НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

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

**Розрахунково графічна робота**

з дисципліни

**«Бази даних та засоби управління»**

**«Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»**

Виконав: студент 3 курсу

ФПМ групи КВ-23

Зленко Артем Андрійович

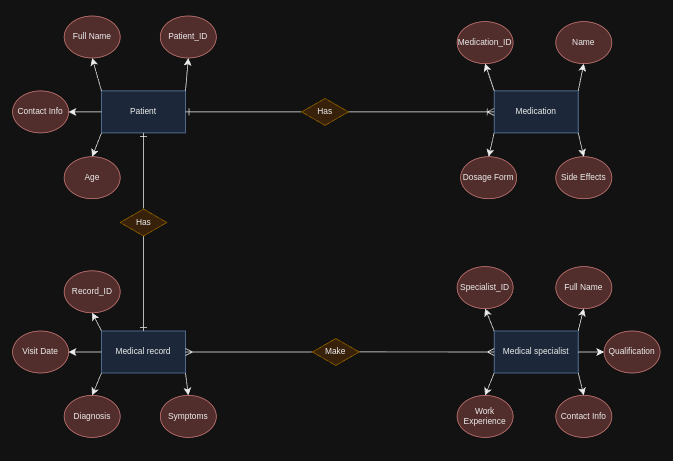
**Київ – 2024**

**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).

Графічне подання концептуальної моделі зображено на рисунку 1



*Рис. 1 ER-діаграма, побудована за нотацією Чена*

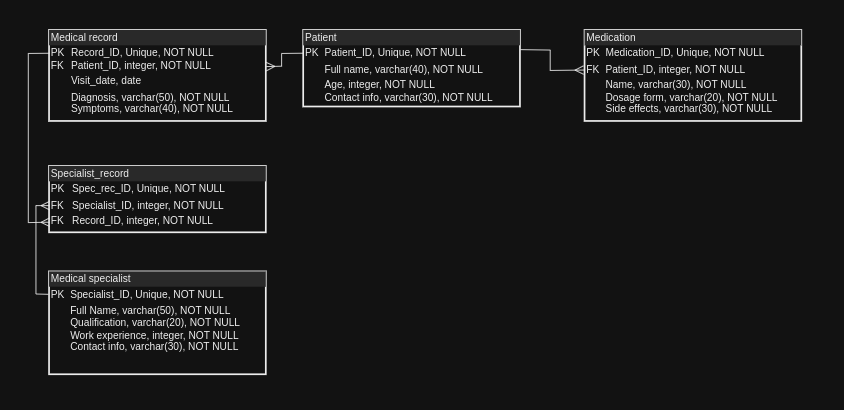
**Перетворення концептуальної моделі на логічну модель та схему бази даних**

1. Сутність Patient перетворено в таблицю Patient
2. Сутність Medical record перетворено в таблицю Medical record
3. Сутність Medical specialist перетворено в таблицю Medical specialist
4. Сутність Medication перетворено в таблицю Medication

В моделі неможливо представити зв’язок N:M але в концептуальній моделі він існує між сутностями Medical specialist i Medical record, для цього вводимо допоміжну таблицю Specialist\_Record

| Сутність | Атрибут | Тип атрибуту |
| --- | --- | --- |
| Patient – користувач, пацієнт | Patient\_ID – ID пацієнта  Full name – ПІБ пацієнта  Contact info – пошта пацієнта  Age – вік пацієнта | integer (числовий)  character varying (рядок)  character varying (рядок)  integer (числовий) |
| Medical record – запис пацієнта, створений медичним працівником | Record\_ID – ID запису  Patient\_ID – ID пацієнта  Visit date – дата візиту пацієнта  Diagnosis – діагноз  Symptoms – симптоми | integer (числовий)  integer (числовий)  date (дата)  character varying (рядок)  character varying (рядок) |
| Medical specialist – медичний працівник, який створює медичний запис | Specialist\_ID – ID спеціаліста  Full name – ПІБ лікаря  Qualification – рівень кваліфікації лікаря(категорія ступінь)  Work experience – стаж роботи  Contact info – пошта лікаря | integer (числовий)  character varying (рядок)  character varying (рядок)  integer (числовий)  character varying (рядок) |
| Medication – ліки, які можуть бути в пацієнта | Medication\_ID – ID ліків Patient\_ID – ID пацієнта  Name - назва ліків  Dosage form – форма випуску(капсули, таблетки, сироп)  Side effects - побічні ефекти | integer (числовий)  integer (числовий)  character varying (рядок)  character varying (рядок)  character varying (рядок) |
| Specialist\_Record – відповідність медичного працівника до медичного запису, який він зробив | Spec\_Rec\_ID – ID запису певного спеціаліста  Specialist\_ID – ID спеціаліста  Record\_ID – ID запису | integer (числовий)  integer (числовий)  integer (числовий) |

**Перетворення розробленої моделі «сутність-зв’язок» у схему бази даних PostgreSQL**



*Рис 2. Схема бази даних у графічному вигляді*

Сутність Patient було перетворено на таблицю Patient. Первинний ключ(ідентифікатор) Id, атрибути: Full name, Age, Contact info.

