МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Информационные технологии»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

К ВЫПОЛНЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ "СЕРВЕРНЫЕ ИНТЕРНЕТ-ТЕХНОЛОГИИ"

Ростов-на-Дону
ДГТУ
2022

УДК 62

Составители: П.В. Васильев, С.С. Кизим

Методические указания к выполнению контрольной работы по дисциплине «Серверные интернет-технологии» / сост. П.В. Васильев, С.С. Кизим. – Ростов-на-Дону: Донской государственный технический университет, 2022. – 9 с.

Описан алгоритм разработки веб-приложения на основе фреймворка Meteor.

Предназначены для обучающихся направлений подготовки 09.03.03 Прикладная информатика всех форм обучения.

УДК 62

Печатается по решению редакционно-издательского совета
Донского государственного технического университета

Ответственный за выпуск зав. кафедрой «Информационные технологии»,
д-р техн. наук, профессор Б.В. Соболь

В печать 6.05.2022.
Формат 60×84/16. Объем 0,6 усл.п.л.
Тираж 50 экз. Заказ № 149

Издательский центр ДГТУ
Адрес университета и полиграфического предприятия:
344000, г. Ростов-на-Дону, пл. Гагарина,1

ПРАВИЛА ВЫПОЛНЕНИЯ КОНТРОЛЬНОЙ РАБОТЫ

Контрольная работа заключается в выполнении контрольных заданий и оформлении отчета по результатам работы.

Требования к содержательной части контрольной работы.

Контрольная работа по курсу характер письменного ответа на контрольные задания.

Оформление контрольной работы.

Ответ на каждое контрольное задание должен начинаться с формулировки этого задания, далее копии экрана и комментарии по процессу выполнения задания. В конце всех выполненных заданий указываются источники, которые были использованы при выполнении заданий.

После проверки контрольной работы преподавателем с каждым обучающимся проводится собеседование (защита контрольной работы) по охваченным в ней темам.

Контрольное задание № 1 «НАСТРОЙКА РАБОЧЕЙ СРЕДЫ METEOR»

Методические указания.

- Если вы используете Windows: Сначала установите Chocolatey, затем выполните эту команду с помощью командной строки администратора: choco install meteor

- Если вы работаете на Mac OS или Linux, запустите эту команду в своем терминале: curl https://install.meteor.com/ | sh

Рис. 1. Структура веб-приложения

Создадим Meteor приложение с помощью команды: meteor create simple-todos-react. Метеор создаст для Вас все необходимые файлы. Также, проверьте папку /server, в которой Meteor настраивает серверную сторону (Node.js), Вы можете видеть, что /server/main.js инициализирует базу данных MongoDB с некоторыми данными. Вам не нужно устанавливать MongoDB, так как Meteor предоставляет встроенную БД, готовую к использованию.

Теперь Вы можете запустить ваше приложение Meteor, используя команду: meteor run. Метеор будет синхронизировать приложение со всеми изменениями в коде с этого момента.

React код будет находиться в каталоге import/ui, а App.jsx файл является корневым компонентом вашего React To-do приложения. Взгляните на все файлы, созданные Meteor, вам не нужно их сейчас понимать, но хорошо знать, где они находятся.

Контрольное задание № 2 «СОЗДАНИЕ КОМПОНЕНТОВ ПРИЛОЖЕНИЯ»

Методические указания.

Создайте новый файл под названием Task.jsx в вашей папке ui. Этот файл экспортирует компонент React под названием Task, который будет представлять одну задачу в вашем списке дел. Так как этот компонент будет находиться внутри списка, вы возвращаете элемент внутри тега <li>.

imports/ui/Task.jsx

```jsx
import React from 'react';

export const Task = ({ task }) => {
  return <li>{task.text}</li>
};
```

Рис. 2. Компонент Task.jsx

Так как вы еще не подключились к серверу и базе данных, давайте определим некоторые примеры данных, которые вскоре будут использованы для отображения списка задач. Это будет массив, и вы можете называть его задачами.

imports/ui/App.jsx

```jsx
import React from 'react';

const tasks = [
  {_id: 1, text: 'First Task'},
  {_id: 2, text: 'Second Task'},
  {_id: 3, text: 'Third Task'},
];

export const App = () => ...
```

