

Прerequisites

Для начала работы на вашем компьютере должны быть:

1. Установлен Visual Studio Code. Скачать можно тут <https://code.visualstudio.com/>
2. Установлен Node.js. Скачиваем тут <https://nodejs.org/en/>
3. Установлен npm или yarn. Скачиваем тут <https://www.npmjs.com/>
4. Хорошее настроение, время, налитый горячий чай или кофе.

Начало работы

1. Открываем проект в Visual Studio Code.
2. Запускаем `npm install` в терминале (`yarn install`)
3. В `package.json` есть скрипт `start`. Запускаем его (`npm run start` в терминале)
4. Открываем наш проект на порту 8080 – <http://localhost:8080>
5. Видим белый экран, радуемся

Начальный HTML

Дальнейшая разработка будет происходить в src папке. Добавим статический код в src/App.jsx - вместо <div></div> вставим следующий код:

```
<div className = "container">
  <header>
    <ul className="menu">
      <li><a href="#">Все новости</a></li>

      <li><a href="#">Создать</a></li>

    </ul>
  </header>
  <div className="top-container">
    <h1>Мы начинаем наш Воркшоп</h1>
    <div className="description">Для кого? Для чего?</div>
    <div className = "green-line" />
  </div>

  <div className="news-container">
    <p>
      <span className="title">.NET Meetup #23</span>-
      <span className="dateTime">Sept 12 at 07:00PM </span>
    </p>
    <h3>Пришло время изучить что -то новенькое, и с этим нам помог Сергей Кривошеин, сделав обзор языка F#.</h3>
    <p>
      Держу пари, многие из вас, особенно кто пишет на C#, знают, что есть такой замечательный язык. На этом все знания о F#, возможно, заканчиваются. А вот Сергей по долгу службы и развития для использует этот язык. И в Карасике он поделился с нами, как же на нем писать, рассказал об основных синтаксических конструкциях, сравнил с C#, а еще поведал о всяких плюсах и минусах.

    </p>
    <p>
      И конечно же, сертификат Сергею вручили – с ним все ж приятней всякие и интересные штуки изучать.

    </p>
```

```

    <p>Смотрим бомбический видосик с попкорном и чайком!</p>
    <div className = "blue-line" />
  </div>
  <div className="news-container">
    <p>
      <span className="title">CoderGames </span>-
    <span className="dateTime">Sept 5 at 06:00PM</span>
    </p>

    <h3>Начало месяца ознаменовалось очередными CoderGames. </h3>
    <p>
      И на этот раз ребята сражались в знаменитую игру 2048. Всего участвовал 2
      1 человек. Вот тут можно посмотреть фоточки, результаты и анонс. А пока поздра
      вим победителей - Евгения Быкова, который занял первое место, а также Ольгу Г
      азизову и Антона Канопкина.

    </p>
    <p>
      И конечно же, скажем спасибо организаторам и судьям за их поддержку. А в
      сем участникам - молодцы!

    </p>
    <p>Кстати, тут недавно начался марафон CoderGames, который закончится в
    октябре. Мы вам о нем в следующий раз расскажем. </p>

    <div className = "green-line" />
  </div>
  <footer>
    <div>Парапарам</div>
    <img src ="https://bipbap.ru/wp-content/uploads/2017/12/65620375-
    6b2b57fa5c7189ba4e3841d592bd5fc1-800-640x426.jpg" />
    </footer>
  </div>

```

Давайте сделаем несколько компонентов – Container.jsx, ItemComponent.jsx, News.jsx.

Container.jsx.

