选课系统实验报告

17120198 姜辰昊

一、小组成员及分工

姜辰昊：学生子系统
胡逸冲、蒋启源：教师、管理员子系统

二、开发工具

IDE：vscode
前端：vue
前端框架：vuetify
后端：django
数据库：sqlite3

三、功能展示

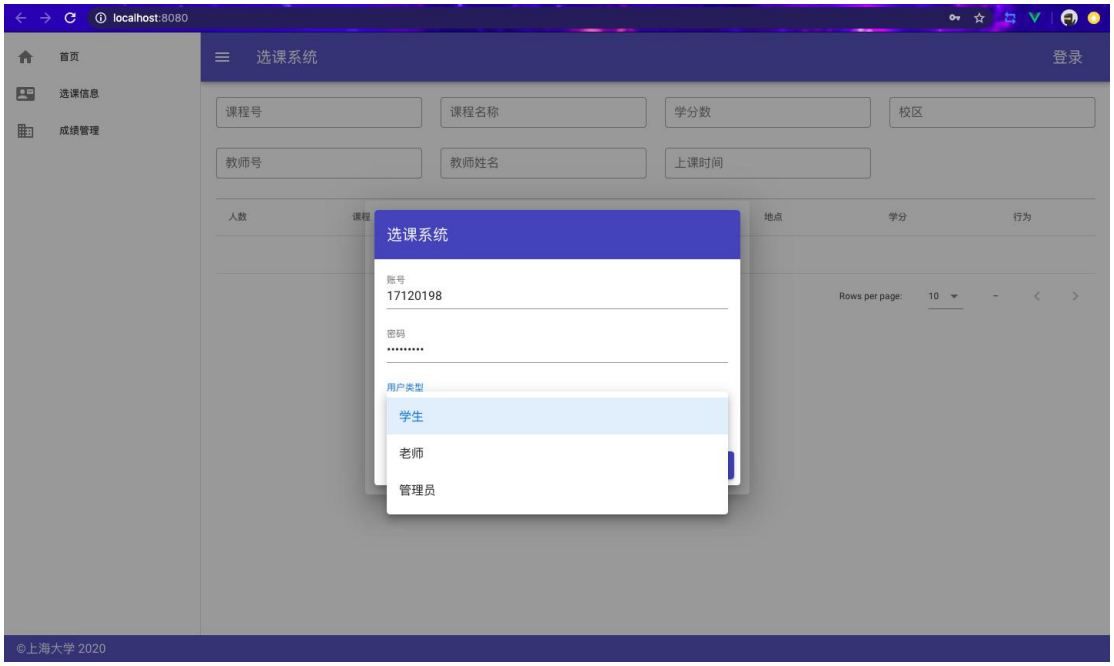

图 1 登录界面

用户点击右上角登录按钮，弹出登录输入框，输入学号和密码，并选择“学生”即可进入学生选课子系统。

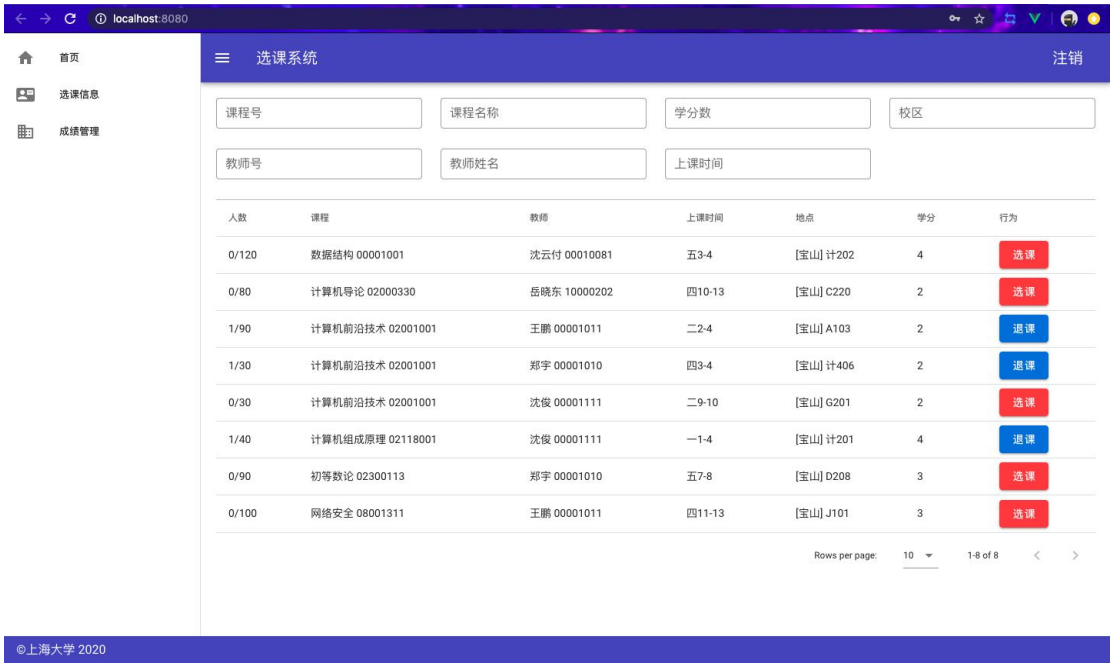

图 2 选课页面

图 3 选课搜索

上方输入框为用户提供了筛选课程的功能，用户通过它进行模糊查询，下方显示搜索条件下满足要求的课程，当然，课程一定是可选的（本学期开课课程）。例如，当输入课程号为“02001001”时，会显示计算机前沿技术的 3 门课程，如果在课程名中输入“前沿”，也能得到相同结果，这是因为“前沿”字符串包含于“计算机前沿技术”字符串中。同样的，教师姓名、校区等也有类似模糊查询的效果，对于上课时间，我却不满足于此，在使用学校教务处的选课系统过程中，我发现上课时间项很难做到模糊查询的效果，因为在本系统中，我先规定输入格式为[星期几][时间序号]-[时间序号],然后进行模糊处理，即如果输入“五”则会显示所有星期五的课程，输入“四”则会显示所有星期四的课程，然后对时间序

