高校毕业设计管理系统 系统说明文档

第七组 吴立文 沈冯毅 陈徐豪 叶一凡 王安康

1. 项目概述

该项目是一个基于 Django 后端和 React 前端的毕业设计管理系统，支持学生、教师和管理员角色。系统允许学生上传论文，教师进行评审并打分，管理员进行全面管理。系统的前后端分离，前端使用 React 和 TypeScript 开发，后端使用 Django 构建，数据库使用 SQLite。通过这一系统，用户能够进行论文查询、评分、评审等操作。

目前系统实现了前端框架，并实现了登录模块、论文上传模块、论文评审模块三大功能模块。

2. 人员分工

吴立文：后端开发、架构设计、系统部署、文档撰写。。

沈冯毅：前端开发、系统部署。

陈徐豪：前端开发、系统部署。

叶一凡：需求分析、文档撰写、系统测试。

王安康：数据库设计、文档撰写、系统测试。

3. 项目框架

3.1. 后端部分

系统的后端采用 Django 框架，后端的入口文件是 manage.py，项目的配置都在 settings.py 中，包括数据库设置、静态文件和媒体文件的路径设置以及已安装应用的列表。开发过程中使用了 SQLite 作为数据库，数据库文件存放在根目录下的 db.sqlite3，这种数据库在开发过程中简洁且便于调试。

系统的核心模块之一是用户管理应用，名为 users。在该应用中，所有用户相关的数据模型都定义在 models.py 文件中，主要包括用户（学生、教师、管理员）及其相关数据（如论文、学科等）。这些数据需要通过 RESTful API 进行处理，API 的序列化器被定义在 serializers.py 中，负责将模型数据转换为 JSON 格式供前端使用，反之也支持将前端发送的 JSON 数据转换为模型对象。在 views.py 文件中，API 控制器和视图被定义，包括用户认证、资料

管理、论文操作等业务逻辑。通过 urls.py 配置路由，所有的 API 端点都被连接到主路由，形成后端系统的整体结构。

3.2. 前端部分

前端部分使用 Vite 构建工具，结合 React 和 TypeScript 开发。Vite 是一种现代化的前端构建工具，相比传统的 Webpack，它具有更快的构建速度和更好的开发体验。项目的入口文件为 main.tsx 和 index.html，这些文件是应用的启动和配置中心。在 App.tsx 中，定义了整个应用的页面骨架，负责根据不同的用户角色渲染对应的仪表盘页面。

系统的页面划分为几个主要部分，包括登录页、管理员仪表盘、教师仪表盘和学生仪表盘。不同的角色拥有不同的权限和页面组件，管理员负责系统管理，教师负责论文评审，而学生则可以提交和查看自己的论文。每个角色的页面组件分别定义在 LoginPage.tsx、AdminDashboard.tsx、TeacherDashboard.tsx 和 StudentDashboard.tsx 中。除此之外，系统的功能组件被按角色分组，分别存放在 admin、teacher 和 student 目录下。这样便于维护和扩展不同角色的功能模块。

3.3. UI 组件与复用

为了保持统一的 UI 风格和提高代码复用性，所有的 UI 组件都被集中在 ui 目录下。该目录中包括了 button.tsx、card.tsx、form.tsx 等组件，它们被广泛应用于整个系统中，确保了各个页面的一致性和可维护性。所有的样式和交互逻辑也都遵循统一的规范，确保用户体验的流畅性和系统界面的美观性。此外，系统的图像和占位符图片都在 figma 目录下存放，供前端开发使用。

前后端交互

在前后端的交互方面，所有的 API 调用都被封装在 api.ts 文件中。通过集中管理 API 请求，能够更方便地处理认证 token、请求基础路径、错误处理等逻辑。这样可以减少重复代码，提高前端开发效率。全局样式被统一管理在 globals.css 中，确保整个应用的样式一致。

3.4. 静态资源与文档

系统中的静态资源包括上传的论文和其他媒体文件，这些文件被组织在 media/ 目录下，按年/月进行组织。这样可以确保文件结构清晰，方便管理和访问。项目的设计文档、界面原型图和使用说明都存放在 docs 目录下，确保

开发人员和设计师能够快速获取相关信息。此外，根目录下的 README.md 文件提供了项目的总体说明，包括如何运行和开发该系统的详细步骤。

3.5. 启动与开发环境配置

在本地开发环境中，后端可以通过以下命令启动：

```
cd backend

python manage.py runserver 0.0.0.0:8000
```

