

**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**

**Виконав:**

**студент ІII курсу**

**групи КВ-01**

**Свисюк Б. О.**

**Перевірив:**

**Київ – 2022**

**Постановка задачі**

*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 рамок діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамок діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

**Відомості про обрану предметну галузь з лабораторної роботи №1**

**Опис предметної області «IT-компанія»**

Обрана предметна область передбачає існування фірм та їх власників, працівників які працюють на певну фірму, проектів які належать певній фірмі, а також інформацію про те, який працівник на яких проектах працює.

**Опис сутностей предметної області**

1) Company (компанія), з атрибутами:код фірми**,** назва фірми,дата заснування фірми**.** Компанія, має власника, та містить проєкти з працівниками на них.

2) Owner (власник), з атрибутами: Код власника, Код компанії, якою він володіє, Ім’я власника, Вік власника. Описує власника компанії.

3) Project (проект), з атрибутами: Код проекту, Код компанії, до якої належить, Ім’я проекту, Скільки проект існує (його вік). Описує проект компанії.

4) Employee(працівник), з атрибутами: Код працівника, Код компанії, до якої належить, Ім’я працівника, Вік працівника. Описує працівника компанії.

**Опис зв’язків між сутностями предметної області**

1) Сутність Company має зв’язок 1 до 1 з таблицею Owner, а також 1(Company) до багатьох з таблицями Project та Employee, для всіх є головною.

2) Сутність Owner має зв’язок 1 до 1 з таблицею Company (за допомогою того, що у сутності Owner ід компанії є унікальним і не дозволяє повторів), є залежною від Company.

3) Сутність Project має зв’язок багато до одного з таблицею Company та є залежною від Company, а також має зв’язок багато до багатьох з сутністю Employee.

4) Сутність Employee має зв’язок багато до одного з таблицею Company, є залежною від Company, а також має зв’язок багато до багатьох з сутністю Project.

**Концептуальна модель предметної області «IT-компанія»**

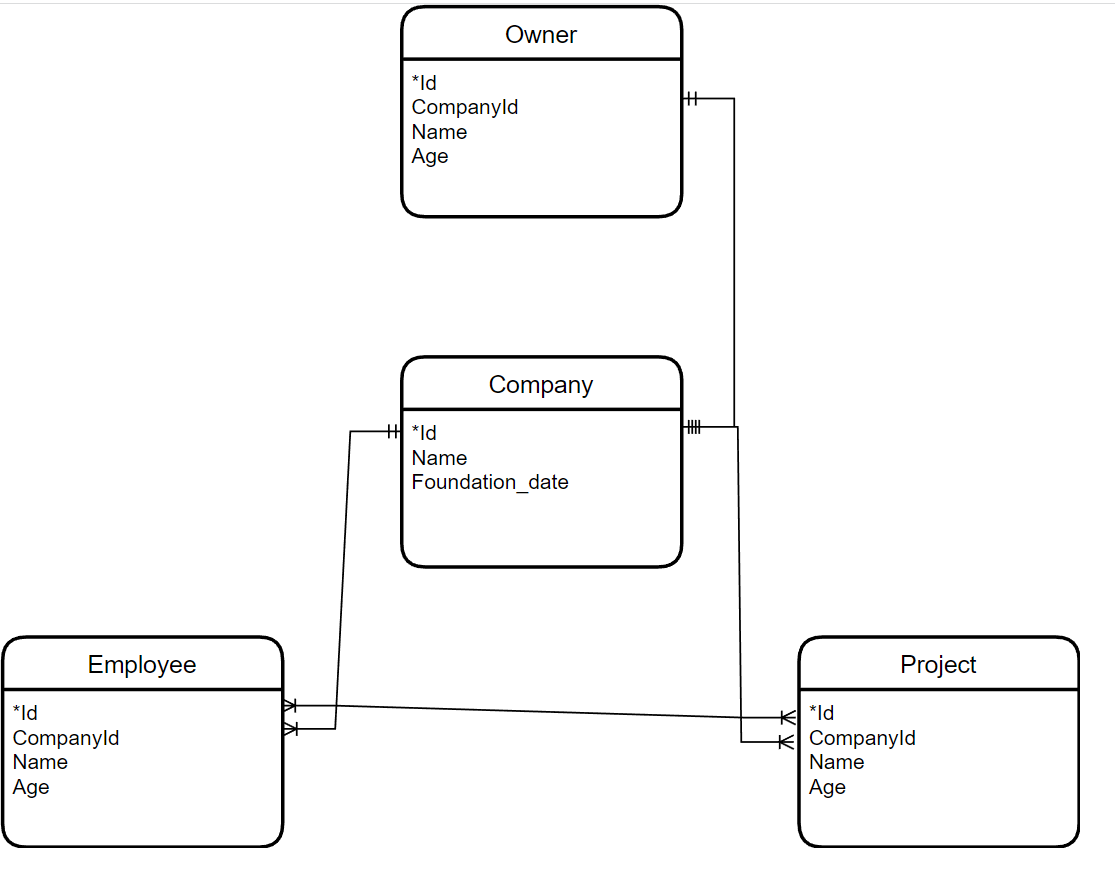
Концептуальна модель наведена на рисунку 1.

Рисунок 1 - ER-діаграма, побудована за нотацією Crow’s feet (інструмент: draw.io)

**Логічна модель (схема) БД «IT-компанія»**

Логічну модель (схему бази даних наведено на рисунку 2.

1)Сутність Company було перетворено в таблицю з відповідною назвою Company.

2)Сутність Owner було перетворено в таблицю з відповідною назвою Owner.

3)Сутність Project було перетворено в таблицю з відповідною назвою Project.

4)Сутність Employee було перетворено в таблицю з відповідною назвою Employee.

