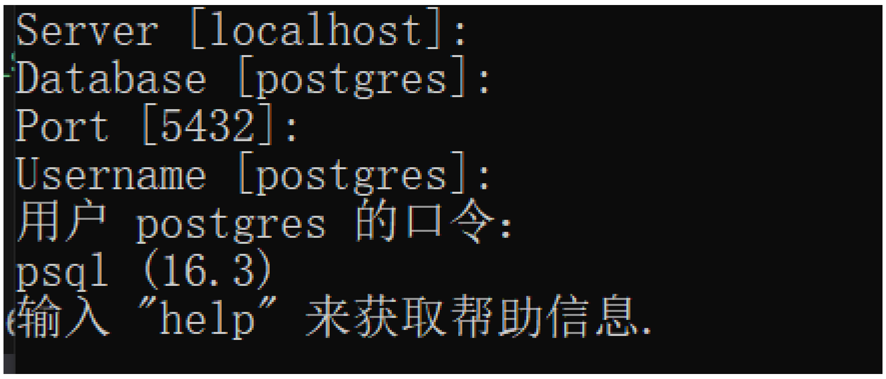
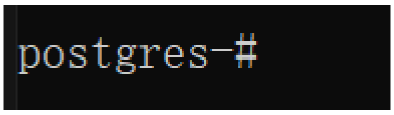
1.数据库配置

1.1 进入默认的postgres数据库



```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
用户 postgres 的口令:
psql (16.3)
输入 "help" 来获取帮助信息.
```

一般会直接默认进入postgres

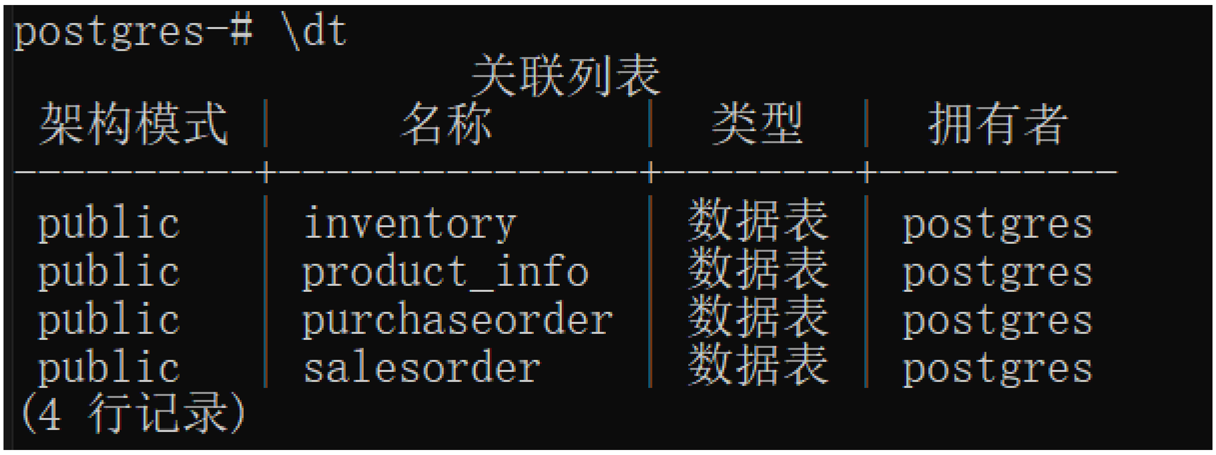


```
postgres-#
```

即命令行显示：postgres-#

1.2构建表格

将要构建下面四个表格



```
postgres-# \dt
                    关联列表
 架构模式 |      名称      |  类型  |  拥有者
----------+----------------+--------+----------
 public   | inventory      | 数据表 | postgres
 public   | product_info   | 数据表 | postgres
 public   | purchaseorder  | 数据表 | postgres
 public   | salesorder     | 数据表 | postgres
(4 行记录)
```

Product_info表 命令行输入

```
CREATE TABLE Product_info (
    product_id INTEGER PRIMARY KEY,
    product_name VARCHAR,
    specification VARCHAR,
    description TEXT,
    classification VARCHAR,
    price NUMERIC
);
```

Inventory表

```
CREATE TABLE Inventory (
    product_id INTEGER PRIMARY KEY REFERENCES Product_info(product_id),
    stock_quantity INTEGER
);
```

Purchaseorder表

```
CREATE TABLE Purchaseorder (
    order_number VARCHAR PRIMARY KEY,
    purchase_date DATE,
    unit_price NUMERIC,
    quantity INTEGER,
    supplier VARCHAR,
    received BOOLEAN,
    product_id INTEGER REFERENCES Product_info(product_id)
);
```

