1. Устанавливаем WebStorm

2. Устанавливаем Node.js

3. Из командной строки (Node.js command prompt):

- установить npm i create-react-app -g

- перейти в исходную папку, куда будет устанавливаться проект (D:)

- запускаем create-react-app diplom\_react

4. Запускам WebStorm -> Открыть -> 'D:\diplom\_react'

5. Первоначальная настройка проекта:

- тестирование проекта какое либо на данном этапе не предполагается, удалим строки из package.json:

"@testing-library/jest-dom": "^4.2.4",

"@testing-library/react": "^9.5.0",

"@testing-library/user-event": "^7.2.1",

"test": "react-scripts test",

- нам не нужно ручное настраивание и ручной контроль всех зависимостей (Webpack, Babel, ESLint), удалим строки из package.json:

"eject": "react-scripts eject"

- настроим webpack чтобы мы могли подключать стили как объекты

В файлах

D:\Refer\_React\node\_modules\react-scripts\config\webpack.config.js

Изменим строку:

options: cssOptions,

на

options: {...cssOptions, modules: **true**},

6. Удалим ненужные файлы:

- \src\App.test.js

- \src\logo.svg

- \src\setupTests.js

- \src\serviceWorker.js

- \public\logo192.png

- \public\logo512.png

- \public\manifest.json

- \public\robots.txt

7. Уберем все ненужное в следующих файлах:

- \public\index.html, сразу переименуем заголовок

<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="utf-8" />  
 <link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />  
 <meta name="viewport" content="width=device-width, initial-scale=1" />  
 <meta name="theme-color" content="#000000" />  
 <title>Курсовая работа по React</title>  
 </head>  
 <body>  
 <noscript>You need to enable JavaScript to run this app.</noscript>  
 <div id="root"></div>  
 </body>  
</html>

- \src\App.css

.App {  
 text-align: center;  
}

- \src\App.js

* **import** React **from** 'react';  
  **import** './App.css';  
    
  **function** App() {  
   **return** (  
   <div className="App">  
   Hello World!!!  
   </div>  
   );  
  }  
    
  **export default** App;

- \src\index.js

**import** React **from** 'react';  
**import** ReactDOM **from** 'react-dom';  
**import** './index.css';  
**import** App **from** './App';  
  
ReactDOM.render(  
 <React.StrictMode>  
 <App />  
 </React.StrictMode>,  
 document.getElementById('root')  
);

Мы настроили проект, удалили из него все лишнее и теперь готовы к разработке. По-хорошему, нам нужно еще настроить Git. Но это не входит в программу курсовой.

8. Установим в наш проект Material-UI и SVG иконки, Axios

- npm i @material-ui/core

- npm i @material-ui/icons

- npm i axios –S

- npm install firebase –save

- npm install redux react-redux –save

- npm install redux-thunk –save

- npm install --save react-d3-speedometer

9. Структура проекта:

- Layout. Здесь расположены файлы, которые задают общую разметку всему сайту. Содержат хедер, футер, майн теги, а также стили для них

- Containers. Здесь расположены файлы, которые содержат собственный стейт, т.е. корневые компоненты

- Components. Здесь будут располагаться чисто функциональные компоненты

10. По структуре, у нас будет 5 контейнеров и один лайоут. Создадим их:

В папке Layout:

* D:\Refer\_React\src\Layout\Layout.js

**import** React, {Component} **from** 'react'  
  
**class** Layout **extends** Component {  
 render() {  
 **return** (  
 <div>  
 <main>  
 { **this**.props.children }  
 </main>  
 </div>  
 )  
 }  
}  
  
**export default** Layout

* D:\Refer\_React\src\Layout\Layout.css

.Layout {  
 height: 100vh;  
 display: flex;  
 flex-direction: column;  
}  
  
.Layout main {  
 display: flex;  
 flex-direction: column;  
 flex-grow: 1;  
}

В папке Containers:

D:\Refer\_React\src\Containers

D:\Refer\_React\src\Containers\Account

D:\Refer\_React\src\Containers\Account\Account.css

D:\Refer\_React\src\Containers\Account\Account.js

D:\Refer\_React\src\Containers\Main

D:\Refer\_React\src\Containers\Main\Main.css

D:\Refer\_React\src\Containers\Main\Main.js

D:\Refer\_React\src\Containers\Register

D:\Refer\_React\src\Containers\Register\Register.css

D:\Refer\_React\src\Containers\Register\Register.js

D:\Refer\_React\src\Containers\Test

D:\Refer\_React\src\Containers\Test\Test.css

D:\Refer\_React\src\Containers\Test\Test.js

**import** React, {Component} **from** 'react'  
**import** classes **from** './Main.css'  
  
**class** Main **extends** Component {  
 render() {  
 **return** (  
 <div className={classes.Main}>  
 <h1>Main</h1>  
 </div>  
 )  
 }  
}  
  
**export default** Main

Main.css:

.Main {  
 color: aqua;  
}

Остальные файлы по аналогии.

11.Добавим меню к лояуту

**import** React, {Component} **from** 'react'  
**import** ApplicationBar **from** "../Components/Navigation/Panel/Panel";  
  
**class** Layout **extends** Component {  
  
 render() {  
 **return** (  
 <div>  
 <ApplicationBar/>  
 <main>  
 { **this**.props.children }  
 </main>  
 </div>  
 )  
 }  
}  
  
**export default** Layout

11.1 Создадим нашу основную навигационную панель:

Как и в любом компоненте сначала идут импорты нужных нам компонент:

**import** React **from** 'react';  
**import** {withRouter} **from** 'react-router-dom'  
**import** {Pages} **from** '../../../Route/pages';  
**import** clsx **from** 'clsx';  
**import** {makeStyles} **from** '@material-ui/core/styles';  
**import** AppBar **from** '@material-ui/core/AppBar';  
**import** Toolbar **from** '@material-ui/core/Toolbar';  
**import** Typography **from** '@material-ui/core/Typography';  
**import** Button **from** '@material-ui/core/Button';  
**import** IconButton **from** '@material-ui/core/IconButton';  
**import** MenuIcon **from** '@material-ui/icons/Menu';  
**import** Drawer **from** '@material-ui/core/Drawer';  
**import** CssBaseline **from** '@material-ui/core/CssBaseline';  
**import** List **from** '@material-ui/core/List';  
**import** Divider **from** '@material-ui/core/Divider';  
**import** ChevronLeftIcon **from** '@material-ui/icons/ChevronLeft';  
**import** ListItem **from** '@material-ui/core/ListItem';  
**import** ListItemIcon **from** '@material-ui/core/ListItemIcon';  
**import** ListItemText **from** '@material-ui/core/ListItemText';  
**import** {connect} **from** "react-redux";  
**import** {signOut} **from** "../../../Store/actions/rootActions";  
**import** {ACCOUNT, LOGIN, MAIN, REGISTER} **from** "../../../Route/path";  
**import** {isTeacher} **from** "../../../Store/helper";

Далее идет описание стилей:

**const** drawerWidth = 240;

**const** useStyles = makeStyles((theme) => ({  
 root: {  
 flexGrow: 1,  
 },  
 appBarShift: {  
 width: `calc(100% - ${drawerWidth}px)`,  
 marginLeft: drawerWidth,  
 transition: theme.transitions.create(['margin', 'width'], {  
 easing: theme.transitions.easing.easeOut,  
 duration: theme.transitions.duration.enteringScreen,  
 }),  
 },  
 menuButton: {  
 marginRight: 36,  
 },  
 menuButtonHidden: {  
 display: 'none',  
 },  
 title: {  
 flexGrow: 1,  
 },  
 drawer: {  
 width: drawerWidth,  
 flexShrink: 0,  
 left: 'none'  
  
 },  
 drawerPaper: {  
 width: drawerWidth,  
 },  
 drawerHeader: {  
 display: 'flex',  
 alignItems: 'center',  
 padding: theme.spacing(0, 1),  
 // necessary for content to be below app bar  
 ...theme.mixins.toolbar,  
 justifyContent: 'flex-end',  
 },  
 accountInfo: {  
 display: 'flex',  
 }  
}));

описание необходимых функций, констант и перменных:

- в classes будут храниться наши стили:

**const** classes = useStyles();

- описание переключателя выезжающего меню (есть / нет):

**const** [open, setOpen] = React.useState(**false**);  
  
**const** handleDrawerOpen = () => {  
 setOpen(**true**);  
};  
  
**const** handleDrawerClose = () => {  
 setOpen(**false**);  
};

- получение наименования текущей станицы:

**const** getTitle = () => {  
 **const** page = Pages.find(item => item.to === props.location.pathname);  
 **return** page ? page.title : '';  
};

- функция, которая разрешает/запрещает выбор страниц, в зависимости от авторизации:

**const** isEnabledItem = (item) => {  
 **if** ((item.to === LOGIN || item.to === REGISTER) && props.user !== **null**) { **return false**; }  
 **if** ((item.to === ACCOUNT || item.to === MAIN) && props.user === **null**) { **return false**; }  
 **if** ((item.to === MAIN) && props.user !== **null** && isTeacher(props.user.email)) { **return false**; }  
 **if** ((item.to === ACCOUNT) && props.user !== **null** && !isTeacher(props.user.email)) { **return false**; }  
 **return true**};

- две функции, которые передают компоненту необходимые данные и редюсеры из стора:

**function** mapStateToProps(state) {  
 **return** {  
 user: state.user  
 }  
}  
**function** mapDispatchToProps(dispatch) {  
 **return** {  
 signOutFirebase: () => dispatch(signOut()),  
 }  
}

- функция всегда должна возвратить нам некий рендеринг:

- стуктура:

**return** (  
 <div className={classes.root}>  
 <CssBaseline/>  
 <AppBar />   
 <Drawer />  
 </div>  
);

CssBaseline – компонент MaterialUI.

AppBar – компонент MU, рисующий верхнюю часть. Имеет заголовок и кнопки регистации/входа или выхода

Drawer – компонент MU , боковое меню

Распишем каждый компонент более подробно. Структура AppBar:

<AppBar position="static" className={clsx(classes.appBar, open && classes.appBarShift)}>  
 <Toolbar />  
</AppBar>

Toolbar - компонент MU, панель инструментов. Структура:

<Toolbar>  
 <IconButton />  
 <Typography />  
 {  
 !(!!props.user && !!props.user.displayName)  
 ? <div>  
 <Button />  
 <Button />  
 </div>  
 : <div className={classes.accountInfo}>  
 <Button />  
 <Divider orientation="vertical" flexItem style={{backgroundColor: '#ffffff'}} variant="middle"/>  
 <Button />  
 </div>  
 }  
</Toolbar>

IconButton – компонент MU, представляющий собой кнопку в виде картинки (иконки). В нашем случае это три горизонтальных черты (по умолчанию таким значком принято обозначать боковое меню). При нажатии на эту кнопку боковое меню выезжает слева

<IconButton edge="start" color="inherit" aria-label="menu" onClick={handleDrawerOpen}  
 className={clsx(classes.menuButton, open && classes.menuButtonHidden)}>  
 <MenuIcon/>  
</IconButton>

MenuIcon – иконка в виде сендвича

Далее следует надпись заголовка:

<Typography variant="h6" className={classes.title}>  
 {getTitle()}  
</Typography>

С правой стороны находятся кнопки для регистрации/входа или входа в личный кабинет или выхода:

Если пользователь не зарегистрирован, !(!!props.user && !!props.user.displayName)

то покажем кнопки:

<Button color="inherit"  
 onClick={() => {  
 props.history.push({  
 pathname: Pages.find(item => item.id === 'login').to  
 })  
 }}>  
 {Pages.find(item => item.id === 'login').title}  
</Button>  
<Button color="inherit"  
 onClick={() => {  
 props.history.push({  
 pathname: Pages.find(item => item.id === 'register').to  
 })  
 }}>  
 {Pages.find(item => item.id === 'register').title}  
</Button>

Иначе:

<Button color="inherit"  
 size="large"  
 startIcon={Pages.find(item => item.id === 'account').icon}  
 onClick={() => {  
 **if** (isTeacher(props.user.email)) {  
 console.log('ЛК');  
 props.history.push(ACCOUNT);  
 }  
 }}>  
 {props.user.displayName }  
</Button>  
<Divider orientation="vertical" flexItem style={{backgroundColor: '#ffffff'}} variant="middle"/>  
<Button color="inherit"  
 onClick={() => {  
 console.log('Выход');  
 props.signOutFirebase();  
 props.history.push(LOGIN);  
 }}>  
 Выйти  
</Button>

Divider – компонент MU, представляющий из себя разделитель в виде вертикальной черты

Боковая панель представляет собой набор компонентов:

<div className={classes.drawerHeader}>  
 <IconButton />  
</div>  
<Divider/>  
<List/>

Иконка в виде стрелки влево, при нажатии на которую закрывается боковое меню

<div className={classes.drawerHeader}>  
 <IconButton onClick={handleDrawerClose}>  
 <ChevronLeftIcon/>  
 </IconButton>  
</div>

Divider – компонент MU, представляющий из себя разделитель в виде горизонтальной черты

Список элементов меню, с иконками, текстом, при нажатии на которые будет переход на соответствующую страницу. Элементы имеют свойство доступности в зависимости от результата функции isEnabledItem, которая была описана ранее:

<List>  
 {Pages.map((page) => (  
 <ListItem button  
 key={page.title}  
 disabled={!isEnabledItem(page)}  
 onClick={() => {  
 props.history.push({  
 pathname: page.to  
 });  
 handleDrawerClose()  
 }}>  
 <ListItemIcon>{page.icon}</ListItemIcon>  
 <ListItemText primary={page.title}/>  
 </ListItem>  
 ))}  
</List>

Данные для формирования страниц были созданы при разработке компонента react-router в файле pages.js

12.прикрутим к проекту react-router

Итого мы имеем несколько страниц. Для перехода между страницами используем компонент react-router, который входит в экосистему React.

