Практическая работа №1 на тему React-Redux

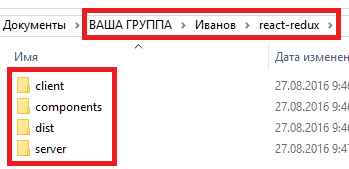
Цель работы: подготовка окружения для работы с React и его использование

Ход работы:

Эта часть практических работ предполагает наличие знаний по React, которые можно получить по следующей ссылке:

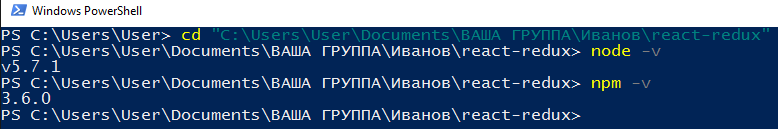
1) <https://maxfarseer.gitbooks.io/react-course-ru/content/>

Создайте в своей папке новую директорию «react-redux» со следующей структурой подкаталогов.



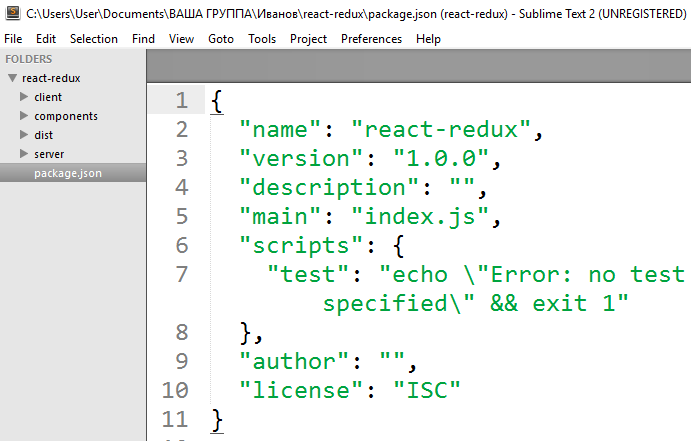
За серверный код будет отвечать Node.js – это платформа, основанная на JavaScript движке V8, и предлагающая асинхронное API для работы с сетью и диском. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода, другими словам это своего рода серверный JavaScript. Для подготовки среды приложений на Node.js используется пакетный менеджер npm, позволяющий скачивать и устанавливать различные библиотеки.

Запустите командную строку (например, PowerShell), перейдите из неё в каталог проекта и убедитесь, что на компьютере установлен Node.js и npm.



Инициализируйте каталог с помощью команды npm init, в процессе инициализации согласитесь со всеми настройками по умолчанию.

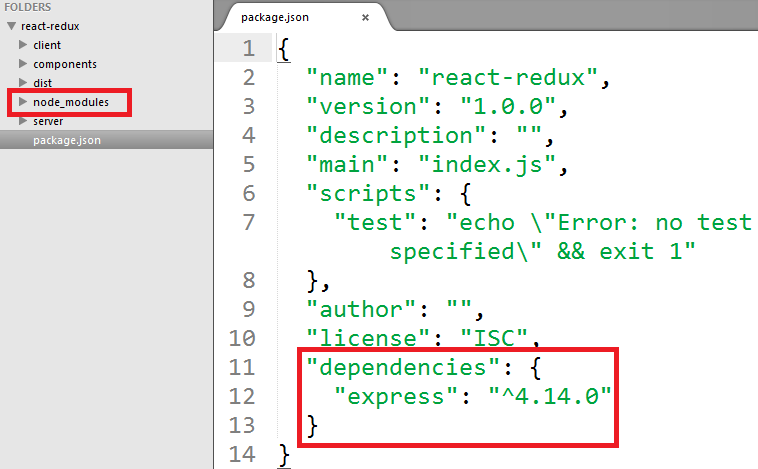
Откройте папку проекта в Sublime Text. После инициализации в каталоге будет создан файл package.json с описанием нашего проекта.



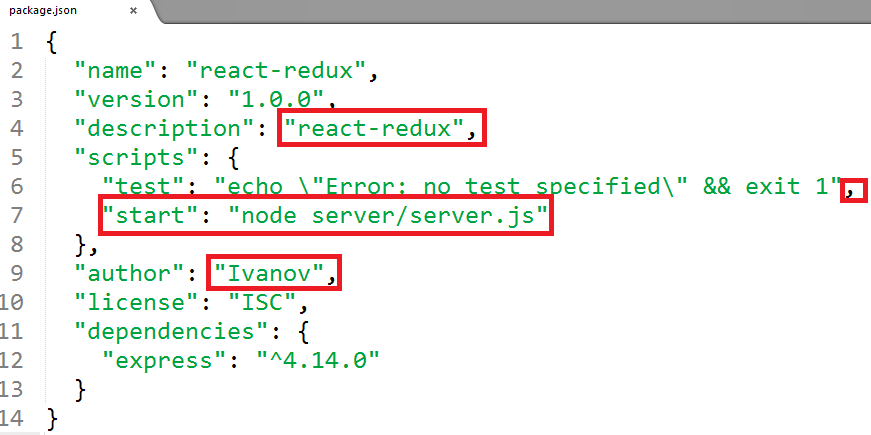
Настроим локальный веб-сервер, для этого скачаем фреймворк Express с помощью команды «npm install express --save» (ключ save добавит в файле package.json версию установленного express).



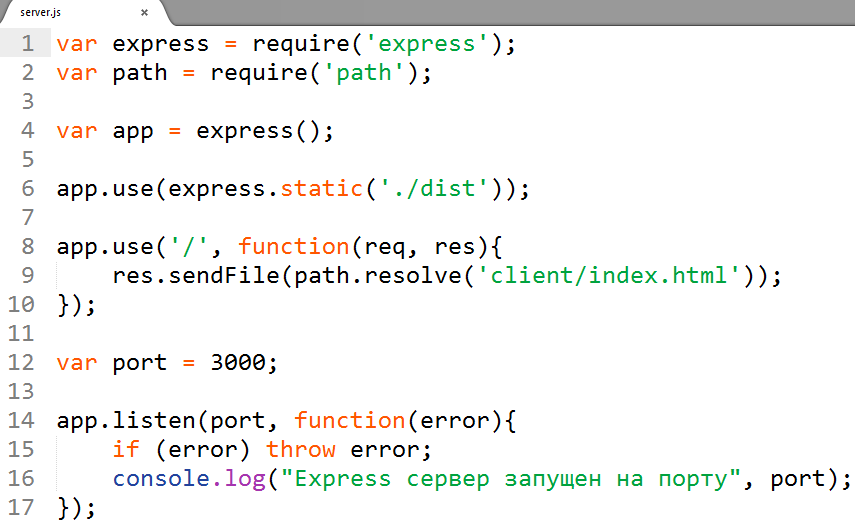
В файле package.json добавится новая зависимость и в каталоге проекта создастся папка node\_modules со скаченными и установленными модулями.



Уберите из файла package.json параметр main, добавьте какое-нибудь описание, новый скрипт start для запуска веб-сервера, укажите себя в качестве автора.

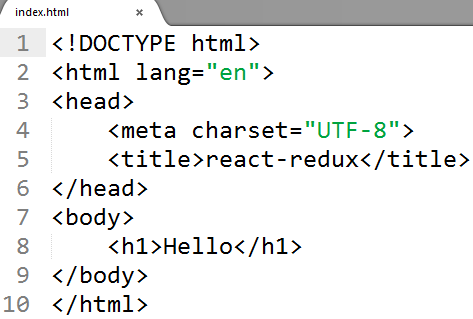


Сохраните изменения и создайте в папке server новый файл server.js со следующим содержимым.

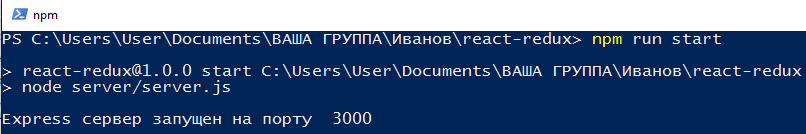


Здесь мы импортируем установленный модуль express и стандартный модуль path для работы с путями файлов и каталогов. Создаем объект экземпляр сервера с помощью вызова express(). Указываем серверу каталог со статическими файлами, в нашем случае это dist, в нем будут храниться клиентские js-файлы и css-стили. На 8 строке указываем серверу, чтобы на запрос «/» выполнялась функция с двумя параметрами (req – request/запрос, res – response/ответ), которая в ответ будет отправлять файл index.html. В конце указываем, на каком порту будет работать сервер.

