**Дополнительно: рефакторинг с использованием контекста**

Проект «Криптоутки» совсем небольшой, поэтому мы не передаём пропсы далеко вниз по иерархии компонентов. Но, если представить, что приложение разрослось, было бы неплохо иметь доступ к объекту userData и к значению параметра loggedIn из любого компонента приложения. В этом случае нам бы помог контекст, который вы изучили [в 11 спринте](https://praktikum.yandex.ru/trainer/web/lesson/97aa314a-2e92-417b-a47e-c54eef85a144).

**Создание и подключение контекста**

Для начала создадим новый файл AppContext.js и разместим его в директории /components:

Скопировать кодJSX

*// AppContext.js*

import React from 'react';

export const AppContext = React.createContext();

Затем импортируем контекст в App.js при помощи директивы import:

import { AppContext } from './AppContext';.

Так сейчас выглядит код внутри метода render компонента App:

Скопировать кодJSX

<Switch>

<ProtectedRoute path="/ducks" loggedIn={this.state.loggedIn} component={Ducks} />

<ProtectedRoute path="/my-profile" loggedIn={this.state.loggedIn} userData={this.state.userData} component={MyProfile} />

<Route path="/login">

<div className="loginContainer">

<Login handleLogin={this.handleLogin} />

</div>

</Route>

<Route path="/register">

<div className="registerContainer">

<Register />

</div>

</Route>

<Route>

{this.state.loggedIn ? <Redirect to="/ducks" /> : <Redirect to="/login" />}

</Route>

</Switch>

Теперь обернём App в свойство Provider контекста. Больше не нужно будет передавать стейт и методы вниз по дереву компонентов, поскольку их внедрит сам провайдер:

Скопировать кодJSX

*// App.js*

<AppContext.Provider value={{state: this.state, handleLogin: this.handleLogin}}>

<Switch>

<ProtectedRoute path="/ducks" component={Ducks} />

<ProtectedRoute path="/my-profile" component={MyProfile} />

<Route path="/login">

<div className="loginContainer">

<Login />

</div>

</Route>

<Route path="/register">

<div className="registerContainer">

<Register />

</div>

</Route>

<Route>

{this.state.loggedIn ? <Redirect to="/ducks" /> : <Redirect to="/login" />}

</Route>

</Switch>

</ AppContext.Provider>

**Подписка на контекст**

Теперь обновим компоненты ProtectedRoute и Login, чтобы они также были подписаны на контекст.

ProtectedRoute — функциональный компонент, что облегчает нашу задачу: контекст в этом случае можно подключить при помощи хука React.useContext.

Скопировать кодJSX

*// ProtectedRoute.js*

import React from 'react';

import { Route, Redirect } from "react-router-dom";

import { AppContext } from './AppContext.js'; *// импортируем контекст*

const ProtectedRoute = ({ component: Component, ...props }) => {

const value = React.useContext(AppContext); *// получаем значения из контекста*

return (

<Route>

{

() => value.state.loggedIn === true ? <Component {...props} userData={value.state.userData} /> : <Redirect to="./login" />

}

</Route>

)}

export default ProtectedRoute;

Обратите внимание: мы поместили контекст в переменную value. Из неё мы получаем значение стейта loggedIn, а также с её помощью передаём значение объекта userData, чтобы компонент MyProfile мог получить к нему доступ.

Login — классовый компонент и для подписки на контекст ему понадобится обновить метод handleLogin.

Скопировать кодJSX

*// Login.js*

import React from 'react';

import { Link, withRouter } from 'react-router-dom';

import Logo from './Logo.js';

import \* as duckAuth from '../duckAuth.js';

import { AppContext } from './AppContext.js'; *// импортируем контекст*

import './styles/Login.css';

class Login extends React.Component {

static contextType = AppContext; *// подключаем контекст*

constructor(props){

super(props);

this.state = {

username: '',

password: '',

message: ''

}

this.handleChange = this.handleChange.bind(this);

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

}

handleChange(e) {

const {name, value} = e.target;

this.setState({

[name]: value

});

}

handleSubmit(e){

const value = this.context; *// получаем значения из контекста*

e.preventDefault();

if (!this.state.username || !this.state.password){

return;

}

duckAuth.authorize(this.state.username, this.state.password)

.then((data) => {

if (!data){

return this.setState({

message: 'Что-то пошло не так!'

});

}

if (data.jwt){

this.setState({email: '', password: '', message: ''} ,() => {

value.handleLogin(); *// подключаем метод из value*

this.props.history.push('/ducks');

return;

})

}

})

.catch(err => console.log(err));

}

render(){

return(

<div onSubmit={this.handleSubmit} className="login">

<Logo title={'CryptoDucks'}/>

<p className="login\_\_welcome">

Это приложение содержит конфиденциальную информацию.

Пожалуйста, войдите или зарегистрируйтесь, чтобы получить доступ к CryptoDucks.

</p>

<p className="login\_\_error">

{this.state.message}

</p>

<form className="login\_\_form">

<label for="username">

Логин:

</label>

<input id="username" required name="username" type="text" value={this.state.username} onChange={this.handleChange} />

<label htmlFor="password">

Пароль:

</label>

<input id="password" required name="password" type="password" value={this.state.password} onChange={this.handleChange} />

<div className="login\_\_button-container">

<button type="submit" className="login\_\_link">Войти</button>

</div>

</form>

<div className="login\_\_signup">

<p>Еще не зарегистрированы?</p>

<Link to="/register" className="signup\_\_link">Зарегистрироваться</Link>

</div>

</div>

)

}

}

export default withRouter(Login);

В коде почти ничего не изменилось, поэтому без комментариев эти изменения можно даже не заметить. Но для подписки на контекст в классовом компоненте мы сделали немало: импортировали контекст, добавили static contextType = AppContext; после отключающей фигурной скобки в теле класса, а в методе handleSubmit — получили значение контекста и вызвали метод value.handleLogin вместо this.props.handleLogin, который использовался раньше.

Вот и всё! В нашем маленьком приложении использовать контекст было вовсе не обязательно. Но обычно приложения значительно сложнее и крупнее, поэтому использование контекста в них — обычная практика.