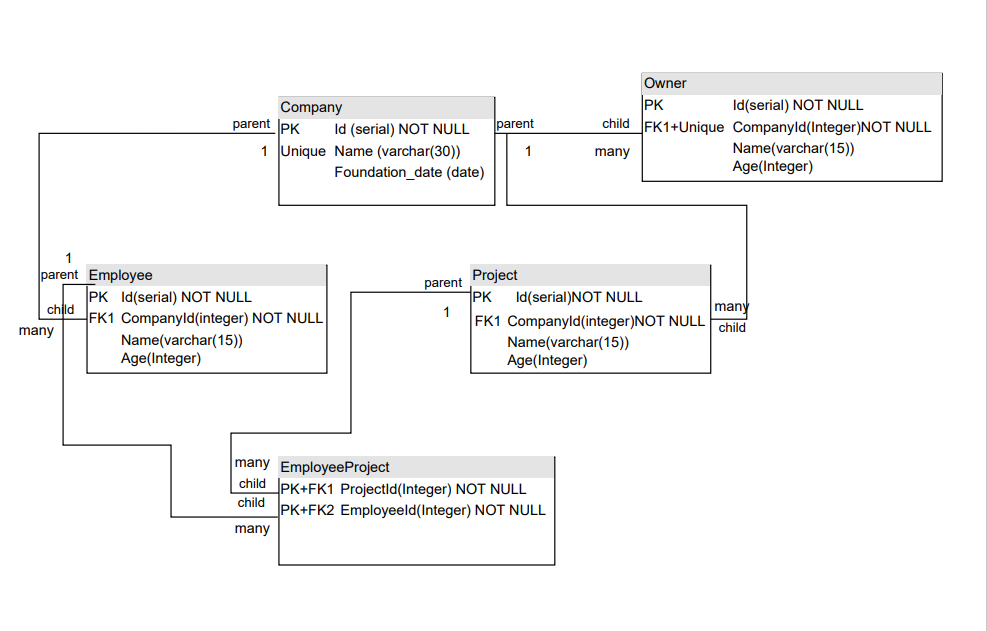
5)Була створена таблиця EmployeeProject зумовив її появу зв’язок M:N між сутностями Project та Employee, є допоміжною для реалізацію цього зв’язку.

Рисунок 2 - Схема бази даних (інструмент: draw.io)

**Опис об’єктів бази даних**

***Company***

|  |  |  |
| --- | --- | --- |
| *ID* | *Serial* | *PK (Not null)* |
| *Name* | *nvarchar(30)* | *Unique* |
| *Foundation\_date* | *date* |  |

***Owner***

|  |  |  |
| --- | --- | --- |
| *ID* | *Serial* | *PK (Not null)* |
| *CompanyId* | *integer* | *FK (Not null, on delete Cascade) Unique* |
| *Name* | *nvarchar(15)* |  |
| *Age* | *integer* |  |

***Employee***

|  |  |  |
| --- | --- | --- |
| *ID* | *Serial* | *PK (Not null)* |
| *CompanyId* | *integer* | *FK (Not null, on delete Cascade)* |
| *Name* | *nvarchar(15)* |  |
| *Age* | *integer* |  |

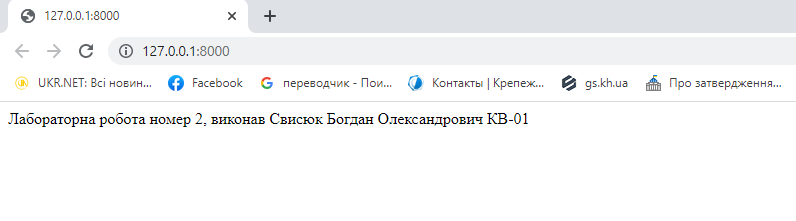
***Project***

|  |  |  |
| --- | --- | --- |
| *ID* | *Serial* | *PK (Not null)* |
| *CompanyId* | *integer* | *FK (Not null, on delete Cascade)* |
| *Name* | *nvarchar(15)* |  |
| *Age* | *integer* |  |
|  |  |  |

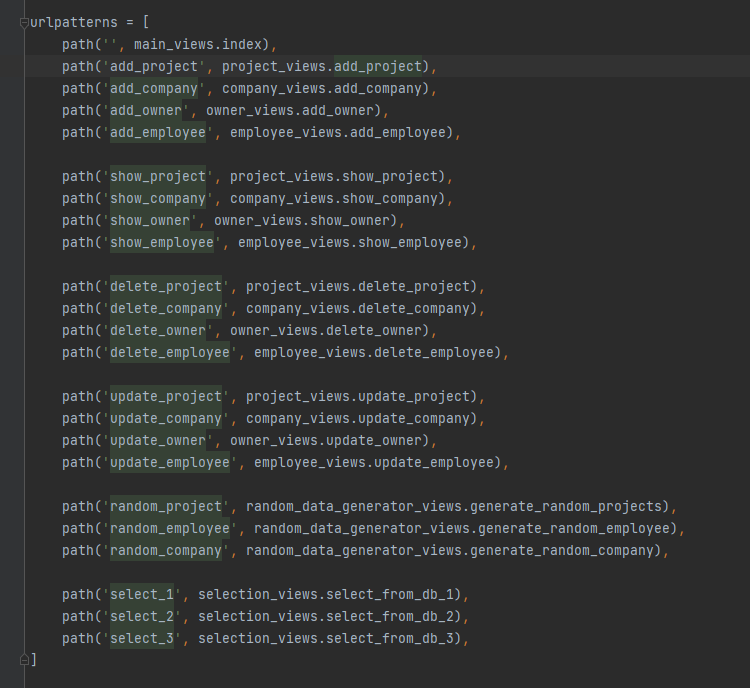
***EmployeeProject***

|  |  |  |
| --- | --- | --- |
| ProjectId | *Serial* | *PK FK (Not null, on delete cascade)* |
| EmployeeId | *Serial* | *PK FK (Not null, on delete cascade)* |

**Схема меню користувача**



Програма реалізована у форматі вебсерверу, написаному на модулі джанго, пункти завдання реалізовані у форматі веб запитів(метод GET, для легкості тестування).



За такими посиланнями можна буде отримувати відповіді у форматах: HttpResponse або JsonResponse.