Рис. 3. Примеры задач

Теперь мы можем реализовать некоторую простую логику рендеринга с помощью React. Мы можем использовать наш предыдущий компонент Task для рендеринга элементов списка. В React вы можете использовать { } для написания кода на Javascript между ними. Вы будете использовать функцию .map из объекта Array для итераций над вашими примерами задач.

imports/ui/App.jsx

```
import React from 'react';
import { Task } from './Task';

const tasks = ...;

export const App = () => (
  <div>
    <h1>Welcome to Meteor!</h1>

    <ul>
      { tasks.map(task => <Task key={ task._id } task={ task }/>) }
    </ul>
  </div>
);
```

Рис. 4. Отображение списка задач

Не забудьте добавить свойство key к вашей задаче, иначе React выдаст предупреждение, потому что увидит много компонентов одного.

Контрольное задание №3 «КОЛЛЕКЦИЯ ДАННЫХ»

Методические указания.

Метеор уже установил MongoDB. Для того, чтобы использовать нашу базу данных, нам необходимо создать коллекцию, в которой будут храниться наши документы, в нашем случае - задачи.

Мы можем создать новую коллекцию для хранения наших задач, создав новый файл import/api/TasksCollection.js, который воплощает в жизнь новую коллекцию Mongo и экспортирует ее.

```
import { Mongo } from 'meteor/mongo';
export const TasksCollection = new Mongo.Collection('tasks');
```

Обратите внимание, что мы хранили файл в каталоге imports/api, который является местом для хранения кода, связанного с API, например, публикаций и методов. Вы можете назвать эту папку как хотите. Вы можете удалить файл links.js в этой папке, так как мы не будем использовать эту коллекцию.

Чтобы наша коллекция заработала, вам нужно импортировать ее на сервер. Вы можете использовать импорт "/imorts/api/TasksCollection" или импорт { TasksCollection } из "/imports/api/TasksCollection", если вы собираетесь использовать его в том же файле, но убедитесь, что он импортирован. Теперь легко проверить, есть ли данные в нашей коллекции или нет, в противном случае мы можем легко вставить некоторые данные выборки. Вам не нужно хранить старое содержимое сервера/main.js.

Итак, вы импортируете TasksCollection и добавляете несколько задач по итерации по массиву строк и для каждой строки вызываете функцию для вставки этой строки в качестве нашего текстового поля в документе задачи.

server/main.js

```
import { Meteor } from 'meteor/meteor';
import { TasksCollection } from '/imports/api/TasksCollection';

const insertTask = taskText => TasksCollection.insert({ text: taskText });

Meteor.startup(() => {
  if (TasksCollection.find().count() === 0) {
    [
      'First Task',
      'Second Task',
      'Third Task',
      'Fourth Task',
      'Fifth Task',
      'Sixth Task',
      'Seventh Task'
    ].forEach(insertTask)
  }
});
```

Рис. 5. Код файла server/main.js

imports/ui/App.jsx

```
import React from 'react';
import { useTracker } from 'meteor/react-meteor-data';
import { TasksCollection } from '/imports/api/TasksCollection';
import { Task } from './Task';

export const App = () => {
  const tasks = useTracker(() => TasksCollection.find({}).fetch());

  return (
    <div>
      <h1>Welcome to Meteor!</h1>

      <ul>
        { tasks.map(task => <Task key={ task._id } task={ task }/>) }
      </ul>
    </div>
  );
};
```

Рис. 6. Отображение списка задач с помощью useTracker

Теперь мы отобразим задачи с помощью React Function Component и Hook, называемого useTracker, из пакета под названием react-meteor-data.

Вы можете изменить Ваши данные в MongoDB на сервере, и Ваше приложение будет реагировать и перерендеривать список задач автоматически.

Вы можете подключиться к MongoDB в терминале из папки Вашего приложения или используя Mongo UI клиент, например NoSQLBooster. MongoDB работает на порту 3001.

