```
const fs = require('fs');
const path = require('path');
const {
  Document, Packer, Paragraph, TextRun, ImageRun, Header, Footer
,
  AlignmentType, PageOrientation, HeadingLevel, TableOfContents,
  LevelFormat, PageNumber
} = require('docx');
const outPath = path.join(__dirname, 'softreg-doc-v1.0-cn.docx')
;
const uiImagePath = path.join(__dirname, 'screenshot-app-ui.png'
);
const exportImagePath = path.join(__dirname, 'screenshot-export-
doc.png');
const makeCenter = (text) => new Paragraph({
  alignment: AlignmentType.CENTER,
  spacing: { after: 120 },
  children: [new TextRun({ text, size: 28 })]
});
const makeParagraph = (text) => new Paragraph({
  spacing: { after: 120 },
  children: [new TextRun({ text, size: 24 })]
});
const makeBullet = (text) => new Paragraph({
  numbering: { reference: 'bullet-list', level: 0 },
  children: [new TextRun({ text, size: 24 })]
});
const makeStep = (text) => new Paragraph({
  numbering: { reference: 'step-list', level: 0 },
  children: [new TextRun({ text, size: 24 })]
});
const scaleToWidth = (w, h, targetW) => {
  const ratio = targetW / w;
  return { width: Math.round(w * ratio), height: Math.round(h *
ratio) };
};
const addImageBlock = (title, imagePath, size, altText) => {
  const data = fs.readFileSync(imagePath);
  return [
    new Paragraph({
      spacing: { after: 120 },
      children: [new TextRun({ text: title, bold: true, size: 24
 })]
    }),
    new Paragraph({
      alignment: AlignmentType.CENTER,
      spacing: { after: 120 },
      children: [new ImageRun({
        type: 'png',
        data,
```

```
        transformation: { width: size.width, height: size.height
, rotation: 0 },
        altText: { title: altText, description: altText, name: a
ltText }
      })]
    })
  ];
};
const uiSize = scaleToWidth(939, 1547, 420);
const exportSize = scaleToWidth(927, 1291, 420);
const doc = new Document({
  styles: {
    default: { document: { run: { font: 'SimSun', size: 24 } } }
,
    paragraphStyles: [
      { id: 'Title', name: 'Title', basedOn: 'Normal',
        run: { size: 56, bold: true, color: '000000', font: 'Mic
rosoft YaHei' },
        paragraph: { spacing: { before: 240, after: 120 }, align
ment: AlignmentType.CENTER } },
      { id: 'Heading1', name: 'Heading 1', basedOn: 'Normal', ne
xt: 'Normal', quickFormat: true,
        run: { size: 32, bold: true, color: '000000', font: 'Mic
rosoft YaHei' },
        paragraph: { spacing: { before: 240, after: 180 }, outli
neLevel: 0 } },
      { id: 'Heading2', name: 'Heading 2', basedOn: 'Normal', ne
xt: 'Normal', quickFormat: true,
        run: { size: 28, bold: true, color: '000000', font: 'Mic
rosoft YaHei' },
        paragraph: { spacing: { before: 180, after: 120 }, outli
neLevel: 1 } }
    ]
  },
  numbering: {
    config: [
      {
        reference: 'bullet-list',
        levels: [{
          level: 0,
          format: LevelFormat.BULLET,
          text: '\u2022',
          alignment: AlignmentType.LEFT,
          style: { paragraph: { indent: { left: 720, hanging: 36
0 } } } }
        }]
      },
      {
        reference: 'step-list',
        levels: [{
```

```
          level: 0,
          format: LevelFormat.DECIMAL,
          text: '%1.',
          alignment: AlignmentType.LEFT,
          style: { paragraph: { indent: { left: 720, hanging: 36
0 } } }
        }]
      }
    ]
  },
  sections: [{
    properties: {
      page: {
        margin: { top: 1440, right: 1440, bottom: 1440, left: 14
40 },
        size: { orientation: PageOrientation.PORTRAIT },
        pageNumbers: { start: 1, formatType: 'decimal' }
      }
    },
    headers: {
      default: new Header({
        children: [new Paragraph({
          alignment: AlignmentType.RIGHT,
          children: [new TextRun({ text: '软著鉴别材料整理器软件 v1.0 说明
书',
size: 20 })]
        })]
      })
    },
    footers: {
      default: new Footer({
        children: [new Paragraph({
          alignment: AlignmentType.CENTER,
          children: [new TextRun({ children: [PageNumber.CURRENT
] })]
        })]
      })
    },
    children: [
      new Paragraph({ heading: HeadingLevel.TITLE, children: [ne
w TextRun('软件说明书')] }),
      makeCenter('软件名称：软著鉴别材料整理器软件 v1.0'),
      makeCenter(`编制日期：${new Date().toISOString().slice(0, 10)}`
),
      makeCenter('适用范围：软著鉴别材料（软件说明与鉴别材料整理）'),
      new Paragraph({ children: [new TextRun({ text: '', size: 2
4 })], pageBreakBefore: true }),
      new TableOfContents('目录', { hyperlink: true, headingStyleR
ange: '1-3' }),
      new Paragraph({ children: [new TextRun({ text: '', size: 2
4 })], pageBreakBefore: true }),
```

```
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
 [new TextRun('1 软件概述')] }),
        makeParagraph('软著鉴别材料整理器软件 v1.0 用于自动化整理软著登记所需的说明与鉴别材料，支持对软件信
 息、功能模块、界面截图、生成文档等内容进行统一管理与输出。'),
        makeParagraph('软件定位：面向需要提交软著材料的开发者、代理机构或企业，通过规范化模板提升材料完整性与
 一致性。'),
        makeParagraph('主要特点：'),
        makeBullet('统一模板：内置说明书模板与结构化字段，确保格式规范。'),
        makeBullet('材料集中管理：截图、功能描述、版本信息集中归档。'),
        makeBullet('一键生成：自动输出 Word 文档，减少人工整理成本。'),
        makeBullet('可追溯：生成记录与版本信息可回溯，便于复用。'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
 [new TextRun('2 运行环境与部署')] }),
        makeParagraph('支持操作系统：Windows 10/11。'),
        makeParagraph('运行依赖：本地安装 Microsoft Word 或兼容的文档查看器（用于查看生成的 doc
 x）。'),
        makeParagraph('部署方式：应用安装包本地安装，无需联网。'),
        makeParagraph('启动方式：双击应用图标启动，进入主界面。'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
 [new TextRun('3 系统架构与模块')] }),
        makeParagraph('系统采用模块化结构，核心模块如下：'),
        makeBullet('资料管理模块：录入软件名称、版本、用途、开发环境等基础信息。'),
        makeBullet('截图管理模块：导入界面截图、输出示例图，支持排序与说明。'),
        makeBullet('说明书生成模块：按模板生成包含封面、目录、正文的说明文档。'),
        makeBullet('导出与校验模块：导出 Word 文档并校验完整性。'),
        makeBullet('设置与日志模块：保存用户配置与生成记录。'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
 [new TextRun('4 功能说明')] }),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
 [new TextRun('4.1 资料采集与整理')] }),
        makeParagraph('提供字段化信息录入，包括软件名称、版本号、运行环境、功能描述、适用范围等。'),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
 [new TextRun('4.2 界面截图管理')] }),
        makeParagraph('支持导入界面截图与输出示例图，自动归类并生成插图说明。'),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
 [new TextRun('4.3 说明书自动生成')] }),
        makeParagraph('按软著登记要求生成包含封面、目录、正文与截图的 Word 说明文档。'),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
 [new TextRun('4.4 导出与打包')] }),
        makeParagraph('支持一键导出 docx 文件，便于后续打印或提交。'),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
 [new TextRun('4.5 日志与版本信息')] }),
```

```
193         makeParagraph('记录每次生成的时间、版本和素材变更，便于追溯。'),
194         new Paragraph({ heading: HeadingLevel.HEADING_1, children:
195   [new TextRun('5 操作流程')] }),
196         makeParagraph('标准操作流程如下：'),
197         makeStep('启动软件并进入主界面。'),
198         makeStep('录入软件基础信息与功能说明。'),
199         makeStep('导入界面截图与输出示例截图。'),
200         makeStep('选择模板并生成说明书。'),
```

```
      makeStep('导出 docx 文档并进行检查。'),
      new Paragraph({ heading: HeadingLevel.HEADING_1, children:
 [new TextRun('6 数据结构与存储')] }),
      makeParagraph('软件在本地保存项目数据，包含基础信息、截图素材与生成结果。数据不上传网络，默认保存于用
户工作目录。'),
      makeParagraph('数据结构包括：项目信息、截图列表、生成记录、导出文档。'),
      new Paragraph({ heading: HeadingLevel.HEADING_1, children:
 [new TextRun('7 安全与权限')] }),
      makeParagraph('软件为本地离线工具，默认不进行网络通信。'),
      makeParagraph('数据仅保存在本地，遵循操作系统文件权限控制。'),
      new Paragraph({ heading: HeadingLevel.HEADING_1, children:
 [new TextRun('8 兼容性与限制')] }),
      makeParagraph('输出文档格式为 docx，需在 Microsoft Word 或兼容软件中打开。'),
      makeParagraph('部分排版效果在不同版本的 Word 中可能存在细微差异。'),
      new Paragraph({ heading: HeadingLevel.HEADING_1, children:
 [new TextRun('9 运行与维护')] }),
      makeParagraph('建议定期备份项目数据与导出文档。'),
      makeParagraph('版本升级时可保留旧版本说明书以便比对。'),
      new Paragraph({ heading: HeadingLevel.HEADING_1, children:
 [new TextRun('10 附录：截图')] }),
      new Paragraph({ heading: HeadingLevel.HEADING_2, children:
 [new TextRun('10.1 软件界面截图')] }),
      ...addImageBlock('图 1 软件主界面（UI 截图）', uiImagePath, uiSize, '软
件主界面截图'),
      new Paragraph({ heading: HeadingLevel.HEADING_2, children:
 [new TextRun('10.2 导出文档示例')] }),
      ...addImageBlock('图 2 导出文档示例', exportImagePath, exportSize,
 '导出文档示例截图')
    ]
  }]
});
Packer.toBuffer(doc).then((buffer) => {
  fs.writeFileSync(outPath, buffer);
  console.log(`Generated: ${outPath}`);
}).catch((err) => {
  console.error(err);
  process.exit(1);
});
const { app, BrowserWindow, dialog, ipcMain, shell } = require('
electron');
const path = require('path');
const fs = require('fs');
const { scanProject } = require('./shared/scanner');
const { scanProjectForAI } = require('./shared/aiScanner');
const os = require('os');
const { writePdfFromDocx } = require('./shared/exportPdf');
const { writeDocx } = require('./shared/exportDocx');
```

```
248  const { writeManualDocx } = require('./shared/exportManualDocx')
249  ;
250  const Store = require('electron-store');
```

```
const { pathToFileURL } = require('url');
const { autoUpdater } = require('electron-updater');
const store = new Store();
let mainWindow;
let lastScan = null;
function getWindowIcon() {
  const baseDir = path.join(__dirname, '..', 'build');
  if (process.platform === 'win32') {
    return path.join(baseDir, 'icon.ico');
  }
  return path.join(baseDir, 'icon.png');
}
function setupAutoUpdate() {
  if (!app.isPackaged) return;
  autoUpdater.autoDownload = true;
  autoUpdater.autoInstallOnAppQuit = true;
  autoUpdater.on('error', async (error) => {
    try {
      await dialog.showMessageBox(mainWindow, {
        type: 'error',
        title: '更新失败',
        message: '检查更新时发生错误',
        detail: String(error?.message || error || '')
      });
    } catch {}
  });
  autoUpdater.on('update-available', async () => {
    try {
      await dialog.showMessageBox(mainWindow, {
        type: 'info',
        title: '发现新版本',
        message: '发现新版本，正在后台下载…',
        buttons: ['知道了'],
        defaultId: 0
      });
    } catch {}
  });
  autoUpdater.on('update-downloaded', async () => {
    try {
      const result = await dialog.showMessageBox(mainWindow, {
        type: 'info',
        title: '更新已下载',
        message: '新版本已下载完成，是否立即重启并安装？',
        buttons: ['立即重启安装', '稍后'],
        defaultId: 0,
        cancelId: 1
      });
      if (result.response === 0) {
        autoUpdater.quitAndInstall();
      }
```

```
    } catch {}
  });
  autoUpdater.checkForUpdates().catch(() => {});
}
function createWindow() {
  mainWindow = new BrowserWindow({
    width: 1100,
    height: 780,
    title: '软著代码资料整理器',
    icon: getWindowIcon(),
    webPreferences: {
      preload: path.join(__dirname, 'preload.js'),
      contextIsolation: true,
      nodeIntegration: false
    }
  });
  mainWindow.loadFile(path.join(__dirname, 'renderer', 'index.ht
ml'));
}
app.whenReady().then(() => {
  createWindow();
  setupAutoUpdate();
  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) createWindow
();
  });
});
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') app.quit();
});
ipcMain.handle('select-project', async () => {
  const result = await dialog.showOpenDialog(mainWindow, {
    properties: ['openDirectory']
  });
  if (result.canceled || result.filePaths.length === 0) return n
ull;
  return result.filePaths[0];
});
ipcMain.handle('select-output-dir', async () => {
  const result = await dialog.showOpenDialog(mainWindow, {
    properties: ['openDirectory']
  });
  if (result.canceled || result.filePaths.length === 0) return n
ull;
  return result.filePaths[0];
});
ipcMain.handle('select-ai-screenshots', async () => {
  const result = await dialog.showOpenDialog(mainWindow, {
    properties: ['openFile', 'multiSelections'],
    filters: [
```

```
      { name: 'Images', extensions: ['png', 'jpg', 'jpeg', 'webp
', 'bmp'] }
    ]
  });
  if (result.canceled || result.filePaths.length === 0) return [
];
  return result.filePaths.map(filePath => ({
    path: filePath,
    url: pathToFileURL(filePath).href,
    name: path.basename(filePath)
  }));
});
ipcMain.handle('scan-project', async (_event, options) => {