Сперва определим список констант, обозначающих наименования страниц (файл Route/path.js):

**export const** MAIN = "/main";  
**export const** ACCOUNT = "/account";  
  
**export const** TEST = "/test/:name";  
**export const** TEST\_PATH = "/test";  
**export const** RESULT = "/result";  
  
**export const** EDIT\_QUESTIONS = "/edit\_questions/:name";  
**export const** EDIT\_QUESTIONS\_PATH = "/edit\_questions";  
  
**export const** EDIT\_ANSWERS = "/edit\_answers/:name";  
**export const** EDIT\_ANSWERS\_PATH = "/edit\_answers";  
  
**export const** MILLIONAIRE = "/millionaire";  
  
**export const** AUTH = "/register/:name";  
**export const** LOGIN = "/register/log";  
**export const** REGISTER = "/register/reg";

Затем опишем объект Pages, содержащий в себе заголовок страницы, иконку, свойство Exact (точное совпадение) и страница, куда ведет данный элемент (файл Route/pages.js):

**import** {ACCOUNT, LOGIN, MAIN, MILLIONAIRE, REGISTER} **from** "./path";  
**import** React **from** 'react';  
**import** HomeIcon **from** '@material-ui/icons/Home';  
**import** AccountBoxIcon **from** '@material-ui/icons/AccountBox';  
**import** LockOpenIcon **from** '@material-ui/icons/LockOpen';  
**import** PersonAddIcon **from** '@material-ui/icons/PersonAdd';  
  
**export const** Pages = [  
 {  
 id: 'main',  
 to: MAIN,  
 title: 'Главная',  
 exact: **true**,  
 icon: <HomeIcon />  
 },  
 {  
 id: 'account',  
 to: ACCOUNT,  
 title: 'Личный кабинет',  
 exact: **false**,  
 icon: <AccountBoxIcon/>  
 },  
 {  
 id: 'login',  
 to: LOGIN,  
 title: 'Вход',  
 exact: **false**,  
 icon: <LockOpenIcon/>  
 },  
 {  
 id: 'register',  
 to: REGISTER,  
 title: 'Зарегистрироваться',  
 exact: **false**,  
 icon: <PersonAddIcon/>  
 },  
 {  
 id: 'millionaire',  
 to: MILLIONAIRE,  
 title: 'Кто хочет стать миллионером',  
 exact: **false**,  
 icon: <PersonAddIcon/>  
 },  
];

В файле index.js добавим необходимый функционал, для работы с react-router:

**import** {BrowserRouter} **from** 'react-router-dom';

оборачиваем наш главный компонент:

**const** Application = (  
 <Provider store={store}>  
 <BrowserRouter>  
 <App />  
 </BrowserRouter>  
 </Provider>  
);

В App.js настраиваем роутинг – какой компонент соответствует какой странице :

**import** {Route, Switch, Redirect, withRouter} **from** 'react-router-dom';

<Switch>  
 <Route path={MAIN} exact component={MainPage}/>  
 <Route path={ACCOUNT} component={Account}/>  
 <Route path={AUTH} component={SignUp}/>  
 <Route path={TEST} component={Test}/>  
 <Route path={RESULT} component={Result}/>  
 <Route path={EDIT\_QUESTIONS} component={EditQuestions}/>  
 <Route path={EDIT\_ANSWERS} component={EditAnswers}/>  
 <Route path={MILLIONAIRE} component={Millionare}/>  
 <Redirect to='/'/>  
</Switch>

Для того, чтобы иметь доступ до данных роутинга в пропсах компонента:

**export default** withRouter(App)

Также, здесь, при старте приложения определим какую страницу будем отрывать – если пользователь залогинен, если он является преподавателем, то в личный кабинет, если студентом, то на главную страницу для выбора прохождения теста. Если не залогинен, то то на страницу входа

**if** (**this**.props.user !== **null**)  
 **if** (isTeacher(**this**.props.user.email))  
 **this**.props.history.replace(ACCOUNT);  
 **else  
 this**.props.history.replace(MAIN);  
**else  
 this**.props.history.push(LOGIN);

13. Прикрутим к проекту redux

В файле index.js добавим redux который будет описывать наш стор (данные идействия над ними), middleware thunk для создания асинхронных действий (мы работаем с бд, поэтому действия по работе с ней должны быть асинхронны) и redux-devtools для отладки и тестирования redux в браузере

**import** {Provider} **from** 'react-redux';

**import** {applyMiddleware, createStore, compose} **from** 'redux';

**import** firebaseReducer **from** './Store/redusers/firebaseReducer';

**import** thunk **from** 'redux-thunk';

**const** composeEnhancers =  
 **typeof** window === 'object' &&  
 window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_COMPOSE\_\_ ?  
 window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_COMPOSE\_\_({  
 }) : compose;  
  
**const** store = createStore(firebaseReducer, composeEnhancers(applyMiddleware(  
 thunk  
)));  
  
  
**const** Application = (  
 <Provider store={store}>  
 <BrowserRouter>  
 <App />  
 </BrowserRouter>  
 </Provider>  
);

после этого у нас в каждом компоненте появляется возможность использования данных и действий над данными из стора. Для этого, импортируем в компоненте функцию

**import** {connect} **from** 'react-redux'

Создаем две функции. Первая, транслирует данные из стора в пропсы компонента:

**function** mapStateToProps(state) {  
 **return** {  
 user: state.user  
 }  
}

после этого, можно обращаться к **this**.props.user для получения доступа к свойству user из стора.

Вторая функция:

**function** mapDispatchToProps(dispatch) {  
 **return** {  
 initFirebase: () => dispatch(initFirebase()),  
 }  
}

она транслирует действия из стора в пропсы компонента. Чтобы вызвать данное действие нужно:

**this**.props.initFirebase();

Ну и для того, чтобы все это связать воедино, нужно обернуть экспорт нашего компонента:

**export default** connect(mapStateToProps, mapDispatchToProps)(withRouter(App))

Теперь опишем и реализуем наш стор. Создаем две папки в store: reusers – здесь будем создавать файлы, описывающие наши данные и модификацию этих данных (данные мы можем изменять мы можем только тут) и action – это действия, которые мы можем вызвать для работы с данными из стора (инициализация, редактирование,…)

В файле Action/actionTypes.js опишем все действия, которые присутсвуют в нашем сторе:

**export const** SET\_FIREBASE = 'SET\_FIREBASE';  
**export const** SET\_LOADER = 'SET\_LOADER';  
  
**export const** ADD\_QUIZ = 'ADD\_QUIZ';  
**export const** SET\_QUIZ = 'SET\_QUIZ';  
**export const** DEL\_QUIZ = 'DEL\_QUIZ';  
  
**export const** CLEAR\_QUESTIONS = 'CLEAR\_QUESTIONS';  
**export const** ADD\_QUESTIONS = 'ADD\_QUESTIONS';  
**export const** SET\_QUESTIONS = 'SET\_QUESTIONS';  
**export const** DEL\_QUESTIONS = 'DEL\_QUESTIONS';  
  
**export const** CLEAR\_ANSWERS = 'CLEAR\_ANSWERS';  
**export const** ADD\_ANSWERS = 'ADD\_ANSWERS';  
**export const** SET\_ANSWERS = 'SET\_ANSWERS';  
**export const** DEL\_ANSWERS = 'DEL\_ANSWERS';

Создаем файл firebaseReduser:

Импортируем наши действия:

**import** {  
 ADD\_ANSWERS,  
 ADD\_QUESTIONS,  
 ADD\_QUIZ, CLEAR\_ANSWERS,  
 CLEAR\_QUESTIONS, DEL\_ANSWERS, DEL\_QUESTIONS,  
 DEL\_QUIZ, SET\_ANSWERS,  
 SET\_FIREBASE, SET\_LOADER,  
 SET\_QUESTIONS,  
 SET\_QUIZ  
} **from** "../actions/actionTypes";

Создаем данные стора:

**const** initialState = {  
 db: **null**, // база данных  
 auth: **null**, // параметр аутентификации пользователя  
 error: **null**, // наличие какой либо ошибки  
 user: **null**, // пользователь, залогиненный в систему  
 listQuizes: [], // список тестов  
 questions: [], // список вопросов для теста  
 answers: [], // список возможных ответов на вопрос  
 loader: **false** // флаг загрузки данных  
  
};

Функция модификации данных (для понимания: clear- инициализация данных, add – добавить, del – удалить, set – модифицировать):

**export default function** firebaseReducer(state = initialState, action) {  
 **switch** (action.type) {  
 **case** SET\_FIREBASE: {  
 **return** {  
 ...state, ...action.params  
 };  
 }  
 **case** SET\_LOADER: {  
 **return** {  
 ...state, loader: action.params  
 };  
 }  
 **case** ADD\_QUIZ: {  
 **return** {  
 ...state  
 };  
 }  
 **case** SET\_QUIZ: {  
 **let** listQuizes = [...state.listQuizes];  
 listQuizes[action.payload.index] = action.payload.quiz;  
 **return** {  
 ...state, listQuizes: [...listQuizes]  
 };  
 }  
 **case** DEL\_QUIZ: {  
 **let** listQuizes = [...state.listQuizes];  
 listQuizes.splice(action.payload.index, 1);  
 **return** {  
 ...state, listQuizes: [...listQuizes]  
 }  
 }  
 **case** CLEAR\_QUESTIONS: {  
 **return** {  
 ...state, questions: []  
 }  
 }  
 **case** ADD\_QUESTIONS: {  
 **return** {  
 ...state  
 }  
 }  
 **case** SET\_QUESTIONS: {  
 **let** questions = [...state.questions];  
 questions[action.payload.index] = action.payload.question;  
 **return** {  
 ...state, questions: [...questions]  
 };  
 }  
 **case** DEL\_QUESTIONS: {  
 **let** questions = [...state.questions];  
 questions.splice(action.payload.index, 1);  
 **return** {  
 ...state, questions: [...questions]  
 }  
  
 }  
 **case** CLEAR\_ANSWERS: {  
 **return** {  
 ...state, answers: []  
 }  
 }  
 **case** ADD\_ANSWERS: {  
 **return** {  
 ...state  
 }  
 }  
 **case** SET\_ANSWERS: {  
 **let** answers = [...state.answers];  
 answers[action.payload.index] = action.payload.answer;  
 **return** {  
 ...state, answers: [...answers]  
 };  
 }  
 **case** DEL\_ANSWERS: {  
 **let** answers = [...state.answers];  
 answers.splice(action.payload.index, 1);  
 **return** {  
 ...state, answers: [...answers]  
 }  
  
 }  
 **default** :  
 **return** state  
 }  
}

в папке action:

файл actions.js – все действия, где не требуется ассинхронный код (во многом повторяет выше описанные действия):

**import** {  
 ADD\_ANSWERS,  
 ADD\_QUESTIONS,  
 ADD\_QUIZ, CLEAR\_ANSWERS,  
 CLEAR\_QUESTIONS, DEL\_ANSWERS,  
 DEL\_QUESTIONS,  
 DEL\_QUIZ, SET\_ANSWERS,  
 SET\_FIREBASE, SET\_LOADER,  
 SET\_QUESTIONS,  
 SET\_QUIZ  
} **from** "./actionTypes";  
  
