manage.py startapp внешний интерфейс
```

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

 Зарегистрируйтесь
с помощью Google

 Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```
5     'todos.apps.TodosConfig' ,    # добавлено
6     'rest_framework' ,    # добавлено
7     'django.contrib.admin' ,
8     'django.contrib.auth' ,
9     'django.contrib.contenttypes' ,
10    'django.contrib.sessions' ,
11    'django.contrib.messages' ,
12    'django.contrib.staticfiles' ,
13 ]
14
15 REST_FRAMEWORK = {    # добавлено
16     'DEFAULT_PERMISSION_CLASSES' : [
17         'Rest_framework.permissions.AllowAny'
18     ],
19     «DATETIME_FORMAT» : «% m /% d /% Y% H:% M:% S» ,
20 }
```

settings.py, размещенный на ❤️ на GitHub

[просмотреть raw](#)

Мы можем указать формат вывода даты и времени, включив его DATETIME_FORMAT в конфигурационный словарь с именем REST_FRAMEWORK .

Давайте применим миграцию и запустим сервер разработки:

```
$ python manage.py migrate
$ python manage.py runserver
```

Посетите <http://127.0.0.1:8000/> с вашим браузером. Если вы видите страницу взлетающей ракеты, это сработало хорошо!

Написание бэкэнд-модулей

Сначала мы создадим простую модель. Откройте `models.py` файл и напишите следующий код:

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```
10     def __str__ ( самостоятельно ) :  
11         вернуть себя . задача
```

models.py с ❤ на GitHub

[просмотреть raw](#)

Далее мы создадим простой API на основе модели с использованием инфраструктуры REST. Давайте создадим новую папку с именем `api` и создавать новые файлы `__init__.py`, `serializers.py`, `views.py` и `urls.py` в нем:

```
todos /  
  api /  
    __init__.py  
    serializers.py  
    urls.py  
    views.py
```

Поскольку `api` это модуль, нам нужно включить `__init__.py` файл.

Давайте определим представление API в `serializers.py` файле:

```
1  # todos / api / serializers.py  
2  
3  из rest_framework импортировать сериализаторы  
4  
5  из задач . импорт моделей Todo  
6  
7  
8  Класс TodoSerializer ( сериализаторы . ModelSerializer ) :  
9      класс Meta :  
10         модель = Todo  
11         fields = '__all__'
```

serializers.py размещенного с помощью ❤ GitHub


[просмотреть сырой](#)


ModelSerializer Класс будет создавать поля , которые соответствуют полям

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

 Зарегистрируйтесь
с помощью Google

 Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```

5  от . импорт сериализаторов  TodoSerializer
6  из  задач . импорт моделей  Todo
7
8
9  Класс  TodoViewSet ( viewsets . ModelViewSet ):
10      queryset = Todo . объекты . все ()
11      serializer_class = TodoSerializer

```

views.py с hos GitHub

[просмотреть raw](#)

Наконец, мы напишем конфигурацию URL, используя Routers :

```

1  # todos / api / urls.py
2
3  из rest_framework импортировать маршрутизаторы
4
5  от . импорт просмотров  TodoViewSet
6
7  маршрутизатор = маршрутизаторы . DefaultRouter ()
8  маршрутизатор . зарегистрироваться ( « задачи » , TodoViewSet , « задачи » )
9  # router.register ('<Префикс URL-адреса>', <Класс набора настроек>, '<Имя URL-адреса>')
10
11  urlpatterns = маршрутизатор . URLs

```

urls.py с with на GitHub

[просмотреть raw](#)

Мы используем три аргумента для `register()` метода, но третий аргумент не требуется.

Написание модулей внешнего интерфейса

В веб-интерфейсе все, что нам нужно сделать, это написать простые представления и шаблоны URL.

Откройте `frontend/views.py` файл и создайте два представления:

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```
10     возвратный рендер ( запрос , 'frontend / index.html' )
11
12
13     Класс TodoDetailView ( DetailView ):
14         модель = Todo
15         template_name = 'frontend / index.html'
```

views.py с hos GitHub

[просмотреть raw](#)

Мы создадим frontend/index.html файл позже. Не беспокойся об этом сейчас.

Добавьте новый urls.py файл в тот же каталог и создайте URL-адрес conf:

```
1  # frontend / urls.py
2
3  из Django . URL путь импорта
4
5  от . индекс импорта просмотров , TodoDetailView
6
7  urlpatterns = [
8      путь ( '' , индекс ),
9      путь ( 'edit / <int: pk>' , TodoDetailView . as_view ()),
10     путь ( 'delete / <int: pk>' , TodoDetailView . as_view ()),
11 ]
```

urls.py с with на GitHub

[просмотреть raw](#)

Как вы можете видеть выше, index представление предназначено для индексной страницы и TodoDetailView вызывается, когда мы запрашиваем конкретный объект.

Подключите URL

Мы включим URL-адреса веб-интерфейса и внутреннего интерфейса в URLconf проекта:

```
1  # todocrud / urls.py
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[ВОЙТИ В СИСТЕМУ](#)

Хотя путь к admin сайту Django оставлен, мы не собираемся использовать его в этом руководстве.