  const scan = scanProject(options.projectPath);
  if (scan.totalPages === 0) {
    throw new Error('未找到可用源码文件。');
  }
  lastScan = {
    ...scan,
    headerText: options.headerText,
    pagesToExport: scan.pagesToExport
  };
  return {
    projectPath: scan.projectPath,
    fileCount: scan.fileCount,
    effectiveLines: scan.effectiveLines,
    totalPages: scan.totalPages,
    selectedPages: scan.pagesToExport.length
  };
});
ipcMain.handle('generate-docx', async (_event, options) => {
  if (!lastScan) throw new Error('请先扫描项目。');
  const outputPath = path.join(options.outputDir, options.fileNa
me);
  await writeDocx(outputPath, lastScan.pagesToExport, lastScan.h
eaderText);
  return outputPath;
});
ipcMain.handle('generate-pdf', async (_event, options) => {
  if (!lastScan) throw new Error('请先扫描项目。');
  const outputPath = path.join(options.outputDir, options.fileNa
me);
  const tempDir = fs.mkdtempSync(path.join(os.tmpdir(), 'softreg
-'));
  const tempDocx = path.join(tempDir, 'program-material.docx');
  await writeDocx(tempDocx, lastScan.pagesToExport, lastScan.hea
derText);
  writePdfFromDocx(tempDocx, outputPath);
  return outputPath;
});
```

```
ipcMain.handle('set-export-pages', async (_event, mode) => {
  if (!lastScan) throw new Error('请先扫描项目。');
  if (mode === 'all') {
    lastScan.pagesToExport = lastScan.pages;
  } else {
    lastScan.pagesToExport = lastScan.selectedPages;
  }
  return {
    pagesToExport: lastScan.pagesToExport.length
  };
});
ipcMain.handle('init-scan-state', async () => {
  lastScan = null;
});
ipcMain.handle('open-external', async (_event, url) => {
  await shell.openExternal(url);
});
ipcMain.handle('open-output-dir', async (_event, outputDir) => {
  const dir = String(outputDir || '').trim();
  if (!dir) return false;
  const result = await shell.openPath(dir);
  if (result) throw new Error(result);
  return true;
});
ipcMain.handle('ensure-output-dir', async (_event, outputDir) =>
 {
  if (!fs.existsSync(outputDir)) {
    fs.mkdirSync(outputDir, { recursive: true });
  }
  return true;
});
ipcMain.handle('ai:save-config', async (_event, config) => {
  store.set('aiConfig', config);
  return true;
});
ipcMain.handle('ai:get-config', async () => {
  return store.get('aiConfig') || {
    baseUrl: 'https://api.openai.com/v1',
    apiKey: '',
    model: 'gpt-3.5-turbo'
  };
});
ipcMain.handle('ai:generate-manual', async (_event, { projectPat
h, config, screenshots }) => {
  try {
    const context = scanProjectForAI(projectPath);
    const shotList = Array.isArray(screenshots) ? screenshots :
[];
    const shotText = shotList.length
      ? shotList
```

```
            .map((s, i) => {
              const note = String(s?.note || '').trim();
              const name = String(s?.name || '').trim();
              const label = note || name || String(s?.path || '').
trim();
              return `${i + 1}. ${label}`;
            })
            .join('\n')
        : '（无）';
    const prompt = `
你是一个专业的软件文档撰写专家。请根据以下项目上下文，撰写一份详细的《软件说明
书》。
请严格使用 Markdown 标题（# / ## / ###）组织结构，标题不要只用纯编号列表代
替。
## 截图列表（可用于插图）
${shotText}
当正文需要插入某张截图时，请在合适位置“单独输出一行”占位符，格式必须严格为：
[[SCREENSHOT:截图说明]]
其中“截图说明”必须与上面的截图列表某一项完全一致（优先使用用户填写的备注）。
## 1. 项目结构
\`\`\`
${context.structure}
\`\`\`
## 2. README 内容
${context.readme}
## 3. 依赖/配置信息
${context.packageJson}
## 4. 核心代码片段
${context.sourceSnippets.map(s => `--- ${s.path} ---\n${s.conten
t}`).join('\n\n')}
请输出 Markdown 格式，包含以下章节：
1. # 软件概述（根据 README 和代码推断）
2. # 功能列表（详细列出功能点）
3. # 技术特点（架构、语言、关键技术）
4. # 运行环境要求
5. # 使用说明（如有）
`;
    const baseUrl = config.baseUrl.replace(/\/$/, '');
    const response = await fetch(`${baseUrl}/chat/completions`,
{
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${config.apiKey}`
      },
      body: JSON.stringify({
        model: config.model,
        messages: [
          { role: 'system', content: '你是一个专业的软件技术文档专家，擅长
编写软件说明书。
' },
```

```
499             { role: 'user', content: prompt }
500           ],
```

```
        temperature: 0.7
      })
    });
    if (!response.ok) {
      const errText = await response.text();
      throw new Error(`AI 请求失败 (${response.status}): ${errText}`
);
    }
    const data = await response.json();
    if (!data.choices || data.choices.length === 0) {
      throw new Error('AI 返回结果为空');
    }
    return data.choices[0].message.content;
  } catch (error) {
    console.error('AI Generation Error:', error);
    throw error;
  }
});
ipcMain.handle('ai:export-manual-docx', async (_event, options)
=> {
  const outputPath = path.join(options.outputDir, options.fileNa
me);
  await writeManualDocx(outputPath, options);
  return outputPath;
});
ipcMain.handle('ai:export-manual-pdf', async (_event, options) =
> {
  const outputPath = path.join(options.outputDir, options.fileNa
me);
  const tempDir = fs.mkdtempSync(path.join(os.tmpdir(), 'softreg
-manual-'));
  const tempDocx = path.join(tempDir, 'manual.docx');
  await writeManualDocx(tempDocx, options);
  writePdfFromDocx(tempDocx, outputPath);
  return outputPath;
});
ipcMain.handle('save-file', async (_event, { content, defaultNam
e }) => {
  const result = await dialog.showSaveDialog(mainWindow, {
    defaultPath: defaultName,
    filters: [{ name: 'Markdown', extensions: ['md'] }]
  });
  if (!result.canceled && result.filePath) {
    fs.writeFileSync(result.filePath, content, 'utf8');
    return result.filePath;
  }
  return null;
});
const { contextBridge, ipcRenderer } = require('electron');
contextBridge.exposeInMainWorld('softreg', {
```

```
  selectProject: () => ipcRenderer.invoke('select-project'),
  selectOutputDir: () => ipcRenderer.invoke('select-output-dir')
,
  scanProject: (options) => ipcRenderer.invoke('scan-project', o
ptions),
  generateDocx: (options) => ipcRenderer.invoke('generate-docx',
 options),
  generatePdf: (options) => ipcRenderer.invoke('generate-pdf', o
ptions),
  setExportPages: (mode) => ipcRenderer.invoke('set-export-pages
', mode),
  initScanState: () => ipcRenderer.invoke('init-scan-state'),
  ensureOutputDir: (outputDir) => ipcRenderer.invoke('ensure-out
put-dir', outputDir),
  openExternal: (url) => ipcRenderer.invoke('open-external', url
),
  openOutputDir: (outputDir) => ipcRenderer.invoke('open-output-
dir', outputDir),
  ai: {
    saveConfig: (config) => ipcRenderer.invoke('ai:save-config',
 config),
    getConfig: () => ipcRenderer.invoke('ai:get-config'),
    generateManual: (projectPath, config, screenshots) => ipcRen
derer.invoke('ai:generate-manual', { projectPath, config, screen
shots }),
    exportManualDocx: (options) => ipcRenderer.invoke('ai:export
-manual-docx', options),
    exportManualPdf: (options) => ipcRenderer.invoke('ai:export-
manual-pdf', options)
  },
  selectAiScreenshots: () => ipcRenderer.invoke('select-ai-scree
nshots'),
  saveFile: (content, defaultName) => ipcRenderer.invoke('save-f
ile', { content, defaultName })
});
const projectPathInput = document.getElementById('projectPath');
const outputDirInput = document.getElementById('outputDir');
const softwareNameInput = document.getElementById('softwareName'
);
const softwareVersionInput = document.getElementById('softwareVe
rsion');
const statFiles = document.getElementById('statFiles');
const statLines = document.getElementById('statLines');
const statPages = document.getElementById('statPages');
const statExport = document.getElementById('statExport');
const statExportLines = document.getElementById('statExportLines
');
const statusEl = document.getElementById('status');
const btnSelectProject = document.getElementById('btnSelectProje
ct');
```

```
const btnSelectOutput = document.getElementById('btnSelectOutput
');
const btnScan = document.getElementById('btnScan');
const btnExportDocx = document.getElementById('btnExportDocx');
const btnExportPdf = document.getElementById('btnExportPdf');
const btnExportBoth = document.getElementById('btnExportBoth');
const lightRed = document.querySelector('.light.red');
const lightAmber = document.querySelector('.light.amber');
const lightGreen = document.querySelector('.light.green');
function updateLights(state) {
  lightRed.classList.remove('active');
  lightAmber.classList.remove('active');
  lightGreen.classList.remove('active');
  switch (state) {
    case 'idle':
      lightRed.classList.add('active');
      break;
    case 'busy':
      lightAmber.classList.add('active');
      break;
    case 'success':
      lightGreen.classList.add('active');
      break;
    case 'error':
      lightRed.classList.add('active');
      break;
  }
}
function formatNum(num, length) {
  return String(num).padStart(length, '0');
}
function getSoftwareType() {
  const selected = document.querySelector('input[name="softwareT
ype"]:checked');
  return selected ? selected.value : 'software';
}
function buildHeaderText() {
  const name = softwareNameInput.value.trim();
  const version = softwareVersionInput.value.trim();
  const type = getSoftwareType() === 'game' ? '游戏软件' : '软件';
  if (!name || !version) return '';
  return `${name}${type}V${version}`;
}
function setStatus(message) {
  statusEl.textContent = message.toUpperCase();
  console.log(`[系统] ${message}`);
}
function validateInputs() {
  if (!projectPathInput.value) {
    setStatus('错误: 缺失项目路径');
```

```
      updateLights('error');
      return false;
    }
    if (!softwareNameInput.value.trim()) {
      setStatus('错误: 需填写软件名称');
      updateLights('error');
      return false;
    }
    if (!softwareVersionInput.value.trim()) {
      setStatus('错误: 需填写版本号');
      updateLights('error');
      return false;
    }
    if (!outputDirInput.value) {
      setStatus('错误: 缺失输出路径');
      updateLights('error');
      return false;
    }
    return true;
  }
  async function handleSelectProject() {
    const selected = await window.softreg.selectProject();
    if (selected) projectPathInput.value = selected;
  }
  async function handleSelectOutput() {
    const selected = await window.softreg.selectOutputDir();
    if (selected) outputDirInput.value = selected;
  }
  const scanOverlay = document.getElementById('scanOverlay');
  let audioCtx = null;
  function playScanSound() {
    if (!audioCtx) audioCtx = new (window.AudioContext || window.w
  ebkitAudioContext)();
    if (audioCtx.state === 'suspended') audioCtx.resume();
    const now = audioCtx.currentTime;
    const duration = 2.5; // 与填充动画时长一致
    const osc1 = audioCtx.createOscillator();
    const osc2 = audioCtx.createOscillator(); // 副振荡器 (正弦波 - 提供底
蕴)
    const filter = audioCtx.createBiquadFilter();
    const gainNode = audioCtx.createGain();
    osc1.type = 'sawtooth';
    osc1.frequency.setValueAtTime(80, now);
    osc1.frequency.exponentialRampToValueAtTime(450, now + duratio
  n);
    osc2.type = 'sine';
    osc2.frequency.setValueAtTime(40, now);
    osc2.frequency.exponentialRampToValueAtTime(120, now + duratio
  n);
    filter.type = 'lowpass';
    filter.Q.value = 15; // 高共振
```

```
  filter.frequency.setValueAtTime(200, now);
  filter.frequency.exponentialRampToValueAtTime(3000, now + dura
tion);
  gainNode.gain.setValueAtTime(0, now);
  gainNode.gain.linearRampToValueAtTime(0.08, now + 0.1); // 快速开
启
  gainNode.gain.exponentialRampToValueAtTime(0.001, now + durati
on); // 逐渐消失
  osc1.connect(filter);
  osc2.connect(filter);
  filter.connect(gainNode);
  gainNode.connect(audioCtx.destination);
  osc1.start(now);
  osc2.start(now);
  osc1.stop(now + duration);
  osc2.stop(now + duration);
}
function startButtonEffect(btn) {
  btn.classList.add('processing');
  playScanSound();
}
function stopButtonEffect(btn) {
  btn.classList.remove('processing');
}
async function handleScan() {
  if (!projectPathInput.value) {
    setStatus('错误: 请先选择项目目录');
    updateLights('error');
    return;
  }
  const headerText = buildHeaderText();
  if (!headerText) {
    setStatus('错误: 名称和版本号缺失');
    updateLights('error');
    return;
  }
  try {
    setStatus('状态: 正在扫描项目文件...');
    updateLights('busy');
    startButtonEffect(btnScan);
    await window.softreg.initScanState();
    const result = await window.softreg.scanProject({
