

项目构成

1. 使用 express 后端框架
2. mongodb 数据库
3. session 和 cookie 用于管理会话，保存登录信息
4. 前后台密码加密传输，后台使用 NodeRSA，前台使用 jsencrypt，非对称加密
5. 后台存储密码采用 hash 加 salt，加密存储密码

运行说明

启动 mongodb 并创建名为 express 的 db，名为 users 的 collection，mongodb 数据库端口为 mongodb://localhost:27017/express

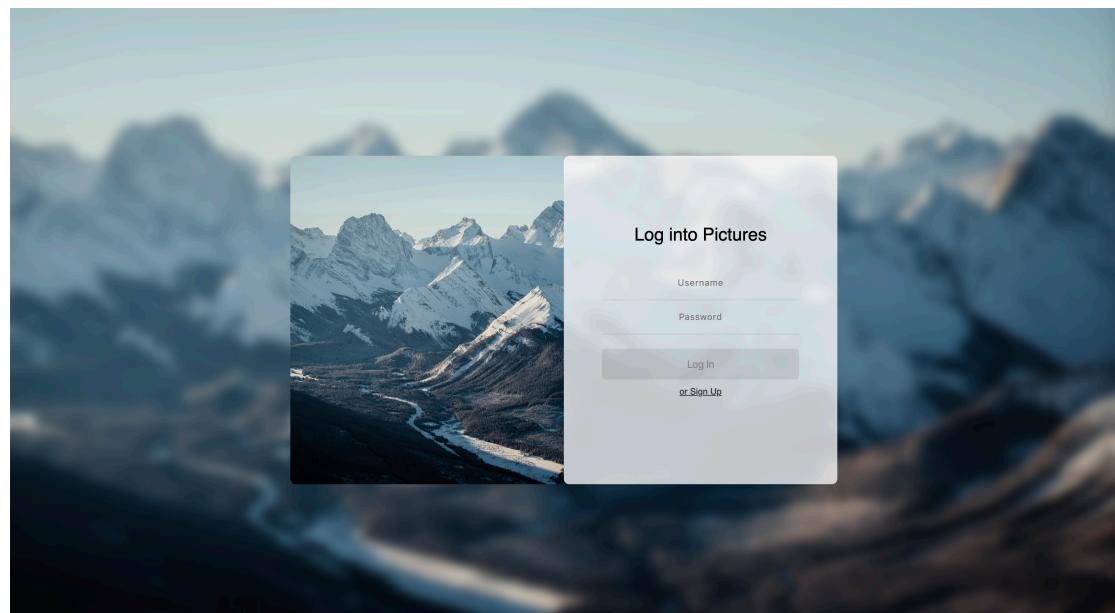
进入项目根目录，在命令行输入 npm start 即可启动后端

浏览器访问 3000 端口

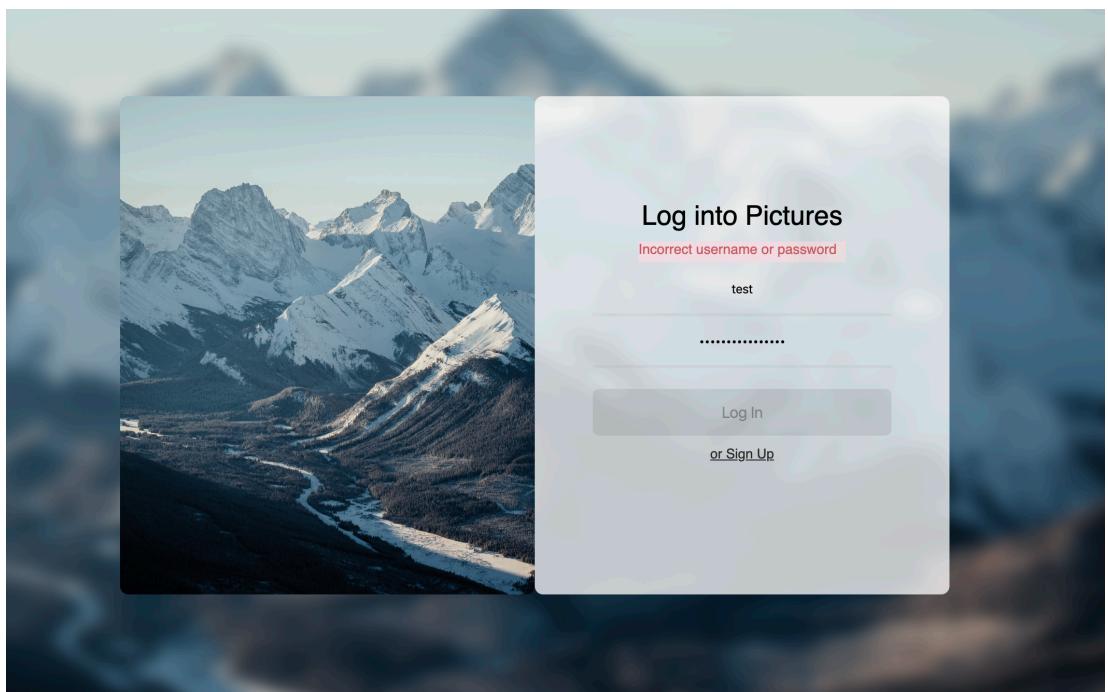
运行截图

以下为在 Chrome 浏览器中运行的截图

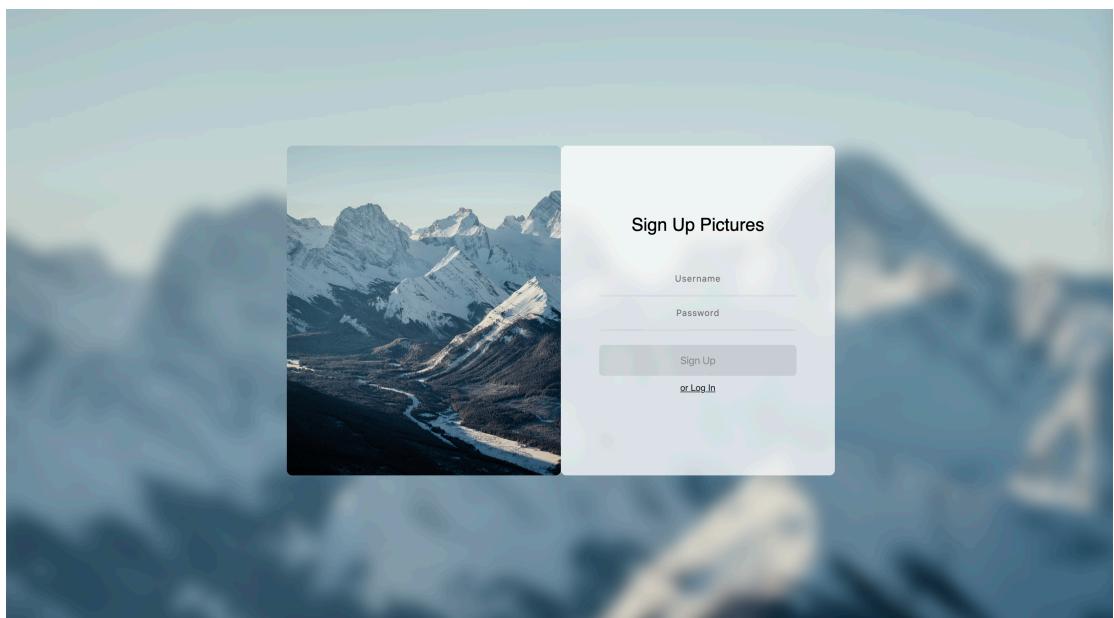
登录界面：



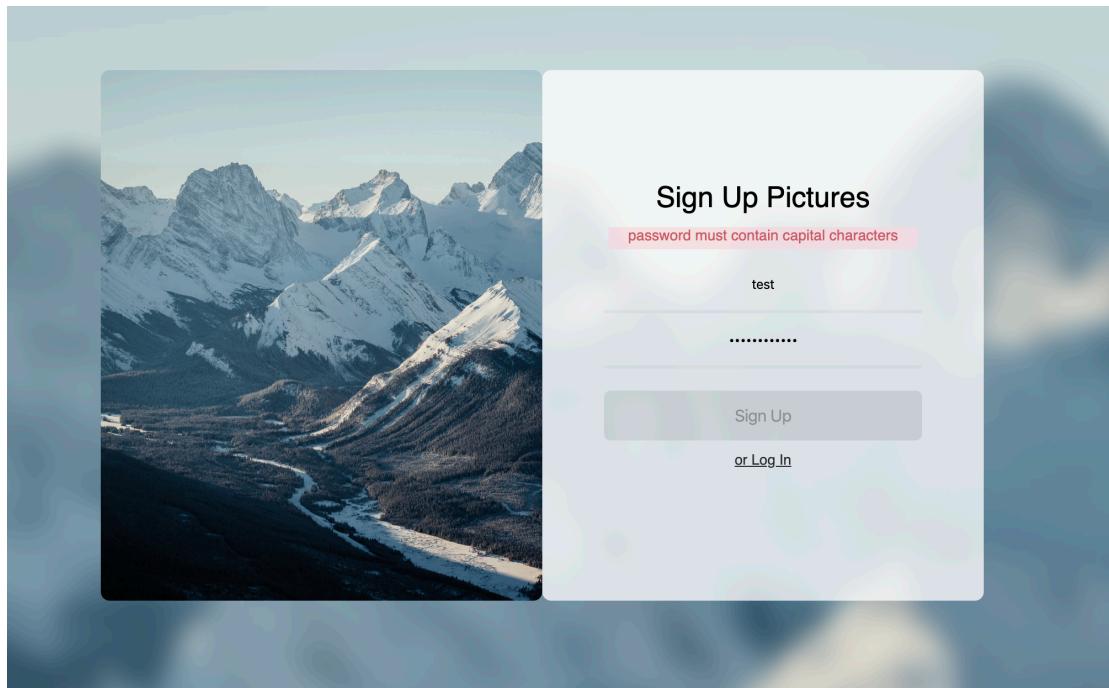
带有错误提示的登录界面：



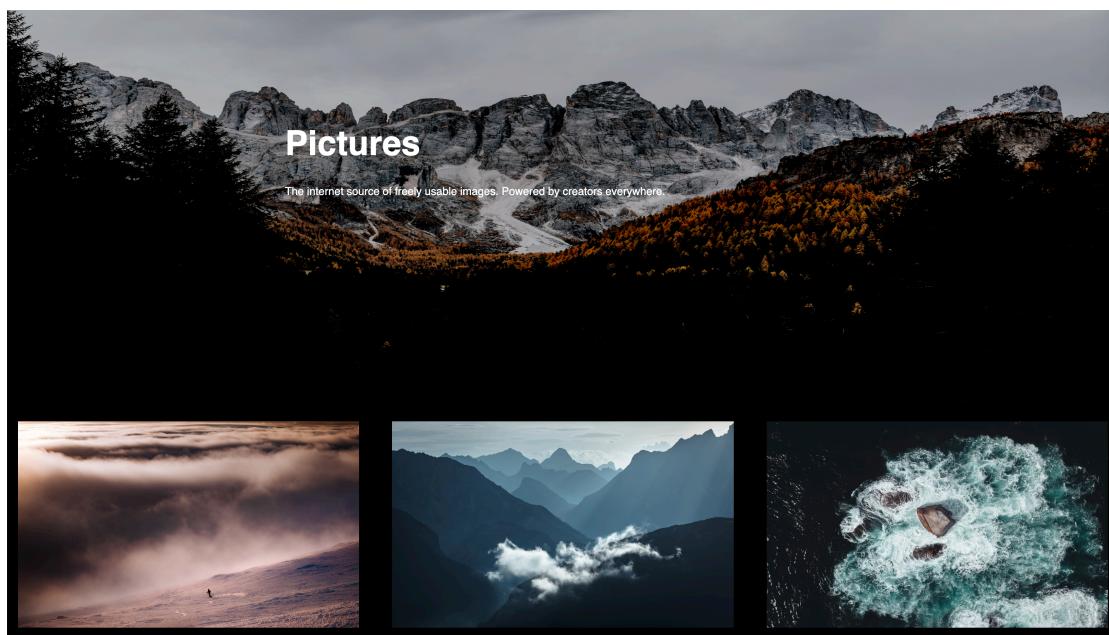
注册界面：



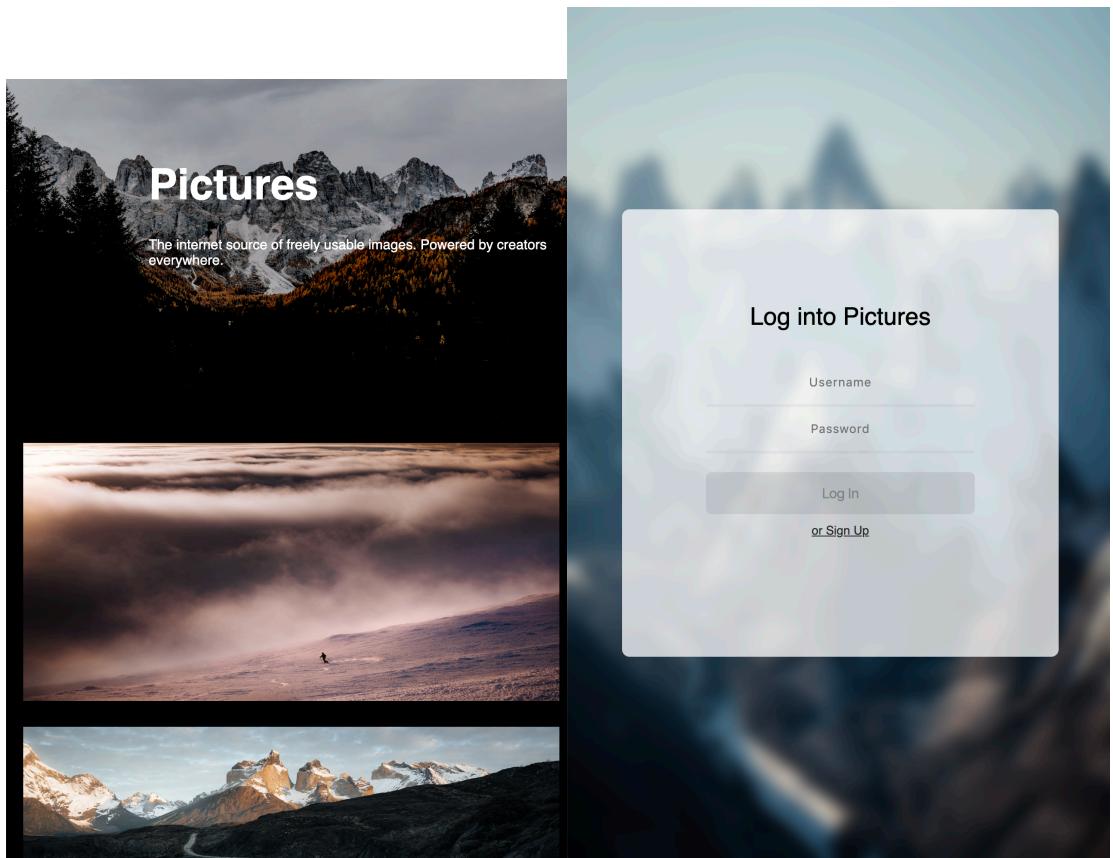
带有错误提示的注册界面：



主页：



自适应截图：



mongodb 数据库截图：