В качестве заключительной части этой главы мы создадим новый файл миграции и применим изменения к нашим базам данных, выполнив следующие команды:

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Настройка React

Создание каталогов

Прежде всего, давайте создадим все каталоги, которые нам нужны:

```
$ mkdir -p ./frontend/src/ {enjcomponents,actions,reducers}
$ mkdir -p ./frontend/ {enjstatic,templates}</frontend
```

Приведенная выше команда должна создать каталоги следующим образом:

```
Фронтенд /
  src /
    Actions /
    Компоненты /
    Редукторы /
    Статические /
    Фронтенд /
    Шаблоны /
    Фронтенд /
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)


```
$ npm init -y
```

Для того чтобы использовать `npm`, Node.js должен быть установлен.

Затем давайте установим все пакеты, которые мы используем с `npm` командой:

```
$ npm i -D webpack webpack-cli
$ npm i -D babel-loader @babel/core @babel/preset-env @babel/preset-react @babel/plugin-proposal-class-properties

$ npm я реагирую реагирую Реакция-роутер-дом
$ npm я редукция Реакция-редукс редукция-редук-редукс-devtools-extension
$ npm я редукция-форма
$ npm i-аксиос
$ npm я
```

Создание конфигурационных файлов

Добавьте файл с именем `.babelrc` в корневой каталог и настройте **Babel** :

```
1 // .babelrc
2
3 {
4   "presets": [
5     [
6       "@babel/preset-env",
7       {
8         "targets": {
9           "node": "current"
10        },
11        "useBuiltIns": "usage",
12        "corejs": 3
13      ]
14    ]
15  }
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

Вторично, добавьте файл с именем `webpack.config.js` в тот же каталог и напишите конфигурацию для `webpack` :

```
1  // webpack.config.js
2
3  модуль . экспорт = {
4    модуль : {
5      правила : [
6        {
7          тест : / \. ( js | jsx ) $ / ,
8          исключить : / node_modules / ,
9          использовать : {
10             загрузчик : 'babel-loader'
11           }
12         }
13       ]
14     }
15   } ;
```

`webpack.config.js`, размещенный на ❤️ от GitHub

[просмотреть raw](#)

Кроме того, нам нужно переписать "scripts" свойство `package.json` файла:

```
1  // package.json
2
3  {
4    // ...
5    "сценарии": {
6      "dev": "webpack --mode development --watch ./frontend/src/index.js --output ./front
7      "build": "webpack - mode mode ./frontend/src/index.js --output ./frontend/static/fr
8    },
9    // ...
10 }
```

`package.json`, размещенного на ❤️ в GitHub

[просмотр raw](#)

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

Мы создадим файл с именем, `index.js` который будет вызываться первым при запуске приложения React:

```
1 // frontend / src / index.js
2
3 import приложение из «./components/App» ;
```

`index.js`, размещенный на ❤️ на GitHub

[просмотреть raw](#)

Далее мы создадим файл с именем, `App.js` который является родительским компонентом:

```
1 // внешний интерфейс / src / components / App.js
2
3 import React , { Component } из 'response' ;
4 import ReactDOM из ' response -dom' ;
5
6 Класс App расширяет Компонент {
7   render ( ) {
8     возврат (
9       < div >
10         < h1 > ToDoCRUD < / h1 >
11       < / div >
12     ) ;
13   }
14 }
15
16 ReactDOM . рендеринга ( < приложение / > , документ . querySelector ( '#app' ) ) ;
```

`App.js`, размещенный в with на GitHub

[просмотреть raw](#)


Наконец, мы создадим файл шаблона с именем `index.html` , указанным в `views.py` файле:


```
1 <! - templates / frontend / index.html ->
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

 Зарегистрируйтесь
с помощью Google

 Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```
11 < link rel = 'stylesheet' href = 'https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.3.1/semantic.min.css' />
12 < title > ToDoCRUD </ title >
13 </ head >
14
15 < тело >
16 < div id = ' app ' > </ div >
17
18 {% load static%}
19 < script src = " {% static 'frontend / main.js'%} " > </ script >
20 </ body >
21
22 </ html >
```

index.html, размещенный в with на GitHub

[просмотреть raw](#)

В этом уроке мы будем использовать Semantic UI в качестве фреймворка CSS.

Поместите оболочку для рендеринга компонента App и связанного скрипта в `<body>` тег.

Проверка дисплея

Посмотрим, правильно ли он отображается.

