

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Лабораторна робота №2**

з дисципліни **Бази даних і засоби управління**

*на тему: “Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”*

Виконав:

студент ІII курсу

групи КВ-01

Орел І. В.

Перевірив:

Київ – 2022

*Метою роботи* є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

*Загальне завдання* роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

**Відомості про обрану предметну галузь**

**Опис предметної області «Магазини»**

Обрана предметна область передбачає купівлю і продаж товарів з точки зору магазинів. Кожен магазин має хоча б одного робітника. У кожному магазині є декілька відділів. У кожному відділі в певний момент часу можуть знаходитись товари. Кожен товар може належати до багатьох покупок, і водночас покупка може містити декілька товарів.

**Логічна модель (схема) БД «Магазини»**

Логічну модель (схему бази даних) наведено на рисунку 2.

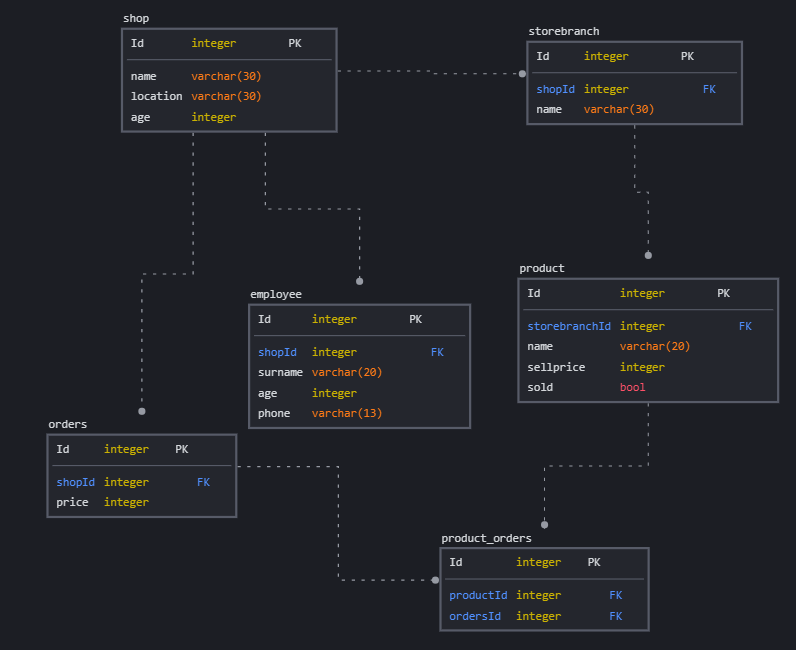


Рисунок 1 - Схема бази даних (інструмент: sqldbm.com)

**Середовище та компоненти розробки**

Для розробки використовувалась мова програмування Python 3.10, середовище розробки PyCharm Community Edition 2022.2.3.

Серед використаних бібліотек:

-бібліотека **Django**, що дозволяє використовувати шаблон проектування МVT;

-бібліотека **psycopg2**, що дозволяє за допомогою мови Python зв’язуватись із СУБД PostgreSQL;

Таким чином, програма реалізована як вебсервер, що використовує **Django**, 1-3 пункти лабораторної реалізовані у форматі веб запитів (http requests).

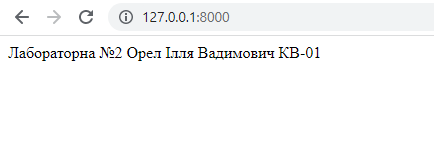


Рисунок 2 - “Homepage” вебсерверу



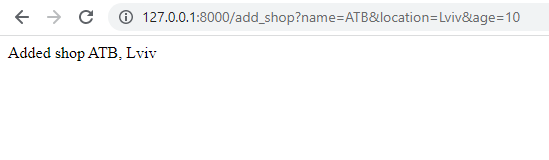
Рисунок 3 - Список валідних веб-запитів

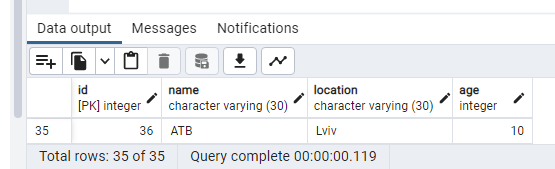
Зі списку видно, що кожній сутності БД “Магазини” відповідає 5 можливих запитів. Запит “add\_...” додає новий рядок до таблиці БД із заданими в запиті параметрами. Запит “show\_…” обирає та виводить задану таблицю БД. Запит “update\_...” змінює інформацію заданого за id рядка в таблиці БД на задану в запиті інформацію. Запит “delete\_...” видаляє інформацію заданого за id рядка з таблиці БД . Запит “generate\_random\_...” створює задану в запиті кількість нових рядків із випадково згенерованими даними(із заданими програмою межами та типами).

Також існує 3 запити “select”, кожен з яких виконує вибір та демонстрацію певних даних за певним заданим програмою алгоритмом з таблиць БД.

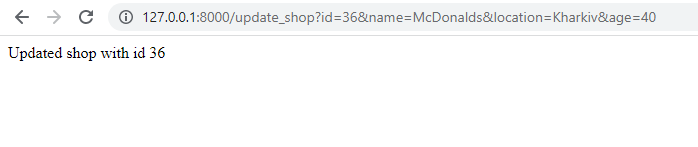
**Завдання 1**

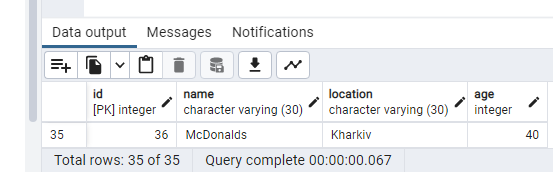
Уведення, редагування, вилучення даних у таблицях БД реалізується за допомогою запитів “add\_...”, “update\_...”, “delete\_...” відповідно. Приклади використання:



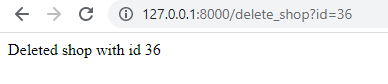


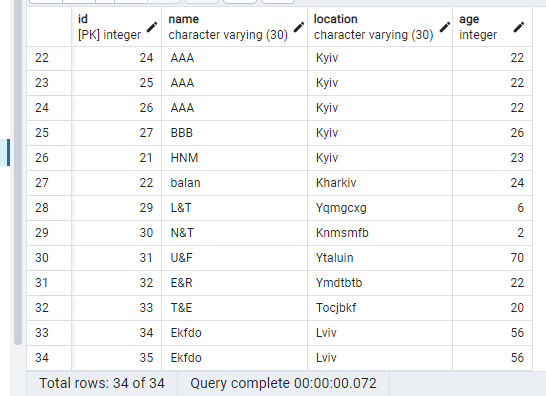
Рисунки 4-5 - Використання команди “add\_shop”





Рисунки 6-87 - Використання команди “update\_shop”





Рисунки 8-9 - Використання команди “delete\_shop”