В нем мы будем хранить хедер и футер. Дальше именно этот компонент будет отвечать – а что нам надо показать.

```
import React, { Component } from 'react';
import News from "../News";

class Container extends Component {
  constructor(props) {
    super(props);
  }

  render () {
    return (<div className = "container">
      <header>
        <ul className="menu">
          <li><a href="#">Все новости</a></li>
          <li><a href="#">Создать</a></li>
        </ul>
      </header>
      <div className="top-container">
        <h1>Мы начинаем наш Воркшоп</h1>
        <div className="description">Для кого? Для чего?</div>
        <div className = "green-line" />
      </div>
      <News />
      <footer>
        <div>Парапарам</div>
        <img src ="https://bipbar.ru/wp-content/uploads/2017/12/65620375-6b2b57fa5c7189ba4e3841d592bd5fc1-800-640x426.jpg" />
      </footer>
    </div>);
  }
}
```

```
        </footer>

    </div>);
    }
};

export default Container;
```

News.jsx

Кроме того, давайте добавим компонент со списком новостей – News, где будет лежать список новостей. Добавим в файл константу news = [...] где ... - это массив объектов из файла db.json.

```
import React, { Component } from 'react';
import ItemComponent from "../ItemComponent";

export class News extends Component {
  constructor() {
    super();
  }

  render() {
    return (<>
      {news.map((item, index) => <ItemComponent
        key={item.id}
        index={index}
        item={item} />)}
    </>)
  }
};

export default News;
```

Вместо списка новостей будем использовать наш ItemComponent.jsx:

```
{
  news.map((item, index) => <ItemComponent
    item={item} key={item.id} index={index} />)
}
```

ItemComponent.jsx

Этот компонент будет принимать в качестве props одну новость - item и отображать ее.

```
import React from 'react';

export const ItemComponent = ({ item, index }) => {

  return(<div className="news-container">
    <p>
      <span className="title">{item.Title}</span>-
      <span className="dateTime">{item.Date} </span>
    </p>
    <h3>{item.Info}</h3>
    <p>
      {item.Description}</p>
    {item.Image && <img src ={item.Image} /> }
    { index % 2 ?
      <div className = "blue-line" /> :
      <div className = "green-line" /> }
    </div>);
};

export default ItemComponent;
```

Добавим в ItemComponent проверку типов, а также проверку что новость действительно существует:

```
import PropTypes from "prop-types";

export const ItemComponent = ({ item, index }) => {
  if (!item)
    return null;
  ...
}

ItemComponent.defaultProps = { index: 0 }

ItemComponent.propTypes = {
  item: PropTypes.object,
  index: PropTypes.number
}
```


Сделаем News.jsx умным компонентом и добавим ему state. Теперь у него есть свое собственное состояние, которое меняется внутри него.

```
import React, { Component } from 'react';
import ItemComponent from "../ItemComponent";

class News extends Component {
  constructor(props) {
    super(props);
    this.state = {
      news: news
    }
  }

  render () {
    return (
      <>
        {this.state.news.map((item, index) => <Item item={item} key={item.id} index={index} />)}
      </>
    );
  }
};

export default News;
```

Жизненный цикл компонента.

Первоначальный рендеринг компонента в DOM называется «монтирование» (mounting). Каждый раз, когда DOM-узел, созданный компонентом, удаляется, происходит «размонтирование» (unmounting).

Чтобы сделать какие-то свои действия в это время есть две функции - `componentDidMount`, `componentDidUnmount`.

Метод `componentDidMount()` запускается после того, как компонент отрендерился в DOM.

В редких случаях может потребоваться позволить компоненту спрятать себя, ничего не показывать. Чтобы этого добиться, верните `null`. Значение вам вернуть надо обязательно, возврат `null` из метода `render` никак не влияет на срабатывание методов жизненного цикла компонента.

Например, `componentDidUpdate` будет всё равно вызван.

`componentDidUpdate` вызывается, если `props` поменялись. При этом в качестве параметров в функцию приходит предыдущие значения пропсов. Метод срабатывает до того, как обновленное значение будет применено.

При монтировании компонента сделаем загрузку нужных нам данных.

```
componentDidMount () {  
  let result = fetch("http://localhost:4000/News")  
  .then(response => response.json())  
  .then(json => this.setState({ news: json, isLoading: false }));  
}
```

Чтобы показывать что-то, пока грузиться, сделаем `Loader.jsx`:

```
render () {  
  if (this.state.isLoading) {  
    return (<div className = "container">Loading</div>)  
  }  
  
  return (<>...</>);  
}
```

А теперь добавим форму, которая позволит нам добавлять новую новость.

Форма работает так же, как и в обычном js – ничего не отправляет и редиректит дальше.