Откройте другой терминал и запустите скрипт:

```
$ npm run dev
```

main.js Файл должен быть создан в static/frontend каталоге.

Затем запустите сервер разработки и зайдите на сайт <http://127.0.0.1:8000/> :

```
$ python manage.py runserver
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

Настало время использовать **Redux** . Мы создадим **Действия** , **Редукторы** и **Магазин** .

ДЕЙСТВИЯ

Давайте определим все **свойства типа** заранее. Добавьте новый файл с именем `types.js` в `src/actions` каталог:

```
1 // actions / types.js
2
3 export const GET_TODOS = 'GET_TODOS' ;
4 export const GET_TODO = 'GET_TODO' ;
5 export const ADD_TODO = 'ADD_TODO' ;
6 export const DELETE_TODO = 'DELETE_TODO' ;
7 export const EDIT_TODO = 'EDIT_TODO' ;
```

`types.js`, размещенный на ❤️ на GitHub

[просмотреть raw](#)

Чтобы создать действия, нам нужно определить **создателей действий** .
Добавьте новый файл с именем `todos.js` в `src/actions` каталог:

```
1 // actions / todos.js
2
3 импорт axios из 'axios' ;
4 импорт { GET_TODOS } из './types' ;
5
6 // ПОЛУЧИТЬ ТОДОС
7 export const getTodos = ( ) => асинхронная отправка => {
8   const res = ждать axios . get ( '/ api / todos /' ) ;
9   отправка ( {
10     тип : GET_TODOS ,
11     полезная нагрузка : рез . данные
12   } ) ;
13 } ;
```


`todos.js`, размещенный на ❤️ на GitHub


[просмотреть raw](#)

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

 Зарегистрируйтесь
с помощью Google

 Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[ВОЙТИ В СИСТЕМУ](#)

```

1 // redurs / todos.js
2
3 импорт _ из 'lodash' ;
4 импортировать { GET_TODOS } из '../actions/types' ;
5
6 экспорт по умолчанию ( состояние = { } , действие ) => {
7   switch ( action . type ) {
8     case GET_TODOS :
9       возврат {
10         ... состояние ,
11         ... _ . mapKeys ( action . payload , 'id' )
12       } ;
13     по умолчанию :
14       возвратное состояние ;
15   }
16 } ;

```

todos.js, размещенный на ❤️ на GitHub

[просмотреть raw](#)

Lodash - это библиотека утилит JavaScript. Это не является обязательным требованием, но оно может сократить время разработки и уменьшить размер кода.

Давайте создадим родительский редуктор, чтобы собрать вместе каждый дочерний редуктор `combineReducers()`. Добавьте новый файл с именем `index.js` в `src/reducers` каталог:

```

1 // redurs / index.js
2
3 импорт { combReducers } из 'redux' ;
4 импорта { редуктор , как formReducer } от «переводы-формы» ;
5 импорт Todos из »./todos' ;
6
7 экспорт по умолчанию combineReducers ( {
8   form : formReducer ,
9   Todos
10 } ) ;

```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

Магазин является объектом для удержания **состояния** нашего приложения. Кроме того, мы будем использовать рекомендованное промежуточное ПО **Redux Thunk** для написания асинхронной логики, которая взаимодействует с хранилищем. Давайте создадим новый файл с именем `store.js` в `src` каталоге:

```
1 // frontend / src / store.js
2
3 import { createStore , applyMiddleware } из 'redux' ;
4 import { composeWithDevTools } из 'redux-devtools-extension' ;
5 импортировать reduxThunk из 'redux-thunk' ;
6 импортировать rootReducer из './reducers' ;
7
8 const store = createStore (
9   rootReducer ,
10   composeWithDevTools ( applyMiddleware ( reduxThunk ) )
11 ) ;
12
13 экспорт магазина по умолчанию ;
```

store.js, размещенный на ❤️ GitHub

[просмотреть raw](#)

Использование **Redux DevTools** **не** является обязательным, но оно очень полезно, потому что оно визуализирует изменения состояния Redux. Я опущу, как использовать это здесь, но это настоятельно рекомендуется.

Компоненты

Сначала создайте новую папку с именем `todos` в `components` каталоге. А затем добавьте новый файл с именем `TodoList.js` в папку, которую мы создали:

```
1 // компоненты / задачи / TodoList.js
2
3 импортировать React , { Component } из 'react' ;
4 импорта { подключения } от 'react-redux' ;
5 import { getTodos } из '../actions/todos' ;
6
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)


```

16     < div className = 'item' key = { todo . id } >
17       < i className = 'большой значок календаря, выровненный по среднему значку'
18       < div className = 'content' >
19         < Имя класса = 'заголовок' > { TODO . задача } < / a >
20         < div className = 'description' > { todo . создал_ат } < / div >
21       < / div >
22     < / div >
23   ) ) }
24 < / div >
25 ) ;
26 }
27 }
28
29 const mapStateToProps = state => ( {
30   Задачи : Объект . значения ( штат . задачи )
31 } ) ;
32
33 экспорт по умолчанию подключить (
34   mapStateToProps ,
35   { getTodos }
36 ) ( TodoList ) ;

```

Мы используем Semantic UI list для оформления списка.

