

广州大学课程设计报告

系统名称 Edu-Hub 在线教育平台

开课学院 计算机科学与网络工程学院

实验室 电子信息楼 518

指导教师 张少宏、颜国风

开课时间 2025 年 12 月 22 日-2026 年 1 月 3 日

组员分工表

学号	姓名	班级	项目角色	业务分工	评语	成绩
32306300071	杨竣淇	软件 233	组长、系统分析师、架构师、测试员	需求分析、系统设计、前端开发、数据库设计、接口测试、性能测试		

目录

第一章 背景	5
1.1 课题背景	5
1.2 国内外现状	5
1.3 课题内容	6
1.4 可行性分析	6
1.4.1 技术可行性分析/技术路线	6
1.4.2 本课题的意义	7
第二章 需求分析	8
2.1 需求概况	8
2.1.1 组织机构图与用户分析	8
2.1.2 主体业务需求	8
2.1.3 质量需求	9
2.1.4 其它（约束性）需求	9
2.2 用例分析（功能需求详细分析）	10
2.2.1 学生用例视图	10
2.2.2 教师用例视图	11
2.2.3 管理员用例视图	13
2.3 用户注册与登录	14
2.3.1 用例中的类	14
2.3.2 关键的类及状态	14
2.3.3 用例主体行为	14
2.3.4 用例异常行为	15
2.4 课程发布与管理	15
2.4.1 用例中的类	15
2.4.2 关键的类及状态	15
2.4.3 用例主体行为	15
2.4.4 用例异常行为	15

2.5	视频学习	15
2.5.1	用例中的类	16
2.5.2	关键的类及状态	16
2.5.3	用例主体行为	16
2.5.4	用例异常行为	16
2.6	社区互动	16
2.6.1	用例中的类	16
2.6.2	关键的类及状态	16
2.6.3	用例主体行为	16
2.6.4	用例异常行为	17
2.7	完整的类图	18
2.8	用户验收测试计划	18
第三章	概要设计	20
3.1	系统分解 (体系结构)	20
3.1.1	架构图	21
3.1.2	子系统职责定义	21
3.2	系统总体流程	22
3.3	界面设计原则 (交互风格规范)	23
3.4	界面原型设计	24
3.4.1	登录与注册页	24
3.4.2	主页 (Home)	24
3.4.3	课程广场	25
3.4.4	课程学习页	25
3.4.5	课程编辑器	26
3.5	系统集成测试计划与设计	26
第四章	详细设计	28
4.1	类 (数据) 设计	28
4.1.1	数据库物理模型	28
4.1.2	设计优化与考量	32
4.2	关键逻辑、算法设计	32
4.2.1	章节树构建算法 (Recursive Tree Building)	32
4.2.2	课程总进度计算算法	33
4.2.3	权限控制逻辑 (Middleware Strategy)	34
4.2.4	基于角色的数据展示逻辑 (Role-Based Data Presentation)	34
4.3	模块接口测试计划	35

4.3.1	用户认证模块 (Auth)	35
4.3.2	课程管理模块 (Course)	35
4.3.3	章节与进度模块 (Chapter)	36
4.3.4	论坛模块 (Forum)	36
第五章 编码实现		37
5.1	编码工具、技术介绍	37
5.1.1	开发环境与工具	37
5.1.2	后端技术栈 (Backend)	37
5.1.3	前端技术栈 (Frontend)	38
5.2	源代码阅读指南	38
5.2.1	后端工程结构 (backend/)	38
5.2.2	前端工程结构 (frontend/src/)	39
5.2.3	文档与数据库 (doc/ & database/)	40
5.3	单元测试方案	40
5.3.1	测试原则	41
5.3.2	关键要素	41
第六章 测试实施		42
6.1	功能测试	42
6.1.1	测试所采用的技术	42
6.1.2	关键测试代码及测试数据集	42
6.1.3	测试统计报告	43
6.2	性能测试	45
6.2.1	测试所采用的技术	45
6.2.2	关键测试代码及测试数据集	46
6.2.3	测试统计报告	46
第七章 安装、使用说明		48
7.1	安装说明	48
7.1.1	安装前准备	48
7.1.2	系统安装步骤	48
7.2	使用说明	49
7.2.1	访问系统	49
7.2.2	典型操作流程	49

第八章 复用说明及示例	51
8.1 复用基础	51
8.2 复用案例需求描述	51
8.3 复用案例的设计	51
8.4 复用案例的实现	52
8.5 复用案例的运行效果	52
8.6 小结	52
第九章 系统功能展示	54

第一章 背景

随着信息技术的飞速发展，“互联网 + 教育”已成为推动教育现代化的重要引擎。本项目旨在构建一个名为“Edu-Hub”的在线教育平台，利用现代 Web 技术打破传统教育的时空限制，提供高效、互动的学习体验。

1.1 课题背景

在数字化转型的浪潮下，传统课堂教学模式正面临着前所未有的挑战与机遇。一方面，优质教育资源的分布不均限制了部分地区学生的学习机会；另一方面，传统的单向灌输式教学难以满足学生个性化、自主化的学习需求。国家《教育信息化 2.0 行动计划》明确提出要构建“互联网 +”条件下的人才培养新模式。在此背景下，开发一款功能完善、交互性强、易于扩展的在线教育平台（Edu-Hub）显得尤为重要。它不仅是技术应用的实践，更是对新型教学模式的探索，旨在通过技术手段实现教育资源的优化配置和共享。

1.2 国内外现状

当前，在线教育市场已呈现出蓬勃发展的态势。

国外方面：以 Coursera、edX 为代表的 MOOC 平台汇聚了全球顶尖高校的课程资源，技术成熟，用户体验优秀，但往往存在语言障碍、证书费用高昂以及对特定教学场景适配不足的问题。

国内方面：中国大学 MOOC、腾讯课堂、网易云课堂等平台发展迅速，覆盖了从 K12 到职业教育的广泛领域。然而，这些大型平台通常面向大规模公网用户，对于特定学校或机构的私有化部署、定制化教学管理（如特定的班级管理、实验报告提交等）支持不够灵活。此外，部分平台在师生互动、社区氛围营造方面仍有提升空间，学生容易产生“孤独学习”的倦怠感。

本课题切入点：Edu-Hub 平台定位于一个轻量级、高互动性的在线教学系统，专注于解决中小型教学场景下的资源管理、过程追踪和师生互动问题。

1.3 课题内容

基于上述现状，Edu-Hub 在线教育平台致力于解决传统在线学习中“互动缺失”和“监管困难”两大痛点。本系统主要包含以下核心内容：

- 构建多角色用户体系**：实现学生、教师、管理员的分级管理与权限控制，确保系统安全有序。
- 全流程课程管理**：支持课程的创建、富文本介绍编辑、章节视频上传与管理，满足多样化的教学资源展示需求。
- 沉浸式学习体验**：开发视频流媒体播放功能，结合章节导航，提供流畅的学习环境。
- 闭环式教学评估**：通过在线测试、题库管理和自动评分系统，实时检验学习效果。（注：因开发周期限制，本功能暂未实现）
- 活跃的社区互动**：建立课程评论区和问答社区，促进师生、生生之间的交流与协作。
- 数据驱动的教学决策**：利用可视化图表展示学习进度和成绩分布，为教师调整教学策略提供数据支持。（注：因开发周期限制，本功能暂未实现）

1.4 可行性分析

1.4.1 技术可行性分析/技术路线

本项目采用当前业界主流且成熟的前后端分离技术栈，具有极高的技术可行性：

- 前端 (Vue.js 3 + Tailwind CSS)**：Vue 3 的组合式 API (Composition API) 提供了优秀的逻辑复用能力和代码组织方式，配合 Vite 构建工具，开发效率极高。Tailwind CSS 提供了原子化的样式类，能快速构建美观且响应式的用户界面。
- 后端 (Node.js + Express)**：Express 是 Node.js 生态中最流行的 Web 框架，轻量灵活，拥有庞大的中间件生态（如 `jsonwebtoken` 用于鉴权，`multer` 用于文件上传），非常适合构建 RESTful API。
- 数据库 (MySQL 8.0)**：MySQL 是最流行的开源关系型数据库，支持事务处理和复杂的关联查询，能够稳定存储用户信息、课程结构、学习进度等核心数据。
- 状态管理与交互 (Pinia + Axios)**：Pinia 作为 Vue 的新一代状态管理库，轻量且直观；Axios 则是处理 HTTP 请求的标准库。

综上所述，开发团队对上述技术栈有充分的掌握，且社区资源丰富，遇到问题易于解决，技术风险可控。

1.4.2 本课题的意义

社会意义：Edu-Hub 平台的推广有助于打破地域限制，促进优质教育资源的跨区域流动，在一定程度上缓解教育资源分配不均的问题，促进教育公平。同时，它为终身学习者提供了便捷的渠道，助力学习型社会的建设。

经济意义：对于教育培训机构或学校而言，部署 Edu-Hub 平台可以大幅降低线下教学的场地租赁、纸质资料印刷等成本。通过自动化的作业批改和数据统计，能显著减轻教师的重复性工作负担，提高教学管理的效率和产出比。

第二章 需求分析

2.1 需求概况

2.1.1 组织机构图与用户分析

Edu-Hub 平台主要涉及三类用户角色，各角色的职责如下：

- **学生 (Student)**: 系统的主要使用者。可以浏览课程列表、选课报名、观看视频教程、参与在线测试、在评论区互动以及查看个人学习进度。
- **教师 (Teacher)**: 课程内容的提供者。负责创建和管理课程、上传教学视频、发布考试题目、批改主观题（如有）、回复学生提问以及查看班级学习数据。
- **管理员 (Administrator)**: 系统的维护者。拥有最高权限，负责用户账号管理（如封禁违规用户）、全站课程审核、系统配置以及数据备份。

2.1.2 主体业务需求

Edu-Hub 平台的核心业务流程围绕“教”与“学”展开，具体需求如下：

1. **用户管理业务**: 学生用户仅需提供用户名和密码即可完成注册。登录时，后端验证账号密码并颁发 JWT 令牌。前端根据令牌中的角色字段 (Role) 动态渲染菜单权限。
2. **课程管理业务**: 教师在后台创建课程，填写标题、简介（富文本），并上传封面图。课程创建后，教师可添加章节，每个章节支持上传视频文件。视频上传后需经服务器处理存储。
3. **视频学习业务**: 学生进入课程详情页，点击章节视频进行播放。系统需记录播放进度，支持断点续播。
4. **在线测试业务**: 教师在题库中添加题目（单选、多选、判断），并组卷发布。学生在规定时间内答题，提交后系统自动比对答案并计算客观题得分。（注：部分功能待完善）

5. **社区互动业务:** 每个课程下方设有评论区，学生可发表疑问或心得，教师和其他学生可进行回复。支持即时显示新消息。
6. **数据分析业务:** 系统自动采集学生的学习行为（视频观看时长、测试成绩），通过可视化图表（如柱状图、饼图）向教师展示课程的及格率、活跃度等指标。（注：部分功能待完善）

注意：安全需求是业务需求的一部分。

2.1.3 质量需求

质量需求表现为软件满足预期的程度：

- **性能需求：**

- 视频首屏加载时间不超过 2 秒（在正常网络环境下）。
- 系统支持至少 100 人同时在线并发访问而不崩溃。
- API 接口平均响应时间低于 500ms。

- **安全性需求：**

- 所有敏感数据（如密码）需经过哈希加盐存储 [TODO: 待实现]。
- API 接口需进行 JWT 身份验证，防止未授权访问。
- 文件上传需校验格式和大小，防止恶意脚本注入。

- **可靠性需求：**

- 系统应具备 99.9% 的可用性。
- 关键数据（如成绩、课程进度）需定期自动备份。

注意：性能需求是质量需求的一部分。

2.1.4 其它（约束性）需求

指软件开发过程、运行环境应满足的（非业务）约束：

- **开发周期：**本项目需在课程设计规定的 2 周时间内完成开发与测试。
- **技术约束：**前端必须使用 Vue.js 3 + Tailwind CSS，后端采用 Node.js (Express) + MySQL 8.0。
- **运行环境：**服务端需部署在 Linux/Windows 服务器上，客户端需支持主流浏览器（Chrome, Edge, Firefox）。

