浙江大学

# 本科实验报告

| | |
|---|---|
| 课程名称： | B/S 体系结构 |
| 实验名称： | B/S 体系结构课程设计文档 |
| 姓　　名： | 宋汝毅 |
| 学　　院： | 计算机学院 |
| 系： | 计算机科学与技术 |
| 专　　业： | 计算机科学与技术 |
| 学　　号： | 计算机科学与技术 |
| 指导教师： | 胡晓军 |

2024 年　12　月　27　日

# 浙江大学实验报告

课程名称：B/S 体系结构 实验类型：编程实验

实验项目名称：B/S 体系结构课程设计文档

学生姓名：宋汝毅 专业：计算机科学与技术 学号：3220105868

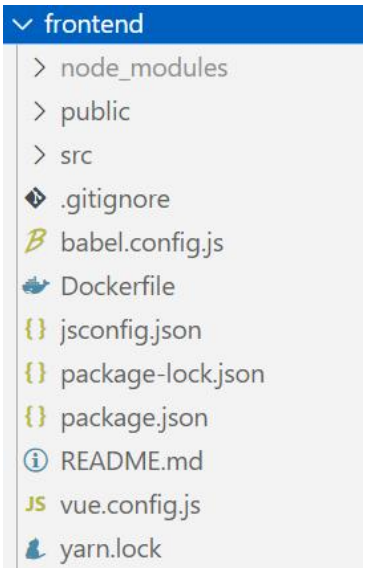
同族学生姓名：/ 指导老师：胡晓军

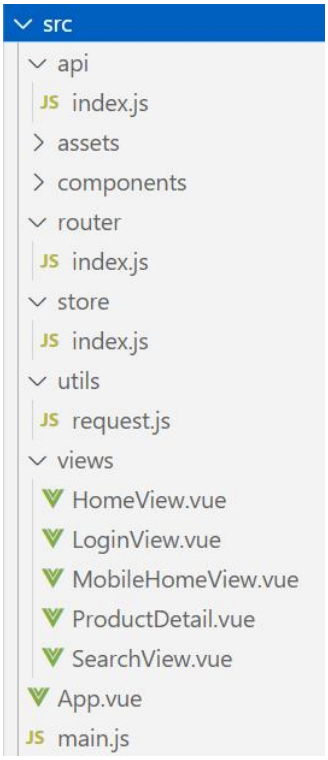
实验地点：线上 实验日期：2024 年 12 月 27 日

# 设计文档

## 一、前端设计——Vue3

1. 文件组织形式：




2. 各组成代码和功能说明：

主要前端显示页面是在./frontend/src/views 的 5 个 vue 文件和./frontend/src 下的 app.vue 文件，其中 app.vue 是用来显示 HomeView.vue 的初始界面，LoginView.vue 是

负责登录与注册的页面，在成功登录后会跳转到 SearchView.vue，即搜索界面，搜索商品进入查看详情时，会跳转显示 ProductDetail.vue，如果适配移动端的话，app.vue 就会显示 MobileHomeView.vue。具体的页面展示已经验收过，也可以看功能视频查看。

api 下的 index.js 定义了一些后端接口和 method 方法，具体如下：

```
import request from '@/utils/request'

// 用户相关
export function login(data) {
    return request({
        url: '/api/login/',
        method: 'post',
        data
    })
}

export function register(data) {
    return request({
        url: '/api/register/',
        method: 'post',
        data
    })
}

// 商品相关
export function searchProducts(keyword) {
    return request({
        url: '/api/products/search/',
        method: 'get',
        params: { keyword }
    })
}

// 获取商品详情 - 使用统一的 request 方式
export function getProductDetail(id) {
    return request({
        url: `/api/products/${id}/`,
        method: 'get',
        headers: {
            'Authorization': `Bearer ${localStorage.getItem('access_token')}`
        }
    })
}
```

router 下的 index.js 定义了一些路由，包括’/’，’/mobile-home’，’/login’，’/search’和’/product/:id’。

store 下的 index.js 定义了一些登入登出相关的 token 信息。

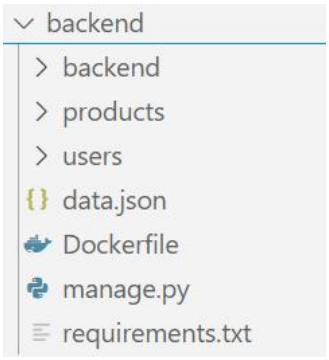
utils 下的 request.js 定义了一些向后端的接口，包括未登录不能访问后端等功能。

