# 常用工具网站 - 项目代码架构设计文档

## 1. 项目概述

### 1.1 项目名称
常用工具网站

### 1.2 项目功能
- 文件转换
- 图片压缩
- 图片裁剪
- 图片格式转换
- 图片水印

### 1.3 技术选型
- 前端：HTML5 + CSS3 + JavaScript
- 后端：Node.js + Express.js
- 数据库：无需数据库（文件处理工具网站）
- 部署：静态文件服务器 + 云函数

## 2. 架构设计

### 2.1 整体架构
```
┌─────────────────────────────────┐    ┌─────────────────────────────────┐
┌─────────────────────────────────┐
│    用户浏览器      │    │    前端页面       │    │    后端服务       │
│                 │    │                 │    │                 │
│ - 文件上传       │◄──►│ - 用户界面       │◄──►│ - 文件处理       │
│ - 结果下载       │    │ - 交互逻辑       │    │ - 格式转换       │
│ - 实时预览       │    │ - 本地处理       │    │ - 图片处理       │
└─────────────────────────────────┘    └─────────────────────────────────┘
└─────────────────────────────────┘
```

### 2.2 是否需要后端服务分析

#### 2.2.1 纯前端实现（无需后端）
- **图片压缩**：使用 Canvas API 在浏览器端处理
- **图片裁剪**：使用 Canvas API 在浏览器端处理
- **图片格式转换**：使用 Canvas API 在浏览器端处理
- **图片水印**：使用 Canvas API 在浏览器端处理

#### 2.2.2 需要后端服务
- **文件转换**：支持文档类文件（TXT、PDF、Word（doc/docx）、PPT、Excel、RTF、HTML、EPUB 等）的批量格式互转，支持批量上传与批量转换。

## 3. 详细架构设计

### 3.1 前端架构

#### 3.1.1 目录结构
```
frontend/
├── index.html              # 首页
├── pages/                  # 功能页面
│   ├── file-convert.html   # 文件转换
│   ├── image-compress.html # 图片压缩
│   ├── image-crop.html     # 图片裁剪
```

```
│   ├── format-convert.html # 格式转换
│   └── watermark.html      # 图片水印
├── css/                    # 样式文件
│   ├── common.css          # 公共样式
│   ├── home.css            # 首页样式
│   └── pages.css           # 页面样式
├── js/                     # JavaScript文件
│   ├── common.js           # 公共函数
│   ├── upload.js           # 文件上传
│   ├── image-process.js    # 图片处理
│   └── file-convert.js     # 文件转换
└── assets/                 # 静态资源
    ├── images/             # 图片资源
    └── icons/              # 图标资源
```

#### 3.1.2 核心模块

**文件上传模块 (upload.js)**
- 拖拽上传功能
- 文件格式验证
- 文件大小限制
- 多文件上传支持

**图片处理模块 (image-process.js)**
- Canvas图片处理
- 压缩算法实现
- 裁剪功能
- 格式转换
- 水印添加

**文件转换模块 (file-convert.js)**
- 支持多文件上传到后端
- 动态识别文档类型与目标格式
- 转换进度显示
- 批量结果下载

### 3.2 后端架构

#### 3.2.1 目录结构
```
backend/
├── server.js               # 服务器入口
├── routes/                 # 路由文件
│   └── convert.js          # 文件转换路由
├── services/               # 服务层
│   └── fileService.js      # 文件处理服务
├── utils/                  # 工具函数
│   ├── fileUtils.js        # 文件工具
│   └── formatUtils.js      # 格式转换工具
├── uploads/                # 上传文件临时目录
├── outputs/                # 输出文件目录
└── package.json            # 项目配置
```

#### 3.2.2 核心模块

**文件转换服务 (fileService.js / formatConvertService.js)**
- 文档类：TXT、PDF、Word、PPT、Excel等互转（集成LibreOffice、unoconv等）
- 批量文件处理
- 文件清理

**路由处理 (convert.js / formatConvert.js)**
- 多文件上传接口
- 转换处理接口
- 批量文件下载接口

## 4. 技术实现方案

### 4.1 前端技术栈
- **HTML5**: 页面结构，文件上传API
- **CSS3**: 响应式布局，动画效果
- **JavaScript ES6+**: 业务逻辑，Canvas处理
- **Canvas API**: 图片处理核心
- **File API**: 文件操作
- **Web Workers**: 大文件处理

### 4.2 后端技术栈
- **Node.js**: 服务器运行环境
- **Express.js**: Web框架
- **Multer**: 多文件上传中间件
- **LibreOffice/unoconv**: 文档格式转换
- **pdf-lib/mammoth**: PDF/Word处理
- **Archiver**: 批量下载打包

### 4.3 第三方库
- **前端**:
  - FileSaver.js: 文件下载
  - Progress.js: 进度条

- **后端**:
  - LibreOffice/unoconv: 文档转换
  - pdf-lib/mammoth: PDF/Word处理
  - Archiver: 批量打包下载

## 5. 部署方案

### 5.1 开发环境
- 前端：本地文件服务器（Live Server）
- 后端：Node.js本地服务器
- 端口：前端3000，后端5000

### 5.2 生产环境
- 前端：静态文件托管（Nginx/Apache）
- 后端：云服务器（阿里云/腾讯云）
- 文件存储：对象存储（OSS/COS）
- CDN：静态资源加速

## 6. 性能优化

### 6.1 前端优化
- 图片懒加载
- 文件分片上传
- Web Workers处理大文件

- 本地缓存处理结果

### 6.2 后端优化
- 文件处理队列
- 临时文件清理
- 内存使用优化
- 并发处理限制

## 7. 安全考虑

### 7.1 文件安全
- 文件类型验证
- 文件大小限制
- 病毒扫描（可选）
- 临时文件清理

### 7.2 接口安全
- 请求频率限制
- 文件上传限制
- 错误信息处理
- HTTPS 传输

## 8. 开发计划

### 8.1 第一阶段（纯前端功能）
- 图片压缩功能
- 图片裁剪功能
- 图片格式转换
- 图片水印功能

### 8.2 第二阶段（后端服务）
- 文件转换功能
- 服务器部署
- 性能优化

### 8.3 第三阶段（完善优化）
- 用户体验优化
- 功能测试
- 部署上线

## 9. 总结

本项目采用前后端分离架构，大部分图片处理功能可在前端完成，只有文档转换需要后端服务支持。整体架构简单清晰，易于开发和维护。