前端则通过以下命令启动：

```
cd frontend

npm install

npm run dev
```

这两个命令将分别启动后端和前端开发服务器，确保前后端能够正确对接。

4. 项目说明

本小节节选三个已实现的模块作重点说明。

4.1. 登录模块

后端核心代码如下：

```python
class LoginAPIView(APIView):
    def post(self, request, *args, **kwargs):
        serializer = LoginSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        identifier = serializer.validated_data['identifier']
        password = serializer.validated_data['password']

        # allow login via student_id (username) or email
        user = None
        try:
            user = User.objects.get(username=identifier)
        except User.DoesNotExist:
            try:
                user = User.objects.get(email=identifier)
            except User.DoesNotExist:
```

```
16.             return Response({'detail': 'Invalid credentials'},
    status=status.HTTP_400_BAD_REQUEST)
17.
18.         if not user.check_password(password):
19.             return Response({'detail': 'Invalid credentials'},
    status=status.HTTP_400_BAD_REQUEST)
20.
21.         token, _ = Token.objects.get_or_create(user=user)
22.         user_data = UserSerializer(user).data
23.         # add display name: prefer last_name+first_name for Chinese
    style
24.         first = getattr(user, 'first_name', '') or ''
25.         last = getattr(user, 'last_name', '') or ''
26.         full = (last + first).strip()
27.         if full:
28.             user_data['name'] = full
29.         else:
30.             full_name = user.get_full_name().strip()
31.             user_data['name'] = full_name if full_name else user.username
32.         return Response({'token': token.key, 'user': user_data})
```

这段代码实现了一个基于 Django REST framework 的登录 API 视图。它继承自 APIView，并重写了 post 方法，用于处理用户登录请求。

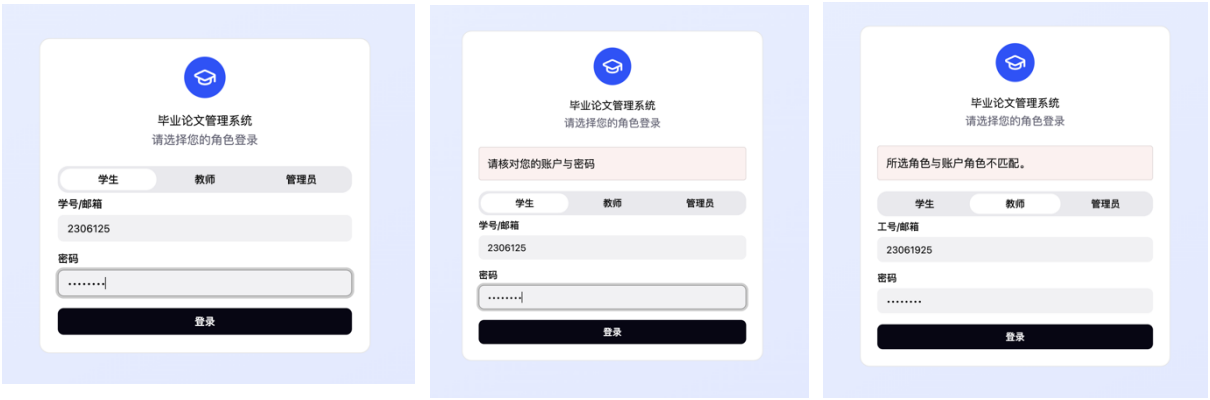
接收到客户端发送的请求数据后，视图会使用 LoginSerializer 对请求数据进行序列化验证，确保传入的数据符合预期格式。如果序列化验证失败，系统会抛出异常并返回错误信息。验证成功后，identifier 和 password 被提取出来，分别代表用户的标识符（可以是用户名或邮箱）和密码。

之后，系统尝试根据提供的标识符（identifier）查找用户。它首先尝试通过用户名查找用户，如果找不到用户，则尝试通过邮箱查找。若两种方式都未找到对应的用户，系统将返回一个“无效凭证”的错误信息，并以 400 错误码响应。一旦找到用户，系统会使用 check_password 方法验证提供的密码是否正确。如果密码不匹配，系统也会返回“无效凭证”错误。