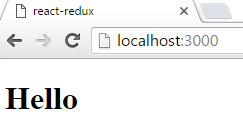
Сохраните изменения и создайте в папке client новый файл index.html со следующим содержимым.



Сохраните изменения и запустите сервер с помощью команды npm run start (где start название скрипта из файла package.json).



Перейдите в браузере по адресу localhost:3000.



Для того чтобы остановить веб-сервер нажмите сочетание клавиш Ctrl + C.

С помощью npm установите react и react-dom.



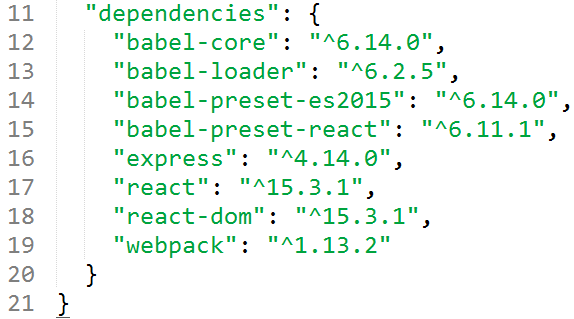
Для сборки клиентских скриптов будем использовать Webpack, установите его с помощью следующей команды.



Дополнительно из пакетов нам понадобится Babel, это специальный компилятор, который будет преобразовывать написанный нами JSX код из React и ECMAScript 6 в ECMAScript 5, для лучшей поддержки старых браузеров. Если перейти на официальный сайт Babel на страницу Setup, то можно найти необходимые пакеты. Для работы нам потребуются установить следующие пакеты: babel-core, babel-loader, babel-preset-es2015, babel-preset-react.

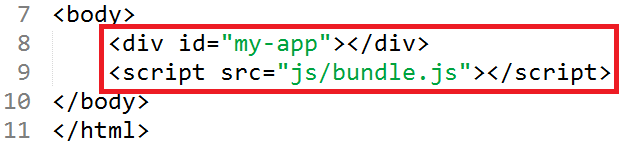


В итоге секция зависимостей в файле package.json будет выглядеть следующим образом.

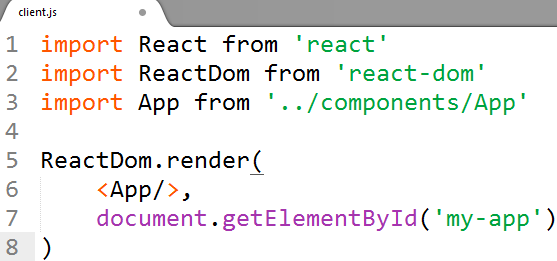


К моменту начала работы версии пакетов могут отличаться.

Добавьте в фале index.html код по образцу.

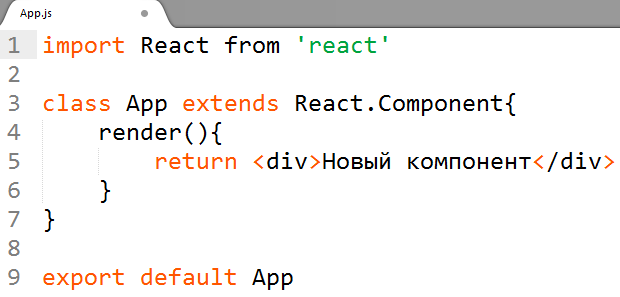


Сохраните и создайте в папке client новый файл client.js со следующим содержимым.



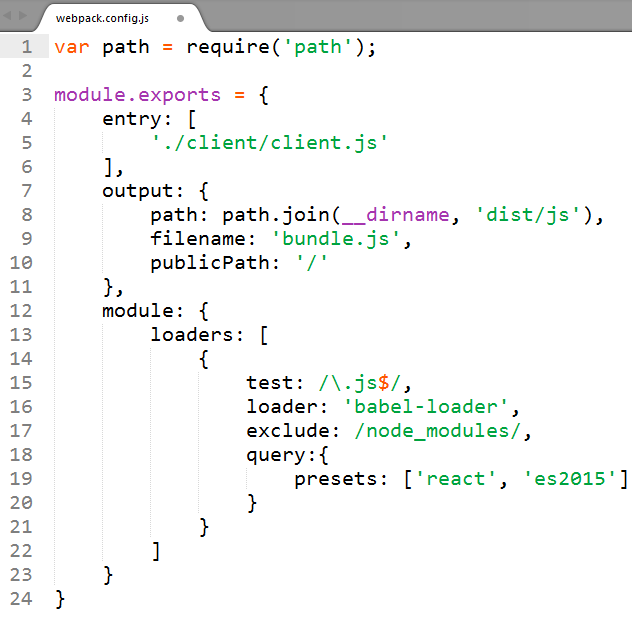
Здесь мы подключаем объекты React и ReactDom, используя синтаксис ES 6, и еще подключаем компонент App, файл который создадим чуть позже. Далее с помощью метода render у ReactDom вызываем отрисовку импортируемого компонента App и вторым параметром указываем элемент на странице, в котором будет производиться отрисовка.

Сохраните изменения и создайте в папке components новый файл App.js со следующим содержимым.



Здесь мы используем новый синтаксис ES 6 для создания класса App, который наследуем от Component из React. У компонента должен быть метод render, который возвращает разметку для отрисовки компонента (здесь используется не HTML код, а похожий по синтаксису JSX). Так как мы импортируем компонент App в другом файле (client.js), то вызываем для него export default.

Сохраните изменения и добавьте в папку проекта новый файл webpack.config.js со следующим содержимым.

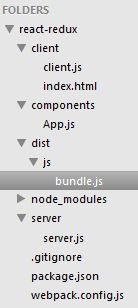


Здесь настраиваем конфигурацию для webpack, в которой указываем начальный файл client.js, параметры для выходного файла. В секции module указываем загрузчики для преобразования кода в ES 5. В параметре test вводим регулярное выражение для поиска js файлов, указываем имя загрузчика и путь к его файлам. В exclude указываем, чтобы babel игнорировал папку node\_modules. В query указываем библиотеки, которые будут участвовать в преобразованиях.

Сохраните изменения и выполните команду webpack --config webpack.config.js для сборки файла bundle.js.



После этого в папке dist создаться каталог с файлом bundle.js, который подключен в index.html. Суть в следующем, у нас есть начальный файл client.js, в котором используются сущности из других файлов (App, React, ReactDom). Webpack анализирует все используемые зависимости и соединяет их в одном файле, дополнительно с помощью загрузчика Babel, производится преобразования JSX кода React и синтаксиса ES 6 в ES 5 для поддержки старых браузеров.

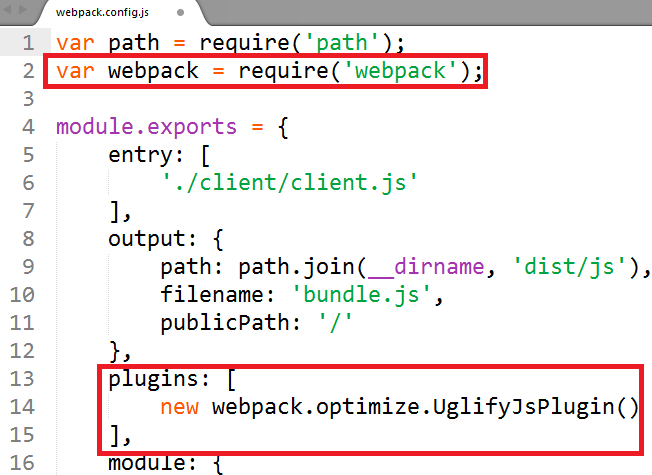
  
Запустите веб-сервер (npm run start) и проверьте в браузере отрисовку компонента.



Сейчас наш собранный файл bundle.js имеет довольно большой размер.

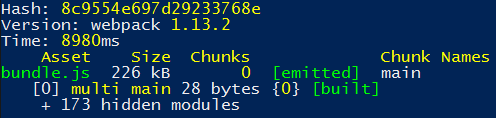


Мы можем сжать его с помощью webpack, используя плагин UglifyJsPlugin, который удалит пробелы и прочие не нужные символы. Введите код по образцу в файл webpack.config.js.