**export function** setFirebasse(params) {  
 **return** {  
 type: SET\_FIREBASE,  
 params  
 }  
}  
**export function** setLoader(params) {  
 **return** {  
 type: SET\_LOADER,  
 params  
 }  
}  
**export function** addQuizInList(quiz) {  
 **return** {  
 type: ADD\_QUIZ,  
 quiz  
 }  
}  
**export function** setQuizInList(quiz, index) {  
 **return** {  
 type: SET\_QUIZ,  
 payload: {quiz, index}  
 }  
}  
**export function** delQuizInList(index) {  
 **return** {  
 type: DEL\_QUIZ,  
 payload: {index}  
 }  
}  
**export function** clearQuestionsList() {  
 **return** {  
 type: CLEAR\_QUESTIONS  
 }  
}  
**export function** addQuestionsInList(question) {  
 **return** {  
 type: ADD\_QUESTIONS,  
 question  
 }  
}  
**export function** setQuestionsInList(question, index) {  
 **return** {  
 type: SET\_QUESTIONS,  
 payload: {question, index}  
 }  
}  
**export function** delQuestionsInList(index) {  
 **return** {  
 type: DEL\_QUESTIONS,  
 payload: {index}  
 }  
}  
**export function** clearAnswersList() {  
 **return** {  
 type: CLEAR\_ANSWERS  
 }  
}  
**export function** addAnswersInList(answer) {  
 **return** {  
 type: ADD\_ANSWERS,  
 answer  
 }  
}  
**export function** setAnswersInList(answer, index) {  
 **return** {  
 type: SET\_ANSWERS,  
 payload: {answer, index}  
 }  
}  
**export function** delAnswersInList(index) {  
 **return** {  
 type: DEL\_ANSWERS,  
 payload: {index}  
 }  
}

Файл initFirebase – асинхронные функции для инициализхации бд:

**import** firebase **from** "firebase/app";  
**import** "firebase/auth";  
**import** "firebase/database";  
**import** {setFirebasse, setLoader} **from** "./actions";  
**import** {getAllQuiz, initDataUser} **from** "./quizFirebase";  
**import** {isTeacher} **from** "../helper";  
  
**export function** initFirebase() {  
 **return** (dispatch) => {  
 **return new** Promise((resolve) => {  
 **const** firebaseConfig = {  
 apiKey: "AIzaSyDenwr8WvOXeFCVtxcQT0x0DYzU64NBTsg",  
 authDomain: "react-dip.firebaseapp.com",  
 databaseURL: "https://react-dip.firebaseio.com",  
 projectId: "react-dip",  
 storageBucket: "react-dip.appspot.com",  
 messagingSenderId: "1028821584667",  
 appId: "1:1028821584667:web:70f1064583b68d084da1be"  
 };  
 // Initialize Firebase  
 dispatch(setLoader(**true**)); // покажем лоадер  
 **const** app = firebase.initializeApp(firebaseConfig);  
 **const** auth = app.auth();  
 // занесем в стор данные по подключению к firebase  
 dispatch(setFirebasse({  
 db: app.database(),  
 auth: auth  
 }));  
 // функция, которая сработает после того как произойдет аутентификация пользователя  
 auth.onAuthStateChanged(**async function**(user) {  
 **if** (user) {  
 // получили залогиненного пользователя. Занесем его в стор  
 **await** dispatch(setFirebasse({user}));  
 **if** (isTeacher(user.email))  
 // если данный пользователь - преподаватель, то обновим данные его личного кабинета  
 **await** dispatch(initDataUser());  
 **else** // иначе, заполним список доступных тестов для прохождения  
 **await** dispatch(getAllQuiz());  
 }  
 dispatch(setLoader(**false**)); // уберем лоадер  
 resolve()  
 });  
  
 })  
 // Your web app's Firebase configuration  
 }  
}

Файл authFirerbase – опишем функции для аутентификации пользователя:

**import** {setFirebasse} **from** "./actions";  
  
// Регистрация пользователя  
**export function** signUp(name, lname, email, password) {  
 **return function**(dispatch, getState) {  
 **return new** Promise((resolve, reject) => {  
 **let** {auth} = getState();  
 // Создаем в firebase нового пользователя  
 auth.createUserWithEmailAndPassword(email, password)  
 .then(user=>{  
 // если пользователь создан успешно, обновим его профиль  
 auth.currentUser.updateProfile({  
 displayName: name + ' ' + lname  
 }).then(**function**() {  
 // если все прошло успешно, обновим наш стор, указав ему нового пользователя  
 dispatch(setFirebasse({auth, user: {...auth.currentUser, displayName: name + ' ' + lname}}));  
 resolve()  
 }, **function**(error) {  
 // Если произошла ошибка обновления профиля  
 console.log(error);  
 reject(error);  
 });  
 })  
 .catch(**function**(error) {  
 // Ошибка создания пользователя (такой e-mail существует,...)  
 dispatch(setFirebasse({error}));  
 console.log(error.code);  
 console.log(error.message);  
 reject(error);  
 }  
 )  
 })  
 ;  
 }  
}  
  
// Вход пользователя в систему  
**export function** signIn(email, password) {  
 **return** (dispatch, getState) => {  
 **return new** Promise((resolve, reject) => {  
 **let** {auth} = getState();  
 // запрос на аутентификацию пользователя в firebase  
 auth.signInWithEmailAndPassword(email, password)  
 .then(()=>{  
 // пользователь аутентифицирован, обновим данные в сторе  
 dispatch(setFirebasse({auth, user: auth.currentUser}));  
 resolve();  
 })  
 .catch(**function**(error) {  
 // ошибка аутентификации (ошибка e-mail или пароля)  
 dispatch(setFirebasse({error}));  
 console.log(error.code);  
 console.log(error.message);  
 reject(error);  
 })  
 })  
 }  
}  
// Выход пользователя из приложения  
**export function** signOut() {  
 **return** (dispatch, getState) => {  
 **let** {auth} = getState();  
 // запрос в firebase на выход из приложения  
 auth.signOut()  
 .then(()=>{  
 // если все успешно, обновим данные в сторе  
 dispatch(setFirebasse({auth, user: auth.currentUser}))  
 })  
 .catch(**function**(error) {  
 // ошибка разлогирования  
 dispatch(setFirebasse({error}));  
 console.log(error.code);  
 console.log(error.message);  
 })  
 }  
}

Файл quizFirerbase – опишем функции для действий над тестами:

**import** {addQuizInList, delQuizInList, setFirebasse, setQuizInList} **from** "./actions";  
**import** {delAllQuestionsFromQuiz} **from** "./questionFirebase";  
  
// Добавить тест в список  
**export function** addQuiz(quiz) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд и зарегистрированного пользователя  
 **let** {db, user} = getState();  
 // получим ссылку в firebase на список тестов  
 **let** ref = db.ref('quiz');  
 // добавим тест в бд  
 ref.push({...quiz, uid: user.uid});  
 // добавим тест в стор  
 dispatch(addQuizInList(quiz));  
 resolve()  
 })  
 }  
}  
// модификация теста  
**export function** setQuiz(quiz, index) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 **let** key = quiz.key;  
 // получим ссылку на тест по ключу  
 **let** ref = db.ref('quiz/'+key);  
 // обновим данные теста в бд  
 ref.update({...quiz, key: **null**});  
 // обновим данные теста в сторе  
 dispatch(setQuizInList(quiz, index));  
 resolve()  
 })  
 }  
}  
// удаление теста  
**export function** delQuiz(quiz) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 **let** key = quiz.key;  
 // получим ссылку на тест по ключу  
 **let** ref = db.ref('quiz'+key);  
 // удалим тест из бд  
 ref.remove();  
 // удалим из стора все вопросы, относящиеся к этому тесту  
 dispatch(delAllQuestionsFromQuiz(key));  
 // удалим из стора сам тест  
 dispatch(delQuizInList(quiz.tableData.id));  
 resolve()  
 })  
 }  
}  
// инициализация списка тестов для залогиненного пользователя (в качестве преподователя)  
**export function** initDataUser() {  
 **return** (dispatch, getState) => {  
 // инициализация списка тестов в сторе  
 dispatch(setFirebasse({listQuizes: []}));  
 // получим данные бд и зарегистрированного пользователя  
 **let** {db, user} = getState();  
 // получим ссылку в firebase на список тестов  
 **let** ref = db.ref('quiz');  
 **let** listQuizes = [];  
 // получим из бд все тесты, которые относятся к данному пользователю  
 ref.orderByChild("uid").equalTo(user.uid).on("child\_added", **function**(data) {  
 **const** dv = data.val();  
 **const** dk = data.key;  
 listQuizes.push({...dv, key: dk});  
 // сформированный список тестов занесем в стор  
 dispatch(setFirebasse({listQuizes: [...listQuizes]}));  
 });  
 }  
}  
// получение всех тестов (для пользователей - студентов)  
**export function** getAllQuiz() {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // инициализация списка тестов в сторе  
 dispatch(setFirebasse({listQuizes: []}));  
 // получим данные бд  
 **let** {db} = getState();  
 // получим ссылку в firebase на список тестов  
 **let** ref = db.ref('quiz');  
 **let** listQuizes = [];  
 // получим из бд все тесты, которые там есть  
 ref.once("value").then(snapshot => {  
 Object.entries(snapshot.val()).forEach((key) => {  
 // Уберем из списка приватные тесты  
 **if** (key[1].access !== 2)  
 listQuizes.push({...key[1], key: key[0]})  
 });  
 // сформированный список тестов занесем в стор  
 dispatch(setFirebasse({listQuizes: [...listQuizes]}));  
 resolve();  
 });  
 });  
 }  
}

Файл questionFirerbase – опишем функции для действий над вопросами для тестов:

**import** {  
 addQuestionsInList,  
 clearQuestionsList,  
 delQuestionsInList,  
 setFirebasse, setQuestionsInList,  
} **from** "./actions";  
**import** {delAllAnswersFromQuestion} **from** "./answerFirebase";  
  
// инициализация списка вопросов теста (key - ключ теста)  
**export function** initQuestionsList(key) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 // получим ссылку в firebase на список вопросов к тесту  
 **let** ref = db.ref('questions');  
 // инициализация вопросов  
 dispatch(clearQuestionsList());  
 **let** questions = [];  
 // отфильтруем список вопросов по ключу теста  
 ref.orderByChild("keyQuiz").equalTo(key).on("child\_added", **function** (data) {  
 **const** dv = data.val();  
 **const** dk = data.key;  
 questions.push({...dv, key: dk});  
 // занесем в стор полученные вопросы  
 dispatch(setFirebasse({questions: [...questions]}));  
 });  
 resolve()  
 })  
 }  
}  
// добавление вопроса в списка вопросов теста (question - данные вопроса, keyQuiz - ключ теста)  
**export function** addQuestion(question, keyQuiz) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 // получим ссылку в firebase на список вопросов к тесту  
 **let** ref = db.ref('questions');  
 // добавим вопрос в бд  
 ref.push({...question, keyQuiz: keyQuiz});  
 // добавим вопрос в стор  
 dispatch(addQuestionsInList(question));  
 resolve()  
 })  
 }  
}  
// редактирование вопроса в списке вопросов теста (question - данные вопроса, index - индекс вопроса в списке)  
**export function** setQuestion(question, index) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 **let** key = question.key;  
 // получим ссылку в firebase на вопрос к тесту по ключу вопроса  
 **let** ref = db.ref('questions/' + key);  
 // обновим данные в бд  
 ref.update({...question, key: **null**});  
 // обновим данные в сторе  
 dispatch(setQuestionsInList(question, index));  
 resolve()  
 })  
 }  
}  
// удаление вопроса из списка вопросов теста (question - данные вопроса)  
**export function** delQuestion(question) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 **let** key = question.key;  
 // получим ссылку в firebase на вопрос к тесту по ключу вопроса  
 **let** ref = db.ref('questions/' + key);  
 // удалим вопрос из бд  
 ref.remove();  
 // удалим все ответы на данный вопрос  
 dispatch(delAllAnswersFromQuestion(key));  
 // удалим вопрос из стора  
 dispatch(delQuestionsInList(question.tableData.id));  
 resolve()  
 })  
 }  
}  
// удаление всех вопросов теста (key - ключ теста)  
**export function** delAllQuestionsFromQuiz(key) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 // получим ссылку в firebase на список вопросов к тесту  
 **let** ref = db.ref('questions');  
 // пробежимся по всем вопросам  
 ref.orderByChild("keyQuiz").equalTo(key).on("child\_added", **function** (data) {  
 **const** dk = data.key;  
 **let** ref = db.ref('questions/' + dk);  
 // удалим сам вопрос из бд  
 ref.remove();  
 // удалим все ответы на этот вопрос из бд  
 dispatch(delAllAnswersFromQuestion(dk));  
 });  
 resolve()  
 })  
 }  
}  
// получение всех вопросов для конкретного теста (keyQuiz - ключ теста)  
**export function** getQuestionsFromQuiz(keyQuiz) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // инициализация (очистка) вопросов в сторе  
 dispatch(clearQuestionsList());  
 // получим данные бд  
 **let** {db} = getState();  
 // получим ссылку в firebase на список вопросов к тесту  
 **let** ref = db.ref('questions');  
 **let** questions = [];  
 // отфильтруем вопросы по ключу теста и пробежимся по ним  
 ref.orderByChild("keyQuiz").equalTo(keyQuiz).once("value").then(snapshot => {  
 **if** (snapshot !== undefined && snapshot.val()) {  
 Object.entries(snapshot.val()).forEach((key) => {  
 questions.push({...key[1], key: key[0]})  
 });  
 }  
 // занесем найденные вопросы в стор  
 dispatch(setFirebasse({questions: [...questions]}));  
 resolve();  
 });  
 });  
 }  
}

