Тема-11. React (продолжение)

Что тут есть?
Создание приложения на React; общие библиотеки для JS: axios, uuid, file-saver, underscore, yup, qrcode; библиотеки, специфичные для React: react-hook-form, react-router-dom, react-toastify, react-select; Тестирование кода в JS и на React.

Создание приложения на React

Приложение на React можно создать стандартным способом, установив две библиотеки, упомянутые в предыдущей лекции, через npm: react и react-dom.

Однако, есть более удобный способ создания приложения, в котором будет использоваться React – программа create-react-app.

Эта программа консольная и позволяет развернуть бойлерплейт приложения на React, то есть заготовку, в которой уже есть все настройки сборки и тестирования проекта.

Создание приложения на React

Чтобы создать приложение с помощью create-react-app нужно выполнить следующую команду в папке, где будет приложение:

npx create-react-app my-app

my-app – это название папки, в которой будет проект, оно может быть любым.

Создание приложения на React

npx – это консольная программа, которая поставляется вместе с npm. Она предназначена для унифицированного запуска различных npm-пакетов, таких как create-react-app.

При этом пакет, который вызывается с помощью npx, не будет храниться глобально на компьютере после выполнения команды. Это удобно для пакетов, которые редко используются и когда каждый раз нужна новейшая версия пакета.

Создание приложения на React

Если в процессе разработки приложения нужно будет обновить боилерплейт, на основе которого оно построено (например, из-за появления угроз безопасности), то для этого нужно обновить специальный пакет, который устанавливается автоматически при создании приложения – react-scripts.

Библиотеки для React

В приложениях на React можно использовать разные другие библиотеки.

Некоторые из библиотек никак не зависят от того, используется ли React или какая-то другая подобная библиотека в приложении. Обычно это библиотеки чистых функций, которые не создают элементов интерфейса. Например, это axios, underscore (они будут рассмотрены позже).

Библиотеки для React

Есть библиотеки, которые написаны специально под использование в приложениях на React. Обычно их названия начинаются с или содержат слово react. Например, это библиотеки: react-select, react-hook-form (они будут рассмотрены далее).

Бывают библиотеки, которые изначально написаны для чистого JS, но для которых потом создаются адаптации под React (когда исходную библиотеку с React использовать нельзя). Такие библиотеки обычно встречаются в npm в нескольких вариантах. Например, blockly, react-blockly, vue-blockly и т.д.

Библиотеки для React

Рассмотрим некоторые библиотеки, которые можно использовать не только с React.

axios

Axios – библиотека для отправки запросов на JS (на клиенте или в среде NodeJS).

Установка: npm install axios

Импорт на клиенте: import axios from 'axios';

axios

Отправка get-запросов с помощью axios:

```
axios.get('/api/v1/data').then(response => {
    console.log(response.data);  // данные с сервера
    console.log(response.status);  // статус запроса (код)
    console.log(response.headers);  // заголовки ответа
});
```

Или через await:

```
const response = (await axios.get('/api/v1/data')).data;
```

axios

Отправка post-запросов (put, delete аналогично) с помощью axios (запросы также асинхронные):

```
const data = {
    id: 4,
    timestamp: 33236546463,
    name: '/api/v1/data'
};
const headers = {'content-type': 'application/json'};

axios.post('/api/v1/data', data, headers);
```

axios

Оправка файлов с помощью axios:

```
...
let formData = new FormData();
formData.append('some_file', fileBlob);
formData.append('another_file', fileBlob2);

const headers = {'content-type': 'multipart/form-data'};

axios.post('/api/v1/data', formData, headers);
```

axios

Пример использования axios с React:

```
const [result, setResult] = useState();

useEffect(async () => {
    if (id && versionNum) {
        setResult(
            (await api.get(`/api/v1/result/?id=${id}&num=${versionNum}`)).data
        );
    }
}, [id, versionNum]);

return <span>{result}</span>;
```

uuid

Uuid – простая библиотека, позволяющая генерировать уникальные универсальные идентификаторы по стандарту RFC4122.

Установка: npm install uuid

Пример использования:

```
import { v4 as uuidv4 } from 'uuid';
..
const newUuid = uuidv4();  // '1b9d6bcd-bbfd-4b2d-9b5d-ab8dfbbd4bed'
```

uuid

С недавнего времени библиотека также предоставляет функции валидации uuid.

```
import { version as uuidVersion, validate as uuidValidate } from 'uuid';

// true если строка – uuid, иначе false
const isValid = uuidValidate('109156be-c4fb-41ea-b1b4-efe1671c5836');

// число от 1 до 5 или исключение TypeError, если строка – не uuid
const uuidVer = uuidVersion('109156be-c4fb-41ea-b1b4-efe1671c5836');
```

file-saver

File-saver – библиотека, позволяющая удобно сохранять файлы, созданные на клиенте.