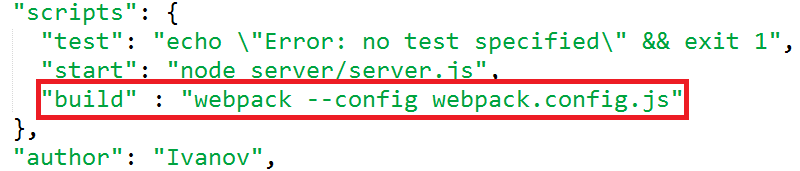
Сохраните изменения и повторно запустите сборку файла.





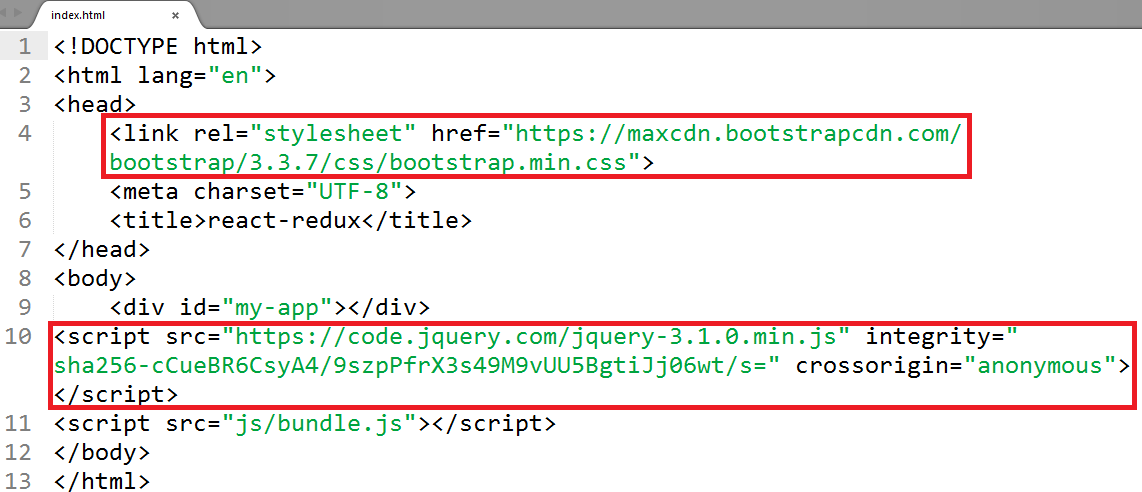
Сейчас собранный файл имеет размер в несколько раз меньше, что ускорит его загрузку. Так как процесс сжатия занимает определенное время, то на этапе разработки приложения его можно отключить.

Добавьте в файл package.json новый скрипт для упрощенного запуска webpack.

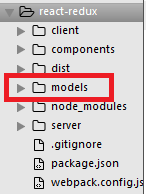


Сейчас сборку файла можно будет запускать командой npm run build.

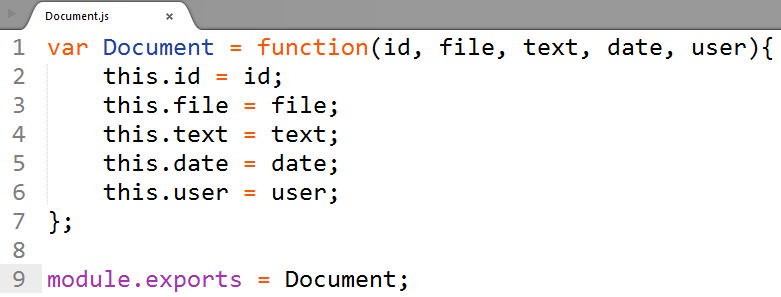
Для начала реализуем только базовый функционал работы с файлами без использования регистрации и авторизации. Перейдите в файл index.html и добавьте CDN ссылки на Bootstrap.min.cs и Jquery.



Создайте в папке проекта новый каталог с именем «models».



На начальном этапе для простоты разработки будем хранить документы не в базе данных, а в памяти, поэтому внутри каталога models создайте новый файл Document.js со следующим содержимым.



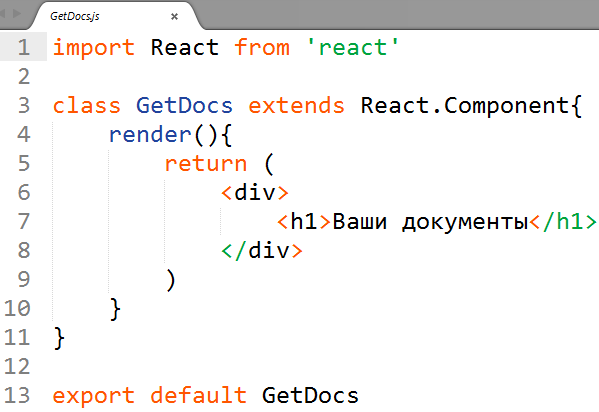
Здесь мы объявили функцию, с помощью которой будем создавать объекты, отражающие структуру документов.

В этом же каталоге models создайте второй файл Repository.js со следующим содержимым.

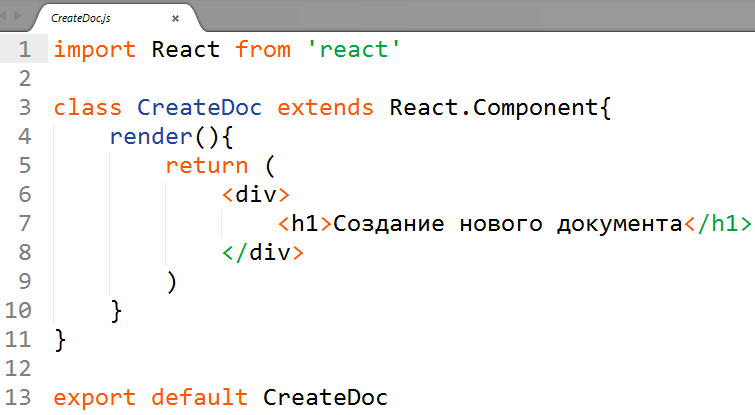


Репозиторий будет включать в себя массив с начальным количеством документов и основные операции с этим массивом (получение, добавление, удаление и изменение).

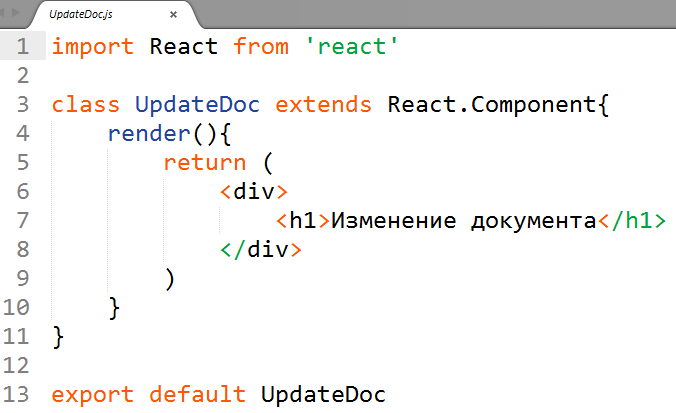
Далее в папке components создайте новый файл GetDocs.js со следующим содержимым.



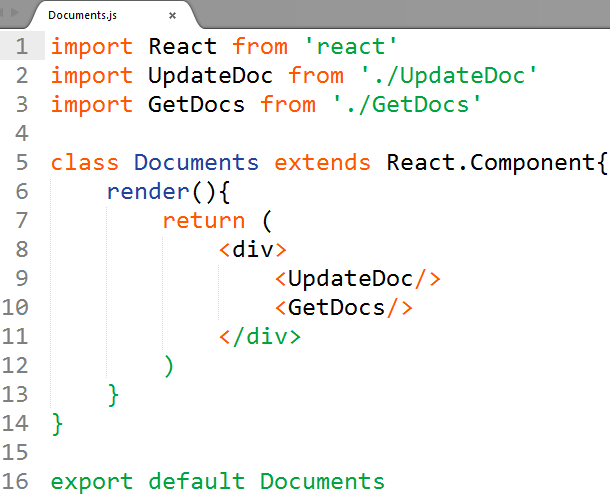
Сохраните и создайте в этой же папке новый файл CreateDoc.js со следующим содержимым.



