1. Компонент React – это класс, который наследуется от класса React.Component (как вы увидите позже, это не единственный способ создать компонент)
2. Функция render возвращает нечто (рассмотрим позже), что будет отрисовано в браузере. Класс-компонент без функции render существовать не может, это его интерфейс

**JSX**

JSX – это похожее на XML-разметку расширение для JavaScript, созданное специально для задач React. React из коробки поставляется с набором компонентов, которые полностью повторяют HTML. По большей части синтаксис и структура JSX и HTML совпадают, но есть некоторые важные различия:

1. Так как это похожий на XML синтаксис, одиночные теги в JSX должны быть закрыты: <hr />
2. Вместо атрибута class в JSX используется имя свойства в DOM: className, так как class — это зарезервированное слово для создания классов

**const** name **=** 'Eva';

**const** cname **=** 'container';

**const** vdom1 **=** <div>Hello, {name}</div>;

**const** vdom2 **=** <div>Hello, {name.repeat(3)}</div>;

**const** vdom3 **=** <div className={cname}>Hello!</div>;

Как видно, вставка (по сути – интерполяция) происходит за счёт использования фигурных скобок, причём внутри них может быть любое выражение. Эта схема работает одинаково как для содержимого тегов, так и для атрибутов.

export default ({ title, text }) => {

  if (!title && !text) {

    return null;

  }

  const titleDom = title && <h4 className="card-title">{title}</h4>;

  const textDom = text && <p className="card-text">{text}</p>;

  return (

    <div className="card">

      <div className="card-body">

        {titleDom}

        {textDom}

      </div>

    </div>

  );

};

ПРОПСЫ!!!

import React from 'react';

// BEGIN (write your solution here)

export default class Card extends React.Component {

  render() {

    const { title, text } = this.props;

    return (

      <div className="card">

        <div className="card-body">

          <h4 className="card-title">{title}</h4>

          <p className="card-text">{text}</p>

        </div>

      </div>

    );

  }

}

Card.defaultProps = {

  text: 'text',

  title: 'title'

};

class List extends React.Component {

render() {

const { data } = this.props

return (

<ul>

{data.map(item => <li>{item.name}</li>)}

</ul>

);

}

}

const items = [

{ name: 'first' },

{ name: 'second' },

{ name: 'three'}

];

const mountNode = document.getElementById('react-root');

const root = ReactDOM.createRoot(mountNode);

root.render(<List data={items} />);

## Использование React.Fragment

class Columns extends React.Component {

render() {

return (

<React.Fragment> <td>Привет</td>

<td>Мир</td>

</React.Fragment> );

}

}

Результатом будет правильный вывод <Table />:

<table>

<tr>

<td>Привет</td>

<td>Мир</td>

</tr>

</table>

### Сокращённая запись

Существует сокращённая запись объявления фрагментов. Она выглядит как пустые теги:

class Columns extends React.Component {

render() {

return (

<> <td>Привет</td>

<td>Мир</td>

</> );

}

}

## Стили

Совсем по-другому работает атрибут style. Если в HTML это обычная строка, то в JSX только объект.

class Component extends React.Component {

render() {

const divStyle = {

color: 'blue',

fontSize: '50px',

};

return <div style={divStyle}>Hello World!</div>;

}

}

export default class Definitions extends React.Component {

  render() {

    const { data } = this.props;

    if (data.length === 0) {

      return null;

    }

    const tags = data.map(({ dt, dd, id }) => (

      <React.Fragment key={id}>

        <dt>{dt}</dt>

        <dd>{dd}</dd>

      </React.Fragment>

    ));

    return (

      <dl>

        {tags}

      </dl>

    );

  }

}

ТОЖЕ САМОЕ

export default class Definitions extends React.Component {

  renderList() {

    const { data } = this.props;

    const tags = data.map(({ dt, dd, id }) => (

      <React.Fragment key={id}>

        <dt>{dt}</dt>

        <dd>{dd}</dd>

      </React.Fragment>

    ));

    return tags;

  }

  render() {

    const { data } = this.props;

    if (data.length === 0) {

      return null;

    }

    return (

      <dl>

        {this.renderList()}

      </dl>

    );

  }

}

Classnames

<https://github.com/JedWatson/classnames>

import cn from 'classnames';

import React from 'react';