注意：通常把接口需求单独列为一类需求。

2.2 用例分析（功能需求详细分析）

本系统主要包含学生、教师、管理员三类角色。根据业务需求分析，各角色的核心用例设计如下。

2.2.1 学生用例视图

学生是系统的主要使用者，其核心需求围绕“课程学习”与“社区互动”展开。

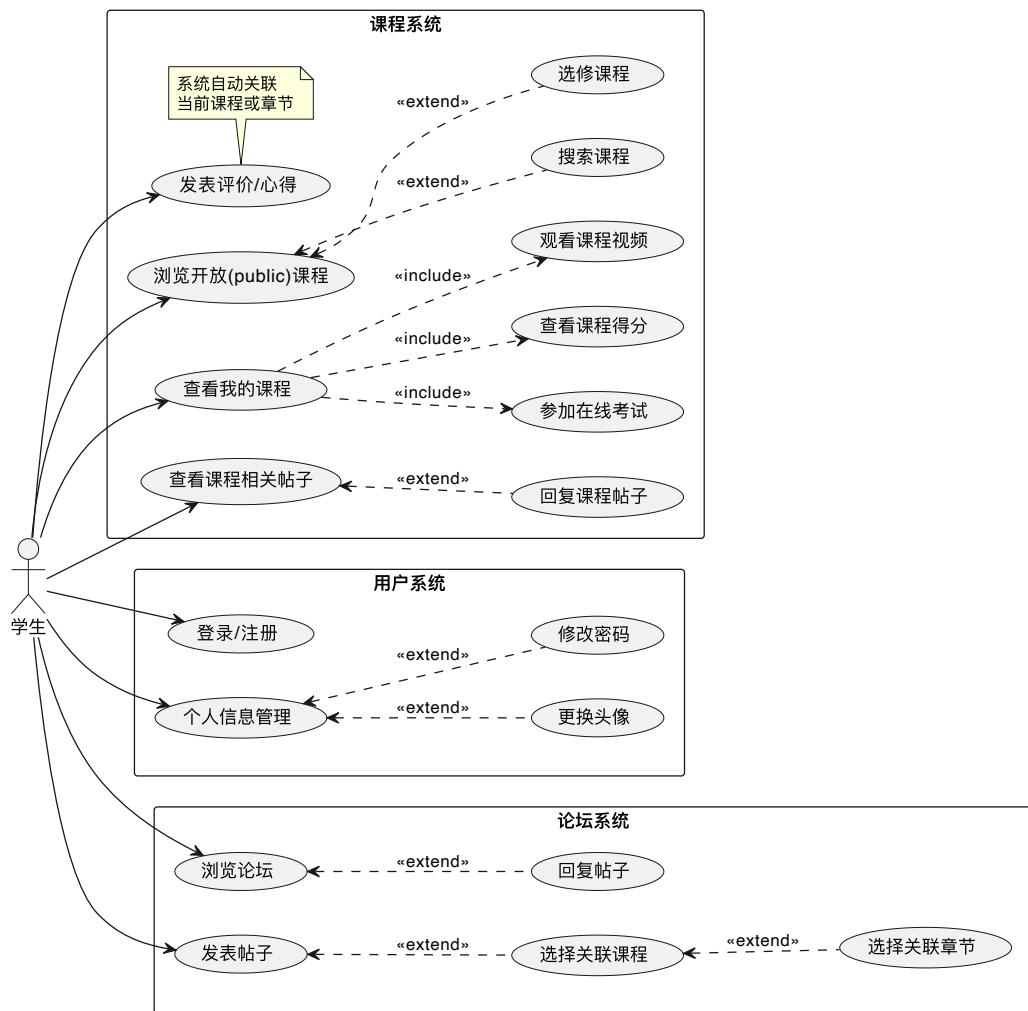


图 2.1: 学生用例图

主要用例包括：

1. **用户系统**: 登录/注册、个人信息管理（修改头像/密码）。
2. **课程系统**:

- **选课前**: 浏览开放课程、搜索课程 [待完善]、选修课程。
 - **选课后**: 查看我的课程、观看视频、参加考试 [待完善]、查看得分。
3. **论坛系统**: 浏览论坛、发表帖子 (关联课程/章节)、回复帖子。

2.2.2 教师用例视图

教师是教学资源的生产者，负责课程的建设与维护。

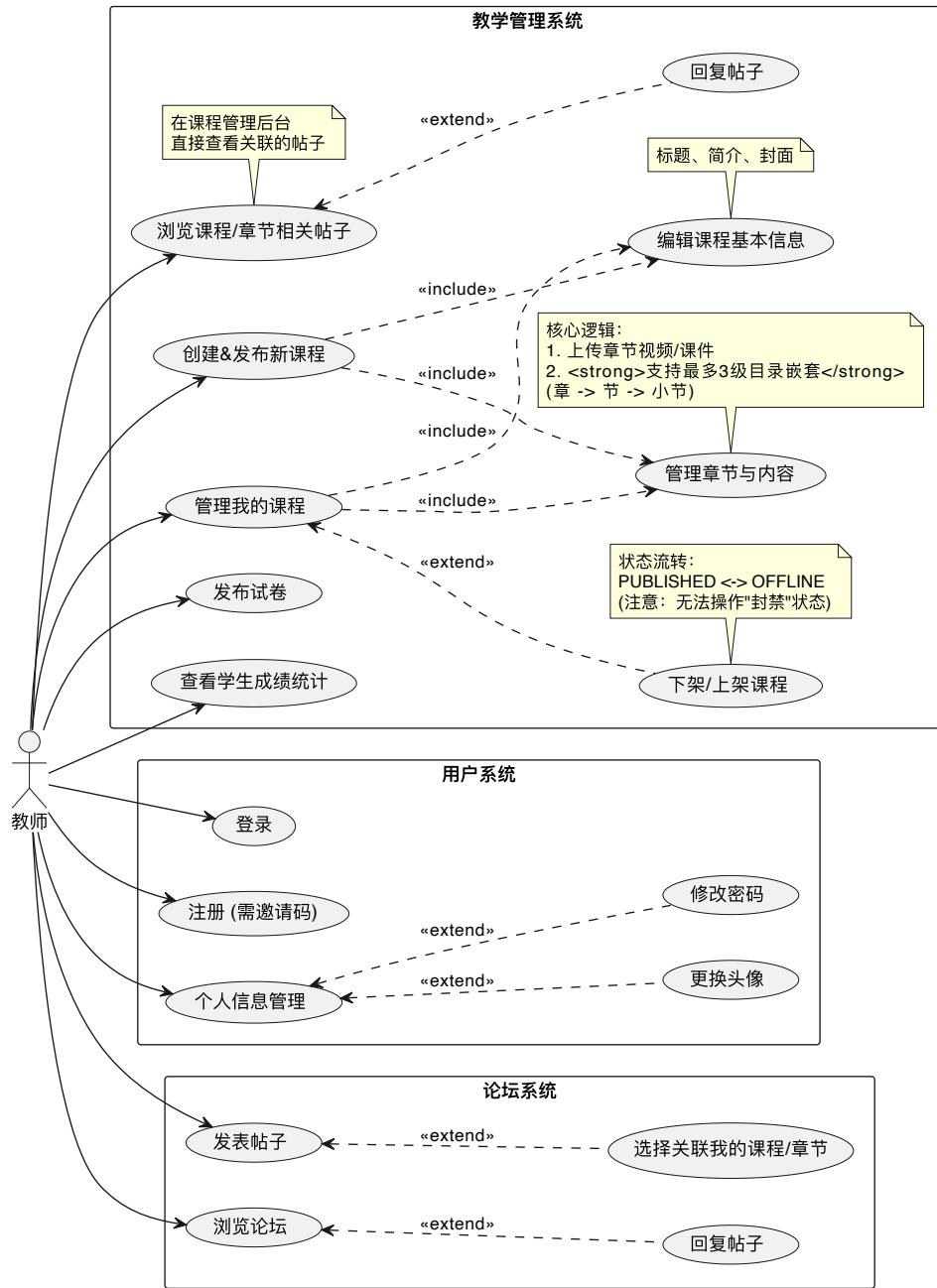


图 2.2: 教师用例图

主要用例包括：

1. **用户系统**: 注册 (需邀请码)、登录、个人信息管理。
2. **教学管理**:
 - **课程生产**: 查看已创建课程列表、创建/发布课程、编辑课程信息、管理章节 (支持三级目录)、上传视频/课件。

- **课程维护:** 下架/上架课程 [下架功能待完善]。
 - **数据查看:** 查看学生成绩统计 [待完善]。
3. **互动管理:** 在课程后台直接查看并回复关联的帖子。

2.2.3 管理员用例视图

管理员负责系统的整体运维与监管。

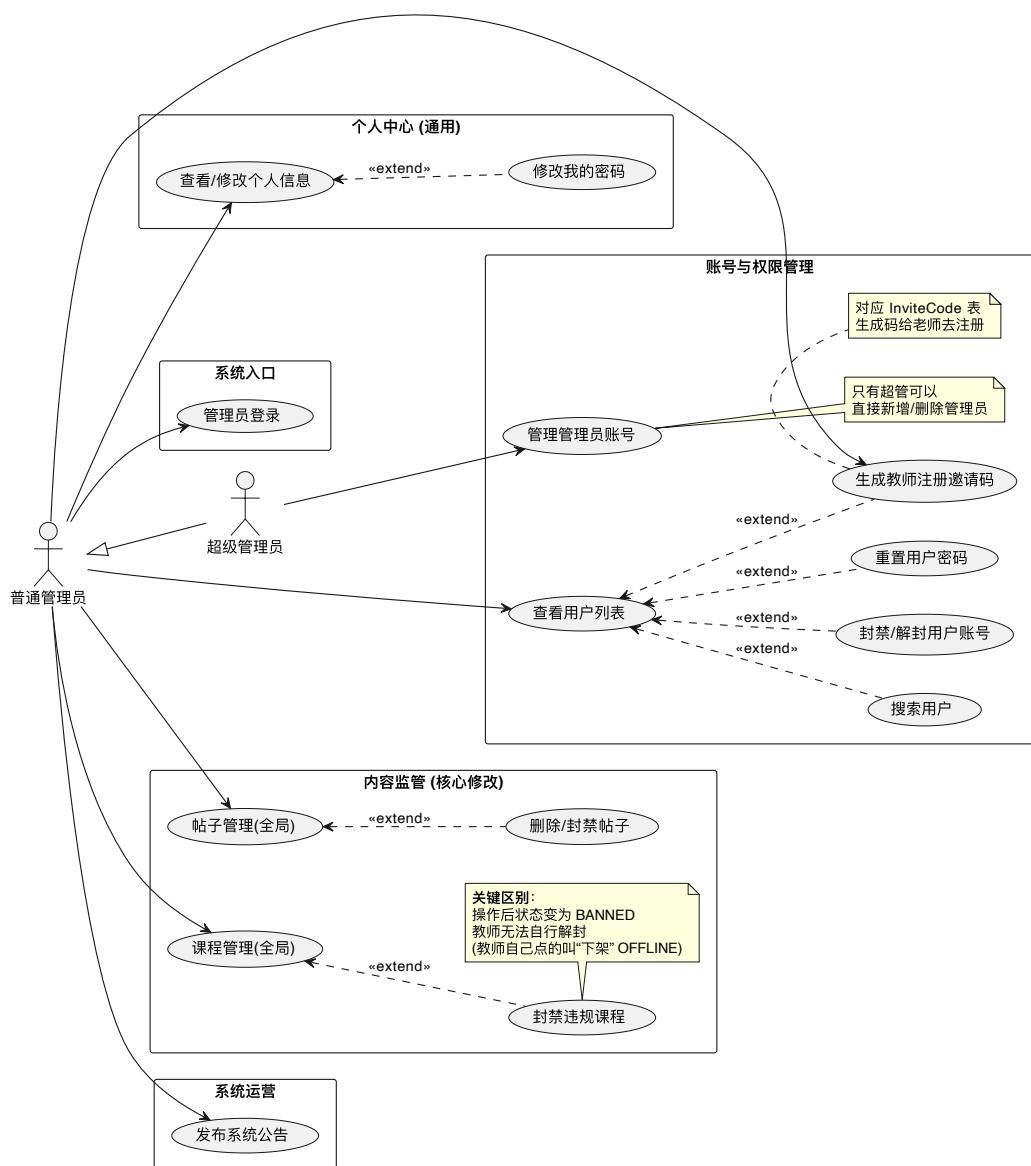


图 2.3: 管理员用例图

主要用例包括:

1. **账号管理:** 生成教师邀请码、查看/搜索用户列表、封禁/解封违规账号、重置用户密码。
2. **内容监管:** 全局课程管理 [待完善]、封禁违规课程 [待完善]。
3. **个人中心:** 修改管理员自身密码 [待完善]。

此后需求分析部分分别对各用例展开详尽的描述。

2.3 用户注册与登录

本用例描述用户（学生、教师、管理员）进入系统并获取身份授权的过程。

2.3.1 用例中的类

参与类包括: `User` (用户实体), `InviteCode` (邀请码实体), `AuthController` (认证控制器)。

2.3.2 关键的类及状态

- `User`: 状态包括 `ACTIVE` (活跃), `BANNED` (封禁)。
- `InviteCode`: 状态包括 `USED` (已使用), `UNUSED` (未使用)。

2.3.3 用例主体行为

1. **进入页面:** 用户访问系统首页，点击“登录”或“注册”。
2. **提交信息:**
 - **注册:** 输入用户名、密码、昵称。(教师角色需输入有效邀请码)。
 - **登录:** 输入用户名、密码。
3. **后端验证:**
 - 系统校验用户名是否唯一。
 - 系统校验密码正确性 (注册时校验格式, 登录时校验匹配)。
 - (仅教师注册) 系统校验邀请码是否存在且未过期。
4. **授权响应:** 验证通过后，服务器生成 JWT Token 并返回给客户端。
5. **跳转:** 前端将 Token 存入 LocalStorage，并根据用户角色 (Role) 跳转至相应 Dashboard。

2.3.4 用例异常行为

- **用户名已存在**: 注册时提示“用户名已被占用”。
- **邀请码无效**: 教师注册时提示“邀请码无效或已过期”。
- **凭据错误**: 登录时提示“用户名或密码错误”。

2.4 课程发布与管理

本用例描述教师创建新课程并发布内容的完整流程。

2.4.1 用例中的类

参与类包括: Course (课程), Chapter (章节), User (教师)。

2.4.2 关键的类及状态

- **Course**: 状态流转: DRAFT (草稿) → PUBLISHED (已发布)。只有 PUBLISHED 状态的课程对学生可见。

2.4.3 用例主体行为

1. **创建课程**: 教师点击“新建课程”，填写标题、简介、上传封面图，设置课程类型（公开/私有）。
2. **构建大纲**: 教师添加章节（Chapter），支持多级目录结构。
3. **上传资源**: 教师在章节下上传教学视频文件。
4. **发布课程**: 教师确认课程内容无误后，点击“发布”按钮。
5. **系统处理**: 系统将课程状态更新为 PUBLISHED，并在课程大厅展示。

