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';

14. Добавим в стор тестовые данные для страницы со входом

15. Сверстаем страницу

16. Добавим в стор тестовые данные для страницы с регистрацией

17. Сверстаем страницу

18. Добавим в стор тестовые данные для страницы с прохождением теста

19. Сверстаем страницу главную и сверстаем страницу с прохождением теста

19.1 Сверстать страницу с результатом прохождения теста

19.2 Продумать над тем, что бы страница с результатами была единой для всех (показать пользователю результат прохождения теста и показать преподу результат прохождения его тестов всеми студентами)

20. Добавим в стор тестовые данные для страницы личным кабинетом

21. Сделаем мультикомпонент с таблицей

22. сверстаем страницу

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