

《软件工程》项目总结

第七组 吴立文 沈冯毅 陈徐豪 叶一凡 王安康

在本学期的《软件工程》，我们小组完成了一个高校毕业设计管理系统。从第六周到第十五周，经过需求分析、系统设计、详细设计、编码、测试等流程，我们成功完成了此次项目。以下是对项目总体执行过程的分析总结。

一、项目启动阶段

首先我们进行了项目分组，对项目负责人、系统分析与架构设计、前端开发、后端开发、测试与文档编写的事项初步分工，明确了各自的职责。我们还查询了往年官网发布的毕设通知，了解了这个系统所必须包含的内容和功能有哪些，并在此基础上编写了项目计划书。这一切使我们快速理清了设计的大致方向，为管理系统定了一个整体框架，有助于后续进行有条不紊的设计。

二、需求分析阶段

在需求分析阶段，我们首先通过查询往年官网发布的毕设通知，明确了系统必须包含的核心内容和功能，即支持学生、教师和管理员三类角色，并围绕论文的全生命周期管理（提交、评审、查询、进度跟踪）展开。在此基础上，我们绘制了业务流程图，详细描述了从学生登录、选择提交版本、上传论文文件，到教师登录、搜索论文、提交评审结果（含评分、审阅意见和结果），再到学生查看评审结果与进度的完整流程，并补充了如修改课题信息等功能。同时，我们完成了数据分析表，明确了用户（Users）、论文（Theses）、评审（Reviews）等核心数据类名称、描述、来源等信息。通过小组集中讨论，我们确定了功能分析表的具体内容，分为基本功能（如登录认证、论文上传与评审）和辅助功能（如进度查看、搜索筛选）两大模块，并设计了统一的提示信息索引（如“无效凭证”、“身份不匹配”、“论文提交成功”等），确保了工程信息的规范性和可维护性。这一阶段的工作为后续系统采用 Django 后端和 React 前端的分离架构奠定了坚实的需求基础。

三、系统设计阶段

系统设计阶段，我们依据需求分析成果，首先完成了模块结构图的绘制，对高校毕业设计管理系统的子系统进行了划分，构建了以角色为核心的自上而下树状模型，明确了学生仪表盘（包含论文提交、我的进度子模块）、教师仪表盘（包含论文评审、审核评阅子模块）和管理员仪表盘。随后，基于界面原型设计（如

登录界面、论文提交弹窗、评审结果页)和模块结构图,我们设计了系统框架草图,确立了前后端分离的技术架构:前端使用 Vite+React+TypeScript,后端使用 Django REST Framework,数据库选用 SQLite,确保了系统结构的直观与便捷。同时,我们绘制了 E-R 图,清晰展示了用户(继承自 Django AbstractUser,含 role 字段区分身份)、论文(与学生用户外键关联)、评审(与论文和教师用户外键关联)等实体的属性及其相互关系,并在此基础上完成了数据库表结构的设计,将实体属性转化为具体的字段定义(如 Theses 表包含 title, file, version, student_id 等字段)。这一阶段的成果为后续的详细设计与实现提供了清晰的架构指导。

四、详细设计阶段

在详细设计阶段,我们进一步细化了系统设计。我们完成了模块程序逻辑设计,绘制了用户登录、论文上传、论文评审三个关键模块的程序流程图,详细描述了从前端界面操作(如输入学工号、密码、选择身份)到后端 API 验证(如 LoginAPIView 中的身份查找、密码校验、Token 生成)、业务逻辑判断(如 ThesisSubmitAPIView 中的角色权限检查)的具体步骤与判断逻辑。根据程序流程图,我们按照规范绘制了 N-S 图(盒图),确保了逻辑的严谨性,例如清晰定义了登录验证失败或论文提交权限不符时的异常处理路径。同时,我们编写了数据库结构文件(db.sqlite3),并通过 Django 迁移文件管理了包含注释头、环境设置、建库、建表、插入初始数据等完整结构,并撰写了接口说明文档,明确了 API 概览(如 POST /api/login/, POST /api/theses/, POST /api/reviews/)、认证方式(TokenAuthentication)、全局响应格式(JSON)及错误码(如 400、403、404)等关键信息。这一阶段的细致设计为编码实现提供了精确的蓝图。

