

首页

前端显示首页

首页组件代码

Home.vue

```
<template>
  <div class="home">
    <Header :current_page="current_page"/>
    <Banner/>
    <Footer/>
  </div>
</template>

<script>
  import Header from "../common/Header"
  import Banner from "../common/Banner"
  import Footer from "../common/Footer"
  export default{
    name:"Home",
    data(){
      return {
        current_page:0,
      };
    },
    components:{
      Header,
      Banner,
      Footer,
    }
  }
</script>

<style scoped>
.home{
  padding-top: 80px;
}
</style>
```

首页头部子组件

Header.vue

[illegible]

```

</div>
</template>

<script>
  export default {
    name: "Header",
    props: ["current_page"],
    data() {
      return {
        is_login: false, /* 是否登录 */
        nav_list: [
          { "name": "首页", "link": "http://http://127.0.0.1:8080/" },
          { "name": "轻课", "link": "http://http://127.0.0.1:8080/" },
          { "name": "学位课", "link": "http://http://127.0.0.1:8080/" },
          { "name": "题库", "link": "http://http://127.0.0.1:8080/" },
          { "name": "教育", "link": "http://http://127.0.0.1:8080/" },
        ],
      }
    },
    created: function() {
      // 获取轮播图
      this.$axios.get("http://api.luffycity.cn:8000/home/nav/").then(res => {
        console.log(res.data)
        this.nav_list = res.data
      }).catch(error => {
        console.log(error);
      });
    }
  }
</script>

<style scoped>
.header{
  width: 100%;
  height: 80px;
  background: #fff;
  position: fixed;
  left: 0;
  top: 0;
  margin: 0 auto;
  z-index: 7;
  text-overflow: ellipsis;
  white-space: nowrap;
  box-shadow: 0 0.5px 0.5px 0 #c9c9c9;
}
.el-container{
  width: 1200px;
  margin: 0 auto;
}
.el-header{
  padding: 0;
}
.el-row, .el-col{

```

```
height: 80px;
}
.logo{
width: 118px;
}
.logo,.nav{
line-height: 80px;
}
.logo img{
vertical-align: middle; /* 设置图片垂直居中 */
}
.nav{
margin-left: 30px!important;
margin-right: 0!important;
}
.nav .el-col .active{
padding-bottom: 16px;
padding-left: 5px;
padding-right: 5px;
border-bottom: 4px solid #ffc210;
}
.nav,.study{
text-align: center;
color: #4a4a4a;
}
.nav .el-col:hover,.nav .el-col a:hover,.study:hover,.study a:hover{
color: #000000;
}
.login-bar{
display: flex;
align-items: center;
}
.cart-ico{
width: 88px;
height: 28px;
margin-right: 20px;
background: #f7f7f7;
border-radius: 17px;
display: flex;
align-items: center;
justify-content: space-around;
cursor: pointer;
position: relative;
font-size: 14px;
}
.cart-ico b{
width: 16px;
height: 16px;
line-height: 17px;
font-size: 12px;
color: #fff;
text-align: center;
background: #fa6240;
```

```

        border-radius: 50%;
        transform: scale(.8);
        position: absolute;
        left: 16px;
        top: -1px;
    }
    .cart-ico img{
        width: 15px;
        height: auto;
        margin-left: 6px;
    }
    .cart-ico span{
        margin-right: 6px;
    }
    .study{
        padding-left: 0;
        font-family: PingFangSC-Regular;
        letter-spacing: 0;
        margin-right: 20px;
        font-size: 15px;
        cursor: pointer;
    }
    .member img{
        width: 26px;
        height: 26px;
        border-radius: 50%;
        display: inline-block;
        cursor: pointer;
        vertical-align: middle;
    }
    .member img:hover{
        border: 1px solid rgb(255, 194, 16);
    }
    .el-dropdown-menu{
        left: 1130px!important;
        width: 180px;
        top: 64px!important;
    }
    .el-dropdown-menu i{
        float:right;
        line-height: 36px;
    }
    .header-login,.header-register{
        cursor: pointer;
        font-size: 15px;
        color: #4a4a4a;
    }
    .header-login:hover,.header-register:hover{
        color: #000000;
    }
</style>

```

脚步子组件

Footer.vue

```
<template>
  <el-container>
    <el-footer height="128px">
      <el-row>
        <el-col :span="4">关于我们</el-col>
        <el-col :span="4">联系我们</el-col>
        <el-col :span="4">商务合作</el-col>
        <el-col :span="4">帮助中心</el-col>
        <el-col :span="4">意见反馈</el-col>
        <el-col :span="4">新手指南</el-col>
      </el-row>
      <el-row class="copyright">
        <el-col :span="24">Copyright © luffycity.com版权所有 | 京ICP备17072161号-1</el-
col>
      </el-row>
    </el-footer>
  </el-container>
</template>

<script>
  export default {
    name: "Footer",
    data() {
      return {}
    }
  }
</script>

<style scoped>
.el-footer {
  font-size: 14px;
  padding-top: 40px;
  background-color: #25292e;
  color: #fff;
}
.el-row{
  width: 1200px;
  margin: 0 auto;
  padding: 0 260px;
}
.copyright{
  padding-top: 15px;
  text-align: center;
}
</style>
```

轮播图子组件

Banner.vue

```
<template>
  <div class="banner">
    <el-carousel trigger="click" height="506px">
      <el-carousel-item v-for="item in banner_list">
        <a :href="item.link"></a>
      </el-carousel-item>
    </el-carousel>
  </div>
</template>

<script>
  export default {
    name: "Banner",
    data() {
      return {
        banner_list: [
          {"link": "http://http://127.0.0.1:8080/", "image": "/static/banner/1.jpeg"},
          {"link": "http://http://127.0.0.1:8080/", "image": "/static/banner/2.png"},
          {"link": "http://http://127.0.0.1:8080/", "image": "/static/banner/3.png"},
          {"link": "http://http://127.0.0.1:8080/", "image": "/static/banner/4.png"},
          {"link": "http://http://127.0.0.1:8080/", "image": "/static/banner/5.png"},
          {"link": "http://http://127.0.0.1:8080/", "image": "/static/banner/6.png"},
        ]
      };
    },
  },
</script>

<style scoped>
  .banner img {
    width: 100%;
  }
</style>
```

注册首页路由

```
import Vue from "vue"
import Router from "vue-router"

// 导入需要注册路由的组件
import Home from "../components/Home"
Vue.use(Router);

// 配置路由列表
export default new Router({
  mode: "history",
  routes: [
```

```
// 路由列表
{
    name: "Home",
    path: "/home",
    component: Home,
},
{
    name: "Home",
    path: "/",
    component: Home,
}
]
})
```

轮播图功能实现

安装依赖模块和配置

图片处理模块

```
pip install pillow
```

上传文件相关配置

settings.py

```
# 访问静态文件的url地址前缀
STATIC_URL = '/static/'
# 设置django的静态文件目录
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "luffy/static")
]

# 项目中存储上传文件的根目录[暂时配置]，注意，static目录需要手动创建否则上传文件时报错
MEDIA_ROOT=os.path.join(BASE_DIR, "luffy/static")
# 访问上传文件的url地址前缀
MEDIA_URL = "/media/"
```

在xadmin中输出上传文件的Url地址

总路由urls.py新增代码：


```

from django.urls import re_path
from django.conf import settings
from django.views.static import serve

urlpatterns = [
    ...
    re_path(r'media/(?P<path>.*)', serve, {"document_root": settings.MEDIA_ROOT}),
]

```

创建模型

```

from django.db import models

# Create your models here.
class BannerInfo(models.Model):
    """
    轮播图
    """
    # upload_to 存储子目录, 真实存放地址会使用配置中的MEDIA_ROOT+upload_to
    image = models.ImageField(upload_to='banner', verbose_name='轮播图',
null=True, blank=True)
    name = models.CharField(max_length=150, verbose_name='轮播图名称')
    link = models.CharField(max_length=150, verbose_name='轮播图广告地址')
    orders = models.IntegerField(verbose_name='显示顺序')
    is_show=models.BooleanField(verbose_name="是否上架", default=False)
    is_delete=models.BooleanField(verbose_name="逻辑删除", default=False)

    class Meta:
        db_table = 'ly_banner'
        verbose_name = '轮播图'
        verbose_name_plural = verbose_name

    def __str__(self):
        return self.name