// BEGIN (write your solution here)

 export default class Alert extends React.Component {

  render () {

    const { type, text } = this.props;

    const alertClass = cn('alert', `alert-${type}`);

    return (

      <div className={alertClass} role="alert">{text}</div>

    )

  }

};

CHILDREN

class Alert extends React.Component {

render() {

const { children } = this.props;

return (<div className="alert alert-primary">

{children}

</div>);

}

}

const vdom = <Alert>

<p>Aww yeah, you successfully read this important alert message. This example text is going to run a bit longer so that you can see how spacing within an alert works with this kind of content.</p>

<hr />

<p class="mb-0">Whenever you need to, be sure to use margin utilities to keep things nice and tidy.</p>

</Alert>;

const mountNode = document.getElementById('react-root');

const root = ReactDOM.createRoot(mountNode);

root.render(vdom);

Обратите внимание на то, что компонент стал использоваться как парный тег в JSX:

**const** vdom **=** (

<**Alert**>

<p>Whenever you need to, be sure to use margin utilities to keep things nice and tidy.</p>

</**Alert**>

);

Все, что находится между открывающим и закрывающим тегом, попадает внутрь пропса *children*.

**Композиция компонентов**

Иерархия компонентов во многом закладывает структуру приложения. То, как вы распределите приложение на компоненты определит кодовую базу вашего проекта.

Есть несколько советов как разбивать приложение на компоненты:

* выделите части интерфейса в приложении, которые имеют осмысленный функционал. Например: навигационная панель, боковое меню, окно чата и т.д. Каждую такую часть приложения можно выделить в отдельный компонент. Такое разбиение интерфейса на состовляющие части заложит основу компонентной базы
* если какие-то элементы повторяются, то их тоже лучше выделить в отдельный компонент для переиспользования. Например это могут быть Button и InputField
* рассматривайте компоненты как чистые функции. Пропсы, как и параметры в чистых функциях, не должны изменяться в компонентах, в React это важное правило

export default class ListGroup extends React.Component {

  render() {

    const { children } = this.props;

    return (

      <ul className="list-group">

        {React.Children.map(children, (child) => <li className="list-group-item">{child}</li>)}

      </ul>

    )

  }

}

import cn from 'classnames';

import React from 'react';

// BEGIN (write your solution here)

export default class BtnGroup extends React.Component {

  constructor(props) {

    super(props);

    this.state = { active: null };

  }

  selectLeft = () => this.setActive('left');

  selectRight = () => this.setActive('right');

  setActive = (active) => {

    this.setState({ active });

  }

  render() {

    const { active } = this.state;

    const sharedClasses = {

      btn: true,

      'btn-secondary': true,

    };

    const leftButtonClass = {

      ...sharedClasses,

      left: true,

      active: active === 'left',

    };

    const rightButtonClass = {

      ...sharedClasses,

      right: true,

      active: active === 'right',

    };

    return (

      <div className="btn-group" role="group">

        <button type="button" onClick={this.selectLeft} className={cn(leftButtonClass)}>Left</button>

        <button type="button" onClick={this.selectRight} className={cn(rightButtonClass)}>Right</button>

      </div>

    );

  }

}

// END

import React from 'react';

import cn from 'classnames';

// BEGIN (write your solution here)

import uniqueId from 'lodash/uniqueId';

export default class Carousel extends React.Component {

  constructor(props) {

    super(props);

    const { images } = props;

    this.state = {

      currentIdx: 0,

      // картинки могут повторяться, key должен быть уникальным

      images: images.map((url) => ({ url, key: uniqueId() })),

    };

  }

  setNext = () => {

    const { currentIdx, images } = this.state;

    const nextIdx = ((currentIdx + 1) % images.length);

    this.setState({ currentIdx: nextIdx });

  };

  setPrev = () => {

    const { currentIdx, images } = this.state;

    const prevIdx = ((currentIdx + (images.length - 1)) % images.length);

    this.setState({ currentIdx: prevIdx });

  };

  renderItems() {

    const { currentIdx, images } = this.state;

    return images.map(({ url, key }, id) => {

      const classes = cn({

        'carousel-item': true,

        active: currentIdx === id,

      });

      return (

        <div key={key} className={classes}>

          <img alt="" className="d-block w-100" src={url} />

        </div>

      );

    });