Сутність Medication було перетворено на таблицю Medication.

Первинний ключ (ідентифікатор) Id, атрибути: Name, Dosage form, Side effects.

Сутність Medical record було перетворено на таблицю Medical record. Первинний ключ (ідентифікатор) Id, атрибути: Visit date, Diagnosis, Symptom.

Сутність Medical specialist було перетворено на таблицю Medical specialist. Первинний ключ (ідентифікатор) Id, атрибути: Full Name, Qualification, Work experience, Contact info.

Було також створено таблицю Specialist\_record. Первинний ключ

(ідентифікатор) Id, атрибути: Specialist\_Id, Record\_Id. Таблиця Specialist\_record була створена для того, щоб утворювати пару

медичний спеціаліст - медичний запис. Можна було б додати в таблицю Specialist поле Record\_Id, яке теоретично б пов’язувало спеціаліста з записом в медичній картці. АЛЕ: якщо б 1 і той самий спеціаліст мав доступ до двох медичних записів, то довелося би мати 2 майже однакові записи в таблиці Specialist. Єдиною відмінністю було б поле Record\_Id. Отже, знову була б виділена пам’ять на всі інші поля (Qualification, Work experience, Contact info). Це вимагало б додаткової пам’яті. А так, завдяки такій таблиці Specialist\_record, реалізована можливість одному спеціалісту мати декілька записів в медичних картках. Це також реалізовує 1НФ, оскільки 1 нормальна форма вимагає відсутні груп полів, які

повторюються. В таблиці Specialist\_Record створено два зовнішніх ключа: FK\_Specialist та FK\_Record. Вони посилаються на поля Id в таблицях Medical specialist та Medical Record відповідно.

Кожен пацієнт має медичну картку і записи в ній. Для отримання цього зв’язку було створено зовнішній ключ FK\_Patient. В таблиці Medical Record створено поле, яке посилається на поле (первинний ключ) Id в таблиці Patient. (1:N)

Кожному пацієнту виписують ліки для лікування. Для отримання цього зв’язку було створено зовнішній ключ FK\_Patient. В таблицю Medication створено поле, яке посилається на поле (первинний ключ) Id в таблиці Patient. (1:N)

**Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL**

**Посилання на github репозиторій:** <https://github.com/ZelenkoArtem/BD>

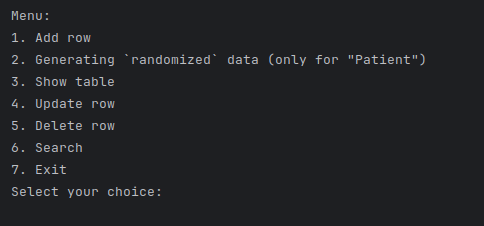
**Середовище для відлагодження** SQL-запитів до бази даних –

PgAdmin4.

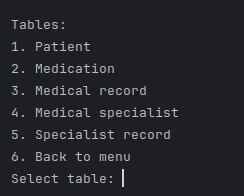
**Мова програмування** – Python 3.12.

**Середовище розробки програмного забезпечення** – PyCharm Community Edition.

**Бібліотека взаємодії** з PostgreSQL - psycopg2 (рисунок 3 та 4)



*Рисунок 3 – Структура меню користувача*



*Рис 4. Варіанти таблиць, з якими можна працювати*

Меню складається з 7 опцій:

1) “Додати рядок”: ця функція дозволяє користувачеві вставляти нові

записи в таблиці. Після вибору опції, необхідно вказати конкретну

таблицю та ввести дані для кожного атрибуту.

2) “Генерування випадкових даних”: цей пункт призначений для

створення випадкових записів у таблиці “Patient”. Користувач

вказує кількість полів, які необхідно додати.

3) “Показати таблицю”: опція дозволяє вивести на екран вміст обраної

таблиці. Користувач перед виведенням обирає необхідну таблицю, і

всі її поля відображаються.

4) “Оновити рядок”: ця функція використовується для редагування

полів у таблиці за ідентифікатором. Користувач вибирає таблицю,

вказує ідентифікатор рядка, який потрібно змінити, і вводить нові

дані для кожного атрибуту.

5) “Видалити рядок”: опція призначена для видалення рядка за його

ідентифікатором. Користувач обирає таблицю та вводить

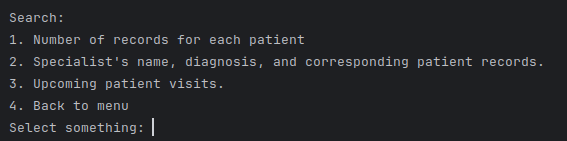
ідентифікатор рядка, який потрібно видалити.

