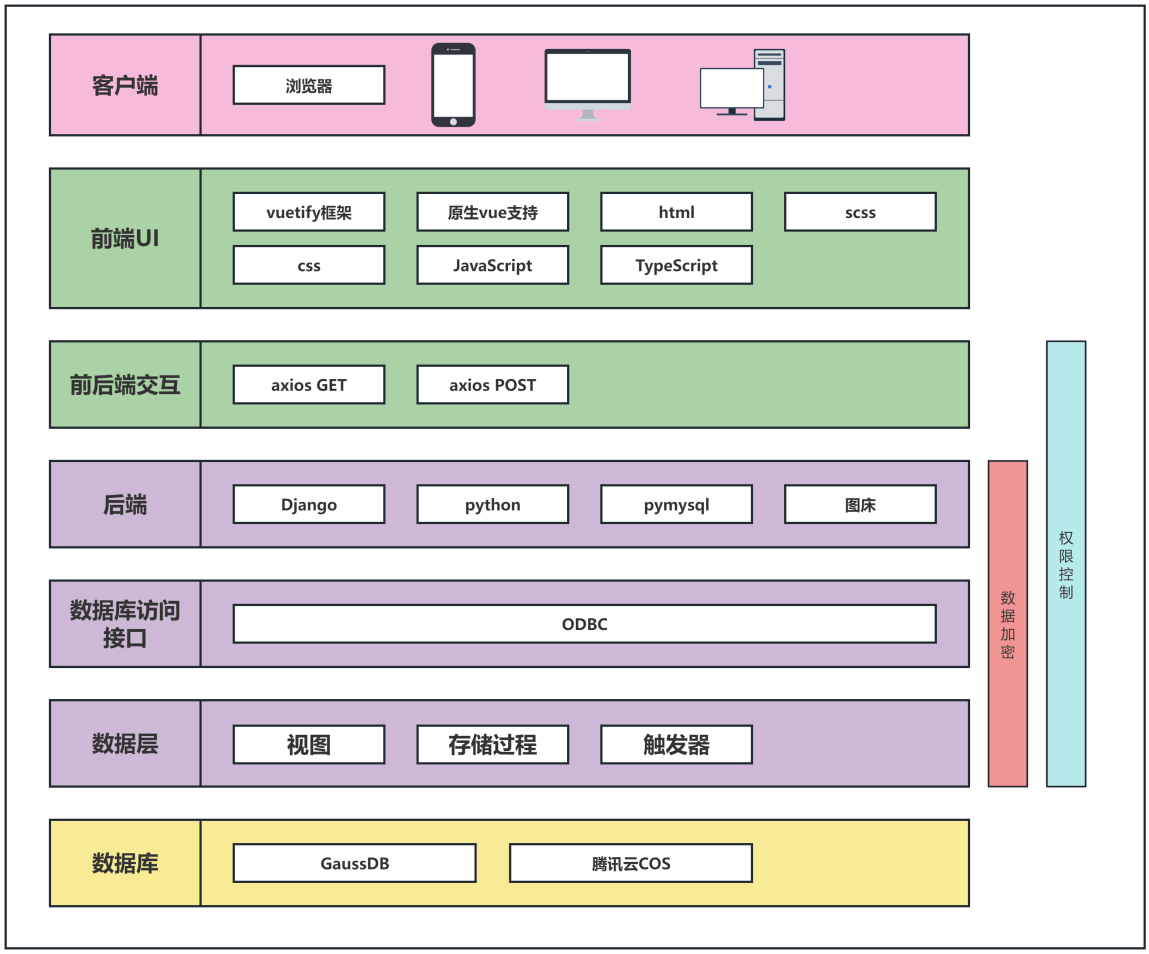
一、系统结构设计

1.1 体系结构

1.1.1 总体结构

项目的总体架构如下：



本项目的应用开发可以分为三个部分，分别是前端开发、后端开发和文档开发。

1.1.2 前端结构

assets

- 用于存储静态资源。
- 包括网站的背景图片和各种样式。

components

- 根据功能或实现方式划分的相互独立的 Vue 组件。
- 分为三部分：总体组件、用户界面组件和管理员界面组件。用户界面组件和管理员界面组件是一一对应的，但是在功能上稍有不同。一般而言，管理员界面的权限较高。
- 总体组件包括登录注册组件、访问权限控制组件、未知页面组件、侧边栏组件、个人信息组件。
- 用户界面组件包括通知和主页组件、讨论区浏览组件、讨论区详情组件、讨论区发帖组件、模拟考试浏览组件、模拟考试详情组件、已公布考试组件、题库浏览组件、题库详情组件、做题组件。

- 管理员组件包括通知和主页组件、讨论区浏览组件、讨论区组件、讨论区发帖组件、模拟考试浏览组件、模拟考试添加题目组件、模拟考试批改浏览组件、模拟考试批改详情组件、题库浏览组件、修改考试组件、修改题目组件、修改题库组件、新通知发布组件、创建题目组件、创建题库组件、创建考试组件。

router

- 控制路由的`index.js`文件
- 采用二级路由机制，在单页面应用（SPA）中，通过配置父级路由和子级路由的嵌套关系，实现页面组件的层级化组织和显示。

store

- 注册信息和用户`Token`的存储。注册时的基本信息会被暂存，便于在注册密码页面发送请求时调用。用户登录后的`Token`存储，用于鉴权操作。
- 状态管理的实现。`state`存储用户`ID`、用户信息，`mutations`提供一系列方法，用户同步更新状态，`getters`用于派生计算状态数据，`modules`用于管理提示信息组件的状态。

styles

- 设置容器为相对定位，并通过`padding-left`预留空间，用于显示行号。行号区域与代码内容分开，保证布局整齐。
- 通过伪元素`::before`在容器左侧添加行号区域，并初始化行号计数器。行号区域具有固定宽度、右边框、浅背景色和柔和文字颜色，用户不可选中。
- 每行代码以块状显示，并通过计数器自动增加行号值。伪元素`::before`为每行动态生成行号，与代码对齐且右对齐显示。

plugins

- 定义`registerPlugins`函数，用于注册插件（如`Vuetify`和路由器`router`），并在主入口文件`main.js`中自动调用，确保插件的统一管理和加载。
- 配置`Vuetify`框架，包含样式文件的引入（如`Material Design`图标和`Vuetify`样式）以及通过`createVuetify`初始化主题，提供应用的`UI`框架支持。

1.1.3 后端架构

使用 Django 框架实现的后端服务，包含核心配置和业务逻辑代码，文件结构如下：

`backend`

- `asgi.py`和`wsgi.py`：项目部署的入口文件，分别用于异步服务器和传统服务器部署。
- `settings.py`：项目配置文件，包括数据库配置、静态文件路径等全局配置项。
- `urls.py`：定义全局路由，将请求分发至不同的应用模块。

`app`
包含各业务模块，每个模块都组织成 Django 子应用的形式，基本结构如下：

- `admin.py`：用于将模型类注册到 Django 管理后台。
- `apps.py`：定义子应用的配置，作为 Django 项目的一部分。
- `models.py`：定义与数据库表对应的模型类，处理数据库操作。
- `urls.py`：定义应用内的路由规则，分发到具体的视图函数。
- `views.py`：视图函数，接收前端请求并返回相应的处理结果。

- `migrations` 文件夹：自动生成的数据库迁移文件，用于同步模型与数据库结构。

`users` 模块
与用户管理相关的逻辑模块：

- 用户模型：包含用户的基本信息和角色管理。
- 序列化功能：用于将用户数据格式化为 JSON 等格式进行传输。
- 黑名单功能：提供用户黑名单相关的管理功能。

`utils` 模块
提供通用的工具类和常量：

- 工具函数：处理常用功能，如加密、Token 生成等操作。
- 常量定义：项目中通用的配置和常量值。

功能模块
包含项目的核心业务功能：

- `board`：公告板功能，展示全局或局部通知信息。
- `broadcast`：广播消息功能，用于推送实时信息。
- `discussions`：讨论区功能，支持发帖和回复操作。
- `exams`：模拟考试功能模块，管理考试的创建、编辑和参与。
- `questions`：题库模块，管理考试题目、题库分类等内容。

静态文件与媒体文件

- `static` 文件夹：存放静态资源（如头像、图片等）。

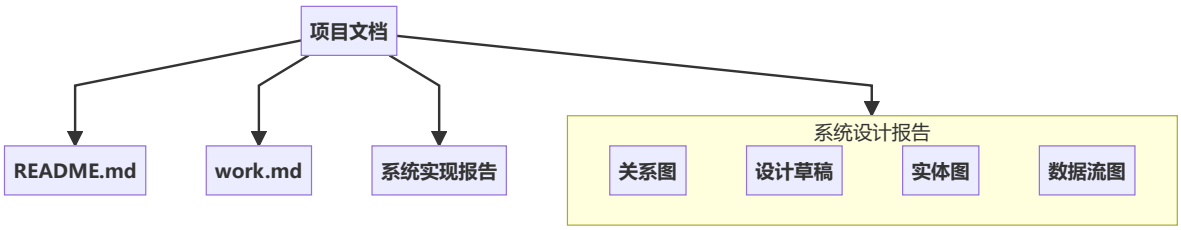
项目根目录文件

- `manage.py`：Django 管理脚本，用于启动开发服务器、执行数据库迁移等操作。
- `db.sqlite3`：SQLite 数据库文件，存储开发环境中的项目数据。
- `requirements.txt`：列出了后端运行所需的 Python 包及其版本信息。

1.1.4 文档架构

- 系统设计报告：描述系统设计中的各个细节。
    - 关系图：用E-R图描述项目中各个实体之间的关系
    - 设计草稿：一份大家商讨出来的设计草案。
    - 实体图：描述项目中出现的实体和属性。
    - 数据流图：描述各种数据的流动和传递方向。
- `README.md`：描述项目的启动方式以及项目的详细介绍。
- `work.md`：用于小组成员同步工作进度。
- 系统实现报告：描述最终系统实现的方法。

架构图如下所示：



1.2 系统实现环境

1.2.1 客户端环境

banhang@0.1.0 ~/db/project/web

├── @kangc/v-md-editor@2.3.18
├── @mdi/font@7.4.47
├── apexcharts@4.1.0
├── axios@1.7.7
├── core-js@3.37.1
├── echarts@5.5.1
├── highlight.js@11.10.0
├── highlightjs-line-numbers.js@2.8.0
├── list@2.0.19
├── markdown-it@14.1.0
├── markdown-it-prism@2.3.0
├── marked@14.1.2
├── moment@2.30.1
├── prismjs@1.29.0
├── roboto-fontface@latest
├── vditor@3.10.6
├── vue@3.4.31
├── vuetify@3.6.11
├── vuex@4.0.2
├── @vitejs/plugin-vue@5.0.5
├── eslint@8.57.0
├── eslint-config-standard@17.1.0
├── eslint-plugin-import@2.29.1
├── eslint-plugin-n@16.6.2
├── eslint-plugin-node@11.1.0
├── eslint-plugin-promise@6.4.0
├── eslint-plugin-vue@9.27.0
├── sass@1.77.6
├── unplugin-fonts@1.1.1
├── unplugin-vue-components@0.27.2
├── unplugin-vue-router@0.10.0
├── vite@5.3.3
├── vite-plugin-vuetify@2.0.3
└── vue-router@4.4.0

1.2.2 服务器端

- **开发语言：Python**

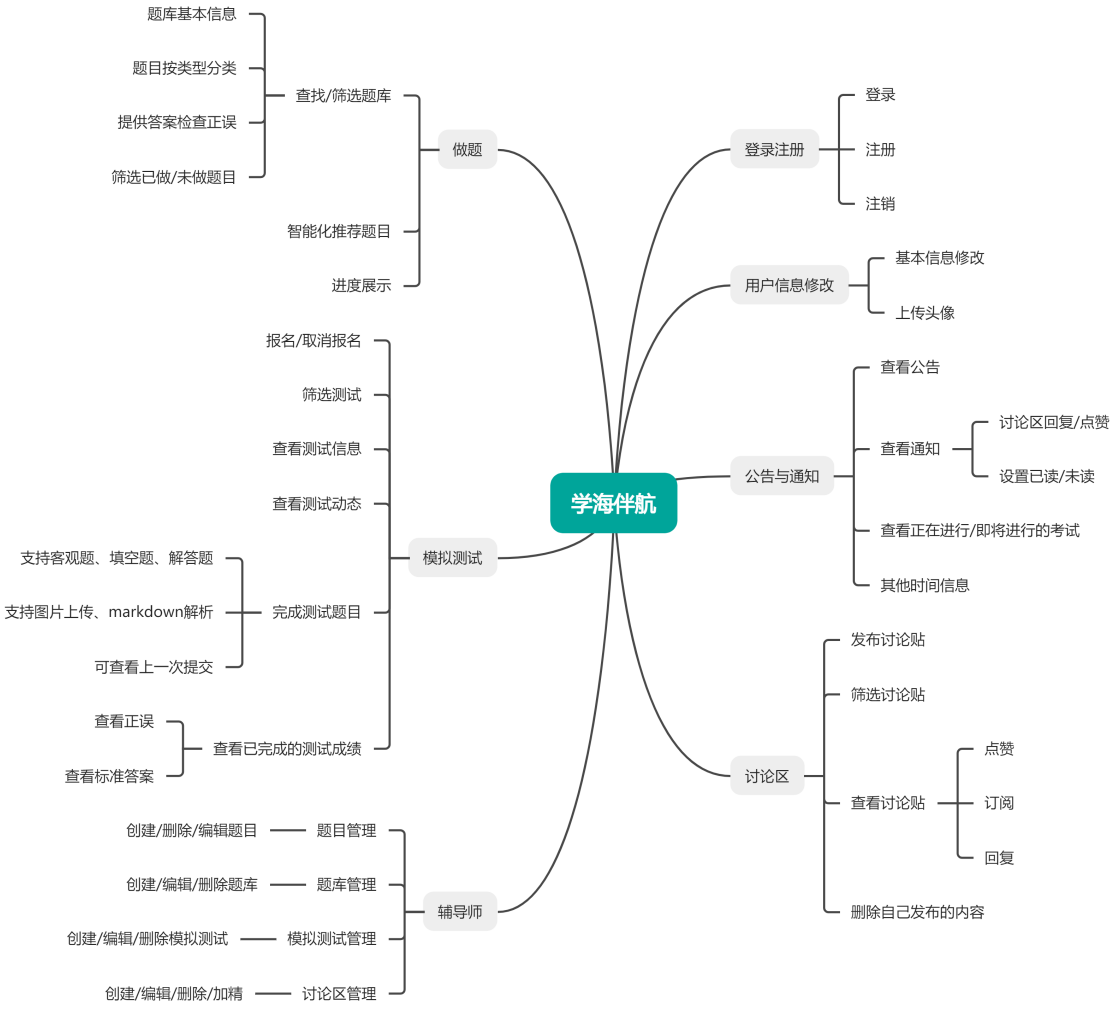
  依赖包

```
Django>=3.2,<4.0
djangorestframework>=3.11,<4.0
gunicorn>=20.0,<21.0
psycopg2>=2.8,<3.0
pyjwt==2.6.0
django-cors-headers
```

- **Web开发框架：**3.2.25
- **DMBS：**MySQL

1.3 功能架构

项目架构的示意图如下：



1.3.1 登录注册

- 登录和注册整体写在一个功能页面之下。
- 登录：可以通过输入用户名和用户名对应的争取密码，登录进入用户或者管理员主页界面。否则，将无法进入主页，自动跳登录注册页面。

- 注册：通过一个弹窗，允许用户输入自己的昵称、学工号（不同用户不同，可以当成主键）、学院、邮箱、入学年份和密码。待系统查验无误后，将这些信息同步到数据库中。
- 登出：用户登录后可以选择登出。登出后用户会丧失访问所有功能界面的权限。

1.3.2 用户信息修改

- 本系统提供了图床功能，用户可以上传自己的头像，并将其显示在左侧的侧边栏中。
- 用户还可以修改个人信息，包括昵称、学院、入学年份和邮箱等。
- 所有修改将实时同步至数据库，确保信息即时更新。

1.3.3 公告和个人信息显示

- 系统支持每位用户查看最近发布的5条公告，方便用户及时了解最新动态。
- 用户可查看自己关注、发表或订阅的讨论的最新更新，并支持将通知标记为已读或未读，同时提供一键标记全部消息的功能，操作便捷。
- 系统能够为用户智能推荐易错练习，帮助强化薄弱知识点。
- 调取用户的做题记录，并生成详细的做题进度表，配合可视化分析，为用户提供清晰的学习数据支持。
- 系统结合特色功能，用户可查看正在进行和即将开始的模拟测试，方便合理安排学习时间。
- 管理员可切换角色，分别进入用户界面和管理员界面，灵活管理系统功能。
- 系统显示当前时间，包括年月日、时、分、秒，为用户提供精确的时间信息。

1.3.4 用户做题

- 用户可以进入个人题库，并根据科目和时间筛选题库中的题目，快速找到所需内容。
- 在做题页面详情中，用户可以查看题目的标题、所属科目、预计用时、创建信息、创建者、题目概述等信息。题目详情按照类别逐项展开，同时通过颜色标注显示每道题是否已被作答。
- 每道题目提供详细信息，包括题面、所属学科、添加时间、来源、标签、类型和难度等。在完成题目后，用户可以查看答案并核对结果。
- 做题记录将实时同步至系统，方便在主页中展示进度并推荐相关题目，帮助优化学习体验。

1.3.5 用户模拟测试

- 模拟测试界面提供了两大功能：考试报名和考试成绩查看。
- 可报名的考试以卡片形式展示，每张卡片包含测试题目、创建日期、所属科目、开始时间和时长等信息。用户可以报名参加考试，也可取消报名，并在报名后进入考试。
- 考试成绩查看功能以卡片形式显示所有已出成绩的考试结果，用户可以通过科目和时间筛选模拟测试，更加方便快捷。
- 进入测试后，界面会显示测试名称、所属科目、时长、测试开始时间、题目数量、已完成题目数量以及剩余时间。测试界面设计了动态变化的进度条和倒计时时钟，直观地显示剩余时间。一旦时间结束，无论用户停留在做题界面还是主页，系统都会自动退出测试，保障测试流程的规范性。
- 用户可以通过点击测试中的题目进入详细的测试题目界面。测试题目界面包含题目的详细描述及选项信息，系统会记住用户上一次的选项。针对不同题型（如多选、单选、填空、判断、解答），测试界面提供不同的展示方式，用户可以选择提交答案或清除答案。在解答题部分，用户还可以上传图片，图片将自动保存至系统图床，并支持多图片上传功能，用户也可以下载自己上一次上传的图片。

- 在测试成绩公布详情界面，用户可以查看测试题目的详细信息，与测试页面内容一致，题目按照题型依次排列。同时，详情界面还展示了用户的测试总成绩及正确题目的数量，帮助用户全面了解考试结果并进行复盘。

1.3.6 用户讨论区

- 用户讨论区主页展示所有讨论帖的浏览区域，用户可以通过标签、时间或是否为精华帖进行筛选。每条讨论帖清晰显示所属科目、是否为精华帖、发布者、发布时间及发布内容等信息。
- 无论是管理员还是普通用户，都可以在讨论区中发帖。发帖时，用户需填写标题，选择所属科目并输入帖子内容。系统支持上传图片功能，图片可直接在编辑界面中显示。此外，发帖页面支持 `markdown` 语法，功能媲美一个小型的 `markdown` 编辑器，为用户提供更丰富的编辑操作。
- 在用户讨论区详情页面，用户可以对帖子和评论进行点赞、订阅和评论，同时可查看帖子的发布时间及最后一次编辑时间。用户也可以编辑自己的评论内容。订阅帖子后，用户可以实时跟踪该帖子的动态更新。页面提供返回按钮，方便用户一键回到讨论区主页。
- 讨论区支持图片的预览功能。

1.3.7 管理员主页

- 系统支持管理员发布新公告，以及对公告进行编辑和删除操作。
- 发布公告的功能逻辑与用户发布讨论帖相似，支持丰富的编辑操作，这里不再详细说明。
- 在管理员主页，管理员可以滑动浏览所有公告，并随时进行编辑或删除操作，方便高效地管理公告内容。

1.3.8 管理员题目管理

- 在管理员的题目管理主页，系统按照科目分类展示题库中的所有题目，管理员可以通过点击“创建新题目”按钮新增题目。
- 在创建题目界面，管理员可以选择题目的所属学科、来源、标签、难度和题型。如果是选择题，还需设置选项数量，选项个数必须多于2个。管理员可在界面中设置题目内容和答案，题面编辑支持上传图片等功能，操作逻辑与用户发布讨论帖相似。在所有信息设置完成后，系统将进入题目预览页面，管理员可在此一次性完整查看题目，包括所属学科、题型、来源、标签、难度、题面和答案等信息。管理员确认无误后，可点击“提交”按钮完成题目创建。系统会自动检查所有输入内容的正确性，如发现错误，将提示具体问题并指导管理员正确设置题目。
- 管理员还可以点击具体题目进入详情界面，查看题目的原始信息。详情界面与预览界面功能一致，管理员可基于原始信息修改题目内容，包括学科、来源、标签、难度、题面及答案等。修改完成后，管理员可预览调整后的题目，并通过点击“提交”按钮将修改内容同步到系统。此外，管理员还可通过点击“删除”按钮直接删除题目。