_id	username	password	salt
ObjectId("5df4ac1c72bf5c175af4f8c0")	feifei	dfK0lq5v1b0wE3hUe3nEz9HSqrUh2254mDiCtXd+phP1Ak04yPw0c0Gn5VgUZ6cGKe1j...	kr9TIPcwHXXJlTe2emw9Z11cB1ZHGM4VwS0SV1Mb16+aNLdQczMf2vXy1uOL4h4MSCqWE...
ObjectId("5df73f08da7de129cf168a0c")		e22Gvn50tVe7r6duZfdNM80ZCDZnq+fACdrt2AfPoPmPhYzz6ABDnRle0sSLBuPCNs...	LCRkyua2wm6368w0TKamapqQE013jKTce0RamziQLnANksDuuIT4lqn7yxnb4Jwx5d...
ObjectId("5df73f3dd07de129cf168a0d")	adsfasd	e22VHGN9gt9E0yB0/rrIN2q6IHNOKS0j1NqAEKZh3B8VaZN+uim93uAGaZY3vyX47el60...	1sVedFeIDNjYrFvJrkebWbOKS3+q585eshVeE8L+SE0ZM4wE7nH0rJYzcl9h1pBgnmpf...
ObjectId("5df73f4eda7de129cf168a0e")	adsfasdf	8LZKG1othsX1LOGTs4CBRE+wzzwl91fa/1x001iuF3BLiwSuNnNFF9R9N5495L9xHOY...	t4Q67a2CZ1d00eG0JD3tjjPf+dfkw20FtzGmwcU1/g3ZKGlabGeuWs9Byh9HoTa1IABM...