  }

  render() {

    return (

      <div id="carousel" className="carousel slide" data-bs-ride="carousel">

        <div className="carousel-inner">

          {this.renderItems()}

        </div>

        <button data-bs-target="#carousel" className="carousel-control-prev" onClick={this.setPrev} type="button" data-bs-slide="prev">

          <span className="carousel-control-prev-icon" aria-hidden="true" />

          <span className="visually-hidden">Previous</span>

        </button>

        <button data-bs-target="#carousel" className="carousel-control-next" onClick={this.setNext} type="button" data-bs-slide="next">

          <span className="carousel-control-next-icon" aria-hidden="true" />

          <span className="visually-hidden">Next</span>

        </button>

      </div>

    );

  }

}

import React from 'react';

import cn from 'classnames';

export default class Collapse extends React.Component {

  constructor(props) {

    super(props);

    const { opened } = props;

    this.state = { opened };

  }

  handleToggle = (e) => {

    e.preventDefault();

    this.setState(({ opened }) => ({ opened: !opened }));

  };

  itemRender() {

    const { opened } = this.state;

    const { text } = this.props;

    const classes = cn({

      collapse: true,

      show: opened,

    });

    return (

      <div className={classes}>

        <div className="card card-body">

          {text}

        </div>

      </div>

    );

  }

  render() {

    const { opened } = this.state;

    // const { text } = this.props;

    return (

      <div>

        <p>

          <a

            className="btn btn-primary"

            onClick={this.handleToggle}

            data-bs-toggle="collapse"

            href="#"

            role="button"

            aria-expanded={opened}

          >

            Link with href

          </a>

        </p>

        {this.itemRender()}

      </div>

    );

  }

}

Collapse.defaultProps = {

  opened: true,

};

ФОРМЫ

В отличие от прямой работы с DOM (даже через jQuery), в React источником правды является состояние, а не DOM. Формы не являются исключением. Любое изменение в форме, посимвольно, если это ввод, должно быть перенесено в состояние. А элементы форм, чьи данные хранятся в состоянии React, называются управляемыми компонентами (controlled components).

Один из множества плюсов управляемых компонентов в том, что становится крайне легко проводить фильтрацию или валидацию. Например, если необходимо вводить данные в верхнем регистре (например, при вводе данных карты), то сделать это можно так:

handleChange **=** (e) **=>** {

**this**.setState({ value: e.target.value.toUpperCase() });

}

**ФОРМА РЕГИСТРАЦИИ**

import React from 'react';

// BEGIN (write your solution here)

export default class MyForm extends React.Component {

  constructor(props) {

    super(props);

    this.state = {

      form: {

        email: '',

        password: '',

        address: '',

        city: '',

        country: '',

        acceptRules: false,

      },

      submition: 'filingForm',

    };

  }

  handleSubmit = (e) => {

    e.preventDefault();

    this.setState({ submition: 'showTable' });

  };

  handleBack = () => {

    this.setState({ submition: 'filingForm' });

  };

  handleChange = ({ target }) => {

    const { form } = this.state;

    const value = target.type === 'checkbox' ? target.checked : target.value;

    this.setState({ form: { ...form, [target.name]: value } });

  };

  renderResult() {

    const { form } = this.state;

    return (

      <div>

        <button type="button" className="btn btn-primary" onClick={this.handleBack}>Back</button>

        <table className="table">

          <tbody>

            <tr>

              <td>acceptRules</td>

              <td>{form.acceptRules.toString()}</td>

            </tr>

            <tr>

              <td>address</td>

              <td>{form.address}</td>

            </tr>

            <tr>

              <td>city</td>

              <td>{form.city}</td>

            </tr>

            <tr>

              <td>country</td>

              <td>{form.country}</td>

            </tr>

            <tr>

              <td>email</td>

              <td>{form.email}</td>

            </tr>

            <tr>

              <td>password</td>

              <td>{form.password}</td>

            </tr>

          </tbody>

        </table>

      </div>

    );