`connect()` Функция связывает этот компонент в магазин. Он принимает `mapStateToProps` в качестве первого аргумента, `Action Creators` в качестве второго аргумента. Мы сможем использовать состояние магазина в качестве **реквизита**, указав `mapStateToProps`.

Мы создадим новый файл с именем `Dashboard.js` в том же каталоге. Это просто контейнер `TodoList` и форма, которую мы создадим в следующей главе:


```


1 // компоненты / задачи / Dashboard.js
2

```

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

 Зарегистрируйтесь
с помощью Google

 Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```
12      < / uliv >
13    ) ;
14  }
15 }
16
17  панель экспорта по умолчанию ;
```

Dashboard.js, размещенного в ❤️ на GitHub

[просмотр необработанного](#)

Откройте App.js файл и обновите его следующим образом:

```
1  // компоненты / App.js
2
3  импортировать Dashboard из './todos/Dashboard' ; // добавлено
4
5  импорта { Провайдер } от 'реагируют-Redux' ; // добавлено
6  импорт магазина из "../store" ; // добавлено
7
8  Класс App расширяет Компонент {
9    render ( ) {
10      возврат (
11        < Provider store = { store } >
12          < Панель инструментов / >
13        < / Provider >
14      ) ;
15    }
16  }
```

App.js, размещенный в with на GitHub

[просмотреть raw](#)

Это Provider делает хранилище доступным для компонента, вложенного в него.

Проверка дисплея

Сначала посетите <http://127.0.0.1:8000/api/todos/> и создайте несколько объектов. А затем посетите <http://127.0.0.1:8000/>.

Вы должны увидеть простой список объектов, которые вы создали. Это

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```

1 // actions / todos.js
2
3 import { reset } из 'redux-формы' ; // добавлено
4 импортировать { GET_TODOS , ADD_TODO } из './types' ; // добавлено ADD_TODO
5
6 // ДОБАВИТЬ TODO
7 экспорт const addTodo = formValues => асинхронная отправка => {
8   const res = ждать axios . post ( '/ api / todos /' , { ... formValues } ) ;
9   отправка ( {
10     тип : ADD_TODO ,
11     полезная нагрузка : рез . данные
12   } ) ;
13   рассылка ( сброс ( 'todoForm' ) ) ;
14 } ;

```

todos.js, размещенный на ❤️ на GitHub

[просмотреть raw](#)

Отправка `reset('formName')` очищает нашу форму после успешной отправки . Мы будем указывать имя формы позже в компоненте `Form`.

Переходники

Откройте `reducers/todos.js` файл и добавьте новое действие в редуктор:

```

1 // redurs / todos.js
2
3 импортировать { GET_TODOS , ADD_TODO } из '../actions/types' ; // добавлено ADD_
4
5 экспорт по умолчанию ( состояние = { } , действие ) => {
6   switch ( action . type ) {
7     // ...
8     case ADD_TODO : // добавлено
9       возврат {
10         ... состояние ,
11         [ действие . полезная нагрузка . id ] : действие . полезная нагрузка
12       } ;
13     // ...
14   }

```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook


Уже есть аккаунт?
[ВОЙТИ В СИСТЕМУ](#)


редактирования. Создайте новый файл с именем `TodoForm.js` в `components/todos` каталоге:

```
1  // компоненты / задачи / TodoForm.js
2
3  импортировать React , { Component } из 'response' ;
4  import { Field , reduxForm } из 'redux-формы' ;
5
6  Класс TodoForm расширяет Компонент {
7    renderField = ( { input , label , meta : { touch , error } } ) => {
8      возврат (
9        < DIV имя класса = { `поле $ { прикоснулся && ошибку ? 'error' : '' } ` } >
10         < label > { label } < / label >
11         < input { ... input } autoComplete = 'off' / >
12         { тронут && ошибка && (
13           < span className = 'интерфейс, указывающий на красную базовую метку' > { err
14         ) }
15       < / div >
16     ) ;
17   } ;
18
19   onSubmit = formValues => {
20     это . реквизит . onSubmit ( formValues ) ;
21   } ;
22
23   render ( ) {
24     возврат (
25       < div className = 'UI сегмент' >
26         < форма
27           onSubmit = { это . реквизит . handleSubmit ( this . onSubmit ) }
28           className = 'Ошибка формы пользовательского интерфейса'
29         >
30           < Имя поля = компонент 'задача' = { это . renderField } label = 'Task' / >
31           < button className = 'основная кнопка пользовательского интерфейса' > Добави
32         < / form >
33       < / div >
34     ) ;
35   } ;
```