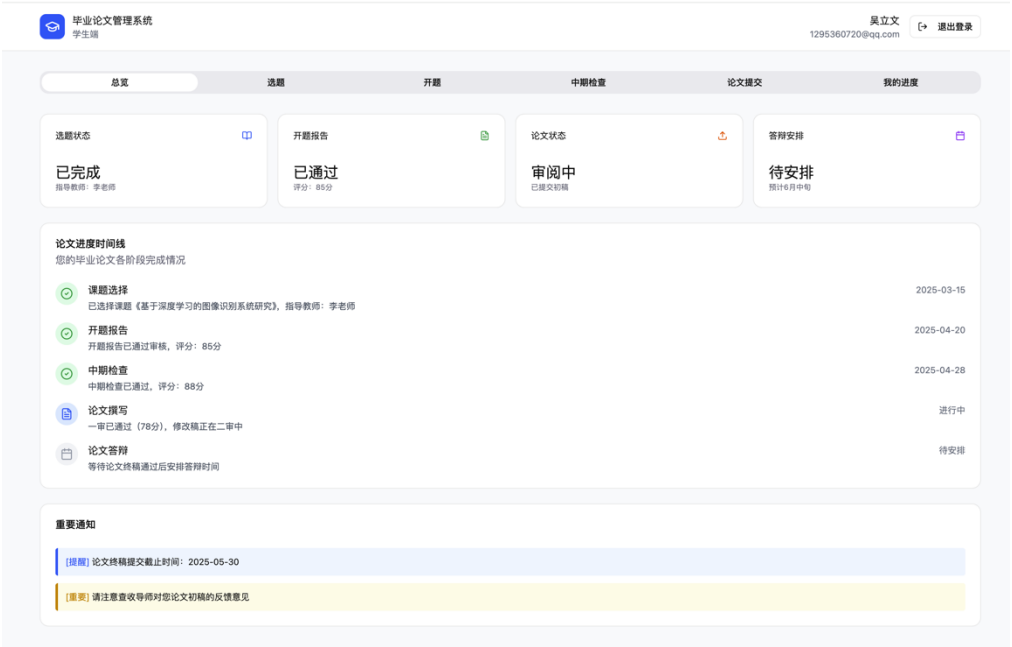
如果用户名和密码都验证通过，系统会生成一个与用户关联的认证 token，使用 Token.objects.get_or_create 方法来获取或创建 token。接下来，系统使用 UserSerializer 将用户数据进行序列化，将其返回给客户端。为了更好地支持中文用户，视图还额外处理了用户姓名的显示格式，优先使用中文姓氏+名字的顺序来构建全名，如果该格式不全，则使用 Django 内建的

get_full_name 方法。最终，API 返回包含用户认证 token 和用户数据的响应。

登录界面如下图所示，可选择以什么身份登录系统，如果密码错误或是选择的身份与系统中存储的账号身份不一致都会输出提示信息。



登录成功则进入用户界面（以学生端为例）。



4.2. 论文上传模块

后端核心代码如下：

```
class ThesisSubmitAPIView(generics.CreateAPIView):
    serializer_class = ThesisSerializer
    authentication_classes = (TokenAuthentication,)
    permission_classes = (IsAuthenticated,)
    parser_classes = (MultiPartParser, FormParser)

```

```
    def get_serializer_context(self):
        context = super().get_serializer_context()
        context['request'] = self.request
        return context

    def create(self, request, *args, **kwargs):
        # Check if user is a student
        if request.user.profile.role != 'student':
            return Response({'detail': 'Only students can submit thesis'}, status=status.HTTP_403_FORBIDDEN)

        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        self.perform_create(serializer)
        return Response(serializer.data, status=status.HTTP_201_CREATED)

    def perform_create(self, serializer):
        serializer.save(student=self.request.user)
```

这段代码实现了一个用于提交论文的 API 视图，继承自 generics.CreateAPIView，用于处理论文提交的创建请求。它首先指定了使用的序列化器 ThesisSerializer，并设置了身份验证和权限检查机制，确保只有经过身份验证的用户才能访问该接口。

