# **Web API 文档**

本文档详细描述了 PDFSharingWebAPI 项目提供的后端 API 接口、其用途、请求和响应格式，以及必要的认证要求。

## **基础信息**

* **API Base URL**: /{your\_server\_address}/api (例如: http://localhost:5146/api)
* **认证方式**: 大部分需要用户身份的接口使用 JWT (JSON Web Token) 进行认证。客户端需要在请求头中包含 Authorization: Bearer <Your\_JWT\_Token>。

## **API 接口详情**

### **1. 认证相关 API (/api/Auth)**

负责用户注册和登录。

#### **1.1 用户注册**

* **Endpoint**: /api/Auth/register
* **方法**: POST
* **描述**: 注册新用户。用户数据（用户名、密码哈希、盐）存储在服务器的 users.txt 文件中。
* **请求体 (JSON)**: RegisterRequest Model  
  {  
   "Username": "string", // 用户名  
   "Password": "string" // 密码 (明文，后端会进行哈希加盐处理)  
  }
* **响应 (JSON)**:
  + **成功 (HTTP 200 OK)**:  
    {  
     "Message": "string" // 注册成功的消息  
    }
  + **失败 (HTTP 400 Bad Request)**: 用户已存在或请求参数无效。  
    {  
     "Message": "string" // 失败原因的消息  
    }
* **认证**: 无 (此接口用于获取认证)
* **调用示例 (使用 curl)**:  
  curl -X POST \  
   http://localhost:5146/api/Auth/register \  
   -H 'Content-Type: application/json' \  
   -d '{  
   "Username": "testuser",  
   "Password": "password123"  
   }'

#### **1.2 用户登录**

* **Endpoint**: /api/Auth/login
* **方法**: POST
* **描述**: 用户登录并获取 JWT Token。
* **请求体 (JSON)**: LoginRequest Model  
  {  
   "Username": "string", // 用户名  
   "Password": "string" // 密码 (明文)  
  }
* **响应 (JSON)**: LoginResponse Model
  + **成功 (HTTP 200 OK)**:  
    {  
     "Success": true,  
     "Message": "string", // 登录成功的消息  
     "Token": "string" // JWT Token，用于后续认证  
    }
  + **失败 (HTTP 401 Unauthorized)**: 用户名或密码错误。  
    {  
     "Success": false,  
     "Message": "string", // 失败原因的消息  
     "Token": null  
    }
* **认证**: 无 (此接口用于获取认证)
* **调用示例 (使用 curl)**:  
  curl -X POST \  
   http://localhost:5146/api/Auth/login \  
   -H 'Content-Type: application/json' \  
   -d '{  
   "Username": "testuser",  
   "Password": "password123"  
   }'
* **调用示例 (前端 JavaScript)**:  
  async function loginUser(username, password) {  
   const response = await fetch('/api/auth/login', {  
   method: 'POST',  
   headers: { 'Content-Type': 'application/json' },  
   body: JSON.stringify({ username, password })  
   });  
   const result = await response.json();  
   if (response.ok && result.Success && result.Token) {  
   console.log('Login successful, token:', result.Token);  
   // Store token (e.g., in localStorage) and redirect  
   } else {  
   console.error('Login failed:', result.Message);  
   }  
  }