2.4.4 用例异常行为

- **空章节发布**: 若课程内无任何章节，系统阻止发布并提示“请至少添加一个章节”。
- **上传失败**: 若视频文件格式不支持或过大，系统提示上传失败。

2.5 视频学习

本用例描述学生进入课程并观看教学视频的学习过程。

2.5.1 用例中的类

参与类包括: Enrollment (选课记录), ChapterProgress (章节进度), Course (课程)。

2.5.2 关键的类及状态

- ChapterProgress: 记录用户的观看进度(progress)和完成状态(is_completed)。

2.5.3 用例主体行为

1. **进入课程:** 学生在“我的课程”列表中点击进入课程详情页。
2. **选择章节:** 在左侧章节树中点击带有视频标识的节点。
3. **播放视频:** 系统加载视频播放器, 自动跳转至上次观看的断点。
4. **进度同步:** 播放过程中, 前端定时向后端发送心跳包, 更新 ChapterProgress。
5. **完成学习:** 视频播放结束, 系统自动标记该章节为“已完成”。

2.5.4 用例异常行为

- **未选课访问:** 若用户未选修该课程, 系统拒绝访问视频并提示“请先加入课程”。

2.6 社区互动

本用例描述用户在课程讨论区发表观点和互动的过程。

2.6.1 用例中的类

参与类包括: Post (帖子), Comment (评论), User (作者)。

2.6.2 关键的类及状态

- Post: 关联特定的 Course 或 Chapter。

2.6.3 用例主体行为

1. **浏览帖子:** 用户进入课程讨论区, 查看帖子列表。
2. **发表帖子:** 用户点击“发帖”, 输入标题和内容, 点击提交。

3. **回复互动**: 用户进入帖子详情，在底部输入框填写评论并提交。

4. **展示更新**: 列表自动刷新，显示最新的帖子和评论。

2.6.4 用例异常行为

- **内容违规**: (预留) 若包含敏感词，系统拦截发布。

2.7 完整的类图

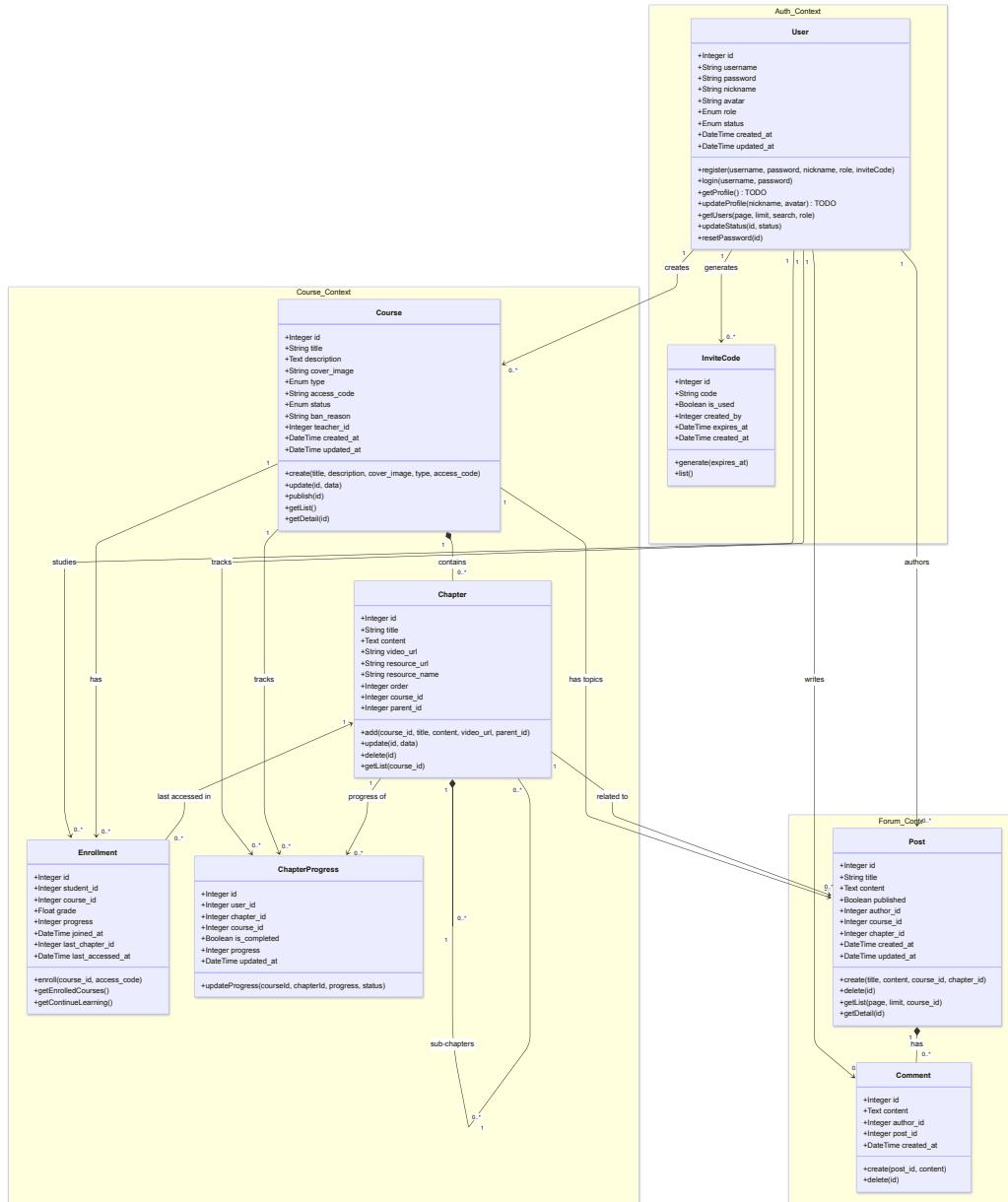


图 2.4: 系统完整类图

2.8 用户验收测试计划

按照 W 模型的观点，在需求分析结束时，应及时规划测试工程，并展开对测试需求的分析，设计确认（用户验收）测试的流程、数据（用例）等。

在行文时，可归并到单独的测试章节。

第三章 概要设计

本系统采用经典的前后端分离架构（B/S 模式），前端基于 Vue.js 3 生态构建单页应用（SPA），后端基于 Node.js（Express）提供 RESTful API 服务，数据库采用 MySQL 8.0。这种架构实现了视图层与业务逻辑层的完全解耦，便于团队并行开发与后期维护。

3.1 系统分解（体系结构）

系统整体架构分为三个主要层次：表现层（Frontend）、业务逻辑层（Backend）和数据持久层（Database）。

3.1.1 架构图

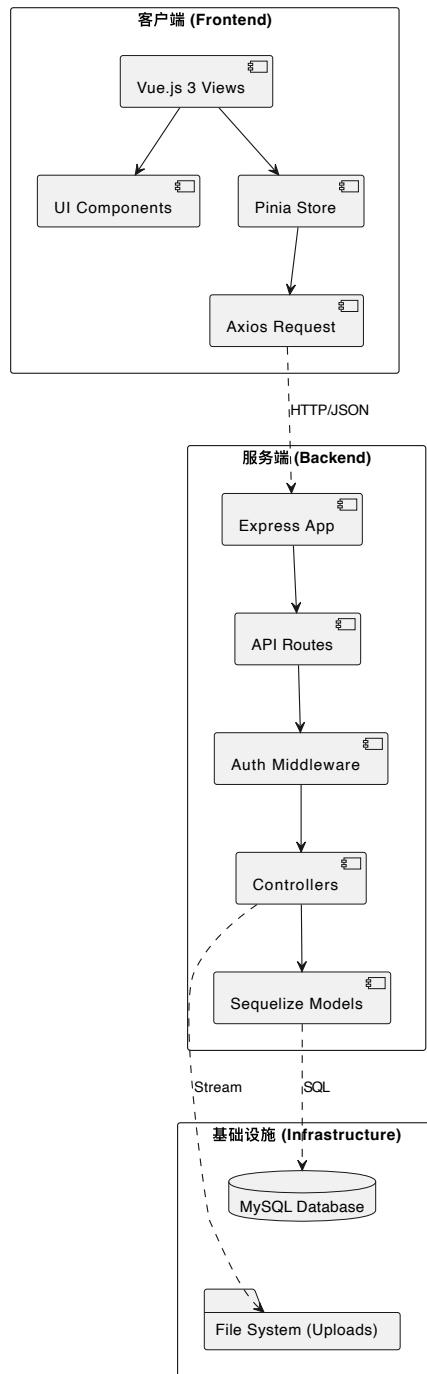


图 3.1: 系统架构图

3.1.2 子系统职责定义

1. 前端子系统 (frontend/)

- **视图层 (views/)**: 负责页面的结构展示。核心模块包括 auth (认证)、course (课程展示)、dashboard (后台管理)。
- **组件层 (components/)**: 封装可复用的 UI 元素，如 ChapterEditor.vue (章节编辑器) 和通用布局组件。
- **状态管理 (stores/)**: 使用 Pinia 管理全局状态。例如 auth.js 存储用户信息和 JWT Token, course.js 缓存课程列表数据。
- **网络层 (utils/request.js)**: 基于 Axios 封装 HTTP 请求，统一处理请求拦截（注入 Token）和响应拦截（错误处理）。

2. 后端子系统 (backend/)

- **接口层 (routes/)**: 定义 API 路由端点。例如 authRoutes.js 处理登录注册, courseRoutes.js 处理课程增删改查。
- **业务层 (controllers/)**: 实现核心业务逻辑。例如 courseController.js 负责解析上传的视频文件、验证数据完整性并调用模型。
- **中间件 (middleware/)**: authMiddleware.js 负责解析 HTTP Header 中的 JWT 令牌，验证用户身份并拦截非法请求。
- **数据层 (models/)**: 使用 Sequelize 定义数据模型 (Schema)，如 User.js, Course.js，自动映射到 MySQL 表结构。

3. 数据存储子系统

- **MySQL**: 存储结构化数据 (用户、课程元数据、评论等)。
- **本地文件系统 (uploads/)**: 存储非结构化数据 (用户头像、课程封面、教学视频)，通过 Express 的静态资源中间件对外提供访问。

3.2 系统总体流程

本系统采用 RESTful 风格进行通信。以下以“教师发布课程”为例，描述跨子系统的数据流转过程（采用异步上传模式）：

1. **封面上传**: 教师在前端选择封面图片，组件调用通用上传接口 POST /api/upload。
 - 后端 uploadController 接收文件流，保存至 backend/uploads/，并返回文件访问 URL。
2. **发起建课请求**: 教师填写课程信息 (包含上一步获取的封面 URL)，点击发布。Vue 组件调用 courseStore.createCourse()。

3. 网络传输: Axios 拦截请求, 在 Header 中添加 Authorization: Bearer <token>, 并将课程数据封装为 JSON 格式发送至 POST /api/courses。
4. 路由分发: Express 接收请求, authMiddleware 验证 Token 有效性。若通过, 将请求转发至 courseController.createCourse。
5. 业务处理:
 - Controller 从 req.body 中提取 title, description, cover_image (URL) 等字段。
 - Controller 验证必填字段非空。
 - Controller 调用 Course.create() 模型方法。
6. 数据持久化: Sequelize 生成 INSERT INTO courses ... SQL 语句并在 MySQL 中执行。
7. 响应反馈: 数据库返回新创建的课程对象, 后端封装 JSON 响应返回给前端。
8. 界面更新: 前端接收响应, Pinia 更新本地课程列表状态, 路由跳转至课程管理页。

3.3 界面设计原则（交互风格规范）

前端界面设计遵循“简洁、响应式、组件化”的原则, 主要依托 Tailwind CSS 框架实现。

1. 原子化样式: 使用 Tailwind 的原子类 (如 p-4, flex, text-blue-500) 构建样式, 确保全站间距、颜色、字体的一致性, 避免传统 CSS 的样式冲突。
2. 主题定制: 引入 Catppuccin 主题配色方案, 为系统提供了一套柔和、低对比度的色彩风格, 既美观又护眼, 提升了用户的视觉体验。
3. 响应式布局: 采用 Mobile-First 策略, 利用 md:, lg: 前缀适配不同屏幕尺寸。例如, 课程列表在移动端为单列, 桌面端自动切换为网格布局。
4. 交互反馈:
 - 加载状态: 数据请求期间展示 Loading 骨架屏或旋转图标。
 - 操作反馈: 使用 useToast.js 封装的全局消息组件, 在操作成功或失败时在页面顶部弹出自动消失的提示框。
5. 组件复用: 将通用的功能模块 (如“文件上传区域”、“富文本编辑器容器”) 封装为独立组件, 保持代码整洁。

3.4 界面原型设计

基于上述设计原则，本系统设计了以下核心界面原型，以直观展示系统的功能布局与交互逻辑。（说明：因开发时间有限，原型图为低保真的色块区分布局。）

3.4.1 登录与注册页



The figure displays two wireframe prototypes for the login and registration pages of EduHub.

(a) 登录页 (Login Page): The title is "Sign in to EduHub". It contains fields for "Username" and "Password", and a "Sign In" button. Below the form is a link: "New to EduHub? Create an account".

(b) 注册页 (Registration Page): The title is "Register to EduHub". It contains fields for "Username", "Password", and "Password Confirm", and a "Register" button. Below the form is a link: "Have account already? Login in to EduHub".