五、系统实现阶段

系统实现阶段,我们依据详细设计文档,开始了具体的编码工作。我们完成了源代码文件的设计与编写,其中 backend 部分基于 Django 框架,在 users 应用中实现了 models.py(定义数据模型)、serializers.py(定义序列化器)、views.py(实现如 LoginAPIView, ThesisSubmitAPIView 等 API 视图)和 urls.py(配置路由),并配置了 settings.py 中的数据库、认证等设置;frontend 部分使用 React 和 TypeScript,开发了 LoginPage.tsx、StudentDashboard.tsx、TeacherDashboard.tsx 等角色主页面,以及 ui 目录下的可复用组件(如 button.tsx, card.tsx),并通过 api.ts 文件集中管理所有 API 调用。项目根目录配置了.gitignore 文件以排除不必要的文件。通过小组集中讨论,我们解决了代

码实现过程中的具体技术问题，如前后端 **Token** 认证的集成、文件上传（**MultiPartParser**）的处理等，确保了代码质量与设计要求的符合。随后，我们整合了完整的系统源代码，并完成了接口测试报告的编写，通过等价类划分法设计了详尽的测试用例（如 **TC1-TC8** 测试登录模块），详细描述了测试概述、测试环境（本地开发服务器）、测试执行过程及结果分析（如发现并记录了“超长字符串导致系统崩溃”等 **BUG**）等内容。这一阶段成功将设计转化为可运行的系统。

六、系统测试阶段

在系统测试阶段，我们重点对系统进行了全面的测试。我们依据接口测试报告，对系统接口进行了详细测试，验证了登录、论文上传、论文评审等接口的协议类型（**HTTP**）、请求方法（**POST**）、请求格式（**JSON/FormData**）及返回标准（状态码、数据格式）的正确性，并确认了如“仅学生可提交论文”、“仅教师/管理员可评审”等权限控制的准确性。同时，我们进行了系统整合与前后端交互联调，修复了测试过程中发现的 **Bug**，例如前端增加了表单验证（如论文题目非空检查、文件类型校验）以弥补后端检查的不足，优化了错误提示的显示机制，确保了系统的稳定性和功能完整性。通过严格的测试流程，我们验证了系统是否满足需求分析阶段所设定的各项功能与性能指标，虽然识别出在用户体验和鲁棒性方面仍有提升空间（如对超长输入的处理），但核心业务流程已全部贯通，为项目的最终交付提供了质量保障。

七、遇到的问题及解决方法

1、需求与设计阶段的逻辑一致性问题

功能逻辑整合困难：团队成员分工完成功能分析表的不同部分后，各部分之间的逻辑衔接可能存在冲突或不一致。数据与接口设计分歧：在定义 **E-R** 图实体关系、数据库表结构以及接口规范时，容易产生设计思路或细节上的分歧。模块流程跳转复杂：系统包含多个模块（如课题申报、审核、答辩），如何清晰定义模块间的联系和操作后的流程跳转逻辑是一大挑战。

解决方法：采用严格的自上而下设计流程，并使用标准化文档约束设计细节。先通过 **E-R** 图（第 10 周）明确核心实体及其关系，奠定数据基础。再根据 **E-R** 图设计详细的数据库表结构。最后，基于数据模型定义 **API** 接口规范，并在接口说明文中事先约定“全局响应格式”等标准。这样做确保了从数据模型到接口