Файл questionFirerbase – опишем функции для действий над возможными ответами для вопроса:

**import** {  
 addAnswersInList,  
 clearAnswersList, clearQuestionsList,  
 delAnswersInList,  
 setAnswersInList,  
 setFirebasse,  
} **from** "./actions";  
// инициализация списка ответов для вопроса (key - ключ вопроса)  
**export function** initAnswerList(key) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 // получим ссылку в firebase на список ответов к вопросу  
 **let** ref = db.ref('answers');  
 // инициализация списка ответов в сторе  
 dispatch(clearAnswersList());  
 **let** answers = [];  
 // отфильтруем список ответов по ключу вопрома и пробежимся по всем ответам  
 ref.orderByChild("keyQuestion").equalTo(key).on("child\_added", **function** (data) {  
 **const** dv = data.val();  
 **const** dk = data.key;  
 answers.push({...dv, key: dk});  
 // занесем очередной ответ в стор  
 dispatch(setFirebasse({answers: [...answers]}));  
 });  
 resolve()  
 })  
 }  
}  
// добавление ответа (answer - данные ответа, keyQuestion - ключ вопроса)  
**export function** addAnswer(answer, keyQuestion) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 // получим ссылку в firebase на список ответов к вопросу  
 **let** ref = db.ref('answers');  
 // добавим вопрос в бд  
 ref.push({...answer, keyQuestion: keyQuestion});  
 // обновим вопрос в сторе  
 dispatch(addAnswersInList(answer));  
 resolve()  
 })  
 }  
}  
// редактирование ответа (answer - данные ответа, index - индекс ответа в списке)  
**export function** setAnswer(answer, index) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 **let** key = answer.key;  
 // получим ссылку в firebase на ответ к вопросу по ключу вопроса  
 **let** ref = db.ref('answers/' + key);  
 // обновим ответ в бд  
 ref.update({...answer, key: **null**});  
 // обновим ответ в сторе  
 dispatch(setAnswersInList(answer, index));  
 resolve()  
 })  
 }  
}  
// удаление ответа (answer - данные ответа)  
**export function** delAnswer(answer) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 **let** key = answer.key;  
 // получим ссылку в firebase на ответ к вопросу по ключу вопроса  
 **let** ref = db.ref('answers/' + key);  
 // удалим ответ в бд  
 ref.remove();  
 // удалим ответ в сторе  
 dispatch(delAnswersInList(answer.tableData.id));  
 resolve()  
 })  
 }  
}  
// удаление всех ответов вопроса (key - ключ вопроса)  
**export function** delAllAnswersFromQuestion(key) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // получим данные бд  
 **let** {db} = getState();  
 // получим ссылку в firebase на список ответов к вопросу  
 **let** ref = db.ref('answers');  
 // отфильтруем ответы по ключу вопроса и пробежимся по ним  
 ref.orderByChild("keyQuestion").equalTo(key).on("child\_added", **function** (data) {  
 **const** dk = data.key;  
 **let** ref = db.ref('answers/' + dk);  
 // удалим вопрос из бд  
 ref.remove();  
 });  
 resolve()  
 })  
 }  
}  
// получение всех ответов для вопроса (keyQuestion - ключ вопроса)  
**export function** getAnswersFromQuestion(keyQuestion) {  
 **return** (dispatch, getState) => {  
 **return new** Promise(resolve => {  
 // инициализация списка ответов в сторе  
 dispatch(clearQuestionsList());  
 // получим данные бд  
 **let** {db} = getState();  
 // получим ссылку в firebase на список ответов к вопросу  
 **let** ref = db.ref('answers');  
 **let** answers = [];  
 // отфильтруем по ключу вопроса и пробежимся по списку ответов  
 ref.orderByChild("keyQuestion").equalTo(keyQuestion).once("value").then(snapshot => {  
 **if** (snapshot !== undefined && snapshot.val()) {  
 Object.entries(snapshot.val()).forEach((key) => {  
 answers.push({...key[1], key: key[0]})  
 });  
 }  
 // обновим данные ответа в сторе  
 dispatch(setFirebasse({answers: [...answers]}));  
 resolve();  
 });  
 });  
 }  
}

Для того, чтобы мы имели одну точку входа по всем действиям, создадим файл rootActions.js:

**export** \* **from** './actions';  
**export** \* **from** './authFirebase';  
**export** \* **from** './initFirebase';  
**export** \* **from** './quizFirebase';  
**export** \* **from** './questionFirebase';  
**export** \* **from** './answerFirebase';

15. Делаем футер. Создаем файл Components/Copyright/copyright.js:

**import** React **from** 'react';  
**import** Typography **from** '@material-ui/core/Typography';  
  
**export default function** Copyright() {  
 **return** (  
 <Typography variant="body2" color="textSecondary" align="center">  
 {'Copyright © Денисова Юлия Алексеевна '}  
 {**new** Date().getFullYear()}  
 {'.'}  
 </Typography>  
 );  
}

Тут все довольно просто: текст и текущий год

17. Добавим компоненты для регистрации/входа в приложение. Создадим файл Register/Register.js.

Необходимые импорты:

**import** React **from** 'react';  
**import** Avatar **from** '@material-ui/core/Avatar';  
**import** Button **from** '@material-ui/core/Button';  
**import** CssBaseline **from** '@material-ui/core/CssBaseline';  
**import** TextField **from** '@material-ui/core/TextField';  
**import** Copyright **from** '../../Components/Copyright/copyright'  
**import** Grid **from** '@material-ui/core/Grid';  
**import** Box **from** '@material-ui/core/Box';  
**import** LockOutlinedIcon **from** '@material-ui/icons/LockOutlined';  
**import** Typography **from** '@material-ui/core/Typography';  
  
**import** { withStyles } **from** "@material-ui/core/styles";  
**import** Container **from** '@material-ui/core/Container';  
**import** {signIn, signUp} **from** "../../Store/actions/rootActions";  
**import** {connect} **from** "react-redux";  
**import** {ACCOUNT, LOGIN, MAIN, REGISTER} **from** "../../Route/path";  
**import** {Link **as** LinkRouter} **from** 'react-router-dom'  
**import** Dialog **from** '@material-ui/core/Dialog';  
**import** DialogActions **from** '@material-ui/core/DialogActions';  
**import** DialogContent **from** '@material-ui/core/DialogContent';  
**import** DialogContentText **from** '@material-ui/core/DialogContentText';  
**import** DialogTitle **from** '@material-ui/core/DialogTitle';  
**import** {isTeacher} **from** "../../Store/helper";

Стили:

**const** useStyles = (theme) => ({  
 paper: {  
 marginTop: theme.spacing(8),  
 display: 'flex',  
 flexDirection: 'column',  
 alignItems: 'center',  
 },  
 avatar: {  
 margin: theme.spacing(1),  
 backgroundColor: theme.palette.secondary.main,  
 },  
 form: {  
 width: '100%', // Fix IE 11 issue.  
 marginTop: theme.spacing(3),  
 },  
 submit: {  
 margin: theme.spacing(3, 0, 2),  
 },  
 link: {  
 color: theme.palette.text.secondary,  
 }  
});

из стора подтянем

**function** mapStateToProps(state) {  
 **return** {  
 error: state.error, // ошибка  
 user: state.user // зарегистрированный пользователь  
 }  
}  
  
**function** mapDispatchToProps(dispatch) {  
 **return** {  
 signUpFirebase: (name, lname, email, password) => dispatch(signUp(name, lname, email, password)), // регистрация нового пользователя  
 signInFirebase: (email, password) => dispatch(signIn(email, password)) // вход зарегистрированного пользователя  
 }  
}

В конструкторе класса объявим стейт компонента и сделаем некоторые настройки

constructor(props) {  
 **super**(props);  
 // привяжем намертво контекст  
 **this**.handleInputChange = **this**.handleInputChange.bind(**this**); // можно было воспользоваться стрелочной ф-ей. Это чисто в учебных целях  
 **this**.typeAuth = **this**.props.match.params.name; // тип аутентификации - вход или регистрация  
 **this**.state = { // стейт компонента  
 firstName: '', // имя  
 lastName: '', // фамилия  
 email: '', // е-mail  
 password: '', // пароль  
 message: {}, // сообщение (либо ок, либо ошибка)  
 showDialog: **false**, // флаг показа диалогового окна  
 redirect: **false** // флаг, можно ли нам покинуть эту страницу  
 }  
}

Добавим несколько функций:

// если пользователь ткнул на ссылку под полем ввода пароля (там мы предлагаем ему сменить тип аутентификации)  
componentWillReceiveProps(nextProps, nextState) {  
 **this**.typeAuth = nextProps.match.params.name; // меняем тип аутентификации  
 **if** (nextState.password !== '')  
 **this**.setState({  
 password: '' // стираем пароль  
 });  
}

// при вводе с клавиатуры, сразу обновляем соответствующие значения в стейте   
handleInputChange(e) {  
 **this**.setState({  
 [e.target.name]: e.target.value  
 });  
}

// при нажатии на кнопку 'Зарегистрироваться'/'Войти'  
**async** onClickHandle() {  
 **let** title = 'Ошибка '; // заголовок ошибки  
 **try** {  
 **if** (**this**.typeAuth === 'reg') { // в зависимости от типа аутентификации  
 title += 'регистрации'; // меняем заголовок ошибки  
 **await this**.props.signUpFirebase( // запрос на регистрацию нового пользователя в firebase  
 **this**.state.firstName,  
 **this**.state.lastName,  
 **this**.state.email,  
 **this**.state.password  
 );  
  
 } **else** {  
 title += 'аутентификации'; // меняем заголовок ошибки  
 **await this**.props.signInFirebase( // запрос на вход зарегистрированного пользователя в firebase  
 **this**.state.email,  
 **this**.state.password  
 )  
 }  
 **this**.setState({ // если все отработало штатно и мы не получаем исключений  
 showDialog: **true**, // покажем приветственное окно  
 redirect: **true**, // после которого перейдем на другую станицу  
 message: { // сообщение, которое покажем пользователю  
 title: `Уважаемый, ${**this**.props.user.displayName}`,  
 message: 'Добро пожаловать в систему прохождения тестов. Очень надеюсь вам понравиться!!!',  
 }  
 });  
 // На этом этапе пользователь аутентифицировался, либо зарегистрировался  
 // можем загрузить из бд все данные пока он вдупляет на окно приветсвия и ищет кнопку ОК  
  
 } **catch** (e) { // если при аутентификации произошла ошибка  
 **this**.setState({  
 showDialog: **true**, // покажем ее пользователю  
 redirect: **false**, // запрет на покидание этой страницы  
 message: {...e, title}, // сообщение, которое покажем пользователю  
 })  
 }  
}

// при показе окна с сообщением для пользователя была нажата кнопка ОК  
handleCloseDialog() {  
 **this**.setState({  
 showDialog: **false** // уберем окно с экрана  
 });  
 **if** (**this**.state.redirect) { // если это приветственное окно об удачной аутентификации и нам разрешено закрыть эту страницк  
 **if** (isTeacher(**this**.props.user.email)) // если это преподаватель, переходим в его личный кабинет  
 **this**.props.history.push(ACCOUNT);  
 **else  
 this**.props.history.push(MAIN); // если это студент, идем на главную страницу  
 }  
};

// функция блокировки/разблокировки кнопки аутентификации в зависимости от введенных значений в поля формы  
enableButton() {  
 **if** (**this**.typeAuth === 'reg') { // если это регистрация, то должны быть заполнены все поля  
 **return** (  
 **this**.state.firstName !== '' &&  
 **this**.state.lastName !== '' &&  
 **this**.state.email !== '' &&  
 **this**.state.password !== ''  
 )  
 } **else** { // если это обычный вход, то должны быть заполнены e-mail и пароль  
 **return** (  
 **this**.state.email !== '' &&  
 **this**.state.password !== ''  
 )  
 }  
};

Ну и сам рендер страницы. Структура

**const** { classes } = **this**.props;  
  
 **return** (  
 <Container component="main" maxWidth="xs">  
 <CssBaseline />  
 <div className={classes.paper}>  
 <Avatar />  
 <Typography />  
 <form />  
 </div>  
 <Box mt={5}>  
 <Copyright />  
 </Box>  
 <Dialog/>  
 </Container>  
 );  
}

Avatar – компонент MU, картинка обрамленная кругом

<Avatar className={classes.avatar}>  
 <LockOutlinedIcon />  
</Avatar>

Typography – компонент MU, текст

<Typography component="h1" variant="h5">  
 {**this**.typeAuth === 'reg' ? 'Регистрация' : 'Авторизация'}  
</Typography>

В форме, уберем обработку по умолчанию, чтобы не презагружалась страница onSubmit={(e) => {e.preventDefault();}}

При помощи компонентов Grid из MU, сделаем сетку из 4 строк, причем первая строка будет разделена на две части, если большой монитор. Между 3 и 4 строкой, расположим кнопку

<Grid container spacing={2}>  
 <Grid item xs={12} sm={6} />  
 <Grid item xs={12} sm={6} />  
 <Grid item xs={12} />  
 <Grid item xs={12} />  
</Grid>  
<Button />  
<Grid container justify="flex-end">  
 <Grid />  
</Grid>

В первые элементы сетки поместим компонент TextField.