1.3.9 管理员题库管理

- 在管理员的题目管理主页，系统以卡片形式展示题库的基本信息，管理员可以通过科目和时间筛选题库内容，还可以点击“创建题库”按钮新增题库。
- 在“创建新题库”功能中，管理员可输入题库名称、学科、时长和题库描述等基本信息，并从系统中对应学科的所有题目中选择题目加入题库。系统提供了便捷的交互设计：左键点击题目按钮可进入题目详情界面，右键点击则可直接将题目添加至题库。同样，已添加的题目也支持右键点击从题库中移除。设置完题目后，系统提供题库预览界面，展示题库的名称、学科、时长、题面和描述等信息，方便管理员核对。确认无误后，管理员可提交题库，系统会将其同步保存。
- 管理员还可以对题库进行编辑和删除操作。在编辑题库功能中，管理员可修改题库的基本信息和题目信息，包括新增或移除题目等。具体的增删题目、修改信息及预览功能与创建新题库的流程类似，这里不再赘述。

1.3.10 管理员模拟测试管理

- 在管理员的模拟测试管理主页，系统将模拟测试划分为三类：进行中的测试、即将进行的测试和已结束的测试。管理员可以通过点击主页中的“创建模拟测试”按钮新增测试。同时，系统支持将所有已结束的测试自动同步到题库中，便于学生在考后复习练习，提高熟练度。
- 在“创建新模拟测试”界面，管理员可以设置测试的基本信息，包括名称、学科、开始时间和时长等。开始时间以日历形式选择，支持精确到年月日，同时系统会自动同步当前的时分秒。在设置题目和预览界面，操作逻辑与管理员题库管理中的“创建新题库”功能一致，这里不再详细说明。
- 模拟测试还支持编辑和删除功能。模拟测试编辑与创建流程基本相同，管理员可以修改测试的基本信息以及题目信息，方便对测试进行调整和优化。

1.3.11 管理员模拟测试评分

- 在管理员的模拟评分主页，系统以卡片形式显示所有已结束的模拟测试，管理员可以通过考试名称、科目和状态等条件筛选测试，并支持分页浏览，便于快速查找目标测试。
- 在批改详情界面，管理员可以查看该测试的详细信息，包括题目、所属科目、时长、测试开始时间以及题目数量等内容。同时，题目按照题型依次展开，管理员可点击题目按钮进入题目详情界面。对于单选题、多选题和判断题，系统会自动完成评分；对于填空题和解答题，管理员可点击题目按钮进入具体的批改页面。
- 在批改页面中，系统显示所有学生对该题的答案，管理员可以通过滚动逐一批改，也可筛选特定学生的答案进行修改。在批改界面，管理员可以标记每位学生的答题正确与否并保存批改结果。同时，页面提供标准答案信息，方便管理员核对，提高批改效率。
- 批改完成后，管理员可点击“公布成绩”按钮，一键发布测试成绩，学生即可在个人界面查看已结束考试的成绩。

1.3.12 管理员讨论区

- 整体逻辑与用户讨论区相似，但管理员拥有更多管理功能。
- 管理员可以为讨论区中的某个讨论帖设置“加精”或取消“加精”状态，还可以编辑所有讨论区内的讨论帖和评论，确保讨论区内容的规范性，避免敏感信息对学生学习产生影响。此外，管理员可以删除所有回复和讨论帖，有效维护评论区的正常秩序，确保讨论区的健康发展。

二、数据基本表定义

数据库由9个实体和14张表组成，采用Django框架提供的ORM（对象关系映射）接口来定义和访问数据库表结构。与直接编写原生SQL语句相比，Django的ORM不仅语义更加直观，还极大地提升了代码的可读性和可维护性。每张表通过一个模型类进行抽象定义，模型类的属性与数据库表的字段一一对应，确保了业务逻辑与数据存储的无缝对接。具体的定义和实现代码如下。

2.1 主页部分

系统的主页功能提供了高效的信息发布与管理体验，用户可以查看最近发布的5条公告，及时获取最新动态。管理员则可以发布新公告、编辑和删除现有公告，通过滑动浏览功能轻松管理所有公告。主页的发布逻辑与讨论帖类似，支持灵活的编辑操作，方便管理员快速维护和更新系统信息，实现信息传递的及时性和准确性。

```python
class MyModel(models.Model):
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        abstract = True
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
created_at	datetime(6)	否	否	否	auto_now_add=True	记录创建时间
updated_at	datetime(6)	否	否	否	auto_now=True	记录最后更新时间

这段代码定义了一个抽象模型类 `MyModel`，提供通用的时间戳功能。`created_at` 在记录创建时自动设置时间，`updated_at` 在每次保存时自动更新。在 `Meta` 类中设置 `abstract = True`，表示该模型不会生成数据库表，但可被其他模型继承，用于简化开发。

2.2 公告部分

`Broadcast` 模型用于存储系统中的广播信息，包括广播的标题、内容、发送者、发送时间、最后更新时间等。该表与用户表（`User`）存在外键关联，用于标识广播的发送者。通过该模型，系统能够管理和查询广播消息的相关数据。

```python
class Broadcast(models.Model):
    id = models.AutoField(primary_key=True)  # 主键
    sender = models.CharField(max_length=255, blank=True)  # 发件人名字
    sender_name = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="broadcasts", default=1)
    sent_at = models.DateTimeField(auto_now_add=True)  # 发送时间
    last_updated = models.DateTimeField(auto_now=True)  # 最新修改时间，自动更新
    title = models.CharField(max_length=255, blank=True)  # 标题
    content = models.TextField()  # 消息内容
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	广播的唯一标识
sender	varchar(255)	是	否	否	无	发件人名字（字符串）
sender_name	bigint	否	否	是	关联 users.User 表	关联的发送者
sent_at	datetime(6)	否	否	否	auto_now_add=True	广播的发送时间

字段名称	数据类型	可否为空	主码	外码	其他	说明
last_updated	datetime(6)	否	否	否	auto_now=True	最新修改时间
title	varchar(255)	是	否	否	无	广播标题
content	text	否	否	否	无	广播内容

`Broadcast` 模型提供了与用户表的外键关联，并记录了广播的标题、内容、发送时间等信息，同时支持高效的消息管理和查询。通过 `sent_at` 和 `last_updated` 字段，还能跟踪广播的创建时间和修改时间，为系统实现动态信息更新提供了支持。

2.3 讨论区部分

2.3.1 讨论帖

`Discussion` 模型是用于表示用户在系统中创建的讨论帖的核心数据结构。它包含讨论帖的标题、内容、发布者、发布时间等基本信息，同时支持订阅、点赞以及与特定学科标签相关的分类功能。模型通过外键关联 `User` 模型来记录发帖者，同时通过 `ManyToManyField` 支持记录点赞和订阅的用户列表。此模型还提供了一个 `notify_subscribers` 方法，用于通知订阅者当讨论帖内容更新时生成消息提醒，进一步增强了讨论帖的交互性和动态性。

```
class Discussion(models.Model):
    SUBJECT_CHOICES = [
        ("工科数学分析（上）", "工科数学分析（上）"),
        ("工科数学分析（下）", "工科数学分析（下）"),
        ("工科高等代数", "工科高等代数"),
        ("离散数学（信息类）", "离散数学（信息类）"),
        ("基础物理学A", "基础物理学A"),
    ]

    id = models.AutoField(primary_key=True)
    title = models.CharField(max_length=255)  # 讨论帖标题
    content = models.TextField()  # 主帖内容，支持 Markdown
    publisher = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="published_discussions")  # 发帖者
    avatar = models.URLField(blank=True, null=True)  # 发帖者头像
    publish_time = models.DateTimeField(default=now)  # 发布时间
    isMarked = models.BooleanField(default=False)
    last_updated = models.DateTimeField(auto_now=True)  # 最后更新时间
    tag = models.CharField(max_length=100, choices=SUBJECT_CHOICES, blank=True,
null=True)  # 标签，学科相关
    subscribers = models.ManyToManyField(User,
related_name="subscribed_discussions", blank=True)  # 订阅用户
    likes = models.ManyToManyField(User, related_name="liked_discussions",
blank=True)  # 点赞的用户

    def notify_subscribers(self, update_content, not_send):
        """通知订阅者更新"""
        for subscriber in self.subscribers.all():
            if subscriber != not_send:
                Message.objects.create(
```

```
                    sender=User.objects.get(id=1),  # 系统消息
                    sender_avatar=User.objects.get(id=1).avatar,
                    receiver=subscriber,
                    content=f"您关注的帖子《{self.title}》更新了: \n
{update_content[:5000]}",
                    is_read=False,
                )

    def __str__(self):
        return f"{self.title} - {self.publisher.name}"
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	讨论帖的唯一标识
title	varchar(255)	否	否	否	无	讨论帖标题
content	text	否	否	否	支持 Markdown 格式	主帖内容
publisher	bigint	否	否	是	关联 users.User 表	发帖者
avatar	varchar(255)	是	否	否	blank=True, null=True	发帖者头像
publish_time	datetime(6)	否	否	否	default=now	发布时间
isMarked	int	否	否	否	default=0	是否标记为精华帖
last_updated	datetime(6)	否	否	否	auto_now=True	最后更新时间
tag	varchar(255)	是	否	否	choices=SUBJECT_CHOICES	学科标签
subscribers	bigint	是	否	是	blank=True	订阅该帖子的用户

字段名称	数据类型	可否为空	主码	外码	其他	说明
likes	bigint	是	否	是	blank=True	点赞该帖子的用户

`Discussion` 模型是讨论帖的核心结构，通过精细化的设计实现了记录、扩展、互动和动态更新等功能。模型以 `id` 唯一标识每条记录，包含标题、内容、发帖者、发布时间和最后更新时间等基础信息，确保对讨论帖内容的完整追溯。通过 `tag` 字段实现学科分类，`avatar` 字段记录发帖者头像，增强了内容的展示效果。同时，模型通过 `subscribers` 和 `likes` 字段记录订阅和点赞用户的多对多关系，支持互动性功能，并通过 `isMarked` 字段标记精华帖，方便管理员筛选优质内容。此外，模型提供了 `notify_subscribers` 方法，当帖子内容更新时自动通知订阅者，有效提升了用户体验和信息传达效率。该模型全面覆盖了讨论帖的基本需求，显著提高了讨论区的实用性和可维护性。

2.3.2 回复

`Reply` 模型用于存储讨论帖的回复信息。该模型包含回复的内容、发布者、发布时间、最后更新时间以及点赞用户等信息，同时通过外键关联到 `Discussion` 模型，标识每条回复所属的主帖。模型还提供了通知功能，通过 `notify_publisher` 方法，当主帖收到新回复时，系统会自动向发帖人发送通知，提升互动性。

```
class Reply(models.Model):
    id = models.AutoField(primary_key=True)
    discussion = models.ForeignKey(Discussion, on_delete=models.CASCADE,
related_name="replies")  # 所属主帖
    content = models.TextField()  # 回复内容
    publisher = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="published_replies")  # 回复者
    avatar = models.URLField(blank=True, null=True)  # 回复者头像
    publish_time = models.DateTimeField(default=now)  # 回复时间
    last_updated = models.DateTimeField(auto_now=True)  # 最后更新时间
    likes = models.ManyToManyField(User, related_name="liked_replies",
blank=True)  # 点赞的用户

    def notify_publisher(self):
        """通知发帖人有新回复"""
        Message.objects.create(
            sender=self.publisher,  # 回复者
            sender_avatar=self.publisher.avatar,
            receiver=self.discussion.publisher,  # 收帖人是主帖发布者
            content=f"您发布的帖子《{self.discussion.title}》收到新回复: \n
{self.content[:5000]}",
            is_read=False,
        )

    def __str__(self):
        return f"Reply by {self.publisher.name} on {self.discussion.title}"
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	回复的唯一标识
discussion	bigint	否	否	是	关联 discussions 表	所属主帖
content	text	否	否	否	无	回复内容
publisher	bigint	否	否	是	关联 users.User 表	回复发布者
avatar	varchar(255)	是	否	否	blank=True, null=True	回复者头像
publish_time	datetime(6)	否	否	否	default=now	回复发布时间
last_updated	datetime(6)	否	否	否	auto_now=True	回复最后更新时间
likes	bigint	是	否	是	blank=True	点赞该回复的用户列表

`Reply` 模型作为讨论帖回复的主要数据结构，具备内容管理和用户交互的核心功能。每条记录通过 `id` 唯一标识，字段 `content` 用于存储回复内容，`publisher` 字段记录回复发布者，同时通过外键 `discussion` 明确该回复隶属的主帖关系。模型中还包含 `publish_time` 和 `last_updated` 两个时间字段，用于追踪回复的发布时间和修改时间。此外，`likes` 字段记录点赞该回复的用户，支持互动功能，而 `avatar` 字段则存储回复者头像信息，增强了界面的展示效果。通过 `notify_publisher` 方法，系统会在新回复发布时自动通知主帖发布者，有效提升了互动性和用户体验。该模型设计精简而实用，能够全面支持讨论区的回复管理和交互需求。

2.4 模拟测试部分

2.4.1 考试

`Exam` 模型用于表示系统中的考试信息，是模拟考试模块的核心数据结构。该模型包含考试的基本信息（标题、所属科目、描述）、时间相关信息（创建时间、开始时间、持续时间、结束时间）以及与考试相关的用户和题目数据（创建者、参与学生、考试题目集合）。此外，模型通过 `calculate_end_time` 和 `get_status` 方法动态计算考试结束时间并返回考试状态（未开始、进行中或已结束），进一步增强了模型的实用性和交互性。

```
class Exam(models.Model):
    SUBJECT_CHOICES = [
        ("工科数学分析（上）", "工科数学分析（上）"),
        ("工科数学分析（下）", "工科数学分析（下）"),
        ("工科高等代数", "工科高等代数"),
        ("离散数学（信息类）", "离散数学（信息类）"),
        ("基础物理学A", "基础物理学A"),
    ]

    id = models.AutoField(primary_key=True)  # 主键
    title = models.CharField(max_length=255)  # 考试标题
```

```
    subject = models.CharField(max_length=100, choices=SUBJECT_CHOICES,
default="工科数学分析（上）")  # 所属科目，限制为可选科目
    description = models.TextField(blank=True, null=True)  # 考试描述
    created_at = models.DateTimeField(auto_now_add=True)  # 创建时间
    start_time = models.DateTimeField()  # 考试开始时间
    duration = models.IntegerField(null=False, default=60)  # 考试持续时间（分钟）
    end_time = models.DateTimeField(null=True, blank=True)  # 考试结束时间
    created_by = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="created_exams")  # 创建者（老师）
    is_published = models.BooleanField(default=False)
    is_checked = models.BooleanField(default=False)

    students = models.ManyToManyField(User, related_name="enrolled_exams",
blank=True)  # 报名考试的学生
    questions = models.ManyToManyField(Question, related_name="exams")  # 考试中的
题目集合

    def __str__(self):
        return self.title

    def calculate_end_time(self):
        """
        如果没有显式设置持续时间，则通过 start_time 和 end_time 动态计算
        """
        if self.start_time and self.duration:
            self.end_time = self.start_time + timedelta(minutes=self.duration)

    def get_status(self):
        """
        返回考试的状态：
        - 'coming'：未开始
        - 'ongoing'：进行中
        - 'past'：已结束
        """
        time_now = now()  # 获取当前时间

        # 计算考试的结束时间
        end_time = self.start_time + timedelta(minutes=self.duration)

        if time_now < self.start_time:
            return "coming"  # 未开始
        elif self.start_time <= time_now <= end_time:
            return "ongoing"  # 进行中
        else:
            return "past"  # 已结束
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	考试的唯一标识

字段名称	数据类型	可否为空	主码	外码	其他	说明
title	varchar(255)	否	否	否	无	考试标题
subject	varchar(100)	否	否	否	choices=SUBJECT_CHOICES	考试所属科目
description	text	是	否	否	blank=True, null=True	考试描述
created_at	datetime(6)	否	否	否	auto_now_add=True	考试创建时间
start_time	datetime(6)	否	否	否	无	考试开始时间
duration	int	否	否	否	default=60	考试持续时间（分钟）
end_time	datetime(6)	是	否	否	blank=True, null=True	考试结束时间
created_by	bigint	否	否	是	关联 users.User 表	考试创建者
is_published	boolean	否	否	否	default=False	是否已发布
is_checked	boolean	否	否	否	default=False	是否已批改
students	bigint	是	否	是	blank=True	报名参加考试的学生
questions	bigint	是	否	是	无	考试题目集合

`Exam` 模型作为考试信息的核心结构，设计合理，功能全面，能够满足考试相关数据的存储和操作需求。模型以 `id` 唯一标识每场考试，记录了考试的标题、所属科目和描述等基本信息，同时通过 `created_at`、`start_time`、`duration` 和 `end_time` 字段详细记录了考试的时间安排。模型通过 `created_by` 字段关联考试创建者（通常为老师），并通过 `students` 字段记录报名参与考试的学生列表，同时通过 `questions` 字段管理考试题目集合。此外，`is_published` 和 `is_checked` 字段提供了考试发布和批改状态的标记功能。

该模型的动态方法进一步提升了其实用性：`calculate_end_time` 方法根据开始时间和持续时间动态计算结束时间，而 `get_status` 方法根据当前时间判断考试的状态（未开始、进行中或已结束）。这些设计为考试管理系统的功能实现提供了坚实的数据支撑，极大地提高了考试管理的效率和用户体验。

2.4.2 考试做题记录

`ExamRecord` 模型用于记录学生在考试中的答题情况，是考试系统的重要数据结构。每条记录包含考试、题目、学生、提交的答案、答题正确性以及提交时间等信息，通过外键关联 `Exam`、`Question` 和 `User` 模型，以标识具体考试中的答题数据。模型支持记录是否提交答案、是否批改等状态，并通过 `unique_together` 确保每个学生在每次考试中对每道题仅能提交一次答案。此外，`has_submitted` 方法判断学生是否已提交该题答案，为系统逻辑提供便利。

```
class ExamRecord(models.Model):
    id = models.AutoField(primary_key=True)  # 主键
    exam = models.ForeignKey(Exam, on_delete=models.CASCADE,
related_name="records")  # 所属考试
    question = models.ForeignKey(Question, on_delete=models.CASCADE,
related_name="exam_records")  # 题目
    student = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="exam_records")  # 学生
    submitted_answer = models.TextField(blank=True, null=True)  # 学生提交的答案
    is_correct = models.BooleanField(null=True, blank=True)  # 是否答对（老师批改）
    is_checked = models.BooleanField(default=False)

    submitted_at = models.DateTimeField(auto_now=True)  # 提交时间

    class Meta:
        unique_together = ('exam', 'question', 'student')  # 确保每个学生在每次考试的
每道题只有一条记录

    def __str__(self):
        return f"Exam {self.exam.id} - Question {self.question.id} - Student
{self.student.id}"

    def has_submitted(self):
        """
        判断学生是否提交了该题的答案
        """
        return self.submitted_answer is not None
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	记录的唯一标识
exam	bigint	否	否	是	关联 exams 表	所属考试
question	bigint	否	否	是	关联 questions 表	所属题目
student	bigint	否	否	是	关联 users.User 表	答题的学生

字段名称	数据类型	可否为空	主码	外码	其他	说明
submitted_answer	varchar(255)	是	否	否	blank=True, null=True	学生提交的答案
is_correct	boolean	是	否	否	blank=True, null=True	答案是否正确
is_checked	boolean	否	否	否	default=False	是否已批改
submitted_at	datetime(6)	否	否	否	auto_now=True	答案提交时间

`ExamRecord` 模型设计精简而功能强大，用于记录每场考试中每位学生对每道题的答题信息。通过 `exam`、`question` 和 `student` 字段，模型关联具体考试、题目和答题学生，确保数据结构的层次清晰。`submitted_answer` 字段存储学生提交的答案，`is_correct` 字段记录答题是否正确，`is_checked` 字段用于标识题目是否已批改，这些字段共同支撑了考试系统的批改与反馈功能。同时，`submitted_at` 字段记录提交时间，方便后续数据统计和分析。

模型的 `unique_together` 元信息确保每个学生在每次考试中对每道题只有一条答题记录，避免数据重复或冲突。此外，`has_submitted` 方法判断学生是否提交答案，为考试逻辑提供了便捷的接口。通过这些功能，`ExamRecord` 模型实现了对考试答题数据的高效管理，为系统的考试统计、批改和反馈提供了坚实支持。