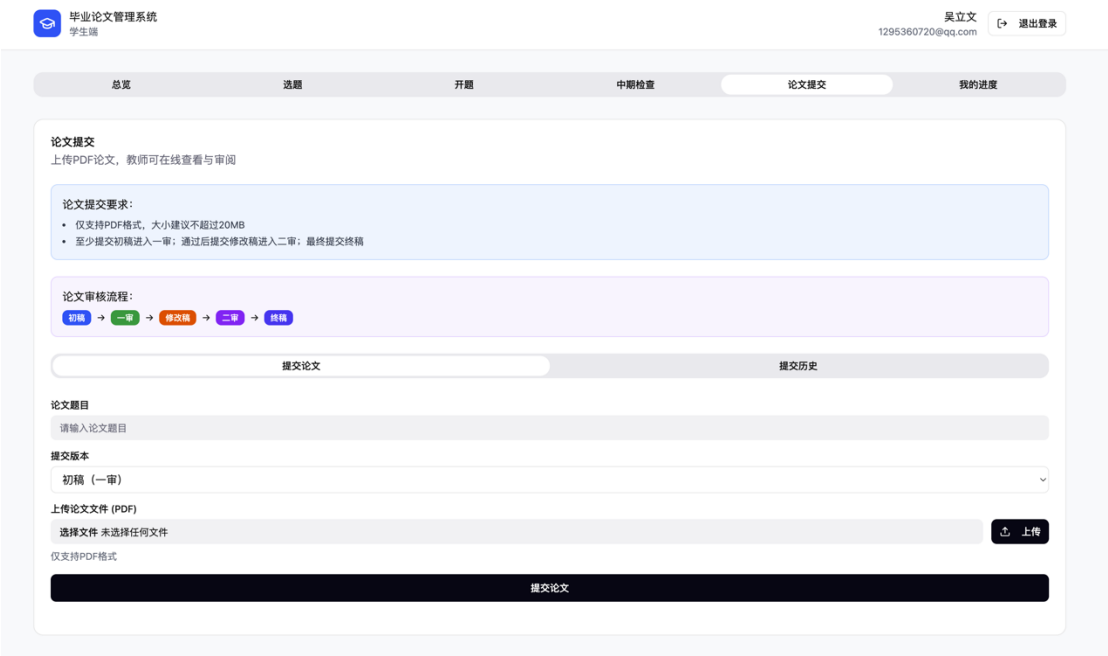
在该视图中，authentication_classes 属性指定了使用 TokenAuthentication，这意味着用户必须提供有效的认证 token 才能访问此视图。permission_classes 属性指定了 IsAuthenticated 权限类，确保请求的用户是已认证的用户。parser_classes 属性则指定了支持的解析器，MultiPartParser 和 FormParser，允许处理包含文件上传的表单数据。

get_serializer_context 方法被重写，用于向序列化器提供额外的上下文数据。在这里，它将当前的 request 对象添加到上下文中，以便序列化器可以访问请求数据。

create 方法是视图的核心逻辑。它首先检查请求用户的角色是否为学生（student）。如果用户的角色不是学生，系统会返回一个 403 禁止访问的错误，提示只有学生可以提交论文。若用户角色合法，系统会通过序列化器验证传入的数据，并调用 perform_create 方法来保存论文数据。在 perform_create 方法中，serializer.save(student=self.request.user) 会将当前用户（学生）关联到提交的论文上，并将论文数据保存到数据库。

如果一切顺利，提交的论文数据将会以 HTTP 201 状态码返回，表示论文提交成功。这个视图提供了一个受限的论文提交接口，只允许学生角色的用户提交论文，确保了角色权限的正确性和数据的完整性。