图 3.2: 认证页面原型

3.4.2 主页 (Home)

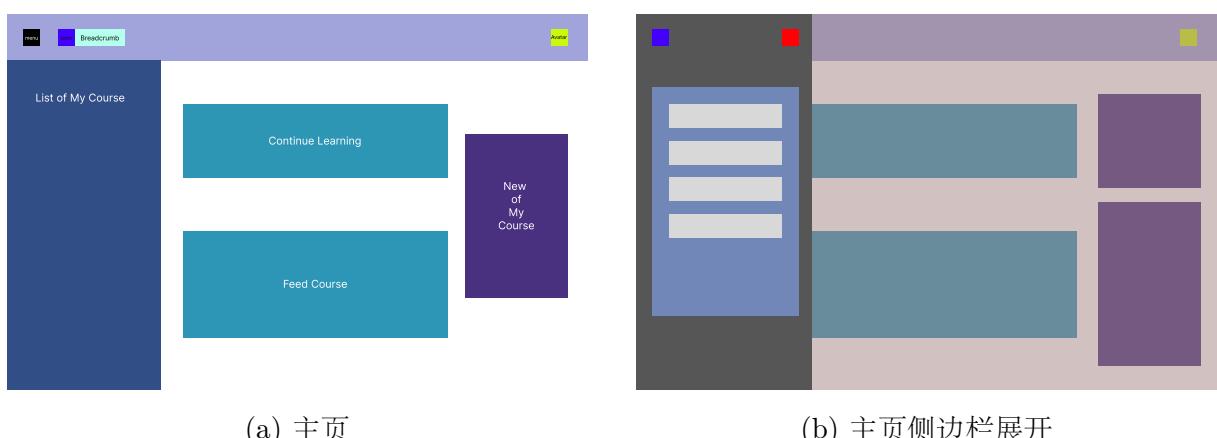


图 3.3: 主页原型

3.4.3 课程广场

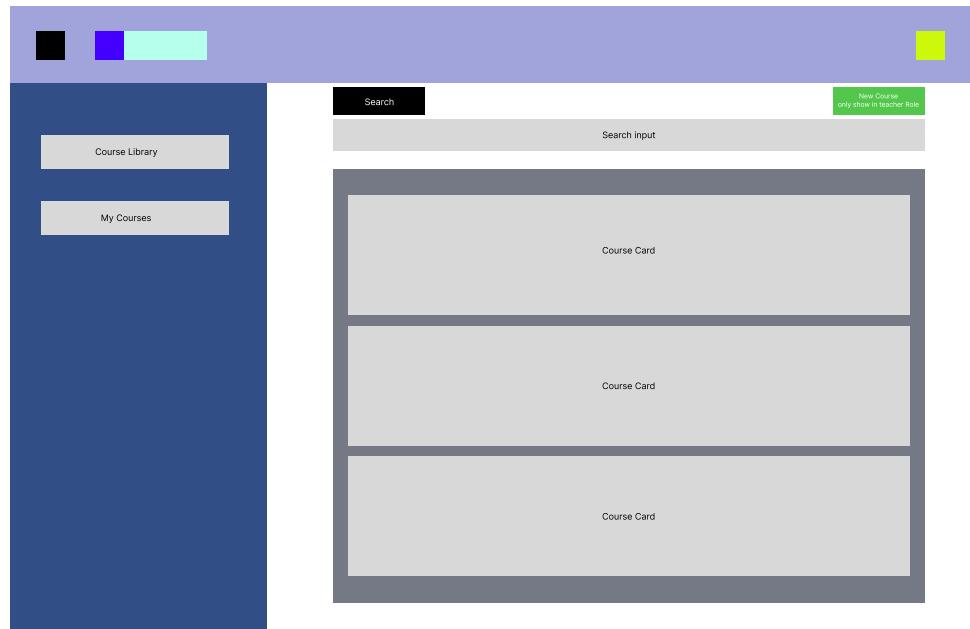


图 3.4: 课程广场原型

3.4.4 课程学习页



图 3.5: 课程学习页原型

3.4.5 课程编辑器

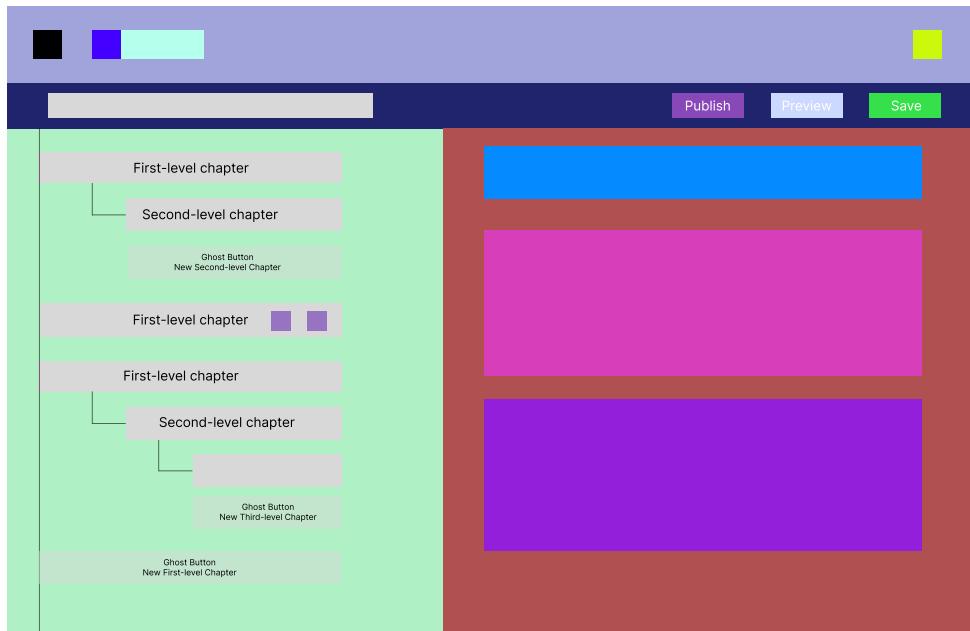


图 3.6: 课程编辑器原型

3.5 系统集成测试计划与设计

集成测试旨在验证前后端接口的连通性及业务流程的整体性。

1. 接口测试 (API Testing)

- **工具:** Apifox。
- **策略:** 独立测试后端 API。重点测试 `authRoutes` 的 Token 生成与验证，以及 `uploadRoutes` 的文件上传功能（边界条件：文件大小限制、格式限制）。

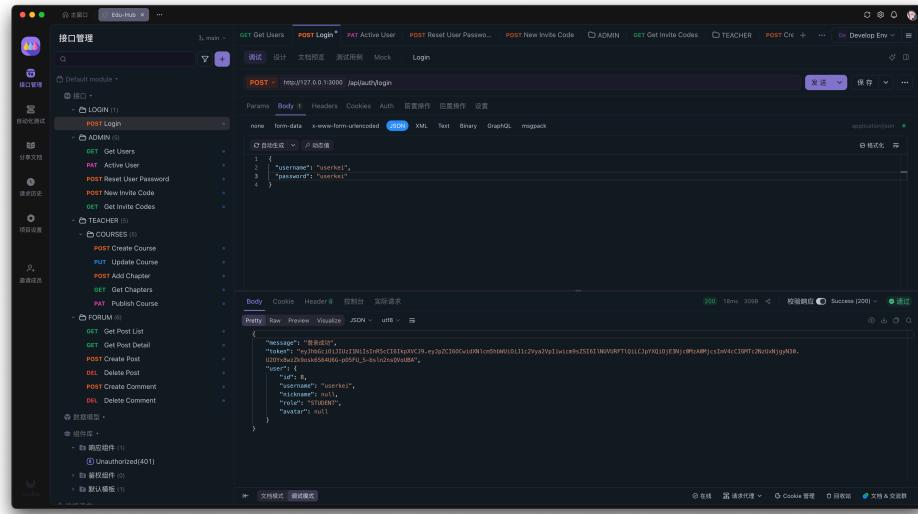


图 3.7: Apifox 接口调试示例

2. 前后端联调 (Integration)

- 环境:** 前端运行于 `localhost:5173`, 后端运行于 `localhost:3000`。
- 测试场景:**
 - 注册登录流:** 注册新账号 → 自动跳转登录 → 登录获取 Token → 访问受保护的 Dashboard。
 - 课程生命周期:** 创建课程 → 添加章节 → 上传视频 → 学生选课 → 观看视频（进度更新）。

3. 异常处理测试

- 模拟网络断开或后端宕机，验证前端 Axios 拦截器是否能正确捕获错误并提示用户。
- 模拟 Token 过期，验证前端是否能自动清除本地存储并跳转至登录页。

第四章 详细设计

本章基于实际代码结构（Reverse Engineering）对系统的核心数据模型、关键业务逻辑及算法进行详细描述。

4.1 类（数据）设计

系统采用 Sequelize ORM 框架定义数据模型，底层映射为 MySQL 关系型数据库表。以下为核心物理数据模型设计。

4.1.1 数据库物理模型

系统数据库的实体关系图（ER 图）如下图所示，清晰展示了用户、课程、章节、进度、评论等核心实体及其之间的关联关系（如一对多、多对多等）。

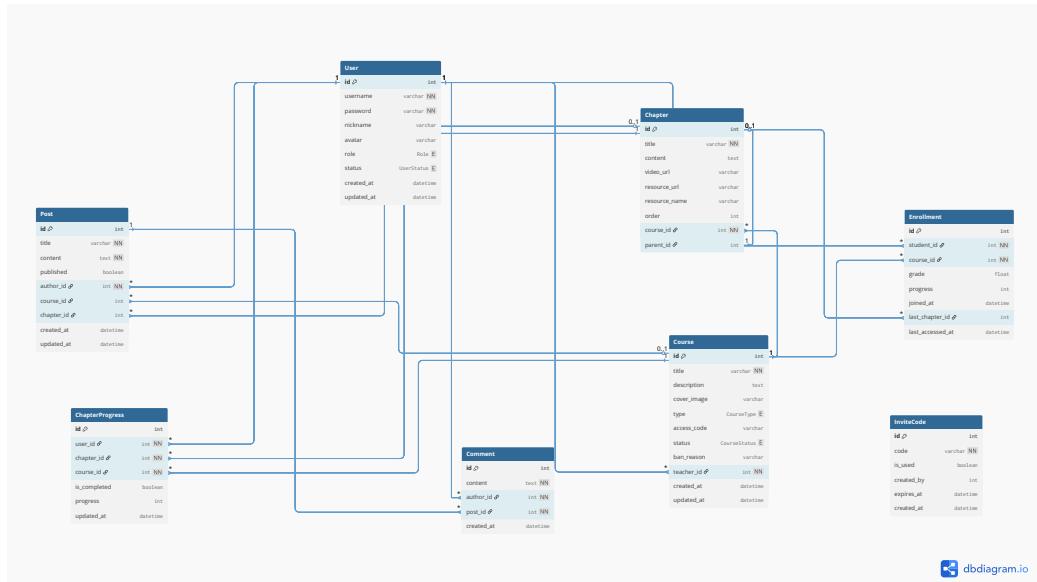


图 4.1: 数据库 ER 图

数据表定义

表 4.1: 用户表 (User)

字段名	类型	必填	默认值	描述
id	INTEGER	是	Auto Inc	主键
username	STRING	是	-	登录账号 (唯一索引)
password	STRING	是	-	登录密码 (当前版本暂存明文, 建议升级为 Bcrypt)
nickname	STRING	否	-	用户昵称
role	ENUM	是	STUDENT	角色: STUDENT, TEACHER, ADMIN
status	ENUM	是	ACTIVE	状态: ACTIVE, BANNED
created_at	DATETIME	是	Now	创建时间

表 4.2: 课程表 (Course)

字段名	类型	必填	默认值	描述
id	INTEGER	是	Auto Inc	主键
title	STRING	是	-	课程标题
description	TEXT	否	-	课程简介
cover_image	STRING	否	-	封面图片 URL
type	ENUM	是	PUBLIC	类型: PUBLIC (公开), PRIVATE (私有)
status	ENUM	是	DRAFT	状态: DRAFT, PUBLISHED, BANNED
teacher_id	INTEGER	是	-	外键 → User.id

表 4.3: 章节表 (Chapter)

字段名	类型	必填	默认值	描述
id	INTEGER	是	Auto Inc	主键
course_id	INTEGER	是	-	外键 → Course.id
parent_id	INTEGER	否	NULL	父章节 ID (实现树形结构)
title	STRING	是	-	章节标题
type	VIRTUAL	-	-	逻辑字段: FOLDER (目录) / FILE (内容)
video_url	STRING	否	-	视频文件地址
resource_url	STRING	否	-	课件下载地址
order	INTEGER	是	0	排序权重

表 4.4: 学习进度表 (ChapterProgress)

字段名	类型	必填	默认值	描述
id	INTEGER	是	Auto Inc	主键
user_id	INTEGER	是	-	外键 → User.id
chapter_id	INTEGER	是	-	外键 → Chapter.id
course_id	INTEGER	是	-	外键 → Course.id (冗余字段优化查询)
progress	INTEGER	是	0	观看进度 (0-100)
is_completed	BOOLEAN	是	false	是否完成

表 4.5: 选课记录表 (Enrollment)

字段名	类型	必填	默认值	描述
user_id	INTEGER	是	-	外键 → User.id
course_id	INTEGER	是	-	外键 → Course.id
progress	INTEGER	是	0	课程总进度 (0-100)
last_chapter_id	INTEGER	否	-	上次学习章节 ID