```

数据迁移

```

python manage.py makemigrations
python manage.py migrate

```

序列化器

serializers.py

```

from rest_framework.serializers import ModelSerializer
from .models import BannerInfo
class BannerInfoSerializer(ModelSerializer):
    """轮播图序列化器"""
    class Meta:
        model=BannerInfo
        fields = ("image","link")

```

视图代码

views.py

```

from django.db.models import Q
from rest_framework.generics import ListAPIView
from .models import BannerInfo
from .serializers import BannerInfoSerializer
class BannerInfoAPIView(ListAPIView):
    """
    轮播图列表
    """
    queryset = BannerInfo.objects.filter( Q(is_show=True) & Q(is_delete=False)
    ).order_by("-orders")
    serializer_class = BannerInfoSerializer

```

路由代码

urls.py

```

from django.urls import path,re_path
from . import views
urlpatterns = [
    path(r"banner/",views.BannerInfoAPIView.as_view()),
]

```

注册模型到xadmin中

修改后端xadmin中子应用名称

apps.py

```

class HomeConfig(AppConfig):
    name = 'home'
    verbose_name = '我的首页'

```

```
__init__.py
```

```
default_app_config = "home.apps.HomeConfig"
```

在当前子应用中创建adminx.py，添加如下代码

```
# 轮播图
from home.models import bannerInfo
class BannerInfoModelAdmin(object):
    list_display=["name", "orders", "is_show"]
xadmin.site.register(bannerInfo, BannerInfoModelAdmin)
```

客户端代码获取数据

Banner.vue代码：

```
<template>
  <div class="banner">
    <el-carousel trigger="click" height="506px">
      <el-carousel-item v-for="item in banner_list">
        <a :href="item.link"></a>
      </el-carousel-item>
    </el-carousel>
  </div>
</template>

<script>
  export default {
    name: "Banner",
    data() {
      return {
        banner_list: [],
      };
    },
    created: function() {
      // 获取轮播图
      this.$axios.get("http://api.luffycity.cn:8000/home/banner/").then(res => {
        console.log(res.data)
        this.banner_list = res.data
      }).catch(error => {
        console.log(error);
      });
    }
  }
</script>

<style scoped>
  .banner img {
    width: 100%;
  }
</style>
```

</style>

导航功能实现

创建模型

```
from django.db import models

# Create your models here.
class NavInfo(models.Model):
    """
    导航
    """
    NAV_OPT = (
        (0, 'top'),
        (1, 'footer')
    )
    name = models.CharField(max_length=50, verbose_name='导航名称')
    link = models.CharField(max_length=250, verbose_name='导航地址')
    orders = models.IntegerField(verbose_name='显示顺序')
    opt = models.SmallIntegerField(choices=NAV_OPT, default=0, verbose_name='位置')
    is_show=models.BooleanField(verbose_name="是否显示",default=False)
    is_delete=models.BooleanField(verbose_name="逻辑删除",default=False)

    class Meta:
        db_table = 'ly_nav'
        verbose_name = '导航'
        verbose_name_plural = verbose_name

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

数据迁移

```
python manage.py makemigrations
python manage.py migrate
```

序列化器代码

```

from rest_framework.serializers import ModelSerializer
from .models import NavInfo
class NavInfoSerializer(ModelSerializer):
    """导航序列化器"""
    class Meta:
        model=NavInfo
        fields = ("name", "link")

```

视图代码

views.py

```

from .models import NavInfo
from .serializers import NavInfoSerializer
class NavInfoAPIView(ListAPIView):
    """
    导航列表
    """
    queryset = NavInfo.objects.filter( Q(is_show=True) & Q(is_delete=False) & Q(opt=0)
).order_by("-orders")
    serializer_class = NavInfoSerializer

```

路由代码

urls.py

```

from django.urls import path, re_path
from . import views
urlpatterns = [
    ...
    path(r"nav/", views.NavInfoAPIView.as_view()),
]

```

注册模型到xadmin中

在当前子应用adminx.py, 添加如下代码

```

# 导航
from home.models import NavInfo
class NavInfoInfoModelAdmin(object):
    list_display=["name", "link", "is_show"]
xadmin.site.register(NavInfo, NavInfoInfoModelAdmin)

```

客户端代码获取数据

Header.vue代码:

[illegible]

```

        </el-col>
      </el-row>
    </el-header>
  </el-container>
</div>
</template>

<script>
export default {
  name: "Header",
  props: ["current_page"],
  data() {
    return {
      is_login: false, /* 是否登录 */
      nav_list: [],
    },
  },
  created: function() {
    // 获取轮播图
    this.$axios.get("http://api.luffycity.cn:8000/home/nav/").then(res => {
      console.log(res.data)
      this.nav_list = res.data
    }).catch(error => {
      console.log(error);
    });
  }
}
</script>

<style scoped>
....
</style>

```

用户的登陆认证

前端显示登陆页面

登录页组件

Login.vue

```

<template>
  <div class="login box">
    
    <div class="login">
      <div class="login-title">
        
        <p>帮助有志向的年轻人通过努力学习获得体面的工作和生活!</p>

```

```

    </div>
    <div class="login_box">
      <div class="title">
        <span @click="login_type=0">密码登录</span>
        <span @click="login_type=1">短信登录</span>
      </div>
      <div class="inp" v-if="login_type==0">
        <input v-model = "username" type="text" placeholder="用户名 / 手机号" class="user">
        <input v-model = "password" type="password" name="" class="pwd" placeholder="密码">
        <div id="geetest1"></div>
        <div class="rember">
          <p>
            <input type="checkbox" class="no" name="a"/>
            <span>记住密码</span>
          </p>
          <p>忘记密码</p>
        </div>
        <button class="login_btn">登录</button>
        <p class="go_login" >没有账号 <span>立即注册</span></p>
      </div>
      <div class="inp" v-show="login_type==1">
        <input v-model = "username" type="text" placeholder="手机号码" class="user">
        <input v-model = "password" type="text" class="pwd" placeholder="短信验证码">
        <button id="get_code">获取验证码</button>
        <button class="login_btn">登录</button>
        <p class="go_login" >没有账号 <span>立即注册</span></p>
      </div>
    </div>
  </div>
</div>
</template>

<script>
export default {
  name: 'Login',
  data(){
    return {
      login_type: 0,
      username: '',
      password: '',
    }
  },
  methods:{

  },
};
</script>

```



```
<style scoped>
.box{
    width: 100%;
    height: 100%;
    position: relative;
    overflow: hidden;
}
.box img{
    width: 100%;
    min-height: 100%;
}
.box .login {
    position: absolute;
    width: 500px;
    height: 400px;
    top: 0;
    left: 0;
    margin: auto;
    right: 0;
    bottom: 0;
    top: -338px;
}
.login .login-title{
    width: 100%;
    text-align: center;
}
.login-title img{
    width: 190px;
    height: auto;
}
.login-title p{
    font-family: PingFangSC-Regular;
    font-size: 18px;
    color: #fff;
    letter-spacing: .29px;
    padding-top: 10px;
    padding-bottom: 50px;
}
.login_box{
    width: 400px;
    height: auto;
    background: #fff;
    box-shadow: 0 2px 4px 0 rgba(0,0,0,.5);
    border-radius: 4px;
    margin: 0 auto;
    padding-bottom: 40px;
}
.login_box .title{
    font-size: 20px;
    color: #9b9b9b;
    letter-spacing: .32px;
    border-bottom: 1px solid #e6e6e6;
```