号进行模糊查询，很多时候学生知道自己哪个时间段有空，想要挑选这个时间段的课程，因此输入的时间段范围应该大于所筛选的课程起止时间，例如输入“四3-8”，则会筛选出周四的课程中开始时间在第 3 节课及以后，结束时间在第 8 节课及以前的课程。

点击右侧行为栏选课键，则发出选课请求，返回选课成功信号，则选课键变成蓝色的退课键，并且左侧选课人数+1，按退课键过程类似。

当课程信息比较多时，会以分页的形式进行显示，并且可以通过对各列的排序，快速查找自己需要的课程。

图 4 选课信息界面

点击左侧“选课信息”栏进入选课信息界面，这里显示了已选的课程列表（已选但课程未开始），界面与首页类似，区别在于点击退课，当退课成功后从列表中删除，需要从首页中重新选课，此外，左上角显示了此学期修的学分数。

图 5 成绩管理界面

点击左侧成绩管理栏进入成绩管理界面，此处列表显示了已结束的课程，左上角显示总学分数和平均成绩。如需加入绩点数据，只需要在后端加入成绩到绩点的映射即可。

四、代码实现

本实验采用前后端分离的技术，前端使用 vue，后端使用 python 的 django 框架，由于 django 使用 ORM 数据库，即数据库表通过类的形式来表示，而查询语言抽象为 python 函数，因此后端代码与普通后端有所不同。本节将介绍密码加密、token 实现、数据库模型和功能实现相关代码，以及触发器和存储过程的实现。