设计的连贯性和一致性，避免了因设计顺序混乱导致的返工。

利用程序流程图和 N-S 图等可视化工具，对模块内部逻辑和模块间交互进行建模。我们团队通过绘制详细的程序流程图和 N-S 图，共同讨论“判断语句后该如何跳转”以及“如何建立模块联系”，将复杂的业务逻辑可视化。图表使复杂的流程和状态跳转变得直观，帮助团队在编码前就明确了所有操作路径，减少了后续开发中的逻辑歧义。

2、用户体验与系统架构的平衡问题

系统架构需要实现复杂的业务功能，而界面设计必须将这些功能以直观、便捷的方式呈现给用户，避免用户因界面复杂而产生困惑。

解决方法：功能驱动与用户体验并重的设计。界面原型设计严格遵循功能分析表，同时将“用户友好性”作为核心设计目标。在第 8 周和第 9 周，团队在讨论界面原型和系统框架草图时，核心议题是如何将功能模块（如课题增删改查）以“直观便捷”的布局和跳转逻辑呈现，确保“方便用户使用”。最终的系统框架草图不仅完整承载了所有业务功能，而且具备了清晰的信息架构和用户操作流程，在架构复杂性和使用简便性之间取得了良好平衡。

3、编码实现与设计方案的契合度问题

在从详细设计文档（如流程图、N-S 图）转向实际编码时，如何确保所有开发人员对实现方式的理解一致，并能精准落地设计细节。

解决方法：详细设计先行与集体编码决策。在编写代码前完成极其详细的程序逻辑设计，并在编码初期进行集体决策以统一技术实现思路。在第 13 周编码开始前，团队已拥有成熟的程序流程图和 N-S 图。团队并未立即分工编码，而是先集中讨论“代码该如何编写才能实现前面详细设计的各项要求”，共同确定技术选型（如采用 Django 框架）和核心代码结构。这种“先讨论，后动手”的方式，确保了所有成员对设计稿的理解和技术实现路径高度一致，从起点上保证了代码与设计方案的契合度，有效避免了后期集成时出现重大偏差。

4、技术实现与架构问题：前后端数据交互与认证机制的实现

在系统实现阶段，需要确保基于 Token 的身份认证机制能在 React 前端和 Django 后端之间稳定工作。同时，如何处理 API 请求（尤其是包含文件上传的请求）并统一管理错误响应是一个关键点。

解决方法：统一 API 管理：在前端创建专用的 api.ts 文件，集中封装所

有 API 请求。在此处统一设置请求基础 URL、请求头，并处理认证 Token 的自动携带和响应拦截，减少了重复代码。规范认证方式：后端 Django REST Framework 使用 TokenAuthentication，并在需要权限的视图（如 ThesisSubmitAPIView）中设置 permission_classes = (IsAuthenticated,)。文件上传处理：后端为处理文件上传的 API 视图（如论文提交）明确配置 parser_classes = (MultiPartParser, FormParser)，以正确解析表单数据。

5、技术实现与架构问题：数据库选择与未来扩展性的矛盾

在开发阶段为了简洁和调试方便，使用了 SQLite 数据库。但项目后期意识到，SQLite 在生产环境中可能存在性能瓶颈和并发能力弱的问题。

解决方法：阶段性策略：在系统设计阶段就明确 SQLite 仅用于开发。在《系统说明文档》的“小结”中已规划好解决方案：在生产环境中，将迁移至更强大、可扩展的数据库系统，如 PostgreSQL 或 MySQL。这确保了架构的可演进性。

6、功能逻辑与业务规则问题：用户身份校验与权限控制

登录时，用户选择的“身份”（角色）需要与系统中该账号实际存储的角色一致。在业务操作中，需严格区分不同角色的权限，例如，禁止教师提交论文或学生评审论文。

解决方法：登录时双重验证：在 LoginAPIView 中，不仅验证用户名/密码，还检查用户选择的身份是否与系统中该用户的 profile.role 字段匹配。如果不匹配，返回“身份不匹配”错误。API 视图级权限控制：在关键 API 中编写明确的业务逻辑代码。例如，在 ThesisSubmitAPIView 的 create 方法中，首先检查 request.user.profile.role != 'student'，如果是则返回 403 错误。同样，在论文评审 API 中检查角色是否为教师或管理员。