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

 Зарегистрируйтесь
с помощью Google

 Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```
45     возвращать  ошибки ;
46   } ;
47
48   экспорт по умолчанию reduxForm ( {
49     форма : 'todoForm' ,
50     touchOnBlur : ложь ,
51     Validate
52   } ) ( TodoForm ) ;
```

Учебник будет длинным, поэтому я не буду использовать **Redux Form** . Чтобы понять, как работает форма Redux, стоит попробовать настроить форму, ссылаясь на документацию .

'todoForm' это название этой формы. Это то, что мы использовали в действии создателя `addTodo` .

Когда мы щелкаем текстовое поле, а затем удаляем фокус, отображается ошибка проверки, поэтому укажите, `touchOnBlur: false` чтобы отключить ее.

Далее, давайте создадим компонент для добавления новых задач. Создайте новый файл с именем `TodoCreate.js` в `components/todos` каталоге:

```
1  // компоненты / задачи / TodoCreate.js
2
3  импортировать React , { Component } из 'response' ;
4  импорта { подключения } от 'реагируют-Redux' ;
5  import { addTodo } из '../actions/todos' ;
6  импортировать TodoForm из './TodoForm' ;
7
8  Класс TodoCreate расширяет Компонент {
9    onSubmit = formValues => {
10      это . реквизит . addTodo ( formValues ) ;
11    } ;
12
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```
22  экспорт по умолчанию подключить (
23    ноль ,
24    { addTodo }
25  ) ( TodoCreate ) ;
```

TodoCreate.js, размещенного в ❤️ на GitHub

[просмотр сырого](#)

Все, что нам нужно сделать, это сделать `TodoForm`. Установив `destroyOnUnmount` для `false`, мы можем отключить, чтобы форма Redux автоматически уничтожала состояние формы в хранилище Redux, когда компонент отключен. Это для отображения состояния формы в форме редактирования.

Если нам не нужно указывать `mapStateToProps` функцию, установите `null` в `connect()`.

Давайте посмотрим и протестируем форму. Откройте `Dashboard.js` файл и обновите его следующим образом:

```
1  // компоненты / задачи / Dashboard.js
2
3  импортировать TodoCreate из './TodoCreate' ; // добавлено
4
5  Панель инструментов класса расширяет Компонент {
6    render ( ) {
7      возврат (
8        < div className = 'ui container' >
9          < TodoCreate / > // добавлено
10         < TodoList / >
11        < / div >
12      ) ;
13    }
14  }
15
16  панель экспорта по умолчанию ;
```

Dashboard.js, размещенного в ❤️ на GitHub

[просмотр необработанного](#)

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```

2
3  импортировать React , { Component } из 'response' ;
4
5  Заголовок класса расширяет Компонент {
6    render ( ) {
7      возврат (
8        < div className = 'UI перевернутое меню' style = { { borderRadius : '0' } } >
9          < Имя класса = 'элемента заголовка' > TodoCRUD < / >
10         < Имя класса = 'пункт' > Главная < / >
11       < / div >
12     ) ;
13   }
14 }
15
16  Заголовок экспорта по умолчанию ;

```

Header.is, размещенный на with GitHub

[просмотреть raw](#)

Откройте App.js файл и вложите Header компонент:

```

1  // компоненты / App.js
2
3  импортировать заголовок из './layout/Header' ; // добавлено
4
5  Класс App расширяет Компонент {
6    render ( ) {
7      возврат (
8        < Provider store = { store } >
9          < Header / > // добавлено
10         < Панель инструментов / >
11       < / Provider >
12     ) ;
13   }
14 }

```

App.js, размещенный в with на GitHub

[просмотреть raw](#)

В этом уроке заголовок - это просто украшение

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)


```
1 // frontend / src / history.js
2
3 import { createBrowserHistory } из 'history' ;
4
5 экспорт по умолчанию createBrowserHistory ( ) ;
```

history.js, размещенный на ❤️ GitHub

[просмотреть raw](#)

действия

Откройте `actions/todos.js` файл и добавьте двух новых создателей действий:

```
1 // actions / todos.js
2
3 импортировать историю из '../history' ; // добавлено
4 импортировать { GET_TODOS , GET_TODO , ADD_TODO , DELETE_TODO } из './types' ;
5
6 // ПОЛУЧИТЬ ТОДО
7 экспорт const getTodo = id => асинхронная отправка => { // добавлено
8   const res = ждать axios . get ( `/ api / todos / $ { id } /` ) ;
9   отправка ( {
10     тип : GET_TODO ,
11     полезная нагрузка : рез . данные
12   } ) ;
13 } ;
14
15 // УДАЛИТЬ ТОДО
16 экспорт const deleteTodo = id => асинхронная отправка => { // добавлено
17   жду аксиос . delete ( `/ api / todos / $ { id } /` ) ;
18   отправка ( {
19     тип : DELETE_TODO ,
20     полезная нагрузка : идентификатор
21   } ) ;
22   история . нажать ( '/' ) ;
```

todos.js, размещенный на ❤️ на GitHub

[просмотреть raw](#)

