# 项目介绍

* 1. 项目描述简介

通过完成一个仿饿了么的在线订餐平台，让大家掌握企业开发特点，以及解决问题的能力。

该项目会涉及以后工作中的一些非常有代表性的功能。

用户端采用前后端分离开发模式，前端通过调用后端API接口来获取和提交数据。掌握项目中前后端分离的开发模式。

我们使用git作为项目的版本控制软件。

在项目中会使用很多其他的知识，比如系统架构、维护、优化等等。

* 1. 主要功能模块

用户模块,商品模块,购物车模块,订单模块

* 1. 开发环境和技术

|  |  |
| --- | --- |
| 开发环境 | Window |
| 开发工具 | pycharm,navicat |
| 相关技术 | django,mysql,session,cookie,sqlite3,redis, ckeditor 等 |

* 1. 项目人员组成周期成本
     1. 人员组成

|  |  |  |
| --- | --- | --- |
| 职位 | 人数 | 备注 |
| 项目经理和组长 | 1 | 一般小公司由项目经理负责管理，中大型公司项目由项目经理或组长负责管理 |
| 开发人员 | 2~3 |  |
| UI设计人员 | 1 |  |
| 前端开发人员 | 1 | 专业前端不是必须的，所以前端开发和UI设计人员可以同一个人 |
| 测试人员 | 1~3 | 有些公司并未有专门的测试人员，测试人员可能由开发人员完成测试。  公司有测试部，测试部负责所有项目的测试。  项目测试由产品经理进行业务测试。 |

* + 1. 项目周期成本

|  |  |  |
| --- | --- | --- |
| 人数 | 周期 | 备注 |
| 3 | 四个月  第一个月需求设计  第二三个月编码实现  第四个月编码实现、测试 |  |

1. 用户功能模块

2.1用户账号注册功能模块

* 1. 1需求

1.注册页面:手机号的正则验证,6位数以上的密码,两次输入密码的正确性, 协议同意的勾选与否的验证,填写完毕后点击获取验证码.

2.以上流程完成后获取到验证码,验证码的验证,点击注册后,加密密码,写入数据库.

2.1.2流程

1.用户填写手机,密码等信息,填写后点击获取验证码,判断手机格式是否有问题,没有就向验证码平台发起验证码请求

2. 验证码平台返回并响应验证码,用户填写验证码,勾选协议,点击注册,form表单返回数据

3.form表单验证数据是否合法:手机是否已注册,手机格式是否合法,密码长度是否足够,两次密码是否一致,协议是否勾选同意,如果有问题,返回错误提示

4.如果用户信息无误,则注册成功,写入数据库并加密密码,跳转登录页面.

2.1.3设计要点（数据库和页面交互）

1.数据库的字段

Id 主键

mobile 手机号 varchar (必填)

password 密码 varchar (必填,哈希加密)

sex 性别 chioce

birthday 生日 datatime

school 学校 varchar

location 地址 varchar

hometown 家庭地址 varchar

is\_delete 是否删除 boolean(假删)

add\_time 创建时间 datetime （用户注册时间）

update\_time 更新时间 datetime （用户修改信息时间）

2. 前端页面对数据的校验

* 1. 对每一个文本框添加placeorder说明字段；
  2. 文本框进行空字段校验，如果为空，点击注册按钮后会提示该项为必填项；
  3. 对必填字段进行字数的校验，如果用户填写的字数少于或多余规定的字符长度，提示用户；
  4. 对手机号码进行正则判断，用户如果输入了格式不正确的手机号，提示用户输入正确的手机号；

2.1.4要点难点及解决方案

难点1：验证码的获取

解决方案：通过阿里云发送，申请一个模板和签名，当用户点击获取验证码时，阿里云向该手机号发送验证码

难点2：项目根目录的判断

解决方案：找到有包含子目录的一级

难点3：验证form表单的数据

解决方案：在子运用到的目录下创建一个forms.py文件定义,创建验证的类来规定验证

2.2用户登录功能模块

* 1. 1需求

1.手机号的正则验证,6位数以上的密码,输入密码的正确性.