要使用论文上传模块，需要以学生身份登录，在菜单中选择“论文提交”。效果如下图所示：

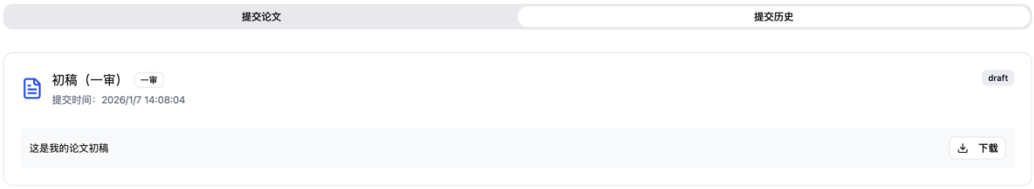


用户可在“提交版本”下拉菜单中选择本次提交的版本：

若未正确选择文件，系统会输出提示：

正确选择并提交网页会输出提示：

同时，在提交历史中可以查看提交了的论文，单击下载即可下载对应论文：



4.3. 论文评审模块

后端核心代码如下：

```
class ThesisReviewAPIView(generics.CreateAPIView):
    serializer_class = ThesisReviewSerializer
    authentication_classes = (TokenAuthentication,)
    permission_classes = (IsAuthenticated,)

    def create(self, request, thesis_id, *args, **kwargs):
        # Check if user is a teacher or admin
        if request.user.profile.role not in ['teacher', 'admin']:
            return Response({'detail': 'Only teachers can review theses'}, status=status.HTTP_403_FORBIDDEN)

        try:
            thesis = Thesis.objects.get(id=thesis_id)
        except Thesis.DoesNotExist:
            return Response({'detail': 'Thesis not found'}, status=status.HTTP_404_NOT_FOUND)

        data = request.data.copy()
        data['thesis'] = thesis_id

        serializer = self.get_serializer(data=data)
        serializer.is_valid(raise_exception=True)
        self.perform_create(serializer)
```

```
22.         return Response(serializer.data, status=status.HTTP_201_CREATED)
23.
24.     def perform_create(self, serializer):
25.         serializer.save(reviewer=self.request.user)
```

这段代码实现了一个用于论文评审的 API 视图，继承自 generics.CreateAPIView，主要用于处理论文评审的创建请求。它通过 ThesisReviewSerializer 来处理提交的评审数据，同时设置了身份验证和权限检查机制，确保只有有权限的用户才能访问该视图。

视图通过 authentication_classes 属性指定了使用 TokenAuthentication，要求用户提供有效的认证 token 来进行身份验证。permission_classes 属性设置了 IsAuthenticated，即只有经过身份验证的用户才能访问此接口。

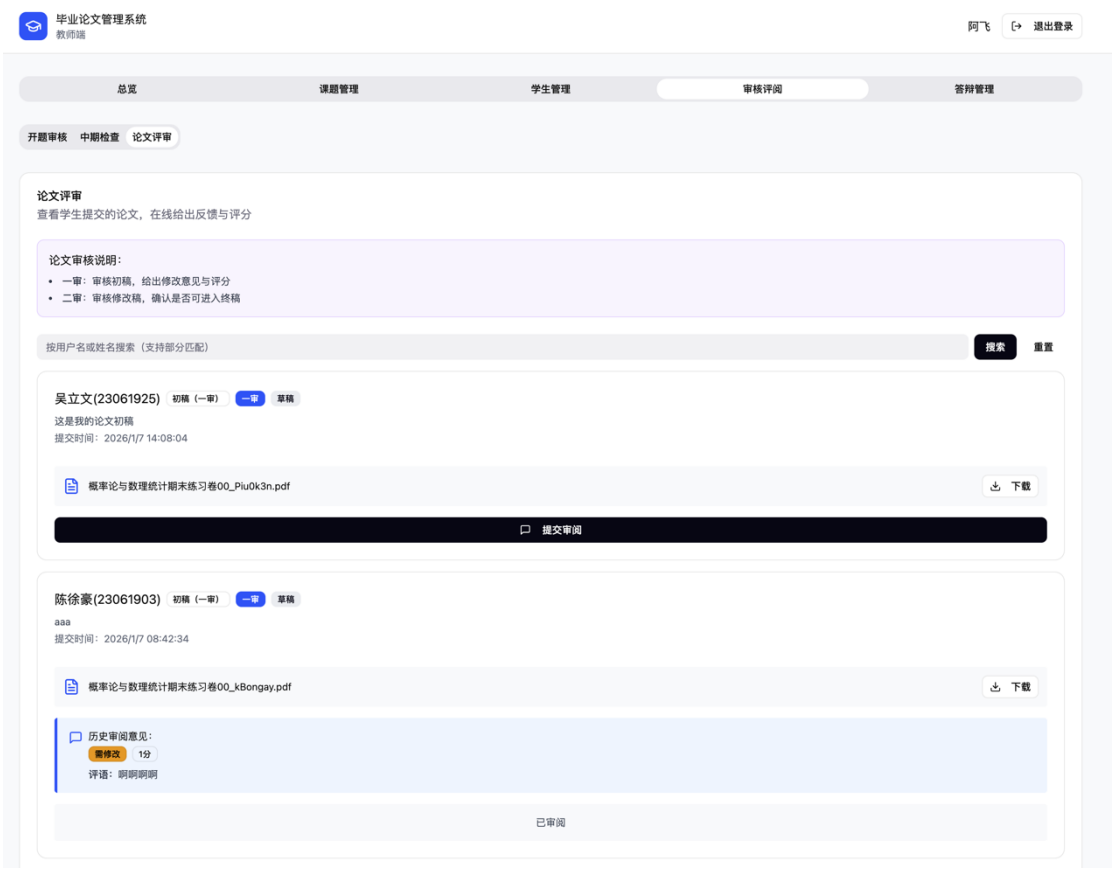
create 方法是视图的核心逻辑，首先会检查当前请求用户的角色，确保用户是教师（teacher）或管理员（admin）。如果用户的角色不是这两者之一，视图会返回一个 403 禁止访问的错误，提示只有教师或管理员才能进行论文评审。

