学号：201910311174

上海海事大学

“应用软件开发”课程设计

题目： 基于家庭的小型NAS系统

班 级： 网络191

姓 名： 康泽奥

指导老师： 史小宏

完成日期： 2022年11月17日

上海海事大学信息工程学院

**2022 年 11 月**

目录

[引言 1](#_Toc119581661)

[1 开发背景 2](#_Toc119581662)

[2. 需求分析 2](#_Toc119581663)

[3. 系统支撑与软件平台 2](#_Toc119581664)

[3.1 系统开发环境 2](#_Toc119581665)

[3.2 系统前后端实现框架 3](#_Toc119581666)

[4. 系统业务流程关系 3](#_Toc119581667)

[4.1 系统功能模块 3](#_Toc119581668)

[4.2 系统流程图 3](#_Toc119581669)

[4.3系统预览 6](#_Toc119581670)

[4.3.1登录界面 6](#_Toc119581671)

[4.3.2 注册界面 7](#_Toc119581672)

[4.3.3 主界面-音视频播放 7](#_Toc119581673)

[4.3.4 主界面-上传下载 8](#_Toc119581674)

[4.3.5 主界面-我的 8](#_Toc119581675)

[4.4 文件夹组织结构 9](#_Toc119581676)

[5 数据库设计 10](#_Toc119581677)

[5.1 数据库分析 10](#_Toc119581678)

[5.2 数据库概念设计 11](#_Toc119581679)

[5.3 数据库实体设计 11](#_Toc119581680)

[6 服务器模块实现与核心代码展示 13](#_Toc119581681)

[6.1 token校验公用模块 13](#_Toc119581682)

[6.2 配置cors，防止跨域问题 14](#_Toc119581683)

[6.3 连接mongodb数据库 15](#_Toc119581684)

[6.4 构造用户信息模型与用户文件信息模型 15](#_Toc119581685)

[6.5 检查用户名是否重复的接口 16](#_Toc119581686)

[6.6 注册接口 16](#_Toc119581687)

[6.7 登录接口 17](#_Toc119581688)

[6.8 请求用户数据接口 18](#_Toc119581689)

[6.9 上传文件接口 19](#_Toc119581690)

[6.10 下载接口 20](#_Toc119581691)

[6.11 预览接口 20](#_Toc119581692)

[6.12 删除文件 21](#_Toc119581693)

[6.13 返回用户头像接口 21](#_Toc119581694)

[6.14 修改用户信息接口 22](#_Toc119581695)

[7 客户端核心模块代码展示 22](#_Toc119581696)

[7.1 对axios请求进行封装 22](#_Toc119581697)

[7.2 引入自定义字体 23](#_Toc119581698)

[7.4 登录界面 25](#_Toc119581699)

[7.5 主界面 25](#_Toc119581700)

[7.5.1 主界面-上传下载 26](#_Toc119581701)

[7.5.2 主界面-音视频观看 27](#_Toc119581702)

[7.5.3 主界面-我的 28](#_Toc119581703)

[8 总结与反思 29](#_Toc119581704)

[参考文献 30](#_Toc119581705)

# 引言

随着我国互联网不断发展，各个企业之间的竞争也越发激烈，时至今日，随着国内云盘企业经过一番洗牌，常用的几家企业已经很难在市场上再看到他们的身影，如今还算广为人知的也就百度云盘这一家。不过，百度云盘虽然说起来是免费的，但实际上使用起来就会发现，如果不充值为会员的话，速度就会非常的慢。所以说百度云盘也几乎是一个收费的网盘，超级会员算下来要将近30元/月。

如果只是收费也无可厚非，毕竟大家也是收钱享受服务，然而，有这样一个事件值得引人思考，网络上有一个人说自己在百度云盘中存了大量自己拍摄的照片，后来因为一些原因手动删除了这些照片，又过了不短的一段时间，该用户再次将自己的这些照片上传至百度云盘，发现居然所有图片都是使用急速秒传的方式上传的。那么就发现了一个问题，急速秒传使用的是文件摘要值对比后选择性上传的一个技术。那问题就来了，该用户自己纯一手拍摄的照片，在没有分享给任何人的情况下删除了，而且是过了不短的一段时间再次上传，居然是秒传。那就意味着百度的服务器上存储了该用户的这些数据，但是用户已经删除了这些数据。意味着什么呢？百度根本没有删除用户的数据，在用户删除之后也只是不展示给用户看了而已。

从上面这个例子可以看出，表面上我们享有对自己上传的文件的享有权和控制权，但实际上我们的数据在上传之后就已经不再完全属于我们自己了，我们某种程度上等于是售卖了我们的数据，还是以极低的代价和极大的风险，连安全性都无法保证。

基于以上原因，再加上云端存储的带宽还无法满足大文件的快速传输，NAS系统开始逐渐在市场上抢占大量云盘的份额，他不仅满足了数据的安全性，对于大文件的传输速度也有极大的提升，除此之外，NAS系统的扩展性也十分强大，以上种种原因都促使NAS系统在较高需求的情况下要完胜云盘。而本次的小型NAS系统也将在一定程度上实现用户需求，满足文件的存储下载管理等功能。

# 1 开发背景

在上海市浦东新区的某户家庭，因为工作与生活的种种原因，有大量的文件需要存储，而现有的手机和电脑都无法满足需求，或者在存储完之后需要在另外一台设备上进行访问时极其不方便，之前他们也有试过使用云盘，但是网络传输速度还是不够理想，因此他们希望能有一个适合他们的小型NAS系统帮助他们管理家庭的数据存储。

# 2. 需求分析

NAS系统座位家庭等小单位用的网络附加存储，首先就必须要满足网络数据的存储要求，由于实在以家庭为单位的局域网内，所以网络的传输速度会比市面上的云盘传输速度更快更稳定，除此之外，为了满足局域网内用户在不同设备上访问系统资源，系统还应实现资源的在线浏览与使用，最后，对于个人资源的管理与个人信息的修改都应在一定基础上实现。

综上所述，该系统应达到以下目标：

1. 界面设计友好、美观
2. 数据存储安全、可靠
3. 信息分类清晰、准确
4. 强大的查询能力，保证数据查询的可靠性
5. 操作简单易用、界面清晰大方
6. 系统安全、稳定
7. 开发技术先进、功能完备、扩展性强
8. 占用资源少、对硬件要求低
9. 提供方便、灵活的权限设置功能，使整个系统的管理分工明确
10. 保证局域网内用户文件传输的快捷与可靠

# 3. 系统支撑与软件平台

## 3.1 系统开发环境

操作系统：Windows系统

数据库：mongodb数据库

Javascript运行时环境：nodejs

开发环境：Visual Studio Code

## 3.2 系统前后端实现框架

前端页面通过js、html、css实现，整体框架与搭建工具是vite和Vue3，考虑到js作为弱语言类型进行开发会导致工程代码可读性较差，逻辑性不够强，因此最终采用TypeScript进行脚本代码的编写。

后端框架采用的是基于Node.js的koa框架，使用TypeScipt语言，结合文档型数据库mongodb，实现服务器整体代码结构。

整体项目最终可在web端运行，最终通过electron将前端代码进行打包封装成可执行文件作为系统的客户端，实现整个应用程序的设计。

# 4. 系统业务流程关系

## 4.1 系统功能模块

该系统大体上分为五个模块，分别为“用户注册”、“用户登录”、“个人音视频资源在线浏览”、“个人文件管理”、“用户个人信息”。本系统各个部分极其包括的具体功能模块如下图4.1所示：

