Создайте приложение CRUD Todo с помощью Django и React / Redux

Учебник для начинающих

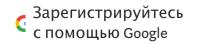




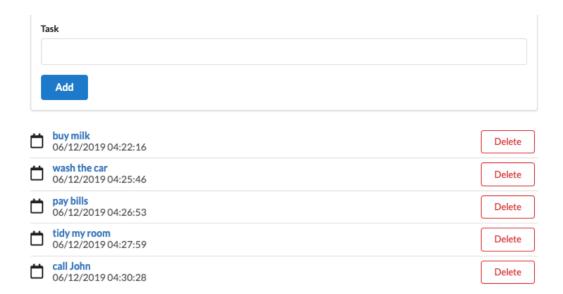
Фото Анете Лусиня на Unsplash

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.







CRUD Todo App

Содержание

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

. . .

Настройка Django

COSTISTINO BIANTIVATI HOM COOTILIC Dinony

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

Зарегистрируйтесь с помощью Google



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:

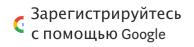
\$ django-admin startproject todocrud.

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.





```
'todos.apps.TodosConfig',
                                      # добавлено
         'rest_framework', # добавлено
         'django.contrib.admin',
         'django.contrib.auth',
         'django.contrib.contenttypes',
         'django.contrib.sessions',
10
         'django.contrib.messages',
11
         'django.contrib.staticfiles',
13
14
15
    REST_FRAMEWORK = {
                           # добавлено
         'DEFAULT PERMISSION CLASSES' : [
16
17
             'Rest_framework.permissions.AllowAny'
18
        ],
        «DATETIME FORMAT»: «% m /% d /% Y% H:% M:% S»,
19
                                                                             просмотреть raw
settings.py, размещенный на 💚 на GitHub
```

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
```

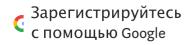
Посетите http://127.0.0.1:8000/ с вашим браузером. Если вы видите страницу взлетающей ракеты, это сработало хорошо!

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.





```
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 арі 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 арі это модуль, нам нужно включить __init__.py файл.

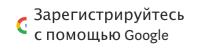
Давайте определим представление API в serializers.py файле:

```
# todos / api / serializers.py
2
3
    из rest framework импортировать сериализаторы
4
5
    из задач импорт моделей Todo
6
7
    Класс TodoSerializer ( сериализаторы . ModelSerializer ):
       класс Meta:
            модель = Todo
            fields = ' all '
11
                                                                       просмотреть сырой
serializers.py размещенного с помощью W GitHub
```

ModelSerializer Класс будет создавать поля, которые соответствуют полям

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





```
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 c hos GitHub просмотреть raw
```

Наконец, мы напишем конфигурацию URL, используя Routers:

```
# todos / api / urls.py
 2
 3
    из rest framework импортировать маршрутизаторы
 4
 5
    от . импорт просмотров TodoViewSet
 7
    маршрутизатор = маршрутизаторы . DefaultRouter ()
    маршрутизатор . зарегистрироваться ( « задачи » , TodoViewSet , « задачи » )
    # 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.

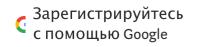
Написание модулей внешнего интерфейса

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.





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

views.py c 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:

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

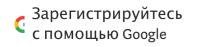
Подключите URL

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.





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:

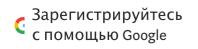
```
$ mkdir -p ./frontend/src/ enjcomponents,actions,reducers}
$ mkdir -p ./frontend/ enjstatic,templates</frontend</pre>
```

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.





```
$ npm init -y
```

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

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

```
$ npm i -D webpack webpack-cli
$ npm i -D babel-loader @ babel / core @ babel / preset-env @ babel
/ preset-response @ babel / plugin-предложение-класс-свойства

$ 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
```

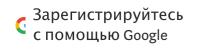
Создание конфигурационных файлов

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

```
// .babelrc
2
3
      "пресеты" : [
          "@ babel / preset-env" ,
            "цели" : {
              "узел" : "текущий"
10
            },
            "useBuiltIns": "использование",
11
            "corejs": 3
12
          }
13
        ],
```

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





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