```

        display: flex;
        justify-content: space-around;
        padding: 50px 60px 0 60px;
        margin-bottom: 20px;
        cursor: pointer;
    }
    .login_box .title span:nth-of-type(1){
        color: #4a4a4a;
        border-bottom: 2px solid #84cc39;
    }

    .inp{
        width: 350px;
        margin: 0 auto;
    }
    .inp input{
        border: 0;
        outline: 0;
        width: 100%;
        height: 45px;
        border-radius: 4px;
        border: 1px solid #d9d9d9;
        text-indent: 20px;
        font-size: 14px;
        background: #fff !important;
    }
    .inp input.user{
        margin-bottom: 16px;
    }
    .inp .rember{
        display: flex;
        justify-content: space-between;
        align-items: center;
        position: relative;
        margin-top: 10px;
    }
    .inp .rember p:first-of-type{
        font-size: 12px;
        color: #4a4a4a;
        letter-spacing: .19px;
        margin-left: 22px;
        display: -ms-flexbox;
        display: flex;
        -ms-flex-align: center;
        align-items: center;
        /*position: relative;*/
    }
    .inp .rember p:nth-of-type(2){
        font-size: 14px;
        color: #9b9b9b;
        letter-spacing: .19px;
        cursor: pointer;
    }

```

```

.inp .rember input{
  outline: 0;
  width: 30px;
  height: 45px;
  border-radius: 4px;
  border: 1px solid #d9d9d9;
  text-indent: 20px;
  font-size: 14px;
  background: #fff !important;
}

.inp .rember p span{
  display: inline-block;
  font-size: 12px;
  width: 100px;
  /*position: absolute;*/
  /*left: 20px;*/
}

#geetest{
  margin-top: 20px;
}

.login_btn{
  width: 100%;
  height: 45px;
  background: #84cc39;
  border-radius: 5px;
  font-size: 16px;
  color: #fff;
  letter-spacing: .26px;
  margin-top: 30px;
}

.inp .go_login{
  text-align: center;
  font-size: 14px;
  color: #9b9b9b;
  letter-spacing: .26px;
  padding-top: 20px;
}

.inp .go_login span{
  color: #84cc39;
  cursor: pointer;
}
</style>

```

绑定登陆页面路由地址

main.js

```

import Vue from "vue"
import Router from "vue-router"

```

```
// 导入需要注册路由的组件
import Home from "../components/Home"
import Login from "../components/Login"
Vue.use(Router);

// 配置路由列表
export default new Router({
  mode:"history",
  routes:[
    // 路由列表
    ...
    {
      name:"Login",
      path: "/login",
      component:Login,
    }
  ]
})
```

调整首页头部子组件中登陆按钮的链接信息

Header.vue

```
<router-link to="/login">登录</router-link>
```

后端实现登陆认证

Django默认已经提供了认证系统。认证系统包含：

- 用户管理
- 权限
- 用户组
- 密码哈希系统
- 用户登录或内容显示的表单和视图
- 一个可插拔的后台系统

Django默认用户的认证机制依赖Session机制，我们在项目中将引入JWT认证机制，将用户的身份凭据存放在Token中，然后对接Django的认证系统，帮助我们来实现：

- 用户的数据模型
- 用户密码的加密与验证
- 用户的权限系统

Django用户模型类

Django认证系统中提供了用户模型类User保存用户的数据，默认的用户包含以下常见的基本字段：

字段名	字段描述
<code>username</code>	必选。150个字符以内。用户名可能包含字母数字, <code>_</code> , <code>@</code> , <code>+</code> , <code>.</code> 和 <code>-</code> 个字符。
<code>first_name</code>	可选 (<code>blank=True</code>)。少于等于30个字符。
<code>last_name</code>	可选 (<code>blank=True</code>)。少于等于30个字符。
<code>email</code>	可选 (<code>blank=True</code>)。邮箱地址。
<code>password</code>	必选。密码的哈希加密串。(Django 不保存原始密码)。原始密码可以无限长而且可以包含任意字符。
<code>groups</code>	与 <code>Group</code> 之间的多对多关系。
<code>user_permissions</code>	与 <code>Permission</code> 之间的多对多关系。
<code>is_staff</code>	布尔值。设置用户是否可以访问Admin 站点。
<code>is_active</code>	布尔值。指示用户的账号是否激活。它不是用来控制用户是否能够登录, 而是描述一种帐号的使用状态。
<code>is_superuser</code>	是否是超级用户。超级用户具有所有权限。
<code>last_login</code>	用户最后一次登录的时间。
<code>date_joined</code>	账户创建的时间。当账号创建时, 默认设置为当前的date/time。

常用方法:

- `set_password (raw_password)`
设置用户的密码为给定的原始字符串, 并负责密码的。不会保存 `User` 对象。当 `None` 为 `raw_password` 时, 密码将设置为一个不可用的密码。
- `check_password (raw_password)`
如果给定的`raw_password`是用户的真实密码, 则返回`True`, 可以在校验用户密码时使用。

管理器方法:

管理器方法即可以通过 `User.objects.` 进行调用的方法。

- `create_user (username, email=None, password=None, **extra_fields)`
创建、保存并返回一个 `User` 对象。
- `create_superuser (username, email, password, **extra_fields)`
与 `create_user()` 相同, 但是设置 `is_staff` 和 `is_superuser` 为 `True`。

创建用户模块的子应用

```
python ../../manage.py startapp users
```

在settings.py文件中注册子应用。

```
INSTALLED_APPS = [  
    ...  
    'users',  
]
```

创建自定义的用户模型类

Django认证系统中提供的用户模型类及方法很方便，我们可以使用这个模型类，但是字段有些无法满足项目需求，如本项目中需要保存用户的手机号，需要给模型类添加额外的字段。

Django提供了 `django.contrib.auth.models.AbstractUser` 用户抽象模型类允许我们继承，扩展字段来使用Django认证系统的用户模型类。

我们可以在apps中创建Django应用users，并在配置文件中注册users应用。

在创建好的应用models.py中定义用户的用户模型类。

```
class User(AbstractUser):  
    """用户模型类"""  
    mobile = models.CharField(max_length=11, unique=True, verbose_name='手机号')  
  
    class Meta:  
        db_table = 'ly_users'  
        verbose_name = '用户'  
        verbose_name_plural = verbose_name
```

我们自定义的用户模型类还不能直接被Django的认证系统所识别，需要在配置文件中告知Django认证系统使用我们自定义的模型类。

在配置文件中设置

```
AUTH_USER_MODEL = 'users.User'
```

`AUTH_USER_MODEL` 参数的设置以 `点.` 来分隔，表示 `应用名.模型类名`。

注意：Django建议我们对于AUTH_USER_MODEL参数的设置一定要在第一次数据库迁移之前就设置好，否则后续使用可能出现未知错误。

执行数据库迁移

```
python manage.py makemigrations  
python manage.py migrate
```

Django REST framework JWT

在用户注册或登录后，我们想记录用户的登录状态，或者为用户创建身份认证的凭证。我们不再使用Session认证机制，而使用Json Web Token认证机制。

Json web token (JWT)，是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准（RFC 7519）。该token被设计为紧凑且安全的，特别适用于分布式站点的单点登录（SSO）场景。JWT的声明一般被用来在身份提供者和服务器提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源，也可以增加一些额外的其它业务逻辑所必须的声明信息，该token也可直接被用于认证，也可被加密。

JWT的构成

JWT就一段字符串，由三段信息构成的，将这三段信息文本用 `.` 链接一起就构成了Jwt字符串。就像这样：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoiYWRtaw4iOnRydWV9.TjVA95OrM7E2cBab30RMHrHDCEfXjoYZgeFONFh7HgQ
```

