Web 应用开发技术课程设计文档

1. 目标与应用场景

CloudFile 应用目的是为用户提供方便快捷的文件临时储存及分享服务，并对办公文档（doc(x), ppt(x), xls(x), pdf）提供了生成文档预览的功能，可以在线直接预览第一页以避免下载错误的文件。

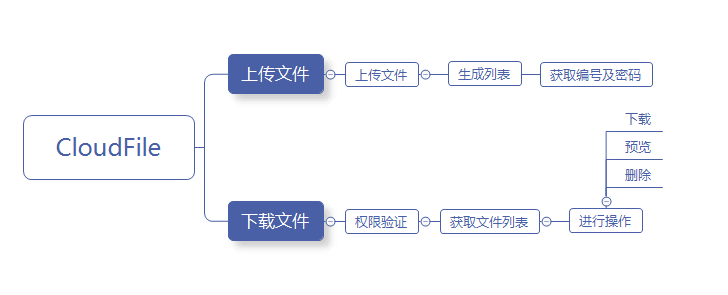
该应用最佳使用场景是在可信网络环境内（如高校校园网）文件的临时保存和分享，可以起到类似于云U盘的作用。一次性上传的多个文件将以文件列表的形式保存，对于一个文件列表增加了编号密码验证以控制访问权限，保证一定的隐私性。

1. 设计思路
2. 技术框架选择

根据上述目标，首先决定使用前后端分离开发的开发模式，这就要求后台需要构建 API 来与前端进行交互，且前端以 SPA 单页应用的形式出现，后台需要在伺服静态文件的同时进行路径重写以交递 URL 控制权。

后台最终选择的框架为跨平台开源框架 ASP.net Core，并使用其 MVC 模式来进行构建；前端最终选择使用 React 库进行单页应用的构建。

1. 产品逻辑



1. 关键问题与实现代码
2. 后端
   1. 文件、ID 等参数的获取

使用 Asp.net Core 动作的 Model Binding 特性来从 URL 及 FormData 中获取数据，其中文件控制器的部分 Action 参数绑定代码如下：

// GET: api/Files

[HttpGet]

public IActionResult GetFile()

// POST: api/Files

[HttpPost]

public async Task<IActionResult> Upload([FromForm]IFormFile uploadFile)

// GET: api/Files/5

[HttpGet("{id}")]

public async Task<IActionResult> GetFile([FromRoute] int id)

* 1. 与数据库的通信及同步

使用 EF(Entity Framework) ORM 框架进行数据库的建立、迁移、获取数据及同步，数据库设计与 Model 模型同步，Files Model 和 Orders Model 如下：

namespace FileUpload.Models

{

// 上传的文件资源类

public class File

{

public int ID { get; set; }

public string FileName { get; set; }

public string Type { get; set; }

public DateTime UploadDate { get; set; }

public string MD5 { get; set; }

// 预览图片，若不是可生成预览的文档则为空

public byte[] PreviewImage { get; set; }

// 文件内容主体

public byte[] FileContent { get; set; }

// 每个文件属于一个订单

public Order Order { get; set; }

}

}

namespace FileUpload.Models

{

// 生成的订单资源类

public class Order

{

public int ID { get; set; }

public DateTime GenerateTime { get; set; }

// 生成的密码

public string Password { get; set; }

// 与文件的一对多关系

public ICollection<File> Files { get; set; }

}

}

* 1. 处理文档预览

文档预览原方案为使用 Open-XML-Tools 来进行处理，不过该库为面向 .Net 完整框架而设计，不能应用在 .Net Core 上，故转向使用外部程序进行处理，最终选择使用 LibreOffice 进行预览图片的生成，代码如下：

namespace FileUpload

{

// 文档预览生成类

public class CovertFile

{

// 生成预览函数

// rootPath - 网站根目录

// fileName - 要储存的临时文件名

// extension - 文件原扩展名，用于外部程序识别

// stream - 包含上传的文件内容的流

public static string Covert(string rootPath, string fileName, string extension, MemoryStream stream)

{

// 预览图生成目录

var outDir = Path.Combine(rootPath, "TempFiles");

// 上传文件临时保存路径

var tmpFileName = Path.Combine(Path.GetTempPath(), $"{fileName}{extension}");