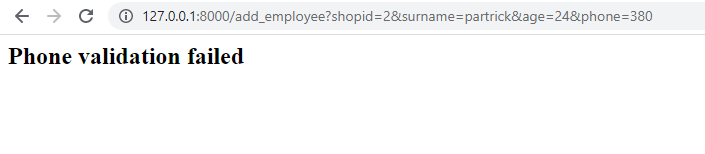
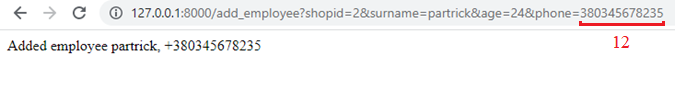


Рисунок 10 - Приклад задіяної валідації даних, отриманих у запиті

У данному випадку від параметру запиту “phone” очікується повернення 12-цифрового числа. Приклад того ж запиту із коректним параметром “phone”:



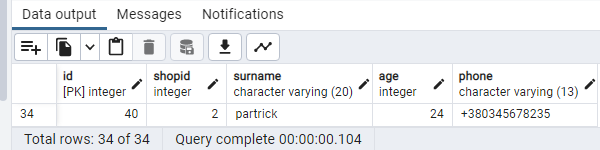


Рисунок 11 – Результат вставки за запитом, що пройшов валідацію

У файлі exceptions.py описано так званий мапінг можливих помилок відносно текстових відповідей, що отримає користувач при їх виникненні. Приклад мапінгу:

PRODUCT\_STOREBRANCHID\_FKEY = "product\_storebranchid\_fkey"  
STOREBRANCHID\_VALIDATION\_FAILED = "storebranchid validation failed"  
SELLPRICE\_VALIDATION\_FAILED= "sellprice validation failed"  
SOLD\_VALIDATION\_FAILED= "sold\_flag validation failed"

def product\_exceptions\_mapper(e):  
 if PRODUCT\_STOREBRANCHID\_FKEY in e.args[0]:  
 return HttpResponseBadRequest("<h2>No such Storebranch</h2>")  
 if STOREBRANCHID\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Storebranchid validation failed</h2>")  
 if SELLPRICE\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Sellprice validation failed</h2>")  
 if SOLD\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Sold\_flag validation failed</h2>")  
 return HttpResponseNotFound("<h2>Product not found</h2>")

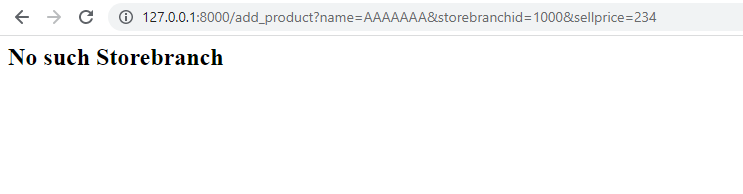
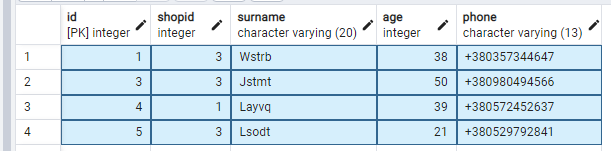
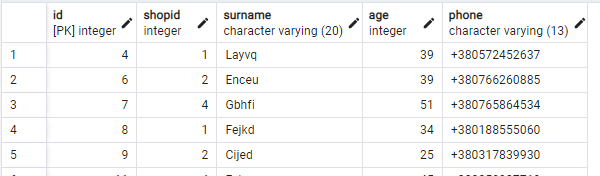


Рисунок 12 - Приклад виведення помилки

Завдяки правилам видалення, заданим при створенні сутностей БД ON DELETE CASCADE(ще в першій лабораторній роботі), помилки із видаленням рядків батьківської таблиці не можуть відбутись. Приклад видалення рядка батьківської таблиці:



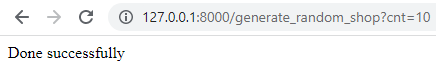




Рисунки 13-15 – Результат видалення рядка батьківської таблиці (в дочірній таблиці каскадно видалилсь рядки із відповідним зовнішнім ключем)

**Завдання 2**

Запити “generate\_random\_...” використовуються для генерування заданної у параметрі запиту “cnt” кількості рядків даних відповідної таблиці БД. Приклад використання:





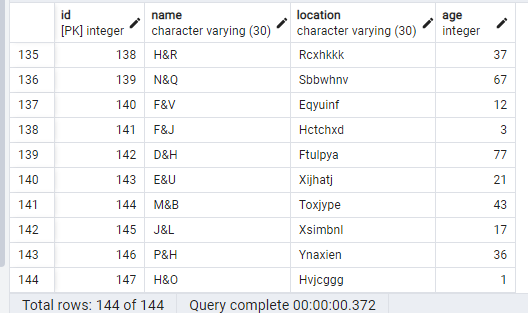
Рисунки 16-17 - Використання команди “ generate\_random \_shop”

Код, застосований за запитом generate\_random\_shop:

def generate\_random(self, cnt):  
 with connection.cursor() as cursor:  
 cursor.execute(f'''do $$  
 begin  
  
 for cnt in 1..{cnt} loop  
 INSERT INTO shop(name, location, age) VALUES(  
  
 chr(trunc(65+random()\*25)::int) || '&' || chr(trunc(65+random()\*25)::int),   
 chr(trunc(65+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int),   
 CAST((random() \* 10) \* (random() \* 15) as INT));  
 end loop;  
 end; $$ ''')

Однак існує недосконалість - при спробі створити рядок дочірньої таблиці із зовнішнім ключем, що відповідає неіснуючому id батьківської таблиці відбувається помилка, тому в даному алгоритмі статично задано обмеження для створення псевдовипадкового зовнішнього ключа.





Рисунки 18-19 - Приклад рандомного генерування великої кількості рядків

**Завдання 3**

Було заготовлено три шаблони об’єднаних вибірок, що викликаються за запитом “Select<номер(1-3)>” після кожного результату вибірки написано час витрачений на виконання запиту в мілісекундах:

Шаблон Select1:

time\_start = current\_milli\_time()  
cursor.execute(f'''SELECT shop.id, shop.name, shop.location, employee.id, employee.age, employee.surname FROM shop   
 INNER JOIN employee ON shop.id=employee.shopid  
 WHERE shop.age > {shop\_age} AND employee.age < {employee\_age}  
 OR employee.surname LIKE '{employee\_surname\_like1}' OR employee.surname LIKE '{employee\_surname\_like2}';  
 ''')  
time\_end = current\_milli\_time()

Із двох з’єднаних таблиць “shop” та “employee” будуть вибрані рядки із первинними ключами магазинів, назвами магазинів, їх місцезнаходженням, первинними ключами їх робітників, їх віком та прізвищами за наступними критеріями: вік магазину більше заданого(50) ТА вік робітника менше заданого(33) АБО прізвище робітника відповідає заданим(partrick, dartrick):



Рисунок 20 - Результат першої шаблонної вибірки

Шаблон Select2:

time\_start = current\_milli\_time()  
 cursor.execute(f'''SELECT storebranch.id, storebranch.name, product.id, product.name, product.sellprice, product.sold FROM storebranch   
 INNER JOIN product ON storebranch.id = product.storebranchid   
 WHERE NOT product.sold AND storebranch.name LIKE '{storebranch\_name\_like}'  
 OR product.sellprice > {product\_sellprice1} AND product.sellprice < {product\_sellprice2};  
 ''')  
 time\_end = current\_milli\_time()

Із двох з’єднаних таблиць “storebranch” та “product” будуть вибрані рядки із первинними ключами відділень магазинів, назвами відділень магазинів, первинними ключами їх товарів, їх назвами, цінами та прапором їх проданності за наступними критеріями: продукт не продано(прапорець проданності продукту “false”) АБО ім’я відділу відповідає заданим(Oybw) ТА ціна продукту більше заданої(30) ТА ціна продукту менше заданої(100).

