Тестовое задание

1. Запуск проекта, подключения стора

Установим библиотеки:

**axios** – для запросов на сервер

**react-router-dom, @types/react-router-dom** – для постраничной навигации

**moment** – для работы с датами https://momentjs.com/ библиотеки antd

**redux, react-redux, @types/reac-redux** – управление состоянием

**redux-thunk** – для асинхронных запросов

Итак,

Оборачиваем приложение в **<Provider>,** чтобы связать приложение React с Redux

Затем оборачиваем приложение в **<BrowserRouter>**, чтобы в дальнейшем работать с постраничной навигацией

ReactDOM.render(  
 <Provider store={}>  
 <BrowserRouter>  
 <App/>  
 </BrowserRouter>  
 </Provider>  
 ,  
 *document*.getElementById('root')  
);

Также определим две страницы: Pages и Events. Соответственно создадим папку **pages** и в ней два файла Pages.tsx и Events.tsx

Далее создадим папку с компонентами **components**.

Далее создадим папку с компонентами **store**. В ней будет написан redux

Далее создадим папку с компонентами **models**. В ней будут описаны типы сущностей проекта.

Итого мы создали изначальные файлы и папки, которые понадобятся для дальнейшей работы.

1. Постраничная навигация

**- Описываем роутер**

Создадим папку **router** в которой будет находиться файл index.ts. В нем будем **описывать** все маршруты, используемые в приложении.

В файле будем использовать enum. Он будет словарем, в котором мы опишем пути и сможем их потом быстро поменять не переписывая все приложение:

import Login from "../pages/Login";  
import Event from "../pages/Event";  
  
export interface IRoute {  
 path: string;  
 component: React.ComponentType;  
 exact?: boolean;  
}  
  
// словарь, в котором будут храниться маршруты к компонентам  
export enum RouteNames {  
 *LOGIN* = '/login',  
 *EVENT* = '/',  
}  
  
// этот массив роутов будет показываться НЕавторизованным пользователям, т.к. есть логинизация  
export const publicRoutes: IRoute[] = [  
 {path: RouteNames.*LOGIN*, exact: true, component: Login}  
];  
  
// этот массив роутов будет показываться авторизованным пользователям, т.к. есть логинизация  
export const privateRoutes: IRoute[] = [  
 {path: RouteNames.*EVENT*, exact: true, component: Event}  
];

**- Физически запиливаем роутер**

Создадим в компонентах файл AppRouter.

Здесь будем использовать Redirect и условие по авторизации.

import React from 'react';  
import {Switch, Route, Redirect} from "react-router-dom";  
import {privateRoutes, publicRoutes, RouteNames} from "../router";  
  
const AppRouter = () => {  
 const auth = false; // если true, попадаем на events; если false, попадаем на login  
 return (  
 auth ? // проверка авторизации  
 <Switch>  
 {privateRoutes.map(*route* =>  
 <Route path={*route*.path}  
 exact={*route*.exact}  
 component={*route*.component}  
 key={*route*.path} // укажем в качестве ключа путь к компоненту, т.к. он будет уникальным  
 />  
 )}  
 <Redirect to={RouteNames.*EVENT*}/>  
 </Switch>  
 :  
 <Switch>  
 {publicRoutes.map(*route* =>  
 <Route path={*route*.path}  
 exact={*route*.exact}  
 component={*route*.component}  
 key={*route*.path} // укажем в качестве ключа путь к компоненту, т.к. он будет уникальным  
 />  
 )}  
 <Redirect to={RouteNames.*LOGIN*}/>  
 </Switch>  
 );  
};  
  
export default AppRouter;

Далее подключим роутер в приложение в App.tsx

import React, {FC} from 'react';  
import './App.css';  
import AppRouter from "./components/AppRouter";  
  
const App: FC = () => {  
 return (  
 <div>  
 <AppRouter/>  
 </div>  
 );  
};  
  
export default App;

1. Layout приложения. Каркас. AntDesign

**- Создадим первичный Layout приложения.**

Для этого будем использовать компонент Layout из antd

<https://ant.design/components/layout/>

Для работы с antd надо ИМПОРТИРОВАТЬ CSS файл **import 'antd/dist/antd.css'. Сделаем импорт в файл с общими стилями App.css. В CSS надо писать так:**

@import '~antd/dist/antd.css';

Создадим компонент Navbar. На всякий случай создадим кнопку, при нажатии на которую авторизованный пользователь сможет зайти на страницу логина. Для этого будем использовать хук useHistory(). Он возвращает объект роутера типа такого:

action: "POP"

block: ƒ block(prompt)

createHref: ƒ createHref(location)

go: ƒ go(n)

goBack: ƒ goBack()

goForward: ƒ goForward()

length: 8

listen: ƒ listen(listener)

location: {pathname: '/', search: '', hash: '', state: undefined, key: '8uzkvp'} // **локейшен**