Всі методи show ілюструють наповнення певної таблиці в базі даних.

Всі методи add додають інформацію до певної таблиці в базі даних.

Всі методи delete видаляють інформацію з певної таблиці в базі даних.

Всі методи update оновлюють інформацію в певній таблиці в базі даних.

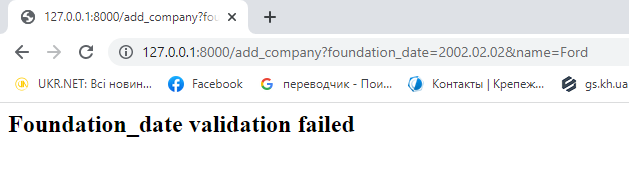
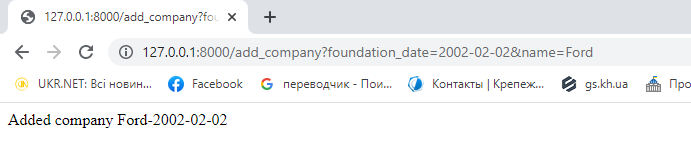
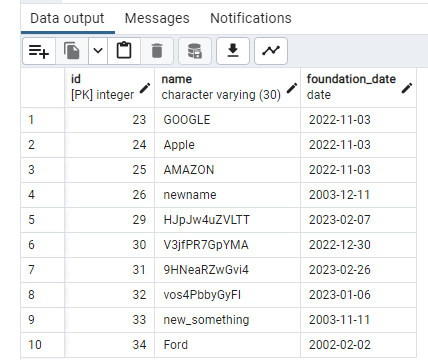
Всі методи random наповнюють рандомізованою інформацією певну таблицю в базі даних.

Всі методи select виконують вибірку даних за певним шаблоном з певної таблиці в базі даних.

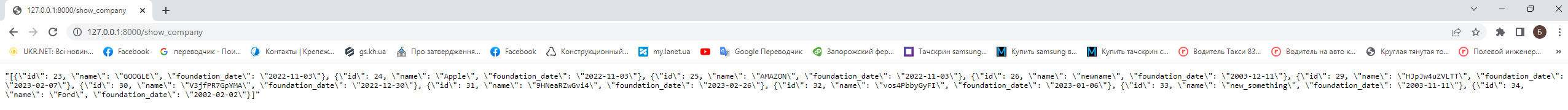
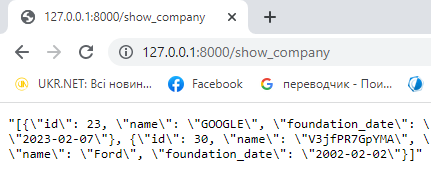
**Завдання 1**

Система обробки помилок в моїй програмі працює на 2 рівнях, перший – це валідація даних які посилаються на сервер в параметрах запиту.

Таким чином, наприклад при надсиланні дати в невірному форматі зі сторони користувача, він побачить відповідну помилку, сповіщуючу його про те, що дані були введені некоректно(запит в базу даних не надходить, помилка виявляється раніше).

Приклад: дата предається в таблицю в форматі рік-місяць-день, якщо дані будуть передані не в такому форматі, клієнт побачить наступну помилку: Якщо ж дані передати в правильному форматі все вийде і в таблиці ми побачимо щойно додану компанію:  

Або ж отримаємо дані через запит show:

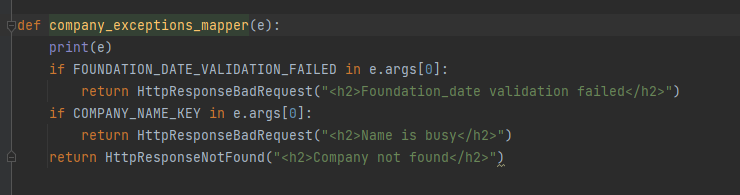
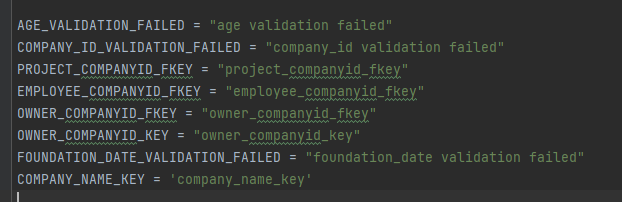
 

Таким чином валідація проходить для усіх параметрів, які мають якісь обмеження, щоб не витрачати зайвий час на запит, який завідомо буде невдалим.

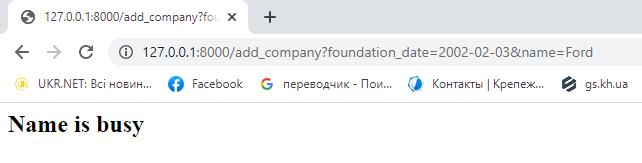
Другим механізмом відлову помилок є перехоплення помилки з бази данних, її mapping до певної помилки яку я хочу показати юзеру і власне її виведення.

Так наприклад помилка яка сповіщує нам, що унікальне ім’я в таблиці зайняте, відловиться попаде в мій метод, який вже скаже користувачу, що саме відбулось а не просто відобразить помилку з бази даних.

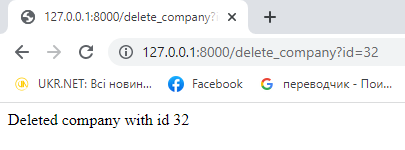
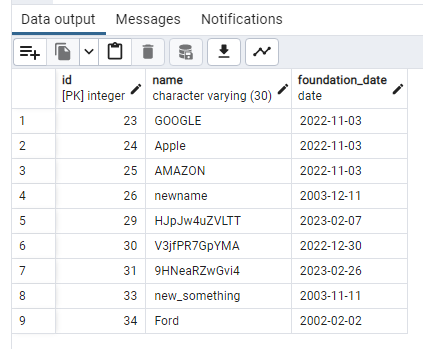
Приклад mapping’у:

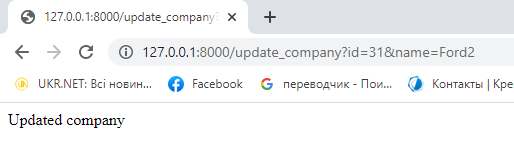
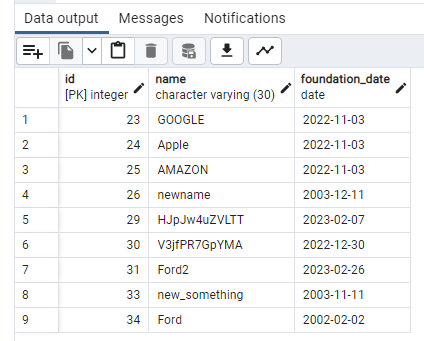
 

Відбувається пошук за ключовими словами у тексті помилки, який приходить нам від бази даних і переводиться у форму яку я задам.