      projectPath: projectPathInput.value,
      headerText
    });
    await new Promise(resolve => setTimeout(resolve, 2500));
    statFiles.textContent = formatNum(result.fileCount, 4);
    statLines.textContent = formatNum(result.effectiveLines, 6);
    statPages.textContent = formatNum(result.totalPages, 3);
    statExport.textContent = formatNum(result.selectedPages, 3);
```

```
    statExportLines.textContent = formatNum(result.selectedPages
 * 50, 4);
    const truncationNote = result.totalPages > 60 ? ' (已截断至 3000
 行)' : '';
    setStatus(`状态: 扫描完成${truncationNote}. 准备就绪.`);
    updateLights('success');
  } catch (err) {
    setStatus(`错误: 扫描失败 // ${String(err.message || err).toUpperC
ase()}`);
    updateLights('error');
  } finally {
    stopButtonEffect(btnScan);
  }
}
function buildFileName(suffix) {
  const headerText = buildHeaderText();
  return `${headerText}_${suffix}`;
}
async function exportDocx() {
  if (!validateInputs()) return;
  try {
    setStatus('状态: 正在生成 DOCX 数据流...');
    updateLights('busy');
    startButtonEffect(btnExportDocx);
    await window.softreg.ensureOutputDir(outputDirInput.value);
    const fileName = buildFileName('程序鉴别材料.docx');
    const outputPath = await window.softreg.generateDocx({
      outputDir: outputDirInput.value,
      fileName
    });
    await new Promise(resolve => setTimeout(resolve, 2500));
    setStatus(`成功: DOCX 已保存至磁盘`);
    updateLights('success');
  } catch (err) {
    setStatus(`错误: DOCX 生成失败 // ${String(err.message || err).toU
pperCase()}`);
    updateLights('error');
  } finally {
    stopButtonEffect(btnExportDocx);
  }
}
async function exportPdf() {
  if (!validateInputs()) return;
  try {
    setStatus('状态: 正在导出 PDF 缓冲区...');
const fs = require('fs');
const path = require('path');
const {
  Document, Packer, Paragraph, TextRun, ImageRun, Header, Footer
,
```

```
  AlignmentType, PageOrientation, HeadingLevel, TableOfContents,
  LevelFormat, PageNumber
} = require('docx');
const outPath = path.join(__dirname, 'softreg-doc-v1.0-cn.docx')
;
const uiImagePath = path.join(__dirname, 'screenshot-app-ui.png'
);
const exportImagePath = path.join(__dirname, 'screenshot-export-
doc.png');
const makeCenter = (text) => new Paragraph({
  alignment: AlignmentType.CENTER,
  spacing: { after: 120 },
  children: [new TextRun({ text, size: 28 })]
});
const makeParagraph = (text) => new Paragraph({
  spacing: { after: 120 },
  children: [new TextRun({ text, size: 24 })]
});
const makeBullet = (text) => new Paragraph({
  numbering: { reference: 'bullet-list', level: 0 },
  children: [new TextRun({ text, size: 24 })]
});
const makeStep = (text) => new Paragraph({
  numbering: { reference: 'step-list', level: 0 },
  children: [new TextRun({ text, size: 24 })]
});
const scaleToWidth = (w, h, targetW) => {
  const ratio = targetW / w;
  return { width: Math.round(w * ratio), height: Math.round(h *
ratio) };
};
const addImageBlock = (title, imagePath, size, altText) => {
  const data = fs.readFileSync(imagePath);
  return [
    new Paragraph({
      spacing: { after: 120 },
      children: [new TextRun({ text: title, bold: true, size: 24
 })]
    }),
    new Paragraph({
      alignment: AlignmentType.CENTER,
      spacing: { after: 120 },
      children: [new ImageRun({
        type: 'png',
        data,
        transformation: { width: size.width, height: size.height
, rotation: 0 },
        altText: { title: altText, description: altText, name: a
ltText }
      })]
```

```
    })
  ];
};
const uiSize = scaleToWidth(939, 1547, 420);
const exportSize = scaleToWidth(927, 1291, 420);
const doc = new Document({
  styles: {
    default: { document: { run: { font: 'SimSun', size: 24 } } }
,
    paragraphStyles: [
      { id: 'Title', name: 'Title', basedOn: 'Normal',
        run: { size: 56, bold: true, color: '000000', font: 'Mic
rosoft YaHei' },
        paragraph: { spacing: { before: 240, after: 120 }, align
ment: AlignmentType.CENTER } },
      { id: 'Heading1', name: 'Heading 1', basedOn: 'Normal', ne
xt: 'Normal', quickFormat: true,
        run: { size: 32, bold: true, color: '000000', font: 'Mic
rosoft YaHei' },
        paragraph: { spacing: { before: 240, after: 180 }, outli
neLevel: 0 } },
      { id: 'Heading2', name: 'Heading 2', basedOn: 'Normal', ne
xt: 'Normal', quickFormat: true,
        run: { size: 28, bold: true, color: '000000', font: 'Mic
rosoft YaHei' },
        paragraph: { spacing: { before: 180, after: 120 }, outli
neLevel: 1 } }
    ]
  },
  numbering: {
    config: [
      {
        reference: 'bullet-list',
        levels: [{
          level: 0,
          format: LevelFormat.BULLET,
          text: '\u2022',
          alignment: AlignmentType.LEFT,
          style: { paragraph: { indent: { left: 720, hanging: 36
0 } } }
        }]
      },
      {
        reference: 'step-list',
        levels: [{
          level: 0,
          format: LevelFormat.DECIMAL,
          text: '%1.',
          alignment: AlignmentType.LEFT,
          style: { paragraph: { indent: { left: 720, hanging: 36
```

```
0 } } }
        }]
      }
    ]
  },
  sections: [{
    properties: {
      page: {
        margin: { top: 1440, right: 1440, bottom: 1440, left: 14
40 },
        size: { orientation: PageOrientation.PORTRAIT },
        pageNumbers: { start: 1, formatType: 'decimal' }
      }
    },
    headers: {
      default: new Header({
        children: [new Paragraph({
          alignment: AlignmentType.RIGHT,
          children: [new TextRun({ text: '软著鉴别材料整理器软件 v1.0 说明
书',
size: 20 })]
        })]
      })
    },
    footers: {
      default: new Footer({
        children: [new Paragraph({
          alignment: AlignmentType.CENTER,
          children: [new TextRun({ children: [PageNumber.CURRENT
] })]
        })]
      })
    },
    children: [
      new Paragraph({ heading: HeadingLevel.TITLE, children: [ne
w TextRun('软件说明书')] }),
      makeCenter('软件名称：软著鉴别材料整理器软件 v1.0'),
      makeCenter(`编制日期：${new Date().toISOString().slice(0, 10)}`
),
      makeCenter('适用范围：软著鉴别材料（软件说明与鉴别材料整理）'),
      new Paragraph({ children: [new TextRun({ text: '', size: 2
4 })], pageBreakBefore: true }),
      new TableOfContents('目录', { hyperlink: true, headingStyleR
ange: '1-3' }),
      new Paragraph({ children: [new TextRun({ text: '', size: 2
4 })], pageBreakBefore: true }),
      new Paragraph({ heading: HeadingLevel.HEADING_1, children:
 [new TextRun('1 软件概述')] }),
      makeParagraph('软著鉴别材料整理器软件 v1.0 用于自动化整理软著登记所需
的说明与鉴别材料，支持对软件信
息、功能模块、界面截图、生成文档等内容进行统一管理与输出。'),
```

```
950          makeParagraph('软件定位：面向需要提交软著材料的开发者、代理机构或企业，通过规范化模板提升材料完整性与
```

```
  一致性。'),
        makeParagraph('主要特点：'),
        makeBullet('统一模板：内置说明书模板与结构化字段，确保格式规范。'),
        makeBullet('材料集中管理：截图、功能描述、版本信息集中归档。'),
        makeBullet('一键生成：自动输出 Word 文档，减少人工整理成本。'),
        makeBullet('可追溯：生成记录与版本信息可回溯，便于复用。'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
   [new TextRun('2 运行环境与部署')] }),
        makeParagraph('支持操作系统：Windows 10/11。'),
        makeParagraph('运行依赖：本地安装 Microsoft Word 或兼容的文档查看器
（用于查看生成的 doc
  x）。'),
        makeParagraph('部署方式：应用安装包本地安装，无需联网。'),
        makeParagraph('启动方式：双击应用图标启动，进入主界面。'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
   [new TextRun('3 系统架构与模块')] }),
        makeParagraph('系统采用模块化结构，核心模块如下：'),
        makeBullet('资料管理模块：录入软件名称、版本、用途、开发环境等基础信
息。'),
        makeBullet('截图管理模块：导入界面截图、输出示例图，支持排序与说明。
'),
        makeBullet('说明书生成模块：按模板生成包含封面、目录、正文的说明文档。
'),
        makeBullet('导出与校验模块：导出 Word 文档并校验完整性。'),
        makeBullet('设置与日志模块：保存用户配置与生成记录。'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
   [new TextRun('4 功能说明')] }),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
   [new TextRun('4.1 资料采集与整理')] }),
        makeParagraph('提供字段化信息录入，包括软件名称、版本号、运行环境、功
能描述、适用范围等。'),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
   [new TextRun('4.2 界面截图管理')] }),
        makeParagraph('支持导入界面截图与输出示例图，自动归类并生成插图说明。
'),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
   [new TextRun('4.3 说明书自动生成')] }),
        makeParagraph('按软著登记要求生成包含封面、目录、正文与截图的 Word 说
明文档。'),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
   [new TextRun('4.4 导出与打包')] }),
        makeParagraph('支持一键导出 docx 文件，便于后续打印或提交。'),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
   [new TextRun('4.5 日志与版本信息')] }),
        makeParagraph('记录每次生成的时间、版本和素材变更，便于追溯。'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
   [new TextRun('5 操作流程')] }),
        makeParagraph('标准操作流程如下：'),
        makeStep('启动软件并进入主界面。'),
        makeStep('录入软件基础信息与功能说明。'),
        makeStep('导入界面截图与输出示例截图。'),
```

```
        makeStep('选择模板并生成说明书。'),
        makeStep('导出 docx 文档并进行检查。'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
  [new TextRun('6 数据结构与存储')] }),
        makeParagraph('软件在本地保存项目数据，包含基础信息、截图素材与生成结
果。数据不上传网络，默认保存于用
  户工作目录。'),
```

```
        makeParagraph('数据结构包括：项目信息、截图列表、生成记录、导出文档。
'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
  [new TextRun('7 安全与权限')] }),
        makeParagraph('软件为本地离线工具，默认不进行网络通信。'),
        makeParagraph('数据仅保存在本地，遵循操作系统文件权限控制。'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
  [new TextRun('8 兼容性与限制')] }),
        makeParagraph('输出文档格式为 docx，需在 Microsoft Word 或兼容软件中
打开。'),
        makeParagraph('部分排版效果在不同版本的 Word 中可能存在细微差异。'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
  [new TextRun('9 运行与维护')] }),
        makeParagraph('建议定期备份项目数据与导出文档。'),
        makeParagraph('版本升级时可保留旧版本说明书以便比对。'),
        new Paragraph({ heading: HeadingLevel.HEADING_1, children:
  [new TextRun('10 附录：截图')] }),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
  [new TextRun('10.1 软件界面截图')] }),
        ...addImageBlock('图 1 软件主界面（UI 截图）', uiImagePath, uiSize,
'软
  件主界面截图'),
        new Paragraph({ heading: HeadingLevel.HEADING_2, children:
  [new TextRun('10.2 导出文档示例')] }),
        ...addImageBlock('图 2 导出文档示例', exportImagePath, exportSize,
  '导出文档示例截图')
      ]
    }]
  });
  Packer.toBuffer(doc).then((buffer) => {
    fs.writeFileSync(outPath, buffer);
    console.log(`Generated: ${outPath}`);
  }).catch((err) => {
    console.error(err);
    process.exit(1);
  });
  const { app, BrowserWindow, dialog, ipcMain, shell } = require('
  electron');
  const path = require('path');
  const fs = require('fs');
  const { scanProject } = require('./shared/scanner');
  const { scanProjectForAI } = require('./shared/aiScanner');
  const os = require('os');
  const { writePdfFromDocx } = require('./shared/exportPdf');
  const { writeDocx } = require('./shared/exportDocx');
  const { writeManualDocx } = require('./shared/exportManualDocx')
  ;
  const Store = require('electron-store');
  const { pathToFileURL } = require('url');
  const { autoUpdater } = require('electron-updater');
  const store = new Store();
```

```
let mainWindow;
let lastScan = null;
```

```
function getWindowIcon() {
  const baseDir = path.join(__dirname, '..', 'build');
  if (process.platform === 'win32') {
    return path.join(baseDir, 'icon.ico');
  }
  return path.join(baseDir, 'icon.png');
}
function setupAutoUpdate() {
  if (!app.isPackaged) return;
  autoUpdater.autoDownload = true;
  autoUpdater.autoInstallOnAppQuit = true;
  autoUpdater.on('error', async (error) => {
    try {
      await dialog.showMessageBox(mainWindow, {
        type: 'error',
        title: '更新失败',
        message: '检查更新时发生错误',
        detail: String(error?.message || error || '')
      });
    } catch {}
  });
  autoUpdater.on('update-available', async () => {
    try {
      await dialog.showMessageBox(mainWindow, {
        type: 'info',
        title: '发现新版本',
        message: '发现新版本，正在后台下载…',
        buttons: ['知道了'],
        defaultId: 0
      });
    } catch {}
  });
  autoUpdater.on('update-downloaded', async () => {
    try {
      const result = await dialog.showMessageBox(mainWindow, {
        type: 'info',
        title: '更新已下载',
        message: '新版本已下载完成，是否立即重启并安装？',
        buttons: ['立即重启安装', '稍后'],
        defaultId: 0,
        cancelId: 1
      });
      if (result.response === 0) {
        autoUpdater.quitAndInstall();
      }
    } catch {}