  }

  renderForm() {

    const { form } = this.state;

    return (

      <form name="myForm" onSubmit={this.handleSubmit}>

        <div className="col-md-6 mb-3">

          <label htmlFor="email" className="col-form-label">Email</label>

          <input

            type="email"

            name="email"

            className="form-control"

            id="email"

            placeholder="Email"

            onChange={this.handleChange}

            value={form.email}

          />

        </div>

        <div className="form-group col-md-6">

          <label htmlFor="password" className="col-form-label">Password</label>

          <input

            type="password"

            name="password"

            className="form-control"

            id="password"

            placeholder="Password"

            onChange={this.handleChange}

            value={form.password}

          />

        </div>

        <div className="col-md-6 mb-3">

          <label htmlFor="address" className="col-form-label">Address</label>

          <textarea

            type="text"

            className="form-control"

            name="address"

            id="address"

            placeholder="1234 Main St"

            onChange={this.handleChange}

            value={form.address}

          />

        </div>

        <div className="col-md-6 mb-3">

          <label htmlFor="city" className="col-form-label">City</label>

          <input

            type="text"

            className="form-control"

            name="city"

            id="city"

            onChange={this.handleChange}

            value={form.city}

          />

        </div>

        <div className="col-md-6 mb-3">

          <label htmlFor="country" className="col-form-label">Country</label>

          <select

            id="country"

            name="country"

            className="form-control"

            onChange={this.handleChange}

            value={form.country}

          >

            <option value="">Choose</option>

            <option value="argentina">Argentina</option>

            <option value="russia">Russia</option>

            <option value="china">China</option>

          </select>

        </div>

        <div className="col-md-6 mb-3">

          <div className="form-check">

            <label className="form-check-label" htmlFor="rules">

              <input

                id="rules"

                type="checkbox"

                name="acceptRules"

                className="form-check-input"

                onChange={this.handleChange}

                checked={form.acceptRules}

              />

              Accept Rules

            </label>

          </div>

        </div>

        <button type="submit" className="btn btn-primary">Sign in</button>

      </form>

    );

  }

  render() {

    const { submition } = this.state;

    switch (submition) {

      case 'filingForm':

        return this.renderForm();

      case 'showTable':

        return this.renderResult();

      default:

        throw new Error('What a FACK!!');

    }

  }

}

РЕШЕНИЕ УЧИТЕЛЯ

export default class MyForm extends React.Component {

  constructor(props) {

    super(props);

    this.state = {

      form: {

        email: '',

        password: '',

        city: '',

        country: '',

        address: '',

        acceptRules: false,

      },

      submittingState: 'fillingForm',

    };

  }

  handleChangeField = ({ target }) => {

    const { form } = this.state;

    const value = target.type === 'checkbox' ? target.checked : target.value;

    this.setState({ form: { ...form, [target.name]: value } });

  }

  handleBackToForm = () => {

    this.setState({ submittingState: 'fillingForm' });

  }

  handleSubmitForm = (e) => {

    e.preventDefault();

    this.setState({ submittingState: 'submitted' });

  }

  renderRow = (key) => {

    const { form } = this.state;

    return (

      <tr key={key}>

        <td>{key}</td>

        <td>{form[key].toString()}</td>

      </tr>

    );

  };

  renderResult() {

    const { form } = this.state;

    const keys = Object.keys(form).sort();

    return (

      <div>

        <button type="button" className="btn btn-primary" onClick={this.handleBackToForm}>Back</button>

        <table className="table">

          <tbody>

            {keys.map(this.renderRow)}

          </tbody>

        </table>

      </div>

    );

  }

  renderForm() {

    const { form } = this.state;

    return (

      <form onSubmit={this.handleSubmitForm} name="myForm">

        <div className="col-md-6 mb-3">

          <label htmlFor="email" className="col-form-label">Email</label>

          <input

            type="email"

            name="email"

            onChange={this.handleChangeField}

            value={form.email}

            className="form-control"

            id="email"

            placeholder="Email"

          />

        </div>

        <div className="col-md-6 mb-3">

          <label htmlFor="password" className="col-form-label">Password</label>

          <input

            type="password"

            onChange={this.handleChangeField}

            value={form.password}

            name="password"

            className="form-control"

            id="password"

            placeholder="Password"

          />

        </div>

        <div className="col-md-6 mb-3">

          <label htmlFor="address" className="col-form-label">Address</label>

          <textarea type="text" name="address" value={form.address} onChange={this.handleChangeField} className="form-control" id="address" placeholder="1234 Main St" />

        </div>

        <div className="col-md-6 mb-3">

          <label htmlFor="city" className="col-form-label">City</label>

          <input type="text" name="city" onChange={this.handleChangeField} value={form.city} className="form-control" id="city" />