1.密码加密与 token 实现

```
//前端登录代码
login() {
        if (this.user_type == "学生") {
            this.$http({
                method: "post",
                url: "/api/student/students/login/",
                data: {
                    sid: this.account,
                    password: this.Base64.encode(this.pwd)
                }
            })
                .then(res => {
                    //登录成功
```

```
                if (res.data) {
                    localStorage.setItem("id", this.account);
                    localStorage.setItem("password", this.pwd);
                    localStorage.setItem("token", res.data.token);
                    this.clearInput();
                    this.$emit("login", res.data.token);
                } else {
                    this.account_failed = true;
                }
            })
            .catch(error => {
                console.log(error.response);
            });
        }
        //教师或管理员登录
        //.....
}
```

前端发送 post 请求发送给后端，sid 为学号，password 通过 Base64 加密，当返回登录成功后，在浏览器中记录学号密码和 token 以便一段时间内免登录，而 token 也用于其他功能发送请求的时候认证身份。

```
#后端登录实现（部分）
#serializers.py
def validate(self, attrs):
        try:
            username = student.models.Student.objects.get(
                sname=attrs['username'],
            ).user.username
        except student.models.Student.DoesNotExist:
            username = get_random_string(
                length=8,
                allowed_chars="abcdefghijklmnopqrstuvwxyz0123456789",
            )
        attrs['username'] = username
        p = attrs['password']
        attrs['password'] = smart_text(base64.decodebytes(bytes(p, 'utf-8')))
        return super().validate(attrs)

#views.py
if serializer.is_valid():
    user = serializer.validated_data['user']
    if has_expired(user.auth_token):
        user.auth_token.delete()
```

```
token, created = Token.objects.get_or_create(user=user)
return Response({
    'token': token.key
})
```

由于 Django 自带 user 表，提供了权限认证、密码等功能，因此我们将每个用户都唯一对应于一个 user，把密码存在 user 中，其他如学号等信息存放在 student 表中，登录时查找对应的 user，将 password 解密，根据 username 和 password 判断是否匹配，如果用户的 token 已经过期，则重修发放一个 token 给用户。

2.数据库模型

```
class Student(models.Model):
    '''
    [学生表]
    sid:学号
    sname:姓名
    user:Django 自带 user
    schoolid:学院号
    '''
    sid=models.CharField(max_length=100,blank=False,unique=True,primary_key=True)
    sname=models.CharField(max_length=100,blank=False)
    user = models.OneToOneField(
        to=User,
        on_delete=models.CASCADE,
        blank=False
    )
    schoolid=models.ForeignKey(
        to=sysadmin.models.School,
        on_delete=models.CASCADE,
        related_name='student',
        to_field="schoolid",
        blank=False
    )
```

我们以学生表为例详细介绍 ORM 数据库表的定义，对于其他表仅复制注释进行简单展示。

可以看出，Student 类继承了 models 的 Model 类，分别定义了 sid、sname、user、schoolid 4 个属性，其中 sid 为主键，user 和 schoolid 都是外键，user 对应与 User 表且是 1 对 1 对应关系，而 schoolid 对应于 School 表，Django 的表允许反向查询。

```
class ChooseCourse(models.Model):
    '''
    [选课表]
```

```
    sid:学号
    ocid:开课号
    pscore:平时成绩
    kscore:考试成绩
    zscore:总评成绩
    '''

class SystemAdmin(models.Model):
    '''
    [管理员表]
    adminid:管理员号
    adminname:管理员姓名
    user:Django 自带 user
    '''

class Campus(models.Model):
    '''
    [校园表]
    xid:校区号
    xname:校区地点
    '''

class Term(models.Model):
    '''
    [学期表]
    termname:学期名
    '''

class School(models.Model):
    '''
    [院系表]
    schoolid:院系 id
    schoolname:院系名
    '''

class Course(models.Model):
    '''
    [课程组表]
    cid:课程号
    cname:课程名
    score:学分
    '''

class Teacher(models.Model):
```

```
    '''
    [教师表]
    tid:学号
    tname:姓名
    user:Django 自带 user
    schoolid:学院号
    '''

class OfferCourse(models.Model):
    '''
    [开课表]
    tid:教师号
    cid:课程号
    termid:学期
    maxnum:最大人数
    choosenum:选课人数
    xid:校区号
    time:上课时间
    place:上课地点
    '''
```