  });
  autoUpdater.checkForUpdates().catch(() => {});
}
function createWindow() {
```

```
  mainWindow = new BrowserWindow({
    width: 1100,
    height: 780,
    title: '软著代码资料整理器',
    icon: getWindowIcon(),
    webPreferences: {
      preload: path.join(__dirname, 'preload.js'),
      contextIsolation: true,
      nodeIntegration: false
    }
  });
  mainWindow.loadFile(path.join(__dirname, 'renderer', 'index.ht
ml'));
}
app.whenReady().then(() => {
  createWindow();
  setupAutoUpdate();
  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) createWindow
();
  });
});
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') app.quit();
});
ipcMain.handle('select-project', async () => {
  const result = await dialog.showOpenDialog(mainWindow, {
    properties: ['openDirectory']
  });
  if (result.canceled || result.filePaths.length === 0) return n
ull;
  return result.filePaths[0];
});
ipcMain.handle('select-output-dir', async () => {
  const result = await dialog.showOpenDialog(mainWindow, {
    properties: ['openDirectory']
  });
  if (result.canceled || result.filePaths.length === 0) return n
ull;
  return result.filePaths[0];
});
ipcMain.handle('select-ai-screenshots', async () => {
  const result = await dialog.showOpenDialog(mainWindow, {
    properties: ['openFile', 'multiSelections'],
    filters: [
      { name: 'Images', extensions: ['png', 'jpg', 'jpeg', 'webp
', 'bmp'] }
    ]
  });
  if (result.canceled || result.filePaths.length === 0) return [
```

```
];
  return result.filePaths.map(filePath => ({
    path: filePath,
    url: pathToFileURL(filePath).href,
    name: path.basename(filePath)
  }));
});
ipcMain.handle('scan-project', async (_event, options) => {
  const scan = scanProject(options.projectPath);
  if (scan.totalPages === 0) {
    throw new Error('未找到可用源码文件。');
  }
  lastScan = {
    ...scan,
    headerText: options.headerText,
    pagesToExport: scan.pagesToExport
  };
  return {
    projectPath: scan.projectPath,
    fileCount: scan.fileCount,
    effectiveLines: scan.effectiveLines,
    totalPages: scan.totalPages,
    selectedPages: scan.pagesToExport.length
  };
});
ipcMain.handle('generate-docx', async (_event, options) => {
  if (!lastScan) throw new Error('请先扫描项目。');
  const outputPath = path.join(options.outputDir, options.fileNa
me);
  await writeDocx(outputPath, lastScan.pagesToExport, lastScan.h
eaderText);
  return outputPath;
});
ipcMain.handle('generate-pdf', async (_event, options) => {
  if (!lastScan) throw new Error('请先扫描项目。');
  const outputPath = path.join(options.outputDir, options.fileNa
me);
  const tempDir = fs.mkdtempSync(path.join(os.tmpdir(), 'softreg
-'));
  const tempDocx = path.join(tempDir, 'program-material.docx');
  await writeDocx(tempDocx, lastScan.pagesToExport, lastScan.hea
derText);
  writePdfFromDocx(tempDocx, outputPath);
  return outputPath;
});
ipcMain.handle('set-export-pages', async (_event, mode) => {
  if (!lastScan) throw new Error('请先扫描项目。');
  if (mode === 'all') {
    lastScan.pagesToExport = lastScan.pages;
  } else {
```

```
    lastScan.pagesToExport = lastScan.selectedPages;
  }
  return {
    pagesToExport: lastScan.pagesToExport.length
  };
});
ipcMain.handle('init-scan-state', async () => {
  lastScan = null;
});
ipcMain.handle('open-external', async (_event, url) => {
  await shell.openExternal(url);
});
ipcMain.handle('open-output-dir', async (_event, outputDir) => {
  const dir = String(outputDir || '').trim();
  if (!dir) return false;
  const result = await shell.openPath(dir);
  if (result) throw new Error(result);
  return true;
});
ipcMain.handle('ensure-output-dir', async (_event, outputDir) =>
 {
  if (!fs.existsSync(outputDir)) {
    fs.mkdirSync(outputDir, { recursive: true });
  }
  return true;
});
ipcMain.handle('ai:save-config', async (_event, config) => {
  store.set('aiConfig', config);
  return true;
});
ipcMain.handle('ai:get-config', async () => {
  return store.get('aiConfig') || {
    baseUrl: 'https://api.openai.com/v1',
    apiKey: '',
    model: 'gpt-3.5-turbo'
  };
});
ipcMain.handle('ai:generate-manual', async (_event, { projectPat
h, config, screenshots }) => {
  try {
    const context = scanProjectForAI(projectPath);
    const shotList = Array.isArray(screenshots) ? screenshots :
[];
    const shotText = shotList.length
      ? shotList
          .map((s, i) => {
            const note = String(s?.note || '').trim();
            const name = String(s?.name || '').trim();
            const label = note || name || String(s?.path || '').
trim();
```

```
            return `${i + 1}. ${label}`;
          })
          .join('\n')
      : '（无）';
    const prompt = `
你是一个专业的软件文档撰写专家。请根据以下项目上下文，撰写一份详细的《软件说明书》。
请严格使用 Markdown 标题（# / ## / ###）组织结构，标题不要只用纯编号列表代替。
## 截图列表（可用于插图）
${shotText}
当正文需要插入某张截图时，请在合适位置“单独输出一行”占位符，格式必须严格为：
[[SCREENSHOT:截图说明]]
其中“截图说明”必须与上面的截图列表某一项完全一致（优先使用用户填写的备注）。
## 1. 项目结构
\`\`\`
${context.structure}
\`\`\`
## 2. README 内容
${context.readme}
## 3. 依赖/配置信息
${context.packageJson}
## 4. 核心代码片段
${context.sourceSnippets.map(s => `--- ${s.path} ---\n${s.conten
t}`).join('\n\n')}
请输出 Markdown 格式，包含以下章节：
1. # 软件概述（根据 README 和代码推断）
2. # 功能列表（详细列出功能点）
3. # 技术特点（架构、语言、关键技术）
4. # 运行环境要求
5. # 使用说明（如有）
`;
    const baseUrl = config.baseUrl.replace(/\/$/, '');
    const response = await fetch(`${baseUrl}/chat/completions`,
{
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${config.apiKey}`
      },
      body: JSON.stringify({
        model: config.model,
        messages: [
          { role: 'system', content: '你是一个专业的软件技术文档专家，擅长编写软件说明书。
' },
          { role: 'user', content: prompt }
        ],
        temperature: 0.7
      })
    });
```

```
1299      if (!response.ok) {
1300        const errText = await response.text();
```

```
      throw new Error(`AI 请求失败 (${response.status}): ${errText}`
);
    }
    const data = await response.json();
    if (!data.choices || data.choices.length === 0) {
      throw new Error('AI 返回结果为空');
    }
    return data.choices[0].message.content;
  } catch (error) {
    console.error('AI Generation Error:', error);
    throw error;
  }
});
ipcMain.handle('ai:export-manual-docx', async (_event, options)
=> {
  const outputPath = path.join(options.outputDir, options.fileNa
me);
  await writeManualDocx(outputPath, options);
  return outputPath;
});
ipcMain.handle('ai:export-manual-pdf', async (_event, options) =
> {
  const outputPath = path.join(options.outputDir, options.fileNa
me);
  const tempDir = fs.mkdtempSync(path.join(os.tmpdir(), 'softreg
-manual-'));
  const tempDocx = path.join(tempDir, 'manual.docx');
  await writeManualDocx(tempDocx, options);
  writePdfFromDocx(tempDocx, outputPath);
  return outputPath;
});
ipcMain.handle('save-file', async (_event, { content, defaultNam
e }) => {
  const result = await dialog.showSaveDialog(mainWindow, {
    defaultPath: defaultName,
    filters: [{ name: 'Markdown', extensions: ['md'] }]
  });
  if (!result.canceled && result.filePath) {
    fs.writeFileSync(result.filePath, content, 'utf8');
    return result.filePath;
  }
  return null;
});
const { contextBridge, ipcRenderer } = require('electron');
contextBridge.exposeInMainWorld('softreg', {
  selectProject: () => ipcRenderer.invoke('select-project'),
  selectOutputDir: () => ipcRenderer.invoke('select-output-dir')
,
  scanProject: (options) => ipcRenderer.invoke('scan-project', o
ptions),
```

```
  generateDocx: (options) => ipcRenderer.invoke('generate-docx',
 options),
  generatePdf: (options) => ipcRenderer.invoke('generate-pdf', o
ptions),
  setExportPages: (mode) => ipcRenderer.invoke('set-export-pages
', mode),
  initScanState: () => ipcRenderer.invoke('init-scan-state'),
  ensureOutputDir: (outputDir) => ipcRenderer.invoke('ensure-out
put-dir', outputDir),
  openExternal: (url) => ipcRenderer.invoke('open-external', url
),
  openOutputDir: (outputDir) => ipcRenderer.invoke('open-output-
dir', outputDir),
  ai: {
    saveConfig: (config) => ipcRenderer.invoke('ai:save-config',
 config),
    getConfig: () => ipcRenderer.invoke('ai:get-config'),
    generateManual: (projectPath, config, screenshots) => ipcRen
derer.invoke('ai:generate-manual', { projectPath, config, screen
shots }),
    exportManualDocx: (options) => ipcRenderer.invoke('ai:export
-manual-docx', options),
    exportManualPdf: (options) => ipcRenderer.invoke('ai:export-
manual-pdf', options)
  },
  selectAiScreenshots: () => ipcRenderer.invoke('select-ai-scree
nshots'),
  saveFile: (content, defaultName) => ipcRenderer.invoke('save-f
ile', { content, defaultName })
});
const projectPathInput = document.getElementById('projectPath');
const outputDirInput = document.getElementById('outputDir');
const softwareNameInput = document.getElementById('softwareName'
);
const softwareVersionInput = document.getElementById('softwareVe
rsion');
const statFiles = document.getElementById('statFiles');
const statLines = document.getElementById('statLines');
const statPages = document.getElementById('statPages');
const statExport = document.getElementById('statExport');
const statExportLines = document.getElementById('statExportLines
');
const statusEl = document.getElementById('status');
const btnSelectProject = document.getElementById('btnSelectProje
ct');
const btnSelectOutput = document.getElementById('btnSelectOutput
');
const btnScan = document.getElementById('btnScan');
const btnExportDocx = document.getElementById('btnExportDocx');
const btnExportPdf = document.getElementById('btnExportPdf');
```

```
const btnExportBoth = document.getElementById('btnExportBoth');
const lightRed = document.querySelector('.light.red');
const lightAmber = document.querySelector('.light.amber');
const lightGreen = document.querySelector('.light.green');
function updateLights(state) {
  lightRed.classList.remove('active');
  lightAmber.classList.remove('active');
  lightGreen.classList.remove('active');
  switch (state) {
    case 'idle':
      lightRed.classList.add('active');
      break;
    case 'busy':
      lightAmber.classList.add('active');
      break;
    case 'success':
      lightGreen.classList.add('active');
      break;
    case 'error':
      lightRed.classList.add('active');
      break;
  }
}
function formatNum(num, length) {
  return String(num).padStart(length, '0');
}
function getSoftwareType() {
  const selected = document.querySelector('input[name="softwareT
ype"]:checked');
  return selected ? selected.value : 'software';
}
function buildHeaderText() {
  const name = softwareNameInput.value.trim();
  const version = softwareVersionInput.value.trim();
  const type = getSoftwareType() === 'game' ? '游戏软件' : '软件';
  if (!name || !version) return '';
  return `${name}${type}V${version}`;
}
function setStatus(message) {
  statusEl.textContent = message.toUpperCase();
  console.log(`[系统] ${message}`);
}
function validateInputs() {
  if (!projectPathInput.value) {
    setStatus('错误: 缺失项目路径');
    updateLights('error');
    return false;
  }
  if (!softwareNameInput.value.trim()) {
    setStatus('错误: 需填写软件名称');
```

```
      updateLights('error');
      return false;
    }
    if (!softwareVersionInput.value.trim()) {
      setStatus('错误: 需填写版本号');
      updateLights('error');
      return false;
    }
    if (!outputDirInput.value) {
      setStatus('错误: 缺失输出路径');
      updateLights('error');
      return false;
    }
    return true;
  }
  async function handleSelectProject() {
    const selected = await window.softreg.selectProject();
    if (selected) projectPathInput.value = selected;
  }
  async function handleSelectOutput() {
    const selected = await window.softreg.selectOutputDir();
    if (selected) outputDirInput.value = selected;
  }
  const scanOverlay = document.getElementById('scanOverlay');
  let audioCtx = null;
  function playScanSound() {
    if (!audioCtx) audioCtx = new (window.AudioContext || window.w
  ebkitAudioContext)();
    if (audioCtx.state === 'suspended') audioCtx.resume();
    const now = audioCtx.currentTime;
    const duration = 2.5; // 与填充动画时长一致
    const osc1 = audioCtx.createOscillator();