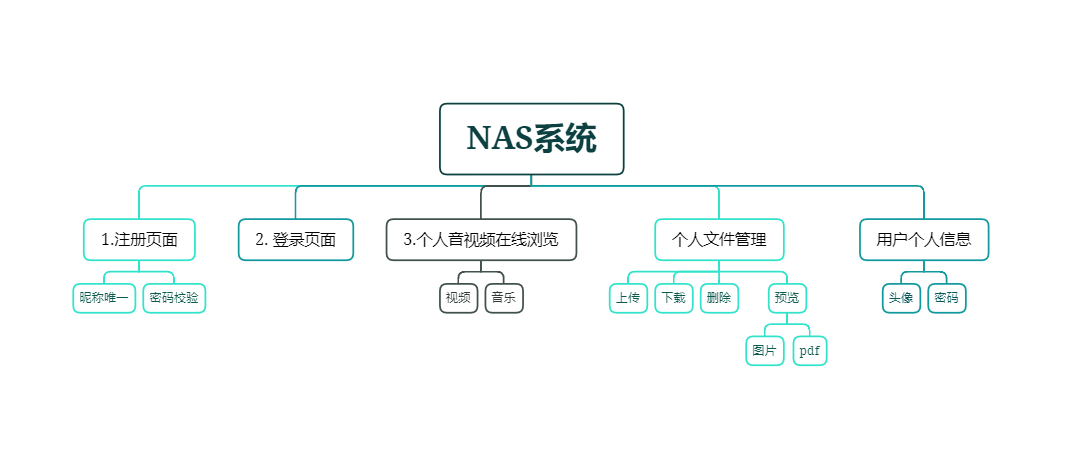


图4.1 系统功能模块图

## 4.2 系统流程图

系统的整体流程是，新用户可以在注册页面注册一个用户，注册信息有用户名和密码，两者都会进行校验，用户名必须唯一，不能与系统已有用户重名，密码必须至少有8位，且包含数字和字母。已有用户可以直接在登录页输入用户名和密码进行登录。在用户登录之后，后端会根据用户信息生成token，之后的每次请求就可以在请求头中携带token信息，方便后端对前端用户进行安全性校验。新用户由于注册时没有头像选择，所以在进入主界面后，可以转到“我的”页面，选择头像并保存。

在进入主界面后，用户可以在“上传下载”界面将自己的数据文件上传至后端文件系统，同时，后端也会同步将数据信息保存至每个用户独有的document，这样，前端查询文件信息就可以转换成数据库信息的查询，大大提高了文件资源的检索效率。同时，用户也能查看系统中自己上传的文件，在其他设备上进行下载或删除远程资源，而且，对于图片和pdf文件，系统也会提供在线浏览的功能，大大方便了用户对远程资源的管理。

在用户上传数据文件后，如果用户跳转至“音视频播放”界面，系统会自动检索用户的资源文件，将其中的音视频文件提取出来，放在展示界面正上方的select下拉选择框中，用户选择文件后，video播放器会自动添加文件路径，播放用户选择的远程音视频文件。

综上所述，详细的流程图将在下面展示：

进入软件后的用户认证与登录如图4.2所示：

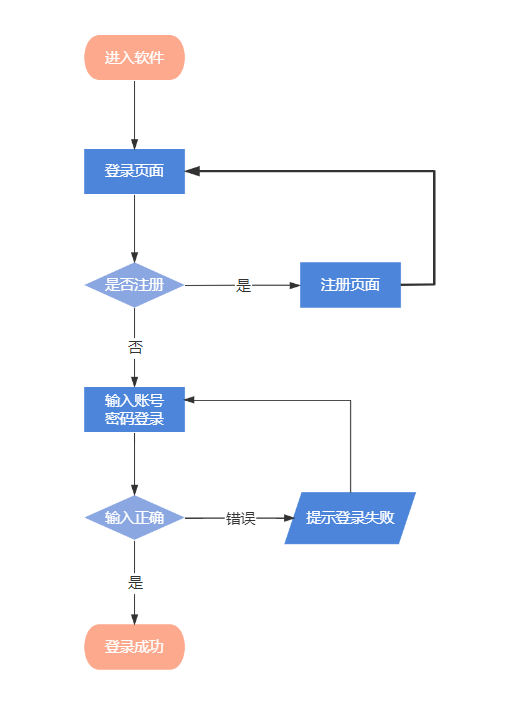


图4.2 进入软件后的用户认证

登录成功后的主界面流程如图4.3所示：

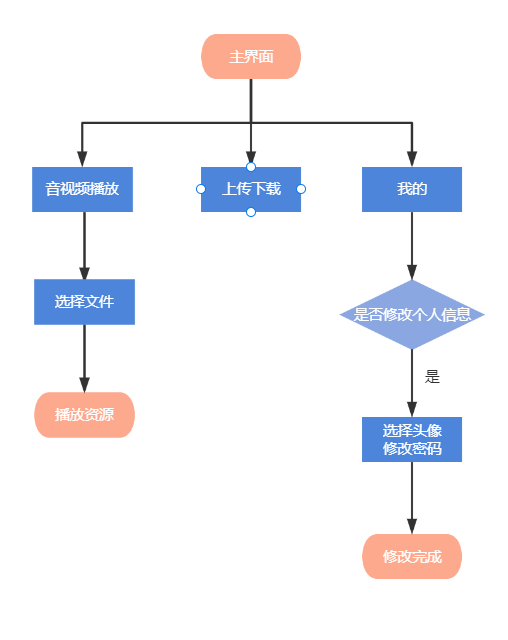


图4.3 进入主界面后的流程图

上传下载界面的流程图相对复杂，如图4.4所示：

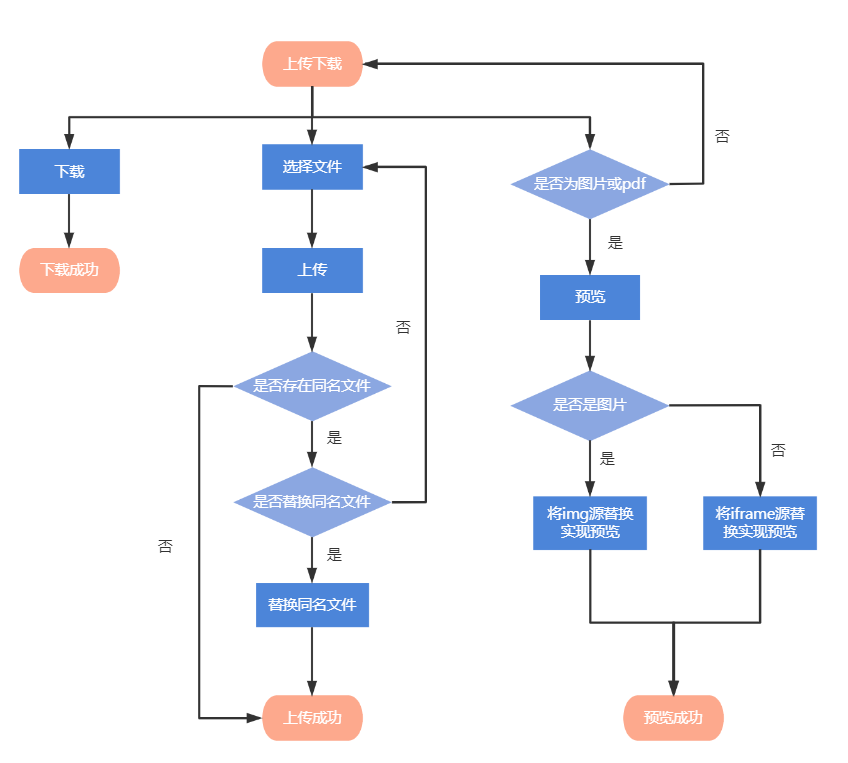


图4.4 上传下载页面的业务流程

## 4.3系统预览

NAS系统大体上由5个程序界面组成，下面将这五个程序界面展示出来：

### 4.3.1登录界面

登录界面是程序进入的第一个界面，用户输入信息后将发送请求给后端，后端查询数据库后根据结果返回是否通过，前端如果收到通过的信息就将转到程序主界面。具体样式如图4.5：