ObjectId("5df73f6fda7de129cf168a0f")	test	1F90xiBF0siMc036j1JNYYakbSH2eNARXvrxEd1AK6Bj02ho/Ce+i3ZgxI+4psGjzXFED...	5KhuoisXXxZ+V1sjdn3fibsnbyN7pjJeIejAsH9NpqY11UEkDSW71E0MviYNoWB5uprn...

主要功能点

前后台数据传输

后台使用的 express 框架封装了 http 请求的解析和响应的发送，比较简单，这里主要介绍一下前端向后端发送消息。

为了不引入过多额外的类库，使用了 js 原生的 fetch API 来实现前端向后端传输数据。

```
var url = 'http://localhost:3000/signup'
fetch(url, {
  method: 'POST',
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    username: username,
    password: encrypted,
  })
}).then((res) => {
  return res.json();
}).then((json)=>{
```

使用 fetch API 的难点在于它默认返回 Promise 对象，通过回调函数的形式对相应进行处理，在第一个 then 中的回调函数中调用了 res.json() 方法，该方法返回的仍是 Promise 对象，因此还可以继续被后续的 then 捕捉到，这样层层调用下。优点就是每次处理都是异步的，对于性能比较友好。

加密传输

为了保证密码传输过程中的安全性，采用加密传输。考察了现有的几种加密实现方法，使用 https 和非对称加密比较多，这里选择了前后台非对称加密来实现加密传输。

前台使用 jsencrypt 加密库，为了加速页面加载，这里先从网上将 jsencrypt.js 下载到文件系统中，用户访问的时候直接连带 html 页面返回给前台，而不是使用标签引入形式临时从网上下载。

在注册的时候，先对密码进行加密：

```
var encrypt = new JSEncrypt();
encrypt.setPublicKey(pkey);
var encrypted = encrypt.encrypt(password);
```

加密后再进行传输，截取开发者工具中看到的请求体：

```
▼ Request Payload    view source
  ▼ {username: "test",...}
    password: "JUKEPsfZ20ae0CMpLEcsDDhC94ZsN1R2sdXxuetDBjrw3CDjJT0G/CnyipTy3Mn+YTpMv/je0k2oyutPl4hQ=="
    username: "test"
```

可以看到注册的请求体中密码就是加密过的。

后台收到请求之后先进行解密：

```
var password = descry(req.body.password);
var username = req.body.username;
```

然后判断是否有重名用户（详见异常处理），如果没有重名用户，那么就将解密之后的原密码再次进行加密，用 hash 加 salt，在数据库中保存加密后的密码，这样即使数据库泄漏，密码明文仍然不会暴露：

```
hash({ password: password }, function (err, pass, salt, hash) {
  if (err) throw err;
  // store the salt & hash in the "db"
  insertDocument(db,
    {username: username,
     password: hash,
     salt: salt},
    ()=>{});
});
```

session

使用了 session 来保存登录状态。可以用于路由守卫（下面会详细介绍）和保存登录状态。首先引入 session 中间件：

```
app.use(session({
  resave: false, // don't save session if unmodified
  saveUninitialized: false, // don't create session until something stored
  secret: 'secret'
}));
```