Для начала самый простой вариант:

```
import React, { Component } from 'react';

class AddItem extends Component {
  constructor(props) {
    super(props);
  }

  render () {
    return (<div className="form-container">
      <form>

        <div className = "row">
          <label>Title:
            <input type="text" name="title" />
          </label>
        </div>

        <div className = "row">
          <label>Description:
            <input type="text" name="description"
/>

          </label>
        </div>

        <div className = "row">
          <label>Info:
            <input type="text" name="info" />
```

```

        </label>
      </div>

      <div className = "row">
        <label>Image(URL):
          <input type="text" name="image" />
        </label>
      </div>
      <div className = "row">
        <button>Send</button>
      </div>
    </form>

    </div>)
  }
}

export default AddItem;

```

Сделаем компонент контролируемым:

Работать с формой можно двумя способами – через контролируемые объекты (рекомендуемая) и неконтролируемые (через ref)

Возьмем второй вариант.

Картинку можно взять <http://images6.fanpop.com/image/photos/34200000/Kawaii-kawaii-anime-34255208-320-375.jpg> отсюда или любую другую.

```
import React, { Component } from 'react';

class AddItem extends Component {
  constructor(props) {
    super(props);
    let currentDate = new Date().toISOString().slice(0
, 10);

    this.state = {
      isSaving: false,
      item: {
        Title: "",
        Date: currentDate,
        Description: "",
        Info: "",
        Image: ""
      }
    }

    this.handleSubmit = this.handleSubmit.bind(this);
    this.onChange = this.onChange.bind(this);
  }

  onChange (e) {    }
```

```

handleSubmit (e) {}

render () {
  return (<div className="form-container">
    <form onSubmit={this.handleSubmit}>

      <div className = "row">
        <label>Title:
        <input type="text" name="Title" value = {this.state.Title} onChange={this.onChange}/>
        </label>
      </div>

      <div className = "row">
        <label>Description:
        <textarea type="text" name="Description" value = {this.state.Description} onChange={this.onChange} />
        </label>
      </div>

      <div className = "row">
        <label>Info:
        <textarea type="text" name="Info" value = {this.state.Info} onChange={this.onChange} />
        </label>
      </div>

      <div className = "row">
        <label>Image(URL):
        <input type="text" name="Image" value = {this.state.Image} onChange={this.onChange} />
        </label>
      </div>
    </form>
  </div>);
}

```

```
        <div className = "row">
          <button>Send</button>
        </div>
      </form>

    </div>)
  }
}

export default AddItem;
```

Добавим обработчик:

```
    onChange (e) {  
      let value = e.target.value;  
      let name = e.target.name;  
  
      this.setState(prevState => {  
        return {  
          item : { ...prevState.item, [name]: value  
        }  
      });  
    }  
  }
```

Отправим форму на сервер:

```
    handleSubmit (e) {  
      let data = this.state.item;  
  
      fetch("http://localhost:4000/News", {  
        method: 'POST',  
        headers: {  
          'Content-Type': 'application/json'  
        },  
        body: JSON.stringify(data)  
      })  
      .then (response => console.log(response.status))  
      .catch(ex => {  
        e.preventDefault()  
      });  
    }  
  }
```


Добавление Формы

Мы хотим добавить страницу с формой, роутера пока у нас нет, поэтому сделаем кривой роутинг.

```
import React, { Component } from 'react';
import News from "../News";
import AddItem from "../AddItem";

class Container extends Component {
  constructor(props) {
    super(props);

    this.state = {
      currentPage: pages.list
    }
  }
  setPage (pageName) {
    this.setState({ currentPage: pageName });
  }
  render () {
    return (<div className = "container">
      <header>
        <ul className="menu">
          <li><a href="#" onClick= {() => this.setPa
ge(pages.list)}>Все новости</a></li>
          <li><a href="#" onClick= {() => this.setPa
ge(pages.add)}>Создать</a></li>
        </ul>
      </header>
      <div className="top-container">
        <h1>Мы начинаем наш Воркшоп</h1>

```

```

        <div className="description">Для кого? Для чего?</div>
        <div className = "green-line" />
    </div>

    { this.state.currentPage === pages.list && <News /
  >}
    { this.state.currentPage === pages.add && <AddItem
  />}

    <footer>
        <div>Папапапам</div>
        <img src ="https://bipbap.ru/wp-
content/uploads/2017/12/65620375-
6b2b57fa5c7189ba4e3841d592bd5fc1-800-640x426.jpg" />
    </footer>

    </div>));
  }
};

const pages = { list: "list", add: "add" };

export default Container;

```

В зависимости от того куда тыкнули у нас будет либо показываться список, либо пустой элемент AddItem.jsx.