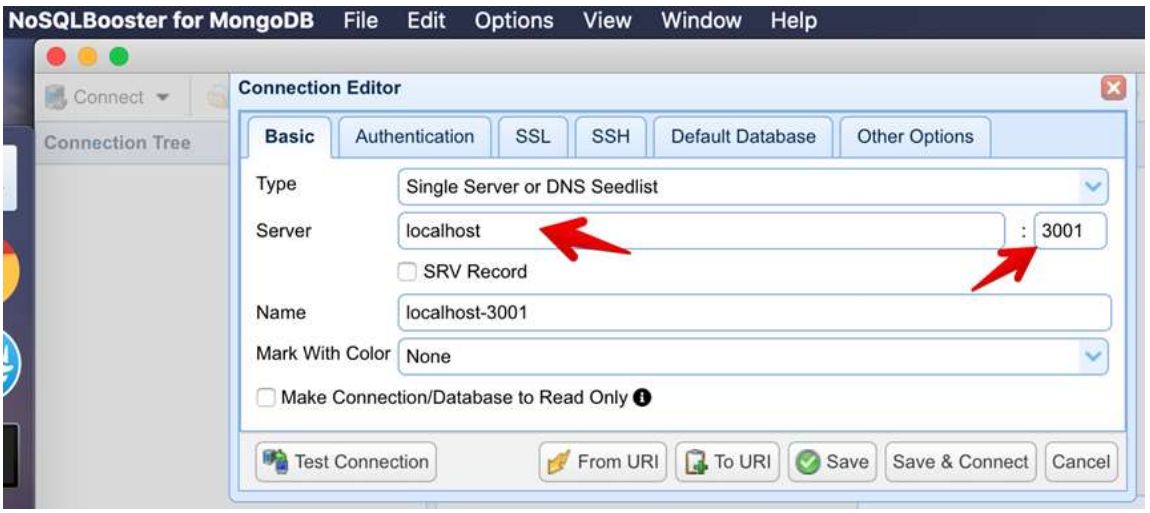

Рис. 7. Подключение к базе данных

Контрольное задание №4 «РАБОТА С ДАННЫМИ»

Методические указания.

Все приложения должны позволять пользователю осуществлять некоторые виды взаимодействия с хранимыми данными. В нашем случае первым типом взаимодействия является создание новых задач. Без него наше приложение To-Do было бы не очень полезным.

Одним из основных способов, с помощью которых пользователь может создавать или редактировать данные на сайте, является использование форм. В большинстве случаев полезно использовать тег <form>, так как он придает смысловое значение элементам внутри него.

Сначала нам нужно создать компонент простой формы для инкапсуляции нашей логики. Как видите, мы настроили useState React Hook. Обратите внимание на деструкцию массива [text, setText], где текст - это хранимое значение, которое мы хотим использовать, в данном случае это будет строка; а setText - это функция, используемая для обновления этого значения. Создайте новый файл TaskForm.jsx в Вашей папке ui. Затем добавьте вывод компонента <TaskForm/> в файл App.jsx.

imports/ui/TaskForm.jsx

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

export const TaskForm = () => {
  const [text, setText] = useState("");

  return (
    <form className="task-form">
      <input
        type="text"
        placeholder="Type to add new tasks"
      />

      <button type="submit">Add Task</button>
    </form>
  );
};
```

Рис. 8. Форма создание задачи

Теперь вы можете прикрепить обработчик отправки к форме, используя событие onSubmit; а также подключить ваш React Hook к событию onChange, присутствующему во входном элементе. Как вы видите, вы используете useState React Hook для хранения значения вашего элемента <input>. Обратите внимание, что вам также нужно установить ваш атрибут значения в текстовую константу, это позволит входному элементу синхронизироваться с нашим хуком.

imports/ui/TaskForm.jsx

```jsx
import React, { useState } from 'react';
import { TasksCollection } from '/imports/api/TasksCollection';

export const TaskForm = () => {
  const [text, setText] = useState("");

  const handleSubmit = e => {
    e.preventDefault();

    if (!text) return;

    TasksCollection.insert({
      text: text.trim(),
      createdAt: new Date()
    });

    setText("");
  };

  return (
    <form className="task-form" onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Type to add new tasks"
        value={text}
        onChange={(e) => setText(e.target.value)}
      />

      <button type="submit">Add Task</button>
    </form>
  );
};
```

Рис. 9. Обработка события submit формы

ЛИТЕРАТУРА

1. Книга “Discover Meteor”: https://github.com/DiscoverMeteor/DiscoverMeteor_Ru.
2. Башлы, П.Н. Современные сетевые технологии: учеб. пособие для вузов / П.Н. Башлы. - М.: ГЛТ , 2006. - 334 с.
3. Кузьменко, Н.Г. Компьютерные сети и сетевые технологии / Н.Г. Кузьменко. - СПб.: Наука и техника, 2013. - 368 с.