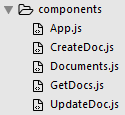
Сохраните и создайте в этой же папке новый файл UpdateDoc.js со следующим содержимым.



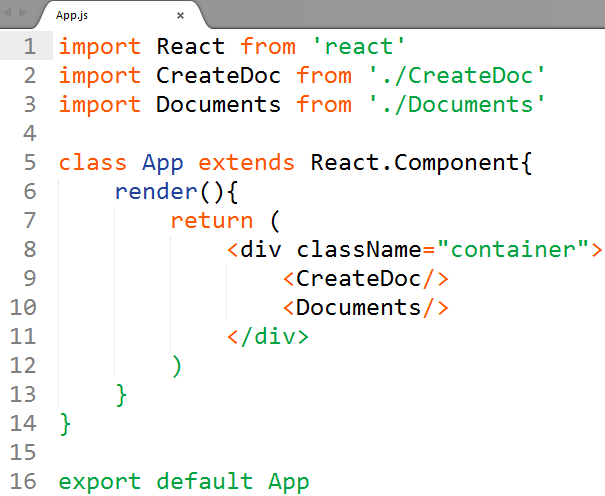
Сохраните и создайте в этой же папке новый файл Documents.js со следующим содержимым.



Текущая структура каталога components.



Перейдите в файл App.js и измените код по образцу.



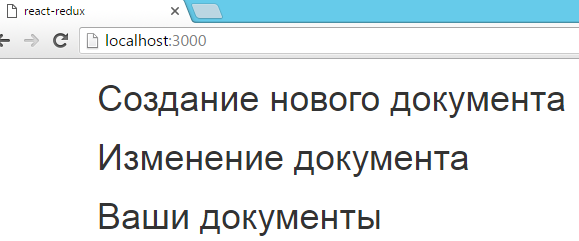
Сохраните изменения и запустите сборку bundle.js.



Запустите веб-сервер.



Обновите страницу в браузере.

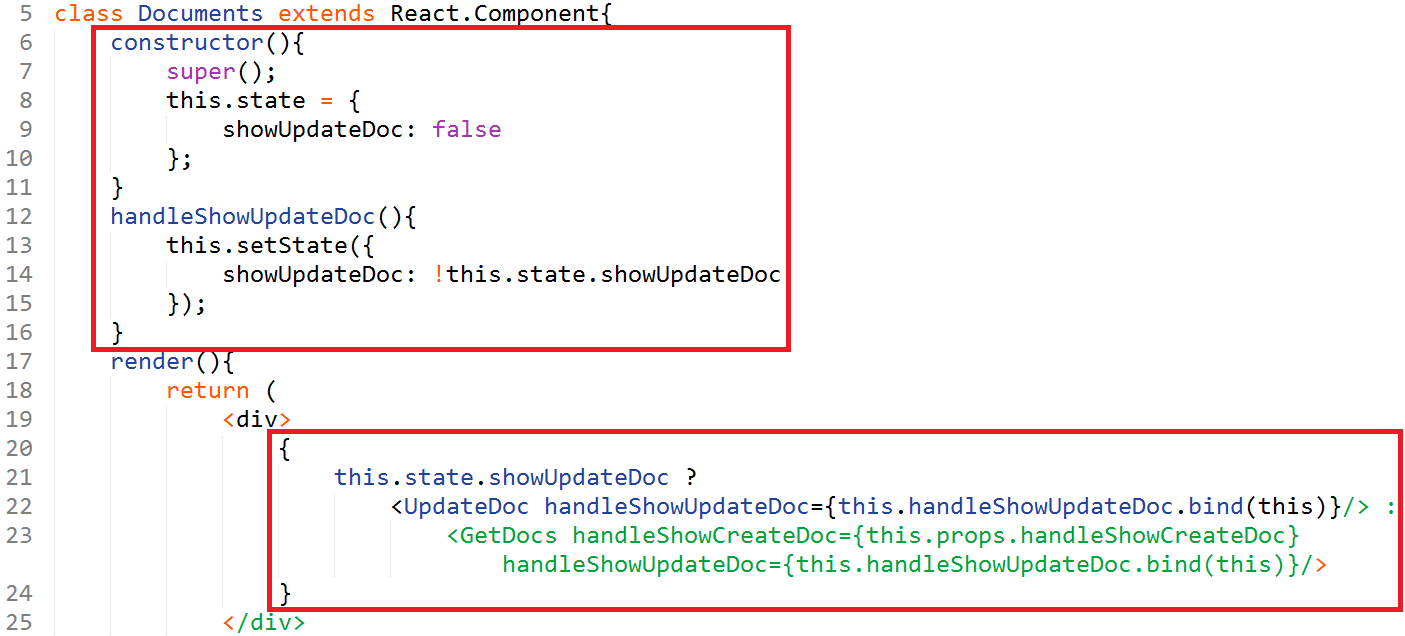


Как видим, все компоненты отображаются вместе. Сейчас необходимо реализовать механизм вывода конкретного компонента в зависимости от ситуации. Для этого перейдите в файл App.js и добавьте код по образцу.



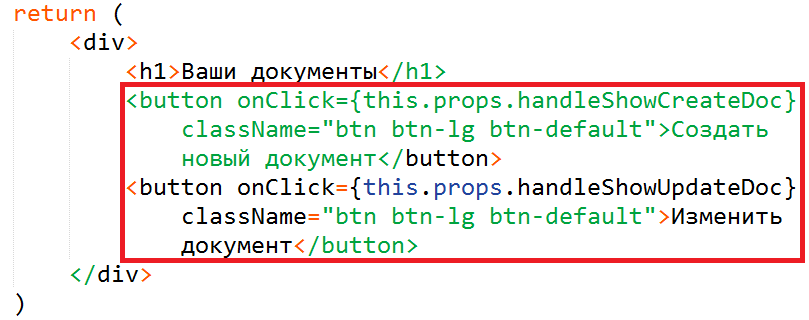
Здесь с помощью тернарного оператора мы будем отображать тот или иной компонент в зависимости от значения showCreateDoc и передавать в эти компоненты ссылку на метод handleShowCreateDoc, который инвертирует состояние showCreateDoc.

Сохраните изменения, перейдите в файл Documents.js и измените код по образцу.

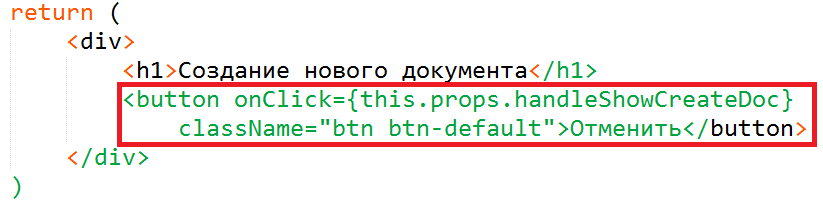


Здесь мы аналогичным образом создаем в state поле showUpdateDoc, в зависимости от значения которого будет отображаться один из компонентов. В компоненты передаем ссылку на метод для изменения значения showUpdateDoc. В компонент GetDocs помимо этого, пробрасываем ссылку на метод handleShowCreateDoc из App.js.

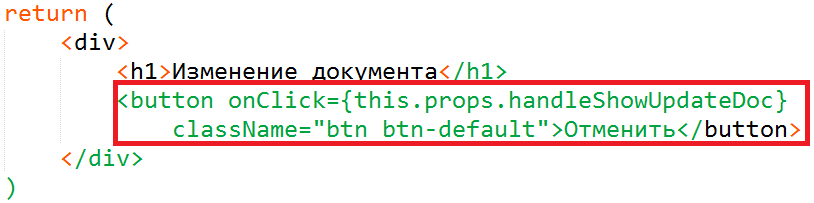
Сохраните изменения, перейдите в файл GetDocs и добавьте кнопки для отображения компонентов CreateDoc и UpdateDoc.



Сохраните изменения, перейдите в файл CreateDoc и добавьте кнопку возврата к GetDocs.



Сохраните изменения, перейдите в файл UpdateDoc и добавьте кнопку возврата к GetDocs.



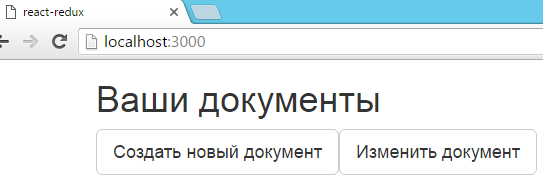
Сохраните изменения и запустите сборку bundle.js.