第一部分我们称它为头部 (header),第二部分我们称其为载荷 (payload, 类似于飞机上承载的物品)，第三部分是签名 (signature)。

header

jwt的头部承载两部分信息：

- 声明类型，这里是jwt
- 声明加密的算法 通常直接使用 HMAC SHA256

完整的头部就像下面这样的JSON：

```
{
  'typ': 'JWT',
  'alg': 'HS256'
}
```

然后将头部进行base64加密（该加密是可以对称解密的),构成了第一部分。

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

payload

载荷就是存放有效信息的地方。这个名字像是特指飞机上承载的货品，这些有效信息包含三个部分

- 标准中注册的声明
- 公共的声明
- 私有的声明

标准中注册的声明 (建议但不强制使用)：

- **iss**: jwt签发者
- **sub**: jwt所面向的用户
- **aud**: 接收jwt的一方
- **exp**: jwt的过期时间，这个过期时间必须要大于签发时间
- **nbf**: 定义在什么时间之前，该jwt都是不可用的。
- **iat**: jwt的签发时间
- **jti**: jwt的唯一身份标识，主要用来作为一次性token,从而回避重放攻击。

公共的声明：公共的声明可以添加任何的信息，一般添加用户的相关信息或其他业务需要的必要信息.但不建议添加敏感信息，因为该部分在客户端可解密。

私有的声明：私有声明是提供者和消费者所共同定义的声明，一般不建议存放敏感信息，因为base64是对称解密的，意味着该部分信息可以归类为明文信息。

定义一个payload:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

然后将其进行base64加密，得到JWT的第二部分。

```
eyJzdWIiOiIxMjMONTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjYWRtaW4iOnRydWV9
```

signature

JWT的第三部分是一个签证信息，这个签证信息由三部分组成：

- header (base64后的)
- payload (base64后的)
- secret

这个部分需要base64加密后的header和base64加密后的payload使用 `.` 连接组成的字符串，然后通过header中声明的加密方式进行加盐 `secret` 组合加密，然后就构成了jwt的第三部分。

```
// javascript
var encodedString = base64UrlEncode(header) + '.' + base64UrlEncode(payload);

var signature = HMACSHA256(encodedString, 'secret'); //
TJVA95OrM7E2cBab30RMHrHDCEfXjoYZgeFONfh7HgQ
```

将这三部分用 `.` 连接成一个完整的字符串,构成了最终的jwt:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpvcXVCJ9.eyJzdWIiOiIxMjMONTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDCEfXjoYZgeFONfh7HgQ
```


注意：secret是保存在服务器端的，jwt的签发生成也是在服务器端的，secret就是用来进行jwt的签发和jwt的验证，所以，它就是你服务端的私钥，在任何场景都不应该流露出去。一旦客户端得知这个secret, 那就意味着客户端是可以自我签发jwt了。

关于签发和核验JWT，我们可以使用Django REST framework JWT扩展来完成。

文档网站<http://getblimp.github.io/django-rest-framework-jwt/>

安装配置JWT

安装

```
pip install djangorestframework-jwt
```

配置

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication',
    ),
}
import datetime
JWT_AUTH = {
    'JWT_EXPIRATION_DELTA': datetime.timedelta(days=1),
}
```

- JWT_EXPIRATION_DELTA 指明token的有效期

生成jwt

Django REST framework JWT 扩展的说明文档中提供了手动签发JWT的方法

```
from rest_framework_jwt.settings import api_settings

jwt_payload_handler = api_settings.JWT_PAYLOAD_HANDLER
jwt_encode_handler = api_settings.JWT_ENCODE_HANDLER

payload = jwt_payload_handler(user)
token = jwt_encode_handler(payload)
```

在用户注册或登录成功后，在序列化器中返回用户信息以后同时返回token即可。

后端实现登陆认证接口

Django REST framework JWT提供了登录获取token的视图，可以直接使用

在子应用路由urls.py中

```
from rest_framework_jwt.views import obtain_jwt_token

urlpatterns = [
    path(r'authorizations/', obtain_jwt_token, name='authorizations'),
]
```

在主路由中，引入当前子应用的路由文件

```
urlpatterns = [
    ...
    path('users/', include("users.urls")),
    # include 的值必须是 模块名.urls 格式,字符串中间只能出现一个圆点
]
```

接下来，我们可以通过postman来测试下功能



前端实现登陆功能

在登陆组件中找到登陆按钮，绑定点击事件

```
<button class="login_btn" @click="loginhandler">登录</button>
```

在methods中请求后端

```
export default {
  name: 'Login',
  data(){
    return {
```

```

        login_type: 0,
        remember: false, // 记住密码
        username: "",
        password: "",
    }
},

methods: {
    // 登录
    loginhandler() {
        this.$axios.post("http://127.0.0.1:8000/users/authorizations/",
        {"username": this.username, "password": this.password}).then(response => {
            console.log(response.data)
        }).catch(error => {
            console.log(error)
        })
    }
},
};

```

2.3.7 前端保存jwt

我们可以将JWT保存在cookie中，也可以保存在浏览器的本地存储里，我们保存在浏览器本地存储中浏览器的本地存储提供了sessionStorage 和 localStorage 两种：

- **sessionStorage** 浏览器关闭即失效
- **localStorage** 长期有效

使用方法

```

sessionStorage.变量名 = 变量值    // 保存数据
sessionStorage.变量名    // 读取数据
sessionStorage.clear()    // 清除所有sessionStorage保存的数据

localStorage.变量名 = 变量值    // 保存数据
localStorage.变量名    // 读取数据
localStorage.clear()    // 清除所有localStorage保存的数据

```

登陆组件代码Login.vue

```

    // 使用浏览器本地存储保存token
    if (this.remember) {
        // 记住登录
        sessionStorage.clear();
        localStorage.token = response.data.token;
    } else {
        // 未记住登录
        localStorage.clear();
        sessionStorage.token = response.data.token;
    }
    // 页面跳转回到上一个页面 也可以使用 this.$router.push("/") 回到首页
    this.$router.go(-1)

```

默认的返回值仅有token，我们还需在返回值中增加username和id，方便在客户端页面中显示当前登陆用户
通过修改该视图的返回值可以完成我们的需求。

在users/utils.py 中，创建

```

def jwt_response_payload_handler(token, user=None, request=None):
    """
    自定义jwt认证成功返回数据
    """
    return {
        'token': token,
        'id': user.id,
        'username': user.username
    }

```

修改settings.py配置文件

```

# JWT
JWT_AUTH = {
    'JWT_EXPIRATION_DELTA': datetime.timedelta(days=1),
    'JWT_RESPONSE_PAYLOAD_HANDLER': 'users.utils.jwt_response_payload_handler',
}

```

登陆组件代码Login.vue

```

    // 使用浏览器本地存储保存token
    if (this.remember) {
        // 记住登录
        sessionStorage.clear();
        localStorage.token = response.data.token;
        localStorage.id = response.data.id;
        localStorage.username = response.data.username;
    } else {
        // 未记住登录
        localStorage.clear();
        sessionStorage.token = response.data.token;
    }

```

```
sessionStorage.id = response.data.id;
sessionStorage.username = response.data.username;
}
```

2.3.9 多条件登录

JWT扩展的登录视图，在收到用户名与密码时，也是调用Django的认证系统中提供的`authenticate()`来检查用户名与密码是否正确。

我们可以通过修改Django认证系统的认证后端（主要是`authenticate`方法）来支持登录账号既可以是用户名也可以是手机号。

修改Django认证系统的认证后端需要继承`django.contrib.auth.backends.ModelBackend`，并重写`authenticate`方法。

`authenticate(self, request, username=None, password=None, **kwargs)` 方法的参数说明：

- request 本次认证的请求对象
- username 本次认证提供的用户账号
- password 本次认证提供的密码

我们想要让用户既可以以用户名登录，也可以以手机号登录，那么对于`authenticate`方法而言，`username`参数即表示用户名或者手机号。

重写`authenticate`方法的思路：

1. 根据`username`参数查找用户`User`对象，`username`参数可能是用户名，也可能是手机号
2. 若查找到`User`对象，调用`User`对象的`check_password`方法检查密码是否正确