        </div>

        <div className="col-md-6 mb-3">

          <label htmlFor="country" className="col-form-label">Country</label>

          <select id="country" name="country" onChange={this.handleChangeField} className="form-control" value={form.country}>

            <option value="">Choose</option>

            <option value="argentina">Argentina</option>

            <option value="russia">Russia</option>

            <option value="china">China</option>

          </select>

        </div>

        <div className="col-md-6 mb-3">

          <div className="form-check">

            <label className="form-check-label" htmlFor="rules">

              <input id="rules" name="acceptRules" className="form-check-input" onChange={this.handleChangeField} type="checkbox" checked={form.acceptRules} />

              Accept Rules

            </label>

          </div>

        </div>

        <button type="submit" className="btn btn-primary">Sign in</button>

      </form>

    );

  }

  render() {

    const { submittingState } = this.state;

    switch (submittingState) {

      case 'fillingForm':

        return this.renderForm();

      case 'submitted':

        return this.renderResult();

      default:

        throw new Error(`'${submittingState}' - unknown state`);

    }

  }

}

**Удаление компонентов**

class List extends React.Component {

removeItem = (id) => (e) => {

e.preventDefault();

const newItems = this.state.items.filter((item) => item.id !== id);

this.setState({ items: newItems });

};

constructor(props) {

super(props);

const items = [1, 2, 3, 4, 5].map((item, index) => ({ item, id: index + 1 }));

this.state = { items };

}

renderItem({ id }) {

return <li><a href="#" onClick={this.removeItem(id)}>{id}</a></li>;

}

render() {

return <ul>

{this.state.items.map((item) => this.renderItem(item))}

</ul>;

}

}

**Использование immutability-helper**

**const** index **=** 5;

**const** newItems **=** update(items, {$splice: [[index, 1]]});

**КНОПКИ + -**

export default class Component extends React.Component {

  constructor(props) {

    super(props);

    this.state = { items: [] };

  }

  handleCount = (value) => {

    const { items } = this.state;

    const currentValue = get(items, [0, 'value'], 0) + value;

    const current = { id: uniqueId(), value: currentValue };

    this.setState({ items: [current, ...items] });

  }

  handleDec = () => this.handleCount(-1);

  handleInc = () => this.handleCount(1);

  handleRemove = (currentId) => () => {

    const { items } = this.state;

    this.setState({ items: items.filter(({ id }) => id !== currentId) });

  }

  renderLog() {

    const { items } = this.state;

    if (items.length === 0) {

      return null;

    }

    return (

      <div className="list-group">

        {items.map(({ id, value }) => (

          <button type="button" className="list-group-item list-group-item-action" key={id} onClick={this.handleRemove(id)}>

            {value}

          </button>

        ))}

      </div>

    );

  }

  render() {

    return (

      <div>

        <div className="btn-group font-monospace" role="group">

          <button type="button" className="btn btn-outline-success" onClick={this.handleInc}>+</button>

          <button type="button" className="btn btn-outline-danger" onClick={this.handleDec}>-</button>

        </div>

        {this.renderLog()}

      </div>

    );

  }

}

Функциональные компоненты

**const Title = (props) => <div className="card-title">{props.children}</div>;**

**const Body = (props) => <div className="card-body">{props.children}</div>;**

**class Card extends React.Component {**

**static Body = Body;**

**static Title = Title;**

**render() {**

**return <div className="card card-block">{this.props.children}</div>;**

**}**

**}**

**const vdom = (<Card>**

**<Card.Body>**

**<Card.Title>What is love?</Card.Title>**

**</Card.Body>**

**</Card>);**

# Контекст (Context API)

 Передача данных в контекст. Работает так: оборачиваем нужные компоненты в компонент контекста <UserContext.Provider> и передаем туда нужные данные в проп value:

*// Контекст будет доступен только внутри тех компонентов, которые он оборачивает и в тех, что вложены в данные компоненты*

*// currentUser - данные текущего пользователя*

<**UserContext**.**Provider** value={currentUser}>

<**App** />

</**UserContext**.**Provider**>

 Получение данных из контекста в нужном месте:

**import** UserContext **from** '...';

*// Любой компонент внутри блока <UserContext.Provider>*

**class** InnerComponent **extends** React.Component {

*// Определяем тип контекста*

**static** contextType **=** UserContext;

render() {

*// Получаем доступ к контексту через this.context*

**return** <**Profile** user={**this**.context} />;

}

}