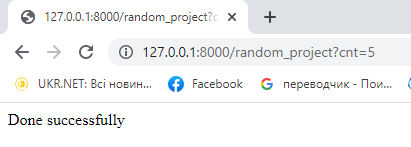
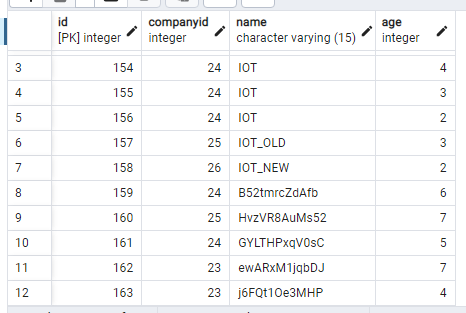
Приклад: спробуймо знову вставити компанію с тим же ім’ям і отримаємо наступне сповіщення: 

Щодо видалення даних з таблиць і пов’язаних з ними даних, то проблем з цим в конкретно моїй структурі немає, повсюду прописані жорсткі правила видалення, якщо видалена основна сутність.

Приклад запиту на видалення:  

Приклад запиту на апдейт:  

**Завдання 2**

Вставка рандомізованих даних в залежності від значення параметру cnt який ми передамо у параметрах запиту:  

Скрипт який ми для цього використали:

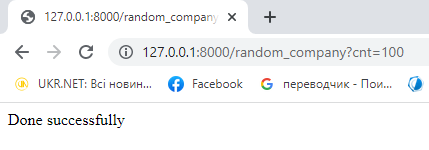
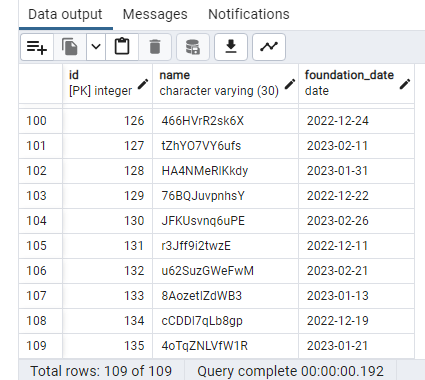
def generate\_random(self, cnt):  
 with connection.cursor() as cursor:  
 cursor.execute(f'''do $$  
 begin  
  
 for cnt in 1..{cnt} loop  
 INSERT INTO project(companyid,"name",age) VALUES(CAST(random() \* 2 + 23 as INT), (SELECT array\_to\_string(array   
 (   
 select substr('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', trunc(random() \* 62)::integer + 1, 1)  
 FROM generate\_series(1, 12)), '')), CAST(random() \* 5 + 3 as INT));  
 end loop;  
 end; $$ ''')

Єдиний момент, що значення рандомізованих зовнішніх ключей хоч і рандомізована, але в певному діапазоні, для того щоб не отримувати помилки референсу до неіснуючого ключа. Така ж функція існує і для вставки в таблицю employee і для company. Якщо для таблиці employee функція виглядає буквально так само, але з зміненим ім’ям таблиці то з company ситуація трішки інша, в нас немає проблем з зовнішнім ключем тому такого обмеження немає. Ну і функція рандому відсутня для таблиці owner, т.я. вона має зв’язок 1 до 1.

Скрипт для таблиці company:

def generate\_random(self, cnt):  
 with connection.cursor() as cursor:  
 cursor.execute(f'''do $$  
 begin  
  
 for cnt in 1..{cnt} loop  
 INSERT INTO company("name",foundation\_date) VALUES( (SELECT array\_to\_string(array   
 (   
 select substr('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', trunc(random() \* 62)::integer + 1, 1)  
 FROM generate\_series(1, 12)), '')), NOW()::date + (random() \* (NOW()+'90 days' - NOW())) + '30 days');  
 end loop;  
 end; $$ ''')

Вставка 100 рядків:

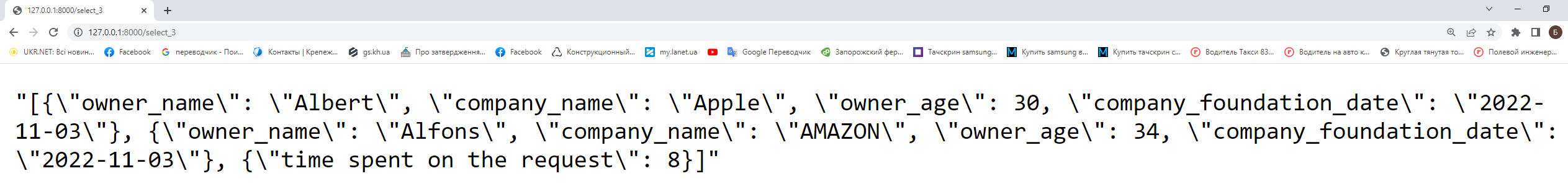
 

**Завдання 3**

Вибірка даних за певним шаблоном:

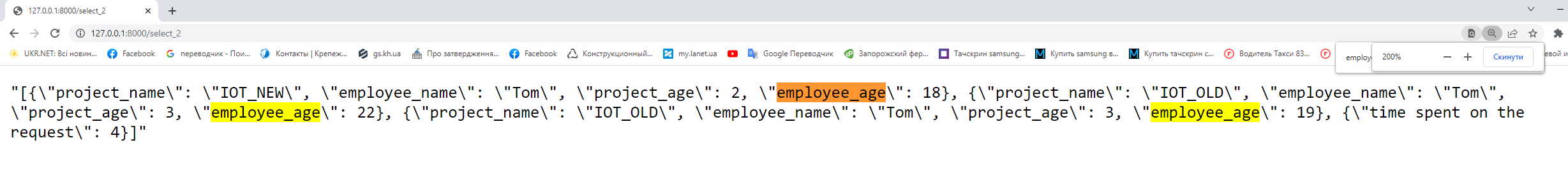
def select\_from\_db\_3(request):  
 if request.method == 'GET':  
 response\_dict = []  
 owner\_age = request.GET.get("owner\_age", 20)  
 company\_foundation\_date = request.GET.get("company\_foundation\_date", '2020-01-01')  
 owner\_name\_like = request.GET.get("owner\_name\_like", '%A%')  
 company\_name\_like = request.GET.get("company\_name\_like", '%A%')  
 with connection.cursor() as cursor:  
 time\_start = current\_milli\_time()  
 cursor.execute(f'''SELECT owner.name, company.name, owner.age, company.foundation\_date FROM company   
INNER JOIN owner ON owner.companyid = company.id  
WHERE owner.age> {owner\_age} AND owner.name LIKE '{owner\_name\_like}' AND company.name LIKE '{company\_name\_like}'   
AND company.foundation\_date > '{company\_foundation\_date}'::date;  
 ''')  
 time\_end = current\_milli\_time()  
 results = cursor.fetchall()  
 print(results)  
 for result in results:  
 temporary\_dict = {"owner\_name": result[0], "company\_name": result[1], "owner\_age": result[2],  
 "company\_foundation\_date": str(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)

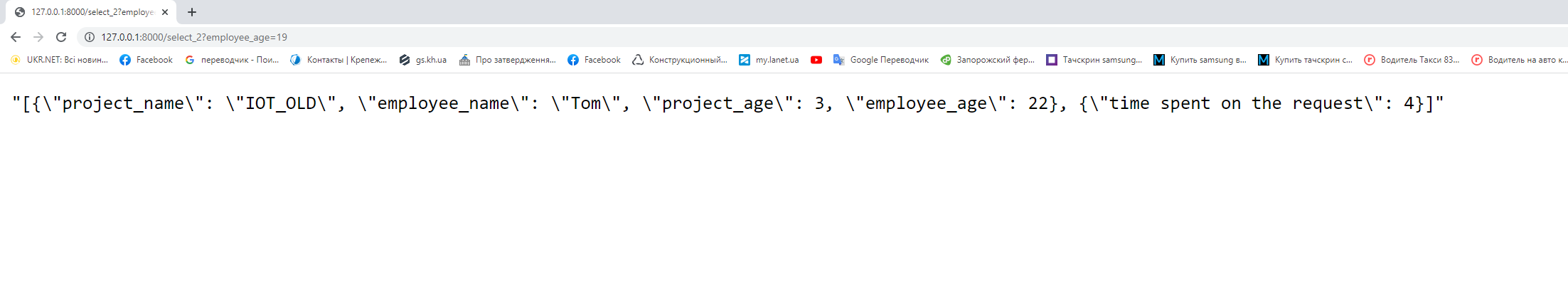
В даному випадку не вказуючи жодного параметру, ми виберемо основні дані з таблиць owner та company(з однаковими ід, тобто конкретно компанію і її власника), де і в імені компанії,і в імені її власника буде символ А, вік власника буде більше ніж 20, а компанія заснована пізніше ніж 2020-01-01.



Вибірка іншим методом:

def select\_from\_db\_2(request):  
 if request.method == 'GET':  
 response\_dict = []  
 employee\_age = request.GET.get("employee\_age", 16)  
 project\_age = request.GET.get("project\_age", 0)  
 project\_name\_like = request.GET.get("project\_name\_like", '%IOT%')  
 employee\_name\_like = request.GET.get("employee\_name\_like", 'Tom')  
 with connection.cursor() as cursor:  
 time\_start = current\_milli\_time()  
 cursor.execute(f'''SELECT project.name, employee.name, project.age, employee.age FROM employeeproject   
INNER JOIN employee ON employeeproject.employeeid = employee.id  
INNER JOIN project ON employeeproject.projectid = project.id  
WHERE project.name LIKE '{project\_name\_like}' AND employee.name LIKE '{employee\_name\_like}'   
 AND project.age > {project\_age} AND employee.age > {employee\_age};  
 ''')  
 time\_end = current\_milli\_time()  
 results = cursor.fetchall()  
 print(results)  
 for result in results:  
 temporary\_dict = {"project\_name": result[0], "employee\_name": result[1], "project\_age": result[2],  
 "employee\_age": 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)

Тут будуть вибрані дані робітників, які конкретно знаходять на якомусь проекті, за наступними критеріями: в імені проекта повинно бути наступна послідовність символів – IOT, а вік проекту більшим за 0, ім’я робітника повинно бути Tom, а вік більшим за 16. 

Ще один важливий момент, параметри фільтрації можна передавати в строку запиту і строка яку ми будемо надсилати в бд буде динамічно змінюватись, якщо я передам вік який дорівнюватиме 19, то ми побачимо наступну картину: 

Також останнім словариком завжди приходитиме час в мілісекундах, який був витрачений на виконання цього запиту.

**Завдання 4**

Реалізація паттерну MVC було виконана на лад Django’вського MVT, де все теж саме, щодо ролі компонентів, але C (mvc) – controller = V- view, a V(mvc)-view = T – template.