Мы создали, `getTodo` чтобы получить конкретный объект и `deleteTodo` удалить

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[ВОЙТИ В СИСТЕМУ](#)

Откройте `reducers/todos.js` файл и добавьте действия в редуктор:

```
1 // reducers / todos.js
2
3 импорт _ из 'lodash' ; // добавлено
4 импортировать { GET_TODOS , GET_TODO , ADD_TODO , DELETE_TODO } из '../actions/'
5
6 экспорт по умолчанию ( состояние = { } , действие ) => {
7   switch ( action . type ) {
8     // ...
9     case GET_TODO : // добавлено
10    case ADD_TODO :
11      возврат {
12        ... состояние ,
13        [ действие . полезная нагрузка . id ] : действие . полезная нагрузка
14      } ;
15    case DELETE_TODO : // добавлено
16      вернуть _ . опустить ( состояние , действие . полезная нагрузка ) ;
17    // ...
18  }
19 } ;
```

`todos.js`, размещенный на ❤️ на GitHub

[просмотреть raw](#)

`GET_TODO` Действие такое же , как `ADD_TODO` действие, так что нам нужно только установить `case` . Для `DELETE_TODO` действия снова используйте `Lodash` в качестве ярлыка.

Компоненты

Давайте создадим модальное окно, которое я только что упомянул. У нас это будет выглядеть так:



Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[ВОЙТИ В СИСТЕМУ](#)

Are you sure you want to delete the task: buy milk

Delete

Cancel

Модальное окно для подтверждения удаления

Создайте новый файл с именем `Modal.js` в `components/layout` каталоге и напишите следующее:

```
1  // компоненты / макет / Modal.js
2
3  импорт React из 'реакции' ;
4  импортировать ReactDOM из 'response -dom' ;
5
6  const Modal = props => {
7    вернуть ReactDOM . createPortal (
8      < div onClick = { реквизит . onDismiss } className = 'ui active dimmer ' >
9        < div onClick = { e => e . stopPropagation ( ) } className = 'ui active modal' >
10          < div className = 'header' > { реквизит . название } < / div >
11          < div className = 'content' > { реквизит . содержание } < / div >
12          < div className = 'actions' > { реквизит . действия } < / div >
13        < / div >
14      < / div > ,
15      документ . querySelector ( '#modal' )
16    ) ;
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[ВОЙТИ В СИСТЕМУ](#)

является визуализируемым дочерним элементом, а второй аргумент является элементом DOM для визуализации.

А затем, откройте `index.html` файл и добавьте контейнер для модальных внутри `<body>` тега:

```
1  <! - templates / frontend / index.html ->
2
3  < тело >
4    < div id = ' app ' > </ div >
5    < div id = " modal " > </ div >
6
7    {% load static%}
8    < script src = " {% static 'frontend / main.js'%} " > </ script >
9  </ body >
```

`index.html`, размещенный в with на GitHub

[просмотреть raw](#)

Далее мы создадим новый компонент `TodoDelete.js` в `components/todos` каталоге:

```
1  // компоненты / задачи / TodoDelete.js
2
3  импортировать React , { Component , Fragment } из 'реакции' ;
4  импорта { подключения } от 'реагируют-Redux' ;
5  импорт { Ссылка } от «реагировать-маршрутизатор-дом» ;
6  импорт Модальные из «../layout/Modal» ;
7  импортировать историю из '../../history' ;
8  import { getTodo , deleteTodo } из '../../actions/todos' ;
9
10 класс TodoDelete расширяет Компонент {
11   componentDidMount ( ) {
12     это . реквизит . getTodo ( это . реквизит . совпадение . params . id ) ;
13   }
14
15   renderContent ( ) {
16     if ( ! this . props . todo ) {
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```