6) “Пошук”: цей пункт дозволяє шукати записи за атрибутами у

декількох таблицях. Користувач обирає один із чотирьох варіантів

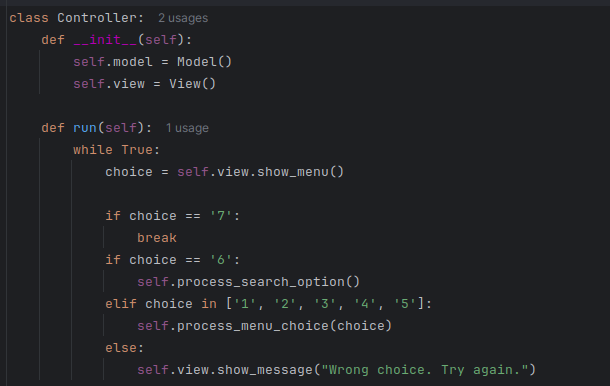
пошуку.

7) “Вихід”: завершення роботи програми.



*Рис 5. – Варіанти запитів*

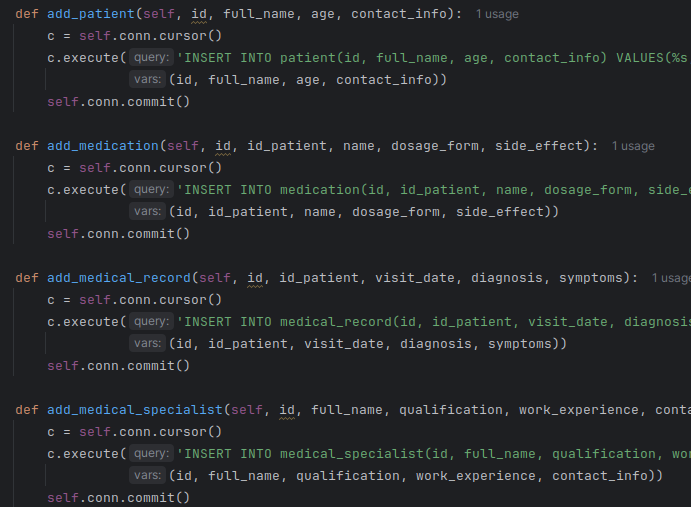
1. **Number of records per patient**: Цей запит повертає список пацієнтів із їхнім ідентифікаційним номером, іменем та кількістю медичних записів (відвідувань) для кожного пацієнта. Він використовує ліве з'єднання (LEFT JOIN) для отримання всіх пацієнтів, навіть якщо у них немає медичних записів
2. **Specialist's name, diagnosis, and corresponding patient records**: Запит отримує список медичних записів з діагнозами, іменами спеціалістів і пацієнтів. З'єднуються таблиці медичних записів, спеціалістів та пацієнтів, щоб визначити, які спеціалісти лікували яких пацієнтів та який діагноз був поставлений
3. **Upcoming patient visits**: Цей запит відображає майбутні відвідування пацієнтів, з іменами пацієнтів, датами візитів та діагнозами. Він вибирає відвідування, які відбудуться після поточної дати, і обмежує кількість результатів до 5
4. **Back to menu**. Повернення до основного меню. Також можна побачити час виконання запиту у мілісекундах, після виведення даних у кожному пункті

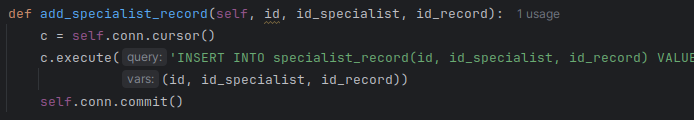
**

*Рис 6. Фрагмент коду (файл сontroller.py), в якому наведено головний цикл роботи програми*

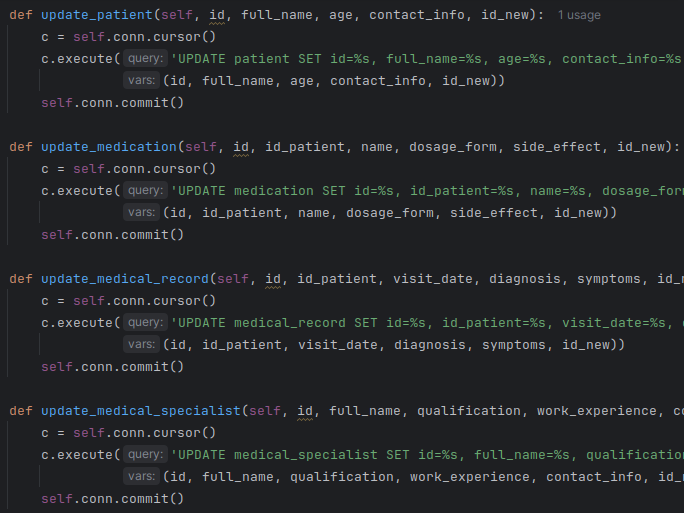
**Фрагменти коду** (файл model.py), в якому наведено функції внесення, редагування, вилучення та генерації даних у базі даних