```
// webpack.config.js
 2
 3
    модуль . экспорт = {
 4
      модуль : {
 5
         правила : [
           {
             тест: / \. ( js | jsx ) $ /,
             исключить : / node_modules / ,
             использовать : {
               загрузчик : 'babel-loader'
10
             }
11
12
           }
         ]
13
14
       }
    } ;
15
                                                                               просмотреть raw
webpack.config.js, размещенный на 💗 от GitHub
```

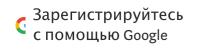
Кроме того, нам нужно переписать "scripts" свойство package.json файла:

```
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 }

раскаде.jsonc, размещенного на ♥ в GitHub
Просмотр raw
```

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





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

```
1 // frontend / src / index.js
2
3 импортировать приложение из «./components/App»;
index.js, размещенный на ♥ на GitHub
просмотреть raw
```

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

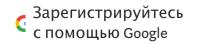
```
// внешний интерфейс / src / components / App.js
2
3
    импортировать React , { Component } из 'response';
    импортировать ReactDOM из 'response -dom';
    Класс Арр расширяет Компонент {
7
     render ( ) {
8
      возврат (
          < div >
9
           < h1 > ToDoCRUD < / h1 >
10
        < / div >
11
12
       ) ;
13
      }
    }
14
15
    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.





```
< tINK ret = Stytesheet nref = https://cunjs.ctoudrtare.com/ajax/tibs/semantiq
1.1
      < title > ToDoCRUD </ title >
12
    </ head >
13
14
15
    < тело >
     < div id = ' app ' > </ div >
16
17
18
     {% load static%}
      < script src = " {% static 'frontend / main.js'%} " > </ script >
19
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
```

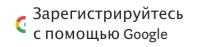
main.js Файл должен быть создан в static/frontend каталоге.

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.





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

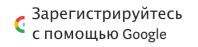
Просмотреть raw
```

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

```
// actions / todos.js
 2
   импорт axios из 'axios';
    import { GET_TODOS } us './types';
 4
 5
 6 // ПОЛУЧИТЬ ТОДОС
    export const getTodos = () => асинхронная отправка => {
    const res = ждать axios . get ( '/ api / todos /' );
      отправка ( {
       тип : GET_TODOS ,
       полезная нагрузка: рез . данные
      } ) ;
12
13 };
                                                                     просмотреть raw
todos.js, размещенный на 💚 на GitHub
```

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





```
// redurs / todos.js
 2
 3
    импорт _ из 'lodash';
    импортировать { GET TODOS } из '../actions/types';
 4
 5
    экспорт по умолчанию ( состояние = { } , действие ) => {
 7
     switch ( action . type ) {
       case GET_TODOS :
          возврат {
            ... состояние ,
            ... _ . mapKeys ( action . payload , 'id' )
12
       по умолчанию :
13
14
          возвратное состояние;
      }
15
16
    } ;
                                                                         просмотреть raw
todos.js, размещенный на 💚 на GitHub
```

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:

```
// redurs / index.js

import { combReducers } из 'redux';

импорта { peдуктор , как formReducer } от «перевождь-формы»;

импорт Todos из »./todos';

жспорт по умолчанию combReducers ( {

form : formReducer ,

Todos
```

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

Зарегистрируйтесь с помощью Google



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:

```
// fronted / src / store.js
 2
 3
    import { createStore , applyMiddleware } us 'redux';
    import { composeWithDevTools } из 'redux-devtools-extension';
    импортировать reduxThunk из 'redux-thunk';
    импортировать rootReducer из './reducers';
 7
    const store = createStore (
8
9
    rootReducer ,
     composeWithDevTools ( applyMiddleware ( reduxThunk ) )
10
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:

```
// компоненты / задачи / TodoList.js

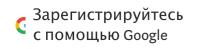
импортировать React , { Component } из 'response';

импорта { подключения } от 'реагируют-Redux';

import { getTodos } из '../../actions/todos';
```

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





```
16
              < div className = 'item' key = { todo . id } >
                < i className = 'большой значок календаря, выровненный по среднему значку
17
                < div className = 'content' >
18
                  < Имя класса = 'заголовок' > { TODO . задача } < / a >
19
                  < div className = 'description' > { todo . создал ат } < / div >
20
                < / div >
              < / div >
            ) ) }
23
2/
          < / div >
        ) ;
25
      }
26
    }
27
29
    const mapStateToProps = state => ( {
      Задачи: Объект . значения ( штат . задачи )
30
    } ) ;
31
    экспорт по умолчанию подключиться (
     mapStateToProps ,
34
      { getTodos }
35
    ) ( TodoList );
```

Мы используем Semantic UI list для оформления списка.

