下面的总结将涵盖项目的整体架构、数据模型、API 端点、已实现的功能、我们共同调试并完成的部分，以及尚未实现的功能。

在线考试平台项目总结与复刻蓝图

1. 项目概述

本项目是一个全栈在线考试平台，采用前后端分离架构。

- **前端 (`frontend`)**: 使用 React 和 Vite 构建，负责用户交互界面，包括学生考试、管理员后台管理等。
- **后端 (`backend`)**: 使用 Node.js 和 Express 构建，负责提供 API 服务、处理业务逻辑，并通过 Prisma ORM 与 PostgreSQL 数据库进行交互。

核心目标

- **学生端**: 参与考试、提交答案、查看结果。
- **管理员端**: 创建和管理考试、题目、学科和知识点。

---

2. 技术栈

- **后端**:
  - **框架**: Node.js, Express.js
  - **数据库 ORM**: Prisma
  - **数据库**: PostgreSQL
  - **身份验证**: JWT (JSON Web Tokens)
  - **密码处理**: bcrypt
- **前端**:
  - **框架**: React
  - **构建工具**: Vite
  - **路由**: `react-router-dom`
  - **状态管理**: React Context (`AuthContext`)
  - **API 请求**: Axios (通过 `src/services/api.js` 封装)

---

3. 数据库模型 (`prisma/schema.prisma`)

这是整个应用的数据结构核心。

<details> <summary>点击展开/折叠 Prisma Schema 详情</summary>

- `User`: 用户模型，包含 `id`, `username`, `password`, `role` (ADMIN 或 STUDENT)。
- `Subject`: 学科模型，如“数学”、“编程”，包含 `id`, `name`。
- `KnowledgePoint`: 知识点模型，如“Java 循环”，包含 `id`, `name`，并关联到 `Subject`。
- `Exam`: 考试模型，包含 `id`, `title`, `description`, `startTime`, `endTime`。
- `Question`: 题目模型，包含 `id`, `questionText`, `questionType` (MULTIPLE_CHOICE, MULTIPLE_RESPONSE), `layoutType` 等，并关联到 `Exam` 和多个 `KnowledgePoint` (多对多关系)。
- `MultipleChoiceOption / MultipleResponseOption`: 选项模型，关联到 `Question`。
- `Submission`: 学生提交的答案记录，关联到 `User` 和 `Exam`。
- `Answer`: 具体的答案，关联到 `Submission` 和 `Question`。

关系亮点:

- 一个 `Subject` 可以有多个 `KnowledgePoint`。
- 一个 `Question` 可以关联多个 `KnowledgePoint`，一个 `KnowledgePoint` 也可以被多个 `Question` 关联（多对多）。
- `onDelete: Cascade` 已在多个关系中设置，确保删除父记录时，相关的子记录也会被删除（例如，删除一个 `Question` 会同时删除其所有 `Options`）。

</details>

---

4. 后端 API 端点 (`backend/src/routes/`)

这是前后端通信的桥梁。

A. 认证接口 (`authRoutes.js`)

- `POST /api/auth/register`: 用户注册。
- `POST /api/auth/login`: 用户登录，成功后返回 JWT。

B. 管理员接口 (`/api/admin/...`)

- **考试管理 (`adminExamRoutes.js`)**:
  - `POST /exams` : 创建新考试。
  - `GET /exams` : 获取所有考试列表。
  - `GET /exams/:id` : 获取单个考试详情。
  - `PUT /exams/:id` : 更新考试信息。
  - `DELETE /exams/:id` : 删除考试。
- **题目管理 (`adminQuestionRoutes.js`)**:
  - `POST /questions` : 为指定考试创建新题目。
  - `PUT /questions/:id` : 更新题目信息。
  - `DELETE /questions/:id` : 删除题目。
- **知识点管理 (`adminKnowledgePointRoutes.js`)** - **我们共同完成的部分**
  - `POST /knowledge-points` : 创建新知识点。 **(已完成)**
  - `DELETE /knowledge-points/:id` : 删除知识点。 **(已完成)**
  - `GET /knowledge-points` : 获取所有知识点。**(待办)**
  - `PUT /knowledge-points/:id` : 更新知识点。**(待办)**
- **标签关联管理 (`adminQuestionTaggingRoutes.js`)** - **我们共同完成的部分**
  - `POST /question-tags/add` : 为问题添加知识点标签。 **(已完成)**
  - `DELETE /question-tags/remove` : 从问题移除知识点标签。 **(已完成)**