// 预览图保存路径

var resultFileName = Path.Combine(outDir, $"{fileName}.png");

// 创建临时文件并将上传文件写入其中

using (var fileStream = new FileStream(tmpFileName, FileMode.Create))

{

stream.Position = 0;

stream.CopyTo(fileStream);

}

// 创建外部进程，调用 LibreOffice 命令行对文档第一页生成预览

var p = new Process();

p.StartInfo.FileName = "D:\\Program Files\\LibreOffice 5\\program\\soffice.exe";

p.StartInfo.Arguments = $"--headless --convert-to png {tmpFileName} --outdir {outDir}";

p.StartInfo.UseShellExecute = false;

p.StartInfo.CreateNoWindow = true;

p.Start();

// 等待转码完成

p.WaitForExit();

// 删除临时文件

File.Delete(tmpFileName);

// 返回预览图路径

return resultFileName;

}

}

}

* 1. 从数据库获取文件的预览图及图片本身

此处需要两个全新的 API，对于图片来说预览和下载为一个 API，文档则需要分开，API 代码如下：

// 查找对应 id 的文件并返回文件的内容

// GET: api/Files/5/Download

[HttpGet("{id}/Download")]

public async Task<IActionResult> Download([FromRoute] int id)

{

var file = await \_context.File.SingleOrDefaultAsync(m => m.ID == id);

if (file == null)

{

return NotFound();

}

return File(file.FileContent, file.Type, file.FileName);

}

// 查找对应 id 的文件并返回文件的预览图片

// GET: api/Files/5/Preview

[HttpGet("{id}/Preview")]

public async Task<IActionResult> Preview([FromRoute] int id)

{

var file = await \_context.File.SingleOrDefaultAsync(m => m.ID == id);

if (file == null)

{

return NotFound();

}

return File(file.PreviewImage, "image/png", $"{file.MD5}.png");

}

* 1. 文件列表密码的生成及与文件对应关系的建立

为保证密码和其 ID 的弱关系性和密码字符为字幕或数字，此处选择使用 GUID 作为密码，选择其第二部分的四位。

与文件的关系通过前端传入文件的ID来进行关系建立，且禁止已有所属的文件被修改所属关系以保证安全，代码如下：

public async Task<IActionResult> PostOrder([FromBody]OrderPostInfo orderInfo)

{

DateTime generateTime = DateTime.Now;

// 生成 GUID 并取前四位作为密码

string password = Guid.NewGuid().ToString().Split('-')[1];

List<File> filesList = new List<File>();

// 判断传来的 fileId 是否存在，存在则建立与 order 实体的所属关系

foreach (var fileId in orderInfo.files)

{

var file = await \_context.File.SingleOrDefaultAsync(m => m.ID == fileId);

if (file != null && file.Order == null)

{

filesList.Add(file);

}

}

if (filesList.Count < 1)

{

return BadRequest();

}

Order order = new Order { GenerateTime = generateTime, Password = password, Files = filesList };

// 与数据库同步

\_context.Order.Add(order);

await \_context.SaveChangesAsync();

return Ok(new OrderReturnInfo(order, true));

}

1. 前端
   1. 获取后台 API 的数据

使用 Fetch + Fetch Polyfill 进行异步数据获取，代码如下：

*/\*\*  
 \* 删除特定文件  
 \** ***@param*** *id  
 \** ***@return*** *{Promise}  
 \*/***export const** *deleteFile* = id => {  
 **return** fetch(**`/api/files/**${id}**`**, {  
 **method**: **'DELETE'**,  
 **credentials**: **'same-origin'** })  
 .then(*checkStatus*)  
 .then(data => data.**json**())  
}  
  
*/\*\*  
 \* 获取制定 id 的订单详情  
 \** ***@param*** *id - 订单编号  
 \** ***@param*** *password - 密码  
 \** ***@return*** *{Promise}  
 \*/***export const** *getOrder* = (id, password) => {  
 **return** fetch(**`/api/orders/**${id}**?password=**${password}**`**, {  
 **credentials**: **'same-origin'** })  
 .then(*checkStatus*)  
 .then(data => data.**json**())  
}  
  