Код классу models.py:

import uuid  
import datetime  
from abc import ABC, abstractmethod  
from django.db import models  
from django.db import connection  
  
# 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 Company(Repository):  
 def \_\_init\_\_(self, name=None, foundation\_date=None):  
 self.name = name  
 self.foundation\_date = foundation\_date  
  
 def add(self):  
 with connection.cursor() as cursor:  
 try:  
 foundation\_date = datetime.datetime.strptime(self.foundation\_date, '%Y-%m-%d')  
 except:  
 raise Exception("foundation\_date validation failed")  
 cursor.execute(f"INSERT INTO company(name, foundation\_date) VALUES ('{self.name}','{self.foundation\_date}')")  
  
 def delete(self, id):  
 with connection.cursor() as cursor:  
 cursor.execute(f"DElETE FROM company WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def update(self, id):  
 with connection.cursor() as cursor:  
 x\_name = f"name='{self.name}'" if self.name else ''  
 if self.foundation\_date is not None:  
 try:  
 foundation\_date = datetime.datetime.strptime(self.foundation\_date, '%Y-%m-%d')  
 x\_name += ',' if x\_name != '' else ''  
 except:  
 raise Exception("foundation\_date validation failed")  
 x\_foundation\_date = f"foundation\_date='{self.foundation\_date}'" if self.foundation\_date else ''  
 cursor.execute(  
 "UPDATE company " + f"SET " + f'{x\_name} ' + f'{x\_foundation\_date}' 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 company')  
 companys = cursor.fetchall()  
 print(companys)  
 for company in companys:  
 temporary\_dict = {"id": company[0], "name": company[1], "foundation\_date": str(company[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 company("name",foundation\_date) VALUES( (SELECT array\_to\_string(array   
 (   
 select substr('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', trunc(random() \* 62)::integer + 1, 1)  
 FROM generate\_series(1, 12)), '')), NOW()::date + (random() \* (NOW()+'90 days' - NOW())) + '30 days');  
 end loop;  
 end; $$ ''')  
  
  
class Project(Repository):  
 def \_\_init\_\_(self, name=None, age=None,company\_id=None):  
 self.name = name  
 self.age = age  
 self.company\_id = company\_id  
  
 def add(self):  
 with connection.cursor() as cursor:  
 try:  
 self.age = int(self.age)  
 except:  
 raise Exception("age validation failed")  
 try:  
 self.company\_id = int(self.company\_id)  
 except:  
 raise Exception("company\_id validation failed")  
  
 cursor.execute(f"INSERT INTO project(companyid, name, age) VALUES ({self.company\_id},'{self.name}', {self.age})")  
  
 def delete(self, id):  
 with connection.cursor() as cursor:  
 cursor.execute(f"DElETE FROM project WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def update(self, id):  
 with connection.cursor() as cursor:  
 x\_name = f"name='{self.name}'" if self.name else ''  
  
 if self.company\_id is not None:  
 try:  
 self.company\_id = int(self.company\_id)  
 x\_name += ', ' if x\_name != '' else ''  
 except:  
 raise Exception("company\_id validation failed")  
  
 x\_company\_id = f'companyid={self.company\_id}' if self.company\_id else ''  
  
 if self.age is not None:  
 try:  
 self.age = int(self.age)  
 x\_company\_id += ', ' if x\_company\_id != '' else ''  
 except:  
 raise Exception("age validation failed")  
  
 x\_age = f'age={self.age}' if self.age else ''  
 print("UPDATE project " + f"SET " + f"{x\_name}" + f'{x\_company\_id} ' + f'{x\_age}' f" WHERE id={id}")  
 cursor.execute(  
 "UPDATE project " + f"SET " + f'{x\_name}' + f'{x\_company\_id}' + 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 project')  
 projects = cursor.fetchall()  
 print(projects)  
 for project in projects:  
 temporary\_dict = {"id": project[0], "name": project[2], "age": project[3],  
 "company\_id": project[1]}  
 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 project(companyid,"name",age) VALUES(CAST(random() \* 2 + 23 as INT), (SELECT array\_to\_string(array   
 (   
 select substr('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', trunc(random() \* 62)::integer + 1, 1)  
 FROM generate\_series(1, 12)), '')), CAST(random() \* 5 + 3 as INT));  
 end loop;  
 end; $$ ''')  
  
  
class Employee(Repository):  
 def \_\_init\_\_(self, name=None, age=None,company\_id=None):  
 self.name = name  
 self.age = age  
 self.company\_id = company\_id  
  
 def add(self):  
 with connection.cursor() as cursor:  
 try:  
 self.age = int(self.age)  
 except:  
 raise Exception("age validation failed")  
 try:  
 self.company\_id = int(self.company\_id)  
 except:  
 raise Exception("company\_id validation failed")  
  
 cursor.execute(f"INSERT INTO employee(companyid, name, age) VALUES ({self.company\_id},'{self.name}', {self.age})")  
  
 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:  
  
 x\_name = f"name='{self.name}'" if self.name else ''  
  
 if self.company\_id is not None:  
 try:  
 self.company\_id = int(self.company\_id)  
 x\_name += ', ' if x\_name != '' else ''  
 except:  
 raise Exception("company\_id validation failed")  
 x\_company\_id = f'companyid={self.company\_id}' if self.company\_id else ''  
  
 if self.age is not None:  
 try:  
 self.age = int(self.age)  
 x\_company\_id += ', ' if x\_company\_id != '' else ''  
 except:  
 raise Exception("age validation failed")  
 x\_age = f'age={self.age}' if self.age else ''  
  
  
 cursor.execute(  
 "UPDATE employee " + f"SET " + f'{x\_name} ' + f'{x\_company\_id} ' + 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 employee')  
 employees = cursor.fetchall()  
 print(employees)  
 for employee in employees:  
 temporary\_dict = {"id": employee[0], "name": employee[2], "age": employee[3],  
 "company\_id": employee[1]}  
 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(companyid,"name",age) VALUES(CAST(random() \* 2 + 23 as INT), (SELECT array\_to\_string(array   
 (   
 select substr('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', trunc(random() \* 62)::integer + 1, 1)  
 FROM generate\_series(1, 12)), '')), CAST(random() \* 5 + 3 as INT));  
 end loop;  
 end; $$ ''')  
  