C. 学生端接口 (`/api/...`)

- **考试接口 (`examRoutes.js` , `questionRoutes.js`)**:
  - `GET /exams` : 获取对学生开放的考试列表。
  - `GET /exams/:id/questions` : 获取某场考试的所有题目。
- **提交接口 (`submissionRoutes.js`)**:
  - `POST /submissions` : 提交整份试卷的答案。
  - `GET /submissions/results/:examId` : 获取某场考试的结果。
- **其他 (`subjectRoutes.js` , `examBoardRoutes.js`)**: 用于获取学科信息和考试看板数据。

---

5. 我们共同调试和完成的工作 (重点)

我们聚焦于**管理员后台的知识点管理**这一垂直领域，并成功完成了以下工作：

1. **实现了创建知识点的功能**:
   - **API**: `POST /api/admin/knowledge-points`
   - **过程**: 解决了 `404 Cannot POST` 路由错误，通过添加日志、修复路由和控制器文件，成功打通了从请求到数据库写入的完整链路。
2. **实现了删除知识点的功能**:
   - **API**: `DELETE /api/admin/knowledge-points/:id`
   - **过程**: 同样通过修复缺失的路由规则，实现了此功能，并添加了对删除不存在记录的错误处理。
3. **实现了为问题"贴标签"的功能**:
   - **API**: `POST /api/admin/question-tags/add`
   - **过程**: 这是我们调试最久的部分。我们解决了：
     - `Prisma P2025` 错误：通过预先验证数据存在性。
     - `400 Bad Request` 错误：通过统一前后端不一致的参数命名 (`knowledgePointId` vs `newPointId`)。
     - `409 Conflict` 错误：通过增加逻辑防止重复添加同一个标签。
4. **实现了从问题"移除标签"的功能**:
   - **API**: `DELETE /api/admin/question-tags/remove`
   - **过程**: 修复了缺失的 `DELETE` 路由规则，并确保了功能的健壮性。

总结: 我们不仅实现了功能，更重要的是建立了一套**健壮的、包含完整错误处理**的 API 服务，并创建了相应的测试用例 (`api-tests.http`) 来验证其正确性。

---

6. 未完成/待办事项 (Roadmap)

基于对整个代码库的分析，以下是项目可以继续推进的方向：

A. 后端 (Backend)

1. **补全 CRUD 操作**:
   - 为 `KnowledgePoint` 添加 `GET` (列表/单个) 和 `PUT` (更新) 的 API。
   - 为 `Subject` 添加完整的 CRUD API。
2. **权限验证中间件**:
   - 创建一个 `authMiddleware.js` ，用于检查请求头中的 JWT，并验证用户角色。
   - 将此中间件应用到所有 `/api/admin/*` 路由上，以保护管理员接口。
3. **高级查询与分页**:
   - 在获取考试列表、题目列表等 `GET` 请求中，加入分页 (`skip` , `take`)、排序 (`orderBy`) 和筛选 (`where`) 功能。
4. **精细化输入验证**:

    - 使用 `express-validator` 等库，对所有接受输入的 API（特别是 `POST` 和 `PUT`）进行严格的字段验证。
5. **单元/集成测试**:
    - 使用 Jest 或 Supertest 等框架编写自动化的测试脚本，替代手动的 `.http` 文件测试。

B. 前端 (Frontend)

1. **管理员仪表盘 (Admin Dashboard)**:
    - 创建专门的 UI 界面来调用我们已经完成的知识点管理 API。
    - 需要有表单用于创建知识点，有列表展示所有知识点，并提供删除和编辑按钮。
    - 需要有一个界面，可以查看某个问题，并为其动态添加或移除知识点标签。
2. **状态管理优化**:
    - 对于复杂的全局状态（如考试数据、题目列表），可以考虑引入更强大的状态管理库，如 Redux Toolkit 或 Zustand，以替代 React Context。
3. **用户体验优化**:
    - 在 API 请求期间显示加载（Loading）状态。
    - 对 API 返回的错误进行捕获，并在 UI 上向用户显示友好的提示信息。
    - 实现考试倒计时功能。
4. **组件化**:
    - 将大型页面（如 `ExamPage.jsx`）拆分为更小的、可复用的组件。

这份文档就是您当前项目的完整快照。任何了解 Node.js 和 React 的开发者都可以基于这份蓝图，理解项目的设计思路，并继续完成后续的开发任务。