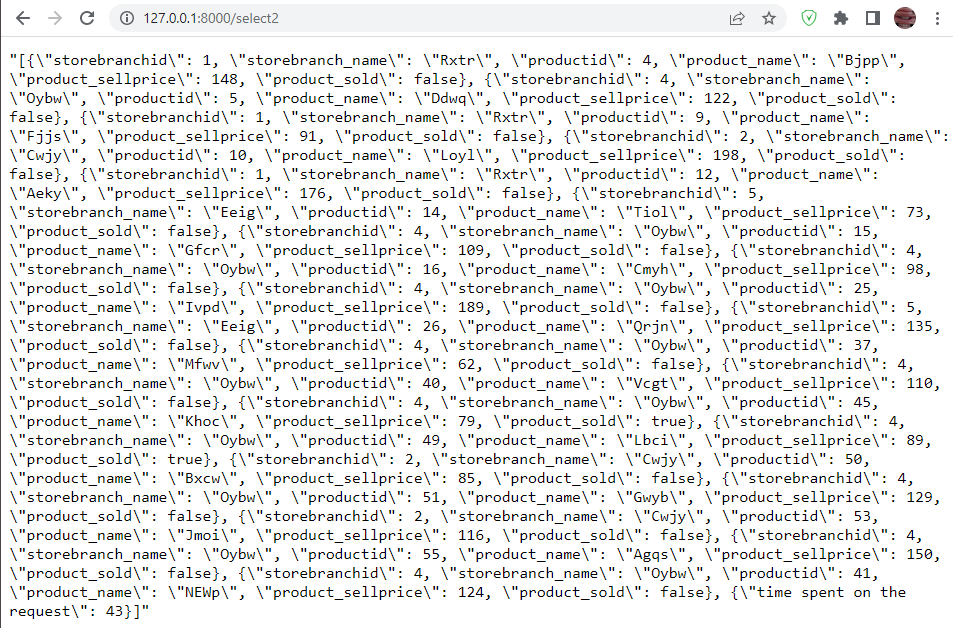


Рисунок 21 - Результат другої шаблонної вибірки

Шаблон Select3:

time\_start = current\_milli\_time()  
 cursor.execute(f'''SELECT orders.id, orders.price, product.id, product.name, product.sellprice FROM product\_orders   
 INNER JOIN orders ON product\_orders.ordersid = orders.id  
 INNER JOIN product ON product\_orders.productid = product.id  
 WHERE orders.price > {orders\_price1} AND orders.price < {orders\_price2};  
 ''')  
 time\_end = current\_milli\_time()

Із трьох з’єднаних таблиць “orders”, “product” та “product\_orders” будуть вибрані рядки із первинними ключами замовлень, їх цінами, первинними ключами відповідних їм товарів, їх назвами та цінами за наступними критеріями: ціна замовлення більше заданої(100) ТА ціна замовлення менше заданої(150).

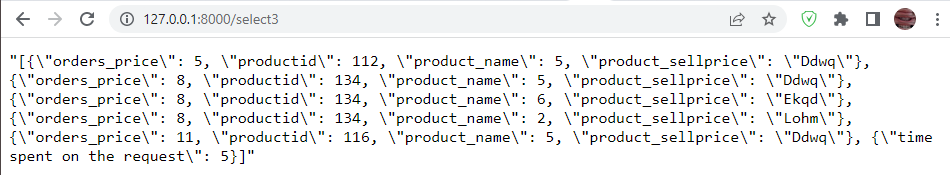


Рисунок 22 - Результат третьої шаблонної вибірки

Кожний із критеріїв фільтрації можна задавати за відповідним йому ім’ям у параметрі запиту. Приклад:

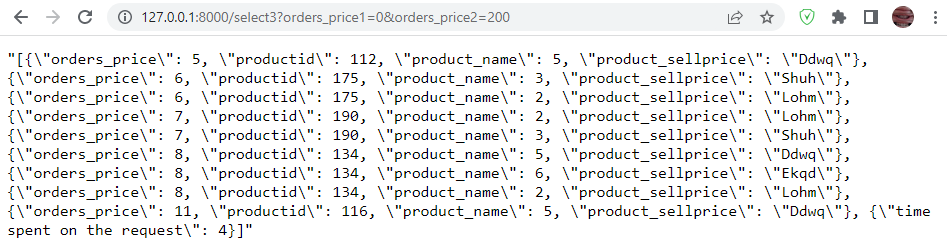


Рисунок 23 - Результат задання критеріїв фільтрації для третього шаблону вибірки

**Завдання 4**

Код організовано згідно шаблону MVT або Model-View-Template, створений для Django. Він є модифікованим шаблоном MVC, в якому Model має те ж значення, View замінено на Template, а Controler на View. Таким чином, Model описує класи, що відповідають таблицям в БД та методи для роботи з ними, View отримує запити та взаємодіє із Model для отримання з неї або з БД даних, Template власне формує html сторінки виходячи з отриманої із View інформації.