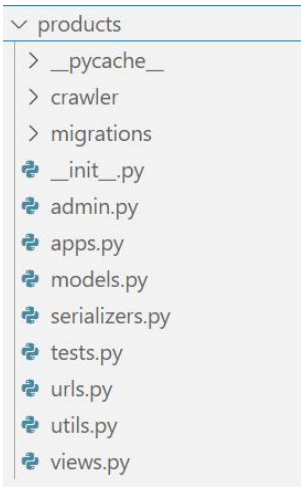
src 下的 main.js 是一个主配置文件，里面包括配置 app 和 axios 的相关信息

```
const app = createApp(App)

// 配置 axios
axios.defaults.baseURL = 'http://127.0.0.1:8000'

app.use(router)
app.use(ElementPlus)

app.mount('#app')
```

## 二、后端设计——Django

1. 文件组织形式：




2. 各组成代码和功能说明：

后端主要由主 app——backend 和两个自己创建的 app——products 和 users 构成，每个 app 主要的文件基本包括__init__.py，admin.py（用于后端注册模型），apps.py（用与后端在数据库建立模型），models.py（用于在数据库中建表），serializers.py（序列化器，用于规范数据），urls.py（后端路由设计），utils（主要用于查询优化，引用了 jieba 库），views.py（定义了后端的视图，可以用 GET 和 POST 等应用层 HTTP 协议 get 或者 post 数据）

admin.py 代码示例：

```
from django.contrib import admin
from .models import Category, Product, Platform, Price

# 注册模型
admin.site.register(Category)
admin.site.register(Product)
admin.site.register(Platform)
admin.site.register(Price)
```

apps.py 代码示例：

```python
from django.apps import AppConfig


class ProductsConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'products'

```

models.py 代码示例：

```python
class Product(models.Model):
    name = models.CharField(max_length=200)
    description = models.TextField()
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
    specifications = models.JSONField()  # 存储规格信息
    image_url = models.URLField()
    keywords = models.TextField()  # 存储分词后的关键词
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        db_table = 'products'
```

serializers.py 代码示例：

```python
class ProductSerializer(serializers.ModelSerializer):
    prices = PriceSerializer(many=True, read_only=True)
    category_name = serializers.CharField(source='category.name', read_only=True)

    class Meta:
        model = Product
        fields = ['id', 'name', 'description', 'category_name',
                  'specifications', 'image_url', 'prices']
```

urls.py 代码示例：

```python
urlpatterns = [
    # ... 其他URL配置 ...
    path('search/', views.search_products, name='search_products'),
    path('<int:product_id>/', views.product_detail, name='product_detail'),
    path('categories/add/', views.add_category, name='add_category'),
    path('products/add/', views.add_product, name='add_product'),
    path('platforms/add/', views.add_platform, name='add_platform'),
    path('prices/add/', views.add_price, name='add_price'),
]
```

utils.py 代码示例：

```python
def process_search_keywords(keyword):
    # 使用结巴分词
    words = jieba.cut(keyword, cut_all=False)
    return ' '.join(words)

def search_products(keyword):
    from .models import Product
    from django.db.models import Q

    processed_keywords = process_search_keywords(keyword)
    keywords_list = processed_keywords.split()

    query = Q()
    for word in keywords_list:
        query |= Q(name__icontains=word) | Q(keywords__icontains=word)

    return Product.objects.filter(query).distinct()
```

views.py 代码示例：

```python
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def search_products(request):
    keyword = request.query_params.get('keyword', '')
    if not keyword:
        return Response({'detail': '请提供搜索关键词'}, status=400)

    try:
        # 搜索商品（名称、描述和关键词）
        products = Product.objects.filter(
            Q(name__icontains=keyword) |
            Q(description__icontains=keyword) |
            Q(keywords__icontains=keyword)
        ).values(
            'id',
            'name',
            'description',
            'image_url',
            'category_id'
        )
```

以上仅仅是一个 app products 的代码示例，products app 负责管理商品的相关内容，还有一个 users app 和他的组织结构一样，负责管理用户注册登录的相关信息