2.5 图片上传部分

`UserAvatar` 模型用于管理用户头像文件的存储与记录，是用户资料管理的一部分。模型包含头像文件的标题、存储路径以及上传时间等信息，通过 `FileField` 将头像文件存储在指定的文件目录中，确保文件的有序管理。模型设计简洁高效，能够支持用户头像的上传与展示功能。

```
class UserAvatar(models.Model):
    title = models.CharField(max_length=255)  # 文件标题（名称）
    avatar = models.FileField(upload_to='avatars/')  # 保存头像文件到本地
    created_at = models.DateTimeField(auto_now_add=True)  # 记录创建时间

    def __str__(self):
        return self.title
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	头像记录的唯一标识
title	varchar(255)	否	否	否	max_length=255	头像文件的标题
avatar	varchar(255)	否	否	否	upload_to='avatars/'	存储头像文件的路径

字段名称	数据类型	可否为空	主码	外码	其他	说明
created_at	datetime(6)	否	否	否	auto_now_add=True	记录文件的创建时间

`UserAvatar` 模型设计简洁明了，专注于用户头像文件的管理。每条记录通过 `id` 唯一标识，`title` 字段存储头像文件的标题，便于用户识别与管理；`avatar` 字段通过 `FileField` 设定文件上传路径，将头像文件保存在 `avatars/` 目录下，确保文件存储的规范性；`created_at` 字段记录文件的创建时间，为后续文件管理和排序提供依据。

该模型通过直观的结构和必要的字段设计，满足了用户头像上传、存储与展示的基本需求，为用户资料管理提供了可靠的数据支撑，同时支持扩展性以适应未来功能需求。

2.6 消息通知部分

`Message` 模型用于存储和管理用户之间的消息记录，是系统消息功能的核心数据结构。模型记录了消息的发送者、接收者、发送时间、内容以及消息的已读状态，并通过外键关联 `User` 模型，以标识具体的发送者和接收者信息。同时，模型支持存储发送者头像，便于在消息界面中进行更直观的展示。

```
class Message(models.Model):
    id = models.AutoField(primary_key=True)  # 主键
    sender = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="sent_messages")  # 发件人
    sender_avatar = models.URLField(blank=True, null=True)  # 发件人头像
    receiver = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="received_messages")  # 收件人
    sent_at = models.DateTimeField(auto_now_add=True)  # 发送时间
    content = models.TextField()  # 消息内容
    is_read = models.BooleanField(default=False)  # 已读/未读
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	消息的唯一标识
sender	bigint	否	否	是	关联 users.User 表	消息的发件人
sender_avatar	varchar(255)	是	否	否	blank=True, null=True	发件人的头像链接
receiver	bigint	否	否	是	关联 users.User 表	消息的接收人
sent_at	datetime(6)	否	否	否	auto_now_add=True	消息的发送时间
content	text	否	否	否	无	消息的具体内容

字段名称	数据类型	可否为空	主码	外码	其他	说明
is_read	int	否	否	否	default=0	消息是否已读的状态

`Message` 模型设计直观且功能完备，专注于用户之间消息的记录与管理。通过 `id` 字段唯一标识每条消息，`sender` 和 `receiver` 字段分别标识消息的发件人和接收人，并与用户表建立外键关系，确保消息记录的关联性和数据一致性。`sent_at` 字段记录消息的发送时间，`content` 字段存储消息内容，`sender_avatar` 字段则用于存储发件人的头像链接，方便在前端界面中实现更加人性化的展示效果。同时，`is_read` 字段标记消息是否已读，为实现消息提醒功能提供了支持。

通过简单而高效的字段设计，`Message` 模型满足了用户消息交流和管理的核心需求，同时为系统消息功能的扩展性提供了良好的基础。

2.7 用户管理部分

2.7.1 用户信息

`User` 模型用于定义系统中的用户信息，是用户管理模块的核心数据结构。模型包含用户的学号、姓名、邮箱、学院、入学年份、头像链接和角色等基本信息，同时通过 `password_digest` 字段存储用户密码的加密结果，确保安全性。模型设计支持唯一性约束和索引优化，为用户数据的高效存储和快速检索提供了良好基础。

```
class User(MyModel):
    student_id = models.CharField(max_length=255, unique=True, db_index=True)
    name = models.CharField(max_length=255, db_index=True)
    password_digest = models.CharField(max_length=255)
    mail = models.EmailField()
    college = models.CharField(max_length=255)
    entryYear = models.CharField(max_length=20)
    avatar = models.CharField(max_length=255)
    user_role = models.IntegerField(default=0)

    class Meta:
        db_table = 'users'
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	用户的唯一标识
student_id	varchar(255)	否	否	否	unique=True, db_index=True	学生学号，唯一且有索引
name	varchar(255)	否	否	否	db_index=True	用户姓名，有索引

字段名称	数据类型	可否为空	主码	外码	其他	说明
password_digest	varchar(255)	否	否	否	无	用户密码的加密存储
mail	varchar(255)	否	否	否	无	用户邮箱
college	varchar(255)	否	否	否	无	用户所属学院
entryYear	varchar(20)	否	否	否	max_length=20	用户入学年份
avatar	varchar(255)	否	否	否	max_length=255	用户头像链接
user_role	int	否	否	否	default=0	用户角色（0为普通用户，其他为管理员等）

`User` 模型以 `student_id` 为唯一标识用户，同时通过 `name` 和 `student_id` 设置索引优化查询效率。`password_digest` 字段存储用户密码的加密值，确保用户信息的安全性。`mail` 字段存储用户邮箱，`college` 和 `entryYear` 字段分别记录用户的学院和入学年份，`avatar` 字段保存用户的头像链接。`user_role` 字段用整数表示用户的角色权限，支持不同角色的权限管理。模型的表名设置为 `users`，通过 `Meta` 类统一管理数据库表名称，确保数据存储的规范性和查询的高效性。通过这些字段设计，`User` 模型全面覆盖了用户信息的核心需求，为系统的用户管理功能提供了可靠的数据支撑。

2.7.2 拉黑用户

`BlackList` 模型用于存储被拉黑的令牌（Token），是系统安全管理的重要组成部分。该模型通过 `token` 字段唯一标识被加入黑名单的令牌，用于实现令牌失效和阻止非法访问。模型通过唯一性约束确保每个被拉黑的令牌只存在一条记录，从而保证数据的准确性和一致性。

```
class BlackList(MyModel):
    token = models.CharField(max_length=255, unique=True)

    class Meta:
        db_table = 'black_list'
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	黑名单记录的唯一标识
token	varchar(255)	否	否	否	unique=True	被拉黑的令牌，必须唯一

`BlackList` 模型通过 `token` 字段记录被拉黑的令牌，并对其设置唯一性约束，防止重复记录的产生。模型继承自 `MyModel`，具备创建和更新时间的通用字段（假设 `MyModel` 包含这些字段）。模型的表名设置为 `black_list`，确保数据库中表命名的规范性和可读性。该模型设计简洁明了，能够高效支持令牌管理，确保系统的安全性与可靠性。

2.8 时间组件

`MyModel` 是一个抽象基类，提供了创建时间和更新时间的通用字段，便于在其他模型中继承使用。通过在 `Meta` 类中设置 `abstract = True`，该类不会直接创建数据库表，而是为继承它的子类添加 `created_at` 和 `updated_at` 字段，统一管理时间相关信息，减少重复代码，提高开发效率。

```
class MyModel(models.Model):
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        abstract = True
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
created_at	datetime(6)	否	否	否	auto_now_add=True	记录数据的创建时间
updated_at	datetime(6)	否	否	否	auto_now=True	记录数据的最后更新时间

`MyModel` 的设计为所有继承它的子类提供了统一的时间管理字段。`created_at` 字段在记录首次创建时自动设置为当前时间，`updated_at` 字段在每次保存记录时自动更新为当前时间。通过将该模型设为抽象类，`MyModel` 不会直接生成数据库表，而是作为一个通用模板供其他模型继承使用，既简化了开发过程，又提高了代码的可维护性。

2.9 题目组件

题目组件是整个系统的核心功能组件，有最多的实体和关系。

2.9.1 题库

`QuestionBank` 模型用于存储题库的基本信息，是考试系统题库管理的核心数据结构。该模型记录了题库的名称、所属科目、预计用时、创建日期、创建者、描述及题目数量等信息，通过外键关联到 `User` 模型，以标识题库的创建者。同时，模型支持通过 `subject` 字段对题库进行学科分类，便于管理和筛选。

```
class QuestionBank(models.Model):
    SUBJECT_CHOICES = [
        ("工科数学分析（上）", "工科数学分析（上）"),
        ("工科数学分析（下）", "工科数学分析（下）"),
        ("工科高等代数", "工科高等代数"),
        ("离散数学（信息类）", "离散数学（信息类）"),
        ("基础物理学A", "基础物理学A"),
    ]
```

```
    id = models.AutoField(primary_key=True)  # 主键
    name = models.CharField(max_length=100, default="")
    subject = models.CharField(max_length=100, choices=SUBJECT_CHOICES)  # 所属科
目，限制为可选科目
    estimated_time = models.IntegerField()  # 预计用时（分钟）
    created_at = models.DateTimeField(auto_now_add=True)  # 创建日期
    creator = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="created_question_banks")  # 创建者
    description = models.TextField(blank=True, null=True)  # 题库描述
    question_count = models.IntegerField(default=0)  # 题目数量（可定期更新）

    def __str__(self):
        return f"{self.subject} - {self.id} - {self.question_count} questions"
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	题库的唯一标识
name	varchar(100)	否	否	否	default=""	题库名称
subject	varchar(100)	否	否	否	无	题库所属科目
estimated_time	int	否	否	否	无	预计完成题库所需时间（分钟）
created_at	datetime(6)	否	否	否	auto_now_add=True	题库的创建时间
creator	bigint	否	否	是	关联 users.User 表	题库的创建者
description	text	是	否	否	无	题库描述
question_count	int	否	否	否	default=0	题库中题目的数量

`QuestionBank` 模型通过合理的字段设计，为系统题库管理提供了全面支持。`id` 是题库的唯一标识符，`name` 字段存储题库名称，`subject` 字段通过预定义选项限制题库所属科目，确保分类清晰。`estimated_time` 字段记录预计完成题库所需的时间，`created_at` 字段存储题库的创建时间，`creator` 字段关联创建者用户，为题库提供来源记录。`description` 字段支持存储题库的附加说明，而 `question_count` 字段记录题库中的题目数量，便于系统统计和展示。

通过这些字段的组合，`QuestionBank` 模型不仅能够满足基本的题库记录需求，还为题库的分类、管理和分析提供了强大的数据支持，确保系统的高效性和易用性。

2.9.2 题目

`Question` 模型用于存储考试系统中的题目信息，是题库管理的核心数据结构。模型支持多种题型，包括单选、多选、判断、填空和解答题，并通过字段记录题目的内容、难度、所属科目、来源、添加者以及与题库的关联关系。模型设计了多种功能方法，用于判断题目类型、获取用户答题状态，以及动态设置选项数量，极大地提高了题库管理的灵活性和智能化。

```
class Question(models.Model):
    TYPE_CHOICES = [
        ("单项选择题", "单项选择题"),
        ("多项选择题", "多项选择题"),
        ("判断题", "判断题"),
        ("填空题", "填空题"),
        ("解答题", "解答题"),
    ]

    DIFFICULTY_CHOICES = [
        ("简单", "简单"),
        ("中等", "中等"),
        ("困难", "困难"),
    ]

    SUBJECT_CHOICES = [
        ("工科数学分析（上）", "工科数学分析（上）"),
        ("工科数学分析（下）", "工科数学分析（下）"),
        ("工科高等代数", "工科高等代数"),
        ("离散数学（信息类）", "离散数学（信息类）"),
        ("基础物理学A", "基础物理学A"),
    ]

    id = models.AutoField(primary_key=True)  # 题目ID
    type = models.CharField(max_length=20, choices=TYPE_CHOICES)  # 题目类型
    content = models.TextField()  # 题目内容（Markdown 格式）
    subject = models.CharField(max_length=100, choices=SUBJECT_CHOICES)  # 所属科
目，限制为可选科目
    added_at = models.DateField()  # 添加时间
    source = models.CharField(max_length=100, blank=True, null=True)  # 题目来源
    tags = models.JSONField()  # JSON 格式

    difficulty = models.CharField(max_length=10, choices=DIFFICULTY_CHOICES)  #
难度
    answer = models.TextField(blank=True, null=True)  # 答案内容
    added_by = models.ForeignKey(User, on_delete=models.SET_NULL, null=True,
related_name="added_questions")  # 创建者
    question_banks = models.ManyToManyField(QuestionBank,
related_name="questions")  # 题库关系
    option_count = models.IntegerField(default=0)  # 选项数量，默认 0

    def is_single_choice(self):
        return self.type == "单项选择题"

    def is_multiple_choice(self):
        return self.type == "多项选择题"

    def is_true_false(self):
```

```
        return self.type == "判断题"

    def get_user_status(self, user):
        """
        获取用户对该题目的做题状态：
        - 未做过
        - 已做对
        - 做错
        """
        try:
            record = self.user_records.get(user=user)
            if record.is_correct is None:
                return "未做过"
            return "已做对" if record.is_correct else "做错"
        except UserQuestionRecord.DoesNotExist:
            return "未做过"

    def __str__(self):
        return f"{self.type} - {self.subject} - {self.id}"

    def save(self, *args, **kwargs):
        """
        根据题目类型设置默认的选项数量。
        """
        if self.type in ["单项选择题", "多项选择题"] and self.option_count == 0:
            self.option_count = 4  # 默认设置为 4 个选项
        elif self.type not in ["单项选择题", "多项选择题"]:
            self.option_count = 0  # 其他类型默认设置为 0
        super().save(*args, **kwargs)
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	题目的唯一标识
type	varchar(20)	否	否	否	无	题目类型，如单选、多选等
content	text	否	否	否	无	题目内容
subject	varchar(100)	否	否	否	无	题目所属科目
added_at	datetime(6)	否	否	否	无	题目添加时间
source	varchar(100)	是	否	否	无	题目来源信息

字段名称	数据类型	可否为空	主码	外码	其他	说明
tags	text	否	否	否	无	题目标签，JSON 格式存储
difficulty	varchar(10)	否	否	否	无	题目难度
answer	text	是	否	否	无	答案内容
added_by	bigint	是	否	是	关联 users.User 表	题目创建者
question_banks	bigint	否	否	是	关联 question_banks 表	题目所属题库
option_count	int	否	否	否	default=0	题目的选项数量，默认 0

`Question` 模型通过精细的字段设计和灵活的功能方法，全面支持题库管理系统中的题目数据存储和逻辑处理。每道题通过 `id` 唯一标识，`type` 字段标明题目类型，`content` 字段记录题目的具体内容，支持 Markdown 格式以丰富题目展示效果。`subject` 字段对题目进行科目分类，`difficulty` 字段标明题目难度，`source` 和 `tags` 提供题目来源和标签信息，便于分类管理。`added_by` 字段关联创建者用户，`added_at` 记录题目添加时间，为题库管理和数据统计提供支持。`question_banks` 字段通过多对多关系记录题目所属的题库，方便题目与题库的灵活关联。

此外，`option_count` 字段专门为选择题设置默认选项数量，而模型中的方法如 `is_single_choice`、`is_multiple_choice` 等则用来快速判断题目类型。`get_user_status` 方法实现了动态查询用户对题目的答题状态，为系统交互功能提供了便利。`save` 方法通过题目类型智能设置选项数量，进一步提升了模型的实用性和自动化程度。综合来看，`Question` 模型结构清晰、功能强大，是考试系统题库模块的核心支撑。

2.9.3 用户与题目之间的做题记录

`UserQuestionRecord` 模型用于记录用户对题目的作答情况，是用户做题统计与分析的核心数据结构。该模型通过外键关联用户和题目，记录了用户对每道题的答题结果、答题时间以及题目的科目信息。为了确保数据唯一性，模型通过 `unique_together` 限制同一用户对每道题目只能生成一条记录。模型设计简洁高效，为系统提供了全面的用户答题记录管理支持。

```
class UserQuestionRecord(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="question_records")
    question = models.ForeignKey(Question, on_delete=models.CASCADE,
related_name="user_records")
    question_subject = models.CharField(max_length=100,
choices=Question.SUBJECT_CHOICES,
                                        default="工科数学分析（上）")  # 题目科目
    is_correct = models.BooleanField(null=True, blank=True)  # 是否做对
```

```
    attempted_at = models.DateTimeField(auto_now_add=True)  # 做题时间

    class Meta:
        unique_together = ('user', 'question')  # 确保同一个用户对每个题目只有一条记录
        verbose_name = "用户做题记录"
        verbose_name_plural = "用户做题记录"

    def __str__(self):
        return f"User {self.user.id} - Question {self.question.id} - {'Correct'
if self.is_correct else 'Incorrect'}"
```

字段名称	数据类型	可否为空	主码	外码	其他	说明
id	bigint	否	是	否	无	用户做题记录的唯一标识
user	bigint	否	否	是	关联 users.User 表	答题用户
question	bigint	否	否	是	关联 questions 表	作答的题目
question_subject	varchar(100)	否	否	否	无	题目所属科目
is_correct	int	是	否	否	无	答题是否正确
attempted_at	datetime(6)	否	否	否	无	用户作答时间

`UserQuestionRecord` 模型设计紧凑而功能全面，能够记录用户对题目的作答信息和答题状态。通过 `user` 和 `question` 外键，模型实现了用户与题目之间的精确关联，同时利用 `unique_together` 限制每位用户对每道题目只能记录一次答题结果，确保了数据的一致性。`question_subject` 字段存储题目的科目信息，`is_correct` 字段记录答题是否正确，`attempted_at` 字段标记用户的答题时间，这些字段为答题记录的管理和分析提供了必要支持。

通过该模型，系统可以方便地统计用户答题情况，例如答题正确率、答题时间分布等，同时也为个性化练习推荐和考试评估提供了数据基础。`UserQuestionRecord` 模型结构清晰、功能实用，是系统中用户做题记录管理的关键模块。

三、系统重要功能实现方法

3.1 鉴权

为了鉴别用户身份，我们在用户浏览器本地，使用`localStorage`与`vuex`存储了服务器返回的用户id。同时设置axios的默认请求头，以便在所有后续的HTTP请求中自动包含授权信息。

```
axios
  .post("http://127.0.0.1:8000/api/myapp2/login/", loginData)
  .then((response) => {
    const token = response.data.studentNumber.value;
    const user = response.data;
    // 使用localStorage与vuex，缓存token与用户信息
    this.setUserId(token);
    this.setUserInfo(user);
    axios.defaults.headers.common["Authorization"] = `Bearer ${token}`;
  })
  .catch((error) => {
    // 处理异常情况
  });
```

`response.data`字段中同时存储了用户的权限信息（角色为学生/辅导师），通过该信息调整对应的网页内容，以下是一个例子：

```
menuItems() {
  const isLoginRoute = this.isLoginRoute;
  // this.currentRole 由response.data段中相关信息得来
  if (this.currentRole === '辅导师') {
    // 如果是辅导师，返回特殊的菜单项
    return [
      { title: "首页", icon: "mdi-home", value: "/admin/home" },
      { title: "题目管理", icon: "mdi-file-edit", value: "/admin/exercise" },
      { title: "题库管理", icon: "mdi-database-edit", value: "/admin/problemset"
},
      { title: "模拟测试管理", icon: "mdi-arrange-send-backward", value:
"/admin/exam" },
      { title: "模拟测试评分", icon: "mdi-chart-arc", value: "/admin/judge" },
      { title: "讨论区", icon: "mdi-forum", value: "/admin/forum" },
      { title: "个人中心", icon: "mdi-account-details", value: "/admin/profile" },
      isLoginRoute
        ? { title: "登录", icon: "mdi-login", value: "/login" }
        : { title: "注销", icon: "mdi-logout", value: "/logout" },
    ];
  } else {
    // 默认菜单项为学生
    return [
      { title: "首页", icon: "mdi-home", value: "/home" },
      { title: "题库", icon: "mdi-database", value: "/problemset" },
      { title: "模拟测试", icon: "mdi-file-sign", value: "/exam" },
      { title: "讨论区", icon: "mdi-forum", value: "/forum" },
      { title: "个人中心", icon: "mdi-account-details", value: "/profile" },
      isLoginRoute
        ? { title: "登录", icon: "mdi-login", value: "/login" }
        : { title: "注销", icon: "mdi-logout", value: "/logout" },
    ];
  }
},
```

为了防止用户未登录直接通过访问个人中心等页面、学生用户访问辅导师页面等，我们设置了网页的全局路由守卫。具体而言，我们对每个路由标记了两项元数据：

- `requiresAuth`：该页面需要登录后才可访问
- `requiresAdmin`：该页面需要辅导师权限才可访问

对每次导航操作，都进行鉴权管理：