表 4.6: 邀请码表 (InviteCode)

字段名	类型	必填	默认值	描述
id	INTEGER	是	Auto Inc	主键
code	STRING	是	-	邀请码 (唯一)
is_used	BOOLEAN	是	false	是否已使用
created_by	INTEGER	否	-	创建者 ID (管理员)
expires_at	DATETIME	否	-	过期时间

表 4.7: 帖子表 (Post)

字段名	类型	必填	默认值	描述
id	INTEGER	是	Auto Inc	主键
title	STRING	是	-	帖子标题
content	TEXT	是	-	帖子内容
author_id	INTEGER	是	-	外键 → User.id
course_id	INTEGER	否	-	关联课程 ID
chapter_id	INTEGER	否	-	关联章节 ID
published	BOOLEAN	是	true	是否发布

表 4.8: 评论表 (Comment)

字段名	类型	必填	默认值	描述
id	INTEGER	是	Auto Inc	主键
content	TEXT	是	-	评论内容
author_id	INTEGER	是	-	外键 → User.id

字段名	类型	必填	默认值	描述
post_id	INTEGER	是	-	外键 → Post.id
created_at	DATETIME	是	Now	创建时间

4.1.2 设计优化与考量

1. 树形结构存储优化:

- **方案:** 采用 Adjacency List (邻接表) 模式，即在 Chapter 表中使用 parent_id 指向父节点。
- **考量:** 虽然 Nested Sets 模型查询效率更高，但考虑到课程章节层级通常较浅（一般不超过 4 层），邻接表模式在写入（拖拽排序、移动章节）时性能更优且实现简单。
- **性能瓶颈:** 若章节数过多，递归查询可能导致性能下降。
- **优化策略:** 在应用层（Node.js）一次性拉取某课程所有章节，在内存中组装树形结构，避免 N+1 查询问题。

2. 进度计算性能优化:

- **问题:** 实时计算课程总进度需要聚合所有章节的完成情况，数据库压力大。
- **方案:** 在 Enrollment 表中增加 progress 冗余字段。
- **逻辑:** 每次更新 ChapterProgress 时，触发异步任务重新计算总进度并更新 Enrollment.progress。读取时直接读取 Enrollment 表，实现 O(1) 复杂度查询。

3. 安全设计 (现状与改进):

- **现状:** 当前代码中密码采用明文存储 (password 字段)，存在极高安全风险。
- **改进计划:** 引入 bcryptjs 库。在 User.create 钩子中对密码进行加盐哈希处理。
- **鉴权:** 全站采用 JWT (JSON Web Token) 无状态认证，Token 载荷包含 id 和 role，避免了 Session 存储的内存开销，利于水平扩展。

4.2 关键逻辑、算法设计

4.2.1 章节树构建算法 (Recursive Tree Building)

后端从数据库获取扁平的章节列表后，需将其转换为前端可渲染的树形结构。

- 算法逻辑:

1. 获取该课程所有章节 `chapters`, 按 `order` 升序排列。
2. 定义递归函数 `buildTree(parentId)`。
3. 过滤出 `parent_id === parentId` 的节点作为当前层级节点。
4. 对每个节点, 递归调用 `buildTree(node.id)` 获取其 `children`。
5. **类型推断:** 若 `children.length > 0`, 强制标记 `type = 'FOLDER'`; 否则标记为 `FILE`。

- 代码实现 (Node.js):

```

1 const buildChapterTree = (chapters,
2   parentId = null) => {
3     return chapters
4       .filter(chapter =>
5         chapter.parent_id === parentId)
6       .map(chapter => {
7         const children = buildChapterTree(chapters, chapter.id);
8         return {
9           ...chapter,
10          type: children.length > 0 ? 'FOLDER' : 'FILE',
11          children
12        };
13      });
14    };

```

Listing 4.1: 章节树构建算法

4.2.2 课程总进度计算算法

当用户更新某一章节的进度时, 系统需实时计算该课程的总体完成度。

- **算法挑战:** 课程包含多级目录, 目录本身不参与进度计算, 仅“叶子节点”(实际视频章节) 参与。
- **算法流程:**

1. 识别叶子节点:

- 获取课程所有章节集合 C 。
- 提取所有 `parent_id` 组成集合 P 。
- 叶子节点集合 $L = \{c \in C \mid c.id \notin P\}$ 。

- 计算总任务数 $Total = |L|$ 。

2. 统计已完成数:

- 查询 ChapterProgress 表, 条件: `user_id = current_user AND is_completed = true AND chapter_id IN L`。
- 得到已完成数 $Completed$ 。

3. 计算百分比:

$$Progress = \text{Math.round} \left(\frac{Completed}{Total} \times 100 \right) \quad (4.1)$$

4. 原子更新:

- 更新 Enrollment 表中的 `progress` 字段。

4.2.3 权限控制逻辑 (Middleware Strategy)

系统采用基于角色的访问控制 (RBAC) 模型, 通过中间件拦截请求。

- 逻辑流程:

1. Token 解析: `verifyToken` 中间件从 Header 解析 Bearer Token。
2. 身份注入: 解析成功后, 将用户信息挂载至 `req.user`。
3. 角色守卫:
 - `requireAdmin`: 检查 `req.user.role === 'ADMIN' 或 SUPER_ADMIN`。
 - `requireTeacher`: [TODO: 待实现] 计划增加检查 `req.user.role === 'TEACHER'` 的中间件。

4. 资源守卫 (如课程修改):

TODO: 待实现 目前 Controller 尚未强制检查

`course.teacher_id === req.user.id`, 计划在下一版本中修复此越权风险。

4.2.4 基于角色的数据展示逻辑 (Role-Based Data Presentation)

系统根据用户角色的不同, 在“我的课程”页面展示不同的数据视图, 以满足不同角色的业务需求。

- 逻辑流程:

1. 前端判断: 页面加载时, 首先检查 `authStore.user.role`。

2. 教师视图 (TEACHER):

- **需求:** 查看自己创建的课程，以便进行管理和编辑。
- **API 调用:** 调用 GET /api/courses/created。
- **后端逻辑:** 查询 Course 表，筛选条件为
`teacher_id = current_user.id`。

3. 学生视图 (STUDENT):

- **需求:** 查看自己已选修的课程，以便进行学习。
- **API 调用:** 调用 GET /api/courses/enrolled。
- **后端逻辑:** 查询 Enrollment 表，筛选条件为
`student_id = current_user.id`，并关联查询 Course 信息。

4. 界面渲染: 前端根据返回的数据结构差异（课程列表 vs 选课记录列表）进行适配，统一渲染为课程卡片。

4.3 模块接口测试计划

本系统的测试计划旨在验证各个核心业务模块接口的正确性与健壮性。测试采用“白盒测试”与“黑盒测试”相结合的方式，针对典型业务场景设计测试用例。

4.3.1 用户认证模块 (Auth)

- **场景 1: 用户注册**

- **输入:** `{ username: "newuser", password: "password123" }`
- **预期:** 若用户名未存在，返回状态码 201 及用户信息；若已存在，返回 400 及错误提示。

- **场景 2: 用户登录**

- **输入:** `{ username: "validUser", password: "validPass" }`
- **预期:** 验证通过，返回状态码 200 及 JWT Token。

4.3.2 课程管理模块 (Course)

- **场景 1: 创建课程**

- **输入:** Header 携带有效 Token，
Body `{ title: "Vue3 入门", type: "PUBLIC" }`

- 预期：返回状态码 201 及新创建的课程对象。

- **场景 2：获取公开课程列表**

- 输入：GET /api/courses
- 预期：返回状态码 200 及课程数组 JSON。

- **场景 3：获取教师创建课程列表**

- 输入：Header 携带教师 Token, GET /api/courses/created
- 预期：返回状态码 200，仅包含该教师创建的课程。

4.3.3 章节与进度模块 (Chapter)

- **场景 1：更新学习进度**

- 输入：POST /progress, Body { chapterId: 1, progress: 100 }
- 预期：返回状态码 200，且数据库中该章节状态变更为“已完成”。

4.3.4 论坛模块 (Forum)

- **场景 1：发布帖子**

- 输入：{ title: " 提问", content: " 如何运行?", courseId: 1 }
- 预期：返回状态码 201，帖子成功关联至指定课程。

第五章 编码实现

5.1 编码工具、技术介绍

本项目采用现代化的全栈开发流程，依托于成熟的开源生态系统。

5.1.1 开发环境与工具

- 操作系统: macOS / Windows 10+ / Linux
- IDE: Visual Studio Code
 - 核心插件: Vue - Official (Volar), ESLint, Prettier, Tailwind CSS IntelliSense
- 运行环境:
 - Node.js: v20.19.0+ (后端运行时及前端构建环境)
 - MySQL: 8.0 (关系型数据库)
- 版本控制: Git (代码版本管理)
- API 调试: Apifox

5.1.2 后端技术栈 (Backend)

- 核心框架: Express v5.2.1 - 下一代 Node.js Web 框架，提供更强的路由处理能力。
- ORM 框架: Sequelize v6.37 - 基于 Promise 的 Node.js ORM，支持事务、关联查询、预加载等高级特性。
- 数据库驱动: mysql2 - 高性能 MySQL 客户端。
- 认证授权: jsonwebtoken (JWT) - 实现无状态的身份验证。
- 文件处理: multer - 处理 multipart/form-data 类型的表单数据（文件上传）。
- 跨域处理: cors - 解决前后端分离架构下的跨域资源共享问题。

5.1.3 前端技术栈 (Frontend)

- **构建工具:** Vite v7.3.0 - 基于 ES Modules 的极速构建工具，提供毫秒级的热更新 (HMR) 体验。
- **核心框架:** Vue v3.5.26 - 采用组合式 API (Composition API) 编写，逻辑复用性更强。
- **状态管理:** Pinia v3.0.4 - Vue 官方推荐的新一代状态管理库，去除了 Mutation，API 更简洁。
- **路由管理:** Vue Router v4.6.4 - 管理单页应用的路由跳转与守卫。
- **UI 样式:** Tailwind CSS v4.1.18 - 原子化 CSS 框架，配合 @catppuccin/tailwindcss 主题插件，实现统一的视觉风格。
- **网络请求:** axios - 封装统一的 HTTP 请求客户端，处理拦截器与错误响应。
- **富文本编辑:** tiptap - 无头 (Headless) 富文本编辑器，用于课程简介的编辑。
- **国际化:** vue-i18n - 支持中英文界面切换。

5.2 源代码阅读指南

本节以“逆向工程”的视角，对项目的文件结构与核心配置进行解析，帮助开发者快速定位关键逻辑。

5.2.1 后端工程结构 (backend/)

后端工程遵循经典的 MVC (Model-View-Controller) 分层架构：

- **入口文件:**
 - app.js: 应用程序的引导文件。负责初始化 Express 实例，配置全局中间件 (CORS, JSON 解析, 静态资源托管)，并注册路由模块。
- **配置层 (config/):**
 - database.js: 数据库连接配置文件。实例化 Sequelize 对象，配置数据库主机、端口、用户名及连接池参数。
- **模型层 (models/):**

- 定义数据表结构与关联关系。核心文件包括 `User.js` (用户), `Course.js` (课程), `Chapter.js` (章节), `Enrollment.js` (选课)。
 - 关键逻辑: 在模型文件中定义了 `hasMany`, `belongsTo` 等关联, 这是进行联表查询的基础。
- **控制层 (`controllers/`):**
 - 业务逻辑的核心载体。
 - `authController.js`: 处理注册、登录逻辑, 签发 JWT。
 - `courseController.js`: 处理课程的 CRUD 操作及文件上传后的元数据绑定。
 - `uploadController.js`: 处理文件流,
将文件保存至本地磁盘并返回访问 URL。
 - **路由层 (`routes/`):**
 - 定义 API URL 端点, 并将请求分发给对应的 Controller。
 - `authRoutes.js`: `/api/auth/*`
 - `courseRoutes.js`: `/api/courses/*`
 - **中间件 (`middleware/`):**
 - `authMiddleware.js`: 拦截请求, 验证 Header 中的 `Authorization: Bearer <token>`, 若验证失败则返回 401。

5.2.2 前端工程结构 (`frontend/src/`)

前端工程采用模块化组件结构:

- **入口与配置:**
 - `main.js`: Vue 应用入口。负责挂载 App, 注册 Pinia, Router, i18n 等插件。
 - `App.vue`: 根组件。包含全局的 `<RouterView>` 和 Toast 提示组件。
- **网络层 (`api/ & utils/`):**
 - `utils/request.js`: 二次封装 Axios。
 - * 请求拦截: 自动在 Header 中携带 Token。
 - * 响应拦截: 统一处理 401 (Token 过期) 跳转登录页, 以及全局错误提示。
 - `api/auth.js`: 定义登录、注册等具体的 API 调用函数。

- 状态管理 (`stores/`):
 - `auth.js`: 存储当前用户信息 (`user`) 和令牌 (`token`)，并提供 `login/logout` 动作。
 - `app.js`: 管理全局 UI 状态 (如侧边栏折叠)。
- 页面视图 (`views/`):
 - `auth/`: 登录 (`Login.vue`)、注册 (`Register.vue`) 页面。
 - `dashboard/`: 不同角色的控制台首页 (`StudentDashboard.vue`, `TeacherDashboard.vue`)。
 - `course/`: 课程相关页面，如课程大厅 (`CourseList.vue`)、课程详情 (`CourseDetail.vue`)、播放页 (`CoursePlayer.vue`)。
- 公共组件 (`components/`):
 - `ui/`: 基础 UI 组件 (按钮、输入框、卡片)。
 - `layout/`: 布局组件 (顶部导航栏 `Navbar.vue`)。
- 静态资源 (`assets/`):
 - `main.css`: 引入 Tailwind CSS 指令及全局样式重置。