26     кнопка <
27       onClick = { ( ) => это . реквизит . deleteTodo ( id ) }
28       className = 'UI отрицательная кнопка'
29     >
30       удалять
31     < / button >
32     < Link to = '/' className = 'UI Button' >
33       Отмена
34     < / Link >
35   < / Fragment >
36 ) ;
37 }
38
39 render ( ) {
40   возврат (
41     < Модальный
42       title = 'Удалить Todo'
43       содержание = { это . renderContent ( ) }
44       действия = { это . renderActions ( ) }
45       onDismiss = { ( ) => история . нажать ( '/' ) }
46     / >
47   ) ;
48 }
49 }
50
51 const mapStateToProps = ( state , ownProps ) => ( {
52   todo : состояние . todos [ ownProps . совпадают . парам . id ]
53 } ) ;
54
55 экспорт по умолчанию подключиться (
56   mapStateToProps ,
57   { getTodo , deleteTodo }
58 ) ( TodoDelete ) ;

```

Код немного длинный, но это не так сложно. Определите вспомогательные функции, которые отображают содержимое и кнопки действий в модальном окне. Затем передайте их в качестве реквизита модальному компоненту.

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)


```


1  // компоненты / задачи / TodoList.js
2
3  импорт { Ссылка } от «реагировать-маршрутизатор-дом» ; // добавлено
4  импорт { getTodos , deleteTodo } из '../..actions/todos' ; // добавлено delete
5
6  Класс TodoList расширяет Компонент {
7    // ...
8
9    render ( ) {
10      возврат (
11        < div className = 'расслабленный разделенный список' style = { { marginTop : '
12          { это . реквизит . Todos . карта ( todo => (
13            < div className = 'item' key = { todo . id } >
14              < div className = 'содержимое с плавающей точкой ' > // добавлено
15                < Ссылка
16                  to = { `/ delete / $ { todo . id } ` }
17                  className = 'маленькая базовая кнопка с отрицательным пользовательским
18                >
19                  удалять
20                < / Link >
21              < / div >
22              < i className = 'большой значок календаря, выровненный по среднему значку'
23              < div className = 'content' >
24                < Имя класса = 'заголовок' > { TODO . задача } < / a >
25                < div className = 'description' > { todo . создал_ат } < / div >
26              < / div >
27            < / div >
28          ) ) }
29        < / div >
30      ) ;
31    }
32  }
33
34  // ...
35
36  экспорт по умолчанию подключиться (
37    mapStateToProps ,
38    { getTodos , deleteTodo } // добавлено deleteTodo
39  ) ( TodoList ) .

```

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

 Зарегистрируйтесь
с помощью Google

 Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[ВОЙТИ В СИСТЕМУ](#)

```

3  импорта { маршрутизатор , маршрут , коммутатор } от 'реагируют-маршрутизатор-DOM
4
5  импортировать историю из '../history' ; // добавлено
6  импортировать TodoDelete из './todos/TodoDelete' ; // добавлено
7
8  Класс App расширяет Компонент {
9    render ( ) {
10      возврат (
11        < Provider store = { store } >
12          < История маршрутизатора = { история } >
13            < Заголовок / >
14            < Switch >
15              < Route точный путь = '/' component = { Панель инструментов } / >
16              < Route точный путь = '/ delete /: id' component = { TodoDelete } / >
17            < / Switch >
18          < / Router >
19        < / Provider >
20      ) ;
21    }
22  }

```

App is размещенный в with на GitHub

[просмотреть raw](#)

Причина использования Router вместо **BrowserRouter** объясняется в документе **REACT TRAINING** следующим образом:


Наиболее распространенный вариант использования низкоуровневого <Router> - это синхронизация пользовательской истории с библиотекой управления состояниями, такой как Redux или Mobx. Обратите внимание, что это не требуется для использования библиотек управления состояниями вместе с React Router, это только для глубокой интеграции.

exact Параметр , указанный в Route возвратах маршрут только в случае , если path точно соответствует текущему URL.

На этом мы завершаем эту главу. Попробуйте удалить некоторые объекты и

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

 Зарегистрируйтесь с помощью Google

 Зарегистрируйтесь через Facebook

Уже есть аккаунт?
[ВОЙТИ В СИСТЕМУ](#)

Откройте `actions/todos.js` файл и добавьте создателя нового действия:

```
1 // actions / todos.js
2
3 импортировать { GET_TODOS , GET_TODO , ADD_TODO , DELETE_TODO , EDIT_TODO } из
4
5 // РЕДАКТИРОВАТЬ TODO
6 экспорт const editTodo = ( id , formValues ) => асинхронная отправка => {
7   const res = ждать axios . patch ( `/ api / todos / $ { id } /` , formValues ) ;
8   отправка ( {
9     тип : EDIT_TODO ,
10    полезная нагрузка : рез . данные
11  } ) ;
12  история . нажать ( '/' ) ;
13 } ;
```

`todos.js`, размещенный на ❤️ на GitHub

[просмотреть raw](#)

Переходники

Откройте `reducers/todos.js` файл и добавьте действие в редуктор:

```
1 // redurs / todos.js
2
3 import {
4   GET_TODOS ,
5   GET_TODO ,
6   ADD_TODO ,
7   DELETE_TODO ,
8   EDIT_TODO // добавлено
9 } из '../actions/types' ;
10
11 экспорт по умолчанию ( состояние = { } , действие ) => {
12   switch ( action . type ) {
13     // ...
14     case GET_TODO :
15     case ADD_TODO :
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