接下来，视图会尝试从数据库中获取指定 ID 的论文（thesis_id）。如果该论文不存在，系统会返回 404 错误，提示论文未找到。如果论文存在，则会将提交的请求数据进行复制，并将论文 ID 加入到数据中，确保评审数据与特定论文关联。

然后，视图通过序列化器 ThesisReviewSerializer 对传入的数据进行验证，若验证成功，调用 perform_create 方法保存评审数据。在 perform_create 方法中，使用 serializer.save(reviewer=self.request.user) 将当前登录的用户（即评审人）与该评审记录关联，并将其保存到数据库中。

最后，如果论文评审创建成功，视图将返回评审数据，状态码为 201，表示创建成功。整体来看，该视图实现了一个论文评审的提交接口，只允许教师和管理员提交评审记录，确保了权限管理的正确性，同时保证了评审数据与论文的正确关联。

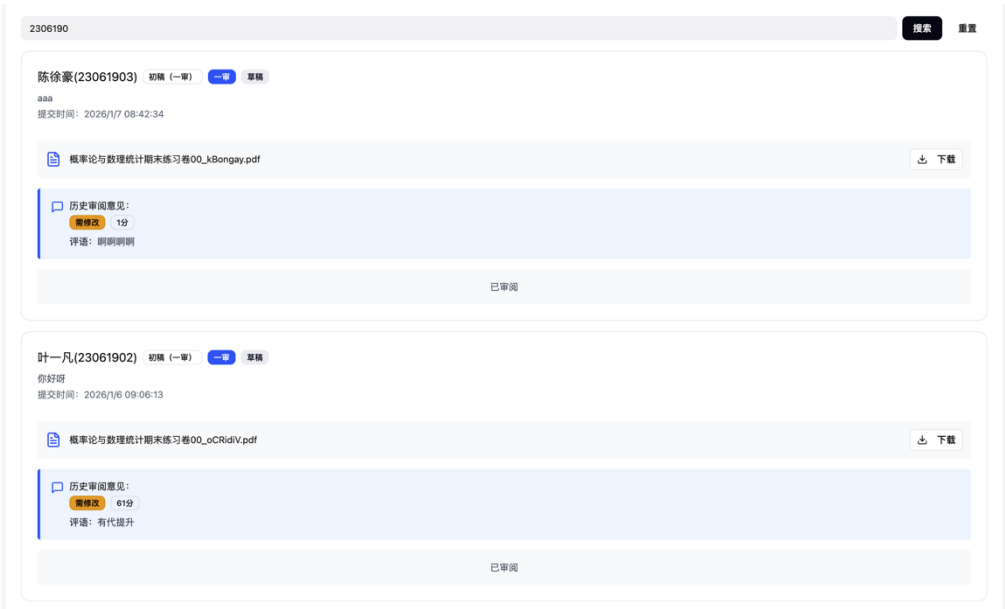
要使用论文评审模块，需要以教师身份登录，在菜单中选择“审核评阅”，在子菜单中选择“论文评审”，便会出现待评审、已评审的论文。如下图所示：