Предохранители

Предохранители это компоненты React, которые **отлавливают ошибки JavaScript в любом месте деревьев их дочерних компонентов, сохраняют их в журнале ошибок и выводят запасной UI** вместо рухнувшего дерева компонентов.

Напишем свой и поставим на самый верх.

```
import React from "react";

export class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }
  componentDidCatch () {
    this.setState({ hasError: true });
  }

  static getDerivedStateFromError(error) {
    return { hasError: true };
  }

  render () {
    if (this.state.hasError) {
      return (
        <div>
```

```

        <div>YYYYYnc.</div>
      </div>
    );
  }
  else
    return this.props.children;
}
}

```

```

ErrorBoundary.propTypes = {
  children: PropTypes.any
};

```

```

export default ErrorBoundary;

```

Тестирование.

Для тестов мы будем использовать связку Jest + Enzyme. У нас будут модульные тесты, компоненты мы будем тестировать с помощью Enzyme в окружении Jest без полного рендеринга. Такие тесты быстрые и помогают проверить каждый компонент без вложенных.

```
Npm install --save-dev enzyme jest react-test-renderer enzyme-adapter-react-16
```

Для начала настроим Jest.

Добавим файл jest.config.js. Конфигурацию можно задать и в package.json, она должна быть в самом начале – но лучше пусть лежит отдельно.

Первый тест:

```
import React from "react";
import ItemComponent from "../src/ItemComponent";

const item = {
  "id": 1,
  "Title": "Essex Police: 39 people found dead in lorry container" ,
  "Date": "2019-08-08",
  "Description": "DescTest",
  "Info": "InfoTest",
};

describe("Test ItemComponent component", () => {
  it("should show correctly", () => {
    const wrapper = shallow(<ItemComponent item={item} />);

    const title = wrapper.find(".title");
    expect(title.length).toBe(1);
    expect(title.text()).toBe(item.Title);
  });
});
```

```
    const image = wrapper.find("img");  
    expect(image.length).toBe(0);  
  });  
});
```

Тестирование fetch и начального состояния:

```
import React from "react";
import News from "../src/News";

describe("Test News component", () => {
  it('fetches data from server when server returns a successful response', done => {
    const mockSuccessResponse = [{ Id: "1", Title: "Test", Info: "Test 2"}];
    const mockJsonPromise = Promise.resolve(mockSuccessResponse);
    const mockFetchPromise = Promise.resolve({
      json: () => mockJsonPromise,
    });

    global.fetch = jest.fn().mockImplementation(() => mockFetchPromise);

    const wrapper = shallow(<News />);

    expect(global.fetch).toHaveBeenCalledTimes(1);
    expect(global.fetch).toHaveBeenCalledWith('http://localhost:4000/News');

    process.nextTick(() => { // 6
      expect(wrapper.state()).toEqual({
        news: mockSuccessResponse,
        isLoading: false
      });

      global.fetch.mockClear();
      delete global.fetch;
    });
  });
});
```

```
        done();  
    });  
});  
});
```


Тестирование состояния

```
import React from "react";
import Container from "../src/Container";

describe("Test Container component", () => {
  it("setPage should set state correctly", () => {
    const wrapper = shallow(<Container />);

    let news = wrapper.find("News");
    expect(news.length).toBe(1);

    let addItem = wrapper.find("AddItem");
    expect(addItem.length).toBe(0);

    wrapper.instance().setPage("add");
    wrapper.update();

    news = wrapper.find("News");
    expect(news.length).toBe(0);

    addItem = wrapper.find("AddItem");
    expect(addItem.length).toBe(1);
  });
});
```