    const osc2 = audioCtx.createOscillator(); // 副振荡器 (正弦波 - 提供底
蕴)
    const filter = audioCtx.createBiquadFilter();
    const gainNode = audioCtx.createGain();
    osc1.type = 'sawtooth';
    osc1.frequency.setValueAtTime(80, now);
    osc1.frequency.exponentialRampToValueAtTime(450, now + duratio
  n);
    osc2.type = 'sine';
    osc2.frequency.setValueAtTime(40, now);
    osc2.frequency.exponentialRampToValueAtTime(120, now + duratio
  n);
    filter.type = 'lowpass';
    filter.Q.value = 15; // 高共振
    filter.frequency.setValueAtTime(200, now);
    filter.frequency.exponentialRampToValueAtTime(3000, now + dura
  tion);
    gainNode.gain.setValueAtTime(0, now);
    gainNode.gain.linearRampToValueAtTime(0.08, now + 0.1); // 快速开
```

```
启
  gainNode.gain.exponentialRampToValueAtTime(0.001, now + durati
on); // 逐渐消失
  osc1.connect(filter);
  osc2.connect(filter);
  filter.connect(gainNode);
  gainNode.connect(audioCtx.destination);
  osc1.start(now);
  osc2.start(now);
  osc1.stop(now + duration);
  osc2.stop(now + duration);
}
function startButtonEffect(btn) {
  btn.classList.add('processing');
  playScanSound();
}
function stopButtonEffect(btn) {
  btn.classList.remove('processing');
}
async function handleScan() {
  if (!projectPathInput.value) {
    setStatus('错误: 请先选择项目目录');
    updateLights('error');
    return;
  }
  const headerText = buildHeaderText();
  if (!headerText) {
    setStatus('错误: 名称和版本号缺失');
    updateLights('error');
    return;
  }
  try {
    setStatus('状态: 正在扫描项目文件...');
    updateLights('busy');
    startButtonEffect(btnScan);
    await window.softreg.initScanState();
    const result = await window.softreg.scanProject({
      projectPath: projectPathInput.value,
      headerText
    });
    await new Promise(resolve => setTimeout(resolve, 2500));
    statFiles.textContent = formatNum(result.fileCount, 4);
    statLines.textContent = formatNum(result.effectiveLines, 6);
    statPages.textContent = formatNum(result.totalPages, 3);
    statExport.textContent = formatNum(result.selectedPages, 3);
    statExportLines.textContent = formatNum(result.selectedPages
 * 50, 4);
    const truncationNote = result.totalPages > 60 ? ' (已截断至 3000
 行)' : '';
    setStatus(`状态: 扫描完成${truncationNote}. 准备就绪.`);
```

```
      updateLights('success');
    } catch (err) {
      setStatus(`错误: 扫描失败 // ${String(err.message || err).toUpperC
ase()}`);
      updateLights('error');
    } finally {
      stopButtonEffect(btnScan);
    }
}
function buildFileName(suffix) {
  const headerText = buildHeaderText();
  return `${headerText}_${suffix}`;
}
async function exportDocx() {
  if (!validateInputs()) return;
  try {
    setStatus('状态: 正在生成 DOCX 数据流...');
    updateLights('busy');
    startButtonEffect(btnExportDocx);
    await window.softreg.ensureOutputDir(outputDirInput.value);
    const fileName = buildFileName('程序鉴别材料.docx');
    const outputPath = await window.softreg.generateDocx({
      outputDir: outputDirInput.value,
      fileName
    });
    await new Promise(resolve => setTimeout(resolve, 2500));
    setStatus(`成功: DOCX 已保存至磁盘`);
    updateLights('success');
  } catch (err) {
    setStatus(`错误: DOCX 生成失败 // ${String(err.message || err).toU
pperCase()}`);
    updateLights('error');
  } finally {
    stopButtonEffect(btnExportDocx);
  }
}
async function exportPdf() {
  if (!validateInputs()) return;
  try {
    setStatus('状态: 正在导出 PDF 缓冲区...');
    updateLights('busy');
    startButtonEffect(btnExportPdf);
    await window.softreg.ensureOutputDir(outputDirInput.value);
    const fileName = buildFileName('程序鉴别材料.pdf');
    const outputPath = await window.softreg.generatePdf({
      outputDir: outputDirInput.value,
      fileName
    });
    await new Promise(resolve => setTimeout(resolve, 2500));
    setStatus(`成功: PDF 导出完成`);
```

```
      updateLights('success');
    } catch (err) {
      setStatus(`错误: PDF 导出失败 // ${String(err.message || err).toUp
perCase()}`);
      updateLights('error');
    } finally {
      stopButtonEffect(btnExportPdf);
    }
}
async function exportBoth() {
  if (!validateInputs()) return;
  await exportDocx();
  await exportPdf();
}
updateLights('idle');
btnSelectProject.addEventListener('click', handleSelectProject);
btnSelectOutput.addEventListener('click', handleSelectOutput);
btnScan.addEventListener('click', handleScan);
btnExportDocx.addEventListener('click', exportDocx);
btnExportPdf.addEventListener('click', exportPdf);
btnExportBoth.addEventListener('click', exportBoth);
document.getElementById('linkBilibili').addEventListener('click'
, (e) => {
  e.preventDefault();
  window.softreg.openExternal('https://space.bilibili.com/309927
2');
});
document.getElementById('linkGithub').addEventListener('click',
(e) => {
  e.preventDefault();
  window.softreg.openExternal('https://github.com/itxys/softwork
-code-organizer');
});
document.getElementById('linkCopyright').addEventListener('click
', (e) => {
  e.preventDefault();
  window.softreg.openExternal('https://www.ccopyright.com.cn/');
});
const btnAiConfig = document.getElementById('btnAiConfig');
const btnAiManual = document.getElementById('btnAiManual');
const btnAiUploadScreenshots = document.getElementById('btnAiUpl
oadScreenshots');
const aiScreenshotList = document.getElementById('aiScreenshotLi
st');
const btnExportAiDocx = document.getElementById('btnExportAiDocx
');
const btnExportAiPdf = document.getElementById('btnExportAiPdf')
;
const btnExportAiBoth = document.getElementById('btnExportAiBoth
');
```

```
const btnOpenOutputDir = document.getElementById('btnOpenOutputDir');
const btnOpenOutputDirAi = document.getElementById('btnOpenOutputDirAi');
const aiConfigModal = document.getElementById('aiConfigModal');
const aiResultModal = document.getElementById('aiResultModal');
const aiProviderPreset = document.getElementById('aiProviderPreset');
const aiBaseUrl = document.getElementById('aiBaseUrl');
const aiApiKey = document.getElementById('aiApiKey');
const aiModelPreset = document.getElementById('aiModelPreset');
const aiModel = document.getElementById('aiModel');
const btnTestAiConnection = document.getElementById('btnTestAiConnection');
const aiConnectionStatus = document.getElementById('aiConnectionStatus');
const aiTestResult = document.getElementById('aiTestResult');
const aiConfigToast = document.getElementById('aiConfigToast');
const aiToastTitle = document.getElementById('aiToastTitle');
const aiToastMsg = document.getElementById('aiToastMsg');
const btnSaveAiConfig = document.getElementById('btnSaveAiConfig');
function showModalToast(title, msg, type = 'info') {
  if (!aiConfigToast) return;
  aiToastTitle.textContent = title;
  aiToastMsg.textContent = msg;
  aiConfigToast.className = `modal-toast active ${type}`;
  setTimeout(() => {
    aiConfigToast.classList.remove('active');
  }, 3000);
}
const btnCloseAiConfig = document.getElementById('btnCloseAiConfig');
const btnSaveAiManual = document.getElementById('btnSaveAiManual');
const btnCloseAiResult = document.getElementById('btnCloseAiResult');
const aiResultText = document.getElementById('aiResultText');
let aiManualMarkdown = '';
let aiScreenshots = [];
let aiGeneratingManual = false;
const AI_PROVIDER_PRESETS = [
  {
    id: 'custom',
    name: '自定义（兼容 OpenAI 接口）',
    baseUrl: '',
    models: []
  },
  {
    id: 'openai',
```

```
    name: 'OpenAI 官方',
    baseUrl: 'https://api.openai.com/v1',
    models: ['gpt-5.2', 'gpt-5.1', 'o3-mini', 'o3-pro', 'gpt-4.1
', 'gpt-4o']
  },
  {
    id: 'siliconflow',
    name: '硅基流动（SiliconFlow）',
    baseUrl: 'https://api.siliconflow.cn/v1',
    models: [
      'deepseek-ai/DeepSeek-R1',
      'deepseek-ai/DeepSeek-V3',
      'Qwen/Qwen2.5-72B-Instruct',
      'Qwen/Qwen2.5-Coder-32B-Instruct',
      'TeleAI/TeleChat2'
    ]
  },
  {
    id: 'deepseek',
    name: 'DeepSeek 官方',
    baseUrl: 'https://api.deepseek.com/v1',
    models: ['deepseek-reasoner', 'deepseek-chat']
  },
  {
    id: 'openrouter',
    name: 'OpenRouter',
    baseUrl: 'https://openrouter.ai/api/v1',
    models: [
      'anthropic/claude-4.5-sonnet',
      'anthropic/claude-4.5-opus',
      'google/gemini-3-pro',
      'deepseek/deepseek-r1',
      'meta-llama/llama-4-70b-instruct'
    ]
  },
  {
    id: 'qwen',
    name: '阿里云 DashScope (Qwen)',
    baseUrl: 'https://dashscope.aliyuncs.com/compatible-mode/v1'
,
    models: ['qwen-max', 'qwen-plus', 'qwen-turbo', 'qwen-long']
  },
  {
    id: 'zhipu',
    name: '智谱 ZhipuAI (GLM)',
    baseUrl: 'https://open.bigmodel.cn/api/paas/v4',
    models: ['glm-4-plus', 'glm-4-air', 'glm-4-flash', 'glm-4-lo
ng']
  },
  {
```

```
    id: 'moonshot',
    name: 'Moonshot AI (Kimi)',
    baseUrl: 'https://api.moonshot.cn/v1',
    models: ['moonshot-v1-128k', 'moonshot-v1-32k', 'moonshot-v1
-8k']
  },
  {
    id: 'minimax',
    name: 'MiniMax',
    baseUrl: 'https://api.minimax.chat/v1',
    models: ['abab7-chat-preview', 'abab6.5s-chat', 'abab6.5-cha
t']
  },
  {
    id: 'doubao',
    name: '字节跳动 Doubao (火山引擎)',
    baseUrl: 'https://ark.cn-beijing.volces.com/api/v3',
    models: ['doubao-pro-32k', 'doubao-lite-32k', 'doubao-pro-12
8k']
  },
  {
    id: 'modelscope',
    name: 'ModelScope (魔搭)',
    baseUrl: 'https://api-inference.modelscope.cn/v1',
    models: ['Qwen/Qwen2.5-72B-Instruct', 'Qwen/Qwen2.5-7B-Instr
uct']
  },
  {
    id: 'claude',
    name: 'Claude (Official/Proxy)',
    baseUrl: 'https://api.anthropic.com/v1',
    models: ['claude-opus-4-5', 'claude-3-5-sonnet-20241022', 'c
laude-3-5-haiku-20241022']
  },
  {
    id: 'gemini',
    name: 'Google Gemini',
    baseUrl: 'https://generativelanguage.googleapis.com/v1beta/o
penai',
    models: ['gemini-3-pro-preview', 'gemini-3-flash-preview', '
gemini-2.0-flash-exp', 'gemini-1.5-pro', 'gemini-1.5-flash']
  }
];
let aiPresetsInitialized = false;
function getProviderPresetById(id) {
  return AI_PROVIDER_PRESETS.find(p => p.id === id) || AI_PROVID
ER_PRESETS[0];
}
function findProviderPresetIdByBaseUrl(baseUrl) {
  const normalized = String(baseUrl || '').trim().replace(/\/$/,
```

```
  '');
    const match = AI_PROVIDER_PRESETS.find(p => p.baseUrl && p.bas
eUrl.replace(/\/$/, '') === normalized);
    return match ? match.id : 'custom';
}
function setSelectOptions(selectEl, options) {
    selectEl.innerHTML = '';
    for (const opt of options) {
      const optionEl = document.createElement('option');
      optionEl.value = opt.value;
      optionEl.textContent = opt.label;
      selectEl.appendChild(optionEl);
    }
}
function populateProviderPresetOptions() {
    setSelectOptions(
      aiProviderPreset,
      AI_PROVIDER_PRESETS.map(preset => ({
        value: preset.id,
        label: preset.name
      }))
    );
}
function populateModelPresetOptions(providerId) {
    const provider = getProviderPresetById(providerId);
    const modelOptions = [{ value: 'custom', label: '自定义' }].conca
t(
      provider.models.map(m => ({ value: m, label: m }))
    );
    setSelectOptions(aiModelPreset, modelOptions);
}
function syncProviderPresetFromBaseUrl() {
    const providerId = findProviderPresetIdByBaseUrl(aiBaseUrl.val
ue);
    if (aiProviderPreset.value !== providerId) {
      aiProviderPreset.value = providerId;
      populateModelPresetOptions(providerId);
      syncModelPresetFromModelName();
    }
}
function syncModelPresetFromModelName() {
    const modelName = String(aiModel.value || '').trim();
    const optionExists = Array.from(aiModelPreset.options).some(o
=> o.value === modelName);
    aiModelPreset.value = optionExists ? modelName : 'custom';
}
function applyProviderPreset(providerId) {
    const provider = getProviderPresetById(providerId);
    populateModelPresetOptions(providerId);
    if (provider.baseUrl) {
```

```
    aiBaseUrl.value = provider.baseUrl;
  }
  if (!String(aiModel.value || '').trim() && provider.models.len
gth > 0) {
    aiModel.value = provider.models[0];
  }
  syncModelPresetFromModelName();
}
function initAiPresetControls() {
  if (aiPresetsInitialized) return;
  aiPresetsInitialized = true;
  populateProviderPresetOptions();