### **2. 书籍列表相关 API (/api/BookList)**

负责 PDF 文件的上传、删除、列表获取和文件内容提供。

#### **2.1 获取 PDF 文件列表**

* **Endpoint**: /api/BookList
* **方法**: GET
* **描述**: 获取服务器上所有 PDF 文件的列表，包含文件名、创建时间和描述。
* **请求参数**: 无
* **响应 (JSON)**: IEnumerable<PdfFileInfo> Model 数组
  + **成功 (HTTP 200 OK)**:  
    [  
     {  
     "FileName": "string",  
     "Description": "string or null",  
     "CreationTime": "DateTime" // UTC 时间  
     }  
     // ... 更多 PdfFileInfo 对象  
    ]
  + **失败 (HTTP 401 Unauthorized)**: 未认证。
  + **失败 (HTTP 403 Forbidden)**: 无权限。
  + **失败 (HTTP 404 Not Found)**: PDF 文件目录不存在。
  + **失败 (HTTP 500 Internal Server Error)**: 服务器内部错误。
* **认证**: 需要认证 (JWT)
* **调用示例 (使用 curl)**:  
  curl -X GET \  
   http://localhost:5146/api/BookList \  
   -H 'Authorization: Bearer <Your\_JWT\_Token>'
* **调用示例 (前端 JavaScript)**:  
  async function fetchAndRenderBooks() {  
   const token = getJwtToken(); // Assuming getJwtToken() retrieves the stored token  
   if (!token) {  
   console.error('No JWT token found.');  
   return;  
   }  
   const response = await fetch('/api/BookList', {  
   headers: { 'Authorization': `Bearer ${token}` }  
   });  
   if (response.ok) {  
   const books = await response.json();  
   console.log('Books list:', books);  
   // Render books on the page  
   } else {  
   console.error('Failed to fetch books:', response.status, response.statusText);  
   // Handle authentication errors (401/403)  
   }  
  }

#### **2.2 上传 PDF 文件**

* **Endpoint**: /api/BookList
* **方法**: POST
* **描述**: 上传新的 PDF 文件。支持文件数据和元数据（名称、描述）。
* **请求体 (FormData)**: AddBookRequest Model
  + Name: string (required) - 文档名称
  + Description: string (optional) - 文档描述 (最大 500 字符)
  + FileData: IFormFile (required) - PDF 文件数据
* **响应 (JSON)**: AddBookResponse Model
  + **成功 (HTTP 200 OK)**:  
    {  
     "Result": "Success",  
     "Message": "string" // 上传成功的消息  
    }
  + **失败 (HTTP 400 Bad Request)**: 请求参数无效（如文件名为空、描述过长、文件为空）。  
    {  
     "Result": "Failure",  
     "Message": "string" // 失败原因的消息  
    }
  + **失败 (HTTP 401 Unauthorized)**: 未认证。
  + **失败 (HTTP 403 Forbidden)**: 无权限。
  + **失败 (HTTP 500 Internal Server Error)**: 服务器内部错误（如文件处理失败）。
* **认证**: 需要认证 (JWT)
* **调用示例 (使用 curl)**:  
  curl -X POST \  
   http://localhost:5146/api/BookList \  
   -H 'Authorization: Bearer <Your\_JWT\_Token>' \  
   -F 'Name=MyDocument.pdf' \  
   -F 'Description=This is a sample document.' \  
   -F 'FileData=@/path/to/your/document.pdf'
* **调用示例 (前端 JavaScript)**:  
  async function uploadFile(file, name, description) {  
   const token = getJwtToken();  
   if (!token) {  
   console.error('No JWT token found.');  
   return;  
   }  
    
   const formData = new FormData();  
   formData.append('Name', name);  
   formData.append('Description', description);  
   formData.append('FileData', file);  
    
   try {  
   const response = await fetch('/api/BookList', {  
   method: 'POST',  
   headers: { 'Authorization': `Bearer ${token}` },  
   body: formData  
   });  
    
   if (response.status === 401 || response.status === 403) {  
   // Handle re-authentication or permission denied  
   console.error('Authentication failed or no permission.');  
   return;  
   }  
    
   const result = await response.json();  
   if (response.ok && result.Result === "Success") {  
   console.log('Upload successful:', result.Message);  
   // Refresh book list  
   } else {  
   console.error('Upload failed:', result.Message);  
   }  
   } catch (error) {  
   console.error('Upload request failed:', error);  
   }  
  }