  
class Owner(Repository):  
 def \_\_init\_\_(self, name=None, age=None, company\_id=None):  
 self.name = name  
 self.age = age  
 self.company\_id = company\_id  
  
 def add(self):  
 with connection.cursor() as cursor:  
 try:  
 self.age = int(self.age)  
 except:  
 raise Exception("age validation failed")  
 try:  
 self.company\_id = int(self.company\_id)  
 except:  
 raise Exception("company\_id validation failed")  
  
 cursor.execute(f"INSERT INTO owner(companyid, name, age) VALUES ({self.company\_id},'{self.name}', {self.age})")  
  
 def delete(self, id):  
 with connection.cursor() as cursor:  
 cursor.execute(f"DElETE FROM owner WHERE id='{id}'")  
 affected\_rows = cursor.rowcount  
 if affected\_rows == 0:  
 raise  
  
 def update(self, id):  
 with connection.cursor() as cursor:  
 x\_name = f"name='{self.name}'" if self.name else ''  
 if self.age is not None:  
 try:  
 self.age = int(self.age)  
 x\_name += ',' if x\_name != '' else ''  
 except:  
 raise Exception("age validation failed")  
 x\_age = f'age={self.age}' if self.age else ''  
 cursor.execute(  
 "UPDATE owner " + f"SET " + f'{x\_name} ' + 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 owner')  
 owners = cursor.fetchall()  
 print(owners)  
 for owner in owners:  
 temporary\_dict = {"id": owner[0], "name": owner[2], "age": owner[3],  
 "company\_id": owner[1]}  
 response\_dict.append(temporary\_dict)  
  
 def generate\_random(self, cnt):  
 pass

Код класу exceptions.py:

from django.http import HttpResponseBadRequest, HttpResponseNotFound  
  
AGE\_VALIDATION\_FAILED = "age validation failed"  
COMPANY\_ID\_VALIDATION\_FAILED = "company\_id validation failed"  
PROJECT\_COMPANYID\_FKEY = "project\_companyid\_fkey"  
EMPLOYEE\_COMPANYID\_FKEY = "employee\_companyid\_fkey"  
OWNER\_COMPANYID\_FKEY = "owner\_companyid\_fkey"  
OWNER\_COMPANYID\_KEY = "owner\_companyid\_key"  
FOUNDATION\_DATE\_VALIDATION\_FAILED = "foundation\_date validation failed"  
COMPANY\_NAME\_KEY = 'company\_name\_key'  
  
def project\_exceptions\_mapper(e):  
 if PROJECT\_COMPANYID\_FKEY in e.args[0]:  
 return HttpResponseBadRequest("<h2>No such Company</h2>")  
 if AGE\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Age validation failed</h2>")  
 if COMPANY\_ID\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Company\_id validation failed</h2>")  
 return HttpResponseNotFound("<h2>Project not found</h2>")  
  
  
def employee\_exceptions\_mapper(e):  
 print(e)  
 if EMPLOYEE\_COMPANYID\_FKEY in e.args[0]:  
 return HttpResponseBadRequest("<h2>No such Company</h2>")  
 if AGE\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Age validation failed</h2>")  
 if COMPANY\_ID\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Company\_id validation failed</h2>")  
 return HttpResponseNotFound("<h2>Employee not found</h2>")  
  
  
def owner\_exceptions\_mapper(e):  
 print(e)  
 if OWNER\_COMPANYID\_KEY in e.args[0]:  
 return HttpResponseBadRequest("<h2>Company has another owner</h2>")  
 if OWNER\_COMPANYID\_FKEY in e.args[0]:  
 return HttpResponseBadRequest("<h2>No such Company</h2>")  
 if AGE\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Age validation failed</h2>")  
 if COMPANY\_ID\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Company\_id validation failed</h2>")  
 return HttpResponseNotFound("<h2>Owner not found</h2>")  
  
  
def company\_exceptions\_mapper(e):  
 print(e)  
 if FOUNDATION\_DATE\_VALIDATION\_FAILED in e.args[0]:  
 return HttpResponseBadRequest("<h2>Foundation\_date validation failed</h2>")  
 if COMPANY\_NAME\_KEY in e.args[0]:  
 return HttpResponseBadRequest("<h2>Name is busy</h2>")  
 return HttpResponseNotFound("<h2>Company not found</h2>")

Код класу main\_views.py:

from django.http import HttpResponse, HttpResponseBadRequest, HttpResponseNotFound, JsonResponse  
import json  
from django.db import connection  
  
  
from Lab2.exceptions import project\_exceptions\_mapper  
  
  
def index(request):  
 return HttpResponse("Лабораторна робота номер 2, виконав Свисюк Богдан Олександрович КВ-01")

Код класу company\_views.py:

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

Код класу employee\_views.py:

from django.http import HttpResponse, HttpResponseBadRequest, HttpResponseNotFound, JsonResponse  
import json  
from django.db import connection  
  
  
from Lab2.exceptions import employee\_exceptions\_mapper  
from Lab2.models import Employee  
  
  
def add\_employee(request):  
 if request.method == 'GET':  
 try:  
 name = request.GET.get("name")  
 age = request.GET.get("age")  
 company\_id = request.GET.get('company\_id')  
 employee = Employee(name, age, company\_id)  
 employee.add()  
  
 return HttpResponse(f"Added employee {name}-{age}")  
 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  
 name = request.GET.get("name")  
 age = request.GET.get("age")  
 company\_id = request.GET.get("company\_id")  
 employee = Employee(name, age, company\_id)  
 employee.update(id)  
  
 return HttpResponse(f"Updated employee")  
 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)

Код класу project\_views.py:

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

Код класу owner\_views.py:

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

Код класу random\_data\_generator\_views.py:

from django.db import connection  
from django.http import HttpResponse  
  
from Lab2.models import Company, Employee, Project  
  
  
def generate\_random\_projects(request):  
 if request.method == "GET":  
 cnt = request.GET.get("cnt")  
 project = Project()  
 project.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\_company(request):  
 if request.method == "GET":  
 cnt = request.GET.get("cnt")  
 company = Company()  
 company.generate\_random(cnt)  
 return HttpResponse('Done successfully')