Установка: npm install file-saver

Пример сохранения JSON на компьютере:

```
import {saveAs} from "file-saver";

const filename = prompt("Название файла: ", "Данные");
const blob = new Blob([JSON.stringify(data)], {type: "application/json;charset=utf-8"});
filename && saveAs(blob, filename + ".json");
```

Библиотеки для работы с файлами

Есть и другие библиотеки для работы с разными файлами (например, с архивами) и файлоподобными объектами. Напрмиер: jszip, is-svg, html-to-image.

Но здесь они не будут рассмотрены.

underscore

Underscore – библиотека, которая предоставляет большое количество функций для обработки данных сложной структуры, таких как массивы и объекты.

Если нужна какая-то функция, которой нет в стандартном JS, то вполне вероятно, что она есть в underscore.

Например, тут есть функция разворачивания вложенных массивов в одномерный (flatten), функция zip (как в Python), uniq (очистка от повторов) и многое другое.

yup

Yup – это библиотека для валидации данных сложной формы. Она позволяет написать очень подробную валидацию для сколь угодно сложного объекта. Обычно библиотеку используют для валидации данных форм, но фактически её можно использовать для валидации любого JSON.

Установка: npm install yup

Использование: import * as yup from 'yup';

yup

Пример валидации небольшой формы:

```
let schema = yup.object().shape({
  name: yup.string().required(),
  age: yup.number().required().positive().integer(),
  email: yup.string().email(),
  website: yup.string().url(),
  createdOn: yup.date().default(function () {
    return new Date();
  }),
});

// check validity
const isValid = await schema.isValid({name: 'jimmy', age: 24}); //true
```

yup

Пример условной валидации поля:

```
Yup.object().shape({
    type: Yup.mixed().oneOf([
        "addPolygon",
        "saveToServer",
        "saveToComputer",
    ]),
    uuid: Yup.mixed().when("type", {
        is: val => !["saveToServer", "saveToComputer"].includes(val),
        then: uuidSchema
    }),
});
```

yup

Где uuidSchema – это проверка, что строка является uuid 4 версии:

```
import {version as uuidVersion, validate as uuidValidate} from "uuid";

export const uuidSchema = Yup.string().test(
    "is uuid v4",
    "error",
    val => (uuidValidate(val) && uuidVersion(val) === 4)
);
```

Qrcode with React

Qrcode – библиотека для генерации QR-кодов по ссылкам (фактически, по любым строкам).

Установка: npm install qrcode

Библиотеку можно использовать как отдельно от React, так и с ним. При использовании с React есть некоторая особенность – нужно использовать useRef (или искать элемент в DOM вручную, но это не React-way).

Qrcode with React

Пример использования QRCode с React:

```
import QRCode from "qrcode";
...
const qrcodeCanvasRef = useRef();

useEffect(() => {
    QRCode.toCanvas(qrcodeCanvasRef.current, urlToEncode); //async
}, [urlToEncode]);

return <canvas ref={qrcodeCanvasRef}/>;
```

Библиотеки, специфичные для React

Теперь рассмотрим некоторые библиотеки, созданные или адаптированные под React.

React-hook-form

React-hook-form – это библиотека для построения форм на React. Она позволяет хранить в одном месте все данные формы и делать первичную валидацию.

Установка: npm install react-hook-form

React-hook-form

Пример простой формы на react-hook-form:

```
import {useForm} from 'react-hook-form';

function App() {
  const {register, handleSubmit, formState: {errors}} = useForm();

  return <form onSubmit={handleSubmit((data) => console.log(data))}>
      <input {...register("firstName")}/>
      <input {...register("lastName", {required: true})}/>
      {errors.lastName && <p>Last name is required.</p>}
      <input {...register("age", {pattern: /\d+/})}/>
      {errors.age && <p>Please enter number for age.</p>}
      <input type="submit"/>
  </form>;
}
```

React-hook-form

Какую валидацию можно сделать на react-hook-form:

- required – поля обязательно заполнять;
- min – минимальный порог значения;
- max – максимальный порог значения;
- minLength – минимальная длина значения;
- maxLength – максимальная длина значения;
- pattern – регулярное выражение, которому должно удовлетворять значение;
- validate – кастомная функцию валидации (true/false).