3.功能实现举例

```
class GetOfferCourses(views.APIView):
    '''
    多条件模糊查询可选课
    '''
    permission_classes = [permissions.IsAuthenticated]

    def get(self, request, *args, **kwargs):
        params = request.query_params
        user=request.user
        q={}
        cid_list=list()
        cname_list=list()
        score_list=list()
        termid=4
        q['termid']=termid
        if params['cid']!="":
            cid_list= list(map(lambda x: x['cid'], [{'cid':params['cid']}]))
        if params['cname']!="":

cids=sysadmin.models.Course.objects.filter(cname__icontains=params['cname']).order_
by("cid").values("cid").distinct()
```

```
            cname_list = list(map(lambda x: x['cid'], cids))
        if params['score']!="":
            try:

score_cids=sysadmin.models.Course.objects.filter(score=int(params['score'])).order_by("
cid").values("cid").distinct()
            except:
                score_cids=[]
            score_list = list(map(lambda x: x['cid'], score_cids))

        if params['cid']!="" or params['cname']!="" or params['score']!="":
            flag=True
            if params['cid']!="":
                con_clist=cid_list
                flag=False
            else:
                con_clist=list()
            if (not flag) and params['cname']!="": #交
                con_clist=list(set(con_clist).intersection(set(cname_list)))
            elif flag and params['cname']!="": #并
                con_clist=cname_list
                flag=False
            if (not flag) and params['score']!="": #交
                con_clist=list(set(con_clist).intersection(set(score_list)))
            elif flag and params['score']!="": #并
                con_clist=score_list
                flag=False
            if not flag:
                q['cid__in']=con_clist

        tid_list=list()
        tname_list=list()
        if params['tid']!="":
            tid_list= list(map(lambda x: x['tid'], [{'tid':params['tid']}]))
        if params['tname']!="":

tids=teacher.models.Teacher.objects.filter(tname__icontains=params['tname']).order_by
("tid").values("tid").distinct()
            tname_list = list(map(lambda x: x['tid'], tids))

        if params['tid']!="" or params['tname']!="":
            flag=True
            if params['tid']!="":
                con_tlist=tid_list
```

```
                flag=False
            else:
                con_tlist=list()
            if (not flag) and params['tname']!="": #交
                con_tlist=list(set(con_tlist).intersection(set(tname_list)))
            elif flag and params['tname']!="": #并
                con_tlist=tname_list
                flag=False
            q['tid__in']=con_tlist

        if params['xname']!="":
            try:

xid=sysadmin.models.Campus.objects.get(xname__icontains=params['xname'])
            except:
                xid=-1
            q['xid']=xid
        q["status"]=0

        qstart=0
        qend=0
        time_flag=False
        if params['time']!="":
            try:
                qstart=int(params['time'].split("-")[0][1:])
                qend=int(params['time'].split("-")[1])
                time_flag=True
            except:
                q['time__icontains']=params['time']
        if time_flag:
            q['course_day']=params['time'][0]
            q['cstart__gte']=qstart
            q['cend__lte']=qend
            offer_courses=teacher.models.OfferCourse.objects.annotate(
                course_day=Substr(F('time'),1,1),

cstart=Cast(Substr(F('time'),2,StrIndex(F('time'),V('-'))-2,output_field=CharField()),Inte
gerField()),

cend=Cast(Substr(F('time'),StrIndex(F('time'),V('-'))+1,output_field=CharField()),Intege
rField())
            ).filter(
                **q
            ).order_by("cid")
```

```
    else:
        offer_courses=teacher.models.OfferCourse.objects.annotate(
            cc_sid=Cast(F('choose_course__sid'),IntegerField())
        ).filter(
            **q
        ).order_by("cid")
    rtn=[]
    for instance in offer_courses:
        serializer = student.serializers.GetOfferCoursesSerializer(
            instance=instance,
            context={'request': request}
        )
        oc=serializer.data
        chosen=(instance.cc_sid==int(user.student.sid))
        #chosen
        # chosen=student.models.ChooseCourse.objects.filter(
        #         sid = user.student,
        #         ocid = instance.id
        # ).exists()
        oc["chosen"]=chosen
        rtn.append(oc)
    return Response(rtn)
```