图4.5 登录界面

### 4.3.2 注册界面

如果是新用户，可以点击“注册”，进入注册界面，具体界面如图4.6：

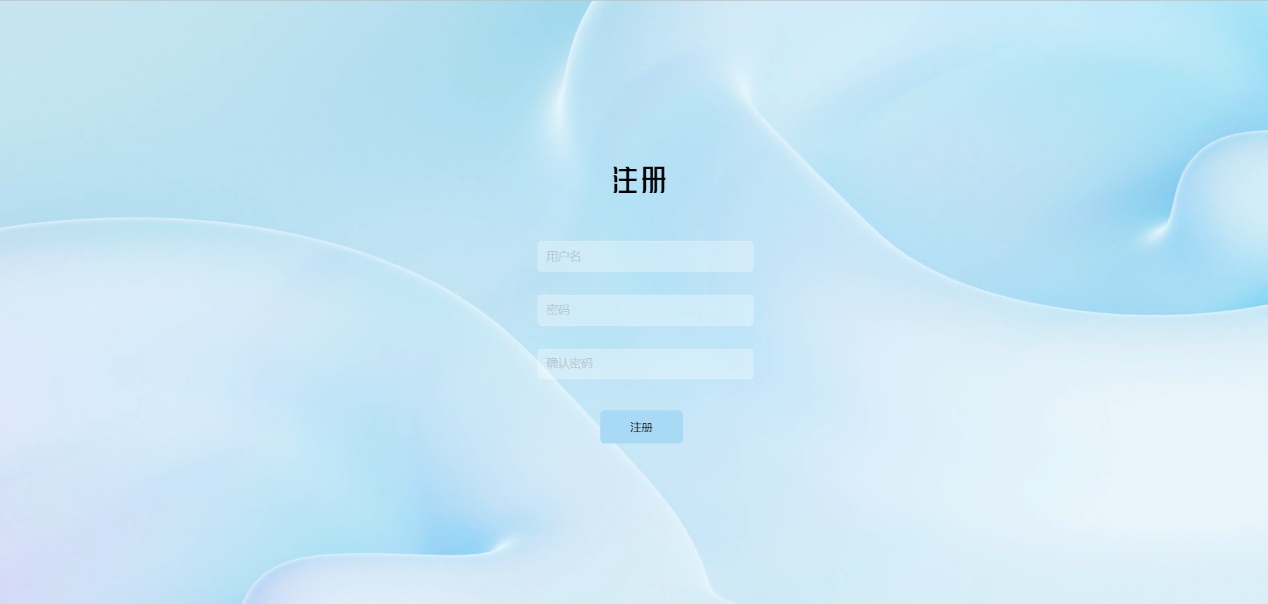


图4.6 注册界面

### 4.3.3 主界面-音视频播放

登录后首先进入主界面的音视频播放界面，在这里用户可以选择远程文件库内自己上传的视频或音乐文件，并进行在线播放。具体界面如图4.7：

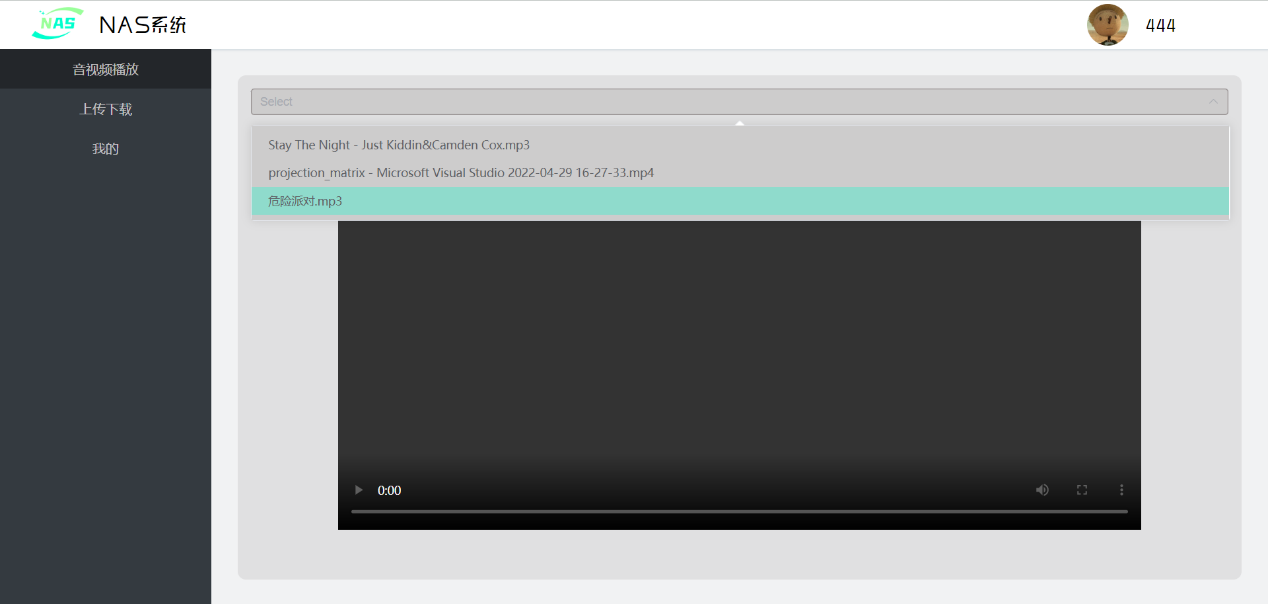


图4.7 主界面-音视频播放界面

### 4.3.4 主界面-上传下载

上传下载界面可以看到用户远程文件库里的文件，用户可以根据自己的需要进行上传、下载、删除，对于图片和pdf文件，用户也可以实现在系统界面进行预览。具体界面图如图4.8所示：