**push: ƒ push(path, state) // с его помощью можно перемещаться по маршрутам**

replace: ƒ replace(path, state)

[[Prototype]]: Object

Повесим обработчик событий на кнопку, передадим в ПУШ путь для ЛОГИНА, на НЕГО БУДЕМ ПЕРЕХОДИТЬ ПРИ НАЖАТИИ на КНОПКУ:

<Menu.Item  
 onClick={() => router.push(RouterNames.*LOGIN*)}  
 key={1}>**Login**</Menu.Item>

import React, {FC} from 'react';  
import {Layout, Row} from "antd";  
  
  
const Navbar: FC = () => {  
 return (  
 <Layout.Header>  
 <Row justify='end'>  
 **button** </Row>  
 </Layout.Header>  
 );  
};  
  
export default Navbar;

и добавим его ВНУТРЬ КОНТЕНТА, Т.К. ИМЕННО ОН БУДЕТ ЗАДАВАТЬ КОНТЕНТ в App, обернем все приложение в Layout из antd.

import React, {FC} from 'react';  
import './App.css';  
import AppRouter from "./components/AppRouter";  
import Navbar from "./components/Navbar";  
import {Layout} from "antd";  
import './App.css'  
  
const App: FC = () => {  
 return (  
 <Layout>  
 <Navbar/>  
 <Layout.Content>  
 <AppRouter/>  
 </Layout.Content>  
 </Layout>  
 );  
};  
  
export default App;

1. Auth reducer, типизация. Заканчиваем «остов» приложения.

**- Создадим редьюсер auth для авторизации**

import {AuthAction, AuthActionEnum, AuthState} from "./types";  
  
const initialState: AuthState = {  
 auth: false  
};  
  
export default function authReducer(*state* = initialState, *action*: AuthAction): AuthState {  
 switch (*action*.type) {  
 case AuthActionEnum.*SET\_AUTH*:  
 return {...*state*, auth: *action*.payload};  
  
 default:  
 return *state*;  
 }  
}

И добавим его в корневой combineReducers. Для этого все редьюсеры будем описывать в одном файле, затем его импортировать:

import {applyMiddleware, combineReducers, createStore} from "redux";  
import thunk from 'redux-thunk'  
import reducers from './reducers'  
  
  
const rootReducer = combineReducers(reducers);  
  
export const store = createStore(rootReducer, applyMiddleware(thunk));  
  
// Типизация  
export type RootStateType = ReturnType<typeof store.getState> // тип стейта  
export type AppDispatch = typeof store.dispatch // тип диспатча

Для типизации создадим отдельный файл types.ts

export interface AuthState {  
 auth: boolean  
}  
// создадим enum, чтобы не писать много переменных дл типов action  
export enum AuthActionEnum {  
 *SET\_AUTH* = "SET\_AUTH"  
}  
// типизация action type И payload  
interface SetAuthAction {  
 type: AuthActionEnum.*SET\_AUTH*;  
 payload: boolean  
}  
// обобщающи тип, объединяющий интерфейсы. Пока их только один, перечисляем с помощью "|"  
export type AuthAction = SetAuthAction

Далее кастомизируем хук useSelect(), т.к. он не понимает, какие поля есть сторе. Для этого создадим папку hooks, файл **TypedUseSelectorHook**:

import {TypedUseSelectorHook, useSelector} from "react-redux";  
import {RootState} from '../store'  
  
export const useTypedSelector: TypedUseSelectorHook<RootState> = useSelector;

**Теперь наш хук знает, какие поля есть в сторе(стейте)**

Достанем данные для компоненты AppRouter:

const AppRouter = () => {  
 const {isAuth} = useTypedSelector(*state* => *state*.auth);  
 return (  
 isAuth ?  
 <Switch>  
 {privateRoutes.map(*route* =>  
 <Route path={*route*.path}

...

и для NavBar:

const Navbar: FC = () => {  
 const router = useHistory(); // см. notes  
 const {isAuth} = useTypedSelector(*state* => *state*.auth);  
 return (  
 <Layout.Header>  
 {isAuth  
 ?  
 <>  
 <div style={{color: 'white'}}>**UlbiTV**</div>

...