**Код програмних модулів:**

**models.py**

import uuid  
from abc import ABC, abstractmethod  
from django.db import models  
from django.db import connection  
import re  
  
# Create your models here.  
  
class Repository(ABC):  
 @abstractmethod  
 def add(self):  
 pass  
  
 @abstractmethod  
 def delete(self, id):  
 pass  
  
 @abstractmethod  
 def update(self, id):  
 pass  
  
 @abstractmethod  
 def show(self, response\_dict):  
 pass  
  
 @abstractmethod  
 def generate\_random(self, cnt):  
 pass  
  
class Product(Repository):  
 def \_\_init\_\_(self, name=None, storebranchid = None, sellprice=None, sold=None):  
 self.name = name  
 self.storebranchid = storebranchid  
 self.sellprice = sellprice  
 self.sold = sold  
  
 def add(self):  
 with connection.cursor() as cursor:  
 try:  
 self.storebranchid = int(self.storebranchid)  
 except:  
 raise Exception("storebranchid validation failed")  
 try:  
 self.sellprice = int(self.sellprice)  
 except:  
 raise Exception("sellprice validation failed")  
  
 cursor.execute(f"INSERT INTO product(name, storebranchid, sellprice) VALUES ('{self.name}', {self.storebranchid}, {self.sellprice})")  
  
 def delete(self, id):  
 with connection.cursor() as cursor:  
 cursor.execute(f"DElETE FROM product WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def update(self, id):  
 with connection.cursor() as cursor:  
  
 if self.name is not None:  
 x\_name = f"name='{self.name}',"  
  
 if self.storebranchid is not None:  
 try:  
 self.storebranchid = int(self.storebranchid)  
 except:  
 raise Exception("storebranchid validation failed")  
 x\_storebranchid = f'storebranchid={self.storebranchid},'  
  
 if self.sellprice is not None:  
 try:  
 self.sellprice = int(self.sellprice)  
 except:  
 raise Exception("sellprice validation failed")  
 x\_sellprice = f'sellprice={self.sellprice},'  
  
 if self.sold is not None:  
 if self.sold == 'false':  
 x\_sold = f'sold=false'  
 elif self.sold == 'true':  
 x\_sold = f'sold=true'  
 else:  
 raise Exception("sold\_flag validation failed")  
  
 print("UPDATE product " + f"SET " + f"{x\_name}" + f'{x\_storebranchid} ' + f'{x\_sellprice}' + f'{x\_sold}'f" WHERE id={id}")  
 cursor.execute(  
 "UPDATE product " + f"SET " + f'{x\_name}' + f'{x\_storebranchid}' + f'{x\_sellprice}' + f'{x\_sold}'f" WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def show(self, response\_dict):  
 with connection.cursor() as cursor:  
 cursor.execute('SELECT \* FROM product')  
 products = cursor.fetchall()  
 print(products)  
 for product in products:  
 temporary\_dict = {"id": product[0], "name": product[1], "storebranchid": product[2], "sellprice": product[3], "sold": product[4]}  
 response\_dict.append(temporary\_dict)  
  
 def generate\_random(self, cnt):  
 with connection.cursor() as cursor:  
 cursor.execute(f'''do $$  
 begin  
 for cnt in 1..{cnt} loop  
 INSERT INTO product(name, storebranchid, sellprice, sold) VALUES(   
 chr(trunc(65+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int),   
 CAST(1 + random() \* 4 as INT),   
 CAST(random() \* 150 + 50 as INT),  
 random() > 0.5);  
   
 end loop;  
 end; $$ ''')  
  
class Orders(Repository):  
 def \_\_init\_\_(self, shopid=None, price=None):  
 self.shopid = shopid  
 self.price = price  
  
 def add(self):  
 with connection.cursor() as cursor:  
 try:  
 self.shopid = int(self.shopid)  
 except:  
 raise Exception("shopid validation failed")  
 try:  
 self.price = int(self.price)  
 except:  
 raise Exception("price validation failed")  
  
 cursor.execute(f"INSERT INTO orders(shopid, price) VALUES ({self.shopid},{self.price})")  
  
 def delete(self, id):  
 with connection.cursor() as cursor:  
 cursor.execute(f"DElETE FROM orders WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def update(self, id):  
 with connection.cursor() as cursor:  
  
 if self.shopid is not None:  
 try:  
 self.shopid = int(self.shopid)  
 except:  
 raise Exception("shopid validation failed")  
  
 x\_shopid = f'shopid={self.shopid},'  
  
 if self.price is not None:  
 try:  
 self.price = int(self.price)  
 except:  
 raise Exception("price validation failed")  
  
 x\_price = f'price={self.price}'  
  
 print("UPDATE orders " + f"SET " + f"{x\_shopid}" + f'{x\_price} ' f" WHERE id={id}")  
 cursor.execute(  
 "UPDATE orders " + f"SET " + f'{x\_shopid}' + f'{x\_price}' f" WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def show(self, response\_dict):  
 with connection.cursor() as cursor:  
 cursor.execute('SELECT \* FROM orders')  
 orderss = cursor.fetchall()  
 print(orderss)  
 for orders in orderss:  
 temporary\_dict = {"id": orders[0], "shopid": orders[1], "price": orders[2]}  
 response\_dict.append(temporary\_dict)  
  
 def generate\_random(self, cnt):  
 with connection.cursor() as cursor:  
 cursor.execute(f'''do $$  
 begin  
 for cnt in 1..{cnt} loop  
   
 INSERT INTO orders(shopid, price) VALUES(   
 CAST(1 + random() \* 4 as INT),   
 CAST(random() \* 150 + 100 as INT));  
   
 end loop;  
 end; $$ ''')  
  
class Storebranch(Repository):  
 def \_\_init\_\_(self, shopid=None, name=None):  
 self.shopid = shopid  
 self.name = name  
  
 def add(self):  
 with connection.cursor() as cursor:  
 try:  
 self.shopid = int(self.shopid)  
 except:  
 raise Exception("shopid validation failed")  
  
 cursor.execute(f"INSERT INTO storebranch(shopid, name) VALUES ({self.shopid},'{self.name}')")  
  
 def delete(self, id):  
 with connection.cursor() as cursor:  
 cursor.execute(f"DElETE FROM storebranch WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def update(self, id):  
 with connection.cursor() as cursor:  
  
 if self.name is not None:  
 x\_name = f"name='{self.name}'"  
  
 if self.shopid is not None:  
 try:  
 self.shopid = int(self.shopid)  
 except:  
 raise Exception("shopid validation failed")  
 x\_shopid = f'shopid={self.shopid},'  
  
 cursor.execute(  
 "UPDATE storebranch " + f"SET " + f'{x\_shopid} ' + f'{x\_name}' f" WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def show(self, response\_dict):  
 with connection.cursor() as cursor:  
 cursor.execute('SELECT \* FROM storebranch')  
 storebranches = cursor.fetchall()  
 print(storebranches)  
 for storebranch in storebranches:  
 temporary\_dict = {"id": storebranch[0], "shopid": storebranch[1], "name": storebranch[2]}  
 response\_dict.append(temporary\_dict)  
  
 def generate\_random(self, cnt):  
 with connection.cursor() as cursor:  
 cursor.execute(f'''do $$  
 begin  
 for cnt in 1..{cnt} loop  
   
