**暂时用一个变量存储用户信息**

**const** express = require('express');  
  
**const** app = express();  
**const** database = {  
 users: [  
 {  
 id:'123',  
 name:'FC',  
 email:'FC@123.com',  
 password:'cookies',  
 entries:0,  
 joined:**new** Date() // javascript  
 },  
 {  
 id:'321',  
 name:'CF',  
 email:'CF@123.com',  
 password:'cookies',  
 entries:0,  
 joined:**new** Date() // javascript  
 }  
 ]  
};

app.get('/',(req, res) => {  
 res.send('this is working');  
});  
  
app.post('/signin',(req, res) => {  
 //res.send('signing');  
 res.json('signing');  
});  
  
app.listen(3000, () => {  
 console.log('app is runing on post 3000')  
});

更改post判断登陆信息(目前暂时和database变量中第一个比较)

app.post('/signin',(req, res) => {  
 //res.send('signing');  
 **if** (req.body.email === database.users[0].email &&   
 req.body.password === database.users[0].password) {  
 res.json('success');  
 } **else** {  
 res.state(400).json('error logging in');//此时并没有处理error log in的情况，会直接报错  
 }  
 res.json('signing');  
});

添加新用户信息（注册）

app.post('/register', (req, res) => {  
 **const** {email, name, password} = req.body;  
 database.users.push({  
 id:'123',  
 name:name,  
 email:email,  
 password:password,  
 entries:0,  
 joined:**new** Date() // javascript  
 })  
 res.json(database.users[database.users.length-1]); // 不写res会一直pending  
});

使用Bcrypt加密密码后保存在数据库

修改前端signin模块，先变成class。

本来只传递signin中两个input到app.js，现在需要捕获这两个input（保存在类的态变量里）传递给app和server

onEmailChange = (event) => {  
 **this**.setState({signInEmail: event.target.value})  
}  
  
onPasswordChange = (event) => {  
 **this**.setState({signInPassword: event.target.value})  
}

<input/>中加入

onChange={**this**.onEmailChange}

password同理

使用fetch与server端通信

onSubmitSignIn = () => {  
 fetch('http://localhost:3000/signin', {  
 method: 'post',  
 headers: {'Content-Type': 'application/json'},  
 body: JSON.stringify({  
 email: **this**.state.signInEmail,  
 password: **this**.state.signInPassword  
 })  
 })  
 .then(response => response.json()) //  
 .then(user => {  
 **if** (user.id) {  
 **this**.props.loadUser(user);  
 **this**.props.onRouteChange('home');  
 }  
 })  
};

类似地，修改register。

在register点击提交应该有一个函数将用户信息传回app（signin同理）

在app中创建user变量保存用户信息，保存当前登陆者的信息

在app中创建loadUser()：使用在<register>中接收到的信息更新app中的user变量信息

将user中的姓名与entry传给rank.js

修改onButtonSubmit()：取回API结果后，用fetch向服务器调用image，同时计数+1

Data Base

createdb 'smart-brain'

psql 'smart-brain'

CREATE TABLE users(

id serial PRIMARY key,

name varchar(100),

email text UNIQUE NOT NULL,

entries BIGINT DEFAULT 0,

joined TIMESTAMP NOT NULL

);

CREATE TABLE login(

id serial PRIMARY key,

hash VARCHAR(100) NOT NULL,

email text UNIQUE NOT NULL

);

Server安装knex，与数据库通信

npm install knex

npm install knex pg

改写后端/register，将数据insert到数据库

app.post('/register', (req, res) => {  
 **const** {email, name, password} = req.body;  
 db('users').insert({  
 email: email,  
 name: name,  
 joined: **new** Date()  
 });  
 res.json(database.users[database.users.length-1]); // 不写res会一直pending  
});

改写/profile 从数据库中查找用户信息

改写/image 更新数据库中的计数

改写/register利用bcrypt加密信息，然后更新login表

改写/signin-根据邮箱查找用户-使用bcrypt比较密码-给前端回信

将API调用部分移到后端：将图片url发到后端，新增/imageurl来接收url并调用API，成功返回结果后再调用/image增加计数

部署改动：

前端app signin register中fetch操作的地址

后端kenx连接db的属性