<TextField  
 autoComplete="fname" // есть ли функция автозаполнения  
 name="firstName" // наименование поля  
 variant="outlined" // вариант представления. Здесь с рамкой  
 required // флаг, поле обязательно для заполнения  
 fullWidth // флаг, что поле будет занимать всю длину родителя  
 id="firstName" // идентификатор поля  
 label="Имя" // заголовок поля  
 autoFocus // флаг, что поле имеет автофокус  
 disabled={**this**.typeAuth === 'log'} // поле заблокировано, если ип аутентификации не регистрация  
 value={**this**.state.firstName} // значение поля  
 onChange={**this**.handleInputChange} // функция, которая будет вызваапри вводе с клавиатуры  
/>

Остальные поля заполняем по аналогии:

<Grid item xs={12} sm={6}>  
 <TextField  
 autoComplete="fname" // есть ли функция автозаполнения  
 name="firstName" // наименование поля  
 variant="outlined" // вариант представления. Здесь с рамкой  
 required // флаг, поле обязательно для заполнения  
 fullWidth // флаг, что поле будет занимать всю длину родителя  
 id="firstName" // идентификатор поля  
 label="Имя" // заголовок поля  
 autoFocus // флаг, что поле имеет автофокус  
 disabled={**this**.typeAuth === 'log'} // поле заблокировано, если ип аутентификации не регистрация  
 value={**this**.state.firstName} // значение поля  
 onChange={**this**.handleInputChange} // функция, которая будет вызваапри вводе с клавиатуры  
 />  
</Grid>  
<Grid item xs={12} sm={6}>  
 <TextField  
 variant="outlined"  
 required  
 fullWidth  
 id="lastName"  
 label="Фамилия"  
 name="lastName"  
 autoComplete="lname"  
 disabled={**this**.typeAuth === 'log'}  
 value={**this**.state.lastName}  
 onChange={**this**.handleInputChange}  
 />  
</Grid>  
<Grid item xs={12}>  
 <TextField  
 variant="outlined"  
 required  
 fullWidth  
 id="email"  
 label="Email адрес"  
 name="email"  
 type="email"  
 autoComplete="email"  
 autoFocus={**this**.typeAuth === 'log'}  
 value={**this**.state.email}  
 onChange={**this**.handleInputChange}  
 />  
</Grid>  
<Grid item xs={12}>  
 <TextField  
 variant="outlined"  
 required  
 fullWidth  
 name="password"  
 label="Пароль"  
 type="password"  
 id="password"  
 autoComplete="current-password"  
 value={**this**.state.password}  
 onChange={**this**.handleInputChange}  
 />  
</Grid>

Далее располагается кнопка, при нажатии на которую происходи попытка входа/регитрации пользователя:

<Button  
 type="submit"  
 fullWidth  
 variant="contained"  
 color="primary"  
 className={classes.submit}  
 onClick={**this**.onClickHandle.bind(**this**)}  
 disabled={!**this**.enableButton()}  
>  
 {**this**.typeAuth === 'reg' ? 'Зарегистрироваться' : 'Войти'}  
</Button>

Внизу располагается ссылка на смену типа аутентификации:

<Grid container justify="flex-end">  
 <Grid item>  
 <LinkRouter to={**this**.typeAuth === 'reg'? LOGIN : REGISTER} className={classes.link} activeclassname={classes.link} >  
 {**this**.typeAuth === 'reg' ? 'Уже зарегистрированы? Войти' : 'Не зарегистрированы? Регистрация'}  
 </LinkRouter>  
 </Grid>  
</Grid>

После формы следует компонент Copyright, который показывает футер:

<Box mt={5}>  
 <Copyright />  
</Box>

Dialog – компонент MU. По умолчанию не видим. Если нужно вывести его на экран, меняем свойство open. В нашем случае, за отображение этого окна отвечает поле showDialog из стейта. Заголовок и текст в окне мы также храним в стейте. По нажатию на кнопку ОК, вызвается метод handleCloseDialog, который мы описывали выше. Таким образом:

<Dialog  
 open={**this**.state.showDialog}  
 onClose={()=>{**this**.handleCloseDialog()}}  
 aria-labelledby="alert-dialog-title"  
 aria-describedby="alert-dialog-description"  
>  
 <DialogTitle id="alert-dialog-title">{**this**.state.message.title}</DialogTitle>  
 <DialogContent>  
 <DialogContentText id="alert-dialog-description">  
 {**this**.state.message.message}  
 </DialogContentText>  
 </DialogContent>  
 <DialogActions>  
 <Button onClick={()=>{**this**.handleCloseDialog()}} color="primary" autoFocus >  
 OK  
 </Button>  
 </DialogActions>  
</Dialog>

16. Добавим компоненты для личного кабинета. Все они довольно однотипны: имеют заголовок и табличную часть. Заголовок отражает к чему относятся данные в табличной части. Данные в таблице отражают составные части тестов – список самих тестов, далее, если провалиться в сам тест, в таблице будут отражены все его вопросы, ну и далее, если провалиться в какой-либо вопрос, мы увидим все возможные ответы на него. Причем здесь можно задать правильные ответы (можно несколько). Если не будет отмечено ни одного правильного ответа, считается, что его нет (при прохождении тестирования, среди вероятных ответов будет добавлен ответ «Среди предложенных вариантов нет правильного», который и будет являться правильным).

Разберем один из компонентов. Для остальных отметим только отличия.

Подключаем необходимые импорты

**import** React, {forwardRef} **from** 'react';  
**import** Container **from** '@material-ui/core/Container';  
**import** Box **from** '@material-ui/core/Box';  
**import** Typography **from** '@material-ui/core/Typography';  
**import** CssBaseline **from** '@material-ui/core/CssBaseline';  
**import** Card **from** '@material-ui/core/Card';  
**import** MaterialTable **from** 'material-table'  
**import** Chip **from** '@material-ui/core/Chip';  
**import** Button **from** '@material-ui/core/Button';  
**import** TextField **from** '@material-ui/core/TextField';  
**import** Dialog **from** '@material-ui/core/Dialog';  
**import** DialogActions **from** '@material-ui/core/DialogActions';  
**import** DialogContent **from** '@material-ui/core/DialogContent';  
**import** DialogContentText **from** '@material-ui/core/DialogContentText';  
**import** DialogTitle **from** '@material-ui/core/DialogTitle';  
  
**import** AddBox **from** '@material-ui/icons/AddBox';  
**import** ArrowDownward **from** '@material-ui/icons/ArrowDownward';  
**import** Check **from** '@material-ui/icons/Check';  
**import** ChevronLeft **from** '@material-ui/icons/ChevronLeft';  
**import** ChevronRight **from** '@material-ui/icons/ChevronRight';  
**import** Clear **from** '@material-ui/icons/Clear';  
**import** DeleteOutline **from** '@material-ui/icons/DeleteOutline';  
**import** Edit **from** '@material-ui/icons/Edit';  
**import** FilterList **from** '@material-ui/icons/FilterList';  
**import** FirstPage **from** '@material-ui/icons/FirstPage';  
**import** LastPage **from** '@material-ui/icons/LastPage';  
**import** Remove **from** '@material-ui/icons/Remove';  
**import** SaveAlt **from** '@material-ui/icons/SaveAlt';  
**import** Search **from** '@material-ui/icons/Search';  
**import** ViewColumn **from** '@material-ui/icons/ViewColumn';  
**import** ListAltIcon **from** '@material-ui/icons/ListAlt';  
  
**import** Grid **from** '@material-ui/core/Grid';  
**import** Copyright **from** "../../Components/Copyright/copyright";  
**import** {connect} **from** "react-redux";  
**import** {withStyles} **from** "@material-ui/core/styles/index";  
**import** {addQuiz, setQuiz, delQuiz} **from** "../../Store/actions/rootActions";  
**import** {EDIT\_QUESTIONS\_PATH} **from** "../../Route/path";

Стили:

**const** useStyles = theme => ({  
 root: {  
 display: 'flex',  
 flexWrap: 'wrap',  
 '& > \*': {  
 margin: theme.spacing(2),  
 width: theme.spacing(200),  
 },  
 },  
 card: {  
 marginTop: theme.spacing(4),  
 height: theme.spacing(16),  
 },  
 cardTable:{  
 width: '100%'  
 },  
 gridRow: {  
 display: 'flex',  
 alignItems: "center",  
 height: '100%'  
 },  
 gridCol: {  
 marginLeft: 20,  
 }  
});

Подключаем иконки, которые мы будем использовать в компоненте таблицы:

**const** tableIcons = {  
 Add: forwardRef((props, ref) => <AddBox {...props} ref={ref}/>),  
 Check: forwardRef((props, ref) => <Check {...props} ref={ref}/>),  
 Clear: forwardRef((props, ref) => <Clear {...props} ref={ref}/>),  
 Delete: forwardRef((props, ref) => <DeleteOutline {...props} ref={ref}/>),  
 DetailPanel: forwardRef((props, ref) => <ChevronRight {...props} ref={ref}/>),  
 Edit: forwardRef((props, ref) => <Edit {...props} ref={ref}/>),  
 Export: forwardRef((props, ref) => <SaveAlt {...props} ref={ref}/>),  
 Filter: forwardRef((props, ref) => <FilterList {...props} ref={ref}/>),  
 FirstPage: forwardRef((props, ref) => <FirstPage {...props} ref={ref}/>),  
 LastPage: forwardRef((props, ref) => <LastPage {...props} ref={ref}/>),  
 NextPage: forwardRef((props, ref) => <ChevronRight {...props} ref={ref}/>),  
 PreviousPage: forwardRef((props, ref) => <ChevronLeft {...props} ref={ref}/>),  
 ResetSearch: forwardRef((props, ref) => <Clear {...props} ref={ref}/>),  
 Search: forwardRef((props, ref) => <Search {...props} ref={ref}/>),  
 SortArrow: forwardRef((props, ref) => <ArrowDownward {...props} ref={ref}/>),  
 ThirdStateCheck: forwardRef((props, ref) => <Remove {...props} ref={ref}/>),  
 ViewColumn: forwardRef((props, ref) => <ViewColumn {...props} ref={ref}/>)  
};

Подключаем из стора некоторые поля и действия:

**function** mapStateToProps(state) {  
 **return** {  
 data: state.listQuizes, // список тестов  
 user: state.user, // пользователь, зарегистрированный в системе  
 }  
}  
  
  
**function** mapDispatchToProps(dispatch) {  
 **return** {  
 addQuiz: (quiz) => dispatch(addQuiz(quiz)), // добавить тест   
 setQuiz: (quiz, index) => dispatch(setQuiz(quiz, index)), // редактировать тест  
 delQuiz: (quiz) => dispatch(delQuiz(quiz)), // удалить тест  
 }  
}

В конструкторе класса опишем стейт компонента:

constructor(props) {  
 **super**(props);  
 **this**.state = {  
 columns: [ // Заголовки колонок таблицы  
 {  
 title: 'Наименование теста', field: 'name', render: rowData =>  
 <Typography component="p" variant="subtitle1">  
 {rowData.name}  
 </Typography>  
 },  
 {  
 title: 'Видимость', field: 'access',  
 render: rowData => {  
 **switch** (rowData.access) {  
 **case** 0:  
 **return** <Chip style={{fontSize: 16}} label="Публичный" color="primary"/>;  
 **case** 1:  
 **return** <Chip style={{fontSize: 16}} label="Ограниченный"/>;  
 **case** 2:  
 **return** <Chip style={{fontSize: 16}} label="Приватный" color="secondary"/>;  
 **default**:  
 **return null**;  
 }  
 },  
 lookup: {0: 'Публичный', 1: 'Ограниченный', 2: 'Приватный'},  
 initialEditValue: '1',  
 type: 'numeric',  
 align: 'right'  
 },  
 ],  
 open: **false**, // отвечает за видимость модального окна - нужно, чтобы ввести пароль видимости теста  
 rowData: **null** // данные, которые мы видим в таблице  
 }  
}

СтруктураЖ

<Container component="main">  
 <CssBaseline/>  
 <div className={classes.root}>  
 <Card />  
 <Card />  
 </div>  
 <Box mt={5}>  
 <Copyright/>  
 </Box>  
 <Dialog />  
</Container>

Card – компонент MU. На верхней карточке расположим информацию, на нижней -табличную часть. В нашем случае, на верхней части будет располагаться фио преподавателя, а в нижней части – те тесты, которые он завел.