7、项目协作与流程管理问题：确保代码与文档的同步和一致性

在多人并行开发（前端、后端、测试）过程中，需要确保所有成员对系统设计、API 接口的理解是一致的。

解决方法：详尽的文档驱动开发：在每个阶段（需求、设计、详细设计）都产出标准化文档，如 E-R 图、接口说明文档、模块流程图。这些文档成为所有开发者的共同蓝图。集中讨论与定期的周会：通过“小组集中讨论”来解决开发中遇到的具体技术分歧和接口联调问题，确保前后端开发进度匹配。

八、总结

在为期十周的软工课设里，我们小组顺利完成了高校毕业设计管理系统的开发。这次项目让我们完整走了一遍软件工程的生命周期，从最初的需求分析、系统设计，再到编码实现和系统测试，每一步都严格按照计划推进。我们重点梳理了学生、教师和管理员这三类用户的核心需求，最终决定采用 **Django REST Framework** 作为后端、**React+TypeScript** 作为前端的分离架构方案，并用 **E-R图**、**程序流程图**等工具完成详细设计。

开发过程中也遇到不少实际问题，比如需求逻辑不一致、技术实现上的难点、权限控制怎么合理设计等等。不过通过组内反复讨论和调试，我们最终把这些难点都一一解决了，也保证了系统功能的完整和稳定。到项目结题时，所有核心业务流程都已顺利跑通，基本达到了我们最初设定的目标。

总的来说，这次课设不仅让我们对课堂上学到的理论有了更实际的理解，也锻炼了团队协作和解决实际问题的能力，为后续的系统优化和扩展打下了不错的基础。

九、个人总结

1、吴立文

本学期的《软件工程》课程，我以组长身份完成了“高校毕业设计管理系统”的开发项目。通过这个项目的开发，我不仅加深了对课程内容的理解，还在实践中提升了自己的技术能力和团队合作能力，获得了宝贵的经验与成长。

《软件工程》课程通过对软件开发全生命周期的介绍，使我对需求分析、系统设计、编码实现、测试以及部署维护等各个环节有了更加清晰的认识。在项目中，我作为组长负责了需求分析和系统设计阶段。我们根据文档中的要求，进行需求调研、功能模块划分以及系统框架搭建。我通过参与 **ER图**的绘制和数据库设计，深刻理解了数据库设计的重要性，特别是在数据表关系和数据存取效率方面的考虑。这些实践与理论的结合，使我能够更好地将课程中学到的知识应用于实际开发中。

在项目的开发过程中，我负责了部分后端开发，尤其是与用户管理和论文模块相关的功能。通过使用 **Django** 框架，我熟悉了 **RESTful API** 的设计和开发流程，学习了如何通过序列化器将数据库模型与前端传输的数据进行转换。在论文上传模块的实现中，我通过设计 **API** 接口和控制权限，确保了系统的功能逻辑严谨且符合先前设计。这一过程中，我深刻理解了权限管理、用户认证及文件上

传等功能的实现原理，提升了我的后端开发能力。

此外，项目中使用 **React** 和 **TypeScript** 进行前端开发也是我首次接触这类技术栈。在前端部分，我参与了系统界面的设计与开发，通过搭建前后端分离的架构，使我对前端的开发流程和调试过程有了深入的了解。通过与前端开发人员的密切合作，我学会了如何与后端 **API** 进行高效对接，如何处理前端交互中的状态管理和界面渲染等问题。

在团队合作方面，我学到了如何进行有效的团队管理与沟通。作为组长，我需要统筹全局，协调各个成员的工作，确保项目按时推进。通过定期的团队会议和任务分配，我逐渐学会了如何在团队中担任领导角色，处理团队成员之间的分歧，确保每个成员都能够高效地完成自己的任务。