本节以最复杂的课程搜索功能举例进行讲解，该函数继承于 views.APIView 类，在 urls.py 处进行定义即可使用。perssions 定义了该功能需要认证身份，如果没有 token 或 token 过期将返回 401 或 400 错误，然后 get 函数定义了获得选课信息的函数体，分别判断 cid，cname，score 等参数是否为空，如果是空则忽略该项，认为在该项处选择查询全部，如果参数非空，则对参数进行处理，获得最终查询表 OfferCourse 的属性数据，如给出 cname，需要向 Course 表查询对应的 cid，如给定 score，也需查询对应的 cid，所有参数非空项的 cid 做交，得出 cid 属性对应的范围，加入 q 中，q[cid__in]=con_clist 的__in 代表查询 cid 在 con_clist 中，同理获得其他属性的约束条件，特别的，对于上课时间项，需要将字符串如“四 3-8”进行分割，所以需要用到 annotate 声明变量，用 Substr 切割字符串，在 filter 筛选条件中加入 q[course_day]、q[cstart__gte]、q[cend__lte]即可实现功能。

4.触发器实现

由于 Django 语言与 SQL 不同，触发器需要以特殊的方式进行实现，如下实现了学生选课，课程的选课人数+1，学生退课，课程的选课人数-1 功能。

```
@receiver(pre_save, sender=ChooseCourse)
def pre_save_choosecourse(sender, instance, **kwargs):
    offercourse = instance.ocid
    offercourse.choosenum += 1
    offercourse.save()
```

```
@receiver(pre_delete, sender=ChooseCourse)
def pre_delete_choosecourse(sender, instance, **kwargs):
    offercourse = instance.ocid
    offercourse.choosenum -= 1
    offercourse.save()
```

5.存储过程实现

Django ORM 数据库实际上无需存在存储过程，因为存储过程无疑是定义在数据库内的函数，通过参数输入函数，减少对数据库查询语言的定义次数，而显然 Django 已经存在类似的功能。由于上学期选课系统后端我使用 java spring、ybatis 实现，我认为其 xml 文件与存储过程非常类似，因此帖上 xml 文件代码当作是对存储过程的实现。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.example.xks.mapper.ChooseCourseMapper">

    <resultMap id="ResultMapDetail" type="com.example.xks.entity.ChooseCourse">
        <id property="id" column="id"/>
        <result property="sid" column="sid"/>
        <result property="cid" column="cid"/>
        <result property="tid" column="tid"/>
        <result property="time" column="time"/>
    </resultMap>

    <insert id="chooseCourse" useGeneratedKeys="true" keyProperty="id">
        insert into choose_course(sid,cid,tid,time)
        values(
        #{sid},#{cid},#{tid},#{time})
    </insert>

    <delete id="cancelCourse">
        delete from choose_course where sid=#{sid} and cid=#{cid} and tid=#{tid} and time=#{time}
    </delete>

    <select id="judgeChosen" resultMap="ResultMapDetail">
        select * from choose_course where sid=#{sid} and cid=#{cid} and tid=#{tid} and time=#{time}
    </select>
</mapper>
```

五、体会

通过本学期数据库的实验课程，我初步掌握了前后端分离 web 项目的前后端制作，无论是 sql 还是 django 数据库，学会了数据库更高级的用法，保证了数据库的完整性。我希望在未来能更详细的钻研前后端技术，对特定的语言/框架，熟悉其函数、类的用法，从照样画葫芦变成深刻理解框架、项目的架构。