在`users/utils.py`中编写：

```
def get_user_by_account(account):
    """
    根据帐号获取user对象
    :param account: 账号，可以是用户名，也可以是手机号
    :return: User对象 或者 None
    """
    try:
        if re.match('^1[3-9]\d{9}$', account):
            # 帐号为手机号
            user = User.objects.get(mobile=account)
        else:
            # 帐号为用户名
            user = User.objects.get(username=account)
    except User.DoesNotExist:
        return None
    else:
        return user

import re
from .models import User
from django.contrib.auth.backends import ModelBackend
```

```
class UsernameMobileAuthBackend(ModelBackend):
    """
    自定义用户名或手机号认证
    """

    def authenticate(self, request, username=None, password=None, **kwargs):
        user = get_user_by_account(username)
        if user is not None and user.check_password(password):
            return user
```

在配置文件settings.py中告知Django使用我们自定义的认证后端

```
AUTHENTICATION_BACKENDS = [
    'users.utils.UsernameMobileAuthBackend',
]
```

前端首页实现登陆状态的判断

Home组件代码:

```
<template>
  <div class="home">
    <Header :is_login="is_login" @logout="logout" :current_page="current_page"/>
    <Banner/>
    <Footer/>
  </div>
</template>

<script>
import Header from "../common/Header"
import Banner from "../common/Banner"
import Footer from "../common/Footer"
export default {
  name: "Home",
  data() {
    return {
      id: sessionStorage.id || localStorage.id,
      username: sessionStorage.username || localStorage.username,
      token: sessionStorage.token || localStorage.token,
      current_page: 0,
      is_login: false,
    };
  },
  components: {
    Header,
    Banner,
    Footer,
  },
  mounted() {
    if (this.id && this.token) {
```

```

        this.is_login=true;
    }
},
methods:{
    logout(){
        this.is_login=false;
    }
}
}
}
</script>

```

Header子组件显示登陆状态

Header.vue

```

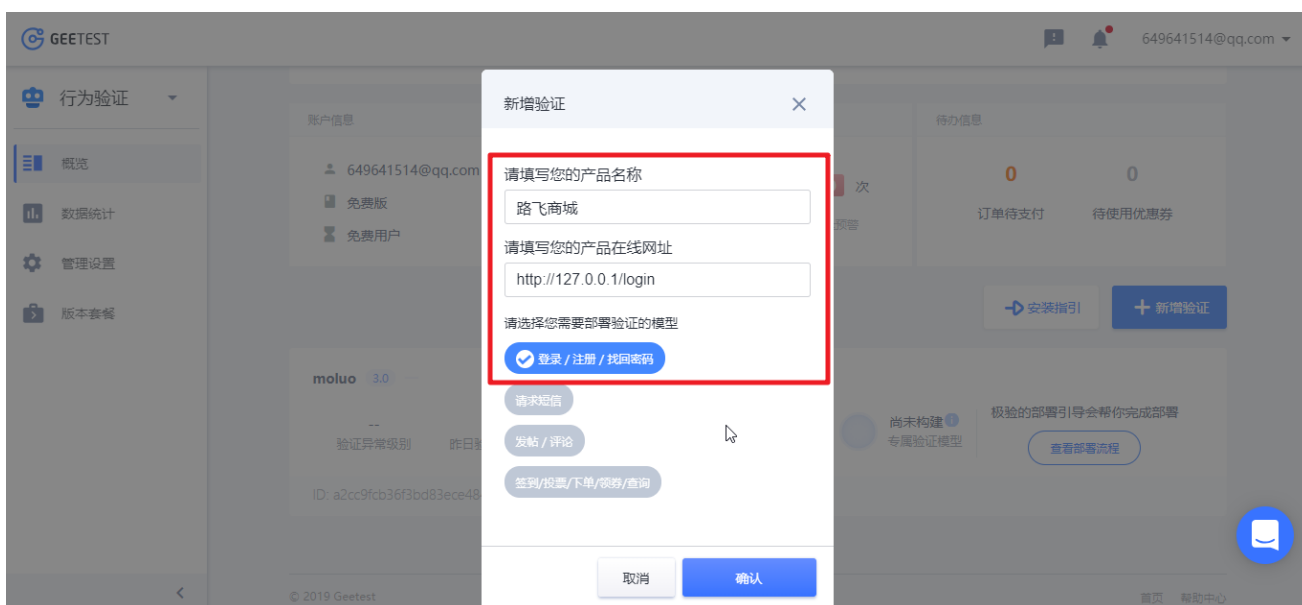
<script>
export default {
    name:"Header",
    props:["current_page","is_login"],
    data(){
        return {
            nav_list: [],
        }
    },
    created: function(){
        // 获取轮播图
        this.$axios.get("http://api.luffycity.cn:8000/home/nav/").then(res => {
            this.nav_list = res.data
        }).catch(error => {
            console.log(error);
        });
    },
    methods:{
        logout(){
            localStorage.clear();
            sessionStorage.clear();
            this.$emit("logout",{"is_login":false})
        }
    }
}
</script>

```

在登录认证中接入极验验证

官网: https://www.geetest.com/first_page/

注册登录以后, 即进入登录后台, 选择行为验证。



接下来，就可以根据官方文档，把验证码集成到项目中了

文档地址: <https://docs.geetest.com/install/overview/start/>

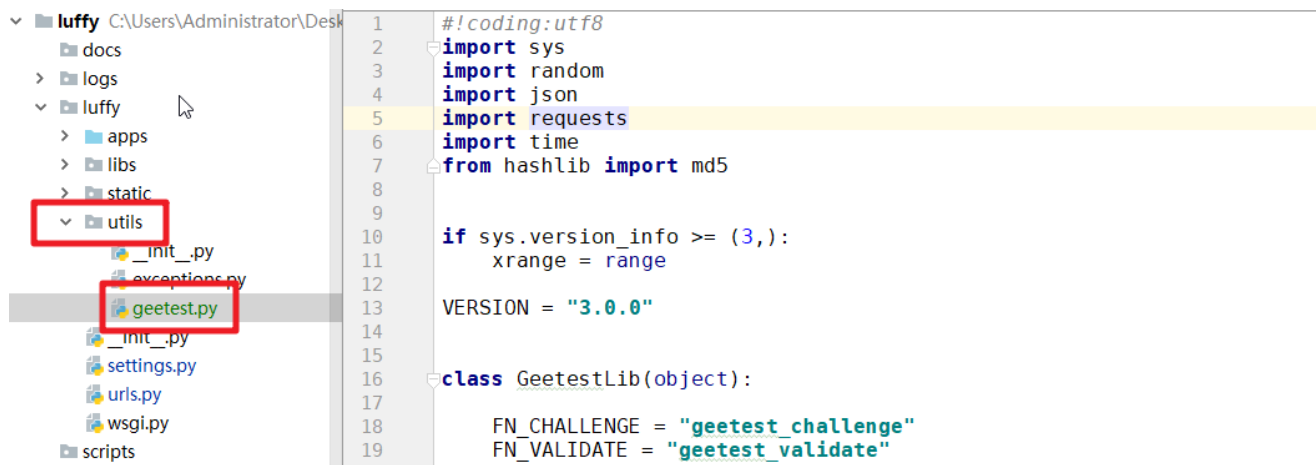
下载和安装验证码模块包。

```
git clone https://github.com/GeeTeam/gt3-python-sdk.git
```