```
router.beforeEach((to, from, next) => {
  const isAuthenticated = store.getters.isAuthenticated;
  // 获取用户角色
  const userRole = store.getters.userRole ? store.getters.userRole : -1;

  if (to.meta.requiresAuth && !isAuthenticated) {
    // 如果目标路由需要认证但用户未认证，重定向到登录页
    next({ path: '/login', query: { redirect: to.fullPath } });
    store.dispatch("snackbar/showSnackbar", {
      message: '你必须先登录，才能访问本系统内容',
      color: 'warning',
      timeout: 2000
    });
  } else if (to.meta.requiresGuest && isAuthenticated) {
    // 如果目标路由仅限未认证用户且用户已认证，重定向到首页
    next({ path: '/home' });
  } else if (to.meta.requiresAdmin && userRole <= 0) {
    // 如果目标路由需要管理员权限但用户角色不足，重定向到未授权页面
    store.dispatch("snackbar/showSnackbar", {
      message: '你的权限无法访问此内容',
      color: 'error',
      timeout: 2000
    });
    next({ path: '/unauthorized' });
  } else {
    // 否则，允许导航
    next();
  }
});
```

除了采取前端鉴权的方法，对于用户进行的每次数据库操作，我们同时在后端进行了鉴权操作：

3.1.1 用户登录

用户登录是鉴权的第一步，用于验证用户身份。

流程：

- **获取用户凭据**：用户提交用户名和密码。
- **验证凭据**：后端检查用户名和密码是否正确。
- **生成 Token**：成功验证后，生成一个唯一的 Token（如 JWT）并返回给用户。

```
class UserLogin(APIView):
    def post(self, request):
        # print(request.data)
        # get user
        try:
            # if no invalid id, User.objects.get will raise exception
            user = User.objects.get(student_id=request.data['student_id'])
```

```
        except Exception as _e:
            return Response(
                response_json(
                    success=False,
                    code=UserErrorCode.USER_NOT_FOUND,
                    message='user not found!'
                )
            )
        # check password
        try:
            password_digest = encode_password(request.data['password'])
            if password_digest != user.password_digest:
                raise Exception()
            jwt_token = generate_jwt(user.id)
            print(jwt_token)
            # print(jwt_token)
        except Exception as _e:
            return Response(response_json(
                success=False,
                code=UserErrorCode.INCORRECT_PASSWORD,
                message='incorrect password!'
            ))
        # login success
        return Response(
            response_json(
                success=True,
                message='login success!',
                data={
                    'jwt': jwt_token,
                    'role': user.user_role
                }
            )
        )
```

3.1.2 保护 API 路由

使用生成的 `Token` 对用户进行权限校验，以确保用户只能访问被允许的资源。

流程：

- 客户端在每次请求时，将 `Token` 通过 `Authorization` Header 发送到后端。
- 后端解析并验证 Token，确保其有效性。
- 检查用户的权限，确保其对资源具有访问权限。

在 Django REST Framework 中：

- **设置全局认证类**： 在 `settings.py` 中配置：

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    )
}
```

- **保护特定 API 路由**

```
from rest_framework.views import APIView
from rest_framework.permissions import IsAuthenticated

class ProtectedView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        return Response({"message": "You are authenticated"})
```

3.2 Markdown解析

我们在前端使用了`v-md-editor`库对markdown文本进行解析。

我们分别导入了预览器`VMdPreview`和编辑器`VueMarkdownEditor`，在只需要预览而不需要编辑的地方只使用预览器可有效优化性能。考虑平台的具体需求，预览器和编辑器都支持代码高亮、代码行数统计、latex格式数学公式显示等功能。

采用markdown格式对文本进行存储，有利于后端统一文本存储格式，方便组织管理。

```
import VueMarkdownEditor from '@kangc/v-md-editor';
import '@kangc/v-md-editor/lib/style/base-editor.css';

import VMdPreview from '@kangc/v-md-editor/lib/preview';
import '@kangc/v-md-editor/lib/style/preview.css';
import githubTheme from '@kangc/v-md-editor/lib/theme/github.js';
import '@kangc/v-md-editor/lib/theme/style/github.css';

// highlightjs
import hljs from 'highlight.js';
import createKatexPlugin from '@kangc/v-md-editor/lib/plugins/katex/cdn';
import createLineNumbertPlugin from '@kangc/v-md-editor/lib/plugins/line-
number/index';

VMdPreview.use(githubTheme, {
  Hljs: hljs,
});
VMdPreview.use(createKatexPlugin());
VMdPreview.use(createLineNumbertPlugin());

import Prism from 'prismjs';

VueMarkdownEditor.use(githubTheme, {
  Prism,
```

```
  Hljs: hljs,
});
VueMarkdownEditor.use(createKatexPlugin());
VueMarkdownEditor.use(createLineNumbertPlugin());
```

3.3 图床

在题目显示、学生提交答案、学生发布讨论贴等过程，都需要上传、下载和显示图片。使用图床可以更方便地对图片文件进行管理。

我们采用腾讯云cos服务搭建了图床，在后端通过调用`upload`函数将对应路径的图片上传到图库。

```
import logging
import sys
import time
from datetime import datetime

from qcloud_cos import CosConfig
from qcloud_cos import CosS3Client

# 正常情况日志级别使用 INFO，需要定位时可以修改为 DEBUG，此时 SDK 会打印和服务端的通信信息
logging.basicConfig(level=logging.INFO, stream=sys.stdout)

secret_id = None
secret_key = None
region = None
bucket_name = None

config = CosConfig(Region=region, SecretId=secret_id, SecretKey=secret_key)
client = CosS3Client(config)

def upload_percentage(consumed_bytes, total_bytes):
    """进度条回调函数，计算当前上传的百分比

    :param consumed_bytes: 已经上传的数据量
    :param total_bytes: 总数据量
    """
    if total_bytes:
        rate = int(100 * (float(consumed_bytes) / float(total_bytes)))
        print('\r{0}% '.format(rate))
        sys.stdout.flush()


def upload(path, title):
    """上传图片逻辑，返回上传好的图片对应的图床url

    :param path: 本地图片地址
    :param title: 图片名称
    """
    # 获取当前时间，以格式化图片名称
    current_time = datetime.now()
    formatted_time = current_time.strftime("%Y_%m_%d_%H_%M_%S")
    print(formatted_time)
    key = "db_imgs/" + formatted_time + "_" + title
    response = client.upload_file(
```

```
        Bucket=bucket_name,
        Key=key,
        LocalFilePath=path,
        PartSize=1,
        MAXThread=5,
        progress_callback=upload_percentage,
        EnableMD5=False,
        ACL='public-read',
    )
    url = client.get_object_url(
        Bucket=bucket_name,
        Key=key
    )
    return url
```

在前端界面，使用异步函数`uploadFile`进行图片上传。

```
async uploadFile() {
  const fileInput = document.createElement("input");
  fileInput.type = "file";
  // 只允许选择图片文件
  fileInput.accept = ".jpg, .jpeg, .png";
  // 文件选择后执行的回调
  fileInput.onchange = (e) => {
    const file = e.target.files[0]; // 获取用户选择的文件
    if (file) {
      const reader = new FileReader();
      const allowedExtensions = /(\.jpg|\.jpeg|\.png)$/i;
      if (!allowedExtensions.exec(file.name)) {
        // 异常处理
        return;
      }
      reader.onload = async (event) => {
        const formData = new FormData();
        formData.append("files", file);
        try {
          const response = await fetch("http://127.0.0.1:8000/api/images/upload-
image/", {
            method: "POST",
            body: formData
          });
          const result = await response.json();
          if (response.ok) {
            // 上传成功
          } else {
            // 上传失败
          }
        } catch (error) {
          // 上传失败
        }
      };
      reader.readAsDataURL(file);
    }
  };
  // 点击输入框，选择文件
```

```
  fileInput.click();
}
```

在前端需要显示图片的时候，后端只需要像前端传递图片的地址，前端通过访问该地址就可显示对应的图片。相比于后端向前端直接传递图片文件，传输图片的链接能提高网页的加载速度。

3.4 动态路由匹配

我们经常需要把某种模式匹配到的所有路由全部映射到同一个组件。例如在`题库详情`页面中，对于每个拥有不用的id，都需要利用`ProblemSetDetail.vue`组件进行渲染。那么可以在路由路径中使用“动态路径参数”（dynamic segment）对id进行匹配：

```
{
    path: "/problemset/:id",
    name: "ProblemSetDetail",
    component: () => import("@/components/user/ProblemSetDetail.vue"),
    props: true,
    meta: {
      appTitle: "题库详情",
      pageTitle: "题库详情",
      requiresAuth: true,
    },
},
```

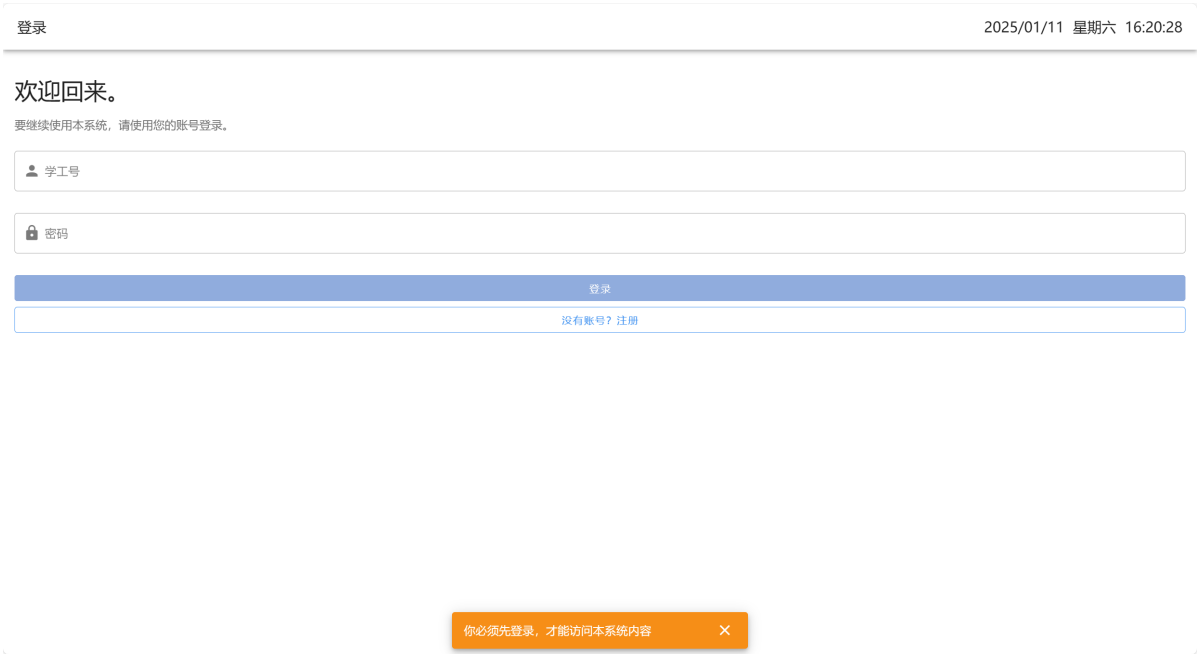
访问该页面时，可以从路由中取出id，然后调用函数加载题库信息。

```
mounted() {
    const id = this.$route.params.id;
    // 获取题库数据
    this.fetchProblemSetData(id);
},
```

四、系统实现效果

4.1 登录注册

默认页面



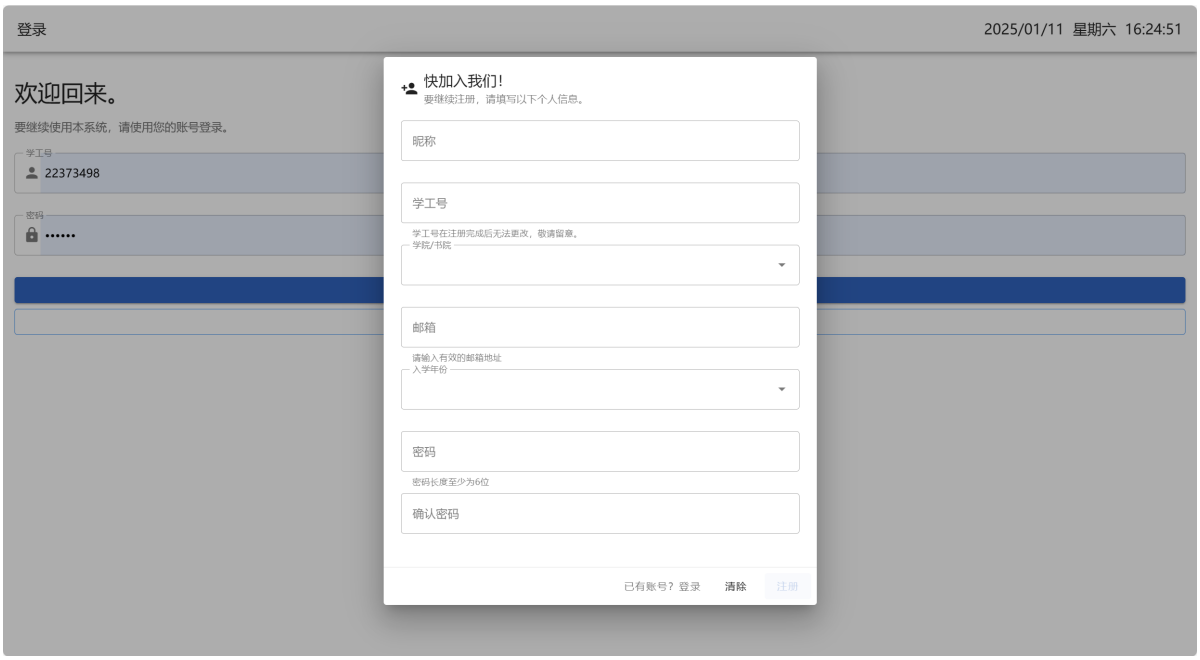
信息

- 弹窗：提示用户进行登录
- 引导语
- 账号（学工号）输入框，密码输入框

功能

- 登录：输入账号密码后，点击登录按钮；后端检验返回是否是正确秘密啊
  - 如果是正确密码，进入主页面
  - 否则弹窗显示错误密码
- 注册：弹出注册对话框

注册页面



信息

- 各种prompt

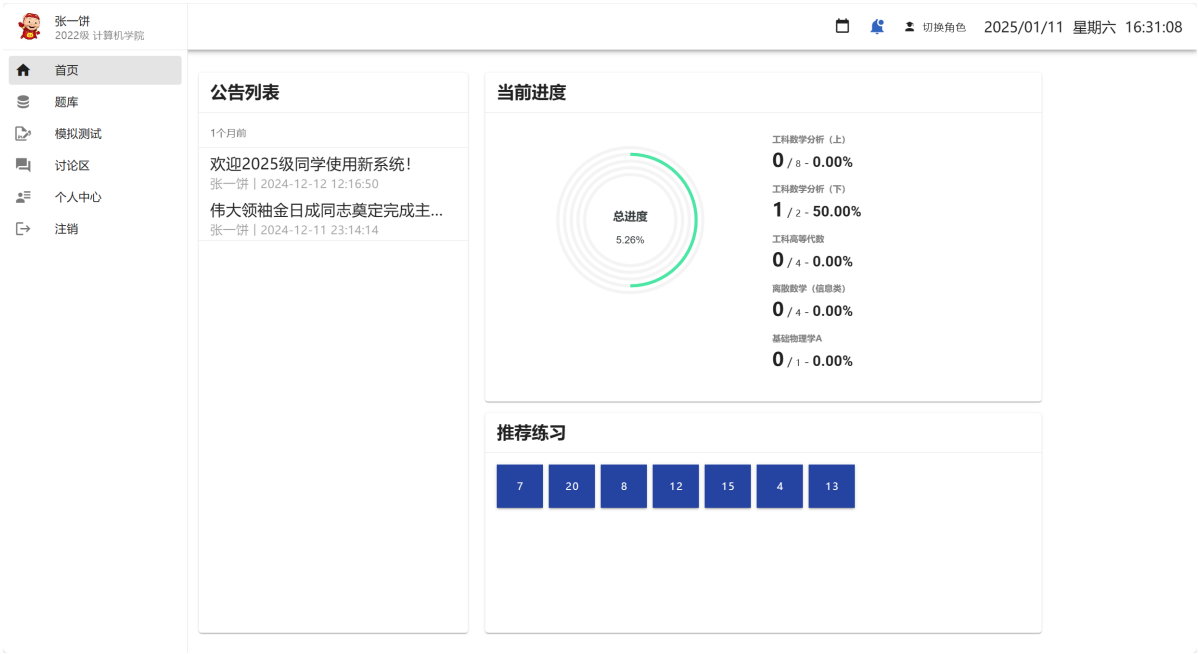
- 操作按钮：返回、清除、注册

功能

- 点击“已有帐号？登录”按钮，或点击对话框外部，将收起注册对话框
- 每个输入框输入完成后会校对该部分信息的完整性
- 点击“清除”后可清除所有的信息和报错内容
- 点击“注册”后将校对学工号是否重复，如果不重复则返回登录页面，并将学工号和密码预先填充好

4.2 学生视角

主页面

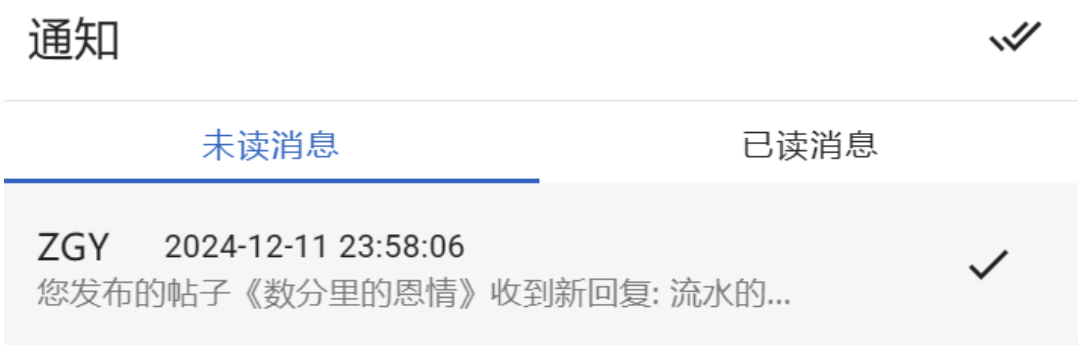


信息

- 公告列表：按照时间顺序排列公告概略，点击可查看内容
- 当前进度：显示用户所作题目占总体库的比例
- 推荐练习：根据用户所作题目的情况，推荐一些题目练习，点击可查看题目信息

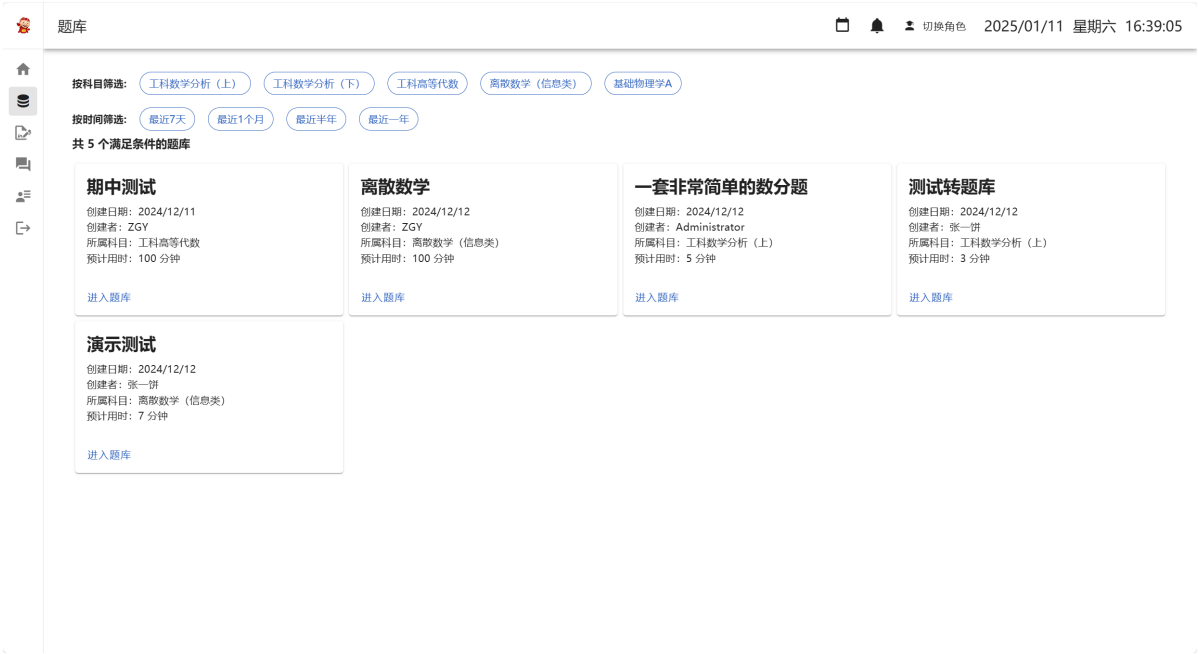
功能

- 侧边栏：导航访问题库、模拟测试等页面
- 点击“注销”可退出登录
- 上方三个按钮的功能分别为
    - 测试日程：显示用户已报名的测试，和正在进行的测试
    - 通知：查看由系统发出的通知，将通知标记为已读/未读