图4.8 主界面-上传下载界面

### 4.3.5 主界面-我的

在这里用户可以修改用户头像和密码，具体界面图如图4.9所示：

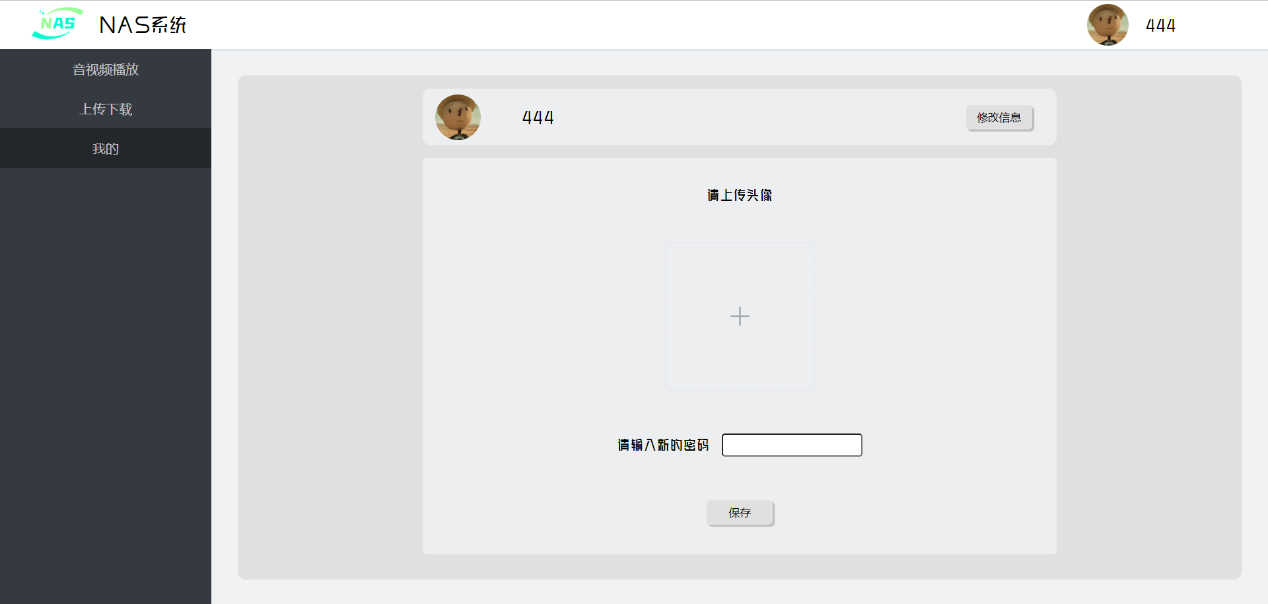


图4.9 主界面-我的页面

## 4.4 文件夹组织结构

由于整个项目是前后端分离的架构，所以文件夹整体分为前端和后端两部分。

’/server’目录存放的是后端文件，’/src’目录存放的是前端文件。

’ components.d.ts’和’ auto-imports.d.ts’是自动导入element-plus插件的脚本代码，element-plus是现在前端广泛使用的组件样式库。

’ package.json’是整个项目用到的依赖记录文件。

’ package-lock.json’是对项目依赖的版本进行记录，指定下次下载依赖时的依赖版本。

’tsconfig.json’是对项目Typescript代码做的一个配置。

’tsconfig.node.json’是对项目的不同部分做的一个Typescript配置，一般来说是对’vite.config.ts’，在本项目中，后端代码也是由’tsconfig.node.json’配置。

‘vite.config.ts’是与项目vite相搭配的项目构建配置文件。

具体的项目文件夹组织结构图如图4.10所示：

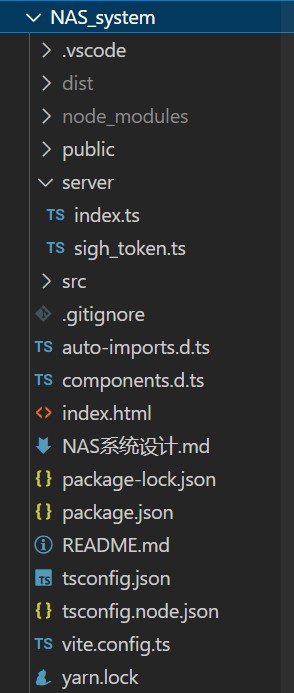


图4.10 项目文件夹组织结构

# 5 数据库设计

## 5.1 数据库分析

MongoDB是面向文档的NoSQL数据库，用于大量数据存储。MongoDB是一个在2000年代中期问世的数据库。属于NoSQL数据库的类别。

面向文档的–由于MongoDB是NoSQL类型的数据库，它不是以关系类型的格式存储数据，而是将数据存储在文档中。这使得MongoDB非常灵活，可以适应实际的业务环境和需求。

## 5.2 数据库概念设计

根据以上对系统所做的需求分析、系统设计，规划出本系统中使用的数据库实体应包含以下几个document：存储用户信息，customs；根据每个用户的唯一标识再创建其归属的document。下面将介绍这两个document的E-R图。

下图5.1是用户信息document：customs和用户文件信息document的E-R图：

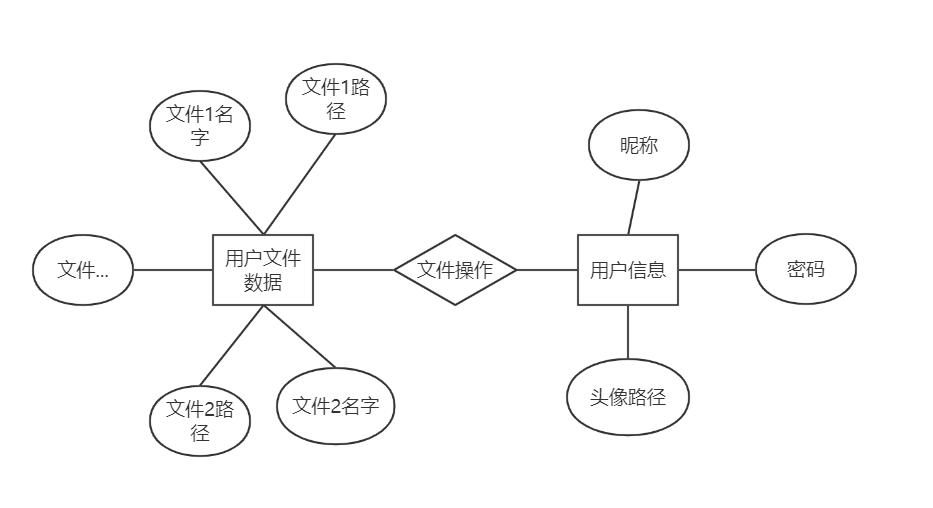


图5.1 数据库document之间的E-R图

## 5.3 数据库实体设计

首先我们需要一个存储一个所有用户个人信息的document，本系统使用的是自定义document结构，有三个主要使用的字段，分别是：name，password，filePath。