1. Логика Логина

**- Создадим компонент LoginForm**

import React, {FC} from 'react';  
import {Form, Input, Button} from 'antd'  
  
const LoginForm: FC = () => {  
 return (  
 <Form>  
 <Form.Item  
 label="Username"  
 name="username"  
 rules={[{required: true, message: 'Please input your username!'}]}  
 >  
 <Input/>  
 </Form.Item>  
 <Form.Item  
 label="Password"  
 name="password"  
 rules={[{required: true, message: 'Please input your password!'}]}  
 >  
 <Input.Password/>  
 </Form.Item>  
 <Form.Item>  
 <Button type="primary" htmlType="submit">  
 **Login** </Button>  
 </Form.Item>  
 </Form>  
 );  
};  
  
export default LoginForm;

и будем использовать ее в Login

const Login: FC = () => {  
 return (  
 <Layout>  
 <Row justify='center' align='middle' className='h100'>  
 <LoginForm/>  
 </Row>  
 </Layout>  
 );  
};

далее создадим файл utils.tsб в который поместим вспомогательные функции. Сейчас поместим туда функцию, которая будет передавать объект валидации в форму:

export const rules = {  
 required: (*message*: string) => ({  
 required: true,  
 *message*,  
 })  
};

после этого заменим аргумент rules в LoginForm:

<Form.Item  
 label="Password"  
 name="password"  
 rules={[rules.required('Please input your password!')]}  
>

ДЛЯ ИМИТАЦИИ РАБОТЫ С СЕРВЕРОМ создадим фейковые данные в папке PUBLIC

Users.json

[  
 {  
 "username": "user",  
 "password": "123"  
 },  
 {  
 "username": "admin",  
 "password": "123"  
 },  
 {  
 "username": "timur",  
 "password": "123"  
 }  
]

Согласно документации antd надо повесить обработчик событий onFinish на Form. Всю логику будем писать в редьюсере. Но прежде создадим модель IUser.ts :

export interface IUser {  
 username: string;  
 password: string;  
}

Также добавим типы:

import {IUser} from "../../../models/IUser";  
  
export interface AuthState {  
 isAuth: boolean;  
 user: IUser;  
 isLoading: boolean;  
 error: string;  
}  
  
// создадим enum, чтобы не писать много переменных дл типов action  
export enum AuthActionEnum {  
 *SET\_AUTH* = "SET\_AUTH",  
 *SET\_ERROR* = "SET\_ERROR",  
 *SET\_USER* = "SET\_USER",  
 *SET\_IS\_LOADING* = "SET\_IS\_LOADING",  
}  
  
// типизация action type И payload  
interface SetAuthAction {  
 type: AuthActionEnum.*SET\_AUTH*;  
 payload: boolean;  
}  
  
interface SetErrorAction {  
 type: AuthActionEnum.*SET\_ERROR*;  
 payload: string;  
}  
  
interface SetUserAction {  
 type: AuthActionEnum.*SET\_USER*;  
 payload: IUser;  
}  
  
interface SetIsLoadingAction {  
 type: AuthActionEnum.*SET\_IS\_LOADING*;  
 payload: boolean;  
}  
  
// обобщающи тип всех action, объединяющий интерфейсы. Пока их только один, перечисляем с помощью "|"  
export type AuthAction =  
 SetAuthAction |  
 SetErrorAction |  
 SetUserAction |  
 SetIsLoadingAction

добавим поля в Initialstate:

const initialState: AuthState = {  
 isAuth: false,  
 error: '',  
 isLoading: false,  
 user: {} as IUser,  
};

И допишем редьюсер:

import {AuthAction, AuthActionEnum, AuthState} from "./types";  
import {IUser} from "../../../models/IUser";  
  
const initialState: AuthState = {  
 isAuth: false,  
 error: '',  
 isLoading: false,  
 user: {} as IUser,  
};  
  
export default function authReducer(*state* = initialState, *action*: AuthAction): AuthState {  
 switch (*action*.type) {  
  
 case AuthActionEnum.*SET\_AUTH*:  
 return {...*state*, isAuth: *action*.payload, isLoading: false};  
  
 case AuthActionEnum.*SET\_ERROR*:  
 return {...*state*, error: *action*.payload, isLoading: false};  
  
 case AuthActionEnum.*SET\_IS\_LOADING*:  
 return {...*state*, isLoading: *action*.payload};  
  
 case AuthActionEnum.*SET\_USER*:  
 return {...*state*, user: *action*.payload};  
  
 default:  
 return *state*;  
 }  
}

Допишем также Action Creators:

import {AuthActionEnum, SetAuthAction, SetErrorAction, SetIsLoadingAction, SetUserAction} from "./types"  
import {IUser} from "../../../models/IUser";  
  
export const AuthActionCreators = {  
 setUser: (*user*: IUser): SetUserAction => ({type: AuthActionEnum.*SET\_USER*, payload: *user*}),  
 setIsAuth: (*auth*: boolean): SetAuthAction => ({type: AuthActionEnum.*SET\_AUTH*, payload: *auth*}),  
 setIsLoading: (*payload*: boolean): SetIsLoadingAction => ({type: AuthActionEnum.*SET\_IS\_LOADING*, *payload*}),  
 setError: (*payload*: string): SetErrorAction => ({type: AuthActionEnum.*SET\_ERROR*, *payload*})  
};