一、项目概述

1.1 项目背景

在当前移动互联网环境下，学生的学习流程往往被迫拆散在多个平台：他们先要在密密麻麻的 PDF 文档中反复查找知识点，然后又得跳转到其他刷题网站练习，遇到不会的题目时还不得不额外搜索资料、或者费力地向老师同学求助，这一整套流程既零散又低效。为了尽可能打通这几步分离的环节，本项目尝试紧密结合 **HTML5 + CSS3 + JavaScript** 等移动前端技术与大语言模型（AI）的能力，在同一个移动端应用中完成从 PDF 教材到习题再到解析与回看原文的闭环，让系统能够自动从用户上传的 PDF 资料中生成题目，智能地给出解析，并且在需要时精确地定位回原始 PDF 内容，从而更集中、更顺畅地支持用户的日常学习与复习。

1.2 项目目标

本项目的总体目标，是在移动端为用户提供一个围绕 PDF 教材展开的智能刷题平台，使学习过程尽可能集中而连贯：系统能够自动地从用户上传的 PDF 中抽取内容并生成相应题目，在用户作答出错时及时给出判定结果，并且精确地定位回 PDF 中的对应位置，帮助他们迅速对照原文理解知识点；同时，每一道题目下方都支持以对话形式向 AI 提问，让用户可以随时就某个困惑展开更深入的追问与讨论；对于包含图片的题目，系统也可以较为自然地展示图文信息并配合解析；此外，用户可自行灵活填写并管理自己的 API Key，系统本身不内置密钥，从而在满足功能需求的同时，尽量兼顾安全性与通用性。

1.3 项目特色与亮点

与传统的静态刷题工具相比，本项目在设计上力求更加智能化和一体化：首先，系统可以自动地从 PDF 材料中抽取内容并生成题库，显著减轻了人工出题的重复工作；其次，每道题目都保留了清晰的来源信息，用户在答错时可以一键跳转回 PDF 的具体页码与段落，迅速回看原文并进行针对性复习；再次，项目引入了题目级别的 AI 智能助教，用户能够直接围绕某一道题发起多轮对话式提问，获得更加细致、渐进式的讲解；在界面层面，前端采用移动端优先的设计理念，使整体视觉效果显得更加现代与柔和；最后，系统不预置任何固定的 API Key，而是允许用户自行安全地配置与管理，从而使应用在不同 AI 服务之间具有良好的可移植性和扩展空间。

二、需求分析

2.1 用户需求分析

1. **目标用户**

本项目面向的核心用户群体主要是经常需要阅读 PDF 教材并进行课后复习的大学生，以及依赖电子讲义进行系统性自学的学习者。他们通常需要在有限的时间内高效地浏览资料、反复做题并查漏补缺，因此尤其希望有一款工具可以在手机上随时打开、轻量地使用，同时又能够在遇到难题时，快速地获得来自 AI 的解释与辅导。

2. **使用场景**

在典型的使用场景中，用户首先会将老师下发的 PDF 讲义或课件上传到系统，由系统自动地从中抽取重要内容并生成对应题库，随后用户便可以在手机端分散的时间里随时刷题，逐步巩固知识点；当他们遇到不会的题目时，不仅能够立即查看基础解析，还可以一键返回到 PDF 的原文位置对照学习，并在题目下方直接向 AI 发起提问，从而在同一个应用中较为顺畅地完成“看教材—做练习—问问题—回看原文”的完整学习闭环。

2.2 功能需求分析

2.2.1 系统整体功能模块（概览）

- 前端（HTML5 + CSS3 + JavaScript / Vue）
    - PDF 上传与管理
    - 题库浏览与选择
    - 答题与判分
    - 错题本与原文定位
    - AI 对话解析模块
    - 设置与 API Key 管理
- 后端
    - PDF 解析与内容抽取模块
    - 自动出题模块（调用 AI）
    - 图片提取与管理模块
    - 题目与 PDF 段落映射管理
- 外部服务
    - AI 模型 API（用户提供 Key）