- 切换角色：如果用户拥有辅导师权限，可切换自己的视图

题库



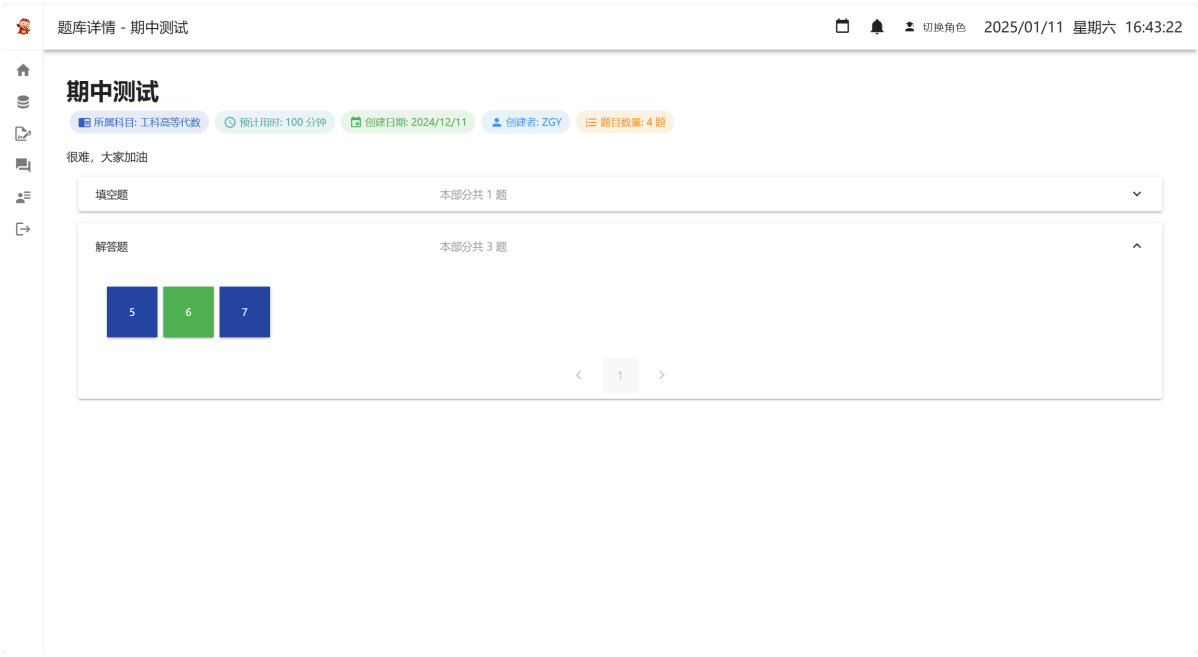
信息

- 题库列表：每个题库包括创建日期、创建者、所属题目、预计用时

功能

- 根据科目和创建日期对题库进行筛选
- 点击卡片可查看题库介绍
- 点击“进入题库”可查看题库详情

题库详情



信息

- 所属科目、预计用时、创建日期、创建者、题目数量
- 题库介绍

功能

- 点击题型标题可查看该题型的所有题目，点击题目可查看题目详情
- 未做过题目/做错题目显示为蓝色，做对题目显示为绿色

题目

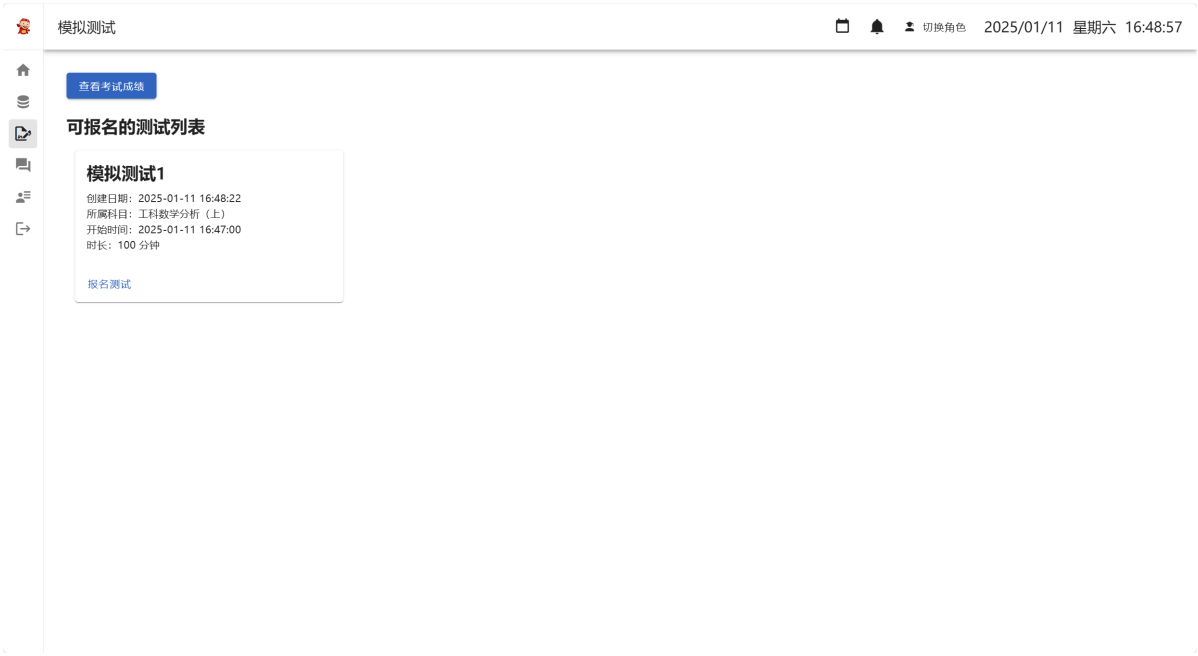
信息

- 题面
- 题目信息：学科、添加事件、来源、标签、难度

功能

- 点击“查看答案”按钮查看答案和解析，并标记自己是否做对了这道题

模拟测试

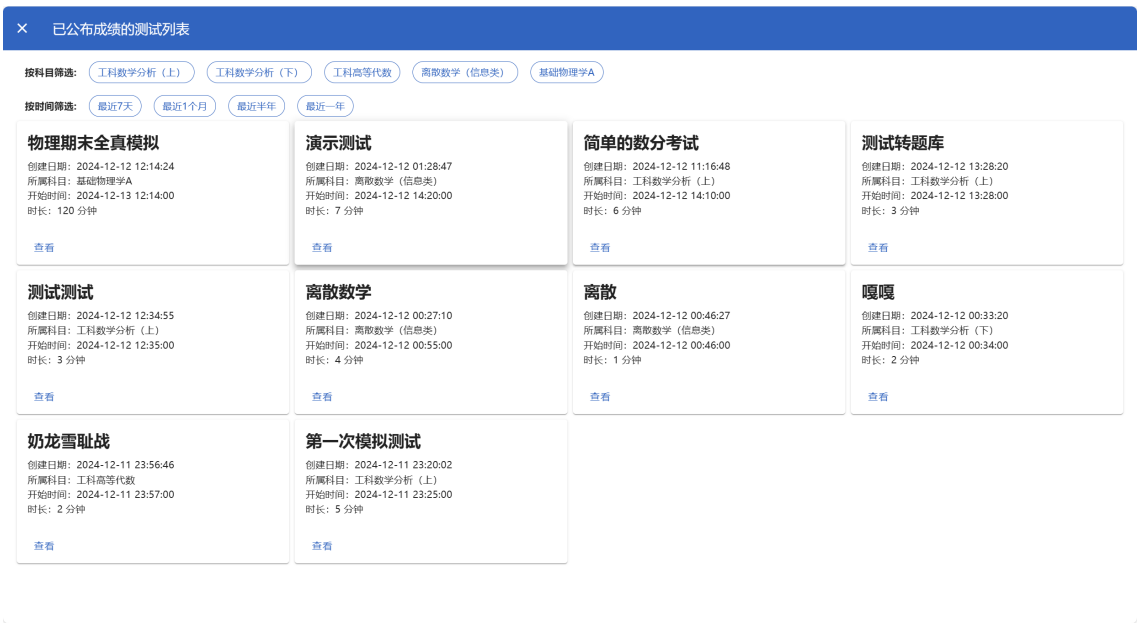


信息

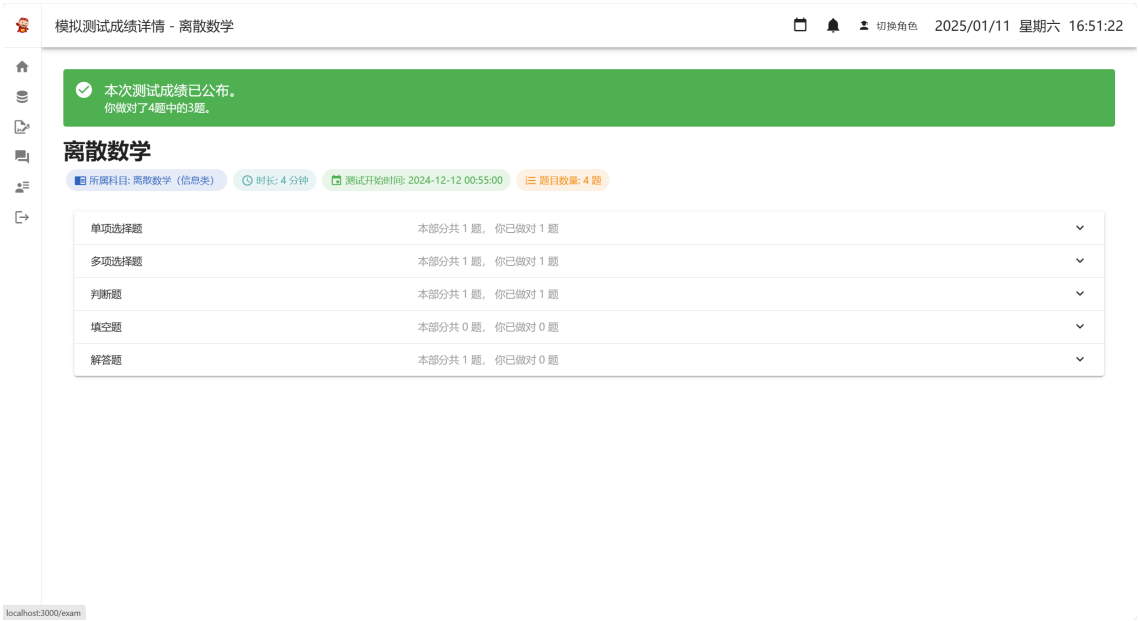
- 模拟测试列表：每个模拟测试包括创建日期、创建者、所属题目、用时

功能

- 点击报名测试后，如果正在进行，可直接进入测试
- 点击查看考试成绩，可查看之前已完成的测试的成绩

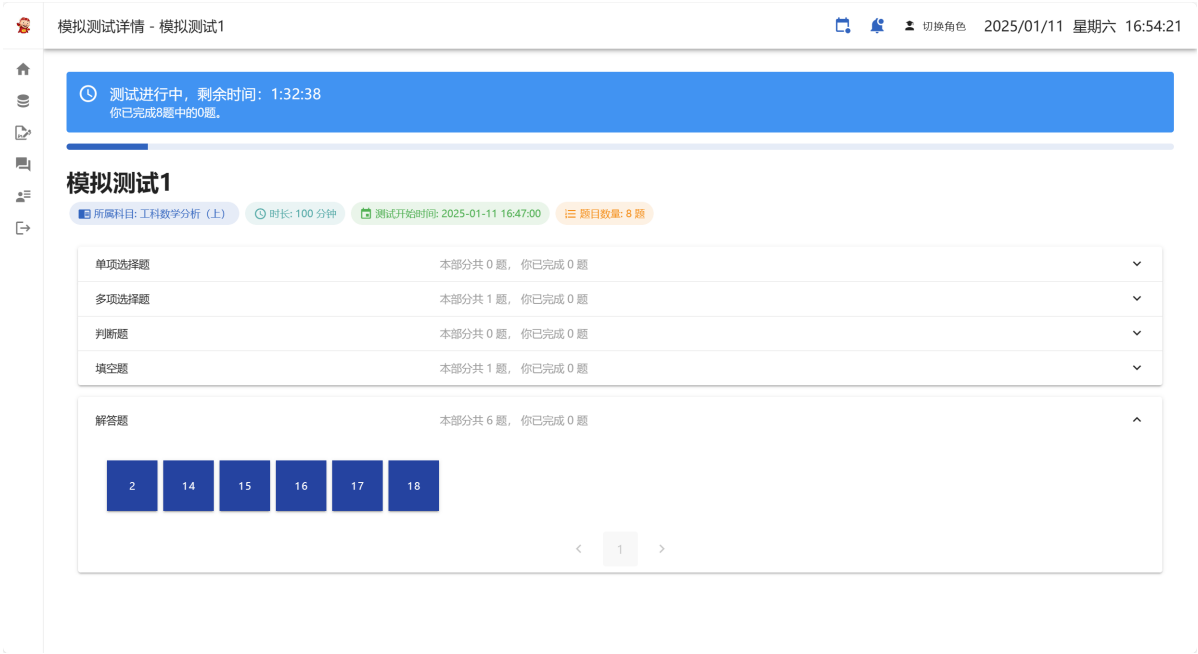


点击查看具体测试信息



点击每一道题，可查看标准答案和用户提交的答案

模拟测试详情



信息

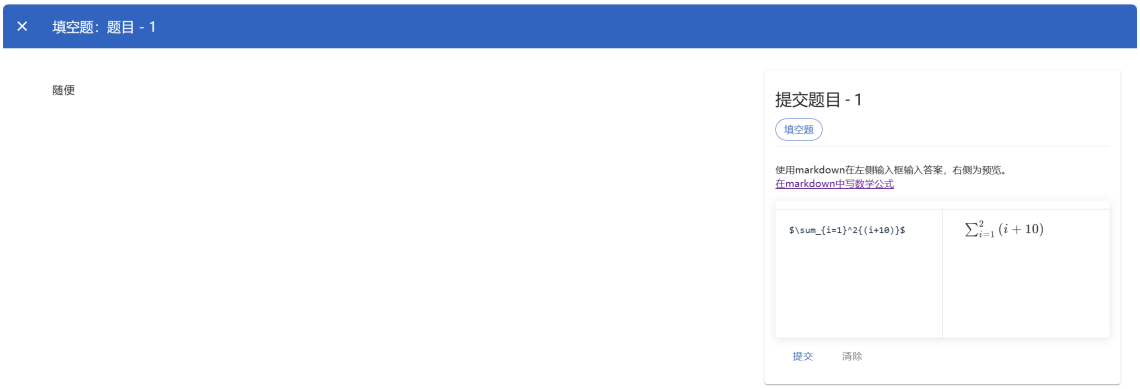
- 所属科目、考试时长、考试开始时间、题目数量
- 已做题目的数量、考试剩余时间
- 考试进度条

功能

- 点击题型标题可查看该题型的所有题目，点击题目可查看题目详情
- 未做过题目显示为蓝色，已做过题目显示为绿色
- 做题

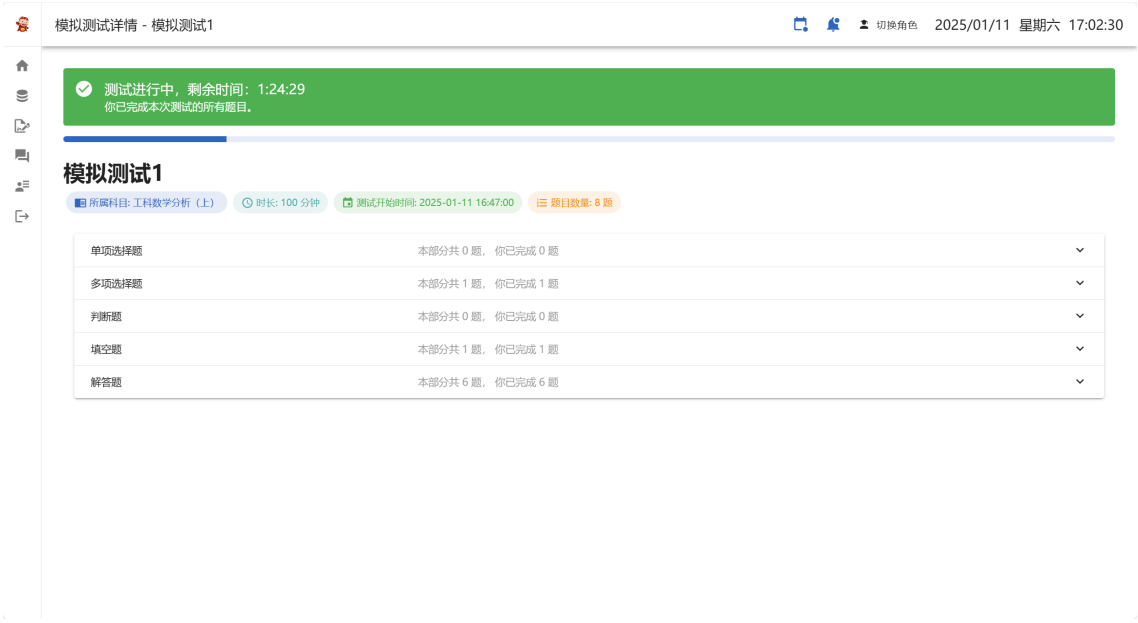
  解答题可上传答案文件；如果之前有提交，可下载上一次提交的文件

  填空题可使用markdown格式来写公式；自动加载上一次的提交

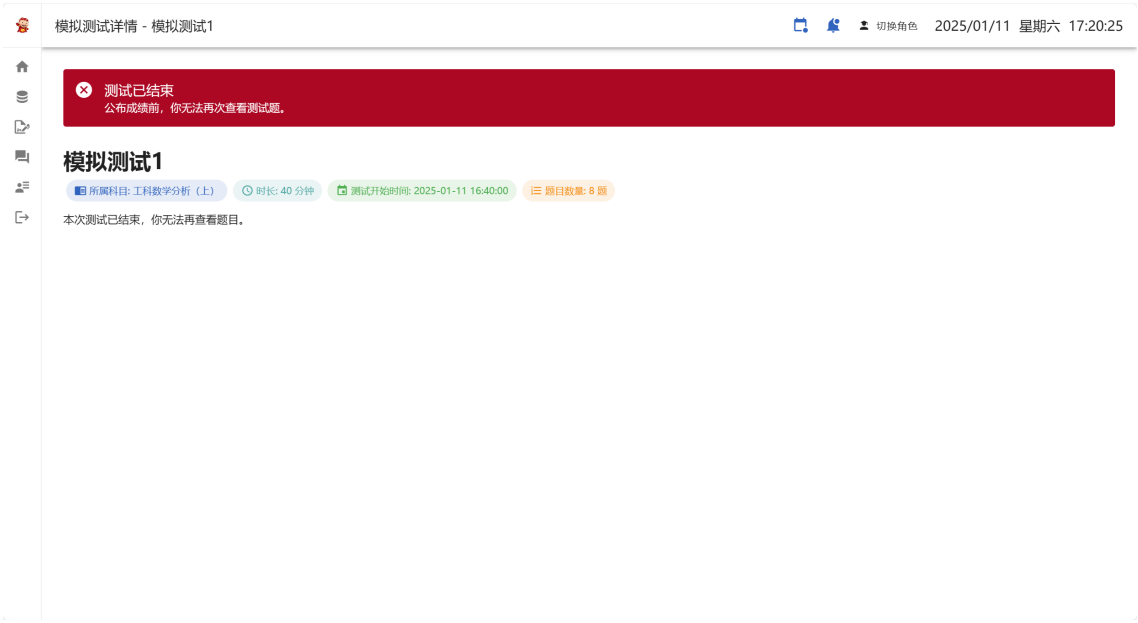


  其他的客观题可直接选择作答

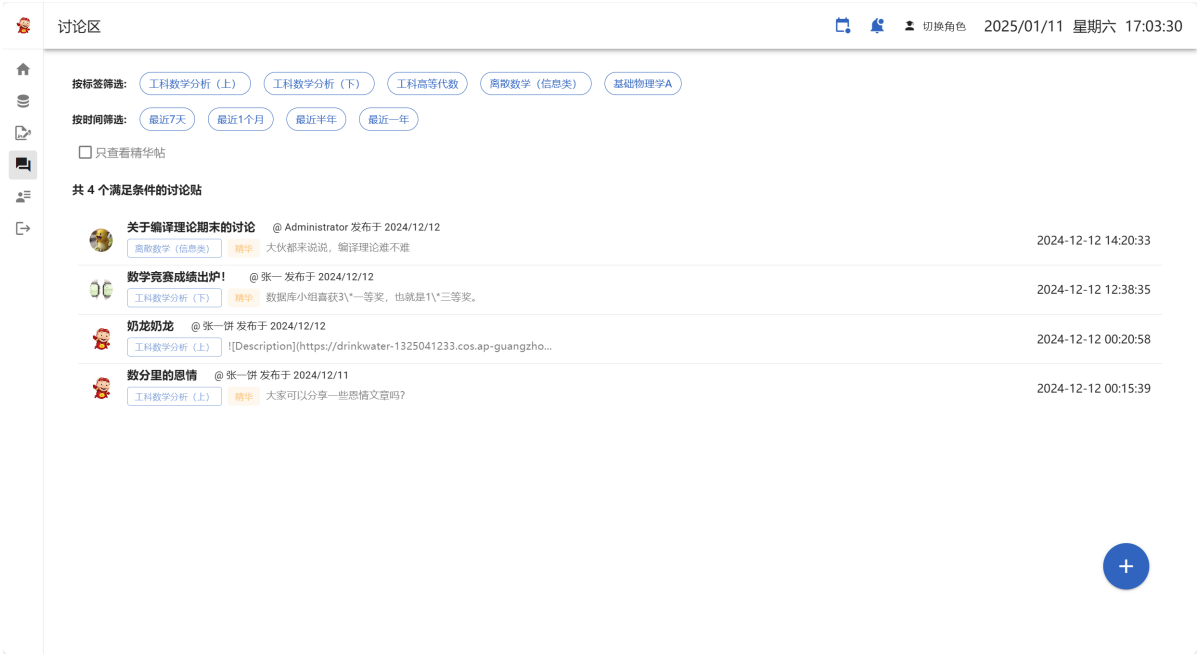
- 如果题目已全部做完，详情页面将显示“你已完成本次测试的所有题目”