Salesorder表s

```
CREATE TABLE Salesorder (
    order_number VARCHAR PRIMARY KEY,
    sales_date DATE,
    unit_price NUMERIC,
    quantity INTEGER,
    product_id INTEGER REFERENCES Product_info(product_id)
);
```

1.2构建触发器

触发器1

```sql
-- 创建触发器函数
CREATE OR REPLACE FUNCTION create_inventory_entry()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Inventory(product_id, stock_quantity)
    VALUES (NEW.product_id, 0);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- 创建触发器
CREATE TRIGGER after_productinfo_insert_trigger
AFTER INSERT ON Product_info
FOR EACH ROW
EXECUTE FUNCTION create_inventory_entry();
```

触发器2

```sql
-- 创建或替换触发器函数
CREATE OR REPLACE FUNCTION update_inventory_after_purchase()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        -- 增加库存
        UPDATE Inventory
        SET stock_quantity = stock_quantity + NEW.quantity
        WHERE product_id = NEW.product_id;
    ELSIF TG_OP = 'UPDATE' THEN
        -- 更新库存（例如，如果数量发生变化）
        UPDATE Inventory
        SET stock_quantity = stock_quantity + (NEW.quantity - OLD.quantity)
        WHERE product_id = NEW.product_id;
    ELSIF TG_OP = 'DELETE' THEN
        -- 减少库存
        UPDATE Inventory
        SET stock_quantity = stock_quantity - OLD.quantity
        WHERE product_id = OLD.product_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- 创建触发器
CREATE TRIGGER after_purchase_trigger
AFTER INSERT OR UPDATE OR DELETE ON Purchaseorder
FOR EACH ROW
EXECUTE FUNCTION update_inventory_after_purchase();
```

触发器3

```
-- 创建或替换触发器函数
CREATE OR REPLACE FUNCTION update_inventory_after_sale()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        -- 减少库存
        UPDATE Inventory
        SET stock_quantity = stock_quantity - NEW.quantity
        WHERE product_id = NEW.product_id;
    ELSIF TG_OP = 'UPDATE' THEN
        -- 更新库存（例如，如果数量发生变化）
        UPDATE Inventory
        SET stock_quantity = stock_quantity - (NEW.quantity - OLD.quantity)
        WHERE product_id = NEW.product_id;
    ELSIF TG_OP = 'DELETE' THEN
        -- 增加库存（在删除销售订单时，相当于撤销销售）
        UPDATE Inventory
        SET stock_quantity = stock_quantity + OLD.quantity
        WHERE product_id = OLD.product_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- 创建触发器
CREATE TRIGGER after_sale_trigger
AFTER INSERT OR UPDATE OR DELETE ON Salesorder
FOR EACH ROW
EXECUTE FUNCTION update_inventory_after_sale();
```

2.后端配置

我会把修改后的后端发在群里，用这个版本的后端代码，但可能要根据自己的数据库参数修改一点配置文件

2.1 配置setting.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD': 'Yza20020928',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

将其替换成自己数据库的参数即可。如果安装数据都是默认，而且跟上面我的数据库一样的话，应该只需要改一个密码就行

2.2 运行后端所需要的python环境

```
conda install django
pip install djangorestframework
pip install django-cors-headers
```