具体实际document的组成结构如下图5.2所示：

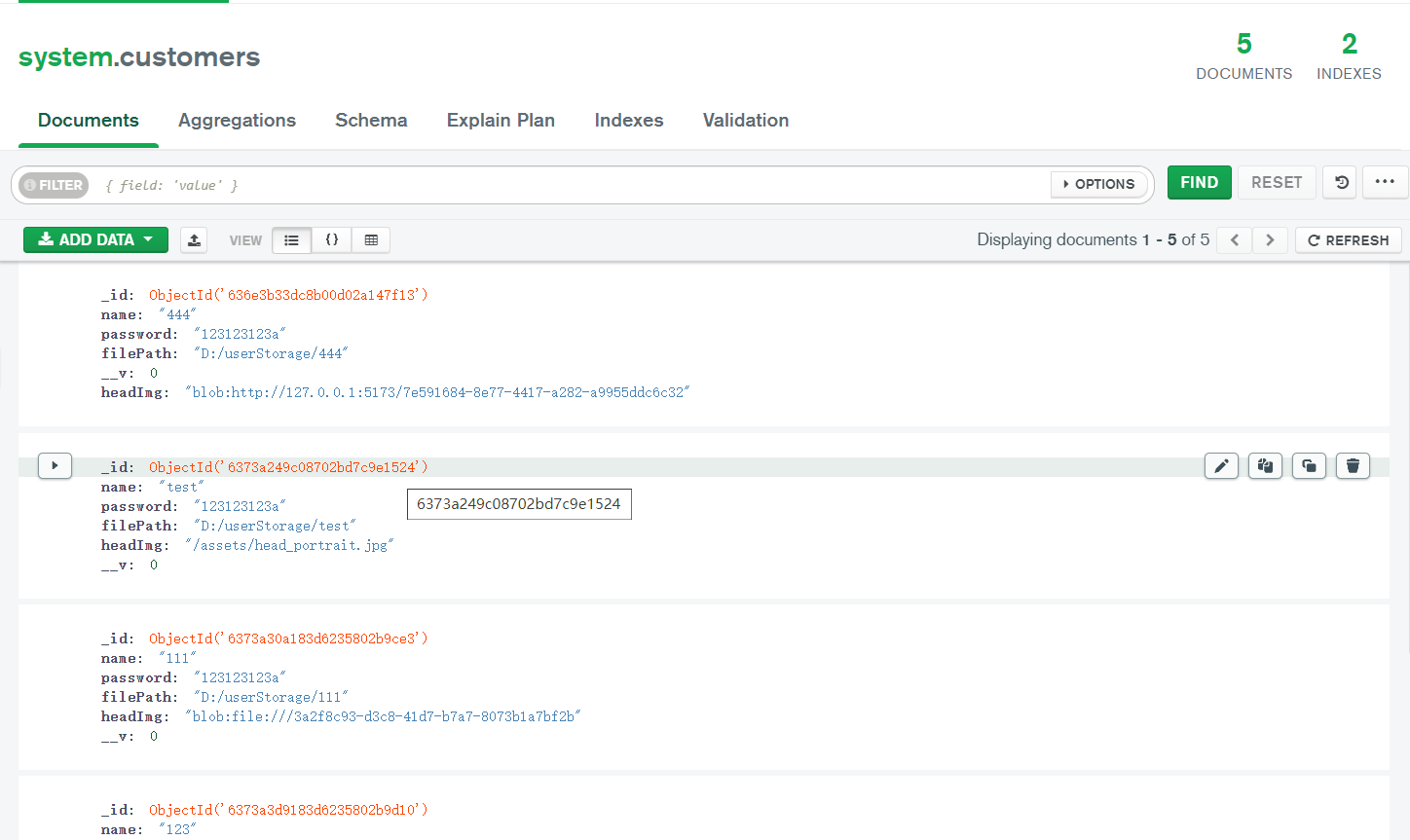


图5.2 customs表的组成

除此之外，每当一个用户注册，系统后端就会新建一个具有该用户唯一字段的documents，具有文件名和文件实际存储路径两个字段，同时也会在服务器的实际文件系统新建同样的文件夹进行存储实际文件，在本系统中，我们使用的是用户名name作为唯一字段，下图5.3是用户文件信息组成的document：

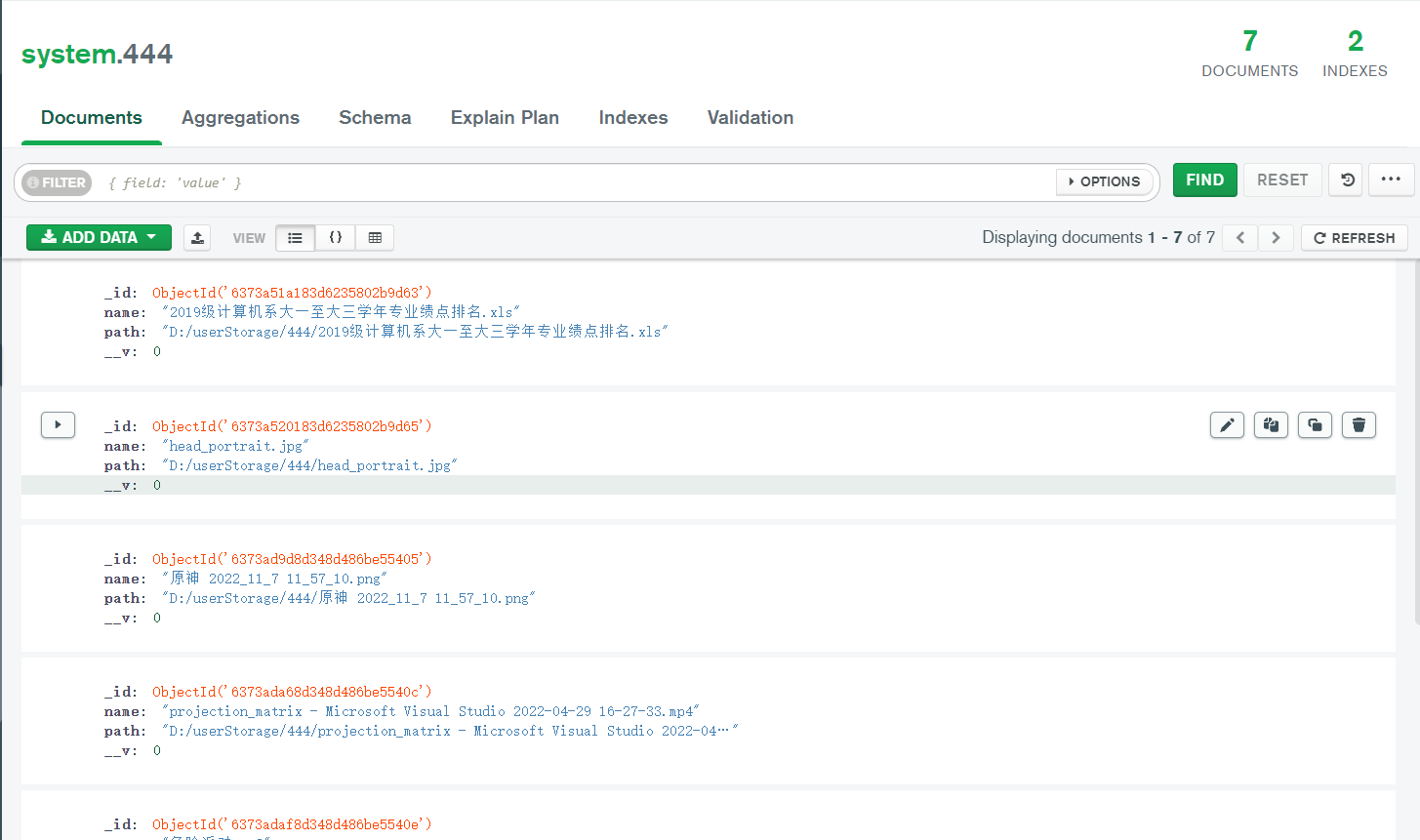


表5.3 用户文件信息表

# 6 服务器模块实现与核心代码展示

## 6.1 token校验公用模块

后台利用token校验可以脱离浏览器原生方法实现前后端通信的同源校验，本系统采用的是jsonwebtoken插件，利用插件的sign()方法实现token的生成，利用verify()方法对token进行解码。具体代码如下：

|  |
| --- |
| // 自定义密钥  const secret = "erzimen\_woshinibaba"  import jwt from 'jsonwebtoken'  /\*\*  \* @param {\*} data 加密的数据，最好不要包括密码之类的  \*/  const sign\_Token = (data:any) => {  const token = jwt.sign(  data,  secret  )  return token  }  const decode\_Token = (token:string) => {  return new Promise((resolve,reject) => {  jwt.verify(token,secret,function(err,decode){  if(err) reject(err)  else resolve(decode)  })  })  }  export {  sign\_Token,  decode\_Token,  secret  } |