如果考试结束，将显示“测试已结束：公布成绩前，你无法再次查看测试题。”



讨论区

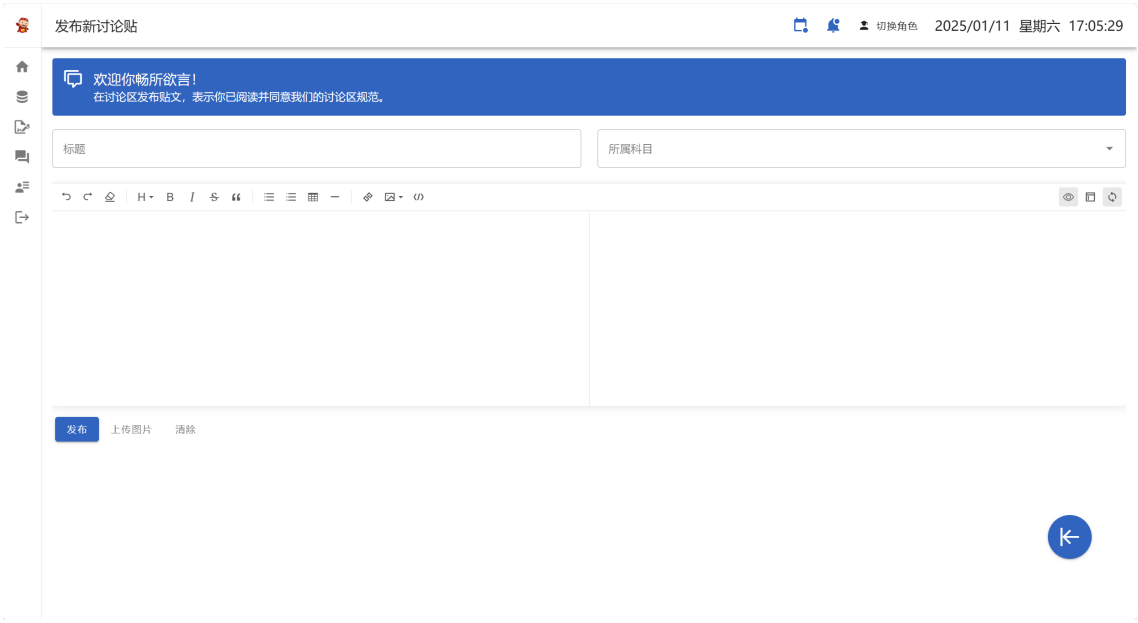


信息

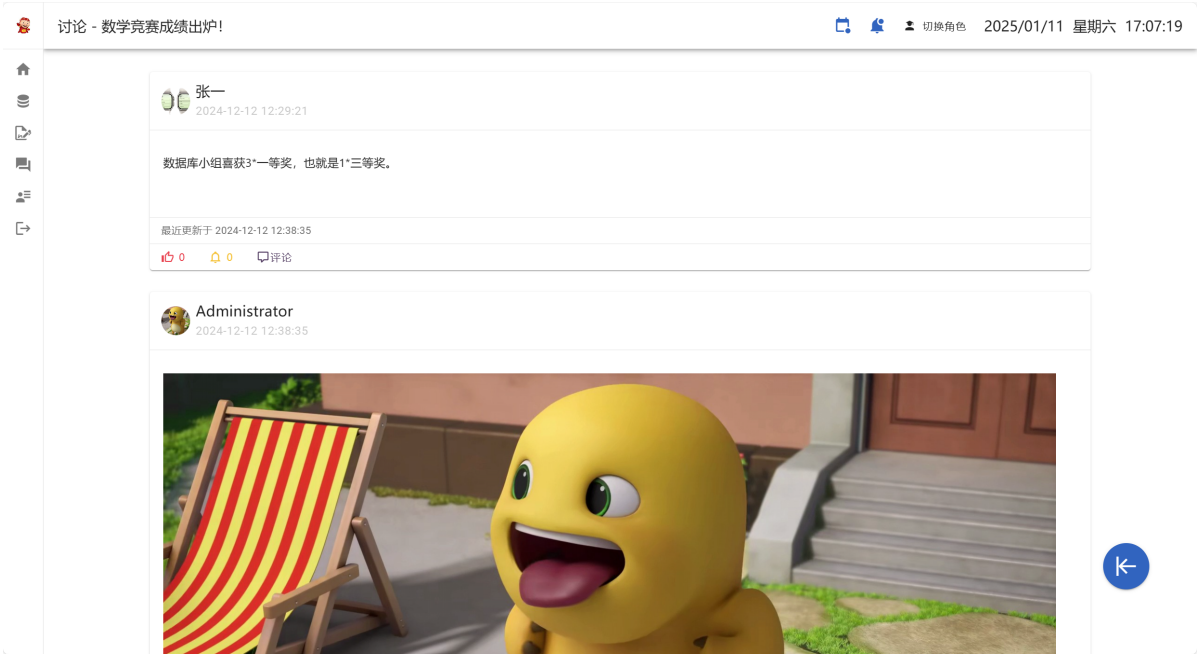
- 讨论贴列表：总数、标题、发布者、发布日期、标签、概览、上次编辑时间

功能

- 点击列表可查看讨论贴详情
- 根据科目和创建日期对讨论贴进行筛选
- 可筛选”只查看精华帖“
- 点击“+”可发布新的讨论贴



讨论帖详情



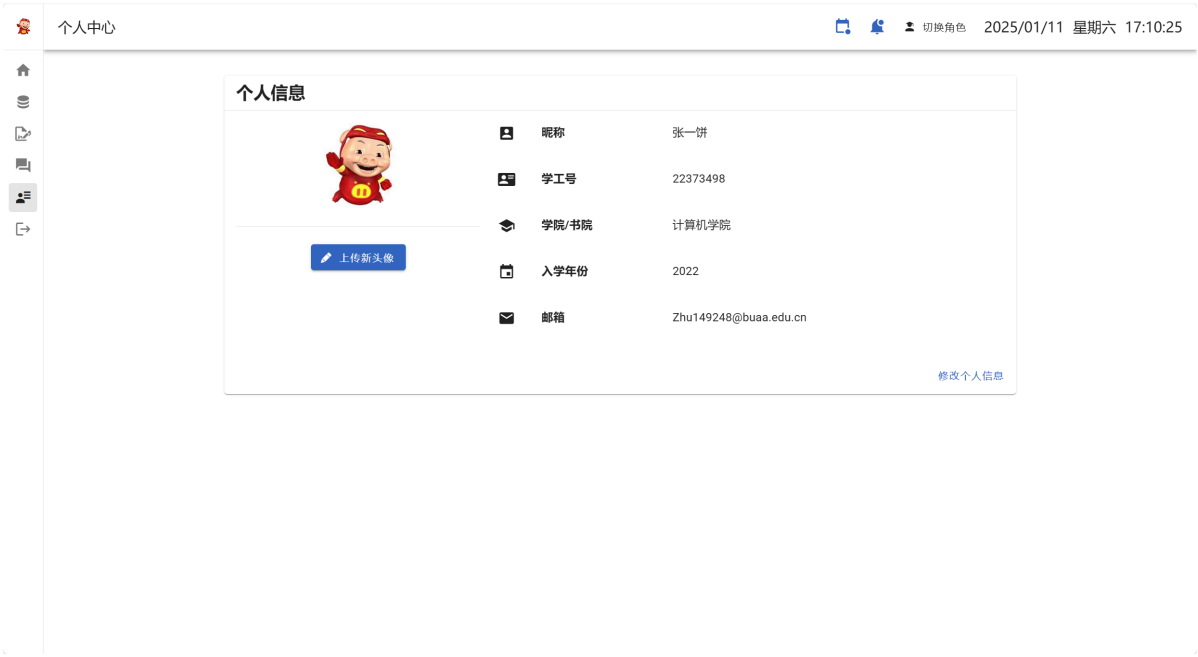
信息

- 讨论贴及回复贴的发布者、发布日期、标签、概览、上次编辑时间

功能

- 点赞、订阅、评论
- 可删除自己发布的讨论贴、回复贴；如果删除讨论贴将删除讨论贴的所有回复贴
- 点击右下角按钮退回到讨论区页面

个人中心



信息

- 昵称、学工号、学院、入学年份、邮箱
- 头像

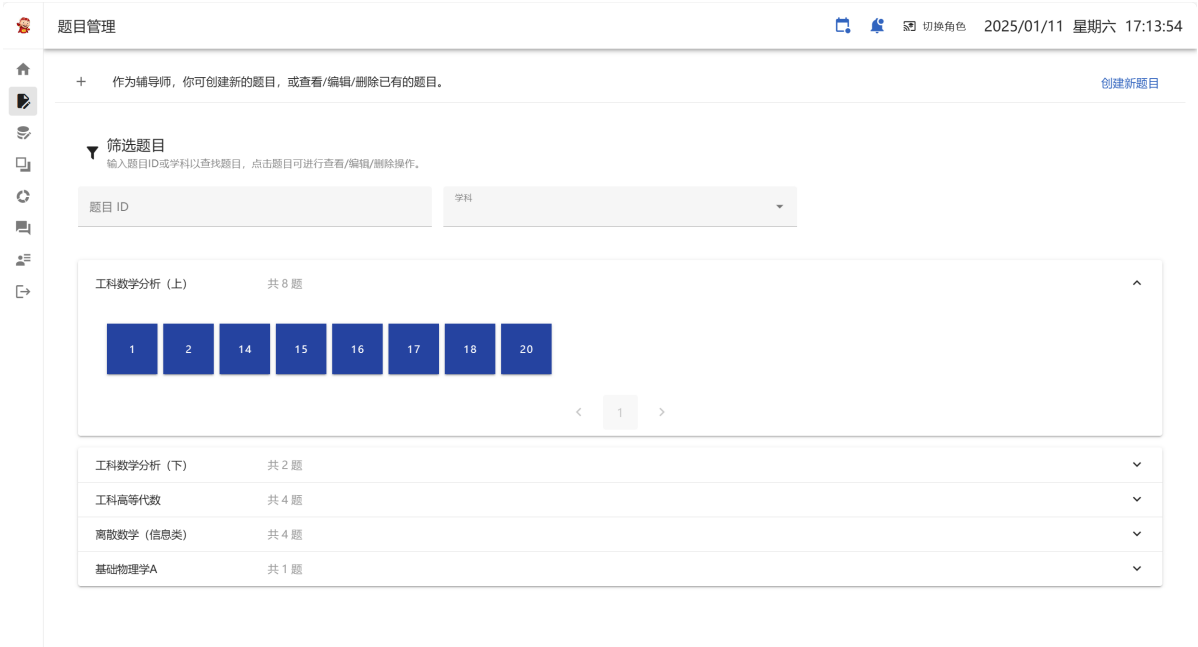
功能

- 上传新头像
- 修改个人信息

4.3 辅导师视角

以下介绍辅导师视角与学生视角存在较大区别的部分。

题目管理

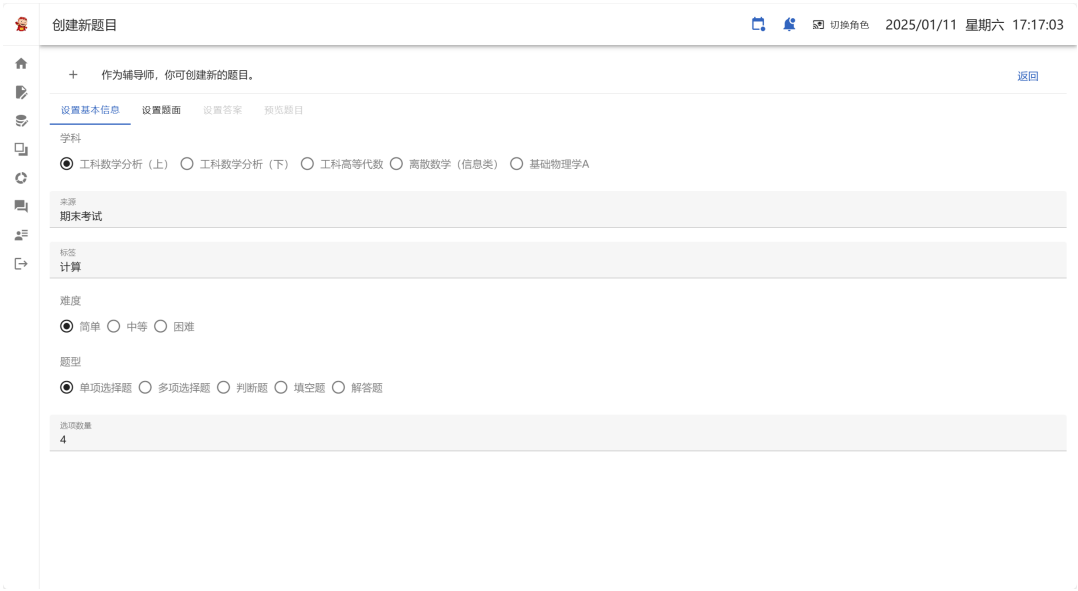


信息

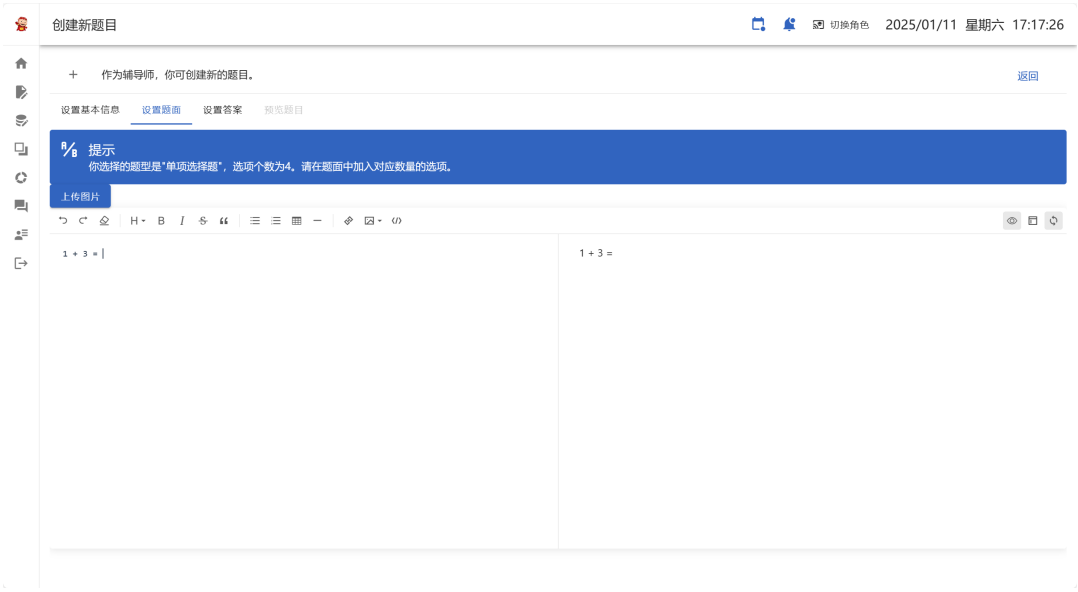
- 每个科目所有的题目
- 可按照题目id和学科对题目进行筛选

功能

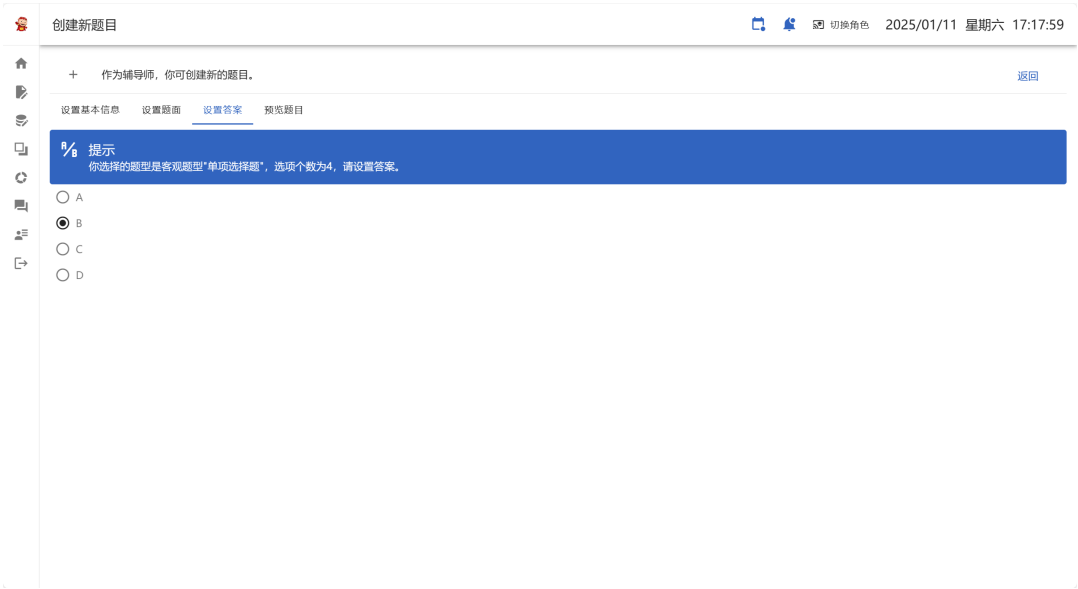