Верхняя карточка:

<Card className={classes.card}>  
 <Grid container spacing={2} className={classes.gridRow}>  
 {**this**.props.user ?  
 <React.Fragment>  
 <Grid item xs={5} sm={2} className={classes.gridCol}>  
 <Typography component="p" variant="h6" color="textSecondary">  
 Преподаватель:  
 </Typography>  
 </Grid>  
 <Grid item xs={12} sm={9}>  
 <Typography component="p" variant="h5" color="textPrimary">  
 {**this**.props.user.displayName}  
 </Typography>  
 </Grid>  
 </React.Fragment>  
 : **null** }  
 </Grid>  
</Card>

Нижняя карточка включает в себя компонент MaterialTable. Это очень гибкий инструмент для показа таблиц на странице. Разберем все используемые нами свойства:

<MaterialTable  
 title="Тесты" // заголовок таблицы  
 icons={tableIcons} // иконки, которые использут компонент для отображения действий  
 columns={**this**.state.columns} // заголовки колонок  
 data={**this**.props.data} // данные, которые отображаются в таблице  
 options={{ // опции таблицы  
 pageSizeOptions: [5, 10, 20], // количество строк на странице (выпадающий список)  
 headerStyle: {fontSize: 16, fontWeight: 600} // стиль отображения заголовков  
 }}  
 actions={[ // нестандартные действия пользователя  
 {  
 icon: ListAltIcon, // иконка  
 tooltip: 'Редактирование вопросов', // подсказка  
 onClick: (event, rowData) => { // что делаем при нажатии на иконку  
 **this**.props.history.push(EDIT\_QUESTIONS\_PATH + '/' + rowData.key); // открываем страницу с редактированием вопросов к тесту  
 }  
 },  
 rowData => ({  
 icon: ChevronRight, // иконка  
 tooltip: 'Изменить пароль доступа', // подсказка  
 onClick: (event, rowData) => **this**.setState({open: **true**, rowData: rowData}), // вызов окна ввода пароля  
 disabled: rowData.access !== 1 // если тип доступа не ограниченный, данная возможность заблокируется  
 })  
 ]}  
 localization={{ // перевод некоторых сообщений  
 body: {  
 emptyDataSourceMessage: 'Данных нет. Для добавления теста нажмите кнопку со знаком +',  
 addTooltip: 'Добавить новый тест',  
 deleteTooltip: 'Удалить',  
 editTooltip: 'Изменить',  
 filterRow: {  
 filterTooltip: 'Отфильтровать'  
 },  
 editRow: {  
 deleteText: 'Вы уверены, что хотите удалить тест? Он будет удален безвозвратно!',  
 cancelTooltip: 'Отменить',  
 saveTooltip: 'ОК'  
 }  
 },  
 grouping: {  
 placeholder: 'Группировать ...',  
 groupedBy: 'Группировка по:'  
 },  
 header: {  
 actions: 'Действия'  
 },  
 pagination: {  
 labelDisplayedRows: '{from}-{to} из {count}',  
 labelRowsSelect: 'строк',  
 labelRowsPerPage: 'строк на странице:',  
 firstAriaLabel: 'Первая страница',  
 firstTooltip: 'Перейти на первую страницу',  
 previousAriaLabel: 'предыдущая страница',  
 previousTooltip: 'Перейти на предыдущую страницу',  
 nextAriaLabel: 'следующая страница',  
 nextTooltip: 'Перейти на следующую страницу',  
 lastAriaLabel: 'последняя страница',  
 lastTooltip: 'Перейти на последнюю страницу'  
 },  
 toolbar: {  
 exportTitle: 'Экспорт',  
 exportAriaLabel: 'Экспорт',  
 exportName: 'Экспорт в CSV',  
 searchTooltip: 'Поиск тестов по вхождению',  
 searchPlaceholder: 'Поиск'  
 }  
 }}  
 editable={{ // стандартные действия  
 onRowAdd: newData => // добавление строки  
 **new** Promise(**async** (resolve) => {  
 **await this**.props.addQuiz({...newData, access: parseInt(newData.access)}); // добавляем тест в бд  
 resolve();  
 }),  
 onRowUpdate: (newData, oldData) => // редактирование строки  
 **new** Promise((resolve) => {  
 setTimeout(() => {  
 **this**.props.setQuiz({...newData, access: parseInt(newData.access)}, oldData.tableData.id); // редактируем тест в бд  
 resolve();  
 }, 0)  
 }),  
 onRowDelete: oldData => // удаление строки  
 **new** Promise(**async** (resolve) => {  
 **await this**.props.delQuiz(oldData); // удаление теста из бд  
 resolve()  
 }),  
 }}  
/>

Далее идет компонент Copyright

Далее компонент Dialog, предназначенный для ввода пароля к тесту. В DialogTitle вводим заголовок. В DialogContent помещаем два компонента: текст DialogContentText и поле ввода пароля TextField

<DialogTitle id="form-dialog-title">Изменить пароль</DialogTitle>  
<DialogContent>  
 <DialogContentText>  
 Доступ к прохождению теста может быть ограничен паролем, который будет запрошен у пользователя.  
 </DialogContentText>  
 <TextField  
 autoFocus  
 margin="dense"  
 id="password"  
 label="Пароль"  
 type="password"  
 fullWidth  
 />  
</DialogContent>

В поле DialogActions помещаем две кнопки. Первая кнопка отменяет ввод пароля и закрывает окно, при нажтии на вторую кнопку заносим новый пароль в базу данных

<DialogActions>  
 <Button onClick={() => **this**.setState({open: **false**})} color="primary">  
 Не хочу менять  
 </Button>  
 <Button onClick={() => {  
 **let** passw = document.getElementById("password");  
 **this**.props.setQuiz({  
 ...**this**.state.rowData,  
 access: parseInt(**this**.state.rowData.access),  
 tableData: **null**, password: passw.value  
 },  
 **this**.state.rowData.tableData.id);  
 **this**.setState({open: **false**})  
 }  
 } color="primary">  
 Изменить  
 </Button>  
</DialogActions>

По аналогии с данным компонентом реализуем остальные компоненты по редактированию вопросов EditQuestions и ответов EditAnswers.

У компонента редактирования вопросов в верхнюю часть добавляется наименование теста

<Grid item xs={5} sm={2} className={classes.gridCol}>  
 <Typography component="p" variant="body1" color="textSecondary">  
 Тест:  
 </Typography>  
</Grid>  
<Grid item xs={12} sm={9}>  
 <Typography component="p" variant="body1" color="textPrimary">  
 {**this**.state.nameQuiz}  
 </Typography>  
</Grid>

В MaterialTable добавляется свойство, которое показывает кнопку в заголовке, для возврата назад к тестам

components={{  
 Toolbar: props => (  
 <div>  
 <MTableToolbar {...props} />  
 <Button  
 type="button"  
 variant="contained"  
 color="primary"  
 className={classes.button}  
 onClick={()=>**this**.props.history.push(ACCOUNT)}  
 >  
 {"Обратно к списку тестов"}  
 </Button>  
 </div>  
 ),  
}}

И соответсвенно подключаются другие действия и данные из стора

**function** mapStateToProps(state) {  
 **return** {  
 listQuizes: state.listQuizes, // список тестов  
 data: state.questions, // список вопросов  
 user: state.user, // пользователь  
 }  
}  
  
  
**function** mapDispatchToProps(dispatch) {  
  
 **return** {  
 initQuestionsList: (key) => dispatch(initQuestionsList(key)), // инициализация списка вопросов теста   
 addQuestion: (question, keyQuiz) => dispatch(addQuestion(question, keyQuiz)), // добавить вопрос  
 setQuestion: (question, index) => dispatch(setQuestion(question, index)), // редактировать вопрос  
 delQuestion: (keyQuiz) => dispatch(delQuestion(keyQuiz)), // удалить вопрос  
 }  
}

у списка вопросов в верхней части отобразим вопрос

<Grid item xs={5} sm={2} className={classes.gridCol}>  
 <Typography component="p" variant="body1" color="textSecondary">  
 Вопрос:  
 </Typography>  
</Grid>  
<Grid item xs={12} sm={9}>  
 <Typography component="p" variant="body1" color="textPrimary">  
 {**this**.state.nameQuestion}  
 </Typography>  
</Grid>

При нажатии на любую строку будем помечать ее5 зелным цветом и заносить в базу данный ответ как правилдьный

onRowClick={((evt, selectedRow) => {  
 **this**.props.setAnswer({...selectedRow, rightAnswer: !selectedRow.rightAnswer, tableData:**null**}, selectedRow.tableData.id);  
})}

Что подтягиваем из стора

**function** mapStateToProps(state) {  
 **return** {  
 listQuizes: state.listQuizes, // список тестов  
 questions: state.questions, // список вопросов  
 data: state.answers, // список ответов  
 user: state.user, // пользователь  
 }  
}  
  
  
**function** mapDispatchToProps(dispatch) {  
  
 **return** {  
 initAnswerList: (key) => dispatch(initAnswerList(key)), // инициализация списка ответов  
 addAnswer: (answer, keyQuestion) => dispatch(addAnswer(answer, keyQuestion)), // добавить ответ  
 setAnswer: (answer, index) => dispatch(setAnswer(answer, index)), // редактировать ответ  
 delAnswer: (keyQuiz) => dispatch(delAnswer(keyQuiz)) // удалить ответ  
 }  
}

Теперь разработаем компонент для главного экрана. Он предназначен для всех пользователей кроме преподавателей. На данной странице мы увидем все тесты, которым преподователи дали доступ либо публичный, либо ограниченный. При ограниченном доступе у пользователя будет запрошен пароль для прохождения теста. Создадим файл Containers/Main.js

Импорт необходимых компонентов:

**import** React, {forwardRef} **from** 'react';  
**import** Container **from** '@material-ui/core/Container';  
**import** Box **from** '@material-ui/core/Box';  
**import** Typography **from** '@material-ui/core/Typography';  
**import** CssBaseline **from** '@material-ui/core/CssBaseline';  
**import** MaterialTable **from** 'material-table'  
**import** Button **from** '@material-ui/core/Button';  
**import** TextField **from** '@material-ui/core/TextField';  
**import** Dialog **from** '@material-ui/core/Dialog';  
**import** DialogActions **from** '@material-ui/core/DialogActions';  
**import** DialogContent **from** '@material-ui/core/DialogContent';  
**import** DialogContentText **from** '@material-ui/core/DialogContentText';  
**import** DialogTitle **from** '@material-ui/core/DialogTitle';  
**import** CircularProgress **from** '@material-ui/core/CircularProgress';  
  