2. 对用户提交过来的数据进行校验，如果数据与数据库内的信息一致，登录成功，跳转到商城主页；

2.2.2流程

1.用户填写手机,密码,判断手机格式是否有问题,没有form表单返回数据

2.form表单验证数据是否合法:手机是否已注册,手机格式是否合法,密码长度是否足够,密码不能为空,如果有问题,返回错误提示

4. 将加密好的密码和手机号码与数据库内的数据进行比较，如果数据一致，用户登录成功。

2.2.3设计要点（数据库和页面交互）

1.数据库的字段

Id 主键

mobile 手机号 varchar (必填)

password 密码 varchar (必填,哈希加密)

sex 性别 chioce

birthday 生日 datatime

school 学校 varchar

location 地址 varchar

hometown 家庭地址 varchar

is\_delete 是否删除 boolean(假删)

add\_time 创建时间 datetime （用户注册时间）

update\_time 更新时间 datetime （用户修改信息时间）

2. 交互设计

1. 对文本框进行空字段校验，如果为空，登录失败，提示用户填写基本信息；
2. 对填写的字段进行基本判断，判断用户输入的字符长度是否符合规范；
3. 为文本框添加placeorder属性，提示用户需要填写的信息。
4. 对手机号码进行正则判断，用户如果输入了格式不正确的手机号，提示用户输入正确的手机号；

2.2.4要点难点及解决方案

难点1：用户填写的密码为明文密码，如何与哈希过的密码进行比较

解决方案：

将用户填写的基本校验成功后的密码进行哈希化并且循环加盐，将加密后的密码与数据库内的密码进行比较是否一致。

难点2：验证form表单的数据

解决方案：

在子运用到的目录下创建一个forms.py文件定义,创建验证的类来规定验证

* 1. 用户修改功能模块
     1. 需求
        1. 用户的信息渲染到个人资料页面；
        2. 用户到个人资料页面修改个人资料页面内的信息；
        3. 忘记密码,修改密码必须在用户登录状态下完成
     2. 流程
        1. 填写好基本信息后，用户点击修改按钮将数据进行提交；
        2. Form验证层对提交过来的数据进行校验；
        3. 如果校验通过了，用户的数据将被修改成功。

用户登录状态下完成忘记密码,修改密码,通过提交form信息进行验证,

通过修改,否则提示用户信息

* + 1. 设计要点（数据库和页面交互）
       1. 数据库同注册数据库一致
       2. 交互
    2. 为文本框添加placeorder属性，提示用户需要填写的信息；
    3. 对填写的修改信息进行校验，数据是否合法，长度是否符合规范；
    4. 上传图片时判断用户是否有上传图片；
    5. 对必填字段进行空字段校验，如果为空提示用户输入必填项。
    6. 如果校验通过了，用户的数据将被修改成功。

用户登录状态下完成忘记密码,修改密码,通过提交form信息进行验证,

通过修改,否则提示用户信息

* + 1. 要点难点及解决方案

1.难点1：当用户登录后，如何将用户的信息渲染到个人信息页面？

解决方案：

使用session来获取用户的唯一信息，再使用唯一信息进行数据查询，将查询到的数据渲染到页面上进行显示。

2.难点2：用户状态的保存

解决方案：

用session将用户的唯一信息进行保存，当用户登录后，session会被传递到浏览器，用户的信息也将被保存,使用django-redis保存session减少数据库压力,做到优化效果。

3. 难点3：session保存后的使用

解决方案：

创建一个保存session的方法,方便在程序中调用

4.用户头像的上传

解决方案：

在infor的html中的form表单中加入enctype="multipart/form-data"属性,通过session查询数据库,找到用户id,安装并使用pillow(专门处理图片)包,创建图片存储地址,通过id调用地址,并且在用户提交后刷新session使用户能马上看到更改后的头像,注意:判断用户头像是否更改,没有则不更新