Компоненты

Создайте новый компонент `TodoEdit.js` в `components/todos` каталоге:

```
1  // компоненты / задачи / TodoEdit.js
2
3  импорт _ из 'lodash' ;
4  импортировать React , { Component } из 'response' ;
5  импорта { подключения } от 'реагируют-Redux' ;
6  импорт { getTodo , editTodo } из '../actions/todos' ;
7  импортировать TodoForm из './TodoForm' ;
8
9  Класс TodoEdit расширяет Компонент {
10   componentDidMount ( ) {
11     это . реквизит . getTodo ( это . реквизит . совпадение . params . id ) ;
12   }
13
14   onSubmit = formValues => {
15     это . реквизит . editTodo ( это . реквизит . match . params . id , formValues ) ;
16   } ;
17
18   render ( ) {
19     возврат (
20       < div className = 'ui container' >
21         < h2 style = { { marginTop : '2rem' } } > Изменить Todo < / h2 >
22         < TodoForm
23           initialValues = { _ . pick ( this . props . todo , 'task' ) }
24           enableReinitialize = { true }
25           onSubmit = { это . onSubmit }
26         / >
27       < / div >
28     ) ;
29   }
30 }
31
32 const mapStateToProps = ( state , ownProps ) => ( {
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.



Зарегистрируйтесь
с помощью Google



Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[ВОЙТИ В СИСТЕМУ](#)

Укажите объект в `initialValues`. Мы можем получить только ценность `task` использования `_.pick` функции **Lodash**. Кроме того, набор `enableReinitialize` для `true` того, чтобы мы можем также получить значение при перезагрузке страницы. Передайте эти необязательные свойства в `TodoForm`.

Откройте `TodoList.js` файл и обновите его `{todo.task}` следующим образом:

```
1 // компоненты / задачи / TodoList.js
2
3 <Ссылка на = { `/ edit / $ { todo . id } ` } className = 'header' >
4   { todo . задача }
5 < / Link >
```

`TodoList.js`, размещенного на ❤️ [GitHub](#)

[просмотр сырого](#)


Давайте добавим новый компонент в `App.js` файл:


```
1 // компоненты / App.js
2
3 импортировать TodoEdit из './todos/TodoEdit' ; // добавлено
4
5 Класс App расширяет Компонент {
6   render ( ) {
7     возврат (
8       < Provider store = { store } >
9         < История маршрутизатора = { история } >
10           < Заголовок / >
11           < Switch >
12             < Route точный путь = '/' component = { Панель инструментов } / >
13             < Route точный путь = '/ delete /: id' component = { TodoDelete } / >
14             < Route точный путь = '/ edit /: id' component = { TodoEdit } / > // доб
15           < / Switch >
16         < / Router >
17       < / Provider >
18     ) .
```

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

 Зарегистрируйтесь
с помощью Google

 Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[войти в систему](#)

```

1  // компоненты / задачи / TodoForm.js
2
3  Класс TodoForm расширяет Компонент {
4    // ...
5
6    render ( ) {
7      const btnText = ` ${ this . реквизит . initialValues ? «Обновить» : «Добавить»
8      возврат (
9        < div className = 'UI сегмент' >
10         < форма
11           onSubmit = { это . реквизит . handleSubmit ( this . onSubmit ) }
12           className = 'Ошибка формы пользовательского интерфейса'
13         >
14           < Имя поля = компонент 'задача' = { это . renderField } label = 'Task' / >
15           < button className = 'ui primary button' > { btnText } < / button > // обно
16         < / form >
17       < / div >
18     ) ;
19   }
20 }

```

TodoForm.js, размещенного на ❤️ GitHub

[просмотр сырого](#)

Теперь нажмите на любую задачу на странице индекса и попробуйте изменить:

TodoCRUD

Home

Edit Todo

Task

buy milk


Update

Форма редактирования

Читайте больше историй в этом месяце.



Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

 Зарегистрируйтесь
с помощью Google

 Зарегистрируйтесь
через Facebook

Уже есть аккаунт?
[ВОЙТИ В СИСТЕМУ](#)

Следующий шаг

<p>Реализовать аутентификацию пользователя в приложении Django & React с помощью Knox</p> <p>Добавьте аутентификацию на основе токенов с помощью Django-rest-knox в приложение, созданное с помощью Django и...</p> <p>medium.com</p>	
--	--

[программирование](#) [кодирование](#) [Джанго](#) [реагировать](#) [Redux](#)


[ОсправкеЮридическая](#)

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

▼

 Зарегистрируйтесь с помощью Google

 Зарегистрируйтесь через Facebook

Уже есть аккаунт?
[войти в систему](#)