2.2.2 具体功能需求

1. **PDF 上传与解析**
    - 用户在前端选择本地 PDF 文件并上传；
    - 后端接收 PDF，解析为按页与段落分组的文本数据；
    - 对于 PDF 中的图片，提取并保存对应页码和位置，用于图片题。
2. **题库生成与管理**
    - 后端基于解析出的 PDF 文本块调用 AI，自动生成题目：
        - 选择题为主，包含题干、选项、正确答案、解析；
        - 每道题包含来源信息（页码、段落索引等）。
    - 前端可以查看当前 PDF 生成的题库列表；
    - 支持按章节/页码分组展示题目（可选）。
3. **答题与判分**
    - 前端以移动端友好的方式呈现题目：
        - 支持文字题、图文题；
        - 支持单选题（后期可扩展多选题）。
    - 用户作答后，前端立即判定对错：
        - 显示是否答对；
        - 显示标准答案；
        - 显示基础解析（可由 AI 预生成或手动生成）。
4. **错题本与原文定位**
    - 将用户答错的题目保存到本地（`localStorage`）或后端存储；
    - 错题题目中必须包含 PDF 来源信息：
        - 至少包括 `pdfId`、`page`、`paragraphIndex` 等字段；
    - 当用户在错题本或答题页点击「查看原文」：

        - 打开 PDF 预览组件；
        - 自动跳转到对应页；
        - 在页面下方或侧边显示对应段落内容，并进行高亮提示。

5. **题目级 AI 对话解析**
    - 每道题目下面提供一个 AI 助教对话区域：
        - 用户输入对当前题的疑问；
        - 系统将题目信息（题干、选项、用户答案、正确答案）、相关 PDF 段落文本与用户问题一并发送给 AI；
        - AI 返回详细解析与引导；
        - 对话历史以问答形式展示，支持多轮互动。
    - API Key 由用户在设置页填写并保存，仅用于当前用户调用 AI。
6. **API Key 管理**
    - 前端设置页面：
        - 提供输入框让用户填写 AI 服务的 API Key；
        - Key 存储在浏览器 `localStorage` 中，不上传给其他用户；
    - 系统调用 AI 时：
        - 使用当前用户填入的 Key；
        - 若未填写 Key，则提示用户填写，或退回到"伪 AI 解析模式"。
7. **统计与用户反馈（可选）**
    - 统计某 PDF 下用户的正确率、答题数量、易错题等；
    - 显示学习进度图表（简单柱状图 / 环形图）。

2.3 非功能需求分析

在非功能需求方面，系统首先需要在易用性上表现得尽可能友好：界面整体采用移动端优先的设计思路，在手机上应能够自然地适配不同尺寸的屏幕，按钮与文字布局清晰、操作路径简洁，

使用户可以顺畅地完成从上传 PDF 到刷题、查看错题的完整流程。与此同时，性能表现也不容忽视，由于 PDF 解析和题目生成往往比较耗时，系统应尽量采用异步方式处理这些操作，并在前端通过加载动画或进度提示及时反馈当前状态，另外还需要通过批量生成题目、复用解析结果等策略，有意识地减少不必要的 AI 调用次数，从而在体验与成本之间取得比较合理的平衡。安全性方面，项目约定由用户自行填写并管理自己的 API Key，前端只在本地保存这一信息，并在界面上明确提示"不上传到公共服务器"；若后端在某些模式下需要短暂接收 Key，也应确保不做持久化存储，在完成本次调用后立即丢弃相关敏感数据。最后，从可扩展性的角度出发，前后端之间尽量采用结构清晰的 RESTful 接口设计，题目、PDF 段落等核心数据结构要有相对稳定的定义，这样一方面方便后续将后端实现语言替换为 Rust 等更高性能的方案，另一方面也便于将当前接入的 AI 服务替换为不同平台或不同模型，从而为系统的长期演进预留足够空间。

三、项目实现方案

3.1 技术选型

1. **前端**

   在技术选型上，前端部分将以 HTML5、CSS3 和 JavaScript 作为基础技术栈，并优先考虑使用 Vue.js（推荐 Vue3 搭配 Vite）来构建整体界面与交互逻辑。通过组件化的方式，可以更加清晰地拆分 PDF 上传区域、题库列表、答题卡片、错题本以及设置页面等界面模块；在样式层面，项目将大量使用 Flex 布局和自适应布局方案，使应用在不同尺寸的手机屏幕上都能较为自然地展示。同时，界面风格上会尝试引入液态玻璃（Glassmorphism）风格的卡片与悬浮面板，借助 `backdrop-filter` 等 CSS 特性营造出略带科技感又不过分炫目的视觉效果。为了在前端直接展示 PDF 内容并支持按页跳转，系统计划集成 `pdf.js` 作为 PDF 预览和渲染的核心库；用户的错题记录、简单配置项以及 API Key 等本地数据，则会通过浏览器的 `localStorage` 进行持久化存储，以减少对后端的依赖并提升使用灵活性。