*/\*\*  
 \* 创建订单并获取编号及密码  
 \** ***@param*** *fileIds - 文件 ID 数组  
 \** ***@return*** *{Promise}  
 \*/***export const** *postOrder* = fileIds => {  
 **return** fetch(**`/api/orders`**, {  
 **method**: **'POST'**,  
 **credentials**: **'same-origin'**,  
 **headers**: {  
 **'Content-Type'**: **'application/json'** },  
 **body**: ***JSON***.stringify({  
 **files**: fileIds  
 })  
 })  
 .then(*checkStatus*)  
 .then(data => data.**json**())  
}

* 1. 上传文件、上传多个文件及显示已上传的文件列表

使用 Ant Design React 组件库中的 Upload 控件进行上传，上传页完整代码如下（不含依赖引入）：

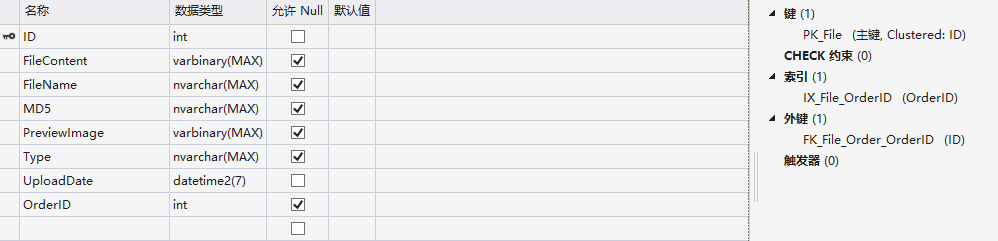
**const** previewFileType = [  
 **'application/vnd.openxmlformats-officedocument.wordprocessingml.document'**,  
**'application/vnd.openxmlformats-officedocument.presentationml.presentation'**,  
 **'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'**,  
 **'application/msword'**,  
 **'application/vnd.ms-powerpoint'**,  
 **'application/vnd.ms-excel'**,  
 **'application/pdf'**]  
  
**class** PageUpload **extends** React.*Component* {  
 **state** = {  
 **fileList**: [],  
 **orderInfo**: **null**,  
 **isGeneratingOrder**: **false** }  
   
 *handleChange* = fileInfo => {  
 **const** file = fileInfo.file  
 **const** fileList = fileInfo.fileList  
   
 **if** (previewFileType.includes(file.type) && file.status === **'uploading'**) {  
 fileList.find(f => f.**uid** === file.uid).**percent** /= 2  
 **if** (file.percent === 100) {  
 message.*info*(**`为** ${file.name} **生成预览图中...`**)  
 }  
 }  
   
 **if**(file.status === **'done'**) {  
 message.*success*(**`**${file.name} **上传成功！`**)  
 fileList.find(f => f.**uid** === file.uid).**percent** = 100  
 }  
   
 **this**.*setState*({  
 fileList  
 })  
 }  
   
 *getSuccessFileList* = () => {  
 **return this**.**state**.**fileList**.filter(file => file.status === **'done'**)  
 }  
   
 *handleSubmitClick* = e => {  
 **this**.*setState*({  
 **isGeneratingOrder**: **true** })  
   
 *postOrder*(**this**.*getSuccessFileList*().map(fileInfo => fileInfo.response.id))  
 .then(data => {  
 **this**.*setState*({  
 **orderInfo**: data,  
 **isGeneratingOrder**: **false** })  
 message.*success*(**`订单上传成功！`**)  
 })  
 }  
   
 *handleModelClose* = () => {  
 **this**.*setState*({  
 **fileList**: [],  
 **orderInfo**: **null** })  
 }  
   
 *handleModelCopy* = () => {  
 **this**.**modelTip**.select()  
 **document**.execCommand(**'copy'**)  
 message.*success*(**'复制成功！'**)  
 }  
   
 render() {  
 **const** fileNum = **this**.*getSuccessFileList*().length  
 **const** hasFile = fileNum > 0  
   
 **let** uploadTip = (  
 <**div**>  
 <**p className='ant-upload-drag-icon'**>  
 <**Icon type='cloud-upload-o'**/>  
 </**p**>  
 <**p className=**{styles[**'upload-text'**]}>  
 拖动或点击此处上传文件  
 </**p**>  
 <**p className=**{styles[**'upload-hint'**]}>  
 支持一次性上传多个文件，可对 Word，Excel，Powerpoint 文档及普通图片生成预览  
 </**p**>  
 </**div**>  
 )  
 **if** (**this**.**state**.**isGeneratingOrder**) {  
 uploadTip = (  
 <**Spin className=**{styles[**'upload-spin-container'**]}>  
 {uploadTip}  
 </**Spin**>  
 )  
 }  
   