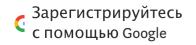
connect() Функция связывает этот компонент в магазин. Он принимает mapStateToProps в качестве первого аргумента, Action Creators в качестве второго аргумента. Мы сможем использовать состояние магазина в качестве реквизита, указав mapStateToProps.

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
```

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





Откройте Арр. ј с файл и обновите его следующим образом:

```
// компоненты / App.js
 2
    импортировать Dashboard из './todos/Dashboard'; // добавлено
 4
    импорта { Провайдер } от 'реагируют-Redux'; // добавлено
    импорт магазина из "../store"; // добавлено
 7
    Класс Арр расширяет Компонент {
     render ( ) {
       возврат (
          < Provider store = { store } >
            < 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.

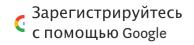
Проверка дисплея

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.





```
// actions / todos.js
 2
 3
    import { reset } из 'redux-формы'; // добавлено
    импортировать { GET TODOS , ADD TODO } из './types'; // добавлено ADD TODO
 4
 5
 6
    // ДОБАВИТЬ ТОДО
    экспорт const addTodo = formValues => асинхронная отправка => {
 7
     const res = ждать axios.post('/api/todos/', { ... formValues });
      отправка ( {
       тип : ADD_TODO ,
10
        полезная нагрузка: рез . данные
12
      рассылка ( cброс ( 'todoForm' ) );
13
14
    } ;
                                                                      просмотреть raw
todos.js, размещенный на 💚 на GitHub
```

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

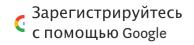
Переходники

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

```
// redurs / todos.js
2
    импортировать { GET_TODOS , ADD_TODO } из '../actions/types' ; // добавлено ADD_
3
4
5
    экспорт по умолчанию ( состояние = { } , действие ) => {
     switch ( action . type ) {
        // ...
       case ADD_TODO : // добавлено
          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.





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

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

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

Зарегистрируйтесь с помощью Google



```
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.

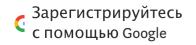
Когда мы щелкаем текстовое поле, а затем удаляем фокус, отображается ошибка проверки, поэтому укажите, touch0nBlur: false чтобы отключить ее.

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

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

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





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

ТоdoCreate.js, размещенного в ♥ на GitHub
```

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.

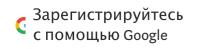
Если нам не нужно указывать mapStateToProps функцию, установите $null\ B$ connect().

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

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

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





```
2
    импортировать React , { Component } из 'response';
 3
 4
     Заголовок класса расширяет Компонент {
      render ( ) {
        возврат (
          < div className = 'UI перевернутое меню' style = { { borderRadius : '0' } } >
            < Имя класса = 'элемента заголовка' > TodoCRUD < / >
            < Имя класса = 'пункт' > Главная < / >
10
11
          < / div >
12
        ) ;
13
      }
    }
14
15
16
     Заголовок экспорта по умолчанию ;
Header.is, размешенный на with GitHub
                                                                           просмотреть raw
```

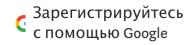
Откройте App.js файл и вложите Header компонент:

```
// компоненты / App.js
2
    импортировать заголовок из './layout/Header'; // добавлено
3
4
    Класс Арр расширяет Компонент {
     render ( ) {
        возврат (
          < Provider store = { store } >
            < Header / > // добавлено
            < Панель инструментов / >
10
11
          < / Provider >
12
        ) ;
13
      }
    }
App.js, размещенный в with на GitHub
                                                                           просмотреть raw
```

В этом уроке заголовок - это просто укращение

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





```
1 // Trontend / STC / HIStory.]S

2

3 import { createBrowserHistory } из 'history';