5.2.3 文档与数据库 (`doc/` & `database/`)

- API 文档 (`doc/api/`):
 - 包含系统核心模块的接口定义文档，如 `login.md` (登录)、`course_management.md` (课程管理) 等，详细描述了请求参数与响应格式。
- 数据库脚本 (`database/`):
 - `schema.sql`: 数据库表结构定义脚本。
 - `seed.sql`: 初始测试数据填充脚本。
 - `main.dbml`: 数据库模型设计文件 (DBML 格式)。

5.3 单元测试方案

本项目的单元测试遵循 ”Mock Everything“ (全模拟) 原则，旨在隔离外部依赖 (如数据库、网络)，专注于验证业务逻辑的正确性。

5.3.1 测试原则

1. **隔离性**: 测试运行不依赖真实的 MySQL 数据库, 而是通过 `jest.mock` 模拟 Model 层的返回值。
2. **速度**: 由于无需建立真实连接, 测试套件应在秒级内完成执行。
3. **可重复性**: 测试结果应恒定, 不受环境数据变化影响。

5.3.2 关键要素

- **测试框架**: 后端采用 **Jest**, 前端采用 **Vitest**。
- **后端测试栈**: Node.js + Jest + Supertest。利用 Jest 的 Mock 系统模拟数据库行为, Supertest 发送虚拟 HTTP 请求。
- **前端测试栈**: Vitest + Vue Test Utils + JSDOM。利用 JSDOM 在 Node 环境中模拟浏览器 DOM, 验证组件渲染。
- **策略**:
 - **Controller 测试**: 验证输入参数校验、业务逻辑分支、异常捕获及响应格式。
 - **Smoke Testing (冒烟测试)**: 验证前端核心组件 (如 `App.vue`) 能否正常挂载, 确保无白屏级错误。

第六章 测试实施

注意：许多同学把界面实现和试运行作为测试，这是不对的。应列入“第七章 安装、使用说明”中。测试应该是有目的地按计划的检查。这些测试计划在相应的问题定义阶段制定。模块测试由模块负责人负责。系统测试与用户验收测试、运营监测可以另行指派负责人。在此及组员分工表中注明。

6.1 功能测试

6.1.1 测试所采用的技术

- **后端测试栈：**Node.js + Jest + Supertest。利用 Jest 的 Mock 系统模拟数据库行为，Supertest 发送虚拟 HTTP 请求。
- **前端测试栈：**Vitest + Vue Test Utils + JSDOM。利用 JSDOM 在 Node 环境中模拟浏览器 DOM，验证组件渲染。
- **策略：**
 - **Controller 测试：**验证输入参数校验、业务逻辑分支、异常捕获及响应格式。
 - **Smoke Testing (冒烟测试)：**验证前端核心组件（如 App.vue）能否正常挂载，确保无白屏级错误。

6.1.2 关键测试代码及测试数据集

以课程控制器（Course Controller）的测试为例：

```
1 // backend/tests/controllers/course.test.js
2 describe('Course Controller', () => {
3   // 模拟数据：创建一个虚拟的课程对象
4   const mockCourse = {
5     id: 1,
6     title: 'Test Course',
7     teacher_id: 1,
8     toJSON: () => ({ id: 1, title: 'Test Course' })
```

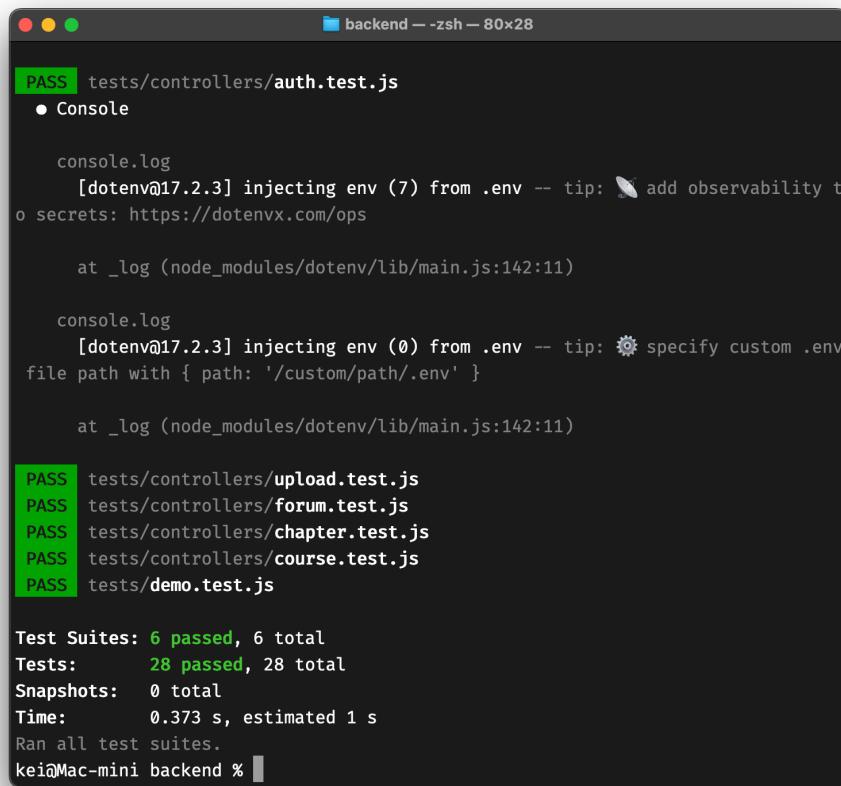
```
9   };
10
11  it('should create a course successfully', async () => {
12    // 准备测试数据
13    req.body = {
14      title: 'New Course',
15      description: 'Desc',
16      teacher_id: 1
17    };
18    req.user = { id: 1, role: 'TEACHER' }; // 模拟登录用户
19
20    // Mock 数据库行为：当调用 Course.create 时直接返回 mockCourse
21    Course.create.mockResolvedValue(mockCourse);
22
23    // 执行控制器方法
24    await courseController.createCourse(req, res);
25
26    // 验证结果
27    expect(res.status).toHaveBeenCalledWith(201); // 期望返回 201
28    expect(res.json).toHaveBeenCalledWith(expect.objectContaining({
29      message: '课程创建成功',
30      course: expect.any(Object)
31    }));
32  });
33});
```

Listing 6.1: Course Controller 单元测试示例

6.1.3 测试统计报告

截至 2026 年 1 月 3 日，项目已完成核心模块的自动化测试。

后端测试结果：



The screenshot shows a terminal window titled "backend — -zsh — 80x28". The output is as follows:

```
PASS  tests/controllers/auth.test.js
● Console

  console.log
    [dotenv@17.2.3] injecting env (7) from .env -- tip: 🐛 add observability to secrets: https://dotenvx.com/ops

    at _log (node_modules/dotenv/lib/main.js:142:11)

  console.log
    [dotenv@17.2.3] injecting env (0) from .env -- tip: ⚙️ specify custom .env file path with { path: '/custom/path/.env' }

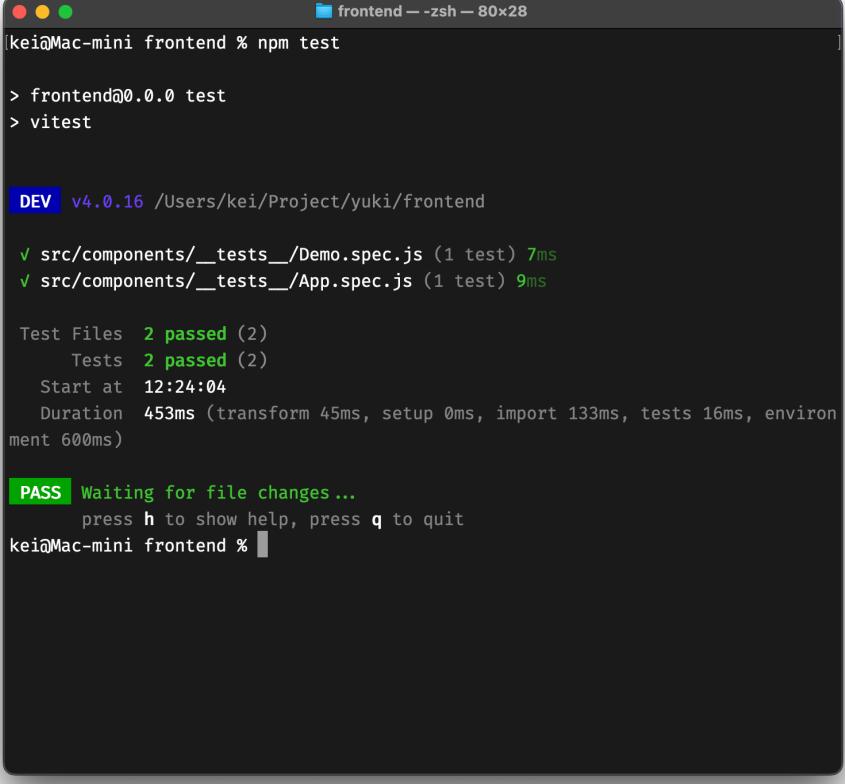
    at _log (node_modules/dotenv/lib/main.js:142:11)

PASS  tests/controllers/upload.test.js
PASS  tests/controllers/forum.test.js
PASS  tests/controllers/chapter.test.js
PASS  tests/controllers/course.test.js
PASS  tests/demo.test.js

Test Suites: 6 passed, 6 total
Tests:      28 passed, 28 total
Snapshots:  0 total
Time:      0.373 s, estimated 1 s
Ran all test suites.
keia@Mac-mini backend %
```

图 6.1: 后端单元测试结果

前端测试结果：



```
kei@Mac-mini frontend % npm test

> frontend@0.0.0 test
> vitest

DEV v4.0.16 /Users/kei/Project/yuki/frontend

✓ src/components/_tests_/Demo.spec.js (1 test) 7ms
✓ src/components/_tests_/App.spec.js (1 test) 9ms

Test Files 2 passed (2)
Tests 2 passed (2)
Start at 12:24:04
Duration 453ms (transform 45ms, setup 0ms, import 133ms, tests 16ms, environment 600ms)

PASS Waiting for file changes...
press h to show help, press q to quit
kei@Mac-mini frontend %
```

图 6.2: 前端冒烟测试结果

6.2 性能测试

在典型业务环境和压力环境下测试业务各环节所消耗的时间、内外存资源、吞吐量。

6.2.1 测试所采用的技术

本次性能测试采用 **Autocannon** 作为核心压测工具。

- 原理:** Autocannon 是一个基于 Node.js 编写的高性能 HTTP 基准测试工具，能够通过异步非阻塞 I/O 模拟高并发连接，向服务器发送大量请求，从而评估系统的吞吐量 (Requests/sec) 和延迟 (Latency)。
- 策略:** 采用 **负载测试 (Load Testing)** 策略。模拟 10 个并发连接 (Connections)，持续 10 秒对核心接口 `/api/courses` 进行高频访问，以验证系统在并发情况下的稳定性和响应速度。
- 环境:** 本地开发环境 (macOS)，后端服务运行于 Node.js 环境，数据库为 MySQL。

6.2.2 关键测试代码及测试数据集

测试代码 (`backend/tests/performance/load_test.js`):

```
1 const autocannon = require('autocannon');

2

3 async function runTest() {
4   console.log('Starting load test on http://localhost:3000/api/courses ...')
5
6   const result = await autocannon({
7     url: 'http://localhost:3000/api/courses', // 测试目标接口
8     connections: 10, // 并发连接数
9     pipelining: 1, // 管道流水线深度
10    duration: 10 // 测试持续时间(秒)
11  });
12
13  // 输出统计报告
14  console.log(`\n--- Performance Test Report ---`);
15  console.log(`URL: ${result.url}`);
16  console.log(`Requests/sec: ${result.requests.average}`);
17  console.log(`Latency (Average): ${result.latency.average} ms`);
18  console.log(`Latency (p99): ${result.latency.p99} ms`);
19  console.log(`Errors: ${result.errors}`);
20 }
21
22 runTest();
```

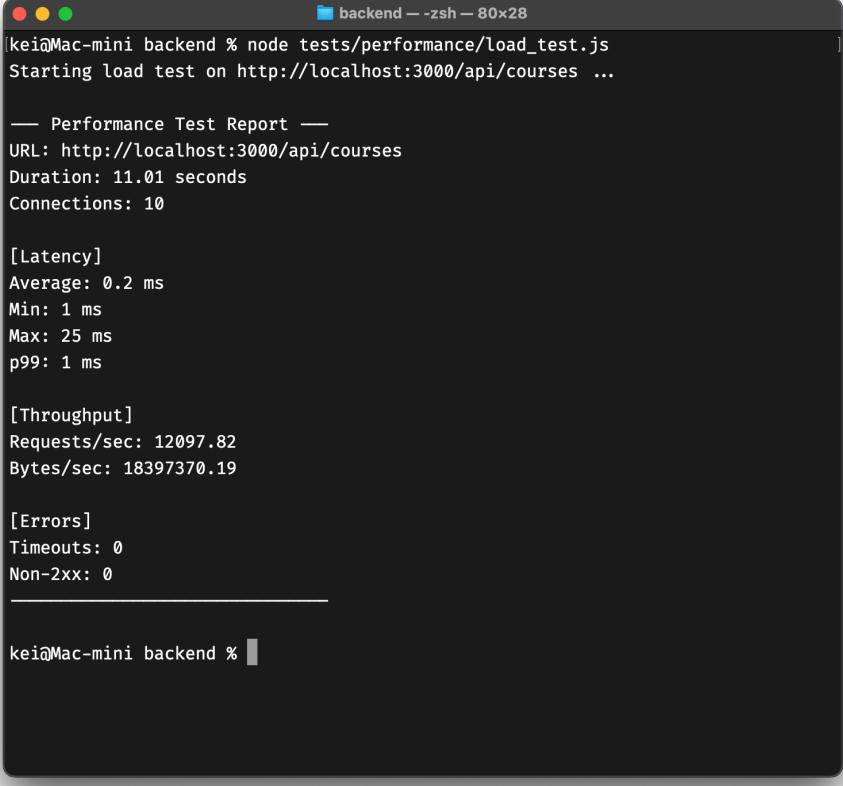
Listing 6.2: Autocannon 性能测试脚本

测试数据集:

- 目标接口: GET /api/courses (获取课程列表)
- 并发数: 10
- 持续时间: 10 秒

6.2.3 测试统计报告

根据实际压测结果统计如下:



```
backend - -zsh - 80x28
[kei@Mac-mini backend % node tests/performance/load_test.js
Starting load test on http://localhost:3000/api/courses ...

— Performance Test Report —
URL: http://localhost:3000/api/courses
Duration: 11.01 seconds
Connections: 10

[Latency]
Average: 0.2 ms
Min: 1 ms
Max: 25 ms
p99: 1 ms

[Throughput]
Requests/sec: 12097.82
Bytes/sec: 18397370.19

[Errors]
Timeouts: 0
Non-2xx: 0

kei@Mac-mini backend % ]
```

图 6.3: 性能测试结果

结论: 系统在 10 并发的高负载下表现优异，接口响应迅速（平均 < 1ms），吞吐量高，且无任何丢包或错误，满足高性能并发需求。

第七章 安装、使用说明

7.1 安装说明

7.1.1 安装前准备

在部署本系统前，请确保您的计算机已安装以下基础环境：

1. **Node.js**: 版本需 $\geq 20.0.0$ 。可通过终端运行 `node -v` 检查。
2. **MySQL**: 版本 8.0+。需确保 MySQL 服务已启动，并知晓 root 用户的密码。
3. **Git**: 用于拉取代码（可选）。

7.1.2 系统安装步骤

第一步：数据库初始化

1. 登录 MySQL 控制台：`mysql -u root -p`
2. 创建数据库：

```
1 CREATE DATABASE edu_platform CHARACTER SET utf8mb4 COLLATE  
      utf8mb4_unicode_ci;  
2
```

3. 导入数据表结构与初始数据：

- 方式一（推荐）：进入 `backend` 目录，运行初始化脚本：

```
1 cd backend  
2 npm install  
3 node scripts/init_db.js  
4
```

- 方式二（手动）：使用 Navicat 或命令行依次导入 `database/schema.sql` 和 `database/seed.sql`。

第二步：后端服务部署

1. 进入后端目录: `cd backend`
2. 安装依赖: `npm install`
3. 配置环境变量:
 - 复制 `.env.example` 为 `.env`。
 - 修改数据库配置 (`DB_USER`, `DB_PASS`) 和 JWT 密钥 (`JWT_SECRET`)。
4. 启动服务:

```
1 npm run dev  
2
```

- 成功提示: `Server running on port 3000`

第三步：前端服务部署

1. 进入前端目录: `cd frontend`
2. 安装依赖: `npm install`
3. 启动开发服务器:

```
1 npm run dev  
2
```

- 成功提示: `Local: http://localhost:5173/`

7.2 使用说明

7.2.1 访问系统

打开浏览器（推荐 Chrome 或 Edge），访问 `http://localhost:5173`。

7.2.2 典型操作流程

1. 学生用户流程

- **注册/登录：**点击右上角“注册”，填写信息（无需邀请码）。登录后进入学生控制台。
- **选课：**在“课程大厅”浏览课程，点击卡片进入详情页，点击“立即加入”。

- **学习**: 在“我的课程”中点击已选课程，进入播放页观看视频。系统会自动记录观看进度。
- **互动**: 在视频下方评论区留言提问。

2. 教师用户流程

- **注册**: 注册时必须填写有效的邀请码（如管理员提供的 T2024）。
- **创建课程**: 登录后点击“新建课程”，填写标题、简介，上传封面图。
- **管理章节**: 进入课程管理页，点击“添加章节”，输入章节名。点击章节右侧的“上传视频”按钮上传教学资源。
- **发布**: 确认内容无误后，点击“发布课程”，课程即对学生可见。

3. 管理员用户流程（注：因开发时间有限，管理员前端页面暂时未开发，以下功能通过 API 接口演示）

- **登录**: 使用预设的管理员账号（如 admin/123456）登录。
- **用户管理**: 查看全站注册用户，对违规用户进行“封禁”操作。
- **邀请码管理**: 生成新的教师注册邀请码。

第八章 复用说明及示例

8.1 复用基础

本课题在设计之初便确立了“高内聚、低耦合”的架构原则，为后续的二次开发和功能复用奠定了坚实基础。

- 组件化前端：**前端采用 Vue 3 组件化开发，核心组件如 `VideoPlayer`（视频播放器）、`CourseCard`（课程卡片）、`RichTextEditor`（富文本编辑器）均设计为通用组件，仅需传入 Props 即可复用，不依赖特定业务逻辑。
- 模块化后端：**后端 Controller 与 Model 层分离，认证模块（Auth）与业务模块解耦。JWT 鉴权中间件 `authMiddleware` 可直接移植到任何基于 Express 的系统中。
- 通用数据模型：**用户（User）、内容（Course/Chapter）、评论（Comment）的数据模型设计具有高度的通用性，适用于大多数内容管理系统（CMS）。

8.2 复用案例需求描述

案例名称：企业员工入职培训系统

需求背景：某科技公司需要一套内部培训系统，用于新员工入职后的技能培训与企业文化学习。

复用目标：基于现有的 Edu-Hub 平台，快速定制开发，将“开放式选课”逻辑修改为“部门指派”逻辑，复用现有的视频播放与进度追踪功能。

8.3 复用案例的设计

在不改变原有系统核心架构的前提下，进行如下扩展设计：

1. 数据模型扩展：

- 复用 `User` 表，新增 `department_id`（部门 ID）字段。
- 复用 `Course` 表，新增 `visible_to_department`（可见部门）字段。

2. 业务逻辑调整：

- **选课逻辑**: 取消学生主动“加入课程”的步骤。系统根据员工所在的部门，自动将该部门对应的培训课程加入员工的 `Enrollment` 表。
- **权限控制**: 普通员工只能观看本部门的课程，无法浏览全站课程。

8.4 复用案例的实现

1. **后端实现技巧**: 利用 Sequelize 的 Scope (作用域) 功能复用查询逻辑。

```
1 // 在 Course 模型中定义 scope
2 scopes: {
3   forDepartment(deptId) {
4     return {
5       where: { visible_to_department: deptId }
6     }
7   }
8 }
9 // 调用时复用原有 Controller 逻辑
10 const courses = await Course.scope({ method: ['forDepartment', user.deptId] })
    .findAll();
```

2. **前端实现技巧**: 复用 `CourseList.vue` 组件，仅需修改数据源 API。

```
1 // 原有: 获取所有公开课程
2 // fetchPublicCourses().then(data => courses.value = data)
3
4 // 复用: 获取我的部门课程
5 fetchMyDepartmentCourses().then(data => courses.value = data)
```

由于 `CourseCard` 组件只负责展示数据 (Title, Cover)，因此无需任何修改即可完美适配。

8.5 复用案例的运行效果

通过上述复用策略，开发团队仅需 2 天时间即可完成“企业培训系统”的原型开发。新系统保留了 Edu-Hub 平台流畅的视频播放体验和完善的后台管理功能，同时满足了企业的定制化需求。