  populateModelPresetOptions('custom');
  aiProviderPreset.addEventListener('change', () => {
    applyProviderPreset(aiProviderPreset.value);
  });
  aiModelPreset.addEventListener('change', () => {
    if (aiModelPreset.value === 'custom') return;
    aiModel.value = aiModelPreset.value;
  });
  aiBaseUrl.addEventListener('input', () => {
    syncProviderPresetFromBaseUrl();
  });
  aiModel.addEventListener('input', () => {
    syncModelPresetFromModelName();
  });
}
function renderAiScreenshotList() {
  if (!aiScreenshotList) return;
  aiScreenshotList.innerHTML = '';
  if (!aiScreenshots.length) return;
  let dragIndex = null;
  aiScreenshots.forEach((item, index) => {
    const row = document.createElement('div');
    row.className = 'screenshot-item';
    row.draggable = true;
    row.dataset.index = String(index);
    const img = document.createElement('img');
    img.className = 'screenshot-thumb';
    img.alt = item.name || `截图 ${index + 1}`;
    img.src = item.url;
    const name = document.createElement('div');
    name.className = 'screenshot-name';
    name.textContent = `${index + 1}. ${item.name || item.path |
| ''}`;
    const noteInput = document.createElement('input');
    noteInput.type = 'text';
    noteInput.className = 'screenshot-note';
    noteInput.placeholder = '备注：例如 首页截图 / 目录截图';
    noteInput.value = String(item.note || '');
```

```
    noteInput.draggable = false;
    noteInput.addEventListener('input', () => {
      const idx = Number(row.dataset.index);
      if (Number.isNaN(idx) || !aiScreenshots[idx]) return;
      aiScreenshots[idx].note = noteInput.value;
    });
    row.appendChild(img);
    row.appendChild(name);
    row.appendChild(noteInput);
    row.addEventListener('dragstart', (e) => {
      if (e.target && e.target.closest && e.target.closest('inpu
t')) {
        e.preventDefault();
        return;
      }
      dragIndex = index;
      row.classList.add('dragging');
      try {
        e.dataTransfer.setData('text/plain', String(index));
        e.dataTransfer.effectAllowed = 'move';
      } catch (_err) {}
    });
    row.addEventListener('dragend', () => {
      row.classList.remove('dragging');
    });
    row.addEventListener('dragover', (e) => {
      e.preventDefault();
      try {
        e.dataTransfer.dropEffect = 'move';
      } catch (_err) {}
    });
    row.addEventListener('drop', (e) => {
      e.preventDefault();
      let fromIndex = dragIndex;
      const toIndex = Number(row.dataset.index);
      try {
        const raw = e.dataTransfer.getData('text/plain');
        if (raw !== '') fromIndex = Number(raw);
      } catch (_err) {}
      if (Number.isNaN(fromIndex) || Number.isNaN(toIndex) || fr
omIndex === toIndex) return;
      const next = aiScreenshots.slice();
      const [moved] = next.splice(fromIndex, 1);
      next.splice(toIndex, 0, moved);
      aiScreenshots = next;
      renderAiScreenshotList();
    });
    aiScreenshotList.appendChild(row);
  });
}
```

```
async function loadAiConfig() {
  initAiPresetControls();
  const config = await window.softreg.ai.getConfig();
  aiBaseUrl.value = config.baseUrl;
  aiApiKey.value = config.apiKey;
  aiModel.value = config.model;
  const providerId = config.providerId || findProviderPresetIdBy
BaseUrl(config.baseUrl);
  aiProviderPreset.value = providerId;
  populateModelPresetOptions(providerId);
  syncModelPresetFromModelName();
}
btnAiConfig.addEventListener('click', async () => {
  initAiPresetControls();
  await loadAiConfig();
  if (aiTestResult) aiTestResult.textContent = '';
  if (aiConnectionStatus) aiConnectionStatus.className = 'status
-dot-outer';
  aiConfigModal.classList.remove('hidden');
});
btnCloseAiConfig.addEventListener('click', () => {
  aiConfigModal.classList.add('hidden');
  if (aiTestResult) aiTestResult.textContent = '';
  if (aiConnectionStatus) aiConnectionStatus.className = 'status
-dot-outer';
});
if (btnTestAiConnection) {
  btnTestAiConnection.addEventListener('click', async () => {
    const baseUrl = aiBaseUrl.value.trim();
    const apiKey = aiApiKey.value.trim();
    const model = aiModel.value.trim();
    if (!baseUrl || !apiKey || !model) {
      if (aiTestResult) {
        aiTestResult.textContent = '请先填写完整的配置信息';
        aiTestResult.className = 'test-result-text error';
      }
      showModalToast('配置不完整', '请填写 API 地址、密钥和模型名称',
'error');
      return;
    }
    if (aiConnectionStatus) {
      aiConnectionStatus.className = 'status-dot-outer';
      aiConnectionStatus.title = '正在测试连接... / Testing Connection
...';
    }
    if (aiTestResult) {
      aiTestResult.textContent = '正在连接服务器...';
      aiTestResult.className = 'test-result-text';
    }
    const originalText = btnTestAiConnection.textContent;
    btnTestAiConnection.disabled = true;
```

```
    btnTestAiConnection.textContent = '测试中...';
    try {
      let cleanBaseUrl = baseUrl.replace(/\/+$/, '');
      const url = `${cleanBaseUrl}/chat/completions`;
      const controller = new AbortController();
      const timeoutId = setTimeout(() => controller.abort(), 150
00); // 15s timeout
      const response = await fetch(url, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': `Bearer ${apiKey}`
        },
        body: JSON.stringify({
          model: model,
          messages: [{ role: 'user', content: 'Hi' }],
          max_tokens: 1
        }),
        signal: controller.signal
      });
      clearTimeout(timeoutId);
      if (response.ok) {
        if (aiConnectionStatus) {
          aiConnectionStatus.classList.add('success');
          aiConnectionStatus.title = '连接成功 / Connection Successf
ul';
        }
        if (aiTestResult) {
          aiTestResult.textContent = '连接测试成功！接口响应正常。';
          aiTestResult.className = 'test-result-text success';
        }
        showModalToast('测试成功', 'AI 接口连接正常，响应成功！',
'success');
      } else {
        const errText = await response.text();
        console.error('AI Test Error:', response.status, errText
);
        if (aiConnectionStatus) {
          aiConnectionStatus.classList.add('error');
          aiConnectionStatus.title = `连接失败 / Connection Failed:
${response.status}`;
        }
        if (aiTestResult) {
          aiTestResult.textContent = `失败: ${response.status} - $
{errText.substring(0, 30)}...`;
          aiTestResult.className = 'test-result-text error';
        }
        showModalToast('测试失败', `状态码: ${response.status}\n 原因:
${e
rrText.substring(0, 50)}`, 'error');
      }
```

```
2050      } catch (error) {
```

```
      console.error('AI Test Exception:', error);
      if (aiConnectionStatus) {
        aiConnectionStatus.classList.add('error');
        aiConnectionStatus.title = `连接错误 / Connection Error`;
      }
      if (aiTestResult) {
        aiTestResult.textContent = `错误: ${error.message}`;
        aiTestResult.className = 'test-result-text error';
      }
      showModalToast('连接错误', error.message, 'error');
    } finally {
      btnTestAiConnection.disabled = false;
      btnTestAiConnection.textContent = originalText;
    }
  });
}
btnSaveAiConfig.addEventListener('click', async () => {
  const config = {
    baseUrl: aiBaseUrl.value.trim(),
    apiKey: aiApiKey.value.trim(),
    model: aiModel.value.trim()
  };
  config.providerId = aiProviderPreset.value;
  await window.softreg.ai.saveConfig(config);
  aiConfigModal.classList.add('hidden');
  if (aiTestResult) aiTestResult.textContent = '';
  if (aiConnectionStatus) aiConnectionStatus.className = 'status
-dot-outer';
  setStatus('状态: AI 配置已保存');
});
async function generateAiManualMarkdown() {
  if (aiGeneratingManual) return '';
  if (!projectPathInput.value) {
    setStatus('错误: 请先选择项目目录');
    updateLights('error');
    return '';
  }
  const config = await window.softreg.ai.getConfig();
  if (!config.apiKey) {
    setStatus('错误: 未配置 AI API KEY');
    updateLights('error');
    await loadAiConfig();
    aiConfigModal.classList.remove('hidden');
    return '';
  }
  try {
    aiGeneratingManual = true;
    setStatus('状态: AI 正在分析项目并撰写说明书...');
    updateLights('busy');
    startButtonEffect(btnAiManual);
```

```
2101      aiResultText.value = '正在分析项目结构...\n 正在读取关键文件...\n 正在
调用 AI 生成（可
2102  能需要 30-60 秒）...\n\n 请稍候...';
2103      aiResultModal.classList.remove('hidden');
2104      const manualMarkdown = await window.softreg.ai.generateManua
2105  l(
2106        projectPathInput.value,
2107        config,
2108        aiScreenshots.map(s => ({
2109          path: s.path,
2110          name: s.name,
2111          note: String(s.note || '').trim()
2112        }))
2113      );
2114      aiManualMarkdown = manualMarkdown;
2115      aiResultText.value = manualMarkdown;
2116      setStatus('状态: 说明书生成完成');
2117      updateLights('success');
2118      return manualMarkdown;
2119    } catch (err) {
2120      setStatus(`错误: 生成失败 // ${String(err.message || err).toUpperC
2121  ase()}`);
2122      updateLights('error');
2123      aiResultText.value = `生成失败:\n${String(err.message || err)}`;
2124      return '';
2125    } finally {
2126      aiGeneratingManual = false;
2127      stopButtonEffect(btnAiManual);
2128    }
2129  }
2130  btnAiManual.addEventListener('click', async () => {
2131    await generateAiManualMarkdown();
2132  });
2133  btnSaveAiManual.addEventListener('click', async () => {
2134    await generateAiManualMarkdown();
2135  });
2136  btnCloseAiResult.addEventListener('click', () => {
2137    aiManualMarkdown = aiResultText.value;
2138    aiResultModal.classList.add('hidden');
2139    setStatus('状态: 已保存生成结果');
2140  });
2141  async function ensureAiManualReady() {
2142    if (String(aiManualMarkdown || '').trim()) return aiManualMark
2143  down;
2144    return await generateAiManualMarkdown();
2145  }
2146  async function exportAiDocx() {
2147    if (!validateInputs()) return;
2148    const markdown = await ensureAiManualReady();
2149    if (!String(markdown || '').trim()) return;
2150    try {
```

```
      setStatus('状态: 正在导出说明书 DOCX...');
      updateLights('busy');
      startButtonEffect(btnExportAiDocx);
      await window.softreg.ensureOutputDir(outputDirInput.value);
      const headerText = buildHeaderText();
      const fileName = `${headerText || '项目'}_软件说明书.docx`;
      await window.softreg.ai.exportManualDocx({
        outputDir: outputDirInput.value,
        fileName,
        markdown,
        screenshots: aiScreenshots.map(s => ({
          path: s.path,
          name: s.name,
          note: String(s.note || '').trim()
        })),
        softwareName: softwareNameInput.value.trim(),
        softwareVersion: softwareVersionInput.value.trim(),
        headerText
      });
      await new Promise(resolve => setTimeout(resolve, 800));
      setStatus('成功: 说明书 DOCX 已保存至磁盘');
      updateLights('success');
    } catch (err) {
      setStatus(`错误: 说明书 DOCX 导出失败 // ${String(err.message ||
err)
.toUpperCase()}`);
      updateLights('error');
    } finally {
      stopButtonEffect(btnExportAiDocx);
    }
  }
  async function exportAiPdf() {
    if (!validateInputs()) return;
    const markdown = await ensureAiManualReady();
    if (!String(markdown || '').trim()) return;
    try {
      setStatus('状态: 正在导出说明书 PDF...');
      updateLights('busy');
      startButtonEffect(btnExportAiPdf);
      await window.softreg.ensureOutputDir(outputDirInput.value);
      const headerText = buildHeaderText();
      const fileName = `${headerText || '项目'}_软件说明书.pdf`;
      await window.softreg.ai.exportManualPdf({
        outputDir: outputDirInput.value,
        fileName,
        markdown,
        screenshots: aiScreenshots.map(s => ({
          path: s.path,
          name: s.name,
          note: String(s.note || '').trim()
        })),
```

```
      softwareName: softwareNameInput.value.trim(),
      softwareVersion: softwareVersionInput.value.trim(),
      headerText
    });
    await new Promise(resolve => setTimeout(resolve, 800));
    setStatus('成功: 说明书 PDF 已保存至磁盘');
    updateLights('success');
  } catch (err) {
    setStatus(`错误: 说明书 PDF 导出失败 // ${String(err.message || err).