#### **2.3 删除 PDF 文件**

* **Endpoint**: /api/BookList
* **方法**: DELETE
* **描述**: 删除指定的 PDF 文件。
* **请求体 (JSON)**: RmvBookRequest Model  
  {  
   "FileName": "string" // 要删除的文件名  
  }
* **响应 (JSON)**: RmvBookResponse Model
  + **成功 (HTTP 200 OK)**:  
    {  
     "Success": true,  
     "Message": "string" // 删除成功的消息  
    }
  + **失败 (HTTP 400 Bad Request)**: 文件名为空或无效。
  + **失败 (HTTP 401 Unauthorized)**: 未认证。
  + **失败 (HTTP 403 Forbidden)**: 无权限。
  + **失败 (HTTP 404 Not Found)**: 目标文件不存在。
  + **失败 (HTTP 500 Internal Server Error)**: 服务器内部错误（如 IO 错误）。
* **认证**: 需要认证 (JWT)
* **调用示例 (使用 curl)**:  
  curl -X DELETE \  
   http://localhost:5146/api/BookList \  
   -H 'Content-Type: application/json' \  
   -H 'Authorization: Bearer <Your\_JWT\_Token>' \  
   -d '{  
   "FileName": "MyDocument.pdf"  
   }'
* **调用示例 (前端 JavaScript)**:  
  async function handleDelete(fileName) {  
   if (!confirm(`确定要删除 ${fileName} 吗 ?`)) return;  
    
   const token = getJwtToken();  
   if (!token) {  
   console.error('No JWT token found.');  
   return;  
   }  
    
   try {  
   const response = await fetch(`/api/BookList`, {  
   method: 'DELETE',  
   headers: {  
   'Content-Type': 'application/json',  
   'Authorization': `Bearer ${token}`  
   },  
   body: JSON.stringify({ FileName: fileName })  
   });  
    
   if (response.status === 401 || response.status === 403) {  
   // Handle re-authentication or permission denied  
   console.error('Authentication failed or no permission.');  
   return;  
   }  
    
   if (response.ok) {  
   const result = await response.json();  
   console.log('Delete successful:', result.Message);  
   // Refresh book list  
   } else {  
   const errorResponse = await response.json();  
   console.error('Delete failed:', errorResponse.Message || response.statusText);  
   }  
   } catch (error) {  
   console.error('Delete request failed:', error);  
   }  
  }

#### **2.4 获取 PDF 文件内容**

* **Endpoint**: /api/BookList/{filename}
* **方法**: GET
* **描述**: 根据文件名获取 PDF 文件的内容。此接口通常用于直接在浏览器中预览 PDF。
* **URL 参数**:
  + filename: string - 要获取的文件名。
* **响应**:
  + **成功 (HTTP 200 OK)**: 返回 PDF 文件的二进制内容 (application/pdf Content-Type)。
  + **失败 (HTTP 400 Bad Request)**: 文件名无效或请求的文件不是 PDF。
  + **失败 (HTTP 403 Forbidden)**: 无权访问文件系统（尽管接口设置为 [AllowAnonymous]，但底层文件系统权限仍可能导致此错误）。
  + **失败 (HTTP 404 Not Found)**: 文件不存在。
  + **失败 (HTTP 500 Internal Server Error)**: 服务器内部错误（如 IO 错误）。
* **认证**: **允许匿名访问** ([AllowAnonymous])，以便通过直链预览。
* **调用示例 (直接在浏览器中访问)**:  
  http://localhost:5146/api/BookList/MyDocument.pdf
* **调用示例 (前端 HTML 链接)**:  
  <a href="/api/BookList/MyDocument.pdf" target="\_blank">在线阅读 MyDocument.pdf</a>