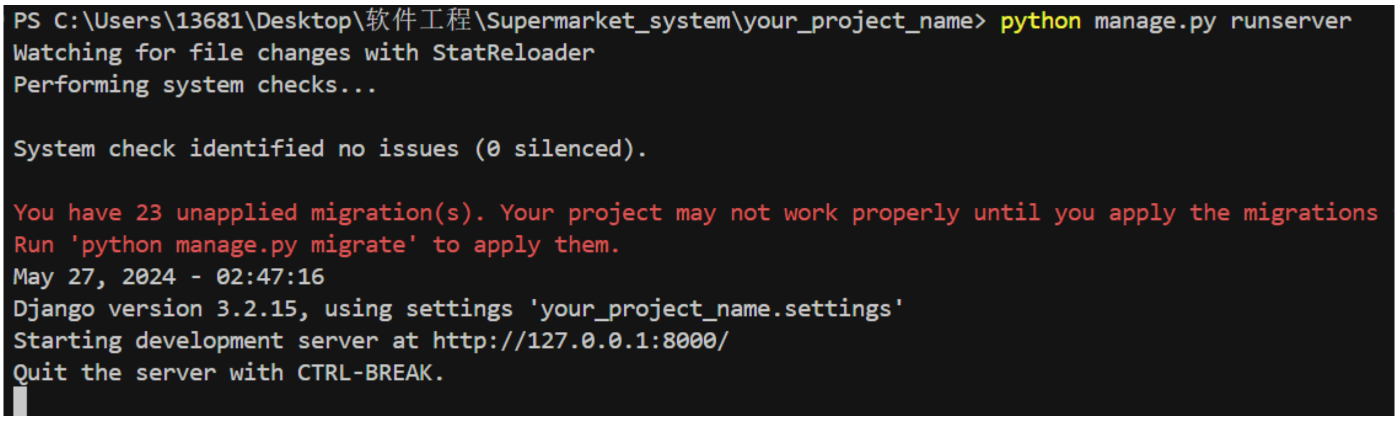
主要安装这3个，如果还有缺失的库根据报错安装即可。

2.3 启动后端

进入your_project_name目录，能看到manage.py 命令行输入

```
python manage.py runserver
```

看到如下结果，即启动成功。

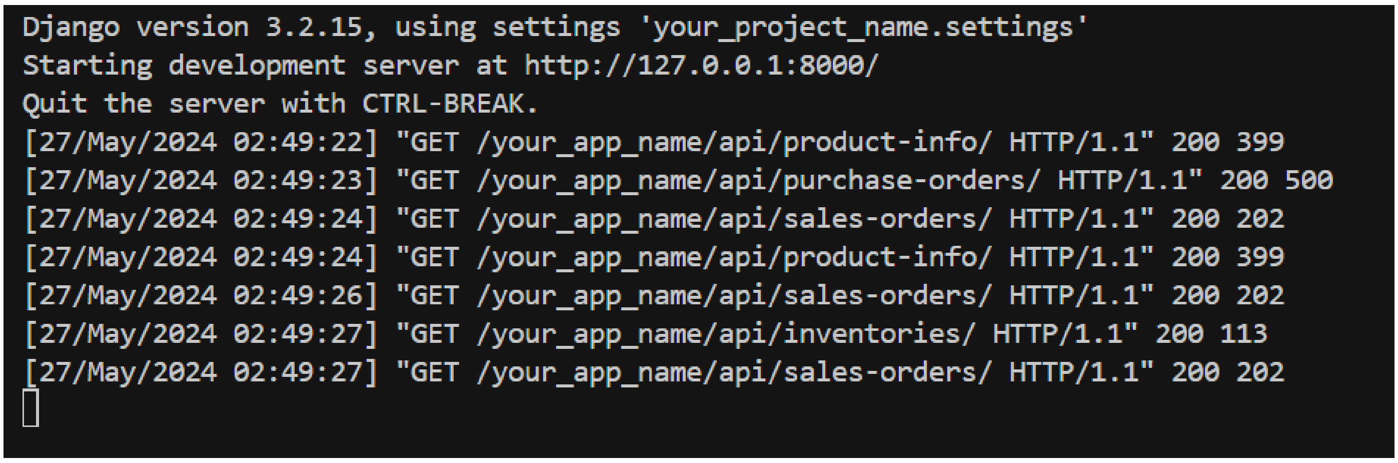


```
PS C:\Users\13681\Desktop\软件工程\Supermarket_system\your_project_name> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 23 unapplied migration(s). Your project may not work properly until you apply the migrations
Run 'python manage.py migrate' to apply them.
May 27, 2024 - 02:47:16
Django version 3.2.15, using settings 'your_project_name.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

如果前后端能够成功运行，应能看到日志输出



```
Django version 3.2.15, using settings 'your_project_name.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[27/May/2024 02:49:22] "GET /your_app_name/api/product-info/ HTTP/1.1" 200 399
[27/May/2024 02:49:23] "GET /your_app_name/api/purchase-orders/ HTTP/1.1" 200 500
[27/May/2024 02:49:24] "GET /your_app_name/api/sales-orders/ HTTP/1.1" 200 202
[27/May/2024 02:49:24] "GET /your_app_name/api/product-info/ HTTP/1.1" 200 399
[27/May/2024 02:49:26] "GET /your_app_name/api/sales-orders/ HTTP/1.1" 200 202
[27/May/2024 02:49:27] "GET /your_app_name/api/inventories/ HTTP/1.1" 200 113
[27/May/2024 02:49:27] "GET /your_app_name/api/sales-orders/ HTTP/1.1" 200 202
```

3.前端配置

直接用我发在群里的修改后的前端代码即可。

4. 关于前后端的修改

主要就是将原本代码里的IP地址替换成127.0.0.1回环地址，等后面搞到稳定的ip后再换成别的吧。