还有一个 django 自带的 manage.py，用于直接运行后端，未作修改，展示如下：

```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'backend.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()

```

特别的，对./backend/settings.py 文件做出展示：

INSTALLED_APPS：

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # ...默认应用
    'rest_framework',
    'corsheaders',
    'users',
    'products',
    'rest_framework_simplejwt',
]
```

MIDDLEWARE：

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'corsheaders.middleware.CorsMiddleware',
]
```

DATABASES：

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'BS_project',
        'USER': 'root',
        'PASSWORD': 'root', #如果连接docker就写root，自己电脑测试用123!@zxcvbnm
        'HOST': 'db', #如果连接docker就写mysql，自己电脑测试用localhost，新的db
        'PORT': '3306',
    }
}
```

## 三、数据库设计——MySQL

1. 数据库组织形式：

本地数据库端口为 localhost:3306，数据库名称叫 BS_project，数据库中有一些 django 自带的表，根据项目需要添加的表为 categories，platforms，prices，products，

users，users_groups（未使用），users_user_permissions（未使用）：


categories 表头为 id（BIDINT），name（VARCHAR(100)），level（INT），created_at（DATETIME(6)）和 parent_id（BIGINT）。

platforms 表头为 id（BIDINT），name（VARCHAR(200)），description（LONGTEXT），specifications（JSON），image_url（VARCHAR(200)），keywords（LONGTEXT），created_at（DATETIME(6)），category_id（BIGINT）。

prices 表头为 id（BIDINT），price（DECIMAL(10, 2)），url（VARCHAR(200)），created_at（DATETIME(6)），platform_id（BIGINT），product_id（BIGINT）。

products 表头为 id（BIDINT），name（VARCHAR(200)），description（LONGTEXT），specifications（JSON），image_url（VARCHAR(200)），keywords（LONGTEXT），created_at（DATETIME(6)），category_id（BIGINT）。

其中每个表的 id 项为 PRIMARY KEY，各个表中的对应 id 与其主表中的 id 构成 FOREIGN KEY 依赖关系。

## 四、Docker 相关设计——docker-compose

1. 前后端 Dockerfile 配置：

前端 Dockerfile：

```dockerfile
# 使用 Node.js 16 版本
FROM node:16

# 设置工作目录
WORKDIR /app

# 复制 package.json 和 package-lock.json
COPY package*.json ./

# 安装依赖
RUN npm install

# 复制项目文件
COPY . .

# 启动开发服务器（或生产模式服务）
EXPOSE 8080
CMD ["npm", "run", "serve"]

```

后端 Dockerfile：

```dockerfile
# 使用 Python 3.10
FROM python:3.10

# 设置工作目录
WORKDIR /app

# 复制 requirements.txt
COPY requirements.txt .

# 安装依赖
RUN pip install --no-cache-dir -r requirements.txt

# 复制项目文件
COPY . .

# 暴露后端服务端口
EXPOSE 8000

# 启动 Django 开发服务器
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]

```

2. 整个项目 docker-compose.yml 配置：

docker-compose.yml：

说明：frontend 前端服务映射到宿主机的 8080 端口，backend 后端服务映射到宿主机的 8000 端口，创建数据库 db 依赖，数据库映射到宿主机的 3307 端口，此外还专门使用了 volume 卷（这里在验收的时候还是有数据库的迁移问题，经过查找资料并尝试后，我用 volume 备份的方式解决了这个问题）。

docker-compose.yml

```yaml
version: '3.9'

services:
  frontend:
    build:
      context: ./frontend
    ports:
      - "8080:8080"  # 将 Vue 前端服务映射到宿主机的 8080 端口
    networks:
      - app-network

  backend:
    build:
      context: ./backend
    ports:
      - "8000:8000"  # 将 Django 后端服务映射到宿主机的 8000 端口
    depends_on:
      - db
    networks:
      - app-network

  db:
    image: mysql:8.0
    container_name: mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: BS_project
    ports:
      - "3307:3306"  # 将数据库端口映射到宿主机
    volumes:
      - db_data:/var/lib/mysql  # 数据持久化
    networks:
      - app-network

volumes:
  db_data:  # 定义 MySQL 数据卷

networks:
  app-network:  # 定义服务网络

```

3. 项目以 docker-compose 的方式启动：

说明：具体参照 README 文件操作即可。

——END