 INSERT INTO storebranch(shopid, name) VALUES(   
 CAST(1 + random() \* 4 as INT),   
 chr(trunc(65+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int));  
  
 end loop;  
 end; $$ ''')  
  
class Employee(Repository):  
 def \_\_init\_\_(self, shopid=None, surname=None, age=None, phone=None):  
 self.shopid = shopid  
 self.surname = surname  
 self.age = age  
 self.phone = phone  
  
 def add(self):  
 with connection.cursor() as cursor:  
 try:  
 self.shopid = int(self.shopid)  
 except:  
 raise Exception("shopid validation failed")  
 try:  
 self.age = int(self.age)  
 except:  
 raise Exception("age validation failed")  
  
 if not re.match(r'^[0-9]{12}$', str(self.phone)):  
 raise Exception("phone validation failed")  
 else:  
 self.phone = int(self.phone)  
  
 cursor.execute(f"INSERT INTO employee(shopid, surname, age, phone) VALUES ({self.shopid},'{self.surname}', {self.age}, '+{self.phone}')")  
  
 def delete(self, id):  
 with connection.cursor() as cursor:  
 cursor.execute(f"DElETE FROM employee WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def update(self, id):  
 with connection.cursor() as cursor:  
  
 if self.shopid is not None:  
 try:  
 self.shopid = int(self.shopid)  
 except:  
 raise Exception("shopid validation failed")  
 x\_shopid = f'shopid={self.shopid},'  
  
 if self.surname is not None: x\_surname = f"surname='{self.surname}',"  
 else: raise  
  
 if self.age is not None:  
 try:  
 self.age = int(self.age)  
 except:  
 raise Exception("age validation failed")  
 x\_age = f'age={self.age},'  
  
 if self.phone is not None:  
 if re.match(r'^[0-9]{12}$', str(self.phone)):  
 x\_phone = f"phone='{'+' + str(self.phone)}'"  
 else: raise Exception("phone validation failed")  
  
 cursor.execute(  
 "UPDATE employee " + f"SET " + f'{x\_shopid} ' + f'{x\_surname} ' + f'{x\_age}' + f'{x\_phone}' f" WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def show(self, response\_dict):  
 with connection.cursor() as cursor:  
 cursor.execute('SELECT \* FROM employee')  
 employees = cursor.fetchall()  
 print(employees)  
 for employee in employees:  
 temporary\_dict = {"id": employee[0], "shopid": employee[1], "surname": employee[2], "age": employee[3], "phone": employee[4]}  
 response\_dict.append(temporary\_dict)  
  
 def generate\_random(self, cnt):  
 with connection.cursor() as cursor:  
 cursor.execute(f'''do $$  
 begin  
 for cnt in 1..{cnt} loop  
 INSERT INTO employee(shopid, surname, age, phone) VALUES(  
  
 CAST(1 + random() \* 4 as INT),   
 chr(trunc(65+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int),   
 CAST(random() \* 40 + 18 as INT),   
 '+380' || CAST(100000000+random()\*899999999 AS int));  
  
 end loop;  
 end; $$ ''')  
  
class Shop(Repository):  
 def \_\_init\_\_(self, name=None, location=None, age=None):  
 self.name = name  
 self.location = location  
 self.age = age  
  
 def add(self):  
 with connection.cursor() as cursor:  
 try:  
 self.age = int(self.age)  
 except:  
 raise Exception("age validation failed")  
  
 if self.location is not None:  
 cursor.execute(f"INSERT INTO shop(name, location, age) VALUES ('{self.name}','{self.location}', {self.age})")  
 else:  
 cursor.execute(f"INSERT INTO shop(name, age) VALUES ('{self.name}', {self.age})")  
  
 def delete(self, id):  
 with connection.cursor() as cursor:  
 cursor.execute(f"DElETE FROM shop WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def update(self, id):  
 with connection.cursor() as cursor:  
  
 if self.name is not None:  
 x\_name = f"name='{self.name}',"  
  
 if self.location is not None:  
 x\_location = f"location='{self.location}',"  
  
 if self.age is not None:  
 try:  
 self.age = int(self.age)  
 except:  
 raise Exception("age validation failed")  
 x\_age = f'age={self.age}'  
  
 cursor.execute(  
 "UPDATE shop " + f"SET " + f'{x\_name} ' + f'{x\_location} ' + f'{x\_age}' f" WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def show(self, response\_dict):  
 with connection.cursor() as cursor:  
 cursor.execute('SELECT \* FROM shop')  
 shops = cursor.fetchall()  
 print(shops)  
 for shop in shops:  
 temporary\_dict = {"id": shop[0], "name": shop[1], "location": shop[2], "age": shop[3]}  
 response\_dict.append(temporary\_dict)  
  
  
  
 def generate\_random(self, cnt):  
 with connection.cursor() as cursor:  
 cursor.execute(f'''do $$  
 begin  
  
 for cnt in 1..{cnt} loop  
 INSERT INTO shop(name, location, age) VALUES(  
  
 chr(trunc(65+random()\*25)::int) || '&' || chr(trunc(65+random()\*25)::int),   
 chr(trunc(65+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int) || chr(trunc(97+random()\*25)::int),   
 CAST((random() \* 10) \* (random() \* 15) as INT));  
 end loop;  
 end; $$ ''')

**main\_views.py**

from django.http import HttpResponse  
  
  
def index(request):  
 return HttpResponse("Лабораторна №2 Орел Ілля Вадимович КВ-01")

**exceptions.py**

from django.http import HttpResponseBadRequest, HttpResponseNotFound  
  
PRODUCT\_STOREBRANCHID\_FKEY = "product\_storebranchid\_fkey"  
STOREBRANCHID\_VALIDATION\_FAILED = "storebranchid validation failed"  
SELLPRICE\_VALIDATION\_FAILED= "sellprice validation failed"  
SOLD\_VALIDATION\_FAILED= "sold\_flag validation failed"  
ORDERS\_SHOPID\_FKEY= "orders\_shopid\_fkey"  
SHOPID\_VALIDATION\_FAILED= "shopid validation failed"  
PRICE\_VALIDATION\_FAILED= "price validation failed"  
STOREBRANCH\_SHOPID\_FKEY= "storebranch\_shopid\_fkey"  
EMPLOYEE\_SHOPID\_FKEY= "employee\_shopid\_fkey"  
AGE\_VALIDATION\_FAILED = "age validation failed"  
PHONE\_VALIDATION\_FAILED= "phone validation failed"  
  
def product\_exceptions\_mapper(e):  
 if PRODUCT\_STOREBRANCHID\_FKEY in e.args[0]:  
 return HttpResponseBadRequest("<h2>No such Storebranch</h2>")  
 if STOREBRANCHID\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Storebranchid validation failed</h2>")  
 if SELLPRICE\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Sellprice validation failed</h2>")  
 if SOLD\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Sold\_flag validation failed</h2>")  
 return HttpResponseNotFound("<h2>Product not found</h2>")  
  
def orders\_exceptions\_mapper(e):  
 print(e)  
 if ORDERS\_SHOPID\_FKEY in e.args[0]:  
 return HttpResponseBadRequest("<h2>No such Shop</h2>")  
 if SHOPID\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Shopid validation failed</h2>")  
 if PRICE\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Price validation failed</h2>")  
 return HttpResponseNotFound("<h2>Orders not found</h2>")  
  
def storebranch\_exceptions\_mapper(e):  
 print(e)  
 if STOREBRANCH\_SHOPID\_FKEY in e.args[0]:  
 return HttpResponseBadRequest("<h2>No such Shop</h2>")  
 if SHOPID\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Shopid validation failed</h2>")  
 return HttpResponseNotFound("<h2>Storebranch not found</h2>")  
  
def employee\_exceptions\_mapper(e):  
 print(e)  
 if EMPLOYEE\_SHOPID\_FKEY in e.args[0]:  
 return HttpResponseBadRequest("<h2>No such Shop</h2>")  
 if SHOPID\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Shopid validation failed</h2>")  
 if AGE\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Age validation failed</h2>")  
 if PHONE\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Phone validation failed</h2>")  
 return HttpResponseNotFound("<h2>Employee not found</h2>")  
  
def shop\_exceptions\_mapper(e):  
 print(e)  
 if AGE\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Age validation failed</h2>")  
 return HttpResponseNotFound("<h2>Shop not found</h2>")