2. **后端与 AI 服务**

   后端实现方面，项目会选用相对轻量且生态成熟的技术路线，例如基于 Node.js 配合 Express 或 Koa，或者采用 Python 搭配 FastAPI、Flask 等框架，来完成文件上传接口、PDF 解析与文本抽取逻辑，以及题目生成相关的业务处理。后端将调用对应语言的 PDF 解析库，从用户上传的文件中提取出按页、按段落组织的文本与图片信息，再在此基础上负责管理题目与 PDF 段落之间的映射关系。AI 服务层则通过外部的大语言模型 API 提供支持，项目本身不内置固定的密钥，而是由用户在前端界面中填写并保存自己的 API Key；在具体调用时，前端或后端会根据当前模式，携带该 Key 调用 AI 接口，完成题目生成、解析生成以及题目级对话等功能。这样的设计既能够充分利用现有 AI 平台的能力，又在一定程度上保持了服务提供方和模型类型的可替换性，方便后续根据实际需求进行调整和升级。

3.2 系统总体架构设计

可以抽象为三层结构：

1. **前端展示层（手机浏览器 / WebView）**
    - 负责 UI 展示、用户交互、题目渲染、PDF 预览、AI 对话展示。
2. **后端业务层**
    - 负责：
        - PDF 文件处理；
        - 文本与图片抽取；
        - 题目生成与存储；
        - 为前端提供统一的 RESTful API。
3. **AI 服务层**
    - 外部 AI 平台，负责根据传入的上下文（题目、原文、用户问题）生成解析与题目。

3.3 模块设计

1. **页面模块**
    - 首页（PDF 列表 / 入口导航）
    - PDF 上传页面
    - 题库列表页面（按 PDF 或章节分类）
    - 答题页面
    - 错题本页面
    - 设置页面（API Key 填写、主题风格等）
2. **核心组件**
    - `QuestionCard` 组件：
        - 展示题干、选项、解析、图片；

        - 采用液态玻璃风格。
    - `AIChatBox` 组件：
        - 展示与 AI 的对话历史；
        - 包含输入框和发送按钮。
    - `PDFViewer` 组件：
        - 基于 `pdf.js` 实现；
        - 支持传入 `page` 参数跳转到指定页。
3. **前端数据管理**
    - 使用全局状态（例如 Vue 的 `pinia` 或简单的全局对象）存储：
        - 当前题库；
        - 用户作答记录；
        - 错题列表；
        - API Key。

3.4 关键流程设计

1. **PDF 上传与题目生成流程**
    1. 用户在前端选择 PDF 文件并上传；
    2. 后端保存文件，解析文本和图片；
    3. 后端调用 AI 为文本块生成题目列表；
    4. 后端将生成的题目列表返回给前端；
    5. 前端将题目缓存到本地并进入题库页面。
2. **答题与判分流程**
    1. 前端从题库中取出题目展示；
    2. 用户选择答案并点击提交；

3. 前端比对 `userAnswerIndex` 与 `answerIndex`；
4. 如果为简答题, 则交由AI处理；
5. 显示对错结果及基础解析；
6. 若答错，则记录到错题本（`localStorage`）。

3. **错题回顾与原文定位流程**
    1. 用户打开错题本页面；
    2. 从 `localStorage` 读取错题列表并展示；
    3. 用户点击某道错题的「查看原文」；
    4. 前端调用 `PDFViewer`，并传入对应 `page`；
    5. 同时展示对应段落文本，并用高亮样式标记。
4. **题目级 AI 对话流程**
    1. 用户在某题下方的对话框中输入问题；
    2. 前端收集：题目信息 + 用户答案 + PDF 段落文本 + 对话历史 + 用户问题；
    3. 前端（或后端）使用用户提供的 API Key 调用 AI 模型；
    4. AI 返回解析内容，前端追加到该题的对话记录中展示。