安装以来模块

```
pip install requests
```

把验证码模块防止在utils中



并在users子应用下创建验证码视图类，并提供验证码和校验验证码的视图方法。

```
from luffy.utils.geetest import GeetestLib
from rest_framework.views import APIView
from rest_framework.response import Response
# Create your views here.

class VerifyCode(APIView):
    def get(self, request):
        user_id = 'test'
        pc_geetest_id = "a9feefab99c8d4bbcf0d9e3021048312"
        pc_geetest_key = "aa467ab83be3c44929bc7da76eb88028"
        gt = GeetestLib(pc_geetest_id, pc_geetest_key)
        status = gt.pre_process(user_id, JSON_FORMAT=0, ip_address="127.0.0.1")
        if not status:
            status = 2
        request.session[gt.GT_STATUS_SESSION_KEY] = status
        request.session["user_id"] = user_id
        response_str = gt.get_response_str()
        print(response_str)
        return Response(response_str)
```

```

def post(self,request):
    pc_geetest_id = "a9feefab99c8d4bbcf0d9e3021048312"
    pc_geetest_key = "aa467ab83be3c44929bc7da76eb88028"
    gt = GeetestLib(pc_geetest_id, pc_geetest_key)

    challenge = request.data.get(gt.FN_CHALLENGE, "")
    validate = request.data.get(gt.FN_VALIDATE, "")
    seccode = request.data.get(gt.FN_SECCODE, "")
    status = request.session[gt.GT_STATUS_SESSION_KEY]
    user_id = request.session["user_id"]
    if status == 1:
        result = gt.success_validate(challenge, validate, seccode, user_id,
JSON_FORMAT=0)
    else:
        result = gt.failback_validate(challenge, validate, seccode)
        request.session["user_id"] = user_id
    return Response({"message":result})

```

路由注册:

```
path(r'code/', views.VerifyCode.as_view()),
```

前端获取显示并校验验证码

把下载会哦图的验证码模块包中的gt.js放置到前端项目中, 并在main.js中引入

```

// 导入gt极验
import '../static/globals/gt.js'

```

Login.vue代码

```

<template>
  <div class="login box">
    
    <div class="login">
      <div class="login-title">
        
        <p>帮助有志向的年轻人通过努力学习获得体面的工作和生活!</p>
      </div>
      <div class="login_box">
        <div class="title">
          <span @click="login_type=0">密码登录</span>
          <span @click="login_type=1">短信登录</span>
        </div>
        <div class="inp" v-if="login_type==0">
          <input v-model = "username" type="text" placeholder="用户名 / 手机号" class="user">

```

```

      <input v-model = "password" type="password" name="" class="pwd"
placeholder="密码">
      <div id="geetest"></div>
      <div class="rember">
        <p>
          <input type="checkbox" class="no" name="a"/>
          <span>记住密码</span>
        </p>
        <p>忘记密码</p>
      </div>
      <button class="login_btn" @click="loginhander">登录</button>
      <p class="go_login" >没有账号 <span>立即注册</span></p>
    </div>
    <div class="inp" v-show="login_type==1">
      <input v-model = "username" type="text" placeholder="手机号码"
class="user">
      <input v-model = "password" type="text" class="pwd"
placeholder="短信验证码">
      <button id="get_code">获取验证码</button>
      <button class="login_btn">登录</button>
      <p class="go_login" >没有账号 <span>立即注册</span></p>
    </div>
  </div>
</div>
</template>

<script>
export default {
  name: 'Login',
  data(){
    return {
      login_type: 0,
      username:"",
      password:"",
      validateResult:false,
    }
  },
  created(){
    this.$axios.get("http://127.0.0.1:8000/users/code",{responseType:
'json',}).then(response=>{
      let data = response.data;
      console.log(data);
      var _this = this;
      //请检测data的数据结构, 保证data.gt, data.challenge, data.success有值
      initGeetest({
        // 以下配置参数来自服务端 SDK
        gt: data.gt,
        challenge: data.challenge,
        offline: !data.success,
        new_captcha: true,
        product: 'popup',
        width: '100%'

```

```

    }, (captchaObj)=>{
        var result = captchaObj.getValidate();

        // 这里可以调用验证实例 captchaObj 的实例方法
        captchaObj.appendTo("#geetest"); //将验证按钮插入到宿主页面中captchaBox元素
    内

        captchaObj.onReady(()=>{
            //your code
        }).onSuccess(()=>{
            var result = captchaObj.getValidate();
            this.validateResult = result;
        }).onError(()=>{
            _this.validateResult=false;
        })
    })
})
},
methods:{
    // 登录
    loginhandler(){
        // 提交数据前判断用户是否通过了验证码校验
        if(!this.validateResult){
            return false;
        }
        this.$axios.post("http://127.0.0.1:8000/users/authorizations/",
{"username":this.username,"password":this.password}).then(response=>{
            // 使用浏览器本地存储保存token
            if (this.remember) {
                // 记住登录
                sessionStorage.clear();
                localStorage.token = response.data.token;
                localStorage.id = response.data.id;
                localStorage.username = response.data.username;
            } else {
                // 未记住登录
                localStorage.clear();
                sessionStorage.token = response.data.token;
                sessionStorage.id = response.data.id;
                sessionStorage.username = response.data.username;
            }
            this.$router.push("/")
        }).catch(error=>{
            console.log(error)
        })
    }
},
};
</script>

```

用户的注册认证

前端显示注册页面并调整首页头部和登陆页面的注册按钮的链接。

注册页面Register,主要是通过登录页面进行改成而成./

```
<template>
  <div class="box">
    
    <div class="register">
      <div class="register_box">
        <div class="register-title">注册路飞学城</div>
        <div class="inp">
          <input v-model = "mobile" type="text" placeholder="手机号码"
class="user">
          <div id="geetest"></div>
          <input v-model = "sms" type="text" placeholder="输入验证码"
class="user">
          <button class="register_btn" >注册</button>
          <p class="go_login" >已有账号 <router-link to="/login">直接登录
</router-link></p>
        </div>
      </div>
    </div>
  </div>
</template>

<script>
export default {
  name: 'Register',
  data(){
    return {
      sms:"",
      mobile:"",
      validateResult:false,
    }
  },
  created(){
  },
  methods:{},
};
</script>

<style scoped>
.box{
  width: 100%;
  height: 100%;
  position: relative;
  overflow: hidden;
}
.box img{
```

```
    width: 100%;
    min-height: 100%;
}
.box .register {
    position: absolute;
    width: 500px;
    height: 400px;
    top: 0;
    left: 0;
    margin: auto;
    right: 0;
    bottom: 0;
    top: -338px;
}
.register .register-title{
    width: 100%;
    font-size: 24px;
    text-align: center;
    padding-top: 30px;
    padding-bottom: 30px;
    color: #4a4a4a;
    letter-spacing: .39px;
}
.register-title img{
    width: 190px;
    height: auto;
}
.register-title p{
    font-family: PingFangSC-Regular;
    font-size: 18px;
    color: #fff;
    letter-spacing: .29px;
    padding-top: 10px;
    padding-bottom: 50px;
}
.register_box{
    width: 400px;
    height: auto;
    background: #fff;
    box-shadow: 0 2px 4px 0 rgba(0,0,0,.5);
    border-radius: 4px;
    margin: 0 auto;
    padding-bottom: 40px;
}
.register_box .title{
    font-size: 20px;
    color: #9b9b9b;
    letter-spacing: .32px;
    border-bottom: 1px solid #e6e6e6;
    display: flex;
    justify-content: space-around;
    padding: 50px 60px 0 60px;
    margin-bottom: 20px;
```

```
        cursor: pointer;
    }
    .register_box .title span:nth-of-type(1){
        color: #4a4a4a;
        border-bottom: 2px solid #84cc39;
    }

    .inp{
        width: 350px;
        margin: 0 auto;
    }
    .inp input{
        border: 0;
        outline: 0;
        width: 100%;
        height: 45px;
        border-radius: 4px;
        border: 1px solid #d9d9d9;
        text-indent: 20px;
        font-size: 14px;
        background: #fff !important;
    }
    .inp input.user{
        margin-bottom: 16px;
    }
    .inp .rember{
        display: flex;
        justify-content: space-between;
        align-items: center;
        position: relative;
        margin-top: 10px;
    }
    .inp .rember p:first-of-type{
        font-size: 12px;
        color: #4a4a4a;
        letter-spacing: .19px;
        margin-left: 22px;
        display: -ms-flexbox;
        display: flex;
        -ms-flex-align: center;
        align-items: center;
        /*position: relative;*/
    }
    .inp .rember p:nth-of-type(2){
        font-size: 14px;
        color: #9b9b9b;
        letter-spacing: .19px;
        cursor: pointer;
    }

    .inp .rember input{
        outline: 0;
        width: 30px;
```

```

    height: 45px;
    border-radius: 4px;
    border: 1px solid #d9d9d9;
    text-indent: 20px;
    font-size: 14px;
    background: #fff !important;
}

.inp .remember p span{
    display: inline-block;
    font-size: 12px;
    width: 100px;
    /*position: absolute;*/
    /*left: 20px;*/
}

#geetest{
    margin-top: 20px;
}

.register_btn{
    width: 100%;
    height: 45px;
    background: #84cc39;
    border-radius: 5px;
    font-size: 16px;
    color: #fff;
    letter-spacing: .26px;
    margin-top: 30px;
}

.inp .go_login{
    text-align: center;
    font-size: 14px;
    color: #9b9b9b;
    letter-spacing: .26px;
    padding-top: 20px;
}

.inp .go_login span{
    color: #84cc39;
    cursor: pointer;
}
</style>

```

前端注册路由:

```

import Register from "../components/Register"

// 配置路由列表
export default new Router({
  mode:"history",
  routes:[
    // 路由列表
    ...