**employee\_views.py**

from django.http import HttpResponse, HttpResponseBadRequest, HttpResponseNotFound, JsonResponse  
import json  
  
from Lab2.exceptions import employee\_exceptions\_mapper  
from Lab2.models import Employee  
  
  
def add\_employee(request):  
 if request.method == 'GET':  
 try:  
 shopid = request.GET.get('shopid')  
 surname = request.GET.get("surname")  
 age = request.GET.get("age")  
 phone = request.GET.get("phone")  
 employee = Employee(shopid, surname, age, phone)  
 employee.add()  
  
 return HttpResponse(f"Added employee {surname}, +{phone}")  
 except Exception as e:  
 return employee\_exceptions\_mapper(e)  
 return HttpResponseBadRequest("Bad Request")  
  
  
def delete\_employee(request):  
 if request.method == 'GET':  
 try:  
 id = request.GET.get("id")  
 if id is None:  
 raise  
 employee = Employee()  
 employee.delete(id)  
  
 return HttpResponse(f"Deleted employee with id {id}")  
 except:  
 return HttpResponseNotFound("<h2>Employee not found</h2>")  
 return HttpResponseBadRequest("Bad Request")  
  
  
def update\_employee(request):  
 if request.method == "GET":  
 try:  
 id = request.GET.get("id")  
 if id is None:  
 raise  
 shopid = request.GET.get('shopid')  
 surname = request.GET.get("surname")  
 age = request.GET.get("age")  
 phone = request.GET.get("phone")  
 employee = Employee(shopid, surname, age, phone)  
 employee.update(id)  
  
 return HttpResponse(f"Updated employee with id {id}")  
 except Exception as e:  
 return employee\_exceptions\_mapper(e)  
 return HttpResponseBadRequest("Bad Request")  
  
  
def show\_employee(request):  
 response\_dict = []  
 employee = Employee()  
 employee.show(response\_dict)  
 return JsonResponse(json.dumps(response\_dict), safe=False)

**orders\_views.py**

from django.http import HttpResponse, HttpResponseBadRequest, HttpResponseNotFound, JsonResponse  
import json  
  
from Lab2.exceptions import orders\_exceptions\_mapper  
from Lab2.models import Orders  
  
  
def add\_orders(request):  
 if request.method == 'GET':  
 try:  
 shopid = request.GET.get('shopid')  
 price = request.GET.get("price")  
 orders = Orders(shopid, price)  
 orders.add()  
 return HttpResponse(f"Added order {price} UAH, Shopid: {shopid}")  
 except Exception as e:  
 return orders\_exceptions\_mapper(e)  
 return HttpResponseBadRequest("Bad Request")  
  
  
def delete\_orders(request):  
 if request.method == 'GET':  
 try:  
 id = request.GET.get("id")  
 if id is None:  
 raise  
 orders = Orders()  
 orders.delete(id)  
 return HttpResponse(f"Deleted orders with id {id}")  
 except:  
 return HttpResponseNotFound("<h2>Order not found</h2>")  
 return HttpResponseBadRequest("Bad Request")  
  
  
def update\_orders(request):  
 if request.method == "GET":  
 try:  
 id = request.GET.get("id")  
 if id is None:  
 raise  
 shopid = request.GET.get('shopid')  
 price = request.GET.get("price")  
 orders = Orders(shopid, price)  
 orders.update(id)  
  
 return HttpResponse(f"Updated order with id {id}")  
 except Exception as e:  
 return orders\_exceptions\_mapper(e)  
 return HttpResponseBadRequest("Bad Request")  
  
  
def show\_orders(request):  
 response\_dict = []  
 orders = Orders()  
 orders.show(response\_dict)  
  
 return JsonResponse(json.dumps(response\_dict), safe=False)

**product\_views.py**

from django.http import HttpResponse, HttpResponseBadRequest, HttpResponseNotFound, JsonResponse  
import json  
  
from Lab2.exceptions import product\_exceptions\_mapper  
from Lab2.models import Product  
  
  
def add\_product(request):  
 if request.method == 'GET':  
 try:  
 name = request.GET.get("name")  
 storebranchid = request.GET.get('storebranchid')  
 sellprice = request.GET.get("sellprice")  
 sold = request.GET.get("sold")  
 product = Product(name, storebranchid, sellprice, sold)  
 product.add()  
 return HttpResponse(f"Added product {name} - {sellprice} UAH")  
 except Exception as e:  
 return product\_exceptions\_mapper(e)  
 return HttpResponseBadRequest("Bad Request")  
  
  
def delete\_product(request):  
 if request.method == 'GET':  
 try:  
 id = request.GET.get("id")  
 if id is None:  
 raise  
 product = Product()  
 product.delete(id)  
 return HttpResponse(f"Deleted product with id {id}")  
 except:  
 return HttpResponseNotFound("<h2>Product not found</h2>")  
 return HttpResponseBadRequest("Bad Request")  
  
  
def update\_product(request):  
 if request.method == "GET":  
 try:  
 id = request.GET.get("id")  
 if id is None:  
 raise  
 name = request.GET.get("name")  
 storebranchid = request.GET.get('storebranchid')  
 sellprice = request.GET.get("sellprice")  
 sold = request.GET.get("sold")  
 product = Product(name, storebranchid, sellprice, sold)  
 product.update(id)  
  
 return HttpResponse(f"Updated product with id {id}")  
 except Exception as e:  
 return product\_exceptions\_mapper(e)  
 return HttpResponseBadRequest("Bad Request")  
  
  
def show\_product(request):  
 response\_dict = []  
 product = Product()  
 product.show(response\_dict)  
  
 return JsonResponse(json.dumps(response\_dict), safe=False)