- 新建题目
  - 设置基本信息



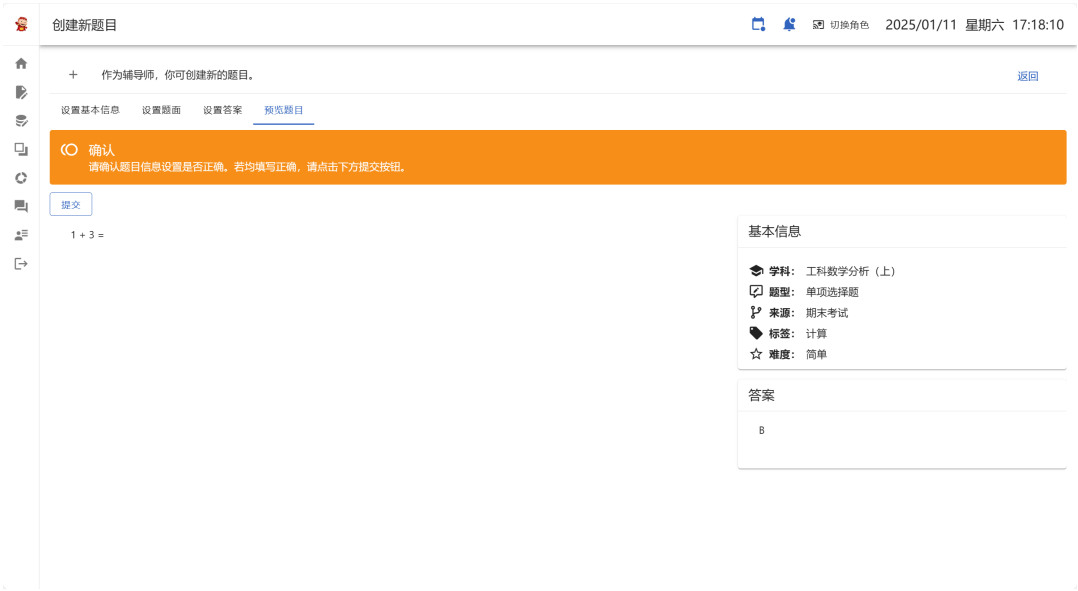
  - 设置题面



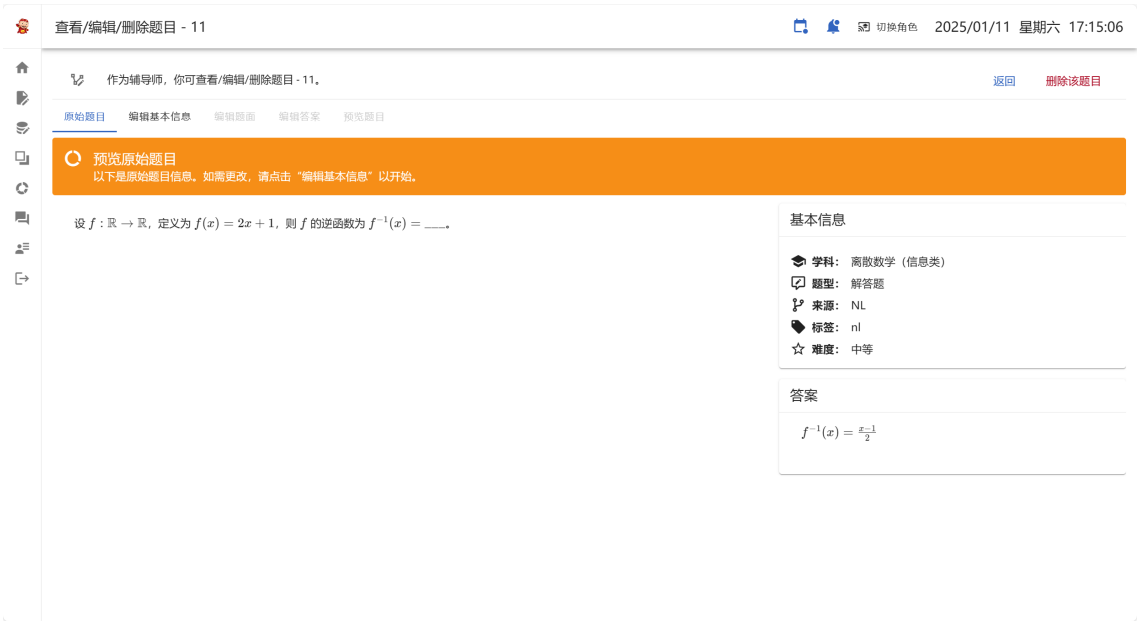
  - 设置答案



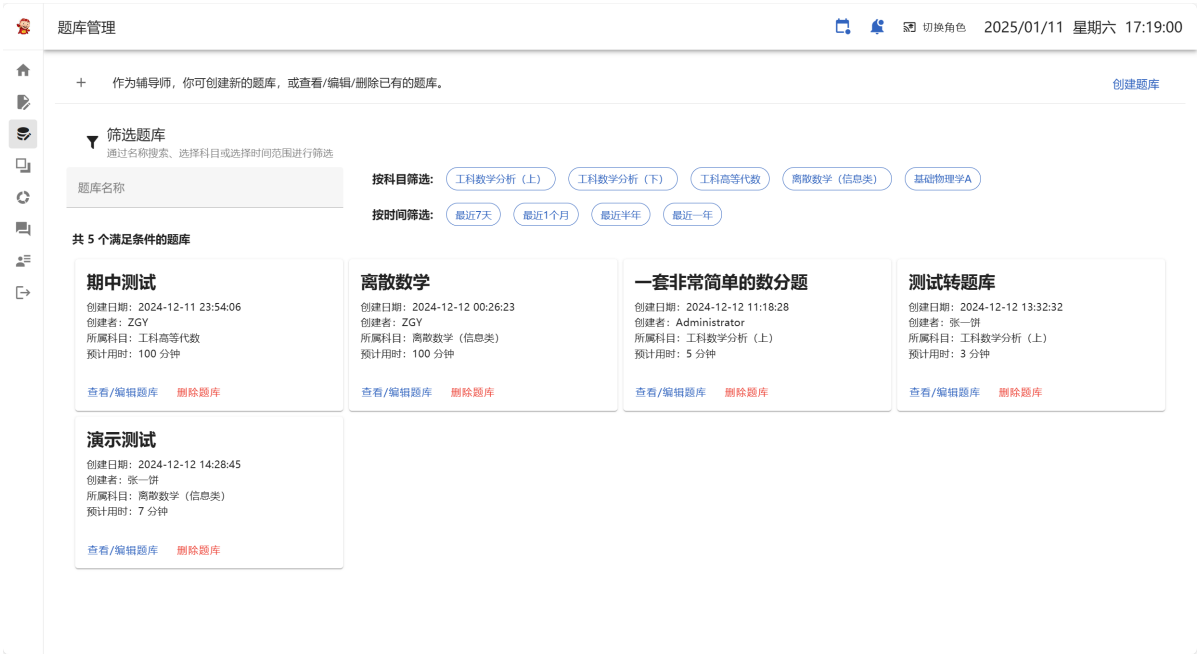
- 预览题目并提交



- 查看/编辑/删除题目：编辑页面与新建页面类似



题库管理



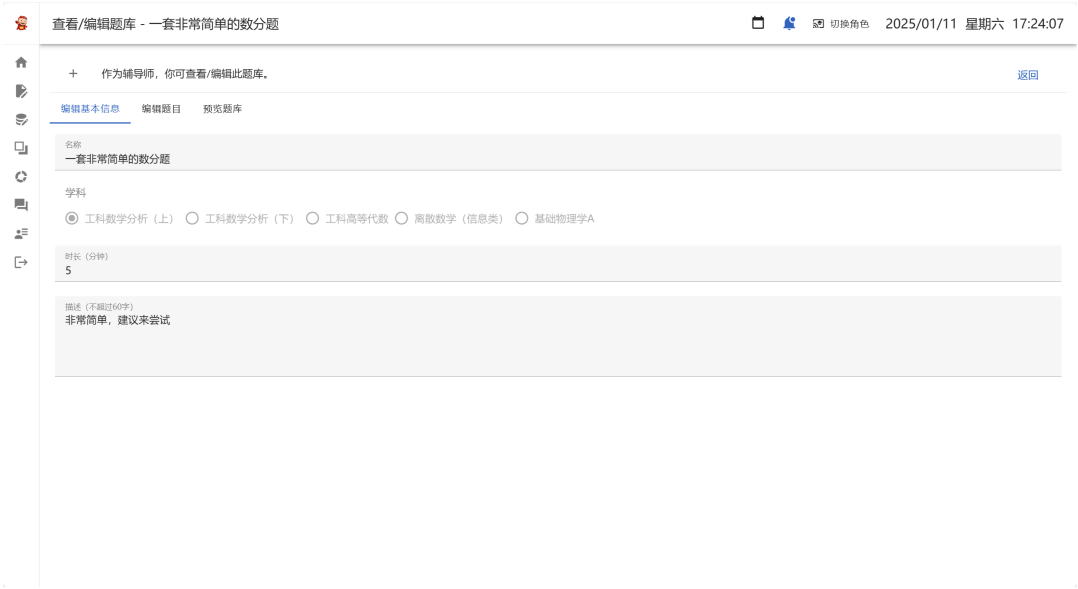
信息

- 题库列表：每个题库包括创建日期、创建者、所属题目、预计用时

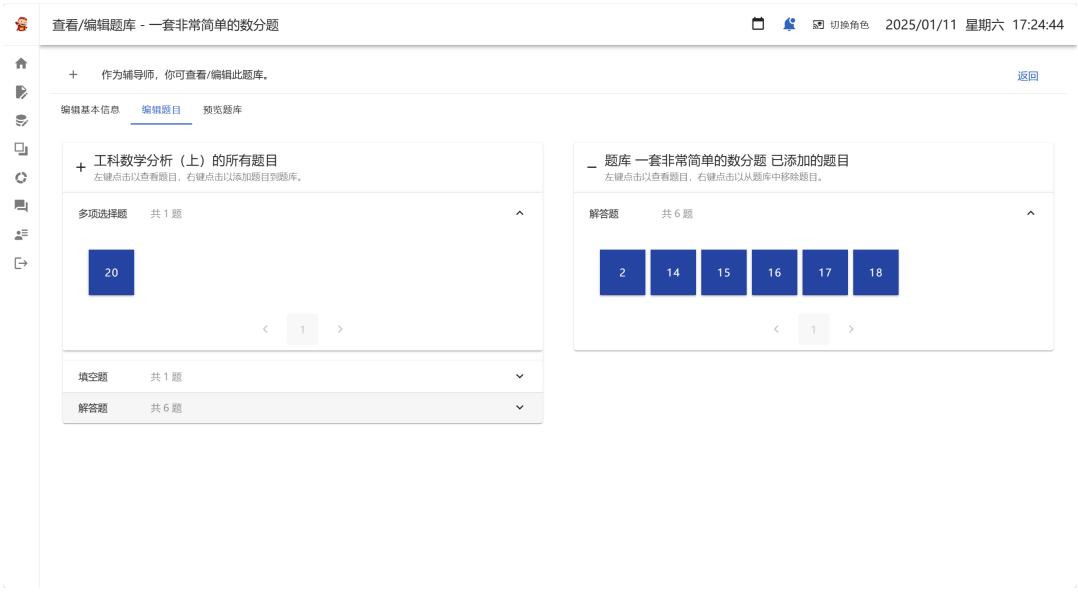
功能

- 根据科目、创建日期、名称对题库进行筛选
- 点击“删除”可删除题库
- 点击“查看/编辑”可查看题库详情；创建题库的流程类似，故一并展示
  - 编辑基本信息

    题库一旦创建，学科无法改变



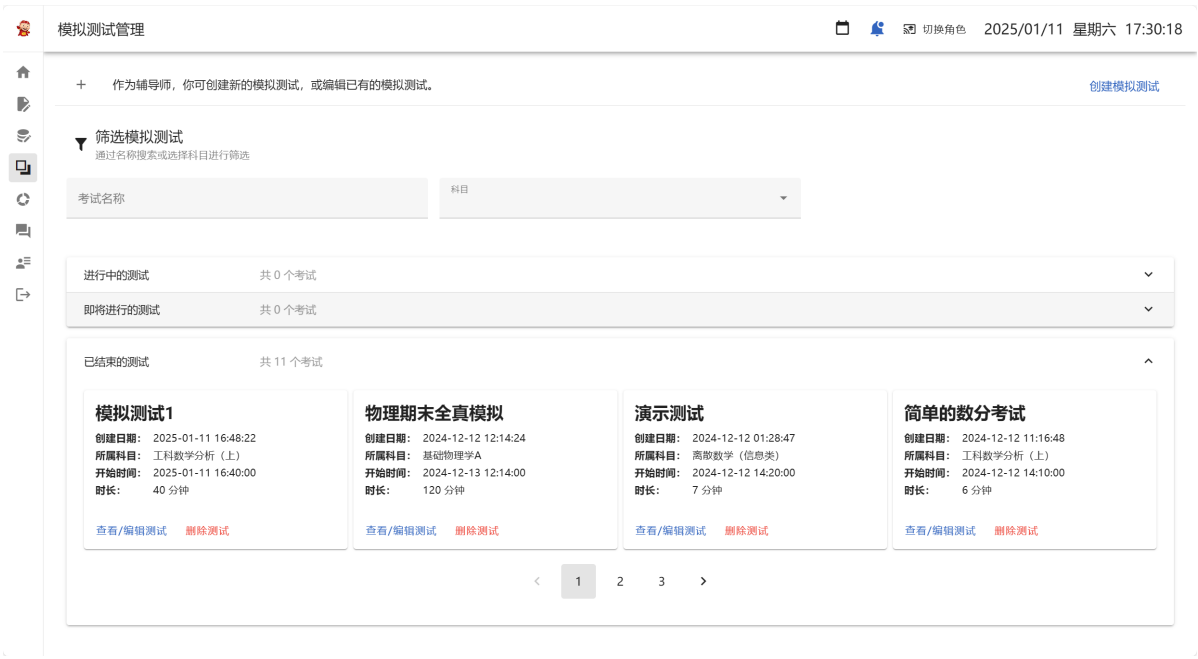
  - 编辑题目



左键点击每个题目可查看详情；右键点击每个题目可将题目移进/移出题库。

- 预览后可点击提交按钮提交新建/编辑操作。

模拟测试管理

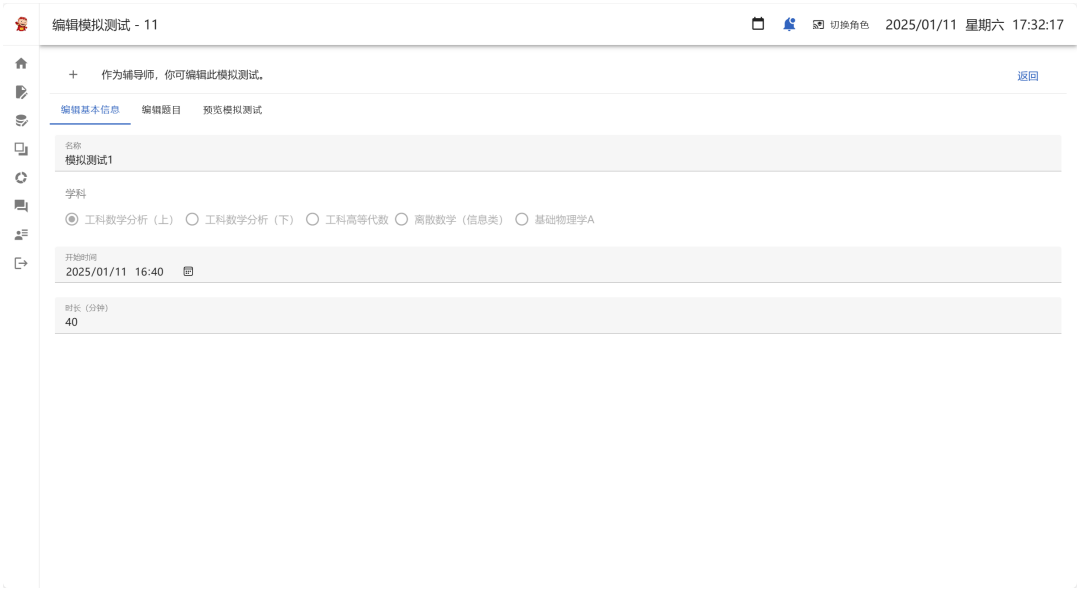


信息

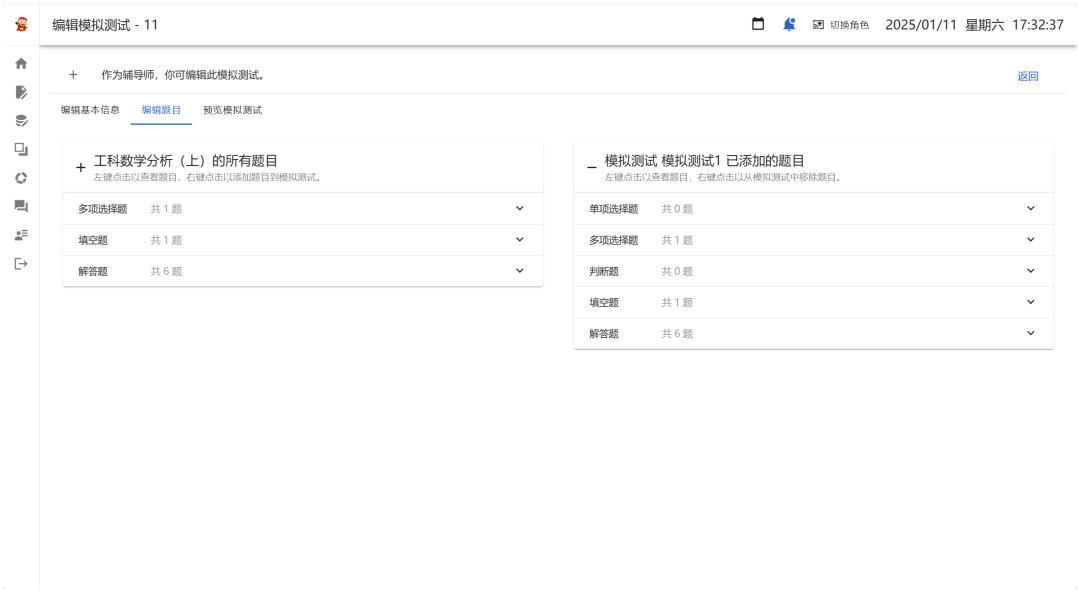
- 模拟测试列表：每个测试包括创建日期、所属题目、开始时间、时长
- 按照“进行中”“即将进行”“已完成”将测试进行分类

