# FIREBASE 101

## СТАРТ

Это NoSQL БД

1. Регаемся на сайте, создаем новый проект.
2. Project overview -> Database -> Начать -> Правила

Тут можно установить правила доступа, пока поставим публичным

{

"rules": {

".read": true,

".write": true

}

}

1. Project overview -> Добавьте Firebase в свое веб-приложение - копируем конфигурационный код оттуда
2. Инсталлим firebase через yarn:

yarn add firebase

1. Создаем папку firebase/firebase.js в проекте, туда пишем:

import \* as firebase from 'firebase';

const config = {

apiKey: "AIzaSyCVJC43rOE3hlgUid3JzzL3-4uZs17ItJs",

...

};

firebase.initializeApp(config);

const database = firebase.database();

database().ref().set({ name: 'Alex' });

1. Импортируем файл в файл приложение, которое можно запустить в браузере, что бы вызвать инициализировать его.
2. После этого, на сайте можно посмотреть, успешно ли прошло соединение:

Develop -> Database -> тут должно быть отображено имя БД: Alex

## МЕТОДЫ

Методы описаны в разделе References, раздел Javascript.

<https://firebase.google.com/docs/reference/js/?authuser=0>

ref - сослаться на корень если без аргумента, или на объект внутри БД

set - записать\перезаписать (если ref без аргумента) или заменить (если ref с аргументом)

Записать в корень БД следующий объект:

database.ref().set({

name: 'Alex',

isSingle: false,

location: {

city: 'Kiev'

}

});

Изменить город для вложенного объекта:

database.ref('location/city').set('New York');

**Добавить** новый вложенный объект

database.ref('attributes').set({

height: '193sm',

weight: '76kg',

}).then(() => {

console.log('saved into DB')

}).catch((e) => {

console.log('Error: ', e)

});

**Удалить** элемент

database.ref('isSingle').set(null);

database.ref('isSingle').remove();

**Update** элемент - одно значение изменить\добавить, второе удалить, третье - обновить вложенный объект, свойство city. Без перечисления через слеш, вложенный объект просто будет перезаписан

database.ref().update({

name: 'Mike',

isSingle: null,

'location/city': 'Boston'

});

**Извлечь** данных происходит при помощи метода once, в который передается евент. Про типы эвентов можно прочитать в документации

database

.ref()

.once('value')

.then((snapshot) => {

console.log(snapshot.val());

});

**Назначить прослушку** изменений объекта, позже снять ее. Для назначения мультипрослушки на один объект, можно назначить имя функции, которую возвращает вызов on():

const onValueChange = database.ref().on('value', (snapshot) => {

console.log(snapshot.val());

}, (err) => {

console.log('Error with data fetching', err);

});

setTimeout(() => database.ref('location/city').set('Dubai'), 3500);

setTimeout(() => database.ref().off(onValueChange), 7000);

## ОРГАНИЗАЦИЯ ХРАНЕНИЯ ДАННЫХ

Firebase не поддерживает массивы. При попытке запихнуть массив в БД, ошибки не будет, но он трансформируется в объект, где индекс массива становится названием объекта. Потому можно позволить Firebase самому генерировать id как название обьектов:

database.ref('notes').push({

title: 'Todo',

body: 'Go for a run'

});

Что бы превратить свойства объекта назад в массив, есть встроенный в snapshot метод forEach:

const expensesWathcer = database.ref('expenses').on('value', (snapshot) => {

const expenses = [];

snapshot.forEach((childSnapshot) => {

expenses.push({

id: childSnapshot.key, // firebase id

...childSnapshot.val()

})

});

console.log(expenses);

});

Так же есть другие эвенты для on() метода

* chilld\_removed - snapshot возвратит удаленный обьект
* child\_changed - возвращает измененный обьект
* child\_added - вызывается уже для существующих обьектов, и для новых
* child\_moved - порядок сортировки изменен

## ВКЛЮЧЕНИЕ АУТЕНТИФИКАЦИИ В FIREBASE

Заходим на продакшн-версию проекта, там заходим во вкладку authentication, выбираем любую соц сеть - сохранить.