toUpperCase()}`);
    updateLights('error');
  } finally {
    stopButtonEffect(btnExportAiPdf);
  }
}
async function exportAiBoth() {
  if (!validateInputs()) return;
  await exportAiDocx();
  await exportAiPdf();
}
btnExportAiDocx.addEventListener('click', exportAiDocx);
btnExportAiPdf.addEventListener('click', exportAiPdf);
btnExportAiBoth.addEventListener('click', exportAiBoth);
function openSelectedOutputDir() {
  const out = String(outputDirInput.value || '').trim();
  if (!out) {
    setStatus('错误: 缺失输出路径');
    updateLights('error');
    return;
  }
  window.softreg.openOutputDir(out);
}
btnOpenOutputDir.addEventListener('click', openSelectedOutputDir
);
btnOpenOutputDirAi.addEventListener('click', openSelectedOutputD
ir);
btnAiUploadScreenshots.addEventListener('click', async () => {
  const items = await window.softreg.selectAiScreenshots();
  if (!Array.isArray(items) || items.length === 0) return;
  const existing = new Set(aiScreenshots.map(s => s.path));
  const merged = aiScreenshots.slice();
  for (const it of items) {
    if (!it || !it.path || existing.has(it.path)) continue;
    merged.push({ ...it, note: '' });
    existing.add(it.path);
  }
  aiScreenshots = merged;
  renderAiScreenshotList();
});
window.addEventListener('DOMContentLoaded', () => {
```

```
  aiConfigModal.classList.add('hidden');
  aiResultModal.classList.add('hidden');
  renderAiScreenshotList();
});
const fs = require('fs');
const path = require('path');
const EXCLUDED_DIRS = new Set([
  'node_modules', 'dist', 'build', 'out', '.git', 'vendor', 'bin
', 'obj',
  'target', '.idea', '.vscode', '.vs', '.svn', '.hg', '.turbo',
'.next',
  '.cache', '.angular', '.gradle'
]);
const SUPPORTED_EXTS = new Set([
  '.js', '.jsx', '.ts', '.tsx',
  '.py', '.java', '.cs', '.c', '.cpp', '.h', '.hpp',
  '.go', '.rs', '.php', '.swift', '.kt', '.json', '.md', '.txt'
]);
function isTextFile(filePath) {
  const ext = path.extname(filePath).toLowerCase();
  return SUPPORTED_EXTS.has(ext) || ext === '';
}
function getProjectStructure(root, depth = 0, maxDepth = 3) {
  if (depth > maxDepth) return '';
  let output = '';
  let entries;
  try {
    entries = fs.readdirSync(root, { withFileTypes: true });
  } catch (e) {
    return '';
  }
  entries.sort((a, b) => {
    if (a.isDirectory() && !b.isDirectory()) return -1;
    if (!a.isDirectory() && b.isDirectory()) return 1;
    return a.name.localeCompare(b.name);
  });
  const indent = '  '.repeat(depth);
  for (const entry of entries) {
    if (entry.name.startsWith('.')) continue; // Skip dotfiles
    if (entry.isDirectory()) {
      if (EXCLUDED_DIRS.has(entry.name)) {
        output += `${indent}├── ${entry.name}/\n`;
      } else {
        output += `${indent}├── ${entry.name}/\n`;
        output += getProjectStructure(path.join(root, entry.name
), depth + 1, maxDepth);
      }
    } else {
      output += `${indent}├── ${entry.name}\n`;
    }
```

```
  }
  return output;
}
function getFileContent(filePath, maxLines = 100) {
  try {
    const content = fs.readFileSync(filePath, 'utf8');
    const lines = content.split('\n');
    if (lines.length > maxLines) {
      return lines.slice(0, maxLines).join('\n') + '\n... (trunc
ated)';
    }
    return content;
  } catch (e) {
    return `Error reading file: ${e.message}`;
  }
}
function scanProjectForAI(projectPath) {
  const context = {
    structure: '',
    readme: '',
    packageJson: '',
    sourceSnippets: []
  };
  context.structure = getProjectStructure(projectPath);
  const readmeNames = ['README.md', 'readme.md', 'README.txt', '
readme.txt'];
  for (const name of readmeNames) {
    const p = path.join(projectPath, name);
    if (fs.existsSync(p)) {
      context.readme = getFileContent(p, 200); // Read up to 200
 lines of README
      break;
    }
  }
  const configFiles = ['package.json', 'requirements.txt', 'pom.
xml', 'go.mod', 'Cargo.toml'];
  for (const name of configFiles) {
    const p = path.join(projectPath, name);
    if (fs.existsSync(p)) {
      context.packageJson += `--- ${name} ---\n${getFileContent(
p, 100)}\n\n`;
    }
  }
  const snippets = [];
  let fileCount = 0;
  const MAX_FILES = 5;
  function walkAndPick(current) {
    if (fileCount >= MAX_FILES) return;
    let entries;
    try {
```

```
      entries = fs.readdirSync(current, { withFileTypes: true })
;
    } catch { return; }
    for (const entry of entries) {
      if (fileCount >= MAX_FILES) return;
      const fullPath = path.join(current, entry.name);
      if (entry.isDirectory()) {
        if (!EXCLUDED_DIRS.has(entry.name) && !entry.name.starts
With('.')) {
          walkAndPick(fullPath);
        }
      } else {
        const ext = path.extname(entry.name).toLowerCase();
        if (['.json', '.md', '.txt', '.lock'].includes(ext)) con
tinue;
        if (SUPPORTED_EXTS.has(ext)) {
          snippets.push({
            path: path.relative(projectPath, fullPath),
            content: getFileContent(fullPath, 50) // First 50 li
nes
          });
          fileCount++;
        }
      }
    }
  }
  const srcPath = path.join(projectPath, 'src');
  if (fs.existsSync(srcPath)) {
    walkAndPick(srcPath);
  }
  if (fileCount < MAX_FILES) {
    walkAndPick(projectPath);
  }
  context.sourceSnippets = snippets;
  return context;
}
module.exports = {
  scanProjectForAI
};
const fs = require('fs');
const {
  AlignmentType,
  Document,
  Footer,
  Header,
  LineRuleType,
  PageNumber,
  NumberFormat,
  Paragraph,
  Packer,
```

```
  TextRun,
  TabStopType,
  TabStopPosition
} = require('docx');
function lineParagraph(line, lineNumber, isFirstLineOfPage) {
  const text = line.length === 0 ? ' ' : line;
  const lineNumStr = String(lineNumber).padStart(5, ' ') + '  ';
  return new Paragraph({
    pageBreakBefore: isFirstLineOfPage,
    spacing: {
      before: 0,
      after: 0,
      line: 270,
      lineRule: LineRuleType.EXACT
    },
    children: [
      new TextRun({
        text: lineNumStr,
        font: 'Consolas',
        size: 21,
        color: '888888' // 浅灰色行号
      }),
      new TextRun({
        text,
        font: 'Consolas',
        size: 21,
        preserve: true
      })
    ]
  });
}
async function writeDocx(outputPath, pages, headerText) {
  const children = [];
  let globalLineNumber = 1;
  pages.forEach((pageLines, pageIndex) => {
    pageLines.forEach((line, lineIndex) => {
      const isFirstLineOfPage = (pageIndex > 0 && lineIndex ===
0);
      children.push(lineParagraph(line, globalLineNumber, isFirs
tLineOfPage));
      globalLineNumber++;
    });
  });
  const header = new Header({
    children: [
      new Paragraph({
        children: [new TextRun({ text: headerText, font: 'Micros
oft YaHei', size: 24 })],
        alignment: AlignmentType.LEFT,
        spacing: { before: 0, after: 0 }
```

```
      })
    ]
  });
  const footer = new Footer({
    children: [
      new Paragraph({
        alignment: AlignmentType.CENTER,
        children: [
          new TextRun({ text: "— 第 ", font: 'Microsoft YaHei', s
ize: 20 }),
          new TextRun({
            children: [PageNumber.CURRENT]
          }),
          new TextRun({ text: " 页 / 共 ", font: 'Microsoft YaHei'
, size: 20 }),
          new TextRun({
            children: [PageNumber.TOTAL_PAGES]
          }),
          new TextRun({ text: " 页 —", font: 'Microsoft YaHei', s
ize: 20 })
        ],
        spacing: { before: 0, after: 0 }
      })
    ]
  });
  const doc = new Document({
    sections: [
      {
        properties: {
          page: {
            margin: {
              top: 1440,
              bottom: 1440,
              left: 1440, // 不再需要为 Word 行号预留空间，因为使用手动行号
              right: 1440,
              header: 720,
              footer: 720
            },
            pageNumbers: {
              start: 1,
              formatType: NumberFormat.DECIMAL
            }
          }
        },
        headers: { default: header, first: header },
        footers: { default: footer, first: footer },
        children
      }
    ]
  });
```

```
  const buffer = await Packer.toBuffer(doc);
  fs.writeFileSync(outputPath, buffer);
}
module.exports = {
  writeDocx
};
const fs = require('fs');
const { nativeImage } = require('electron');
const path = require('path');
const {
  AlignmentType,
  Document,
  Footer,
  Header,
  HeadingLevel,
  ImageRun,
  NumberFormat,
  PageNumber,
  Paragraph,
  Packer,
  TableOfContents,
  TextRun
} = require('docx');
function normalizeScreenshotKey(value) {
  return String(value || '').trim().toLowerCase();
}
function buildScreenshotEntry(raw) {
  if (typeof raw === 'string') {
    return { path: raw, name: path.basename(raw), note: '' };
  }
  const p = String(raw?.path || '').trim();
  return {
    path: p,
    name: String(raw?.name || (p ? path.basename(p) : '')).trim(
),
    note: String(raw?.note || '').trim()
  };
}
function buildScreenshotKey(entry) {
  return normalizeScreenshotKey(entry.note || entry.name || entr
y.path);
}
function scaleImageSize(original, maxWidth, maxHeight) {
  const width = Math.max(1, Number(original?.width || 1));
  const height = Math.max(1, Number(original?.height || 1));
  const widthRatio = maxWidth / width;
  const heightRatio = maxHeight / height;
  const ratio = Math.min(widthRatio, heightRatio, 1);
  return {
    width: Math.max(1, Math.floor(width * ratio)),
```

```
    height: Math.max(1, Math.floor(height * ratio))
  };
}
function imageParagraphForScreenshot(entry) {
  const img = nativeImage.createFromPath(entry.path);
  const size = img.getSize();
  const scaled = scaleImageSize(size, 520, 700);
  const data = fs.readFileSync(entry.path);
  const children = [
    new ImageRun({
      data,
      transformation: scaled
    })
  ];
  return new Paragraph({
    alignment: AlignmentType.CENTER,
    children,
    spacing: { after: 200 }
  });
}
function markdownToDocChildren(markdown, screenshots) {
  const lines = String(markdown || '').replace(/\r\n/g, '\n').sp
lit('\n');
  const children = [];
  let inCodeBlock = false;
  const screenshotEntries = (Array.isArray(screenshots) ? screen
shots : []).map(buildScreenshotEntry);
  const screenshotByKey = new Map(screenshotEntries.map(e => [bu
ildScreenshotKey(e), e]));
  const usedScreenshotKeys = new Set();
  for (const rawLine of lines) {
    const line = rawLine.trimEnd();
    if (line.startsWith('```')) {
      inCodeBlock = !inCodeBlock;
      continue;
    }
    if (inCodeBlock) {
      children.push(
        new Paragraph({
          children: [
            new TextRun({
              text: line.length === 0 ? ' ' : line,
              font: 'Consolas',
              size: 21,
              preserve: true
            })
          ]
        })
      );
      continue;
```

```
    }
    const shot = line.match(/^\[\[SCREENSHOT:(.+?)\]\]\s*$/);
    if (shot) {
      const key = normalizeScreenshotKey(shot[1]);
      const entry = screenshotByKey.get(key);
      if (entry) {
        try {
          children.push(imageParagraphForScreenshot(entry));
          usedScreenshotKeys.add(key);
        } catch (_err) {
          children.push(
            new Paragraph({
              children: [
