选课系统实验报告

17120198 姜辰昊

1. 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

1. 开发工具

IDE：vscode

前端：vue

前端框架：vuetify

后端：django

数据库：sqlite3

1. 功能展示

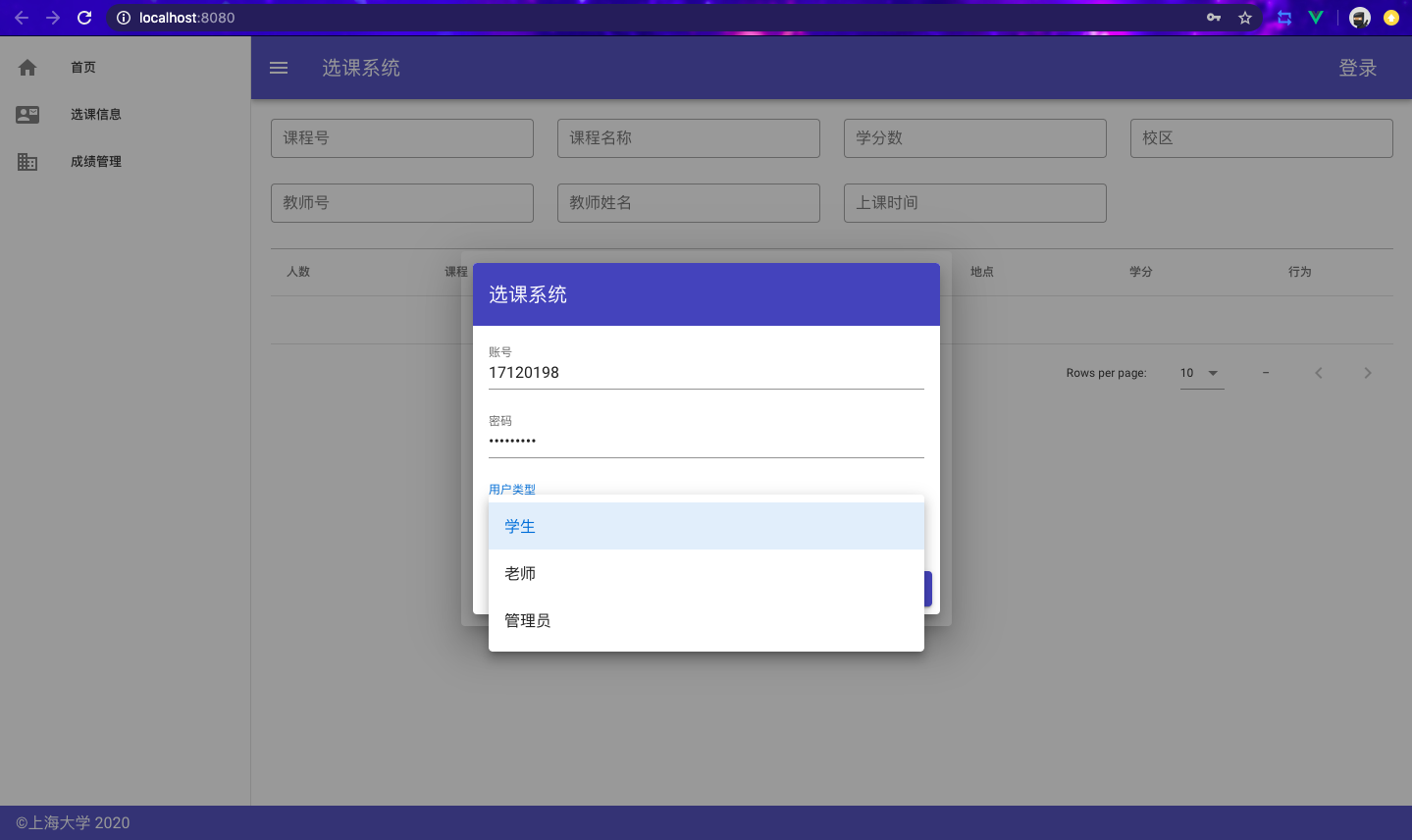


图1 登录界面

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。

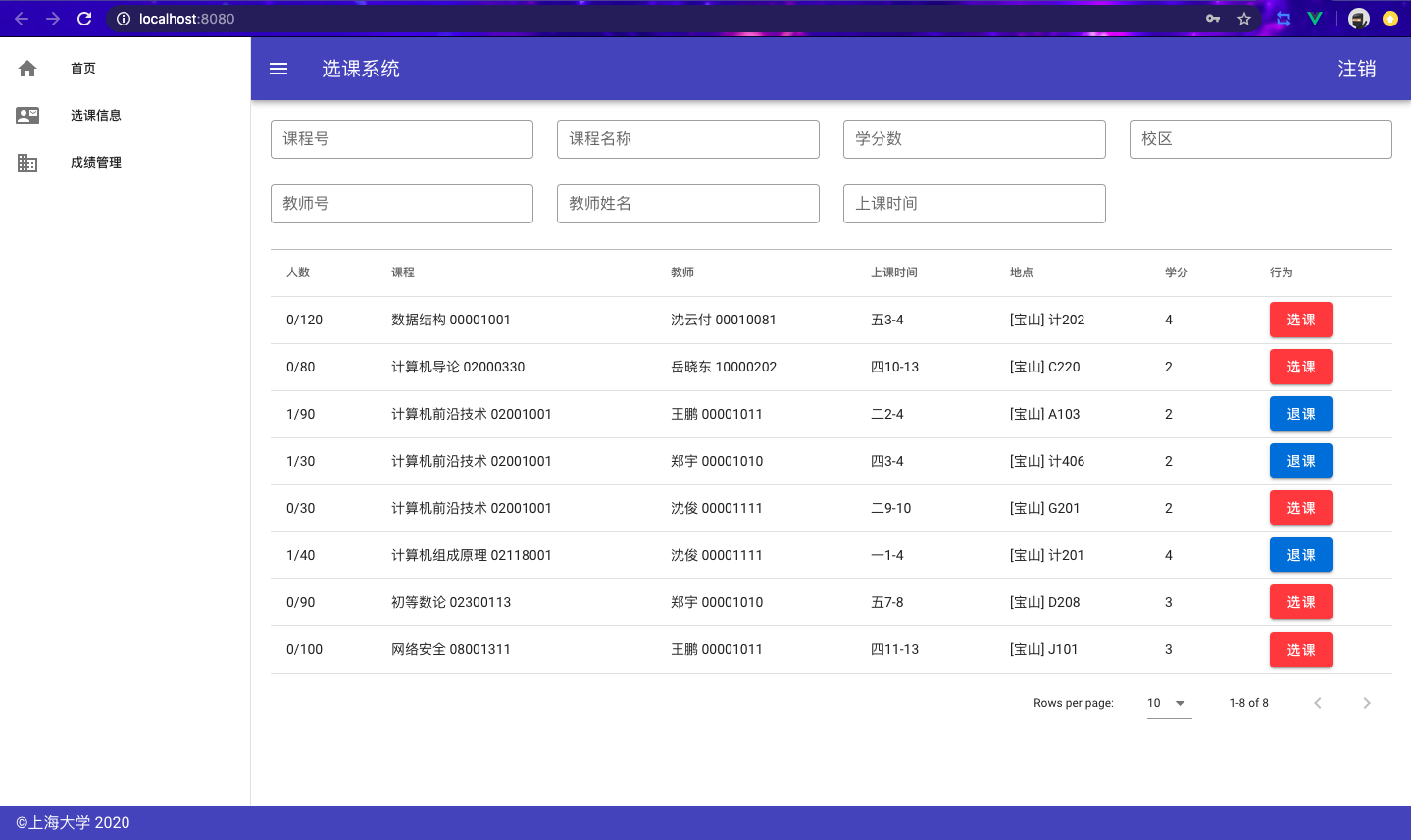


图2 选课页面



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()),IntegerField()),

cend=Cast(Substr(F('time'),StrIndex(F('time'),V('-'))+1,output\_field=CharField()),IntegerField())

).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。

1. 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()

1. 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>

1. 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。