Теперь в конфигурационный файл **firebase/firebase.js** добавим функцию:

const googleAuthProvider = new firebase.auth.GoogleAuthProvider();

export {firebase, googleAuthProvider, database as default};

В actiuons создадим файл **actions/auth.js,** где создадим функцию startLogin для логина через firebase:

import {firebase, googleAuthProvider} from '../firebase/firebase';

export const startLogIn = () => {

return () => {

return firebase.auth().signInWithPopup(googleAuthProvider);

};

};

Теперь можно создать компонент, который будет логинить: **components/LoginPage.js**

import React from 'react';

import { connect } from 'react-redux';

import { startLogIn } from "../actions/auth";

export const LoginPage = ({startLogIn}) => (

<div>

<button onClick={startLogIn}>Login</button>

</div>

);

const mapDispatchTpProps = (dispatch) => ({

startLogIn: () => dispatch(startLogIn())

});

export default connect(undefined, mapDispatchTpProps)(LoginPage);

Теперь при клике, будет открываться попап окно с логином через гугел

Для того, что бы отследить процесс логина\логаута, в можно в корневой файл добавить обработчик эвента, в данном случае в **app.js**

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

firebase.auth().onAuthStateChanged((user) => {

if(user){

console.log('log in')

} else {

console.log('log out')

}

});

**ПРИВАТНЫЙ ДОСТУП К БД**

Допустим мы изменили структуру данных в БД. Теперь она выглядит таким образом:

{

users:{

'userId1':{

expenses:{

'expenseNum1':{

amount: 123,

description: 'some note',

note: '',

createdAd: 30000

}

}

}

}

}

Мы хотим ограничить права записи\чтения только если userId отвечает тому, под которым юзер логинится в БД. Идем в **Rules**, меняем:

root - чтение\запись запрещена, работает только с обьектом users

"users":

* "expenses" проверка, что юзер записывает именно в этот обьект,
* "$other" - проверка любых других записей, кроме expenses в обьект users, запрещена
* newData - переменная для входящего значения, которое будем тестировать

**{**

**"rules": {**

**".read": false,**

**".write": false,**

**"users": {**

**"$user\_id": {**

**".read": "$user\_id === auth.uid",**

**".write": "$user\_id === auth.uid",**

**"expenses": {**

**"$expense\_id":{**

**".validate": "newData.hasChildren(**

**['description', 'note', 'createdAt', 'amount']**

**)",**

**"description":{**

**".validate": "newData.isString() && newData.val().length > 0"**

**},**

**"note":{**

**".validate": "newData.isString()"**

**},**

**"createdAt":{**

**".validate": "newData.isNumber()"**

**},**

**"amount":{**

**".validate": "newData.isNumber()"**

**},**

**"$other": {**

**".validate": false**

**}**

**}**

**},**

**"$other": {**

**".validate": false**

**}**

**}**

**}**

**}**

**}**

Теперь в симуляции справа, можно потестить разные ситуации. Вводим в строку ссылки наподобие такого:

/users/fb3fa64a-135d-45e1-97fc-6cd4e756d5da/expenses/abc123

В данные вводим напободие такого:

{

"description": "trulala",

"note": "",

"createdAt": 123213,

"amount": 123213

}

Тестим. Все должно подойти. Если данные не соответствуют, будут видны ошибки, в каких местах стоят запреты

## ВКЛЮЧЕНИЕ ВОЗМОЖНОСТИ ДОСТУПА С ДРУГОГО URL

По умолчанию в Firebase аутентификация работает на локалхост и на предложенный ими урл. Что бы аутентификация работала и для другого урла (напр. Heroku - прим. переводчика), его необходимо добавить.

Authentication - способ входа - снизу "добавить домен":

en1ight-expensify-app.herokuapp.com

После этого подождать полминуты