5.用户上传时间问题

解决方案：

在infor的html中将时间的input框加入style=’date’,就不用转换数据格式

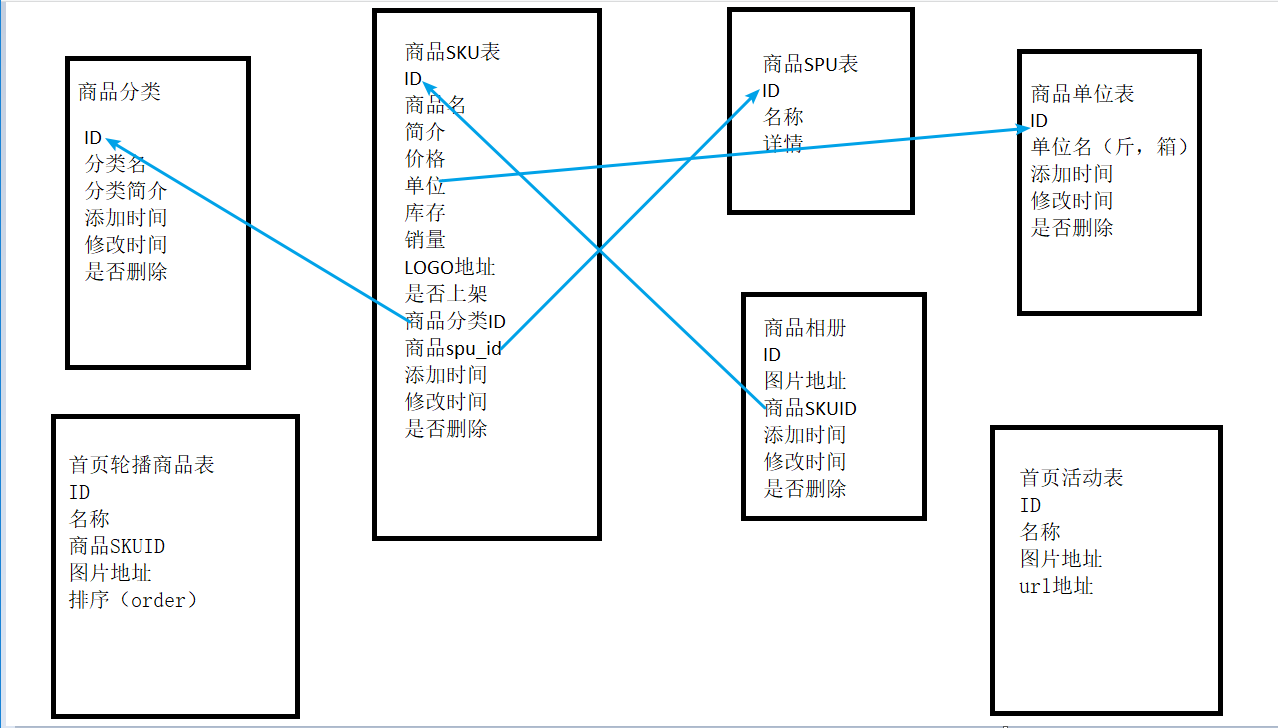
’

1. 商品功能模块

3.1商品增删改查模块

* + 1. 需求
       1. 商家登录后台.
       2. 商家可以增删查改或者看商品信息
       3. 完成添加的信息能写入数据库并且渲染到前台
    2. 流程
       1. 设计好数据库；
       2. 将数据库内供商家控制的模型添加到admin模块中方便更新管理；
       3. 创建超级用户来管理后端数据；
       4. 通过后台账户进入到后台；
       5. 对数据进行增删查改。
    3. 设计要点（数据库和页面交互）

数据库:

分析表:

* + - 1. 商品分类

# 商品模型一：  
# 商品分类  
# ID  
# 分类名称  
# 描述  
# 排序  
class Category(BaseModel):  
 cate\_name = models.CharField(verbose\_name='分类名称',  
 max\_length=20  
 )  
 brief = models.CharField(verbose\_name='描述',  
 max\_length=200,  
 null=True,  
 blank=True  
 )  
 order = models.SmallIntegerField(default=0, verbose\_name="排序")  
  
 def \_\_str\_\_(self):  
 return self.cate\_name  
  
 class Meta:  
 verbose\_name = "商品分类管理"  
 verbose\_name\_plural = verbose\_name

* + - 1. 商品SKU单位

# 商品模型二:  
# 商品SKU单位  
# ID  
# 单位（斤，箱）  
class Unit(BaseModel):  
 name = models.CharField(max\_length=20,  
 verbose\_name="单位")  
  
 def \_\_str\_\_(self):  
 return self.name  
  
 class Meta:  
 verbose\_name = "商品单位管理"  
 verbose\_name\_plural = verbose\_name

* + - 1. 商品SPU表

# 商品模型三：  
# 商品SPU表  
# ID  
# 商品SPU名称  
# 商品详情  
class GoodsSPU(BaseModel):  
 spu\_name = models.CharField(verbose\_name='商品SPU名称',  
 max\_length=20  
 )  
 content = models.TextField(verbose\_name="商品详情")  
  
 def \_\_str\_\_(self):  
 return self.spu\_name  
  
 class Meta:  
 verbose\_name = "商品SPU"  
 verbose\_name\_plural = verbose\_name

* + - 1. 商品SKU表

# 模型四：  
# 商品SKU表  
# ID  
# 商品SKU名称  
# 商品的简介  
# 价格  
# 单位  
# 库存  
# 销量  
# 封面图片  
# 是否上架  
# 商品分类  
# 商品SPU  
class GoodsSKU(BaseModel):  
 sku\_name = models.CharField(verbose\_name='商品SKU名称',  
 max\_length=100,  
 )  
 brief = models.CharField(verbose\_name="商品的简介",  
 max\_length=200,  
 null=True,  
 blank=True,  
 )  
 price = models.DecimalField(verbose\_name='价格',  
 max\_digits=9,  
 decimal\_places=2,  
 default=0,  
 )  
 unit = models.ForeignKey(to="Unit", verbose\_name="单位")  
  
 stock = models.IntegerField(verbose\_name='库存',  
 default=0)  
  
 sale\_num = models.IntegerField(verbose\_name='销量',  
 default=0)  
  
 # 默认相册中的第一张图片作为封面图片  
 logo = models.ImageField(verbose\_name='封面图片',  
 upload\_to='goods/%Y%m/%d'  
 )  
  
 is\_on\_sale = models.BooleanField(verbose\_name="是否上架",  
 choices=is\_on\_sale\_choices,  
 default=False)  
  
 category = models.ForeignKey(to="Category",  
 verbose\_name='商品分类'  
 )  
  
 goods\_spu = models.ForeignKey(to="GoodsSPU", verbose\_name="商品SPU")  
  
 def \_\_str\_\_(self):  
 return self.sku\_name  
  
 class Meta:  
 verbose\_name = "商品SKU管理"  
 verbose\_name\_plural = verbose\_name

* + - 1. 商品相册管理

# 模型五：  
# 商品相册管理  
# ID  
# 商品相册  
# 相册图片地址  
# 商品SKU  
class Gallery(BaseModel):  
 img\_url = models.ImageField(verbose\_name='相册图片地址',  
 upload\_to='goods\_gallery/%Y%m/%d'  
 )  
  
 goods\_sku = models.ForeignKey(to="GoodsSKU", verbose\_name="商品SKU")  
  
 class Meta:  
 verbose\_name = "商品相册管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return "商品相册:{}".format(self.img\_url.name)