 **const** modelEles = **this**.**state**.**orderInfo** ? (  
 <**div className=**{styles[**'model-text'**]}>  
 <**p**>您的订单编号为：<**span**>{**this**.**state**.**orderInfo**.id}</**span**></**p**>  
 <**p**>下载密码为：<**span**>{**this**.**state**.**orderInfo**.**password**}</**span**></**p**>  
 <**p**>请牢记编号和密码以便下载</**p**>  
 <**input ref=**{el => **this**.**modelTip** = el}  
 **type="text"  
 value=**{**`CloudFile 订单编号：**${**this**.**state**.**orderInfo**.id} **密码：**${**this**.**state**.**orderInfo**.**password**}**`**}/>  
 </**div**>  
 ) : **null  
   
 return** (  
 <**div className="section"**>  
 <**Dragger  
 name='uploadFile'  
 multiple=**{**true**}  
 **action='/api/files'  
 fileList=**{**this**.**state**.**fileList**}  
 **onChange=**{**this**.*handleChange*}  
 **className=**{styles.dragger}  
 **disabled=**{**this**.**state**.**isGeneratingOrder**}>  
 {uploadTip}  
 </**Dragger**>  
 <**div className=**{styles[**'btn-wrapper'**]}>  
 <**Badge count=**{fileNum}>  
 <**Button  
 type="primary"  
 size="large"  
 disabled=**{!hasFile}  
 **loading=**{**this**.**state**.**isGeneratingOrder**}  
 **onClick=**{**this**.*handleSubmitClick*}>  
 生成列表  
 </**Button**>  
 </**Badge**>  
 <**Modal  
 visible=**{!!**this**.**state**.**orderInfo**}  
 **title="列表生成成功"  
 onCancel=**{**this**.*handleModelClose*}  
 **footer=**{[  
 <**Button key="copy" onClick=**{**this**.*handleModelCopy*}>复制信息</**Button**>,  
 <**Button key="close" onClick=**{**this**.*handleModelClose*}>关闭</**Button**>  
 ]}>  
 {modelEles}  
 </**Modal**>  
 </**div**>  
 </**div**>  
 )  
 }  
}

* 1. 前端路由控制

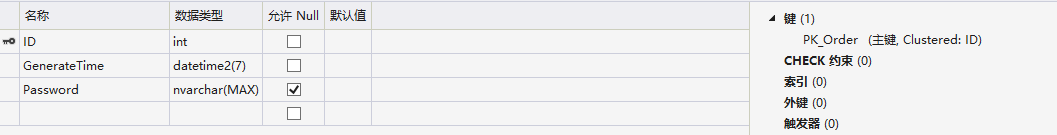
前端路由使用 React Router 进行控制，因代码复杂不再列出。

1. 数据库结构
2. 表 File

File 与 Order 为多对一关系，通过 OrderID 外键关联



1. 表 Order



1. 完整程序代码

完整代码与本文件同目录，主要目录结构如下（后端若需运行需通过 dotnet 命令进行数据库配置，参考[官方文档](https://docs.microsoft.com/zh-cn/aspnet/core/tutorials/first-mvc-app/adding-model#add-initial-migration-and-update-the-database)）：

|- 根目录

|- Readme.docx – 本文档

|- CloudFile.xmind – 产品逻辑图

|- course-WebTechHomeword.sln – 后端项目解决方案

|- Website – 前端项目文件夹（建议使用 Webstorm 打开）

|- build – 构建配置与方法文件夹

|- src – 源代码文件夹

|- gulpfile.js – Gulp 配置文件

|- package.json – npm 配置文件

|- FileUpload – 后端项目文件夹（建议使用 Visual Studio 2017 打开）

|- Models – Model 文件夹

|- Controllers – Controller 文件夹

|- wwwroot – 前端编译后静态文件文件夹

|- Utils – 工具函数文件夹

1. 运行效果图