```



```

    {
        name: "Register",
        path: "/register",
        component: Register,
    }
]
})

```

修改首页头部的连接:

```

# Header.vue
<span class="header-register"><router-link to="/register">注册</router-link></span>
#Login.vue
<p class="go_login" >没有账号 <router-link to="/register">立即注册</router-link></p>

```

注册功能的实现

接下来，我们把注册过程中一些注册信息（例如：短信验证码）和session缓存到redis数据库中。

安装django-redis。

```
pip install django-redis
```

在settings.py配置中添加一下代码：

```

# 设置redis缓存
CACHES = {
    # 默认缓存
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        # 项目上线时,需要调整这里的路径
        "LOCATION": "redis://127.0.0.1:6379/0",

        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
    # 提供给xadmin或者admin的session存储
    "session": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
        }
    },
}

```

```
# 提供存储短信验证码
"sms_code": {
    "BACKEND": "django_redis.cache.RedisCache",
    "LOCATION": "redis://127.0.0.1:6379/2",
    "OPTIONS": {
        "CLIENT_CLASS": "django_redis.client.DefaultClient",
    }
}

# 设置xadmin用户登录时,登录信息session保存到redis
SESSION_ENGINE = "django.contrib.sessions.backends.cache"
SESSION_CACHE_ALIAS = "session"
```

关于django-redis 的使用, 说明文档可见http://django-redis-chs.readthedocs.io/zh_CN/latest/

django-redis提供了get_redis_connection的方法, 通过调用get_redis_connection方法传递redis的配置名称可获取到redis的连接对象, 通过redis连接对象可以执行redis命令

<https://redis-py.readthedocs.io/en/latest/>

使用范例:

```
from django_redis import get_redis_connection
// 链接redis数据库
redis_conn = get_redis_connection("default")
```

使用云通讯发送短信

在登录后的平台上获取一下信息:

```
ACCOUNT SID: 8aaf0708697b6beb01699f4442911776
AUTH TOKEN : b4dea244f43a4e0f90e557f0a99c70fa
AppID(默认): 8aaf0708697b6beb01699f4442e3177c
Rest URL(生产): app.cloopen.com:8883 [项目上线时使用真实短信发送服务器]
Rest URL(开发): sandboxapp.cloopen.com:8883 [项目开发时使用沙箱短信发送服务器]
```

找到sdkdemo进行下载

容联·云通讯

首页

产品

价格

案例

下载

资讯

文档帮助

中文 / En

控制台

退出



短信Demo下载



IMPlus SDK下载



企业容信下载



语音Demo下载

在开发过程中,为了节约发送短信的成本,可以把自己的或者同事的手机加入到测试号码中.



后端生成短信验证码

```
from .yuntongxun.sms import CCP
class SMSCodeAPIView(APIView):
    """
    短信验证码
    """
    def get(self, request, mobile):
        """
        短信验证码
        """
        # 生成短信验证码
        sms_code = "%06d" % random.randint(0, 999999)

        # 保存短信验证码与发送记录
        redis_conn = get_redis_connection('verify_codes')
        # 使用redis提供的管道操作可以一次性执行多条redis命令
        pl = redis_conn.pipeline()
        pl.multi()
        pl.setex("sms_%s" % mobile, 300, sms_code) # 设置短信有效期为300s
        pl.setex("sms_time_%s" % mobile, 60, 1) # 设置发送短信间隔为60s
        pl.execute()

        # 发送短信验证码
        ccp = CCP()
        ccp.send_template_sms(mobile, [code, "5"], 1)

        return Response({"message": "OK"}, status.HTTP_200_OK)
```

后端保存用户注册信息

创建序列化器对象[暂时不涉及到手机验证码功能]

```
from rest_framework import serializers
from .models import User
import re

class UserModelSerializer(serializers.ModelSerializer):
    """用户信息序列化器"""
    sms_code = serializers.CharField(label='手机验证码', required=True, allow_null=False,
allow_blank=False, write_only=True)
    password2 = serializers.CharField(label='确认密码', required=True, allow_null=False,
allow_blank=False, write_only=True)

    class Meta:
        model=User
        fields = ('sms_code', 'mobile', 'password', 'password2')
        extra_kwargs={
            "password":{
                "write_only":True
            }
        }

    def validate_mobile(self, value):
        """验证手机号"""
        if not re.match(r'^1[345789]\d{9}$', value):
            raise serializers.ValidationError('手机号格式错误')

        # 验证手机号是否已经被注册了
        # try:
        #     user = User.objects.get(mobile=value)
        # except:
        #     user = None
        #
        # if user:
        #     raise serializers.ValidationError('当前手机号已经被注册')

        # 上面验证手机号是否存在的代码[优化版]
        try:
            User.objects.get(mobile=value)
            # 如果有获取到用户信息,则下面的代码不会被执行,如果没有获取到用户信息,则表示手机号没有注册
            # 过,可以直接pass
            raise serializers.ValidationError('当前手机号已经被注册')
        except:
            pass

        return value

    def validate(self, data):
        # 验证密码
        password = data.get("password")
        password2 = data.get("password2")
        if len(password)<6:
            raise serializers.ValidationError('密码太短不安全~')
```

```

        if password !=password2:
            raise serializers.ValidationError('密码和确认必须一致~')

        return data

def create(self, validated_data):
    # 删除一些不需要保存到数据库里面的字段
    del validated_data['password2']
    del validated_data['sms_code']

    # 因为数据库中默认用户名是唯一的,所以我们把用户手机号码作为用户名
    validated_data["username"] = validated_data["mobile"]

    # 继续调用ModelSerializer内置的添加数据功能
    user = super().create(validated_data)

    # 针对密码要加密
    user.set_password(user.password)
    # 修改密码等用于更新了密码,所以需要保存
    user.save()

    return user

```

视图代码:

```

# users/views.py
from rest_framework.generics import CreateAPIView
from .models import User
from .serializers import UserModelSerializer
class UserAPIView(CreateAPIView):
    """用户管理"""
    queryset = User.objects.all()
    serializer_class = UserModelSerializer

```

设置路由

```

# 子应用路由 urls.py
urlpatterns=[
    ...
    path(r"user", views.UserAPIView.as_view()),
]

```

客户端发送注册信息和发送短信

```

<template>
  <div class="box">
    
    <div class="register">