**import** AddBox **from** '@material-ui/icons/AddBox';  
**import** ArrowDownward **from** '@material-ui/icons/ArrowDownward';  
**import** Check **from** '@material-ui/icons/Check';  
**import** ChevronLeft **from** '@material-ui/icons/ChevronLeft';  
**import** ChevronRight **from** '@material-ui/icons/ChevronRight';  
**import** Clear **from** '@material-ui/icons/Clear';  
**import** DeleteOutline **from** '@material-ui/icons/DeleteOutline';  
**import** Edit **from** '@material-ui/icons/Edit';  
**import** FilterList **from** '@material-ui/icons/FilterList';  
**import** FirstPage **from** '@material-ui/icons/FirstPage';  
**import** LastPage **from** '@material-ui/icons/LastPage';  
**import** Remove **from** '@material-ui/icons/Remove';  
**import** SaveAlt **from** '@material-ui/icons/SaveAlt';  
**import** Search **from** '@material-ui/icons/Search';  
**import** ViewColumn **from** '@material-ui/icons/ViewColumn';

**import** Copyright **from** "../../Components/Copyright/copyright";  
**import** {connect} **from** "react-redux";  
**import** {withStyles} **from** "@material-ui/core/styles/index";  
**import** { TEST\_PATH} **from** "../../Route/path";  
**import** {withRouter} **from** 'react-router-dom'

Стили:

**const** useStyles = theme => ({  
 root: {  
 display: 'flex',  
 flexWrap: 'wrap',  
 '& > \*': {  
 margin: theme.spacing(2),  
 width: theme.spacing(200),  
 },  
 },  
 card: {  
 marginTop: theme.spacing(4),  
 height: theme.spacing(16),  
 },  
 gridRow: {  
 display: 'flex',  
 alignItems: "center",  
 height: '100%'  
 },  
 gridCol: {  
 marginLeft: 20,  
 },  
 button: {  
 margin: theme.spacing(0, 0, 2, 3),  
  
 },  
 circularProgress: {  
 display: 'flex',  
 alignItems: "center",  
 justifyContent: 'center',  
 height: '100vh'  
 }  
});

иконки для таблицы:

**const** tableIcons = {  
 Add: forwardRef((props, ref) => <AddBox {...props} ref={ref}/>),  
 Check: forwardRef((props, ref) => <Check {...props} ref={ref}/>),  
 Clear: forwardRef((props, ref) => <Clear {...props} ref={ref}/>),  
 Delete: forwardRef((props, ref) => <DeleteOutline {...props} ref={ref}/>),  
 DetailPanel: forwardRef((props, ref) => <ChevronRight {...props} ref={ref}/>),  
 Edit: forwardRef((props, ref) => <Edit {...props} ref={ref}/>),  
 Export: forwardRef((props, ref) => <SaveAlt {...props} ref={ref}/>),  
 Filter: forwardRef((props, ref) => <FilterList {...props} ref={ref}/>),  
 FirstPage: forwardRef((props, ref) => <FirstPage {...props} ref={ref}/>),  
 LastPage: forwardRef((props, ref) => <LastPage {...props} ref={ref}/>),  
 NextPage: forwardRef((props, ref) => <ChevronRight {...props} ref={ref}/>),  
 PreviousPage: forwardRef((props, ref) => <ChevronLeft {...props} ref={ref}/>),  
 ResetSearch: forwardRef((props, ref) => <Clear {...props} ref={ref}/>),  
 Search: forwardRef((props, ref) => <Search {...props} ref={ref}/>),  
 SortArrow: forwardRef((props, ref) => <ArrowDownward {...props} ref={ref}/>),  
 ThirdStateCheck: forwardRef((props, ref) => <Remove {...props} ref={ref}/>),  
 ViewColumn: forwardRef((props, ref) => <ViewColumn {...props} ref={ref}/>)  
};

В конструкторе класса опишем заголовки столбцов таблицы:

constructor(props) {  
 **super**(props);  
 **this**.state = {  
 columns: [  
 {  
 title: 'Тест', field: 'name', render: rowData =>  
 <Typography component="p" variant="subtitle1">  
 {rowData.name}  
 </Typography>  
 },  
 ],  
 open: **false**,  
 password: **null**,  
 key: ''  
 }  
}

Подключим необходимые данные из стора

**function** mapStateToProps(state) {  
 **return** {  
 data: state.listQuizes, // список тестов  
 questions: state.questions, // список вопросов  
 user: state.user, // пользователь  
 loader: state.loader // флаг, говорящий что система занята  
 }  
}

Рендер страницы:

Здесь применим прием ожидания загрузки страницы и данных. Если у нас нет данных или свойство loader, говорящий нам о том, что система сейчас занята (пытается прочитать данные из бд) то покажем некий индикатор загрузки

{!**this**.props.data || **this**.props.loader ?  
 <div className={classes.circularProgress}>  
 <CircularProgress />  
 </div>

CircularProgress – компонент MU, представлюящий из себя индикатор загрузки

Если все готово к показу данных, покажем таблицу

<div className={classes.root}>  
 <MaterialTable  
 title="Список доступных тестов" // Загловок таблицы  
 icons={tableIcons} // Иконки таблицы  
 columns={**this**.state.columns} // Заголовки столбцов таблицы  
 data={**this**.props.data} // Данные, которые будут отображены в строках таблицы  
 options={{ // Опции  
 pageSizeOptions: [5, 10, 20], // Количество строк на экране  
 headerStyle: {fontSize: 16, fontWeight: 600}, // Стили заголовков таблицы  
 }}  
 onRowClick={((evt, selectedRow) => { // При нажатии на строку таблицы  
 **if** (selectedRow.access === 1) { // Если доступ до теста - ограниченный  
 **this**.setState({ // Запросим пароль  
 open: **true**,  
 password: selectedRow.password ? selectedRow.password : '123456',  
 key: selectedRow.key  
 })  
 } **else** // иначе, сразу перейдем на страницу для прохождения теста  
 **this**.props.history.push(TEST\_PATH + '/' + selectedRow.key);  
 })}  
 localization={{ // перевод основных сообщений  
 body: {  
 emptyDataSourceMessage: 'Тестов для прохождения нет. ',  
 addTooltip: 'Добавить новый тест',  
 deleteTooltip: 'Удалить',  
 editTooltip: 'Изменить',  
 filterRow: {  
 filterTooltip: 'Отфильтровать'  
 },  
 editRow: {  
 deleteText: 'Вы уверены, что хотите удалить ответ? Он будет удален безвозвратно!',  
 cancelTooltip: 'Отменить',  
 saveTooltip: 'ОК'  
 }  
 },  
 grouping: {  
 placeholder: 'Группировать ...',  
 groupedBy: 'Группировка по:'  
 },  
 header: {  
 actions: 'Действия'  
 },  
 pagination: {  
 labelDisplayedRows: '{from}-{to} из {count}',  
 labelRowsSelect: 'строк',  
 labelRowsPerPage: 'строк на странице:',  
 firstAriaLabel: 'Первая страница',  
 firstTooltip: 'Перейти на первую страницу',  
 previousAriaLabel: 'предыдущая страница',  
 previousTooltip: 'Перейти на предыдущую страницу',  
 nextAriaLabel: 'следующая страница',  
 nextTooltip: 'Перейти на следующую страницу',  
 lastAriaLabel: 'последняя страница',  
 lastTooltip: 'Перейти на последнюю страницу'  
 },  
 toolbar: {  
 nRowsSelected: '{0} Ответ(ов) укзаны как правильные',  
 exportTitle: 'Экспорт',  
 exportAriaLabel: 'Экспорт',  
 exportName: 'Экспорт в CSV',  
 searchTooltip: 'Поиск вопросов по вхождению',  
 searchPlaceholder: 'Поиск'  
 }  
 }}  
 />  
</div>

Покажем Copyright

<Box mt={5}>  
 <Copyright/>  
</Box>

Диалоговое окно для ввода пароля. В DialogTitle вводим заголовок. В DialogContent помещаем два компонента: текст DialogContentText и поле ввода пароля TextField

<Dialog open={**this**.state.open} onClose={() => **this**.setState({open: **false**})} aria-labelledby="form-dialog-title">  
 <DialogTitle id="form-dialog-title">Пароль</DialogTitle>  
 <DialogContent>  
 <DialogContentText>  
 Доступ к прохождению теста ограничен паролем. Введите пароль для рпазблокировки теста.  
 </DialogContentText>  
 <TextField  
 autoFocus  
 margin="dense"  
 id="password"  
 label="Пароль"  
 type="password"  
 fullWidth  
 />  
 </DialogContent>

В поле DialogActions помещаем две кнопки. Первая кнопка отменяет ввод пароля и закрывает окно, при нажтии на вторую кнопку проверяем на правильность ввода и если пароль правильные переходим на страницу прохождения теста, иначе выводим предупреждение

<DialogActions>  
 <Button onClick={() => **this**.setState({open: **false**})} color="primary">  
 Не знаю пароль  
 </Button>  
 <Button onClick={() => {  
 **let** passw = document.getElementById("password");  
 **this**.setState({open: **false**});  
 **if** (passw.value === **this**.state.password) {  
 **this**.props.history.push(TEST\_PATH + '/' + **this**.state.key);  
 } **else** {  
 alert('Пароль введен неправильный')  
 }  
 }  
 } color="primary">  
 ОК  
 </Button>  
 </DialogActions>  
</Dialog>

Создадим компонент для прохождения теста. Файл Test/Test.js

Необходимый импорт

**import** React, {forwardRef} **from** 'react';  
**import** Container **from** '@material-ui/core/Container';  
**import** Box **from** '@material-ui/core/Box';  
**import** Typography **from** '@material-ui/core/Typography';  
**import** CssBaseline **from** '@material-ui/core/CssBaseline';  
**import** Card **from** '@material-ui/core/Card';  
**import** MaterialTable, {MTableToolbar} **from** 'material-table'  
**import** CircularProgress **from** '@material-ui/core/CircularProgress';  
**import** Button **from** '@material-ui/core/Button';  
  
**import** AddBox **from** '@material-ui/icons/AddBox';  
**import** ArrowDownward **from** '@material-ui/icons/ArrowDownward';  
**import** Check **from** '@material-ui/icons/Check';  
**import** ChevronLeft **from** '@material-ui/icons/ChevronLeft';  
**import** ChevronRight **from** '@material-ui/icons/ChevronRight';  
**import** Clear **from** '@material-ui/icons/Clear';  
**import** DeleteOutline **from** '@material-ui/icons/DeleteOutline';  
**import** Edit **from** '@material-ui/icons/Edit';  
**import** FilterList **from** '@material-ui/icons/FilterList';  
**import** FirstPage **from** '@material-ui/icons/FirstPage';  
**import** LastPage **from** '@material-ui/icons/LastPage';  
**import** Remove **from** '@material-ui/icons/Remove';  
**import** SaveAlt **from** '@material-ui/icons/SaveAlt';  
**import** Search **from** '@material-ui/icons/Search';  
**import** ViewColumn **from** '@material-ui/icons/ViewColumn';  
  
**import** Grid **from** '@material-ui/core/Grid';  
**import** Copyright **from** "../../Components/Copyright/copyright";  
**import** {connect} **from** "react-redux";  
**import** {withStyles} **from** "@material-ui/core/styles/index";  
**import** {getAnswersFromQuestion, getQuestionsFromQuiz} **from** "../../Store/actions/rootActions";  
**import** {MAIN, RESULT} **from** "../../Route/path";

Стили:

**const** useStyles = theme => ({  
 root: {  
 display: 'flex',  
 flexWrap: 'wrap',  
 '& > \*': {  
 margin: theme.spacing(2),  
 width: theme.spacing(200),  
 },  
 },  
 card: {  
 marginTop: theme.spacing(4),  
 height: theme.spacing(16),  
 display: 'flex',  
 justifyContent: "center",  
 flexDirection: 'column',  
 },  
 cardTable: {  
 width: '100%'  
 },  
 titleHead: {  
 marginLeft: theme.spacing(-1),  
 color: theme.palette.primary.dark,  
 },  
 wrapperDescription: {  
 margin: theme.spacing(-2, 3, 2, 2),  
 display: 'flex',  
 justifyContent: 'space-between',  
 },  
 description: {},  
 gridCol: {  
 marginLeft: theme.spacing(2),  
 },  
 button: {  
 margin: theme.spacing(0, 0, 2, 2),  
  
 },  
 circularProgress: {  
 display: 'flex',  
 alignItems: "center",  
 justifyContent: 'center',  
 height: '100vh'  
 },  
 warning: {  
 display: 'flex',  
 alignItems: "center",  
 justifyContent: 'center',  
 height: '100vh',  
 },  
 cardWarning: {  
 display: 'flex',  
 flexDirection: 'column',  
 alignItems: "center",  
 justifyContent: 'center',  
 height: theme.spacing(30),  
 width: theme.spacing(80),  
 },  
 textWarning: {  
 textAlign: 'center',  
 marginBottom: theme.spacing(5),  
 }  
});