React-hook-form

Можно регистрировать компоненты и поля в отдельных компонентах:

```
const Input = ({label, register, required}) => <>
  <label>{label}</label>
  <input {...register(label, {required})}/>
</>;

const App = () => {
  const {register, handleSubmit} = useForm();
  return <form onSubmit={handleSubmit(() => alert(JSON.stringify(data)))}>
    <Input label="First Name" register={register} required/>
    <input type="submit"/>
  </form>;
};
```

React-hook-form

Если к коду компонента нет доступа (например, он из сторонней библиотеки), можно использовать специальную обертку – Controller, который возьмет на себя управлением состояния поля:

```
import {useForm, Controller} from "react-hook-form";
...
const {control, handleSubmit} = useForm();
...
<input {...register('firstName')}/>
<Controller
  name="lastName"
  control={control}
  render={({field}) => <Input {...field}/>}
/>
...
```

React-hook-form

Если нужно использовать значения полей форм где-то вне формы, можно использовать объект watch:

```
import {useForm} from "react-hook-form";
...
const {register, watch, formState: {errors}, handleSubmit} = useForm();
const currentName = watch("lastName");
...
<input {...register('firstName')}/>
<Controller
  name="lastName"
  control={control}
  render={({field}) => <Input {...field}/>}
/>
...
```

React-router-dom

React-router-dom – библиотека, которая позволяет сделать маршрутизацию по страницам на клиенте без перезагрузки страницы (то есть, без полной перезагрузки приложения).

Установка: npm install react-router-dom

Здесь будет рассматриваться версия 5, новейшая может отличаться

React-router-dom

Пример маршрутизации:

```
import {BrowserRouter, Switch, Route} from "react-router-dom";
...
<BrowserRouter>
   <Switch>
      <Route path="/research" component = {Research}/>
      <Route path="/store" component = {Store}/>
      <Route path="/" component = {StudioWithProviders}/>
   </Switch>
</BrowserRouter>
```

React-router-dom

Маршруты можно параметризовать:

```
...
<Route path='/account/:id' component={Account}/>

...

import {useParams} from "react-router-dom";

function Account() {
    const {id} = useParams();
...
```

React-router-dom

Можно получить доступ к истории переходов на запись с помощью хука useHistory (для чтения текущего состояния можно использовать useLocation):

```
import {useHistory} from "react-router-dom";
...

function Main() {
    const history = useHistory();
    ...
    const openProductPage = () => history.push(`/product/${id}`);
```

React-toastify

React-toastify – библиотека, предоставляющая красивые всплывающие окошки (не модальные), с помощью которых можно выводить пользователю предупреждения, ошибки и другую информацию.

Установка: npm install react-toastify

React-toastify

Пример использования React-toastify:

```
import {ToastContainer, toast} from "react-toastify";
...
const doAction => toast.success("Успешно");
...
return
  ...
  <ToastContainer/> {/*контейнер нужно отрендерить в
приложении один раз*/}
  ...
```

React-toastify

Есть несколько видов окошек по умолчанию (они различаются только по внешнему виду):

```
toast.success("Text");
toast.info("Text");
toast.warn("Text");
toast.error("Text");
```

React-toastify

Окошки можно кастомизовать глобально (параметры применятся для всех окошек) или локально (для конкретного окошка):

```
<ToastContainer
   position="top-right"
   autoClose={5000}
   hideProgressBar={false}
   newestOnTop={false}
   closeOnClick
   rtl={false}
   pauseOnFocusLoss
   draggable
   pauseOnHover
   theme="light"
/>
```

```
toast('     Wow so easy!',
{
   position: "top-right",
   autoClose: 5000,
   hideProgressBar:
false,
   closeOnClick: true,
   pauseOnHover: true,
   draggable: true,
   progress: undefined,
   theme: "light",
});
```

React-select

React-select – библиотека, позволяющая удобно создавать элементы с вариантами выбора (так называемые „комбобоксы“).

Установка: npm install react-select

React-select

Простейший способ использования react-select:

```
import Select from 'react-select';

const options = [
  { value: 'chocolate', label: 'Chocolate' },
  { value: 'strawberry', label: 'Strawberry' },
  { value: 'vanilla', label: 'Vanilla' }
];

const MyComponent = () => {
  return <Select options={options} isMulti/>;
};
```

React-select

Каждый из Select (обычный и асинхронный) имеет также два варианта – с возможностью создания вариантов на ходу и без неё. Пример:

```
import CreatableSelect from 'react-select/creatable';

const MyComponent = () => {
  return <CreatableSelect isClearable options={colourOptions} />;
};
```

Для создания асихронного creatable Select нужно импортировать:
import AsyncCreatableSelect from 'react-select/async-creatable';
В остальном работа как с async.