```

```

    <div class="register_box">
    <div class="register-title">注册路飞学城</div>
      <div class="inp">
        <!--<el-select v-model="region">-->
          <!--<el-option v-for="item in region_list"
:label="item.nation_name+'('+item.nation_code+')" :value="item.nation_code"></el-
option>-->
          <!--</el-select>-->
            <input v-model = "mobile" type="text" placeholder="手机号码"
class="user">
            <input v-model = "password" type="password" placeholder="密码"
class="user">
            <input v-model = "password2" type="password" placeholder="确认密码"
class="user">
            <div id="geetest"></div>
              <div class="sms">
                <input v-model="sms_code" maxlength="16" type="text" placeholder="输入验证码"
class="user">
                <span class="get_sms" @click="send_sms">{{get_sms_text}}</span>
              </div>
              <button class="register_btn" @click="registerHandler">注册</button>
              <p class="go_login" >已有账号 <router-link to="/login">直接登录
</router-link></p>
            </div>
          </div>
        </div>
      </div>
    </div>
  </template>

<script>
export default {
  name: 'Register',
  data(){
    return {
      region:"+86",
      sms_code:"",
      password:"",
      password2:"",
      mobile:"",
      validateResult:false,
      get_sms_text:"获取验证码",
    }
  },
  created(){
    // 页面初始化的时候设置号码的地区号
    // this.region_list = this.$nation;

    // 显示图片验证码
    this.$axios.get("http://127.0.0.1:8000/users/verify").then(response=>{
      // 请求成功
      let data = response.data;
      // 使用initGeetest接口
      // 参数1: 配置参数

```

```

// 参数2: 回调, 回调的第一个参数验证码对象, 之后可以使用它做appendTo之类的事件
console.log(response.data);
data = JSON.parse(data);
initGeetest({
  gt: data.gt,
  challenge: data.challenge,
  width: "350px",
  product: "embed", // 产品形式, 包括: float, embed, popup。注意只对PC版验证码有效
  offline: !data.success // 表示用户后台检测极验服务器是否宕机, 一般不需要关注
  // 更多配置参数请参见: http://www.geetest.com/install/sections/idx-client-
sdk.html#config
}, this.handlerPopup);
}).catch(error=>{
  console.log(error)
})

},
methods:{
  send_sms(){
    let reg = /^[1-9]{2}\d{8}$/;
    if( !reg.test(this.mobile) ){
      return false;
    }

    // 如果get_sms_text 不是文本, 而是数字, 则表示当前手机号码还在60秒的发送短信间隔内
    if(this.get_sms_text !== "获取验证码"){
      return false;
    }

    // 发送短信
    let _this = this;
    this.$axios.get("http://127.0.0.1:8000/users/sms?
mobile="+this.mobile).then(response=>{
      console.log(response);
      // 显示发送短信以后的文本倒计时
      let time = 60;
      let timer = setInterval(()=>{
        --time;
        if(time <=1){
          // 如果倒计时为0, 则关闭当前定时器
          _this.get_sms_text = "获取验证码";
          clearInterval(timer);
        }else{
          _this.get_sms_text = time;
        }
      },1000)
    }).catch(error=>{
      console.log(error);
    })

  },
  registerHander(){
    // 注册信息提交

```

```

// 提交数据前判断用户是否通过了验证码校验
if(!this.validateResult){
    alert("验证码验证有误");
    return false;
}
this.$axios.post("http://127.0.0.1:8000/users/user",{
    "mobile":this.mobile,
    "password":this.password,
    "password2":this.password2,
    "sms_code":this.sms_code,
},{
    responseType:"json",
}).
then(response=>{
    // 请求成功, 保存登陆状态
    localStorage.removeItem("token");
    let data = response.data;
    sessionStorage.token = data.token;
    sessionStorage.id = data.id;
    sessionStorage.username = data.mobile;
    // 注册成功以后默认表示已经登录了, 跳转用户中心的页面
    // this.$router.push("/user");
    alert("注册成功!");
}).catch(error=>{
    console.log(error);
})
},
handlerPopup(captchaObj){
    // 验证码成功的回调
    let _this = this;
    captchaObj.onSuccess(function () {
        var validate = captchaObj.getValidate();
        _this.$axios.post("http://127.0.0.1:8000/users/verify",{
            geetest_challenge: validate.geetest_challenge,
            geetest_validate: validate.geetest_validate,
            geetest_seccode: validate.geetest_seccode
        },{
            responseType:"json",
        }).then(response=>{
            // 请求成功
            console.log(response.data);
            if(response.data.status == "success") {
                _this.validateResult = true; // 获取验证结果
            }
        }).catch(error=>{
            // 请求失败
            console.log(error)
        })
    });
    // 将验证码加到id为captcha的元素里
    captchaObj.appendTo("#geetest");
}
},

```



```
};
</script>

<style scoped>
.box{
    width: 100%;
    height: 100%;
    position: relative;
    overflow: hidden;
}
.el-select{
    width:100%;
    margin-bottom: 15px;
}
.box img{
    width: 100%;
    min-height: 100%;
}
.box .register {
    position: absolute;
    width: 500px;
    height: 400px;
    top: 0;
    left: 0;
    margin: auto;
    right: 0;
    bottom: 0;
    top: -338px;
}
.register .register-title{
    width: 100%;
    font-size: 24px;
    text-align: center;
    padding-top: 30px;
    padding-bottom: 30px;
    color: #4a4a4a;
    letter-spacing: .39px;
}
.register-title img{
    width: 190px;
    height: auto;
}
.register-title p{
    font-family: PingFangSC-Regular;
    font-size: 18px;
    color: #fff;
    letter-spacing: .29px;
    padding-top: 10px;
    padding-bottom: 50px;
}
.sms{
    margin-top: 15px;
```

```

    position: relative;
}
.sms .get_sms{
    position: absolute;
    right: 15px;
    top: 14px;
    font-size: 14px;
    color: #ffc210;
    cursor: pointer;
    border-left: 1px solid #979797;
    padding-left: 20px;
}

.register_box{
    width: 400px;
    height: auto;
    background: #fff;
    box-shadow: 0 2px 4px 0 rgba(0,0,0,.5);
    border-radius: 4px;
    margin: 0 auto;
    padding-bottom: 40px;
}
.register_box .title{
    font-size: 20px;
    color: #9b9b9b;
    letter-spacing: .32px;
    border-bottom: 1px solid #e6e6e6;
    display: flex;
        justify-content: space-around;
    padding: 50px 60px 0 60px;
    margin-bottom: 20px;
    cursor: pointer;
}
.register_box .title span:nth-of-type(1){
    color: #4a4a4a;
    border-bottom: 2px solid #84cc39;
}

.inp{
    width: 350px;
    margin: 0 auto;
}
.inp input{
    border: 0;
    outline: 0;
    width: 100%;
    height: 45px;
    border-radius: 4px;
    border: 1px solid #d9d9d9;
    text-indent: 20px;
    font-size: 14px;
    background: #fff !important;
}

```

```

.inp input.user{
    margin-bottom: 16px;
}
.inp .rember{
    display: flex;
    justify-content: space-between;
    align-items: center;
    position: relative;
    margin-top: 10px;
}
.inp .rember p:first-of-type{
    font-size: 12px;
    color: #4a4a4a;
    letter-spacing: .19px;
    margin-left: 22px;
    display: -ms-flexbox;
    display: flex;
    -ms-flex-align: center;
    align-items: center;
    /*position: relative;*/
}
.inp .rember p:nth-of-type(2){
    font-size: 14px;
    color: #9b9b9b;
    letter-spacing: .19px;
    cursor: pointer;
}

.inp .rember input{
    outline: 0;
    width: 30px;
    height: 45px;
    border-radius: 4px;
    border: 1px solid #d9d9d9;
    text-indent: 20px;
    font-size: 14px;
    background: #fff !important;
}

.inp .rember p span{
    display: inline-block;
    font-size: 12px;
    width: 100px;
    /*position: absolute;*/
    /*left: 20px;*/
}
#geetest{
    margin-top: 20px;
}
.register_btn{
    width: 100%;
    height: 45px;
}

```

```
    background: #84cc39;
    border-radius: 5px;
    font-size: 16px;
    color: #fff;
    letter-spacing: .26px;
    margin-top: 30px;
}
.inp .go_login{
    text-align: center;
    font-size: 14px;
    color: #9b9b9b;
    letter-spacing: .26px;
    padding-top: 20px;
}
.inp .go_login span{
    color: #84cc39;
    cursor: pointer;
}
</style>
```