除此之外，由于token校验的代码复用性较高，代码结构较复杂，后端将有关token处理的代码进行了封装，具体代码如下所示：

|  |
| --- |
| //校验token，返回对象，包含code和name两个属性  function checkToken(token: string) {  //如果token不为''  if (token.length != 0) {  //解码token  return decode\_Token(token).then(async res => {  let endTime = new Date().getTime()  const { name, startTime, timeout } = res as payload  // console.log(startTime, endTime,timeout)  //检查name是否为系统用户  let nameLegal: token\_return  //查询数据库，为nameLegal赋值,方便后面判断  return new Promise((resolve, reject) => {  user\_model.find({ name: name }).exec((err, res) => {  if (err) nameLegal = { legal: false, name: '' }  else nameLegal = { legal: true, name: res[0].name }  resolve(1)  })  }).then(res => {  //如果时间过期  if (endTime - startTime > timeout) {  return { code: 402, name: '', message: 'token过期' }  }  //如果名字不合法  else if (!nameLegal.legal) return { code: 402, name: '', message: '名字不存在' }  //既没过期名字也合法  else return { code: 200, name: nameLegal.name }  })  }, err => {  return { code: 402, name: '', message: 'token解码错误' }  })  }  //token长度为0  else {  return Promise.resolve({ code: 401, name: '', message: 'token长度为0' })  }  } |

## 6.2 配置cors，防止跨域问题

浏览器自带跨域问题的校验，凡是协议，域名，端口号有一项不同就会触发同源策略被浏览器拦截，导致前后端无法通信，因此，我们可以在后端设置cors，防止跨域问题的出现。下面是具体代码的实现：

|  |
| --- |
| app.use(cors({  //生产环境  // origin:'http://127.0.0.1:5500',  //开发环境  origin: 'http://127.0.0.1:5173',  maxAge: 5,  credentials: true,  allowMethods: ['GET', 'POST', 'DELETE'],  // allowHeaders: ['Content-Type', 'Authorization', 'Accept','X-token'],  })) |

## 6.3 连接mongodb数据库

后端在开启服务的同时就应与数据库保持连接，以方便后续对数据库的操作，本系统是通过mongoose插件实现对数据库的连接与操作。下面是具体代码的实现：

|  |
| --- |
| //连接数据库  import mongoose from 'mongoose'  mongoose.connect('mongodb://sys\_admin02:123@localhost:27017/system');  const db = mongoose.connection;  db.on('error', console.error.bind(console, '连接失败'));  db.once('open', function () {  console.log('数据库连接成功');  }); |

## 6.4 构造用户信息模型与用户文件信息模型

Mongoose对数据库的操作都是基于model模型往库或表中进行操作，因此，我们也应提前规定好用户信息模型与用户文件信息模型。具体代码如下所示：

|  |
| --- |
| //用户信息原型  const login\_schema = new mongoose.Schema({  name: { type: String, default: 'user\_default', unique: true },  password: { type: String },  filePath: { type: String },  headImg: {type:String, default:'/assets/head\_portrait.jpg'}  })  const file\_schema = new mongoose.Schema({  name: { type: String, unique: true },  path: { type: String }  })  //构造用户信息模型  const user\_model = db.model("customers", login\_schema)  /在数据库创建该用户的文件信息集合  const file\_db = db.model(add\_name, file\_schema) |

## 6.5 检查用户名是否重复的接口

在用户注册时需要检测用户名是否已被注册，保证用户名的唯一性，具体代码如下所示：

|  |
| --- |
| //检查名字是否重复  router.post('/nameCheck', async (ctx: koa.Context) => {  console.log(ctx.request.body)  let add\_name: string = ctx.request.body.name;  //个人理解：返回一个Promise，并在回调执行异步任务阻止ctx返回  return new Promise((resolve) => {  //强制转为宏任务  setTimeout(() => user\_model.find({ name: add\_name }).exec((err, res) => {  if (err) ctx.body = 'err'  else if (res.length === 0) ctx.body = 'success'  else ctx.body = 'existed'  resolve(1)  }), 10);  })  }) |

## 6.6 注册接口

当用户输入注册信息后，点击注册就会发送注册请求，后端根据这个接口处理请求，如下是具体代码的展示：

|  |
| --- |
| //将用户信息注册到数据库,开辟文件管理数据库，开辟存储文件夹  router.post('/register', async (ctx: koa.Context) => {  let add\_name = ctx.request.body.name  let add\_passwd = ctx.request.body.password  let newUser = new user\_model({ name: add\_name, password: add\_passwd, filePath: `D:/userStorage/${add\_name}` })  //添加该用户的存储文件夹  fs.mkdir(`D:/userStorage/${add\_name}`, err => {  if (err) console.log("用户资源文件创建失败 ", err)  console.log("用户资源文件创建成功")  })  //在数据库创建该用户的文件信息集合  const file\_db = db.model(add\_name, file\_schema)  return new Promise((resolve) => {  setTimeout(() => {  newUser.save(function (err) {  if (err) return handleError(err)  else ctx.body = '注册成功'  resolve(1)  })  }, 100);  })  }) |

## 6.7 登录接口

当用户输入登录信息后发送请求，后端接口处理代码如下所示：

|  |
| --- |
| //登录  router.post('/login', async (ctx: koa.Context) => {  // console.log(ctx.request)  let login\_name = ctx.request.body.name  let login\_passwd = ctx.request.body.password  return new Promise((resolve) => {  setTimeout(() => user\_model.find({ name: login\_name, password: login\_passwd }).exec((err, res) => {  if (err || res.length === 0) ctx.body = 'err'  else {  let now = new Date().getTime()  const token = sign\_Token({ name: login\_name, startTime: now, timeout: 1000 \* 60 \* 60 \* 24 })  ctx.body = {  message: '登录成功',  code: 200,  token  }  }  resolve(1)  }), 10);  })  }) |

## 6.8 请求用户数据接口

当用户登录后，在页面挂载前请求用户数据，有两部分，通用的是请求用户个人信息接口，代码如下所示：

|  |
| --- |
| //请求用户数据  router.get('/userdata', async (ctx: koa.Context) => {  let token: string = ctx.request.headers['x-token'] as string  // console.log('token = ', token)  return new Promise((resolve, reject) => {  setTimeout(async () => {  await checkToken(token).then(res => {  //token不合法  if (res!.code === 402 || res!.code === 401) {  ctx.body = { code: res!.code, message: res.message }  resolve(1)  }  //token校验成功  else {  return setTimeout(async () => {  user\_model.find({ name: res.name }).select('name filePath headImg').exec((err, res) => {  ctx.body = {  code: 200,  userinfo: res[0]  }  resolve(1)  })  }, 10)  }  })  }, 10);  })  }) |