回顾整个项目开发过程，我也发现了许多不足之处，尤其是在系统的性能和安全性方面。尽管项目基本功能已经实现，但未来可以考虑引入更强的数据库支持、更高效的代码优化以及更加严格的安全措施。通过对这些问题的思考，我更加明白了软件开发中如何平衡功能实现与性能优化之间的关系。

总的来说，《软件工程》课程与项目开发的结合，为我提供了理论与实践的双重提升。我不仅学会了如何使用技术工具实现一个完整的系统，还体会到团队合作与项目管理的重要性。未来，我将继续提升自己的技术水平，并在实际项目中不断总结经验，追求更高效的工作方法和更加完善的系统设计。

2、叶一凡

在本次“高校毕业设计管理系统”的开发实践中，我主要承担了需求分析、技术文档撰写以及系统功能测试工作。通过贯穿软件开发生命周期（**SDLC**）的深度参与，我不仅在业务逻辑层面协助团队完成了系统建模，更在质量保证环节验证了系统设计的严密性。

在项目初期的需求分析阶段，我针对学生、教师、管理员三类利益相关者的业务场景，进行了细致的功能解构。由于系统涉及论文提交、评审反馈及评分管理等多个交叉业务流，我重点通过对业务逻辑的梳理，确保了权限控制与数据流向的一致性。这些分析工作最终沉淀为系统的设计规范，不仅为后续前后端开发提供了清晰的逻辑基准，也最大程度地规避了开发过程中的功能偏离风险。

作为文档撰写的主要负责人之一，我力求将复杂的系统逻辑转化为清晰的技术语言。从项目架构的全局说明到各个 **API** 接口的功能定义，我通过规范化的文档记录，确保了团队内部的沟通效率与信息透明度。在实际协作中我深刻体会

到，高质量的文档不仅是项目交付的一部分，更是确保系统后期可维护性的关键资产。

在系统测试阶段，我主要针对登录鉴权、论文上传及评审模块进行了黑盒测试与逻辑回归。利用 **Django REST framework** 提供的接口特性，我重点验证了 **API** 视图层的安全性，特别是在 **ThesisSubmitAPIView** 等核心组件中，通过模拟跨权限请求，验证了 **Token** 校验机制和角色限制的有效性。此外，我也参与了评审数据与论文实体关联性的校验，确保了核心业务闭环的准确可靠。

通过对项目的整体复盘，我也发现了一些亟待优化的技术细节。目前系统采用的 **SQLite** 数据库虽然能够满足开发调试阶段的需求，但在应对高并发访问和数据一致性要求较高的场景时，仍存在扩展性瓶颈，未来应考虑向 **PostgreSQL** 等更成熟的关系型数据库进行架构迁移。同时，系统的自动化测试覆盖率仍有提升空间，这是提升系统长期稳定性的重要方向。

本次实践让我对软件工程的方法论有了更具象的认知：优秀的软件系统不仅取决于代码质量，更取决于前期的严谨规划与后期的严格验证。在未来的学习中，我将继续探索如何利用工程化手段，进一步提升软件开发的可靠性与交付质量。

3、沈冯毅

本报告旨在总结我在“毕业设计管理系统”项目中的前端开发与系统部署工作。作为项目小组的主要前端开发者与部署负责人，我参与了从技术选型、功能实现到最终上线的完整周期，在实战中深化了对现代 **Web** 开发与 **DevOps** 流程的理解。以下将围绕我的具体工作内容、技术实践、问题解决及个人感悟进行阐述。

一、 工作内容概述

本项目是一个多角色协作的 **B/S** 架构系统，后端采用 **Django** 框架提供 **RESTful API**，前端需构建一个交互友好、功能清晰的单页面应用（**SPA**）。我承担的核心职责包括：