Запустите веб-сервер.

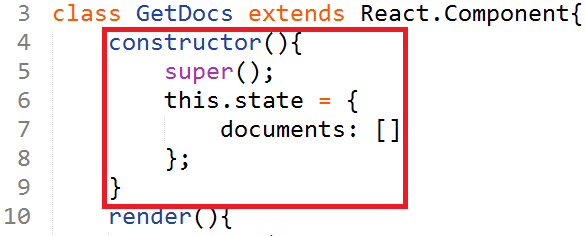


Обновите страницу в браузере.

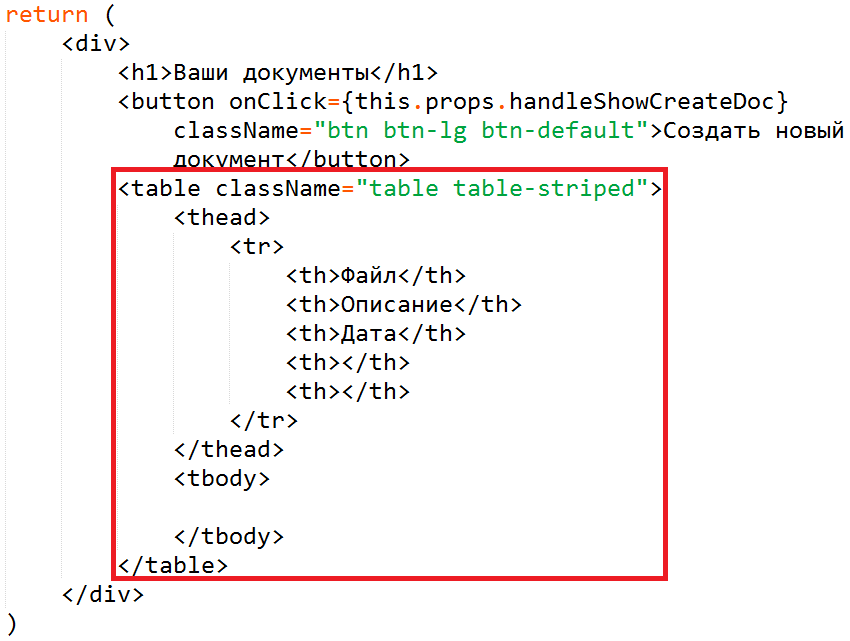


При нажатии на кнопки вызываются соответствующие методы, которые меняют состояние полей в state и отображаются нужные компоненты.

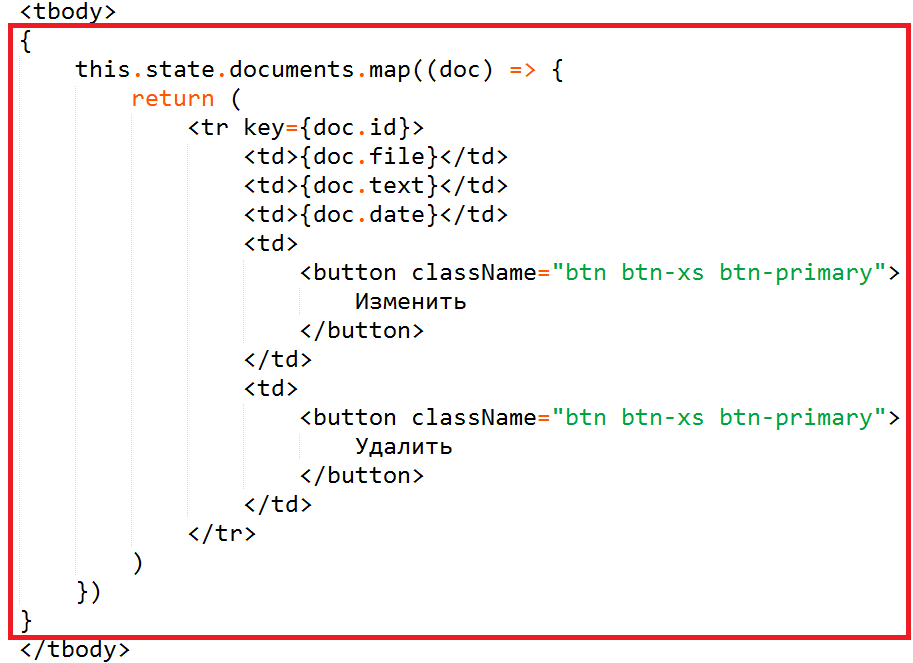
Далее реализуем вывод документов в компоненте GetDocs, для этого в конструкторе в объекте state объявим пустой массив documents.



В этом же файле в методе render добавьте вывод таблицы.

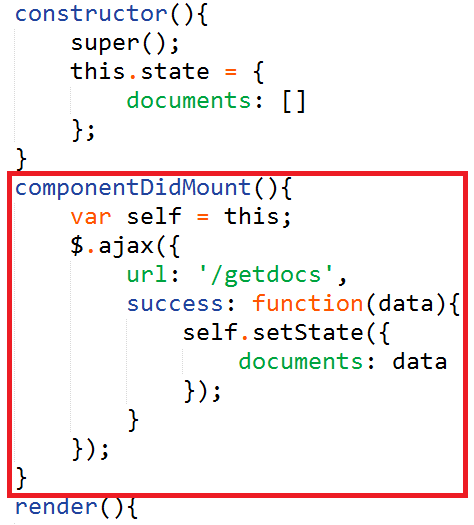


В теле таблицы введите код по образцу.



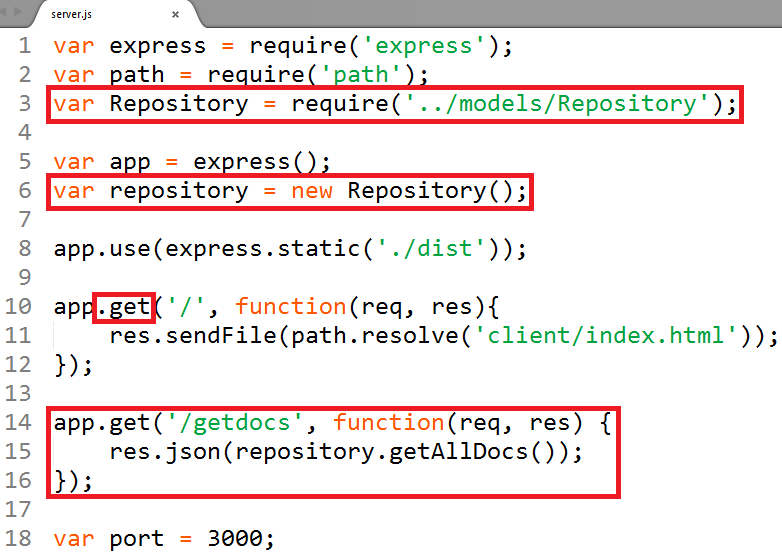
Здесь с помощью функции map перебираются элементы массива documents и выводятся в табличной форме.

На данном этапе массив documents является пустым, поэтому добавим специальный метод жизненного цикла React компонентов, который срабатывает сразу после отрисовки компонента.



Здесь с помощью Jquery мы отправляем ajax запрос по адресу getdocs и заполняем массив documents полученными из запроса данными.

Сохраните изменения, перейдите в файл server.js и введите код по образцу.



Здесь мы импортируем репозиторий, создаем его экземпляр, меняем метод для обработки корневого запроса с «use» на «get» и добавляем обработку «get» запроса для адреса getdocs, на который будет приходить ajax запрос, в ответ мы будем отправлять данные из репозитория в JSON формате.

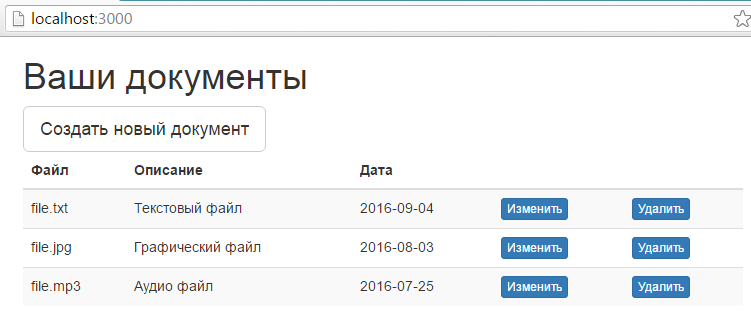
Сохраните изменения и запустите сборку bundle.js.