除此之外，用户在某些模块还需要请求文件列表，代码如下所示：

|  |
| --- |
| //返回指定用户的文件列表  router.get('/userdata/filelist', async (ctx: koa.Context) => {  const name = ctx.request.query.name as string;  const file\_model = db.model(name, file\_schema)  return new Promise((resolve, reject) => {  setTimeout(() => {  file\_model.find().sort({ name: 1 }).exec((err, res) => {  if (err) return handleError(err)  ctx.body = {  code: 200,  fileList: res  }  resolve(1)  })  }, 100)  })  }) |

## 6.9 上传文件接口

当用户上传文件时，后台对数据库与服务器文件系统进行操作，并将结果返回给前端，具体代码如下所示：

|  |
| --- |
| //上传文件  router.post('/upload', async (ctx: koa.Context) => {  const name = ctx.request.body.name  const files = ctx.request.files; // 获取上传文件  const file\_model = db.model(name, file\_schema) //用户个人文件管理系统集合  return new Promise((resolve) => {  setTimeout(() => {  Object.keys(files!).forEach((item) => {  const reader = fs.createReadStream((files![item] as BodyFile).filepath); // 创建可读流  // const ext = (files![item] as BodyFile).originalFilename.split('.').pop(); // 获取上传文件扩展名  const originalFilename = (files![item] as BodyFile).originalFilename  const upStream = fs.createWriteStream(`D:/userStorage/${name}/${originalFilename}`); // 创建可写流  reader.pipe(upStream); // 可读流通过管道写入可写流  const upload\_file = new file\_model({ name: originalFilename, path: `D:/userStorage/${name}/${originalFilename}` })  upload\_file.save(function (err) {  if (err && !err.message.includes('E11000')) return handleError(err)  else ctx.body = '上传成功'  resolve(1)  })  })  }, 100);  })  }) |

## 6.10 下载接口

前端选择下载文件后，后端根据文件名和用户名进行文件的选择和传输，具体代码如下所示：

|  |
| --- |
| import send from 'koa-send'  router.post('/download', async (ctx: koa.Context) => {  const name = ctx.request.body.name  const filename = ctx.request.body.filename  const path = `/${name}/${filename}`  ctx.attachment(path)  return await send(ctx, path, { root: 'D:/userStorage' })  // try{  // await send(ctx, path, { root: 'D:/userStorage' })  // }catch(err){ctx.throw(404,'文件不存在')}  }) |

## 6.11 预览接口

预览接口与下载接口大体一致，只是后端进行设置，只传url，前端不会自动进行下载，具体代码如下：

|  |
| --- |
| //预览与音视频在线播放  router.post('/preview', async (ctx: koa.Context) => {  const name = ctx.request.body.name  const filename = ctx.request.body.filename  const path = `/${name}/${filename}`  return send(ctx, path, { root: 'D:/userStorage' })  }) |

## 6.12 删除文件

前端传删除文件的文件名和自身用户名，后端进行处理，代码如下所示：

|  |
| --- |
| //删除文件  router.get('/removefile', async (ctx: koa.Context) => {  const name = ctx.request.query.name as string  const filename = ctx.request.query.filename  const file\_model = db.model(name, file\_schema)  return new Promise((resolve, reject) => {  setTimeout(() => {  fs.unlinkSync(`D:/userStorage/${name}/${filename}`)  //删除数据库数据  file\_model.remove({ name: filename }).exec((err, res) => {  if (err) return handleError(err)  ctx.body = '文件删除成功'  resolve(1)  })  }, 100)  })  }) |

## 6.13 返回用户头像接口

当用户上传完头像后，后端将其保存并返回给前端，具体代码如下：

|  |
| --- |
| router.post('/head\_img', async (ctx: koa.Context) => {  const files = ctx.request.files; // 获取上传文件  const reader = fs.createReadStream((files!.file as BodyFile).filepath); // 创建可读流  const ext = (files!.file as BodyFile).originalFilename.split('.').pop();  const temp = `${new Date().getTime()}.${ext}`  const upStream = fs.createWriteStream(`D:/userStorage/cache/${temp}`); // 创建可写流  reader.pipe(upStream); // 可读流通过管道写入可写流  return new Promise((resolve) => {  setTimeout(async () => {  await send(ctx,temp,{root:'D:/userStorage/cache/'})  resolve(1)  }, 100)  })  }) |

## 6.14 修改用户信息接口

用户将头像与密码传给后端，后端进行修改，具体代码如下所示：

|  |
| --- |
| router.post('/changeinfo',async (ctx:koa.Context) => {  const body = ctx.request.body  // console.log('body: ',body)  const {headimg, name, password} = body  return new Promise((resolve) => {  setTimeout(async() => {  user\_model.where({name:name}).update({headImg:headimg, password:password}).exec((err) => {  if(err) console.log(err)  ctx.body = 'success'  resolve(1)  })  }, 100)  })  }) |

# 7 客户端核心模块代码展示

## 7.1 对axios请求进行封装

由于前后端通信主要通过http请求，而axios正是对http请求的封装，再次基础上我们再进行一层封装，最后再将其暴露出去。由于代码量过多，这里只战士核心代码：

|  |
| --- |
| import axios from 'axios'  const request = axios.create({  baseURL: 'http://127.0.0.1:3000',  timeout: 5000 ,  withCredentials: true,  })  //请求拦截器  request.interceptors.request.use(config => {  return config  },(error) => {  return Promise.reject(error)  })  //响应拦截器  request.interceptors.response.use((response => {  return response  }),(error) => {  return Promise.reject(error)  })  export default request |

请求的最后形式如下所示：

|  |
| --- |
| import request from "./request";  import axios from "axios";  export function register(name:string, passwd:string){  return request({  url:'/register',  method:'post',  data:{'name':name, 'password':passwd}  })  } |

## 7.2 引入自定义字体

为了增加系统的特异性，本系统客户端部分字体采用的是网上下载的字体，能给人耳目一新的感觉，具体引用方式如下所示：

|  |
| --- |
| font.css  @font-face {  font-family: 'fontType';  src: url('./FengGuang.ttf');  font-weight: normal;  font-style: normal;  }  App.vue  @import './assets/font/font.css'; |

#### 7.3 注册界面

对用户名输入框失焦绑定事件nameCheck，具体代码如下所示：

|  |
| --- |
| //输入框失焦后进行用户名验证与非空判断  const nameCheck = ref()  nameCheck.value = async () => {  if (!name.value) {  nameEmpty.value = true  nameExist.value = false  }  else {  nameEmpty.value = false  name\_check(name.value).then((res) => {  if (res.data == 'err') console.log('服务器出错')  else if (res.data == 'existed') nameExist.value = true  else if (res.data == 'success') nameExist.value = false  })  }  } |

在密码输入框失焦后，也要判断密码是否满足至少8个字符，且包含字母和数字，具体代码如下：

|  |
| --- |
| //输入框失焦后进行密码验证  const passwordSpec = ref()  passwordSpec.value = () => {  //至少8个字符，至少一个字符和一个数字  let p = /^(?=.\*[A-Za-z])(?=.\*\d)[A-Za-z\d]{8,}$/  if (!p.test(password.value)) illegal.value = true;  else illegal.value = false  } |