1. 前端架构设计与开发：负责整个 **React** 前端的搭建、核心功能模块的实现，以及与后端 **API** 的联调。
2. 系统部署与运维：负责将前后端应用部署至生产环境，并配置相关的 **Web** 服务器、反向代理及基础运行环境。

二、 技术选型与前端开发实践

开发工具链：选择 **Vite** 作为构建工具。与传统的 **Webpack** 相比，**Vite** 基于原生 **ES** 模块，提供了极快的冷启动和热更新速度，显著提升了开发体验和效率，尤其适合 **React + TypeScript** 的项目。

核心框架与语言：采用 **React 18** 配合 **TypeScript**。**React** 组件化思想完美契合了系统内如用户仪表盘、论文列表、评审表单等功能模块的复用需求。**TypeScript** 的静态类型检查在开发大型应用时优势明显，它帮助我在编码阶段就捕获了许多潜在的类型错误，增强了代码的健壮性和可维护性，同时也使组件 **Props** 和 **API** 响应数据的结构清晰易懂。

状态管理与路由：使用 **Context API** 与 **Hooks**（如 `useState`、`useReducer`、`useContext`）管理应用级状态（如用户登录信息、全局通知）。对于复杂的表单状态（如多步骤的论文提交、评审打分），则采用了 `useReducer` 进行集中管理。路由库选用 `React-Router-DOM`，实现了基于角色（学生、教师、管理员）的动态路由和权限导航守卫，确保用户只能访问被授权的页面。

功能实现要点：

1. 用户认证与授权：实现了完整的 **JWT** 令牌认证流程。用户登录后，令牌被安全存储，并在后续请求中通过拦截器自动附加至请求头。根据后端返回的角色信息，动态渲染侧边栏菜单和可访问路由。

2. 论文管理模块：为学生角色开发了论文上传（支持拖拽、进度显示）、版本管理、状态查看界面；为教师角色开发了待评审论文列表、在线预览、打分与撰写评语的表单；为管理员角色开发了论文全周期查看与强制状态流转功能。

三、 系统部署实践

我负责了以下环境搭建与配置工作：

1. 前端部署：运行 `npm run build` 生成前端静态资源（位于 `dist` 目录）。将 `dist` 目录放置于服务器，并配置 **Nginx**，将其作为静态文件服务。同时，将所有非静态文件的 **API** 请求代理到后端 **Gunicorn** 服务。此过程涉及 **Nginx** 配置文件的编写，特别是对路由重写和代理规则的仔细调试。

2. 跨域与安全配置：在开发阶段通过 **Vite Proxy** 解决跨域。生产环境中，则在 **Nginx** 和后端 **Django** 中配置了 **CORS** 策略。同时，对 **Nginx** 进行了基础的安全加固，如隐藏版本号、设置安全头部等。

四、 遇到的问题与解决方案

1. 复杂表单状态管理：论文评审表单包含多个评分项和富文本评语，状态变化复杂。最初使用多个`useState`导致逻辑分散。后重构为使用`useReducer`，将所有相关的状态变更集中处理，逻辑变得清晰且易于测试。

2. 生产环境路由刷新 404：React Router 在前端控制路由，但浏览器直接访问非根路径或刷新时，Nginx 会将其当作真实路径请求，导致 404。解决方案是在 Nginx 配置中，将所有前端路由请求重定向到`index.html`。

3. API 联调数据类型不一致：前后端对某些字段（如日期时间、枚举值）的格式约定初期不够明确。通过完善后端 API 文档，并在前端定义清晰的 TypeScript 接口（`interface`）与之对应，建立了强类型的契约，极大减少了联调误解。

五、工作感想与收获

通过这个完整的项目实践，我深刻体会到：

工程化工具的价值：Vite、TypeScript、ESLint、Prettier 等工具链不仅是“用起来时髦”，它们实实在在地提升了开发效率、代码质量和团队协作的规范性。尤其是 TypeScript，它不仅是“有类型的 JavaScript”，更是一种设计思维的转变，促使我在编写函数和组件时更严谨地思考数据流和接口契约。