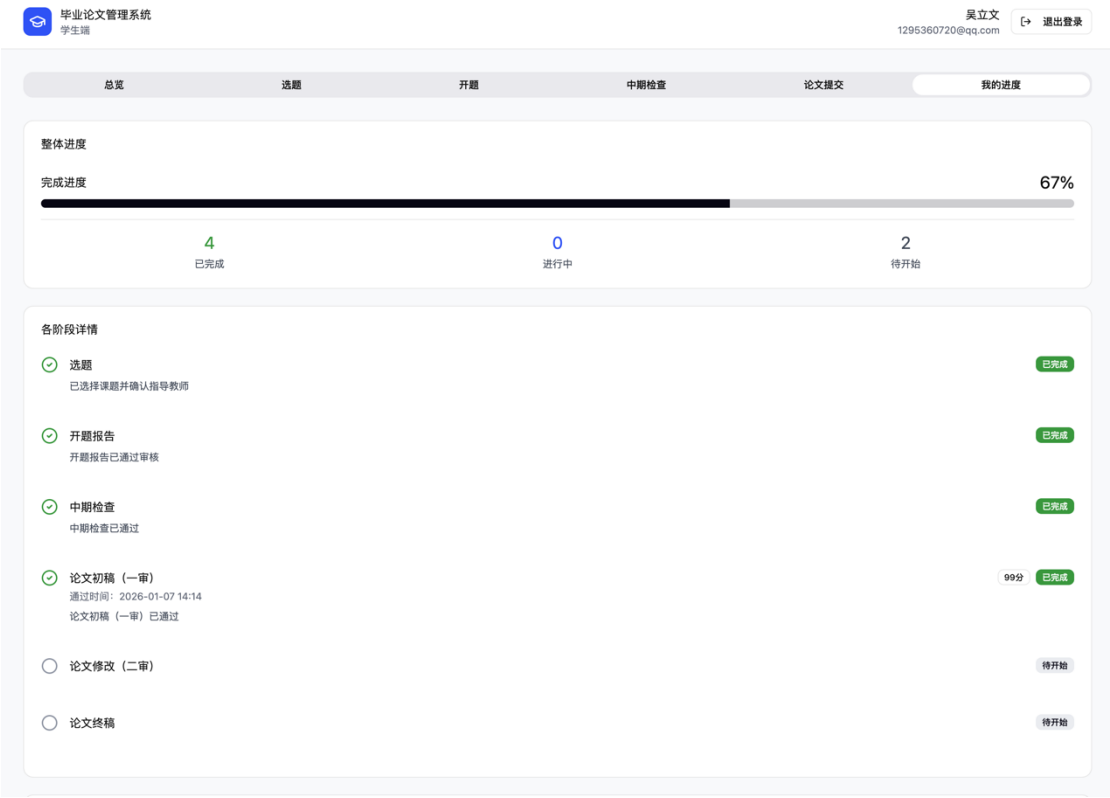
点击“提交审阅”便会弹出评审菜单，如下图所示：

填写相关信息，并提交后，便完成了审阅，如下图所示：

UI 中的搜索框支持输入姓名或学号搜索对应内容。还支持部分关键词搜素。输入一个学号前缀，搜索结果如下：



待教师提交评审结果后，学生可以查看评审结果，评审结果还会同步到学生的进度管理上。以学生身份登录系统，在菜单中选择“我的进度”即可查看：



5. 小结

目前系统只实现了基本的功能模块，包括论文上传、论文评审及用户认证等，部分功能和性能有待进一步开发和提升，具体如下：

系统的用户权限管理可以更加细化，例如引入更多的用户角色和权限配置，以适应更复杂的应用场景；

系统目前的数据库使用 SQLite，虽然适合开发阶段，但在生产环境中，可能需要迁移到更高效、可扩展的数据库系统，如 PostgreSQL 或 MySQL，以提升系统的处理能力和数据一致性。

前端的用户体验方面也可以进一步优化，例如通过增强表单验证、改进错误提示信息和界面交互，提高系统的可用性。此外，论文评审模块虽然已实现基础功能，但未来可以考虑加入更多的评审规则和评估指标，使得评审过程更加多样化和智能化。

系统的安全性也是一个需要进一步强化的方面，特别是在数据传输和存储过程中，采用更严格的加密方式和安全认证机制将有助于保护用户隐私和数据安全。

为了提升系统的可维护性，未来的开发可以考虑引入更多的自动化测试，确保每次功能更新后的系统稳定性，并在开发过程中加入更加详细的日志记录和监控机制，以便及时发现和解决潜在问题。

总的来说，该系统已经具备了基本的功能框架，但仍有许多方面需要在后续开发和优化中完善，以提供更加优质、稳定和安全的服务。