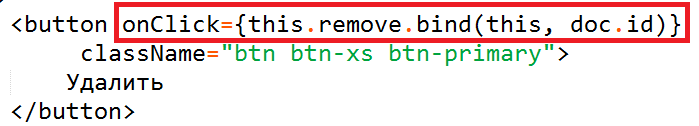
Запустите веб-сервер.



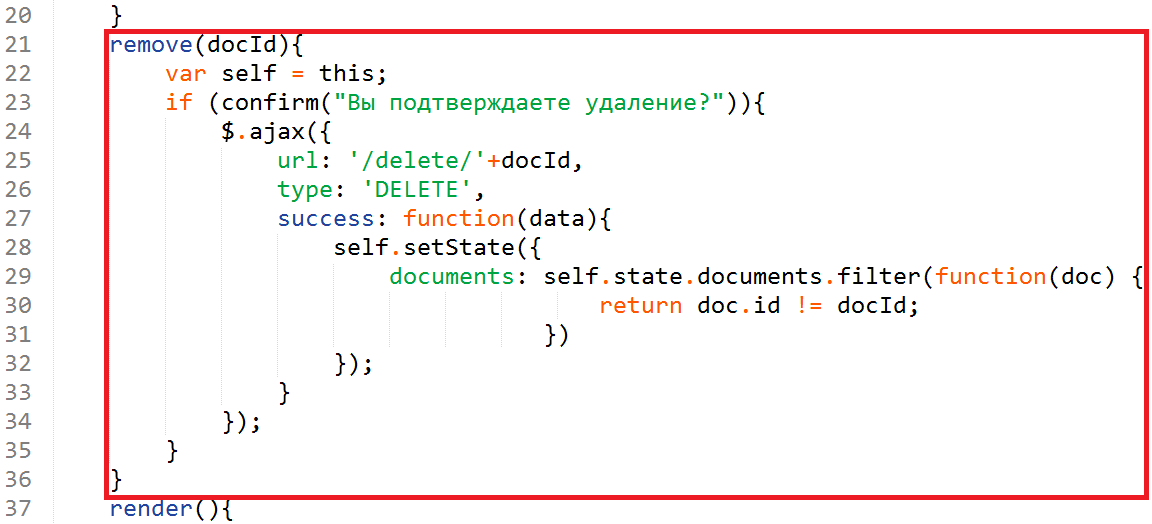
Обновите страницу в браузере.



Реализуем удаление документов, для этого внесите изменения в файле GetDocs.js.

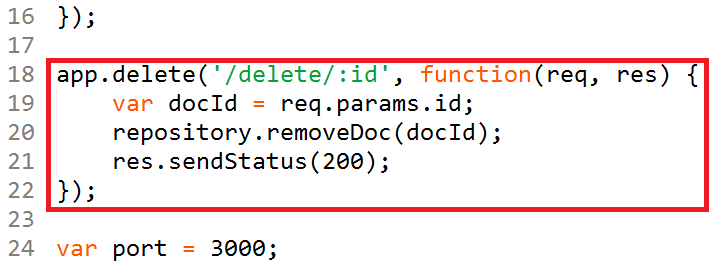


На кнопку удалить вешаем обработчик нажатия, в котором вызывается метод remove с идентификатором текущего документа. Создайте в этом же файле метод remove по образцу.



Здесь мы спрашиваем подтверждение удаления и отправляем ajax запрос методом delete, если запрос прошел успешно, то обновляем массив documents, отфильтровав удаленный элемент.

Сохраните изменения, перейдите в файл server.js и добавьте обработку нового запроса по образцу.



Здесь мы получаем из запроса идентификатор документа, удаляем его и отправляем в ответ успешный статус запроса.

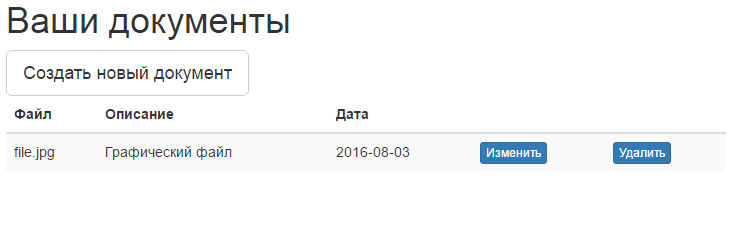
Сохраните изменения и запустите сборку bundle.js.



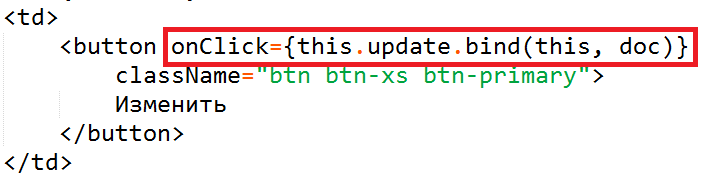
Запустите веб-сервер.



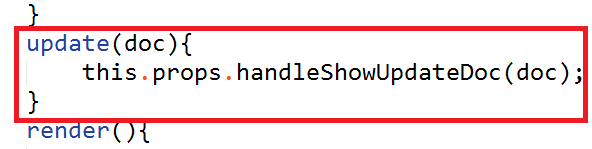
Обновите страницу в браузере и протестируйте удаление документов.



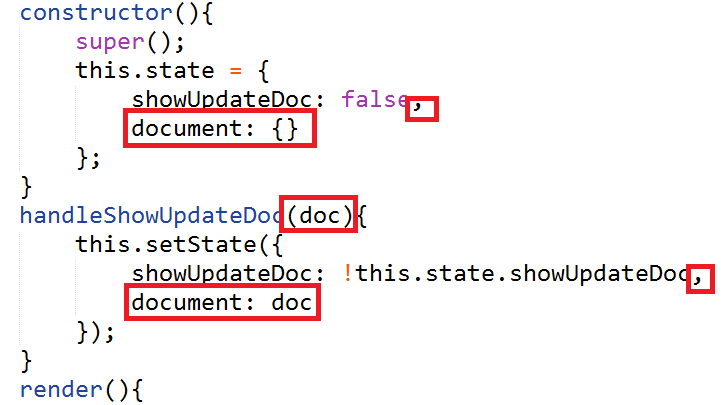
Далее реализуем механизм изменения документов, для этого в файле GetDocs.js создайте обработчик нажатия кнопки изменить.



В метод update будет передаваться весь текущий документ. Создайте метод update по образцу.

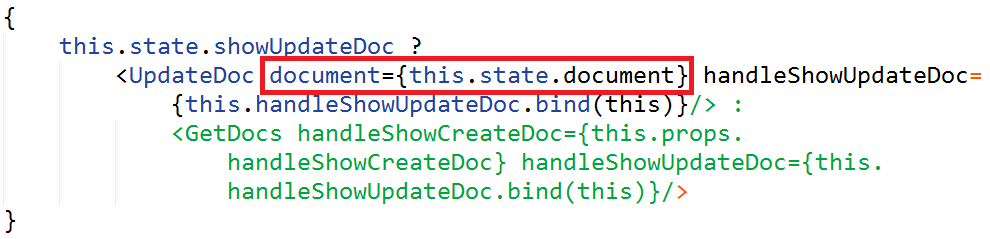


Здесь мы вызываем метод меняющий состояние поля showUpdateDoc из компонента Documents, чтобы вывести на экран компонент UpdateDoc. Нам требуется пробросить изменяемый документ в компонент UpdateDoc, для этого мы вначале отправим документ в компонент Documents, а потом в UpdateDoc. Так как исходный метод handleShowUpdateDoc не обрабатывает входные параметры, то требуется его модифицировать. Для этого перейдите в файле Documents.js и введите код по образцу.



Здесь в state создаем пустой объект document, при вызове handleShowUpdateDoc присваиваем в document входной параметр.

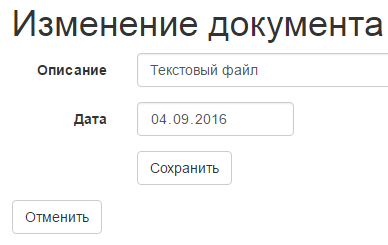
В этом же файле нужно пробросить изменяемый документ в компонент UpdateDoc.