功能

- 根据科目、名称对模拟测试进行筛选
- 点击“删除”可删除测试
- 点击“查看/编辑”可查看或编辑测试；创建测试的流程类似
  - 编辑基本信息

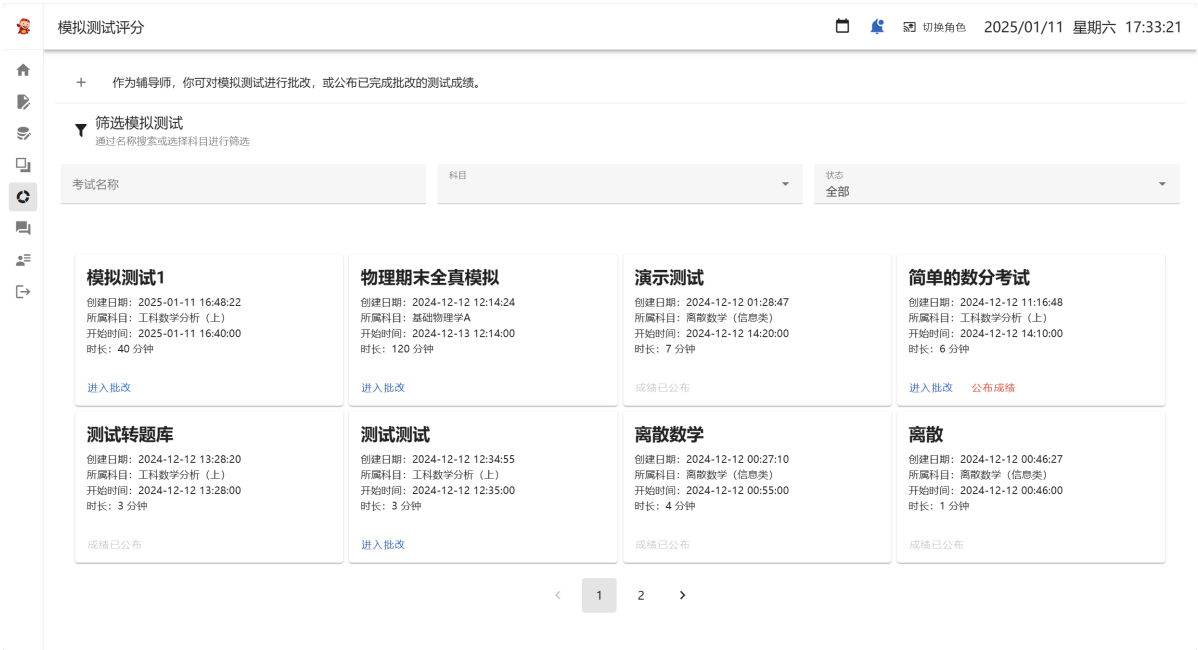


- 编辑题目：操作方法与题库管理类似



- 预览后可点击提交按钮提交新建/编辑操作。

模拟测试评分

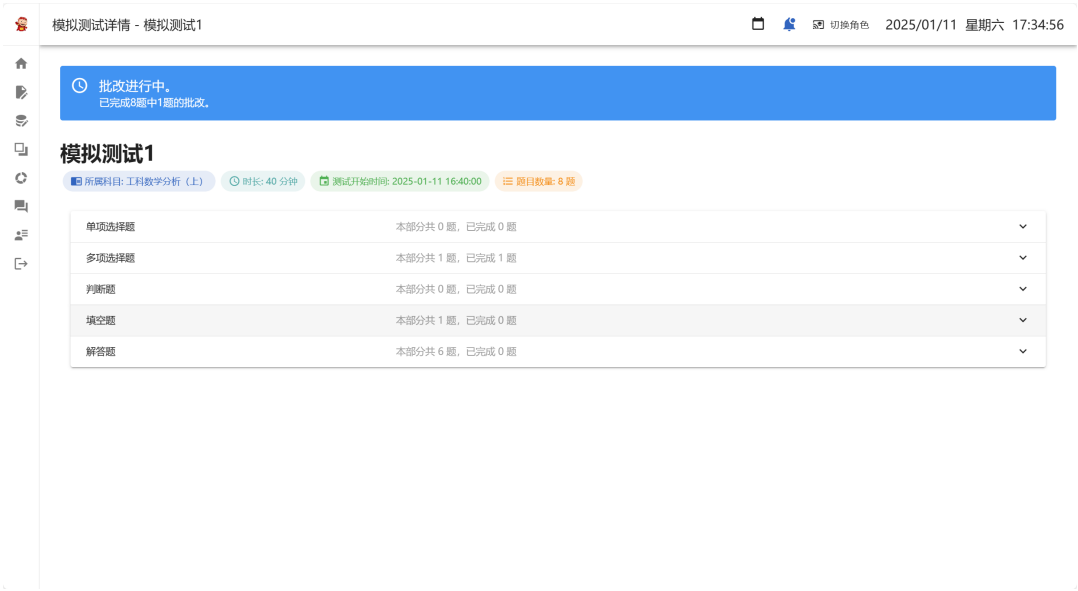


信息

- 模拟测试列表：每个测试包括创建日期、所属题目、开始时间、时长
- 按照“进行中”“即将进行”“已完成”将测试进行分类

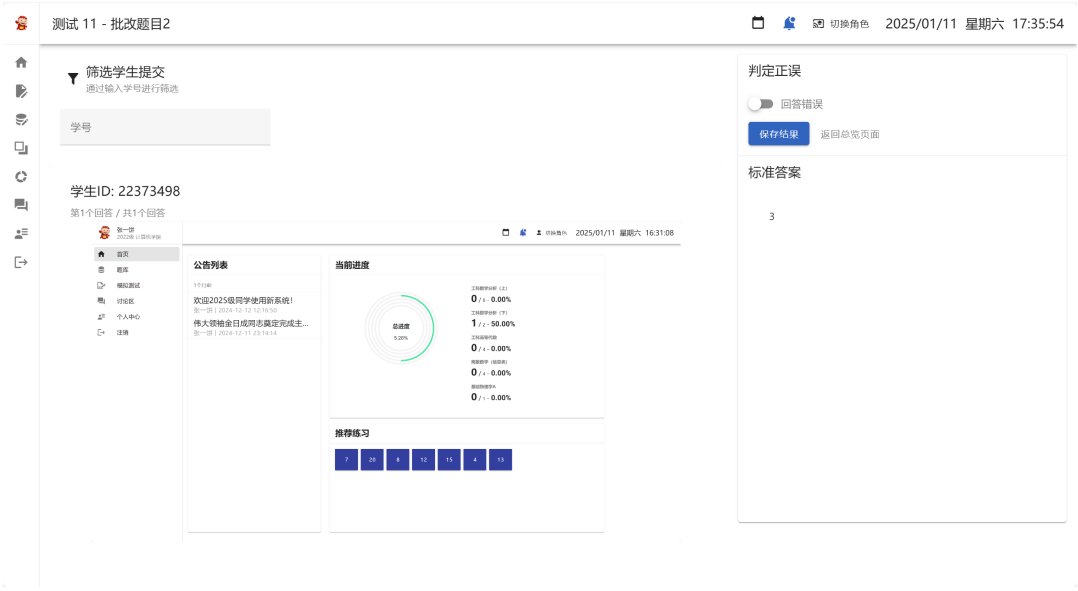
功能

- 根据科目、名称、状态对模拟测试进行筛选
- 点击“进入批改“可对测试题目进行批改
    - 批改页面详情

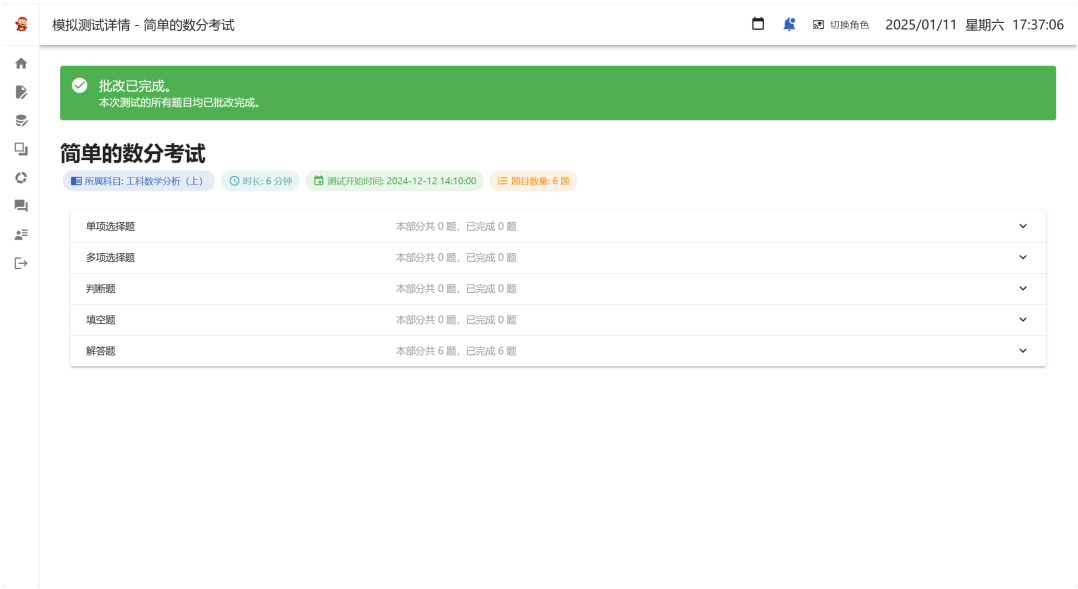


客观题可自动进行批改

主观题需要点击题目，查看该题目的所有提交

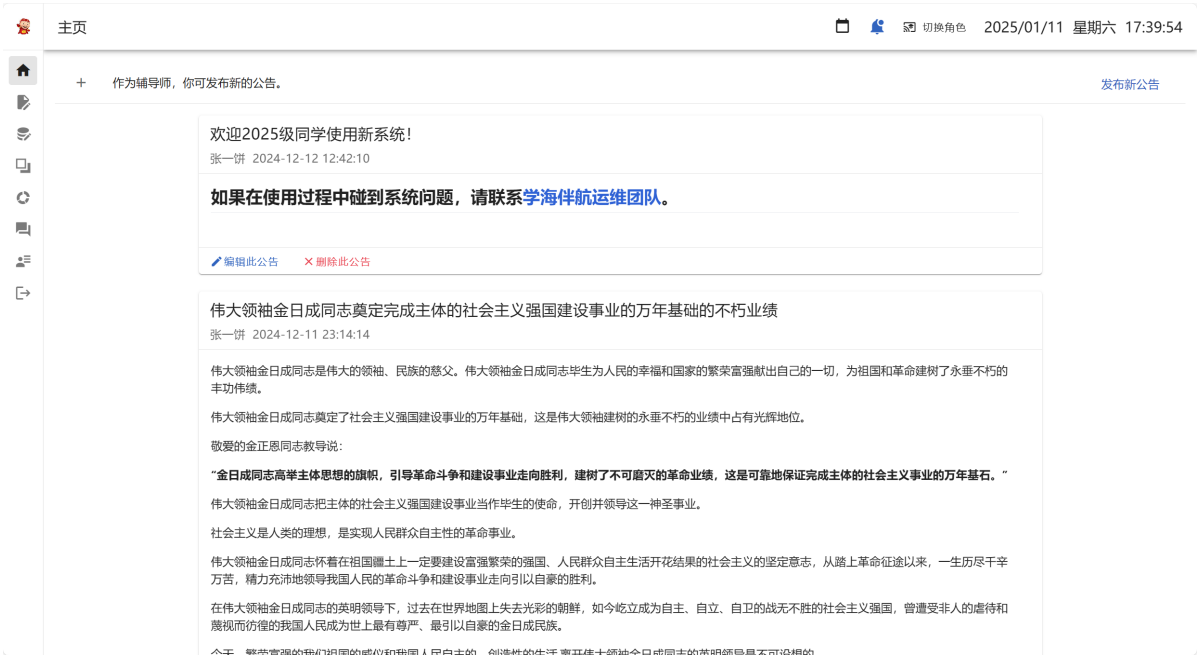


    - 如果所有题目均已批改完成，可执行公布成绩操作



- 点击”公布成绩“按钮后，状态转为”成绩已公布“，此时报名参加考试的学生可在”模拟测试/查看考试成绩“里查看自己的成绩

公告管理

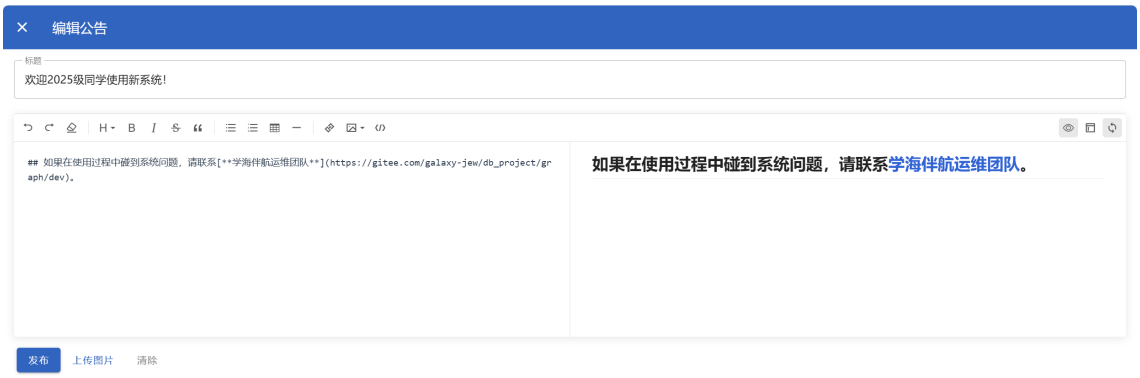


信息

- 每份公告的标题、发布者、最近修改时间、详细内容

功能

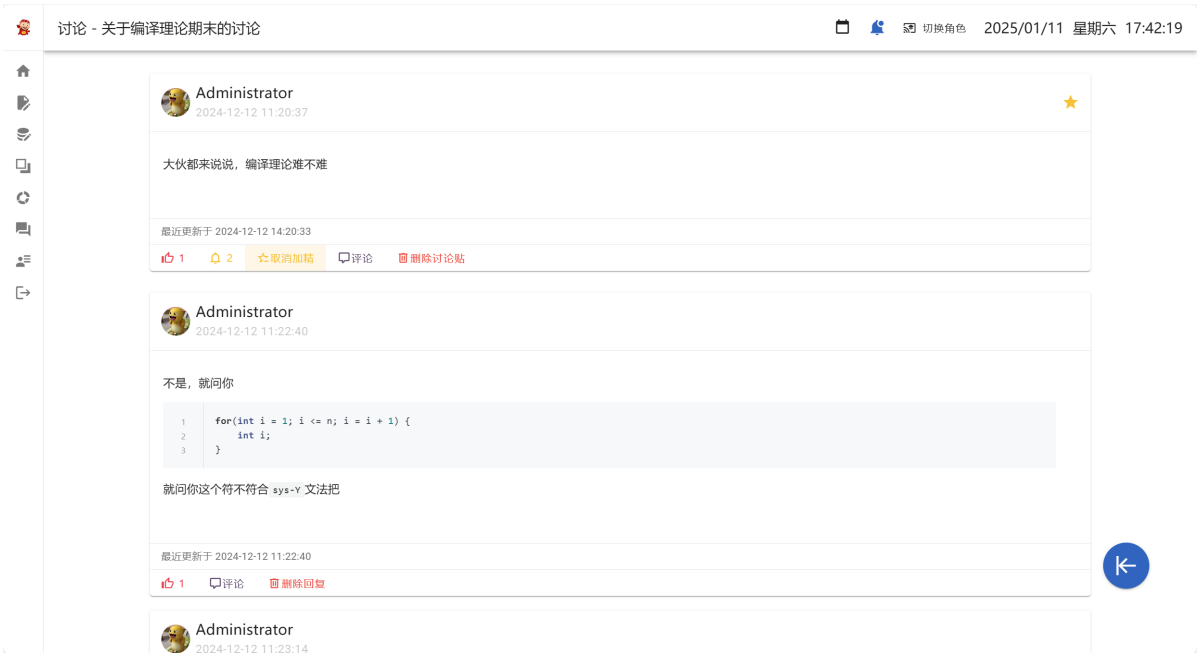
- 编辑、删除、发布公告



讨论区管理

对每一条讨论贴，可进行加精/取消加精操作

可以删除任意的讨论贴/回复贴



五、总结

我们的项目开发历程是一段充满挑战但收获颇丰的旅程。从 2024 年 10 月开始到 2024 年 12 月 12 日，我们高效地完成了一个功能复杂且多样的系统开发。这段时间虽然紧张，但却让我深刻体会到团队协作和项目设计的重要性。

在功能层面上，我们的系统包含了丰富的模块，是我们大学以来开发过的工程量最大的项目。尽管开发过程中遇到了一些困难，但我们的整体开发效率还是较高的。特别是在系统设计上，我们采用了“设计-讨论修改-再设计”的开发思路，循序渐进，让数据库的结构和各功能模块之间的关系更加合理，在系统的可扩展性和稳定性上也有了很大的提升。我们的功能实现不仅满足了预期，还展现出了一些创新的亮点，这也让我们对于自身的开发能力有了更大的信心。

当然，作为首次接触如此规模的开发项目，经验不足带来了一些问题，但这些并未成为阻碍。比如，在接口对接方面，由于初期没有编写接口文档，导致了一些前后端的命名不一致和对接障碍。但正是通过这些问题的解决，我们深刻认识到接口文档的重要性，并在后续的开发中逐步建立了规范，显著提升了效率。类似的问题在版本管理上也有所体现，虽然起初因为没有完善的版本控制机制而导致了一些代码覆盖问题，但通过及时调整，我们学会了如何更好地使用 Git 进行版本管理，后期的开发更加井然有序。

在团队协作方面，虽然最初的沟通还不够充分，出现了思路不一致的问题。但随着开发的深入，我们逐渐形成了高效的组会机制和进度同步方式，组员之间直接沟通。这样不仅避免了很多思路上的分歧，还让团队每个人的能力都得到了最大化的发挥。尤其值得一提的是，在对齐接口和调试错误方面，我们三位成员的代码开发能力和纠错能力都有了显著的增强。这种成长是一次宝贵的经验，为未来的开发积累了丰富的实践基础。

整个项目完成后，大家都有满满的成就感。回顾整个开发过程，从最初的需求分析、设计数据库结构、到系统功能逐步实现，我们不仅提升了自己的开发技能，也更加明白了协作和设计的重要性。这段经历让我们意识到，工程开发不仅是对技术能力的考验，更是对思维能力、沟通能力和团队协作能力的全面锻炼。总的来说，这是一次令人难忘的经历，也是一段充满收获的成长旅程！

下面这张图，就展示了团队在2个月的开发历程中，多达216次的版本更新历程。


Merge branch 'dev' of gitee.com:galaxy_jee/db_project into dev
添加图片上传接口
modify显示的bug
fix: 恢复查看成绩bug
finish
正确错误
update
bug恢复
恢复解答题显示bug
push
恢复意外弹窗bug
finish
modify sql
great merge
fuck
完成考试日程视频
1
美化
添加解答题问答下载按钮
解答
merge too
添加上一次答案显示
修改考试格式
go
Merge branch 'dev' of gitee.com:galaxy_jee/db_project into dev
修复考试概览样式
修完图片
用户exam
fix: 完善获取考试视图函数逻辑
fix: 完善获取考试视图函数逻辑
delete questionbank
初步检查接口无误
恢复讨论区弹窗bug
完成题目和题库功能
real merge
merge
添加loading动画效果
Merge branch 'dev' of gitee.com:galaxy_jee/db_project into dev
修改页面加载异步问题
adminborad
题目，题目
公告栏添加动画
初步解决闪现问题
big merge
Merge branch 'dev' of gitee.com:galaxy_jee/db_project into dev
尝试添加加载动画
fix: 恢复了编辑题目和考试的视图函数
删除讨论帖与评论添加确认弹窗
fix: 完善学生查看考试批改页面视图
添加成绩显示页面，真不想写前端了
Merge branch 'dev' of gitee.com:galaxy_jee/db_project into dev
批改页面修改
fix: 完善自动批改逻辑
fix: 完善考试科目
feat：新增老师获取所有考试是否已批改
添加评分筛选逻辑
添加公布接口
merge
Merge branch 'dev' of gitee.com:galaxy_jee/db_project into dev
修改批改页面
feat：新增老师获取当前考试批改状态视图函数
feat：完善模拟考试公布结果相关视图函数逻辑
没加上跨域
恢复前端
预习message，修改完一些小bug
添加题库按照名称的筛选逻辑
Merge branch 'dev' of gitee.com:galaxy_jee/db_project into dev
fix css
feat：完善模拟考试相关视图函数逻辑
Merge remote-tracking branch 'origin/dev' into dev
fix: 更改显示消息和公告数量
回车键触发登录
feat：补全当前学生获取考试内某题的信息，包含题目信息和学生的已有作答的视图函数
feat：补全当前学生获取考试内某题的信息，包含题目信息和学生的已有作答的视图函数
feat：优化发送消息逻辑，新增已读所有消息，已读单未读，获取所有未读消息函数
1
Merge branch 'dev' of gitee.com:galaxy_jee/db_project into dev
完成消息中心
fix: 恢复了发送的消息不显示头像的bug，完善了消息格式
fix
修改缓存机制
fix ...的bug
修改个人中心的bug
fix：恢复图片上传bug
完成图像上传，恢复点赞bug
恢复已知的bug
Merge branch 'dev' of gitee.com:galaxy_jee/db_project into dev
统一弹窗接口
Merge branch 'dev' of gitee.com:galaxy_jee/db_project into dev
feat:于backend文件夹内完成图片上传功能
恢复一些前端报错
添加文件上传方法，修改解答题提交逻辑
尝试部署
修改已知bug
Merge branch 'dev' of gitee.com:galaxy_jee/db_project into dev
修改讨论区bug
最新
刚刚不小心把前端覆盖了
完成加精，完成时间修改，删除讨论区和回复
添加最近更新时间显示
完成主页修改

完成board、broadcast、discussions、用户界面（除图片上传）
完成讨论区功能80%
修改个人中心无法更改，讨论区路径错误的bug
discussions
修复主页显示不同步问题
完成board、broadcast、个人主页的接口（除图片上传），前端还有一些地方有点问题
!24 最新版前端
board、broadcast
需要完成讨论区功能
!23 完成前端基本
前端基本完成
添加设置题目分值功能
!22 完善前端设计
修改样式
fix
更改导览语
add
!21 最新版前端
修改时间显示
添加编辑题库功能
修改分页样式
对所有的panel元素使用换页
实现题库添加题目，待实现预览、编辑与删除功能
Merge branch 'ZGTdev' of https://gitee.com/galaxy-jem/db_project into dev
feat: 完善了question类的逻辑，新增编辑指定id题目，编辑指定题库内题目，向指定题库添加指定id的题目，在指定题库内创建新题目，删除指定题库内指定id题目的功能并完成本地测试
尝试实现题库提交
完成了编辑删除题目的逻辑
Merge branch 'ZGTdev' of https://gitee.com/galaxy-jem/db_project into dev
feat: 完善了question类的逻辑
Merge branch 'ZGTdev' of https://gitee.com/galaxy-jem/db_project into dev
feat: 完善了question类的逻辑
Merge branch 'ZGTdev' of https://gitee.com/galaxy-jem/db_project into dev
fix: 修复了一个愚蠢的get post bug
添加创建新题目界面
Merge branch 'ZGTdev' of https://gitee.com/galaxy-jem/db_project into dev
feat:完成了根据考试的相关逻辑，支持学生提交、老师批改，并在本地完成了相应测试，测试可能较简单，同时修改了Questionbank detail返回数据，便于前端获取信息
添加删除确认弹窗
Merge branch 'dev' of gitee.com:galaxy-jem/db_project into dev
修改部分文锁
Merge branch 'ZGTdev' of https://gitee.com/galaxy-jem/db_project into dev
feat:完善broadcast的设计与逻辑，新增修改删除broadcast接口，完成讨论区与回复设计，均完成本地测试，讨论区测试样例可能存在不够完备的情况
添加删除按钮
完成修改公告操作，待完成删除公告
!20 完善前端设计
✨ feat(辅导师页面)：添加查看通知和发布通知的页面
准备添加辅导师页面
!19 前后端贯通
✨ feat(个人信息)：profilecontent实现从本地缓存读取信息
✨ feat(登录注册)：登录之后显示对应的名字等信息
!18 返回信息
返回信息
!17 最新版前端
✨ feat(登录注册)：完成登录注册信息的本地缓存
Merge branch 'ZGTdev' of https://gitee.com/galaxy-jem/db_project into dev
打通登录注册前后端