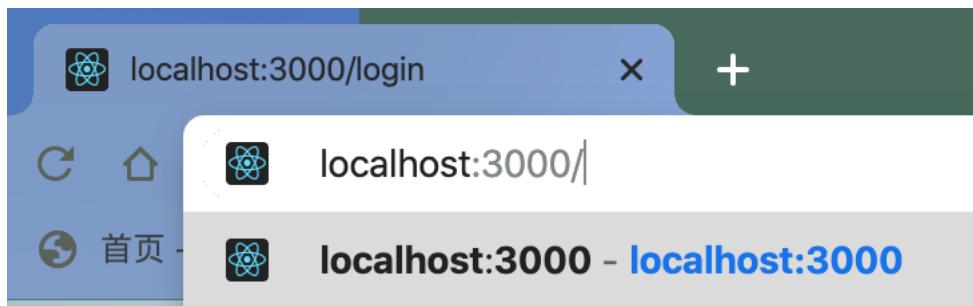
当用户登录成功之后，后台会记录用户的登录信息：

```
req.session.login = 1;
res.status(200).json({message:"登录成功", success: "true"});
```

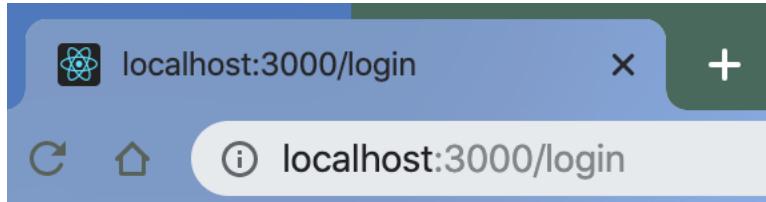
用户如果已经成功登陆过，那么再次访问网站的时候，无需重新登录，具体实现即为在访问主页的时候添加关于 session 的判断，见下面的路由守卫部分。

路由守卫

在没有登录的情况下（使用 session 来判断是否已经登录），如果直接访问主页，那么并不会访问成功，而是直接重定向到 login 页面。



自动重定向：



在代码中体现为，访问主页的时候增加判断，如果 session 中没有记录已经登陆的信息，那么就返回 login 界面而不是主页，而且还可以在用户已经登陆之后，再次访问网站无需重复登录。

```
/* GET home page. */
router.get('/', function(req, res, next) {
  if(!req.session.login){
    res.redirect("/login");
  } else
    res.sendFile(path.join(__dirname, '../views/main_page.html'));
});
```

异常处理

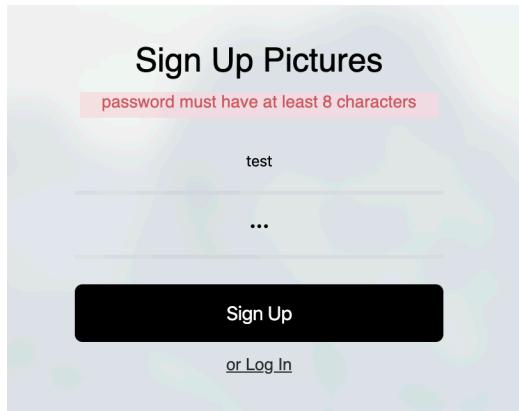
这一部分介绍一下在用户登录、注册时候出现登录或是注册失败的时候的处理。

首先是注册时候密码的检验

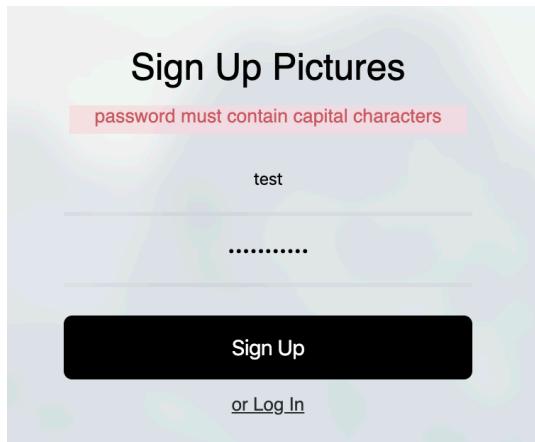
这里在前台校验密码的强度，校验的规则包括：密码长度必须大于 8 位，必须包含大写字母，必须包含小写字母。