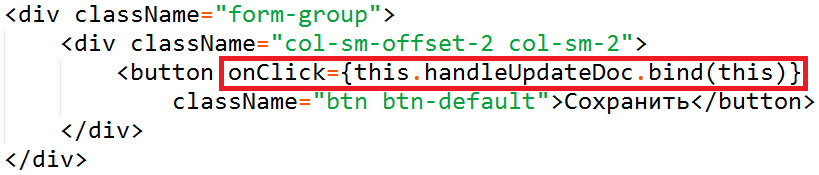
Сохраните изменения, перейдите в файл UpdateDoc.js и добавьте форму изменения документа по образцу.



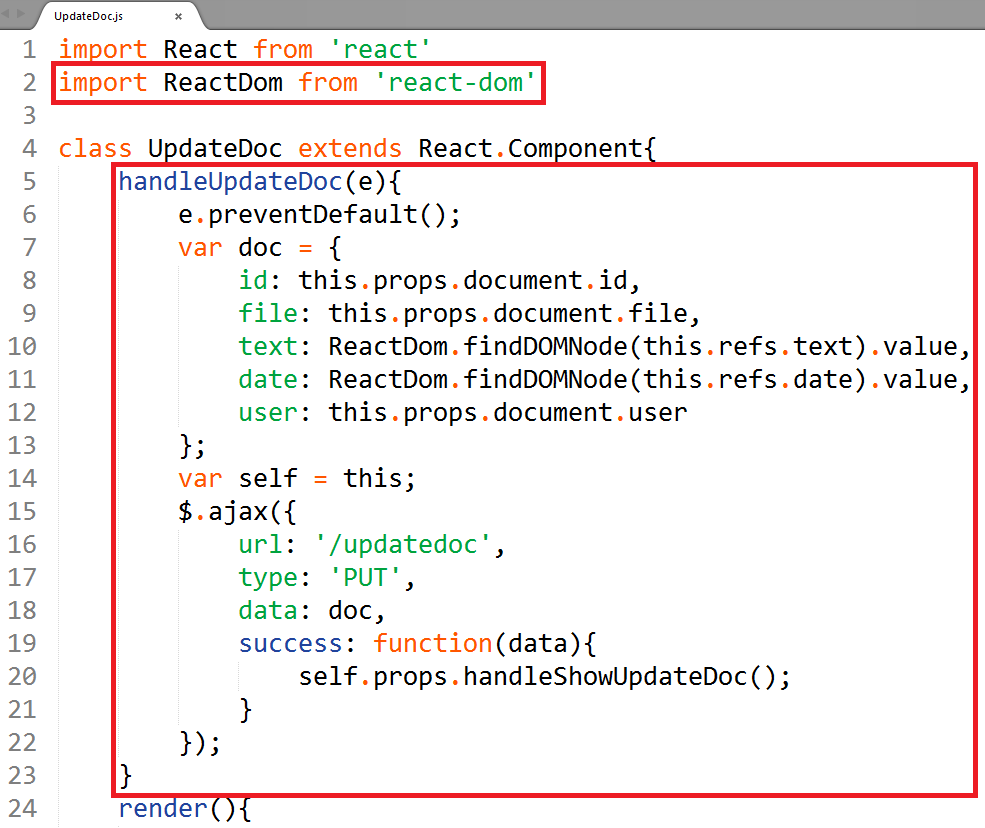
Сохраните изменения, соберите bundle.js, запустите веб-сервер и протестируйте кнопку изменить.



Далее реализуем логику сохранения изменений документа, для этого вернитесь в файл UpdateDoc.js и обработчик для кнопки сохранить.



Сейчас создадим метод handleUpdateDoc, который будет получать содержимое полей формы и отправлять эти данный на сервер. Чтобы получить значения полей формы воспользуемся методом findDOMNode из библиотеки react-dom, для этого её необходимо подключить.

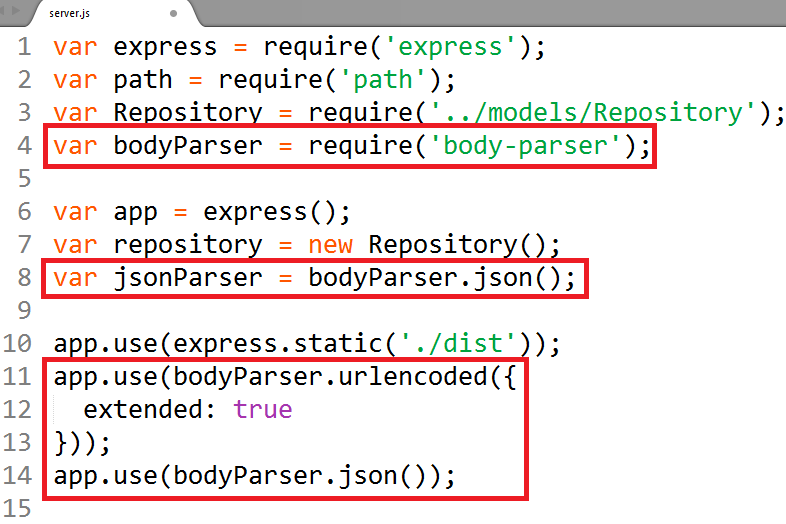


Параметр «e» представляет собой контекст, из которого вызывается метод. Нам нужен контекст, чтобы взывать метод preventDefault, который отменит стандартной поведение формы при отправке, потому что мы будем отправлять данные с помощью ajax запроса. Далее создается новый экземпляр документа (неизменяемую часть берем из проброшенного документа из компонента Documents, остальное берем из полей формы). Обратите внимание, что у полей формы прописан атрибут refs, именно по нему мы обращаемся к нужным полям, чтобы получить их содержимое. Далее отправляем обновленный документ ajax запросом с помощью метода PUT, если запрос успешный, то вызываем ссылку на метод из компонента Documents, который скроет компонент UpdateDoc и отобразит GetDocs.

Сохраните изменения. Для того чтобы было удобнее получать данные из запроса на стороне сервера, установим с помощью npm пакет body-parser.

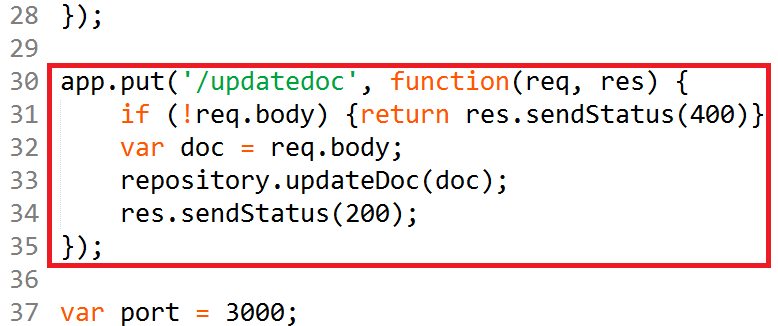


После этого в package.json появится соответствующая зависимость. Далее перейдите в файл server.js и добавьте код для инициализации body-parser.



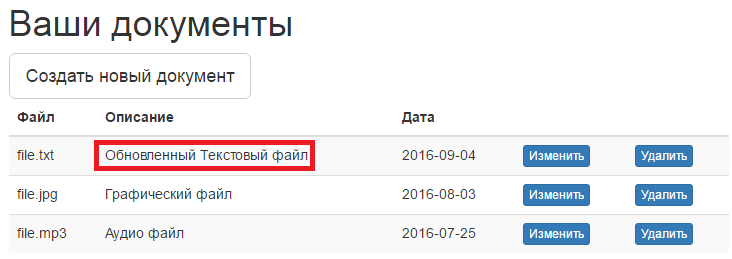
Это стандартная инициализация с официального сайта.

Далее создай обработчик для PUT запроса по образцу.