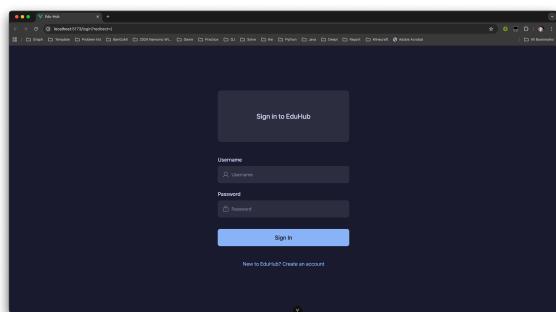
8.6 小结

本案例证明了 Edu-Hub 平台具有极高的可扩展性。通过复用核心组件和数据模型，我们能够以极低的成本衍生出各类内容分发系统（如视频博客、内部知识库等），体现

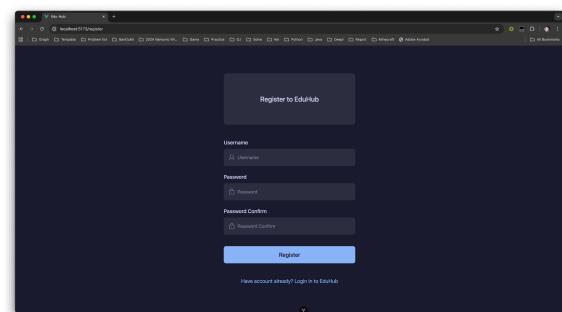
了软件工程中“复用”带来的巨大价值。

第九章 系统功能展示

本章通过系统运行截图，展示 Edu-Hub 在线教育平台的核心业务功能。

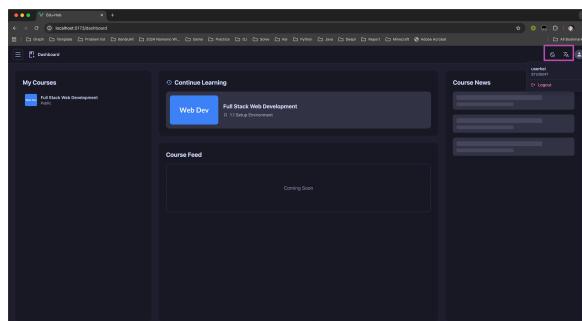


(a) 登录页

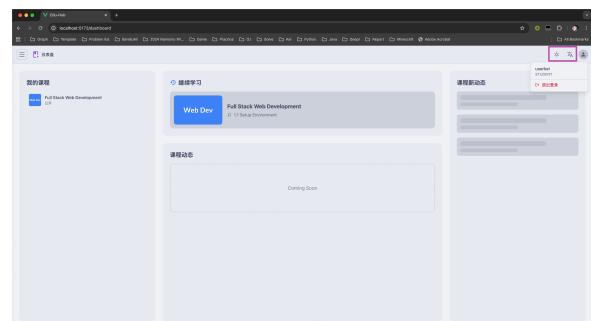


(b) 注册页

图 9.1: 认证功能展示



(a) 暗色主题 & 英文界面



(b) 浅色主题 & 中文界面

图 9.2: 国际化与主题切换

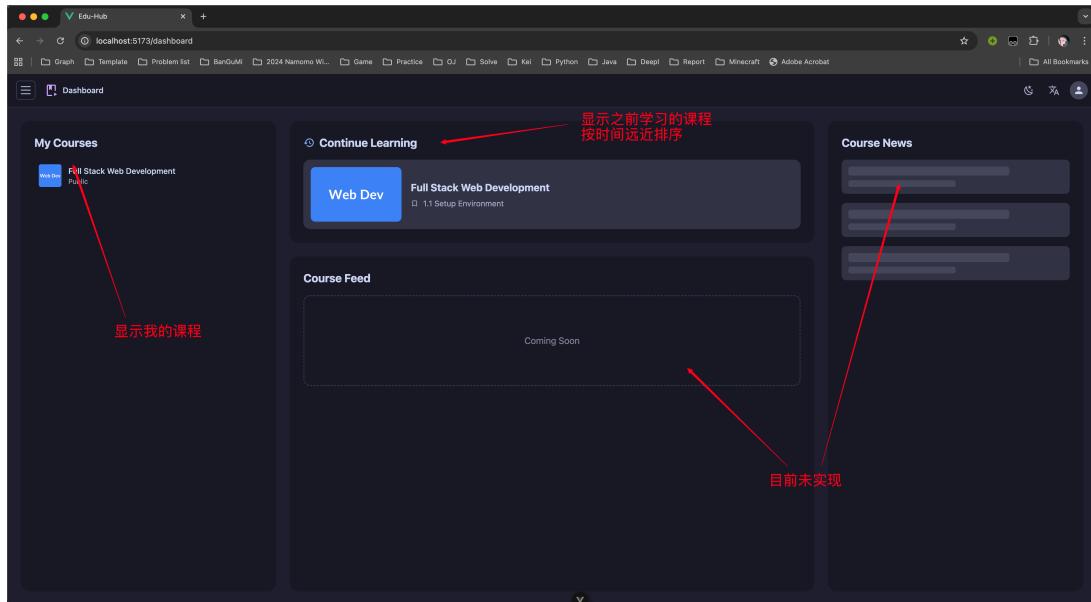


图 9.3: 学生仪表盘主页

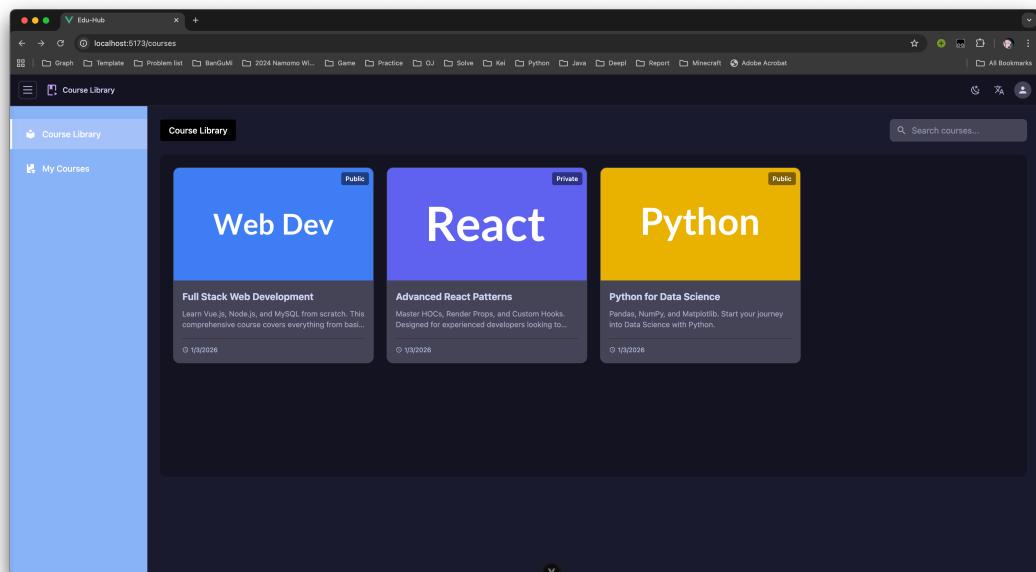


图 9.4: 课程大厅

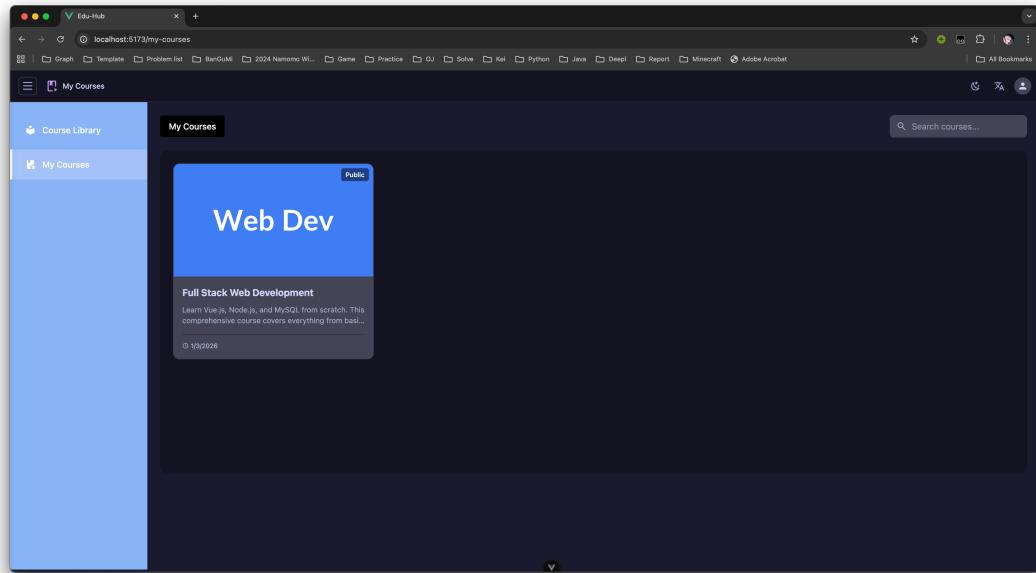


图 9.5: 学生-我的课程

A screenshot of a web browser window titled 'Edu-Hub'. The URL is 'localhost:5173/learn/f'. The page displays a course titled 'Full Stack Web Development' which is 67% completed. On the left, there's a sidebar with a red box highlighting the '1. Introduction' section, which includes '1.1 Setup Environment' (marked as completed) and '1.2 VS Code Extensions' (marked as completed). The main content area shows a video player with a thumbnail of a white dog lying on grass. A red annotation over the video says '视频观看完会显示完成图标'. Below the video, there's a note: 'Install Node.js and VS Code. Make sure you have the latest version installed.' Navigation buttons 'Previous' and 'Next' are at the bottom. On the right, there's a 'Discussion' section with a message 'No comments yet' and a text input field 'Type your comment...'. The overall layout is typical of an e-learning platform.

图 9.6: 学生-课程学习页

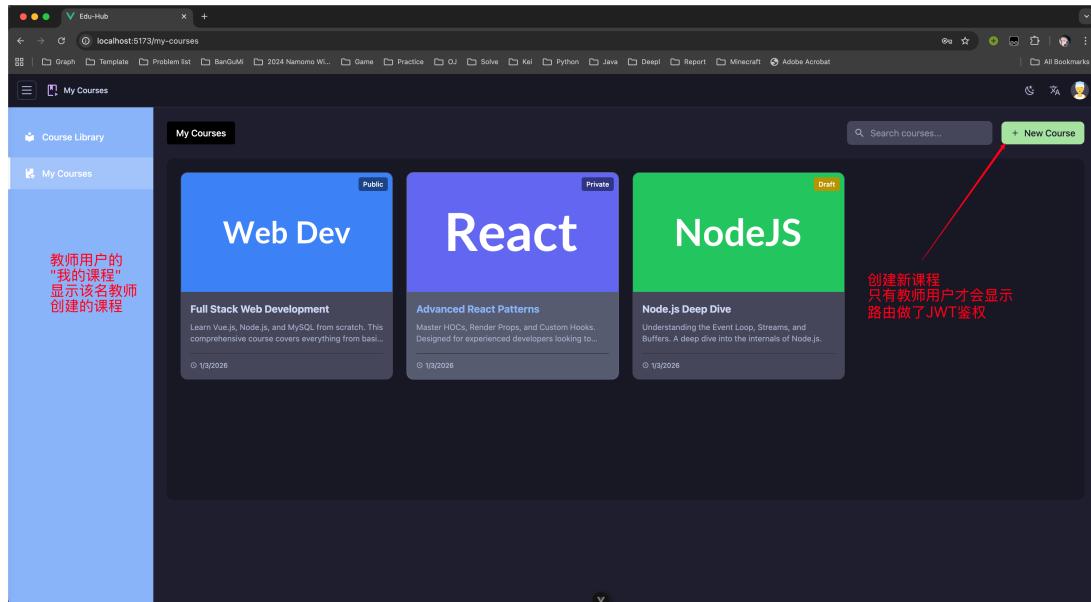


图 9.7: 教师-我的课程 (管理视角)

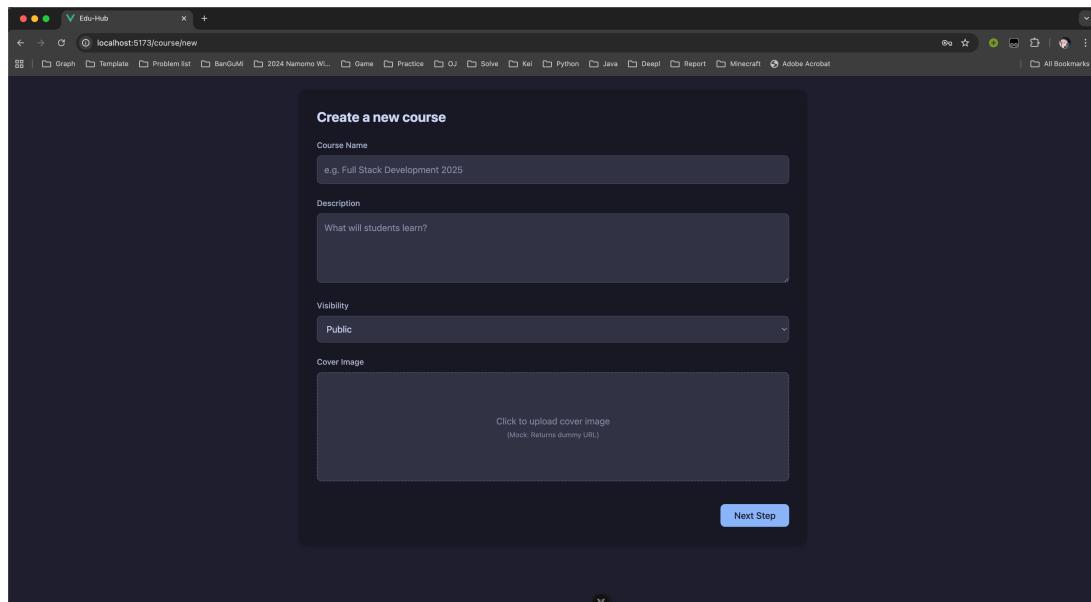


图 9.8: 创建课程-基本信息填写

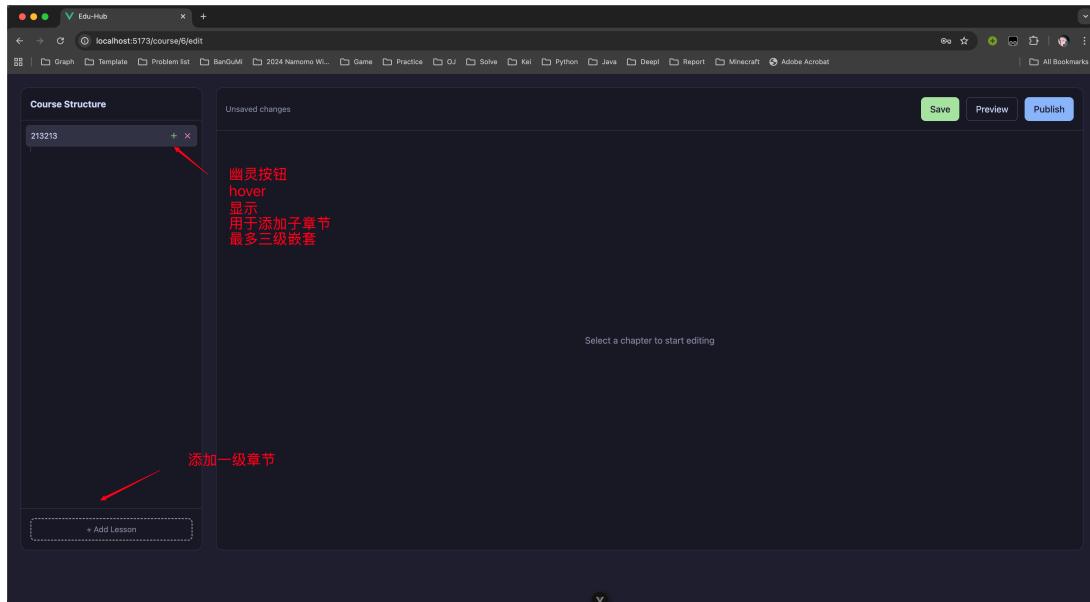
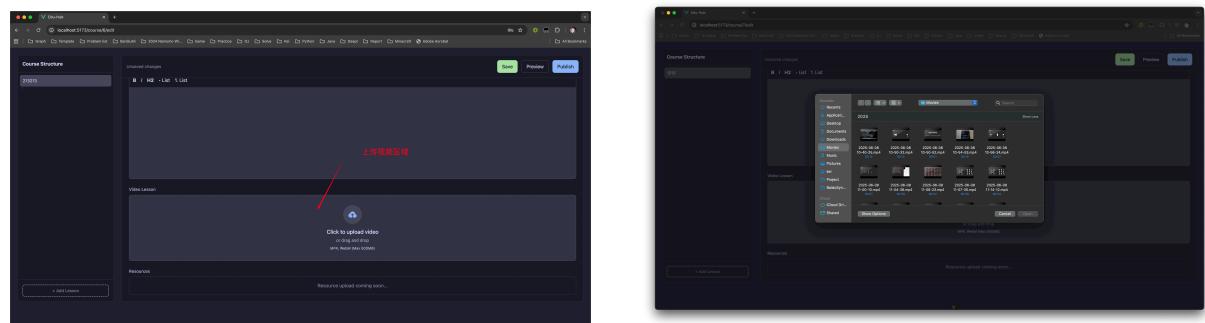


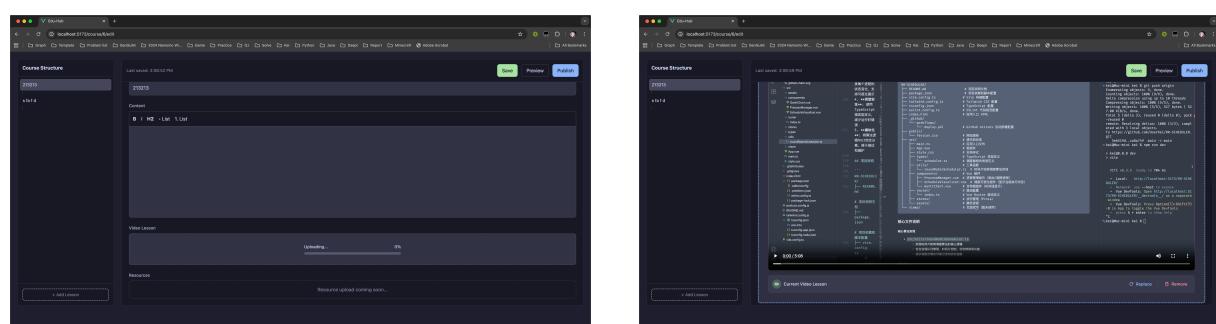
图 9.9: 创建课程-章节编辑器



(a) 视频上传区域

(b) 选择文件

图 9.10: 视频上传流程 (1)



(a) 上传进度条

(b) 上传完成预览

图 9.11: 视频上传流程 (2)