前后端分离协作模式：清晰定义的 API 接口是前后端并行开发的基石。本次项目中，我们通过契约（接口文档+TypeScript 类型定义）进行协作，降低了沟通成本，使得前后端关注点分离又紧密配合。

解决问题的能力：遇到技术难题时，从官方文档、技术社区、调试工具中寻找答案的过程，极大地锻炼了我的自主学习和问题排查能力。每一次解决问题的经历，都加深了对相关技术原理的理解。

总而言之，本次毕业设计项目对我而言是一次宝贵的实践。它不仅巩固和深化了我的 React、TypeScript 前端技术能力，更让我初步掌握了将一个完整的 Web 应用从开发环境推向生产环境的必备技能。我深刻认识到，构建一个可用的系统需要严谨的代码逻辑，而构建一个健壮、可维护、可部署的系统，则需要全面的工程化思维和对全链路细节的关注。这段经历为我未来的职业生涯打下了坚实而宝贵的基础。

4、陈徐豪

在本毕业设计管理系统的研发过程中，我主要负责前端应用架构的设计与实

现, 以及系统集成部署方案的制定与实施。项目采用前后端分离的现代化架构范式, 前端技术栈基于 **React 18** 框架, 结合 **TypeScript** 的静态类型系统, 并使用 **Vite** 作为核心构建工具, 构建了一个高性能、可维护的单页 **Web** 应用。

在技术架构设计与实现层面, 我主导了前端工程化体系的搭建。通过配置 **TypeScript** 的严格模式与 **ESLint** 规范, 我们建立了高标准的代码质量控制机制。项目采用基于功能与角色维度的模块化组织方式, 将用户界面拆分为可独立开发的领域组件。我设计并实现了一套可复用的 **UI** 组件库, 包含表单控件、数据表格、模态框等基础构件, 这些组件通过 **Props** 与 **Slots** 机制实现了高度的可配置性, 显著提升了开发效率与界面一致性。在与后端的通信层面, 我封装了统一的 **API** 客户端, 利用 **Axios** 拦截器实现了请求认证、错误统一处理与加载状态管理, 确保了网络层的健壮性。针对核心业务模块, 如论文提交流程, 我运用 **React Hook Form** 处理复杂表单状态, 并实现了包含分片上传与进度反馈的文件上传组件。在论文评审模块, 我开发了支持排序、过滤与关键词模糊搜索的增强型数据表格, 优化了教师用户的评审效率。

在系统部署与持续集成方面, 我规划并实施了容器化的本地开发环境。通过编写 **Docker Compose** 配置文件, 将前端开发服务器、**Django** 后端及 **SQLite** 数据库集成在统一的环境中, 简化了团队成员的开发环境搭建流程。为解决跨域资源共享问题, 我配置了 **Nginx** 反向代理, 并优化了前端静态资源的缓存策略与按需加载方案。在部署流水线中, 我引入了环境变量管理机制, 使应用能够根据开发、测试、生产不同环境动态配置 **API** 端点与功能开关。

在项目推进过程中, 我深入解决了若干技术挑战。例如, 为优化大规模论文列表的渲染性能, 我引入了虚拟滚动技术。针对跨组件状态共享的复杂度, 我采用 **React Context API** 与 **useReducer** 组合, 实现了可预测的状态管理。此外, 我通过 **CSS Grid** 与 **Flexbox** 布局模型, 结合媒体查询, 确保了管理界面在多种终端设备上的响应式显示效果。