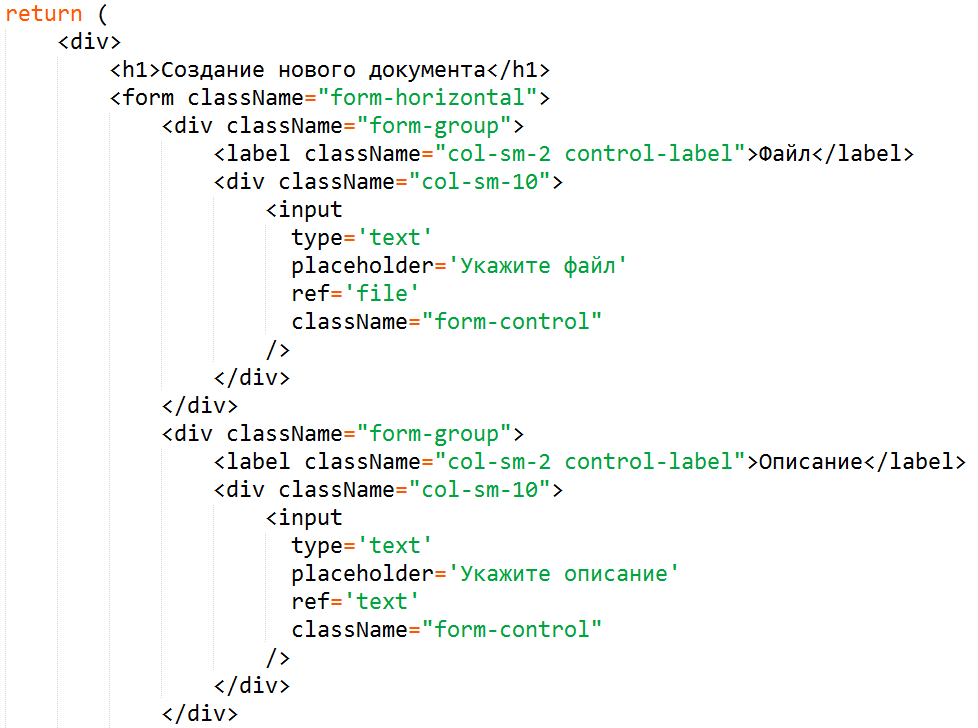


Здесь благодаря работе body-parser проверяем наличие данных в теле запроса, если их нет, то отправляем в ответ ошибку, иначе получаем данные из тела запроса, вызываем через репозиторий обновление документа и отправляем в ответ успешный статус запроса.

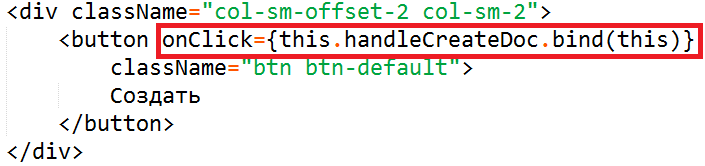
Сохраните изменения, соберите файл bundle.js, запустите веб-сервер и протестируйте изменение документа.



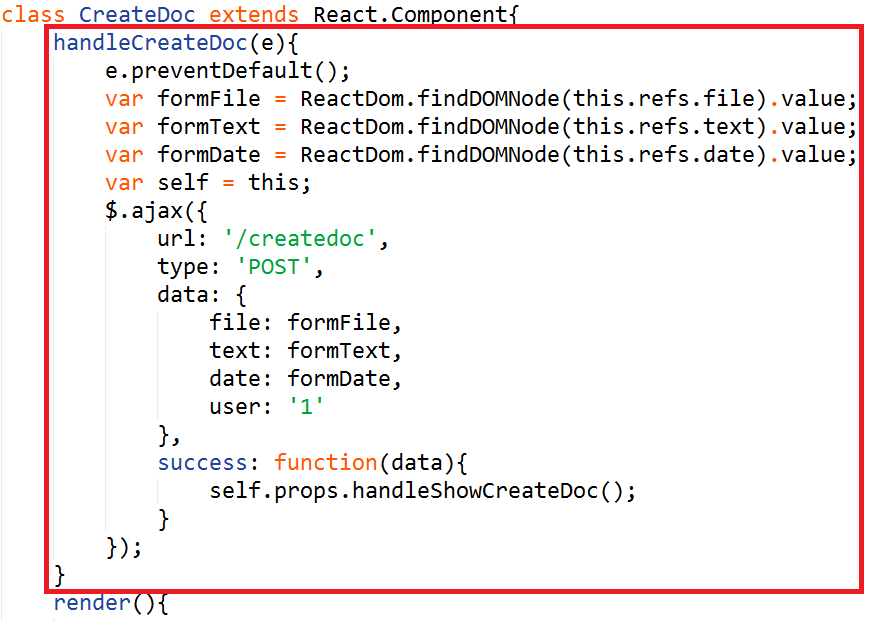
Далее реализуем компонент по созданию нового документа, для этого перейдите в файл CreateDoc.js и введите код формы в методе render по образцу, часть кода формы можно позаимствовать из файла UpdateDoc.js.



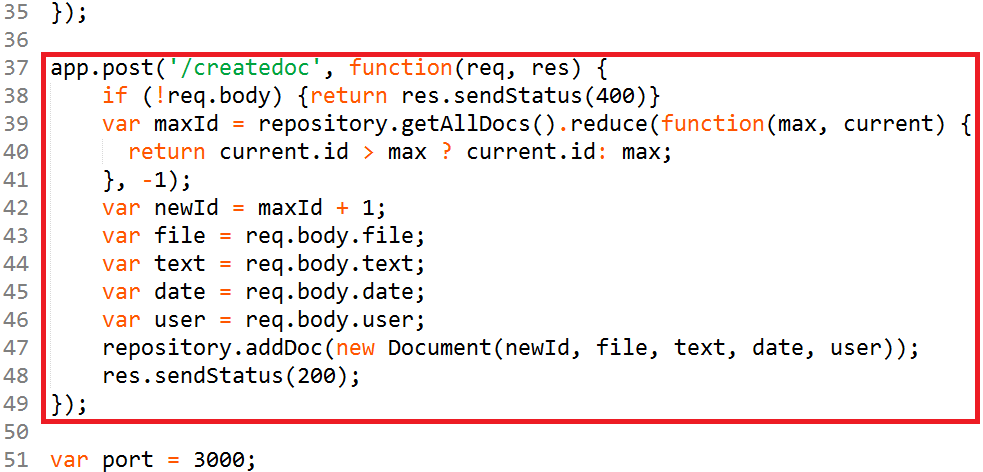
Для кнопки создания добавьте обработчик.



Реализуйте метод handleCreateDoc по образцу.

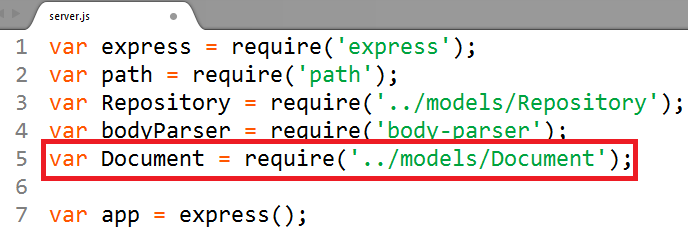


Отправляем имеющиеся данные по адресу createdoc с помощью метода POST, если запрос успешный, то вызываем метод, который скроет компонент CreateDoc и отобразит GetDocs. Сохраните изменения, перейдите в файл server.js и добавьте обработку для соответствующего запроса.

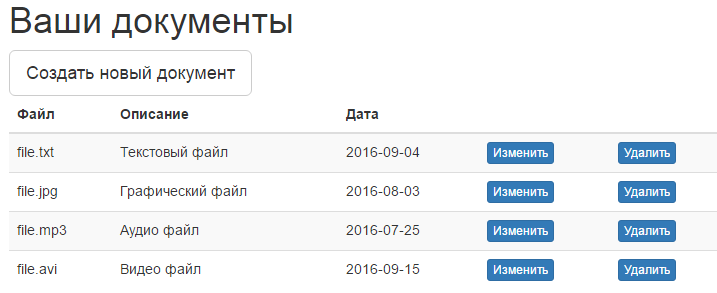


Здесь мы проверяем наличие данных в теле запроса, при их отсутствии отправляем в ответ ошибку. Так как наш репозиторий не умеет генерировать новый ID документа, то реализуем эту логику прямо здесь, вначале находим максимальный ID с помощью функции reduce и увеличиваем его на единицу. Передаем репозиторию в метод addDoc новый объект Document и отправляем в ответе удачный статус запроса.

Так как в коде мы используем конструктор Document, то его необходимо подключить.



Сохраните изменения, соберите bundle.js, запустите веб-сервер и протестируйте работу добавления нового документа.



В следующей работе пойдет речь о том, что такое Redux.