pdf-audio-server

JavaScript是关键技术栈

核心编程语言是 JavaScript，但它需要与 HTML、CSS、Node.js 运行环境、Express.js 框架以及一些第三方 JavaScript 库紧密配合才能构成这个完整的应用程序。

除了 JavaScript 之外，这个程序还依赖于以下关键技术：

1. **HTML (HyperText Markup Language):** 这是网页的基础骨架。`index.html` 文件的主体就是 HTML，它定义了页面上有什么元素（按钮、画布、输入框等）。
2. **CSS (Cascading Style Sheets):** 负责网页的样式和布局。在你的 `.html` 文件中，`<style>` 标签里的所有代码都是 CSS，用来控制颜色、大小、位置等外观效果。
3. **Node.js:** 这是一个 JavaScript 的运行环境。它让你的 JavaScript 代码 (`server.js`) 能够脱离浏览器，在服务器（你的电脑）上运行，执行文件操作、网络服务等后端任务。
4. **Express.js:** 这是一个基于 Node.js 的框架。`server.js` 文件中 `require('express')` 就是在引入它。它简化了编写后端 Web 服务的过程，比如处理路由（URL 请求）、管理请求和响应等。
5. **库 (Libraries):** 前端 JavaScript 代码还用到了几个重要的第三方库：
   - **pdf.js:** 用于在浏览器中加载和渲染 PDF 文件。
   - **pdf-lib:** 可能用于未来潜在的 PDF 修改功能（虽然目前主要用 pdf.js 显示）。
   - **marked:** 用于将 Markdown 格式的文本笔记转换成 HTML 显示。

总结：
主要使用的编程语言是 JavaScript，它同时用在了前端和后端。融合以下几种 Web 技术组合编写：

1. **前端 (Frontend - 浏览器里运行的部分):**
   - **HTML (`.html` 文件):** 定义网页的结构和内容。
   - **CSS (在 `.html` 文件里的 `<style>` 标签内):** 定义网页的样式和外观。
   - **JavaScript (在 `.html` 文件里的 `<script>` 标签内):** 处理用户交互、操作页面元素、与后端通信等所有动态逻辑。
2. **后端 (Backend - 服务器上运行的部分):**
   - **JavaScript (运行在 Node.js 环境下，即 `server.js` 文件):** 处理来自前端的请求（比如保存笔记、上传文件）、读写文件、管理数据等。
   - **Node.js:** 是一个让 JavaScript 可以在服务器端运行的环境。
   - **Express.js:** 是一个基于 Node.js 的框架，用来简化后端服务器的开发。

1. 整体架构：两个程序协同工作 (Client-Server)

- 不像单个的 C 程序那样从头运行到尾，它更像是两个独立的程序在对话：

- **前端 (Client - 客户端):** `index.html` 运行在我们用户的**浏览器**里（就像 Chrome、Edge）。它负责展示界面（PDF 内容、按钮、标注图标）、响应用户的操作（点击、滚动、录音），并且在需要的时候向“后台程序”发送请求。
- **后端 (Server - 服务器):** `server.js` 处于运行状态时，它像一个服务员，时刻准备接收前端发来的“指令”（请求），比如“保存这个笔记”、“上传这个录音文件”，然后它会处理这些指令（比如把笔记存到文件里），处理完再告诉前端结果。

2. 前端：构建用户界面和交互 (HTML, CSS, JavaScript)

- **HTML (`.html` 文件):** 对比 C 语言，可以把它想象成定义程序**窗口布局**的描述文件。它规定了界面上应该有哪些元素（比如按钮 `button`、画布 `canvas`、文本输入框 `input`），以及这些元素的大致结构。它本身没有逻辑。
- **CSS (在 `<style>` 标签里):** 这是用来**美化** HTML 元素的。它控制元素的颜色、大小、位置、边框等等。C 语言里通常没有直接对应的东西，需要在代码里手动设置控件的各种属性，而 CSS 把这些样式规则分离出来了。
- **JavaScript (在 `<script>` 标签里):** 这是前端的**核心逻辑**，最像 C 代码的部分。
  - 它负责处理用户的**事件**（比如点击按钮 `onclick`、鼠标移动）。这有点像 C 语言 GUI 编程里的事件回调函数。
  - 它能**动态修改** HTML 和 CSS（比如显示/隐藏元素、改变文本内容）。
  - 它使用**库 (Libraries)** 来完成复杂任务，比如 `pdf.js` 这个库专门负责读取和显示 PDF 文件，就像 C 语言里 `#include` 各种头文件来使用别人写好的函数库（比如 `stdio.h`）。
  - **关键：** 它通过网络向后端发送 **HTTP 请求** (使用 `fetch` 函数) 来请求数据或保存数据。这就像 C 程序通过某种方式（比如读写文件、或者更复杂的网络套接字 Socket）和另一个 C 程序通信。

3. 后端：处理数据和逻辑 (.js, Express)

- **.js:** 提供了一个**运行环境**，让 JavaScript 代码可以像 C 程序一样，直接在**服务器电脑**上运行，而不仅仅是在浏览器里。它提供了访问文件系统（`fs` 模块，类似 C 的 `fopen`，`fwrite`）、处理网络请求等功能。
- **Express (代码里的 `require('express')`):** 这是一个构建在 Node.js 之上的**框架 (Framework)**。它极大地简化了编写后端服务的过程。
  - 它帮助我们处理来自前端的 **HTTP 请求**。比如定义当访问 `/save-notes` 这个“地址” (URL) 时，应该执行哪个 JavaScript 函数。这比在 C 里面从头处理原始的网络连接和 HTTP 协议要简单得多。
  - `app.post('/save-notes', ...)` 这种代码就是定义一个**路由 (Route)**：当前端向 `/save-notes` 发送 POST 请求时，执行后面的函数。
  - `app.use(express.json(...))` 这种叫做**中间件 (Middleware)**，它在请求到达最终处理函数之前，先做一些预处理，可以让服务器能接收更大的 JSON 数据。
- **数据存储:** 后端把笔记数据存储在 `notes.json` 文件里，把上传的音频和图片文件存储在服务器的特定文件夹里。这和 C 程序将数据写入本地文件是类似的概念。

4. 通信方式：HTTP 和 JSON

- **HTTP:** 这是前端和后端之间通信的**协议**（规则）。前端发送一个**请求 (Request)**（包含想做什么、需要什么数据），后端回复一个**响应 (Response)**（包含操作结果、请求的数据）。
- **JSON:** 这是一种轻量级的**文本数据格式**，用来在前端和后端之间传递结构化数据（比如你的笔记列表 `notes`）。它比 C 语言的二进制结构体 `struct` 更通用，易于阅读，并且是网络通信的事实标准。后端将 JavaScript 对象转换成 JSON 字符串发送给前端，前端接收后再解析回 JavaScript 对象。