通过本次全栈工程实践, 我系统性地深化了对现代 **Web** 开发生态的理解。在技术维度, 我掌握了企业级前端应用的架构设计原则、性能优化策略与类型安全实践。在工程管理维度, 我体验了从需求分析、技术选型、模块实现到集成部署的完整软件开发生命周期, 认识到清晰的接口契约、详尽的文档记录与自动化工具链对项目成功的关键作用。展望未来, 我认为系统在状态管理工具(如 **Redux Toolkit** 或 **Zustand**) 的引入、单元测试与端到端测试覆盖率的提升, 以及基于 **GitHub Actions** 的 **CI/CD** 管道建设等方面, 仍有持续的优化空间。

5、王安康

在本学期的《软件工程》课程设计中，我们小组成功开发了“高校毕业设计管理系统”。在此项目中，我主要承担了数据库设计、项目文档撰写以及系统测试三项核心职责。通过全程参与这个基于 **Django** 和 **React** 的真实项目，我不仅将课堂上学到的软件工程理论应用于实践，更在团队协作、技术设计和质量保障方面获得了深刻的体会和宝贵的经验。

我深知数据库是整个系统的数据心脏，其设计的优劣直接关系到系统的性能、稳定性和可扩展性。在项目初期，我积极与负责需求分析的同学沟通，深入理解业务逻辑，明确了学生、教师、管理员不同角色的操作流程和数据实体。在此基础上，我主导完成了数据库的概念模型和逻辑模型设计。

我仔细定义了用户表、论文表、评审表等核心数据表的结构，合理规划了主外键关系，确保数据的一致性和完整性。例如，在设计论文与评审的关联时，我考虑到一篇论文可能被多位教师评审，因此设计了合理的一对多关系。同时，我也充分考虑了字段的类型、约束和索引，为后续后端开发提供了清晰、可靠的数据库蓝图。这个过程让我深刻认识到，前期的周密设计是避免后期返工、保证项目顺利推进的关键。

作为项目的主要文档撰写者，我的任务是将团队的开发成果、技术选型和系统功能清晰、准确地呈现出来。我负责撰写了本项目系统说明文档的绝大部分内容，包括项目概述、人员分工、项目框架详解以及功能模块说明。

在撰写过程中，我不仅需要准确理解后端 **Django** 的 **MVT** 架构、前端 **React** 的组件化思想以及前后端分离的交互方式，还需要用精炼、专业的语言将其描述出来。为了说明登录、论文上传、论文评审等模块，我仔细研究了团队成员编写的代码，提炼其核心逻辑，并配以界面截图，使得文档既具备技术深度，又易于理解。这项工作极大地锻炼了我的技术理解能力、归纳总结能力和书面表达能力，让我体会到规范文档对于软件维护、升级和团队知识传承的重要性。

在功能开发完成后，我投入了大量精力进行系统测试。我与另一位负责测试的同学协作，制定了详细的测试用例，对各个功能模块进行了全面的黑盒测试。这包括但不限于：验证不同角色用户的登录权限、测试学生论文上传的各种边界情况（如文件类型、大小限制）、检查教师评审流程的完整性与数据准确性等。

在测试中，我模拟了多种用户操作场景，成功发现并记录了数个潜在的前端显示问题和后端逻辑漏洞，并及时反馈给开发同学进行修复。通过这个过程，我

切身体会到测试是保障软件质量的最后一道防线，培养了我注重细节、追求完美的工程素养，也提升了从用户角度思考问题的能力。

通过本次项目实践，我全面提升了在数据库设计、文档编写和软件测试方面的实战能力，更深刻理解了软件工程中团队协作的重要性。我认识到，一个成功的项目离不开每个环节的紧密配合和每个成员的尽职尽责。当然，我们也清醒地看到系统仍有提升空间，例如未来可引入更复杂的权限控制、迁移至性能更强的数据库（如 **PostgreSQL**）以及增强前端用户体验等。这些将是我们后续努力的方向。总而言之，本次毕业设计管理系统的开发经历是一次极其宝贵的学习和成长过程，为我未来的职业生涯奠定了坚实的实践基础。