                new TextRun({
                  text: `截图加载失败：${entry.path}`,
                  font: 'Microsoft YaHei',
                  size: 22
                })
              ]
            })
          );
        }
      } else {
        children.push(
          new Paragraph({
            children: [
              new TextRun({
                text: `未找到匹配截图：${shot[1]}`,
                font: 'Microsoft YaHei',
                size: 22
              })
            ]
          })
        );
      }
      continue;
    }
    if (line.trim().length === 0) {
      children.push(new Paragraph({ text: '' }));
      continue;
    }
    const h3 = line.match(/^###\s+(.*)$/);
    if (h3) {
      children.push(new Paragraph({ text: h3[1].trim(), heading:
HeadingLevel.HEADING_3 }));
      continue;
    }
    const h2 = line.match(/^##\s+(.*)$/);
    if (h2) {
      children.push(new Paragraph({ text: h2[1].trim(), heading:
```

```
 HeadingLevel.HEADING_2 }));
      continue;
    }
    const h1 = line.match(/^#\s+(.*)$/);
    if (h1) {
      children.push(new Paragraph({ text: h1[1].trim(), heading:
 HeadingLevel.HEADING_1 }));
      continue;
    }
    children.push(
      new Paragraph({
        children: [new TextRun({ text: line, font: 'Microsoft Ya
Hei', size: 24 })]
      })
    );
  }
  const unused = screenshotEntries.filter(e => !usedScreenshotKe
ys.has(buildScreenshotKey(e)));
  return { children, unused };
}
function buildHeaderFooter(headerText) {
  const header = new Header({
    children: [
      new Paragraph({
        children: [new TextRun({ text: headerText || '', font: '
Microsoft YaHei', size: 24 })],
        alignment: AlignmentType.LEFT,
        spacing: { before: 0, after: 0 }
      })
    ]
  });
  const footer = new Footer({
    children: [
      new Paragraph({
        alignment: AlignmentType.CENTER,
        children: [
          new TextRun({ text: '— 第 ', font: 'Microsoft YaHei', s
ize: 20 }),
          new TextRun({ children: [PageNumber.CURRENT] }),
          new TextRun({ text: ' 页 / 共 ', font: 'Microsoft YaHei'
, size: 20 }),
          new TextRun({ children: [PageNumber.TOTAL_PAGES] }),
          new TextRun({ text: ' 页 —', font: 'Microsoft YaHei', s
ize: 20 })
        ],
        spacing: { before: 0, after: 0 }
      })
    ]
  });
  return { header, footer };
```

```
}
async function writeManualDocx(outputPath, options) {
  const softwareName = String(options?.softwareName || '').trim(
);
  const softwareVersion = String(options?.softwareVersion || '')
.trim();
  const headerText = String(options?.headerText || '').trim();
  const markdown = String(options?.markdown || '');
  const screenshots = Array.isArray(options?.screenshots) ? opti
ons.screenshots : [];
  const { header, footer } = buildHeaderFooter(headerText);
  const coverTitle = softwareName ? `${softwareName} 软件说明书` : '软
件说明书';
  const coverSubtitle = softwareVersion ? `版本：${softwareVersion}
` : '';
  const coverChildren = [
    new Paragraph({ text: '', spacing: { after: 400 } }),
    new Paragraph({
      alignment: AlignmentType.CENTER,
      children: [new TextRun({ text: coverTitle, font: 'Microsof
t YaHei', size: 56, bold: true })],
      spacing: { before: 2400, after: 500 }
    })
  ];
  if (coverSubtitle) {
    coverChildren.push(
      new Paragraph({
        alignment: AlignmentType.CENTER,
        children: [new TextRun({ text: coverSubtitle, font: 'Mic
rosoft YaHei', size: 28 })],
        spacing: { before: 200, after: 2000 }
      })
    );
  }
  const bodyChildren = [];
  bodyChildren.push(new Paragraph({ text: '目录', heading: Heading
Level.HEADING_1 }));
  bodyChildren.push(
    new TableOfContents('目录', {
      hyperlink: true,
      headingStyleRange: '1-3'
    })
  );
  bodyChildren.push(new Paragraph({ pageBreakBefore: true, text:
 '正文', heading: HeadingLevel.HEADING_1 }));
  const parsed = markdownToDocChildren(markdown, screenshots);
  bodyChildren.push(...parsed.children);
  if (parsed.unused.length > 0) {
    bodyChildren.push(new Paragraph({ pageBreakBefore: true, tex
t: '截图（未引用）', heading: HeadingLevel.HEADING_1 }));
```

```
    for (const entry of parsed.unused) {
      try {
        bodyChildren.push(imageParagraphForScreenshot(entry));
      } catch (_err) {
        bodyChildren.push(
          new Paragraph({
            children: [new TextRun({ text: `截图加载失败：${entry.path}
`, font: 'Microsoft YaHei', size: 22 })]
          })
        );
      }
    }
  }
  const doc = new Document({
    features: { updateFields: true },
    sections: [
      {
        properties: {
          page: {
            margin: { top: 1440, bottom: 1440, left: 1440, right
: 1440 }
          }
        },
        children: coverChildren
      },
      {
        properties: {
          page: {
            margin: {
              top: 1440,
              bottom: 1440,
              left: 1440,
              right: 1440,
              header: 720,
              footer: 720
            },
            pageNumbers: { start: 1, formatType: NumberFormat.DE
CIMAL }
          }
        },
        headers: { default: header, first: header },
        footers: { default: footer, first: footer },
        children: bodyChildren
      }
    ]
  });
  const buffer = await Packer.toBuffer(doc);
  fs.writeFileSync(outputPath, buffer);
}
module.exports = {
```

```
  writeManualDocx
};
const childProcess = require('child_process');
function toEncodedCommand(script) {
  return Buffer.from(script, 'utf16le').toString('base64');
}
function escapePsPath(value) {
  return value.replace(/'/g, "''");
}
function writePdfFromDocx(docxPath, pdfPath) {
  const docx = escapePsPath(docxPath);
  const pdf = escapePsPath(pdfPath);
  const script = `
$word = New-Object -ComObject Word.Application
$word.Visible = $false
$doc = $word.Documents.Open('${docx}')
$doc.ExportAsFixedFormat('${pdf}', 17)
$doc.Close()
$word.Quit()
`;
  const encoded = toEncodedCommand(script);
  childProcess.execFileSync('powershell', [
    '-NoProfile',
    '-NonInteractive',
    '-ExecutionPolicy',
    'Bypass',
    '-EncodedCommand',
    encoded
  ], { stdio: 'ignore' });
}
module.exports = {
  writePdfFromDocx
};
const fs = require('fs');
const path = require('path');
const EXCLUDED_DIRS = new Set([
  'node_modules', 'dist', 'build', 'out', '.git', 'vendor', 'bin
', 'obj',
  'target', '.idea', '.vscode', '.vs', '.svn', '.hg', '.turbo',
'.next',
  '.cache', '.angular', '.gradle'
]);
const EXCLUDED_FILES = new Set([
  'package-lock.json', 'yarn.lock', 'pnpm-lock.yaml'
]);
const SUPPORTED_EXTS = new Set([
  '.js', '.jsx', '.ts', '.tsx',
  '.py', '.java', '.cs', '.c', '.cpp', '.h', '.hpp',
  '.go', '.rs', '.php', '.swift', '.kt'
]);
```

```
const MAX_COLUMNS = 64;
const LINES_PER_PAGE = 50;
const TARGET_PAGES = 60;
const TARGET_LINES = LINES_PER_PAGE * TARGET_PAGES;
const END_LINE_REGEX = /(}\s*;?\s*$|end\s*;?\s*$|return\s*;?\s*$
)/i;
function getCommentSyntax(ext) {
  switch (ext) {
    case '.py':
      return { line: ['#'], block: ['"""', "'''"] };
    case '.php':
      return { line: ['//', '#'], block: ['/*'] };
    state.blockEnd = nextBlock.start === '/*' ? '*/' : nextBlock
.start;
    i = nextBlock.idx + nextBlock.start.length;
  }
  return { code: output, state };
}
function stripLineComment(code, syntax) {
  let earliest = -1;
  for (const marker of syntax.line) {
    const idx = code.indexOf(marker);
    if (idx !== -1) {
      if (earliest === -1 || idx < earliest) earliest = idx;
    }
  }
  if (earliest === -1) return code;
  return code.slice(0, earliest);
}
function isEffectiveLine(line, state, syntax) {
  if (line.trim().length === 0) return false;
  const blockResult = stripComments(line, state, syntax);
  const afterBlock = blockResult.code;
  const withoutLine = stripLineComment(afterBlock, syntax);
  if (withoutLine.trim().length === 0) return false;
  return true;
}
function readLines(filePath) {
  try {
    const raw = fs.readFileSync(filePath, 'utf8');
    return raw.replace(/\r\n/g, '\n').replace(/\r/g, '\n').split
('\n');
  } catch {
    return [];
  }
}
function paginate(lines, perPage) {
  const pages = [];
  for (let i = 0; i < lines.length; i += perPage) {
    pages.push(lines.slice(i, i + perPage));
```

```
  }
  return pages;
}
function normalizeLine(line) {
  return line.replace(/\t/g, '    ');
}
function wrapLine(line, maxCols) {
  if (line.length <= maxCols) return [line];
  const parts = [];
  for (let i = 0; i < line.length; i += maxCols) {
    parts.push(line.slice(i, i + maxCols));
  }
  return parts;
}
function scanProject(projectPath) {
  const files = walkDir(projectPath);
  const fileBlocks = [];
  let effectiveLines = 0;
  for (const file of files) {
    const ext = path.extname(file).toLowerCase();
    const syntax = getCommentSyntax(ext);
    const state = { inBlock: false, blockEnd: null };
    const fileLines = readLines(file);
    const effective = [];
    for (const line of fileLines) {
      const isEffective = isEffectiveLine(line, state, syntax);
      if (isEffective) {
        effective.push(normalizeLine(line));
      }
    }
    if (effective.length > 0) {
      const wrappedLines = [];
      for (const line of effective) {
        const chunks = wrapLine(line, MAX_COLUMNS);
        for (const chunk of chunks) {
          if (chunk.trim().length === 0) continue;
          wrappedLines.push(chunk);
        }
      }
      if (wrappedLines.length > 0) {
        fileBlocks.push({
          path: file,
          lines: wrappedLines,
          lastLine: wrappedLines[wrappedLines.length - 1]
        });
      }
    }
  }
  const lastFileIdx = (() => {
    for (let i = fileBlocks.length - 1; i >= 0; i -= 1) {
```

```
      if (END_LINE_REGEX.test(fileBlocks[i].lastLine)) return i;
    }
    return fileBlocks.length - 1;
  })();
  const eligibleBlocks = lastFileIdx >= 0 ? fileBlocks.slice(0,
lastFileIdx + 1) : [];
  const wrappedLines = eligibleBlocks.flatMap((block) => block.l
ines);
  effectiveLines = wrappedLines.length;
  const pages = paginate(wrappedLines, LINES_PER_PAGE);
  const totalPages = pages.length;
  let exportLines = [];
  if (wrappedLines.length >= TARGET_LINES) {
    if (totalPages > TARGET_PAGES) {
      exportLines = pages.slice(0, 30).concat(pages.slice(-30)).
flat();
    } else {
      exportLines = wrappedLines.slice(0, TARGET_LINES);
    }
  } else {
    exportLines = wrappedLines.slice();
  }
  if (wrappedLines.length === 0) {
    exportLines = [];
  } else if (exportLines.length < TARGET_LINES) {
    const paddingCount = TARGET_LINES - exportLines.length;
    const padLines = [];
    let idx = 0;
    while (padLines.length < paddingCount) {
      padLines.push(wrappedLines[idx % wrappedLines.length]);
      idx += 1;
    }
    exportLines = padLines.concat(exportLines);
  } else if (exportLines.length > TARGET_LINES) {
    exportLines = exportLines.slice(0, TARGET_LINES);
  }
  const pagesToExport = paginate(exportLines, LINES_PER_PAGE);
  return {
    projectPath,
    fileCount: files.length,
    effectiveLines,
    totalPages,
    pages,
    pagesToExport,
    selectedPages: pagesToExport.length
  };
}
module.exports = {
  scanProject
};
```