Иконки для таблицы

**const** tableIcons = {  
 Add: forwardRef((props, ref) => <AddBox {...props} ref={ref}/>),  
 Check: forwardRef((props, ref) => <Check {...props} ref={ref}/>),  
 Clear: forwardRef((props, ref) => <Clear {...props} ref={ref}/>),  
 Delete: forwardRef((props, ref) => <DeleteOutline {...props} ref={ref}/>),  
 DetailPanel: forwardRef((props, ref) => <ChevronRight {...props} ref={ref}/>),  
 Edit: forwardRef((props, ref) => <Edit {...props} ref={ref}/>),  
 Export: forwardRef((props, ref) => <SaveAlt {...props} ref={ref}/>),  
 Filter: forwardRef((props, ref) => <FilterList {...props} ref={ref}/>),  
 FirstPage: forwardRef((props, ref) => <FirstPage {...props} ref={ref}/>),  
 LastPage: forwardRef((props, ref) => <LastPage {...props} ref={ref}/>),  
 NextPage: forwardRef((props, ref) => <ChevronRight {...props} ref={ref}/>),  
 PreviousPage: forwardRef((props, ref) => <ChevronLeft {...props} ref={ref}/>),  
 ResetSearch: forwardRef((props, ref) => <Clear {...props} ref={ref}/>),  
 Search: forwardRef((props, ref) => <Search {...props} ref={ref}/>),  
 SortArrow: forwardRef((props, ref) => <ArrowDownward {...props} ref={ref}/>),  
 ThirdStateCheck: forwardRef((props, ref) => <Remove {...props} ref={ref}/>),  
 ViewColumn: forwardRef((props, ref) => <ViewColumn {...props} ref={ref}/>)  
};

подключим необходимые данные и действия из стора

**function** mapStateToProps(state) {  
 **return** {  
 listQuizes: state.listQuizes, // список тестов  
 questions: state.questions, // список вопросов  
 answers: state.answers, // список ответов  
 user: state.user, // пользователь  
 }  
}  
  
  
**function** mapDispatchToProps(dispatch) {  
  
 **return** {  
 getAnswersFromQuestion: (keyQuestion) => dispatch(getAnswersFromQuestion(keyQuestion)), // Получение всех ответов по ключу вопроса  
 getQuestionsFromQuiz: (keyQuiz) => dispatch(getQuestionsFromQuiz(keyQuiz)) // Получение всех вопросов по ключу теста  
 }  
}

В контрукторе класса опишем стейт компонента

constructor(props) {  
 **super**(props);  
 **this**.state = {  
 columns: [ // Заголовки таблицы  
 {  
 title: 'Варианты ответов:', field: 'name', render: rowData =>  
 <Typography component="p" variant="subtitle1">  
 {rowData.name}  
 </Typography>  
 },  
 ],  
 nameQuiz: '', // Наименование теста  
 data: [], // Данные теста  
 indexCurrentQuestion: 0, // Текущий вопрос теста  
 loader: **false** // Флаг занятости системы  
 }  
}

При старте страницы заполним поле data данными теста. Формат данных будет примерно таким:

[

{ name: ‘Вопрос 1’,

answers: [

{

name: ‘Первый вариант ответа’,

rightAnswer: true,

answerUser: true

},

{…}

]

},

{…}

]

Где rightAnswer – флаг, является ли данный вариант ответа правильным, answerUser – выбрал ли пользователь данный вариант ответа.

// при старте страницы  
**async** componentDidMount() {  
 **this**.setState({  
 loader: **true** });  
 **await this**.props.getQuestionsFromQuiz(**this**.props.match.params.name); // Очистим список вопросов и заполним список вопросами редактируемого теста  
 **let** data = [];  
 **for** (**const** question **of this**.props.questions) { // пробежимся по всем вопросам теста  
 **await this**.props.getAnswersFromQuestion(question.key); // получим для каждого вопроса список вариантов ответа  
 **if** (**this**.props.answers.length > 0) { // если на данный вопрос есть хотя бы один вариант ответа, продолжаем, иначе, данный вопрос будет проигнорирован  
 **let** answers = [...**this**.props.answers]; // получим список ответов на данный вопрос  
 answers.forEach(item => item.answerUser = **false**); // проинициализируем все пользовательские ответы, как отрицательные  
 **let** rightEmptyAnswer = answers.findIndex(item => item.rightAnswer) === -1; // если при создании теста не был выбран правильный ответ  
 data.push({  
 ...question,  
 answers: [...**this**.props.answers,  
 { // к каждому вопросу добавим вариант еще один вариант ответа  
 name: 'Среди предложенных вариантов нет правильного',   
 rightAnswer: rightEmptyAnswer,  
 answerUser: **false** }]  
 })  
 }  
 }  
 // обновим стейт полученными данными  
 **this**.setState({  
 indexCurrentQuestion: 0,  
 data: data,  
 nameQuiz: **this**.props.listQuizes.find(item => item.key === **this**.props.match.params.name).name  
 });  
 **this**.setState({  
 loader: **false** });  
}

Опишем сообщение которое увидит пользователь, если попытается выбрать тесть, где нет ни одного вопроса

warningMessage() {  
 **const** {classes} = **this**.props;  
 **return** (  
 <div>  
 <Container component="main">  
 <CssBaseline/>  
 <div className={classes.warning}>  
 <Card className={classes.cardWarning}>  
 <Typography component="p" variant="h6" color="textPrimary" className={classes.textWarning}>  
 К сожалению, тест на данную тему сейчас не доступен. Попробуйте пройти его немного позже.  
 </Typography>  
 <Button  
 type="button"  
 variant="contained"  
 color="primary"  
 className={classes.button}  
 onClick={() => {  
 **this**.props.history.push(MAIN);  
 }}  
 >  
 {"Вернуться к списку тестов"}  
 </Button>  
 </Card>  
 </div>  
 </Container>  
 </div>  
 )  
}

Рендер компонента

Если система занята и нет данных, покажем индикатор загрузки

{!**this**.state.data || **this**.state.loader  
 ?  
 <div className={classes.circularProgress}>  
 <CircularProgress/>  
 </div>

Если был выбран тест, в котором нет вопросов покажем сообщение

: **this**.state.data.length === 0  
 ? **this**.warningMessage()

Если все ок начинаем прохождение теста. Верстка страницы сделана по аналогии с личным кабинетом. Сверху отображается сводная информация о тесте

<Card className={classes.card}>  
 <Grid item className={classes.gridCol}>  
 <Typography component="p" variant="body1" color="textSecondary">  
 Тестирование на тему:  
 </Typography>  
 </Grid>  
 <Grid item className={classes.gridCol}>  
 <Typography component="p" variant="h5" color="primary">  
 {**this**.state.nameQuiz}  
 </Typography>  
 </Grid>  
</Card>

В нижней части располагается таблица с очередным вопросом и вариантами ответов. Пользователь может мышью выбрать как ему кажется правильный вариант, который будет подсвечен другим цветом. Используем все тот же компонент MaterialTable. Кроме стандартных настроек отметим следующие

options={{  
 pageSizeOptions: [5],  
 search: **false**,  
 paging: **false**,  
 headerStyle: {fontSize: 16, fontWeight: 600},  
 rowStyle: rowData => {  
 **return** {backgroundColor: (rowData.answerUser) ? '#6acc75' : '#FFF'}  
 }  
}}

количество строк выводимых в таблице ограничим 5, уберем функцию поиска, уберем функцию пагинации, отмеченную пользователем строку выделим другим цветом

При кликании мышью по тому или иному варианту ответа, подсветим его. Если же он уже был подсвечен, то наоборот, подсветку убираем. При этом есть проверка, если вариант ответа один в этом вопросе, то пользовательский ответ будет переключаться между разными вариантами ответа, если же вариантов ответа много, позволим пользователю отметить несколько ответов. Все действия пользователя фиксируем в стейте

onRowClick={((evt, selectedRow) => {  
 **let** data = [...**this**.state.data];  
 **let** answerUser = data[**this**.state.indexCurrentQuestion].answers[selectedRow.tableData.id].answerUser;  
 **if** ((**this**.state.data[**this**.state.indexCurrentQuestion].answers.reduce((total, item) =>  
 total += item.rightAnswer ? 1 : 0, 0) === 1)) {  
 data[**this**.state.indexCurrentQuestion].answers.forEach((item, index, obj) => {  
 obj[index].answerUser = **false** });  
 }  
 data[**this**.state.indexCurrentQuestion].answers[selectedRow.tableData.id].answerUser = !answerUser;  
 **this**.setState({  
 data: data  
 });  
})}

Переопределим заголовок таблицы. Выведем туда информацию о количестве пройденных вопросов, а также информацию о необходимости выбора ответа – содержит ответ один иди несколько правильных ответов

components={{  
 Toolbar: props => (  
 <Grid container>  
 <Grid item xs>  
 <div className={classes.titleHead}>  
 <MTableToolbar {...props}/>  
 </div>  
 <div className={classes.wrapperDescription}>  
 <Typography component="p" variant="body1" className={classes.description}  
 color="textSecondary">  
 {(**this**.state.data[**this**.state.indexCurrentQuestion].answers.reduce((total, item) =>  
 total += item.rightAnswer ? 1 : 0, 0) === 1) ? 'Выберите один вариант ответа' : 'Выберите несколько вариантов ответа'}  
 </Typography>  
 <Typography component="p" variant="body1" className={classes.description}  
 color="textSecondary">  
 {`${**this**.state.indexCurrentQuestion + 1} вопрос из ${**this**.state.data.length}`}  
 </Typography>  
  
 </div>

также поместим туда кнопку, при нажатии на которую будет осуществлен переход на следующий вопрос теста. Если данный вопрос последний, то будет переход на страницу показа результата прохождения тестирования

<Button  
 type="button"  
 variant="contained"  
 color="primary"  
 className={classes.button}  
 disabled={!**this**.state.data[**this**.state.indexCurrentQuestion].answers.find(item => item.answerUser)}  
 onClick={() => {  
 **if** (**this**.state.indexCurrentQuestion === **this**.state.data.length - 1) {  
 **this**.props.history.push({  
 pathname: RESULT,  
 state: {data: **this**.state.data, quiz: **this**.state.nameQuiz}  
 });  
 }  
 **this**.setState({  
 indexCurrentQuestion: **this**.state.indexCurrentQuestion + 1  
 })  
 }}  
>  
 {**this**.state.indexCurrentQuestion < **this**.state.data.length - 1 ? "Следующий вопрос" : "Завершить тестирование"}  
</Button>

Остальные настройки такие же и для других компонент.

---------------- ПЕРЕНЕСТИ В НАЧАЛО

В качестве бэкенд сервера будем использовать систему Firebase от компании Google. Для того, чтобы создать проект в fb необходимо создать аккаунт в gmail. Заходим на сайт

[firebase.google.com авторизуемся под gmail-аккаунтом и переходим в консоль (верхний правый угол). Жмем на кнопку создать проект. Вводим имя (react-dip). От дополнительных опций (Google Аналитика) отказываемся. Жмем на кнопку Создать проект. После создания убедимся, что был создан проект с бесплатным тарифом Spark.](https://firebase.google.com/" \t "_blank)

Слева, на вкладке Develop выбираем Real Database. Далее жмем на кнопку Создать базу данных.

Выбираем тестовый режим

Переносим конфигурацию в проект

Устанавливаем режим аутентификации

Устанавливаем правила для фильтрации, иначе будет сильно ругаться

Структура базы данных. По данной структуре будет заполняться бд.

- quiz: {

"access" : 2, // тип доступа

"name" : "Компьютерная сеть. Топология сети", // наименование теста

"uid" : "H2FfnJfpDKfT2THljCGtNeCbyh73" // идентификатор пользователя, кому принадлежит тест

“[password](https://console.firebase.google.com/project/react-dip/database/react-dip/data/quiz/-MKRW9c61plRhnbYXGpX/password)”: “1” // пароль от теста

}

- [questions](https://console.firebase.google.com/project/react-dip/database/react-dip/data/questions): {

"name" : "Сколько байт в бите?", // Наименование вопроса

"keyQuiz" : "-MK7mB5W2y0YnTZW7N2G", // ключ теста

}

[answers](https://console.firebase.google.com/project/react-dip/database/react-dip/data/answers): {

"keyQuestion" : "-MKRWpjPjK7WqRzGw5Wh", // ключ вопроса

"name" : " for i:=1 to 10 do", // текст ответа

"rightAnswer" : false // является ли данный ответ правильным

}

npm install -g firebase-tools