### **3. 分享相关 API (/api/Share)**

负责生成、管理和提供带有时效性的分享链接。

#### **3.1 生成分享链接**

* **Endpoint**: /api/Share/generate
* **方法**: POST
* **描述**: 为指定的 PDF 文件生成一个带有时效性的分享链接。
* **请求体 (JSON)**: GenerateShareLinkRequest Model  
  {  
   "FileName": "string", // 要分享的文件名 (required)  
   "Duration": int, // 时效时长 (e.g., 1, 30, 24)  
   "DurationUnit": "string" // 时效单位 (e.g., "seconds", "minutes", "hours", "days", "permanent") (optional, default is 1 hour)  
  }
* **响应 (JSON)**: GenerateShareLinkResponse Model
  + **成功 (HTTP 200 OK)**:  
    {  
     "Success": true,  
     "Message": "string", // 成功消息  
     "ShareLinkUrl": "string" // 生成的分享链接 URL  
    }
  + **失败 (HTTP 400 Bad Request)**: 请求参数无效。
  + **失败 (HTTP 401 Unauthorized)**: 未认证。
  + **失败 (HTTP 403 Forbidden)**: 无权限。
  + **失败 (HTTP 404 Not Found)**: 要分享的文件不存在。
  + **失败 (HTTP 500 Internal Server Error)**: 服务器内部错误。
* **认证**: 需要认证 (JWT)
* **调用示例 (使用 curl)**:  
  curl -X POST \  
   http://localhost:5146/api/Share/generate \  
   -H 'Content-Type: application/json' \  
   -H 'Authorization: Bearer <Your\_JWT\_Token>' \  
   -d '{  
   "FileName": "MyDocument.pdf",  
   "Duration": 7,  
   "DurationUnit": "days"  
   }'
* **调用示例 (前端 JavaScript)**:  
  async function generateShareLink(fileName, duration, unit, bookCardElement) {  
   const token = getJwtToken();  
   if (!token) {  
   console.error('No JWT token found.');  
   return;  
   }  
   // ... show loading status ...  
   try {  
   const response = await fetch(`/api/share/generate`, {  
   method: 'POST',  
   headers: {  
   'Content-Type': 'application/json',  
   'Authorization': `Bearer ${token}`  
   },  
   body: JSON.stringify({ fileName: fileName, duration: duration, durationUnit: unit })  
   });  
    
   if (response.status === 401 || response.status === 403) {  
   // Handle re-authentication or permission denied  
   console.error('Authentication failed or no permission.');  
   return;  
   }  
    
   const result = await response.json();  
   if (response.ok && result.Success && result.ShareLinkUrl) {  
   console.log('Share link generated:', result.ShareLinkUrl);  
   // Display the link in the UI  
   } else {  
   console.error('Generate share link failed:', result.Message);  
   }  
   } catch (error) {  
   console.error('Generate share link request failed:', error);  
   }  
   // ... hide loading status ...  
  }

#### **3.2 获取分享状态**

* **Endpoint**: /api/Share/status/{token}
* **方法**: GET
* **描述**: 根据分享 Token 获取分享文件的状态信息，包括文件名、描述和过期时间。用于分享页面加载时检查分享有效性。
* **URL 参数**:
  + token: string - 分享 Token。
* **响应 (JSON)**: SharedFileInfoResponse Model
  + **成功 (HTTP 200 OK)**:  
    {  
     "Success": true,  
     "Message": "string", // 文件描述  
     "FileName": "string", // 文件名  
     "ExpirationTime": "DateTime" // 分享过期时间 (UTC)  
    }
  + **失败 (HTTP 401 Unauthorized)**: 分享链接已过期。
  + **失败 (HTTP 404 Not Found)**: 无效的分享码或分享的文件不存在。
  + **失败 (HTTP 500 Internal Server Error)**: 服务器内部错误。
* **认证**: **允许匿名访问** ([AllowAnonymous])。
* **调用示例 (使用 curl)**:  
  curl -X GET \  
   http://localhost:5146/api/Share/status/your\_share\_token
* **调用示例 (前端 JavaScript)**:  
  async function fetchFileInfoAndLoadPdf() {  
   const shareToken = urlParams.get('token'); // Get token from URL  
   if (!shareToken) {  
   showError('分享码缺失');  
   return;  
   }  
   // ... show loading status ...  
   try {  
   const infoResponse = await fetch(`/api/share/status/${shareToken}`);  
    
   if (!infoResponse.ok) {  
   const errorBody = await infoResponse.json();  
   showError(errorBody.Message || `HTTP错误! 状态码: ${infoResponse.status}`);  
   return;  
   }  
    
   const fileInfo = await infoResponse.json();  
   if (fileInfo.Success) {  
   console.log('Share status received:', fileInfo);  
   // Display file info, update countdown, and proceed to load PDF  
   } else {  
   showError(fileInfo.Message || '获取文件信息失败');  
   }  
   } catch (error) {  
   console.error('Fetch file info failed:', error);  
   showError(`获取文件信息失败: ${error.message}`);  
   }  
  }