**random\_data\_generator\_views.py**

from django.http import HttpResponse  
from Lab2.models import Product, Orders, Employee, Storebranch, Shop  
  
  
def generate\_random\_product(request):  
 if request.method == "GET":  
 cnt = request.GET.get("cnt")  
 product = Product()  
 product.generate\_random(cnt)  
 return HttpResponse('Done successfully')  
  
def generate\_random\_orders(request):  
 if request.method == "GET":  
 cnt = request.GET.get("cnt")  
 orders = Orders()  
 orders.generate\_random(cnt)  
 return HttpResponse('Done successfully')  
  
  
def generate\_random\_employee(request):  
 if request.method == "GET":  
 cnt = request.GET.get("cnt")  
 employee = Employee()  
 employee.generate\_random(cnt)  
 return HttpResponse('Done successfully')  
  
def generate\_random\_storebranch(request):  
 if request.method == "GET":  
 cnt = request.GET.get("cnt")  
 storebranch = Storebranch()  
 storebranch.generate\_random(cnt)  
 return HttpResponse('Done successfully')  
  
  
def generate\_random\_shop(request):  
 if request.method == "GET":  
 cnt = request.GET.get("cnt")  
 shop = Shop()  
 shop.generate\_random(cnt)  
 return HttpResponse('Done successfully')

**selection\_views.py**

import time  
  
from django.http import HttpResponse, HttpResponseBadRequest, HttpResponseNotFound, JsonResponse  
import json  
from django.db import connection  
  
  
def current\_milli\_time():  
 return round(time.time() \* 1000)  
  
  
def select\_from\_db\_1(request):  
 if request.method == 'GET':  
 response\_dict = []  
 employee\_age = request.GET.get("employee\_age", 33)  
 shop\_age = request.GET.get("shop\_age", 50)  
 employee\_surname\_like1 = request.GET.get("employee\_name\_like1", 'partrick')  
 employee\_surname\_like2 = request.GET.get("employee\_name\_like2", 'dartrick')  
  
 with connection.cursor() as cursor:  
 time\_start = current\_milli\_time()  
 cursor.execute(f'''SELECT shop.id, shop.name, shop.location, employee.id, employee.age, employee.surname FROM shop   
 INNER JOIN employee ON shop.id=employee.shopid  
 WHERE shop.age > {shop\_age} AND employee.age < {employee\_age}  
 OR employee.surname LIKE '{employee\_surname\_like1}' OR employee.surname LIKE '{employee\_surname\_like2}';  
 ''')  
 time\_end = current\_milli\_time()  
 results = cursor.fetchall()  
 print(results)  
 for result in results:  
 temporary\_dict = {"shopid": result[0], "shop\_name": result[1], "shop\_location": result[2], "employeeid": result[3], "employee\_age": result[4], "employee\_surname": result[5]}  
 response\_dict.append(temporary\_dict)  
 response\_dict.append({'time spent on the request': time\_end - time\_start})  
 return JsonResponse(json.dumps(response\_dict), safe=False)  
  
  
def select\_from\_db\_2(request):  
 if request.method == 'GET':  
 response\_dict = []  
 product\_sellprice1 = request.GET.get("product\_sellprice1", 30)  
 product\_sellprice2 = request.GET.get("product\_sellprice2", 100)  
 storebranch\_name\_like = request.GET.get("storebranch\_name\_like", 'Oybw')  
 with connection.cursor() as cursor:  
 time\_start = current\_milli\_time()  
 cursor.execute(f'''SELECT storebranch.id, storebranch.name, product.id, product.name, product.sellprice, product.sold FROM storebranch   
 INNER JOIN product ON storebranch.id = product.storebranchid   
 WHERE NOT product.sold OR storebranch.name LIKE '{storebranch\_name\_like}'  
 AND product.sellprice > {product\_sellprice1} AND product.sellprice < {product\_sellprice2};  
 ''')  
 time\_end = current\_milli\_time()  
 results = cursor.fetchall()  
 print(results)  
 for result in results:  
 temporary\_dict = {"storebranchid": result[0], "storebranch\_name": result[1], "productid": result[2], "product\_name": result[3], "product\_sellprice": result[4], "product\_sold": result[5]}  
 response\_dict.append(temporary\_dict)  
 response\_dict.append({'time spent on the request': time\_end - time\_start})  
 return JsonResponse(json.dumps(response\_dict), safe=False)  
  
  
def select\_from\_db\_3(request):  
 if request.method == 'GET':  
 response\_dict = []  
 orders\_price1 = request.GET.get("orders\_price1", 100)  
 orders\_price2 = request.GET.get("orders\_price2", 150)  
 with connection.cursor() as cursor:  
 time\_start = current\_milli\_time()  
 cursor.execute(f'''SELECT orders.id, orders.price, product.id, product.name, product.sellprice FROM product\_orders   
 INNER JOIN orders ON product\_orders.ordersid = orders.id  
 INNER JOIN product ON product\_orders.productid = product.id  
 WHERE orders.price > {orders\_price1} AND orders.price < {orders\_price2};  
 ''')  
 time\_end = current\_milli\_time()  
 results = cursor.fetchall()  
 print(results)  
 for result in results:  
 temporary\_dict = {"orders\_price": result[0], "productid": result[1], "product\_name": result[2], "product\_sellprice": result[3]}  
 response\_dict.append(temporary\_dict)  
 response\_dict.append({'time spent on the request': time\_end - time\_start})  
 return JsonResponse(json.dumps(response\_dict), safe=False)

**shop\_views.py**

from django.http import HttpResponse, HttpResponseBadRequest, HttpResponseNotFound, JsonResponse  
import json  
  
from Lab2.exceptions import shop\_exceptions\_mapper  
from Lab2.models import Shop  
  
  
def add\_shop(request):  
 if request.method == 'GET':  
 try:  
 name = request.GET.get("name")  
 location = request.GET.get("location")  
 age = request.GET.get("age")  
 shop = Shop(name, location, age)  
 shop.add()  
  
 if location is not None:  
 return HttpResponse(f"Added shop {name}, {location}")  
 else:  
 return HttpResponse(f"Added shop {name}, Kyiv")  
 except Exception as e:  
 return shop\_exceptions\_mapper(e)  
 return HttpResponseBadRequest("Bad Request")  
  
  
def delete\_shop(request):  
 if request.method == 'GET':  
 try:  
 id = request.GET.get("id")  
 if id is None:  
 raise  
 shop = Shop()  
 shop.delete(id)  
 return HttpResponse(f"Deleted shop with id {id}")  
 except:  
 return HttpResponseNotFound("<h2>Shop not found</h2>")  
 return HttpResponseBadRequest("Bad Request")  
  
  
def update\_shop(request):  
 if request.method == "GET":  
 try:  
 id = request.GET.get("id")  
 if id is None:  
 raise  
 name = request.GET.get("name")  
 location = request.GET.get("location")  
 age = request.GET.get("age")  
 shop = Shop(name, location, age)  
 shop.update(id)  
  
 return HttpResponse(f"Updated shop with id {id}")  
 except Exception as e:  
 return shop\_exceptions\_mapper(e)  
 return HttpResponseBadRequest("Bad Request")  
  
  
def show\_shop(request):  
 response\_dict = []  
 shop = Shop()  
 shop.show(response\_dict)  
 return JsonResponse(json.dumps(response\_dict), safe=False)