4

5 экспорт по умолчанию createBrowserHistory ( );

history.js, размещенный на ♥ GitHub
```

действия

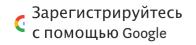
Откройте actions/todos.js файл и добавьте двух новых создателей действий:

```
// actions / todos.js
 2
    импортировать историю из '../history'; // добавлено
    импортировать { GET_TODOS , GET_TODO , ADD_TODO , DELETE_TODO } из './types';
 5
    // ПОЛУЧИТЬ ТОДО
    экспорт const qetTodo = id => асинхронная отправка => { // добавлено
     const res = \mathsf{ждать} axios . get ( \dot{} api / todos / f { id f / f );
      отправка ( {
10
        тип : GET_TODO ,
        полезная нагрузка: рез . данные
11
      } ) ;
13
    } ;
14
15
    // УДАЛИТЬ ТОДО
    экспорт const deleteTodo = id => асинхронная отправка => { // добавлено
16
     жду аксиос . delete ( `/ api / todos / $ { id } /` );
17
18
     отправка ( {
        тип : DELETE_TODO ,
19
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.





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

```
// redurs / todos.js
 2
    импорт из 'lodash'; // добавлено
    импортировать { GET_TODOS , GET_TODO , ADD_TODO , DELETE_TODO } из '../actions/
 4
 5
    экспорт по умолчанию ( состояние = { } , действие ) => {
     switch ( action . type ) {
 7
        // ...
        case GET_TODO : // добавлено
        case ADD_TOD0 :
10
11
          возврат {
12
            ... состояние ,
            [ действие . полезная нагрузка . id ] : действие . полезная нагрузка
13
14
          } ;
        case DELETE_TODO : // добавлено
15
          вернуть _ . опустить ( состояние , действие . полезная нагрузка ) ;
16
        // ...
17
      }
18
    } ;
19
                                                                          просмотреть raw
todos.js, размещенный на 💚 на GitHub
```

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.

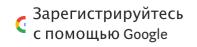
Компоненты

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

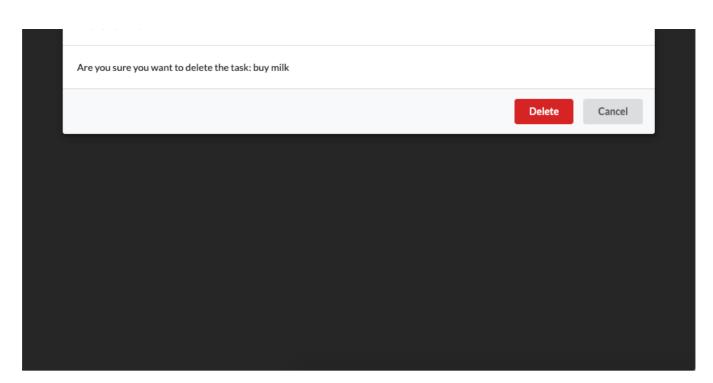


Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.







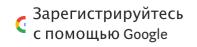
Модальное окно для подтверждения удаления

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

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

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





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:

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

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

Зарегистрируйтесь с помощью Google

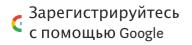


```
кнопка <
27
               onClick = { ( ) => это . реквизит . deleteTodo ( id ) }
               className = 'UI отрицательная кнопка'
              удалять
             < / button >
             < Link to = '/' className = 'UI Button' >
               0тмена
             < / Link >
           < / Fragment >
         ) ;
       render ( ) {
         возврат (
41
           < Модальный
             title = 'Удалить Todo'
42
             содержание = { это . renderContent ( ) }
43
             действия = { это . renderActions ( ) }
45
             onDismiss = { ( ) => история . нажать ( '/' ) }
           / >
46
47
         ) ;
      }
48
    }
49
    const mapStateToProps = ( state , ownProps ) => ( {
51
52
      todo : состояние . todos [ ownProps . совпадают . парам . id ]
    } ) ;
53
54
55
    экспорт по умолчанию подключиться (
56
      mapStateToProps ,
       { getTodo , deleteTodo }
57
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.





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

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

Зарегистрируйтесь с помощью Google



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

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

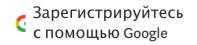
Наиболее распространенный вариант использования низкоуровневого <Router> - это синхронизация пользовательской истории с библиотекой управления состояниями, такой как Redux или Mobx. Обратите внимание, что это не требуется для использования библиотек управления состояниями вместе с React Router, это только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





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