React-select

Можно делать асинхронные комбобоксы (варианты подгружаются с сервера или как-то ещё не мгновенно):

```
import AsyncSelect from 'react-select/async';

const loadOptions = (inputValue, callback) => {
  setTimeout(() => {
    callback(loadValues(inputValue));
  }, 1000);
};

const MyComponent = () => {
  return <AsyncSelect cacheOptions loadOptions={loadOptions} defaultOptions/>
};
```

Тестирование

При создании приложения на React через create-react-app, сразу устанавливаются все необходимые инструменты для тестирования кода (по умолчанию используется библиотека Jest).

Можно писать тесты как для чистых функций, так и для компонентов React.

Тестирование

Файлы с тестами записываются в файлы формата name.test.js.

Пример:
- functions.js – файл с функциями;
- functions.test.js – файл с тестами к функциям из файла functions.js.

Тестирование

Формат записи теста:

```
test("Название теста", () => {
    //код теста
});
```

Тестирование

При тестировании чистых функций обычно перечисляются пары наборов входных и выходных данных. Пример тестирования чистой функции:

```
import {sum} from './functions.js';

test('sum function test', () => {
    expect(sum(2, 6)).toEqual(8);
    expect(sum(10, 7)).toEqual(17);
    expect(sum(-12, 2)).toEqual(-10);
    expect(sum(-20, 0)).toEqual(-20);
    expect(sum(0, 28)).toEqual(28);
    expect(sum(0, 0)).toEqual(0);
});
```

Тестирование

Функции test(), expect() импортировать не нужно, они существуют в глобальной области видимости при тестировании.

Вместо test() можно писать it(). Это тоже самое. Запись it() немного короче. Её сущствование связано с тем, что есть ещё одна функция xit(), которая позволяет временно исключить тест (он не будет проеряться).

Есть и более сложные тесты. Ссылка на Jest в конце.

Тестирование

Тестирование компонентов делается похожим образом, нужно сравнить результат рендера компонента с некоторым эталоном.

Результат рендера компонента получается с помощью дополнительной библиотеки – react-test-renderer.

Эталонное значение, или снапшот компонента, автоматически создается при первом запуске тестов (то есть нужно убедиться, что при написании теста компонент написан правильно). Хранятся снапшоты в папке и именем __snapshot__, которая находится рядом с тестами (обычно её не нужно трогать).

Тестирование

Пример теста компонента:

```
import renderer from 'react-test-renderer';
import CustomText from "./CustomText";

it('CustomText render test', () => {
  const result = renderer.create(<CustomText
      id = {"f34297e4-095b-4cdd-81c1-b65ec9b7b9d5"}
      className = {"figure"}
      text = {"Some text      "}
      fontSize = {16}
      scaleCoefficient = {1}
  />).toJSON();
  expect(result).toMatchSnapshot();
});
```

Тестирование

Чтобы запустить тесты нужно выполнить команду: npm test.

Эта команда была автоматически добавлена к приложению, когда выполнялось создание проекта через create-react-app.

Полезные ссылки

- https://create-react-app.dev/ – **сайт create-react-app;**
- https://medium.com/devschacht/introducing-npx-an-npm-package-runner-a72a658cd9e6 **– об npx;**
- https://www.npmjs.com/package/axios **– библиотека axios;**
- https://www.npmjs.com/package/uuid **– библиотека uuid;**
- https://www.npmjs.com/package/file-saver **– библиотека file-saver;**
- https://underscorejs.org/ **– библиотека underscore;**
- https://www.npmjs.com/package/yup **– библиотека yup;**
- https://github.com/soldair/node-qrcode **– библиотека QRcode;**

Полезные ссылки

- https://react-hook-form.com/ – **библиотека react-hook-form;**
- https://reactrouter.com/en/main – **библиотека react-router-dom;**
- https://www.npmjs.com/package/react-toastify – **библиотека toastify;**
- https://react-select.com/home – **библиотека react-select;**
- https://jestjs.io/ – **сайт Jest;**
- https://reactjs.org/docs/test-renderer.html – **про react-test-renderer.**

Дальше Docker!