对于以上三个规则，分别有明确的错误提示对应。

例如当长度过短的时候：



当不包含大写字母的时候：



具体的实现是写成三个独立的判断来对应三个校验：

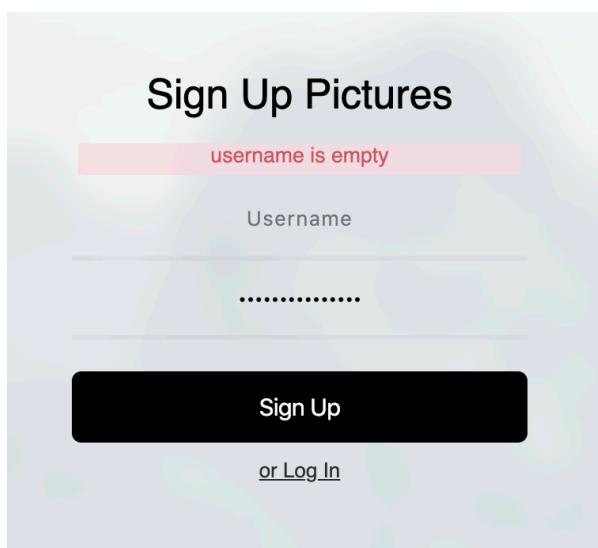
```
if(password.length<8){  
  
var reg = /[ABCDEFGHIJKLMNOPQRSTUVWXYZ]+/;  
if(!reg.test(password)){  
  
reg = /[abcdefghijklmnopqrstuvwxyz]+/;  
if(!reg.test(password)){
```

错误信息的显示实现思路是：在指定的位置，先创建出错误提示的 div，然后 css 属性设置为 hidden，当出现某种错误的时候，再将对应错误的 div 的 css 设置成 visible

```
document.getElementById("lower").style.visibility = "visible";  
document.getElementById("length").style.visibility = "hidden";  
document.getElementById("capital").style.visibility = "hidden";  
document.getElementById("empty").style.visibility = "hidden";  
document.getElementById("already").style.visibility = "hidden";
```

注册时候的用户名检验

为了防止重名用户和用户名为空的用户，需要对用户名进行检验，前端先判断用户名是否为空，如果用户名为空，那么直接错误提示：



对于重名用户，前端无法判断，只能传输到后端，检索数据库（其中 docs 为数据库中 username 为注册 username 的记录的条目）：

```
if(docs.length!=0)
    res.status(200).json({message:"用户名已存在", success: "false"});
```

如果发现重名用户，那么返回注册失败，前端接受之后，判断是否注册成功，否则错误提示：



数据库操作

因为数据库插入和查找的操作独立于应用逻辑，因此单独抽象出两个函数，并且为了和 nodejs 的风格相匹配，也添加了回调函数的接口，来实现插入和搜索数据之后进行业务逻辑的处理。这样也可以利用 nodejs 异步非阻塞执行的特性来实现性能的优化。

```
const insertDocument = function(db, user, callback) {
    console.log(user);
    // Get the documents collection
    const collection = db.collection('users');
    // Insert some documents
    collection.insertOne(user, function(err, result) {
        assert.equal(err, null);
        console.log("Inserted 1 document into the collection");
        callback(result);
    });
}

const findDocument = function(db, user, callback, res, req) {
    // Get the documents collection
    const collection = db.collection('users');
    // Find some documents
    collection.find(user).toArray(function(err, docs) {
        assert.equal(err, null);
        console.log("Found the following records");
        console.log(docs);
        callback(docs, res, req);
    });
}
```

使用的时候很简洁：

```
findDocument(db,
  {username: username},
  callback,
  res, req
)
```

体会和总结

完整完成了前后端的对接、登录注册功能，以及加密传输和存储等问题，中间也遇到了很多问题，例如和 mongodb 数据库的连接问题等。最后查找了不少资料解决了问题。也调研了一些解决方案，例如加密的方案，数据库的选择方案等。总的来说是一次收获满满的作业，更加熟悉了前后台交互流程，同时也对安全问题有了一些初步的认识。在构建项目代码的过程中也体会到了 nodejs 独特的回调函数的优雅和简明，js 原生 fetch API 中 Promise 对象对性能的优化，对前端开发有了更深入的认识和了解。