**storebranch\_views.py**

from django.http import HttpResponse, HttpResponseBadRequest, HttpResponseNotFound, JsonResponse  
import json  
from Lab2.models import Storebranch  
  
from Lab2.exceptions import storebranch\_exceptions\_mapper  
  
  
def add\_storebranch(request):  
 if request.method == 'GET':  
 try:  
 shopid = request.GET.get("shopid")  
 name = request.GET.get("name")  
 storebranch = Storebranch(shopid, name)  
 storebranch.add()  
 return HttpResponse(f"Added storebranch {shopid}, {name}")  
 except Exception as e:  
 return storebranch\_exceptions\_mapper(e)  
 return HttpResponseBadRequest("Bad Request")  
  
  
def delete\_storebranch(request):  
 if request.method == 'GET':  
 try:  
 id = request.GET.get("id")  
 if id is None:  
 raise  
 storebranch = Storebranch()  
 storebranch.delete(id)  
 return HttpResponse(f"Deleted storebranch with id {id}")  
 except:  
 return HttpResponseNotFound("<h2>Storebranch not found</h2>")  
 return HttpResponseBadRequest("Bad Request")  
  
  
def update\_storebranch(request):  
 if request.method == "GET":  
 try:  
 id = request.GET.get("id")  
 if id is None:  
 raise  
 shopid = request.GET.get("shopid")  
 name = request.GET.get("name")  
 storebranch = Storebranch(shopid, name)  
 storebranch.update(id)  
 return HttpResponse(f"Updated storebranch with id {id}")  
 except Exception as e:  
 return storebranch\_exceptions\_mapper(e)  
 return HttpResponseBadRequest("Bad Request")  
  
  
def show\_storebranch(request):  
 response\_dict = []  
 storebranch = Storebranch()  
 storebranch.show(response\_dict)  
  
 return JsonResponse(json.dumps(response\_dict), safe=False)

**urls.py**

*"""DjangoProject1 URL Configuration  
  
The `urlpatterns` list routes URLs to views. For more information please see:  
 https://docs.djangoproject.com/en/4.1/topics/http/urls/  
Examples:  
Function views  
 1. Add an import: from my\_app import views  
 2. Add a URL to urlpatterns: path('', views.home, name='home')  
Class-based views  
 1. Add an import: from other\_app.views import Home  
 2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')  
Including another URLconf  
 1. Import the include() function: from django.urls import include, path  
 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))  
"""*from django.urls import path  
  
from Lab2.views import storebranch\_views as storebranch\_views, shop\_views as shop\_views, \  
 random\_data\_generator\_views as random\_data\_generator\_views, product\_views as product\_views, \  
 selection\_views as selection\_views, employee\_views as employee\_views, orders\_views as orders\_views  
  
from Lab2 import main\_views as main\_views  
  
urlpatterns = [  
 path('', main\_views.index),  
 path('add\_product', product\_views.add\_product),  
 path('show\_product', product\_views.show\_product),  
 path('update\_product', product\_views.update\_product),  
 path('delete\_product', product\_views.delete\_product),  
 path('generate\_random\_product', random\_data\_generator\_views.generate\_random\_product),  
 path('add\_orders', orders\_views.add\_orders),  
 path('show\_orders', orders\_views.show\_orders),  
 path('update\_orders', orders\_views.update\_orders),  
 path('delete\_orders', orders\_views.delete\_orders),  
 path('generate\_random\_orders', random\_data\_generator\_views.generate\_random\_orders),  
 path('add\_storebranch', storebranch\_views.add\_storebranch),  
 path('show\_storebranch', storebranch\_views.show\_storebranch),  
 path('update\_storebranch', storebranch\_views.update\_storebranch),  
 path('delete\_storebranch', storebranch\_views.delete\_storebranch),  
 path('generate\_random\_storebranch', random\_data\_generator\_views.generate\_random\_storebranch),  
 path('add\_employee', employee\_views.add\_employee),  
 path('show\_employee', employee\_views.show\_employee),  
 path('update\_employee', employee\_views.update\_employee),  
 path('delete\_employee', employee\_views.delete\_employee),  
 path('generate\_random\_employee', random\_data\_generator\_views.generate\_random\_employee),  
 path('add\_shop', shop\_views.add\_shop),  
 path('show\_shop', shop\_views.show\_shop),  
 path('update\_shop', shop\_views.update\_shop),  
 path('delete\_shop', shop\_views.delete\_shop),  
 path('generate\_random\_shop', random\_data\_generator\_views.generate\_random\_shop),  
 path('select1', selection\_views.select\_from\_db\_1),  
 path('select2', selection\_views.select\_from\_db\_2),  
 path('select3', selection\_views.select\_from\_db\_3),  
]

**settings.py**

*"""  
Django settings for DjangoProject1 project.  
  
Generated by 'django-admin startproject' using Django 4.1.3.  
  
For more information on this file, see  
https://docs.djangoproject.com/en/4.1/topics/settings/  
  
For the full list of settings and their values, see  
https://docs.djangoproject.com/en/4.1/ref/settings/  
"""*from pathlib import Path  
  
# Build paths inside the project like this: BASE\_DIR / 'subdir'.  
BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent  
  
  
# Quick-start development settings - unsuitable for production  
# See https://docs.djangoproject.com/en/4.1/howto/deployment/checklist/  
  
# SECURITY WARNING: keep the secret key used in production secret!  
SECRET\_KEY = 'django-insecure-05t47)^hb#22q8wpg8mh^ciu^4jg$lttbwy+fi1a4kaox)tzj^'  
  
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = True  
  
ALLOWED\_HOSTS = []  
  
  
# Application definition  
  
INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'Lab2',  
]  
  
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',  
]  
  
ROOT\_URLCONF = 'DjangoProject.urls'  
  
TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': [],  
 'APP\_DIRS': True,  
 'OPTIONS': {  
 'context\_processors': [  
 'django.template.context\_processors.debug',  
 'django.template.context\_processors.request',  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = 'DjangoProject.wsgi.application'  
  
  
# Database  
# https://docs.djangoproject.com/en/4.1/ref/settings/#databases  
  
DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.postgresql',  
 'NAME': 'lab1',  
 'USER': 'postgres',  
 'PASSWORD': 'orel5299',  
 'HOST': 'localhost',  
 'PORT': '',  
 }  
}  
  
  
# Password validation  
# https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validators  
  
AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',  
 },  
]  
  
  
# Internationalization  
# https://docs.djangoproject.com/en/4.1/topics/i18n/  
  
LANGUAGE\_CODE = 'en-us'  
  
TIME\_ZONE = 'UTC'  
  
USE\_I18N = True  
  
USE\_TZ = True  
  
  
# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/4.1/howto/static-files/  
  
STATIC\_URL = 'static/'  
  
# Default primary key field type  
# https://docs.djangoproject.com/en/4.1/ref/settings/#default-auto-field  
  
DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'