再次确认密码，在用户按下键之后，检测再次输入的密码是否和第一次输入的相同，为了减少性能损耗，还可以对事件作防抖处理，具体代码如下：

|  |
| --- |
| //检验第二次输入密码是否与第一次相同  let time: NodeJS.Timeout|null = null  const passwdAgain\_check = ref()  passwdAgain\_check.value = () => {  if (time) clearTimeout(time)  time = setTimeout(() => {  // console.log(1)  if (passwordAgain.value !== password.value) passwdIsSame.value = false  else passwdIsSame.value = true  }, 500)  } |

最后在收到后端发来的注册成功消息后，将路由跳转到登录界面，具体代码如下：

|  |
| --- |
| //注册用户信息  const registerUser = ref()  registerUser.value = () => {  register(name.value, password.value).then((response) => {  if(response.data === '注册成功')  router.push({  name:'login',  })  })  } |

## 7.4 登录界面

登录界面就是在用户点击登录后发送请求并根据返回消息判断是否跳转主界面，具体代码如下：

|  |
| --- |
| //登录  const Login = ref()  Login.value = () => {  login(name.value,passwd.value).then((res) => {  if(res.data.message === '登录成功'){  let token = res.data.token;  window.localStorage.setItem('X-token',token)  router.push({  name:'menu',  })  }  else window.alert('登录失败')  },err => {  console.log(err)  })  } |

## 7.5 主界面

在主界面挂载前，请求用户信息，具体代码如下：

|  |
| --- |
| onMounted(async () => {  //请求数据  // console.log('修改后的token',window.localStorage.getItem('X-token'))  await getdata().then(res => {  if (res.data.code === 200) {  // console.log(res.data)  userinfo.name = res.data.userinfo.name  userinfo.headImg = res.data.userinfo.headImg  } else {  router.push({ name: 'login' })  }  })  }) |

同时，为了在用户修改信息后，第一时间修改界面上的个人信息，本系统采用监听子组件emit的时间，进行同时更改，具体代码如下：

|  |
| --- |
| watch(update, async (newvalue, oldvalue) => {  if (newvalue) {  await getdata().then(res => {  if (res.data.code === 200) {  // console.log(res.data)  userinfo.name = res.data.userinfo.name  userinfo.headImg = res.data.userinfo.headImg  } else {  router.push({ name: 'login' })  }  })  update.value = false  }  },{immediate:true}) |

### 7.5.1 主界面-上传下载

为了方便用户查找文件，系统在收到后端传来的文件列表后，对文件名进行排序，最后展示在界面上，具体代码如下：

|  |
| --- |
| //按首字母排序插入文件  const pushFile = (files: Ref<{ name: string, path: string, }[]>, obj: fileInfo) => {  if (files.value.length === 0) files.value.push(obj)  for (let t = 0; t < files.value.length; t++) {  if (files.value[t].name.charAt(0) > obj.name.charAt(0)) {  files.value.splice(t, 0, obj)  return  }  }  } |

其他的逻辑实现大体上都是通过点击按钮后获取DOM元素得到信息，然后进行相关接口的请求，再根据接口返回信息进行反馈，值得注意的是，系统为了增强用户反馈，在异步请求阶段会增加一个遮罩层向用户表示请求正在加载中，这里拿一个例子来展示，具体代码如下：

|  |
| --- |
| //删除文件  const remove = ref()  remove.value = (e: PointerEvent) => {  if (e) {  const fileName = ((e.target as HTMLSpanElement).parentNode as HTMLLIElement).outerText.split('\n').shift()  const loadingInstance = ElLoading.service({ target: '.file-show-list' })  removeFile(props.name, fileName as string).then(res => {  loadingInstance.close()  if (res.data === '文件删除成功') {  let id  for (id = 0; id < files.value.length; id++) {  if (files.value[id].name === fileName) break  }  files.value.splice(id, 1)  ElMessage({  message: '文件删除成功',  type: 'success',  })  }  else {  ElMessage.error('文件删除失败')  }  })  }  } |

### 7.5.2 主界面-音视频观看

在”音视频观看”界面挂载前同样需要请求用户文件列表，并进行筛选，将音视频文件放到选择列表中，这样用户就只能选择音视频文件进行观看，同时，在用户选择之后，利用监听函数监听选择框的值，在值不为空时，将值作为video的src，具体代码如下：

|  |
| --- |
| /筛选出多媒体文件  const media = () => {  const suffix = ['mp4', 'avi', 'mp3', 'wav', 'mov']  files.value.forEach((file) => {  const t = file.name.split('.').pop()  if (t && suffix.includes(t)) media\_file.value.push(file.name)  })  }  watch(value, (value, prevalue) => {  if (value !== prevalue && value !== '') {  previewFile(props.name, value).then(res => {  if (res.status === 200) {  let href = window.URL.createObjectURL(res.data);  let video = document.getElementById('video\_0') as HTMLVideoElement  video.src = href  // window.URL.revokeObjectURL(href)  }  else {  ElMessage.error('请求下载失败')  }  })  }  }) |

### 7.5.3 主界面-我的

“我的”界面在用户进行上传头像时发送请求并接收后端返回blob类型图片文件流，然后将其转为url做为图片src，使图片可视化，具体代码如下：

|  |
| --- |
| const uploadSectionFile = ref()  uploadSectionFile.value = (params: any) => {  const file = params.file,  fileType = file.type,  isImage = fileType.indexOf("image") != -1,  isLt2M = file.size / 1024 / 1024 < 2;  // 这里常规检验，看项目需求而定  if (!isImage) {  ElMessage.error("只能上传图片格式png、jpg、gif!");  return;  }  if (!isLt2M) {  ElMessage.error("只能上传图片大小小于2M");  return;  }  // 根据后台需求数据格式  const form = new FormData();  // 文件对象  form.append("file", file);  upload\_headpic(form).then(res => {  if (res.status === 200) imageUrl.value = URL.createObjectURL(res.data)  })  } |

# 8 总结与反思

通过这次课程设计，我总体上已经熟悉整体项目的搭建流程与大部分项目是实现的过程中可能遇到的问题，同时，它也培养了我实现自己想法应该如何去做的能力，但是，最后来看，整个项目还有很大的提升空间，比如，这里的头像上传可以将其存到用户文件列表中，而不是存blob数据，这样在用户第一次登录时头像src是不可访问的，而如果存到文件列表的话，每次传的就是实际的文件流，这样头像就不会有不可见的情况。总体来说，由于前期已经将整体结构搭建好了，导致后面再碰到结构上的问题时再想改动就会很麻烦，因此，在项目搭建前，我们还是应该将项目结构尽可能细化并完善，这样就能大大加快后面程序设计过程的速度与提高准确度。

# 参考文献

1. suoh's Blog. vue+electron将项目打包为exe程序---成功跨过所有坑，完美打包. <https://blog.csdn.net/qq_41579104/article/details/107340431>. 2020-07-15
2. Element Plus. <https://github.com/element-plus/element-plus>. 2022-11-12
3. Vue3. <https://github.com/vuejs/>. 2022-11-11
4. Koa. <https://github.com/koajs/koa>. 2022-10-25
5. Mongoose. <https://github.com/Automattic/mongoose>. 2022-11-14