* + - 1. 首页轮播

# 模型六：  
# 首页轮播  
# ID  
# 轮播活动名  
# 轮播图片地址  
# 排序  
# 商品SKU  
class Banner(BaseModel):  
 name = models.CharField(verbose\_name="轮播活动名",  
 max\_length=150,  
 )  
 img\_url = models.ImageField(verbose\_name='轮播图片地址',  
 upload\_to='banner/%Y%m/%d'  
 )  
 order = models.SmallIntegerField(verbose\_name="排序",  
 default=0,  
 )  
  
 goods\_sku = models.ForeignKey(to="GoodsSKU", verbose\_name="商品SKU")  
  
 class Meta:  
 verbose\_name = "轮播管理"  
 verbose\_name\_plural = verbose\_name  
  
 def \_\_str\_\_(self):  
 return self.name

* + - 1. 首页活动

# 模型七：  
# 首页活动  
# ID  
# 活动名称  
# 活动图片地址  
# 活动的url地址  
class Activity(BaseModel):  
 title = models.CharField(verbose\_name='活动名称', max\_length=150)  
 img\_url = models.ImageField(verbose\_name='活动图片地址',  
 upload\_to='activity/%Y%m/%d'  
 )  
 url\_site = models.URLField(verbose\_name='活动的url地址', max\_length=200)  
  
 def \_\_str\_\_(self):  
 return self.title  
  
 class Meta:  
 verbose\_name = "活动管理"  
 verbose\_name\_plural = verbose\_name

* + - 1. 首页活动专区

# 模型八：  
# 首页活动专区  
# ID  
# 活动专区名称  
# 排序  
# 上否上线  
# 商品  
class ActivityZone(BaseModel):  
 title = models.CharField(verbose\_name='活动专区名称', max\_length=150)  
 brief = models.CharField(verbose\_name="活动专区的简介",  
 max\_length=200,  
 null=True,  
 blank=True,  
 )  
 order = models.SmallIntegerField(verbose\_name="排序",  
 default=0,  
 )  
 is\_on\_sale = models.BooleanField(verbose\_name="上否上线",  
 choices=is\_on\_sale\_choices,  
 default=0,  
 )  
 goods\_sku = models.ManyToManyField(to="GoodsSKU", verbose\_name="商品")  
  
 def \_\_str\_\_(self):  
 return self.title  
  
 class Meta:  
 verbose\_name = "活动专区管理"  
 verbose\_name\_plural = verbose\_name

页面交互

1)在创建时注意要在模型里创建图片地址,方便后台上传图片

2)每张表的对应关系要分析清楚再创建,以免以后修改麻烦

* + 1. 要点难点及解决方案
       1. 后台admin创建orm操作比较吃力

解决方案：

查看Django文档,网上找资料

* + - 1. 在视图中写入查询数据库显示到网址时,index网页报错

解决方案：

由于我先渲染了index页面才发现detail页面中需要定义每个商品的id,通过id查询商品,定义后index中html的detail反向路由便缺少id,所以报错,将商品的SKU在主页的视图中查询出来,并在index的html中detail反向路由后加上for循环的id

* + - 1. 富文本编辑器ckeditor的使用,使文本丰富,便于后台操作

解决方案：

首先安装django-ckeditor, 将ckeditor添加到您的INSTALLED\_APPS设置中,注意配置静态文件STATIC\_ROOT = os.path.join(BASE\_DIR, "static"),将django中安装在INSTALLED\_APPS的其他应用所有的静态文件收集到STATIC\_ROOT指定目录下, 配置好后注意注释掉静态文件, 设置ckeditor的上传目录 ,在主路由中配置 ckeditor 上传文件使用到的url地址 ,安装pillow(已经安装不用再安装), 设置ckeditor在后台显示的样式,这样admin后台就能很好的编辑文本了

* + - 1. 商品模块的前台显示的设计在,数据库的模型关系问题

解决方案：

在视图设计中,打开models,看到每个表之间的设计关系,多对多,一对多,还是一对一,在调用自己想要的数据时可以通过Ctrl+鼠标右键跳转到上一级关系上,多看就不会头晕,一步一步的来,不容易出错,要是出错不报错没数据显示,可以尝试断点+debug,调试一下,大部分问题都能解决