```
// actions / todos.js
 2
    импортировать { GET TODOS , GET TODO , ADD TODO , DELETE TODO , EDIT TODO } из
 3
 4
 5
    // РЕДАКТИРОВАТЬ ТОДО
    экспорт const editTodo = (id, formValues) => асинхронная отправка => {
     const res = ждать axios.patch ( `/ api / todos / $ { id } /` , formValues );
     отправка ( {
       тип : EDIT_TODO ,
10
        полезная нагрузка: рез . данные
11
      } ) ;
      история . нажать ( '/' );
12
13 };
                                                                       просмотреть raw
todos.js, размещенный на 💚 на GitHub
```

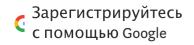
Переходники

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

```
// redurs / todos.js
2
3
   import {
  GET_TODOS ,
4
    GET_TODO ,
5
    ADD_TODO ,
7
    DELETE_TODO ,
    EDIT_TODO // добавлено
   } us '../actions/types';
10
    экспорт по умолчанию ( состояние = { } , действие ) => {
11
      switch ( action . type ) {
12
13
       // ...
       case GET_TOD0 :
14
      case ADD TODO:
```

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





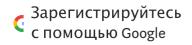
Компоненты

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

```
// компоненты / задачи / TodoEdit.js
    импорт _ из 'lodash';
    импортировать React , { Component } из 'response';
    импорта { подключения } от 'реагируют-Redux';
    import { getTodo , editTodo } us '../../actions/todos';
7
    импортировать TodoForm из './TodoForm';
8
    Класс TodoEdit расширяет Компонент {
10
      componentDidMount ( ) {
        это . реквизит . getTodo ( это . реквизит . совпадение . params . id ) ;
11
12
      }
13
14
      onSubmit = formValues => {
15
        это . реквизит . editTodo ( это . реквизит . match . params . id , formValues );
16
      } ;
17
18
      render ( ) {
19
        возврат (
          < div className = 'ui container' >
            < h2 style = { { marginTop : '2rem' } } > Изменить Todo < / h2 >
21
22
            < TodoForm
              initialValues = { _ . pick ( this . props . todo , 'task' ) }
23
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.





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.

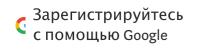
Откройте TodoList.js файл и обновите его {todo.task} следующим образом:

Давайте добавим новый компонент в App. js файл:

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

Читайте больше историй в этом месяце.

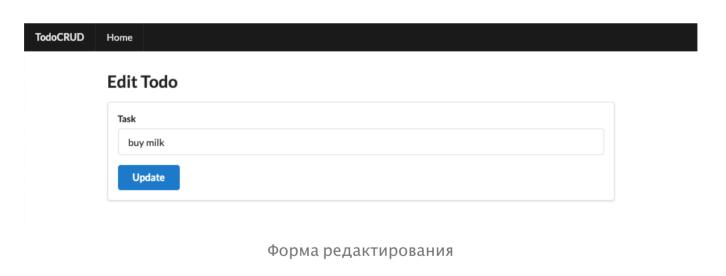
Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





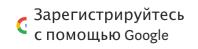
```
// компоненты / задачи / TodoForm.js
 2
 3
    Класс TodoForm расширяет Компонент {
 4
      // ...
 5
       render ( ) {
 6
        const btnText = `$ { this . реквизит . initialValues ? «Обновить» : «Добавить»
        возврат (
          < div className = 'UI сегмент' >
10
             < форма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

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



Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.





Следующий шаг

Реализовать аутентификацию пользователя в приложении Django & React C помощью Knox

Добавьте аутентификацию на основе токенов с помощью Django-rest-knox в приложение, созданное с помощью Django и...

medium.com

программирование кодирование Джанго реагировать Redux

ОсправкеЮридическая

Читайте больше историй в этом месяце.

Создайте бесплатный аккаунт, чтобы узнать больше о Medium.