#### **3.3 获取分享文件内容**

* **Endpoint**: /api/Share/file/{token}
* **方法**: GET
* **描述**: 根据分享 Token 获取 PDF 文件的内容。用于在分享页面中加载和预览 PDF。
* **URL 参数**:
  + token: string - 分享 Token。
* **响应**:
  + **成功 (HTTP 200 OK)**: 返回 PDF 文件的二进制内容 (application/pdf Content-Type)。
  + **失败 (HTTP 401 Unauthorized)**: 分享链接已过期。
  + **失败 (HTTP 404 Not Found)**: 无效的分享码或分享的文件不存在。
  + **失败 (HTTP 500 Internal Server Error)**: 服务器内部错误。
* **认证**: **允许匿名访问** ([AllowAnonymous])。
* **调用示例 (前端 JavaScript)**:  
  async function loadPdf(shareToken) {  
   const pdfUrl = `/api/share/file/${shareToken}`;  
   // ... use PDF.js to load this URL ...  
   try {  
   const loadingTask = pdfjsLib.getDocument(pdfUrl); // PDF.js can load from URL directly  
   const pdf = await loadingTask.promise;  
   console.log('PDF loaded successfully.');  
   // Render PDF pages  
   } catch (error) {  
   console.error('Error loading PDF:', error);  
   showError(`加载 PDF 失败: ${error.message}`);  
   }  
  }

## **后端实现原理概述**

* **文件存储**:
  + PDF 文件存储在项目根目录下的 pdf 文件夹中。
  + 文件描述存储在 pdf 文件夹下的 Description.txt 中，格式为 文件名---描述，每行一个文件。
  + 用户注册信息存储在项目根目录下的 users.txt 中，格式为 用户名|密码哈希|盐，每行一个用户。
  + 分享链接信息存储在项目根目录下的 shares.txt 中，格式为 Token|文件名|过期时间(ISO 8601 UTC)，每行一个分享记录。
* **认证 (JWT)**:
  + 用户登录成功后，后端根据配置 (appsettings.json 中的 Jwt:Key, Issuer, Audience) 生成一个包含用户名的 JWT Token。
  + 这个 Token 会返回给前端，前端需要将其存储（例如在 localStorage 中）。
  + 对于需要认证的 API ([Authorize])，前端在请求头中携带 Authorization: Bearer <Token>。
  + 后端接收到请求后，会验证 Token 的签名、签发者、接收者和有效期。验证通过后，从 Token 中提取用户信息（用户名），并将用户的身份信息附加到 HttpContext 中，供 Controller 中的 Action 使用（例如通过 User.Identity.Name 或 User.FindFirst(ClaimTypes.NameIdentifier)?.Value 获取）。
* **密码存储**:
  + 用户注册时，密码不会明文存储。后端会为每个用户生成一个随机的“盐”（Salt），将用户输入的密码与盐组合后，使用 SHA256 算法进行哈希计算。
  + 存储在 users.txt 中的是用户名、密码哈希和盐。
  + 用户登录时，后端获取存储的盐，将用户输入的密码与该盐组合后进行哈希计算，然后与存储的密码哈希进行比对。
  + **注意**: SHA256 是一种哈希算法，虽然比明文存储安全，但对于密码存储而言，更推荐使用专门的密码哈希算法，如 bcrypt 或 scrypt，它们具有抗彩虹表攻击和计算成本可调的优点。
* **文件操作**:
  + 使用 System.IO 命名空间下的类进行文件和目录的读写操作。
  + 在处理用户上传的文件名时，使用 Path.GetFileName() 进行安全处理，防止路径遍历攻击。
  + 在 ShareController 中，使用 private static readonly object FileLock = new object(); 和 lock (FileLock) 确保在读写 shares.txt 文件时的线程安全，避免并发问题。
  + 在 AuthController 中，使用 ReaderWriterLockSlim 来优化用户文件的读写，允许多个读取者同时访问，但在写入时独占锁定。
* **分享链接管理**:
  + 生成唯一的 GUID 作为分享 Token。
  + 将分享 Token、文件名和过期时间存储在 shares.txt 文件中。
  + 实现了一个后台任务 (StartCleanupTask) 定期检查并清理 shares.txt 中已过期的分享记录。
* **静态文件服务**:
  + 在 Program.cs 中配置了静态文件服务 (app.UseStaticFiles)，将 PDFSharingFE 目录映射到 /static 路径，使得前端文件可以通过 /static/... 访问。
  + 设置了默认文件 (app.UseDefaultFiles)，当访问根路径 / 时重定向到 /static/main.html。