Код класу selection\_views.py:

import datetime  
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", 18)  
 project\_age = request.GET.get("project\_age", 3)  
 project\_name\_like = request.GET.get("project\_name\_like", '%IOT%')  
 employee\_name\_like = request.GET.get("project\_name\_like", 'Tom')  
 with connection.cursor() as cursor:  
 time\_start = current\_milli\_time()  
 cursor.execute(f'''SELECT project.name, employee.name, project.age, employee.age FROM project INNER JOIN employee ON project.companyid=employee.companyid  
 WHERE project.name LIKE '{project\_name\_like}' AND employee.name LIKE '{employee\_name\_like}'   
 AND project.age > {project\_age} AND employee.age > {employee\_age};  
 ''')  
 time\_end = current\_milli\_time()  
 results = cursor.fetchall()  
 print(results)  
 for result in results:  
 temporary\_dict = {"project\_name": result[0], "employee\_name": result[1], "project\_age": result[2],  
 "employee\_age": 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)  
  
  
def select\_from\_db\_2(request):  
 if request.method == 'GET':  
 response\_dict = []  
 employee\_age = request.GET.get("employee\_age", 16)  
 project\_age = request.GET.get("project\_age", 0)  
 project\_name\_like = request.GET.get("project\_name\_like", '%IOT%')  
 employee\_name\_like = request.GET.get("employee\_name\_like", 'Tom')  
 with connection.cursor() as cursor:  
 time\_start = current\_milli\_time()  
 cursor.execute(f'''SELECT project.name, employee.name, project.age, employee.age FROM employeeproject   
INNER JOIN employee ON employeeproject.employeeid = employee.id  
INNER JOIN project ON employeeproject.projectid = project.id  
WHERE project.name LIKE '{project\_name\_like}' AND employee.name LIKE '{employee\_name\_like}'   
 AND project.age > {project\_age} AND employee.age > {employee\_age};  
 ''')  
 time\_end = current\_milli\_time()  
 results = cursor.fetchall()  
 print(results)  
 for result in results:  
 temporary\_dict = {"project\_name": result[0], "employee\_name": result[1], "project\_age": result[2],  
 "employee\_age": 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)  
  
  
def select\_from\_db\_3(request):  
 if request.method == 'GET':  
 response\_dict = []  
 owner\_age = request.GET.get("owner\_age", 20)  
 company\_foundation\_date = request.GET.get("company\_foundation\_date", '2020-01-01')  
 owner\_name\_like = request.GET.get("owner\_name\_like", '%A%')  
 company\_name\_like = request.GET.get("company\_name\_like", '%A%')  
 with connection.cursor() as cursor:  
 time\_start = current\_milli\_time()  
 cursor.execute(f'''SELECT owner.name, company.name, owner.age, company.foundation\_date FROM company   
INNER JOIN owner ON owner.companyid = company.id  
WHERE owner.age> {owner\_age} AND owner.name LIKE '{owner\_name\_like}' AND company.name LIKE '{company\_name\_like}'   
AND company.foundation\_date > '{company\_foundation\_date}'::date;  
 ''')  
 time\_end = current\_milli\_time()  
 results = cursor.fetchall()  
 print(results)  
 for result in results:  
 temporary\_dict = {"owner\_name": result[0], "company\_name": result[1], "owner\_age": result[2],  
 "company\_foundation\_date": str(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)

Роутінг відбувається за допомогою джанго у файлі urls.py:

*"""djangoProject 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.contrib import admin  
from django.urls import path  
from Lab2.views import project\_views as project\_views  
from Lab2.views import employee\_views as employee\_views  
from Lab2.views import owner\_views as owner\_views  
from Lab2.views import company\_views as company\_views  
from Lab2.views import random\_data\_generator\_views as random\_data\_generator\_views  
from Lab2.views import selection\_views as selection\_views  
from Lab2 import main\_views as main\_views  
  
urlpatterns = [  
 path('', main\_views.index),  
 path('add\_project', project\_views.add\_project),  
 path('add\_company', company\_views.add\_company),  
 path('add\_owner', owner\_views.add\_owner),  
 path('add\_employee', employee\_views.add\_employee),  
  
 path('show\_project', project\_views.show\_project),  
 path('show\_company', company\_views.show\_company),  
 path('show\_owner', owner\_views.show\_owner),  
 path('show\_employee', employee\_views.show\_employee),  
  
 path('delete\_project', project\_views.delete\_project),  
 path('delete\_company', company\_views.delete\_company),  
 path('delete\_owner', owner\_views.delete\_owner),  
 path('delete\_employee', employee\_views.delete\_employee),  
  
 path('update\_project', project\_views.update\_project),  
 path('update\_company', company\_views.update\_company),  
 path('update\_owner', owner\_views.update\_owner),  
 path('update\_employee', employee\_views.update\_employee),  
  
 path('random\_project', random\_data\_generator\_views.generate\_random\_projects),  
 path('random\_employee', random\_data\_generator\_views.generate\_random\_employee),  
 path('random\_company', random\_data\_generator\_views.generate\_random\_company),  
  
 path('select\_1', selection\_views.select\_from\_db\_1),  
 path('select\_2', selection\_views.select\_from\_db\_2),  
 path('select\_3', selection\_views.select\_from\_db\_3),  
]

Підключення баз даних, проекту в проект джанго та інших конфігурацій у файлі settings.py:

*"""  
Django settings for djangoProject project.  
  
Generated by 'django-admin startproject' using Django 4.1.2.  
  
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-ot)86x$^b@)08(-5ky8led0)ws)36em4t%ytsmsih^\*auk9ug9'  
  
# 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.apps.Lab2Config',  
]  
  
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': [BASE\_DIR / 'templates']  
 ,  
 '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': 'bodi4ka2003',  
 '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'

**Посилання на гітхаб:**

https://github.com/bogdan213114/Lab1/tree/Lab2