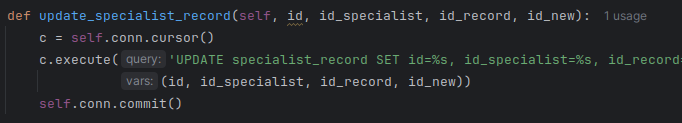
Функції внесення даних:



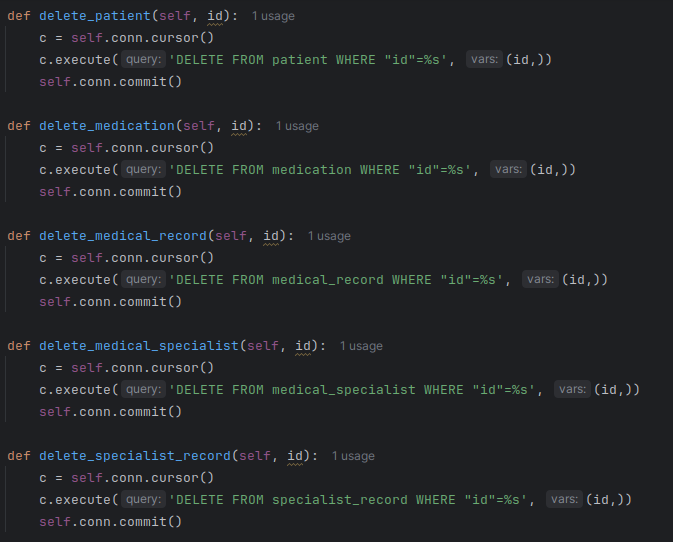


Функції оновлення даних:

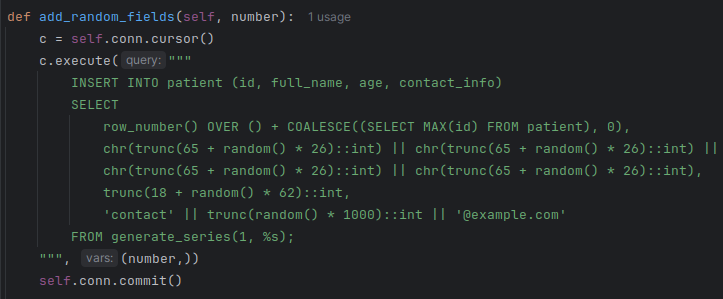




Функції видалення значень:



Функція генерування даних (лише для пацієнтів):

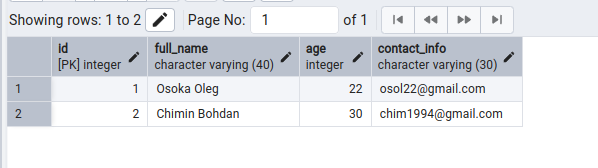


**Таблиця Patient**

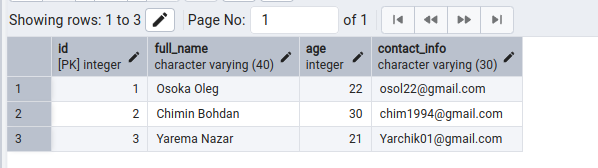
*Додавання рядка*



*Початковий стан таблиці*

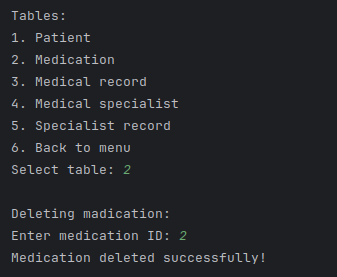
**

*Кінцевий стан таблиці*

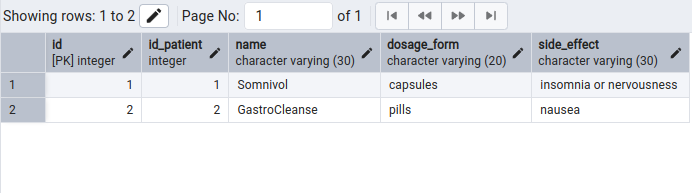


**Таблиця Medication**

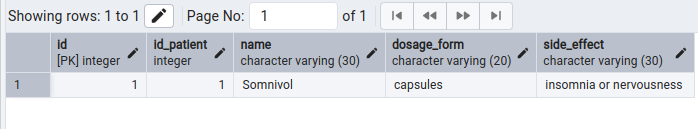
*Видалення рядка*

**

*Початковий стан таблиці*

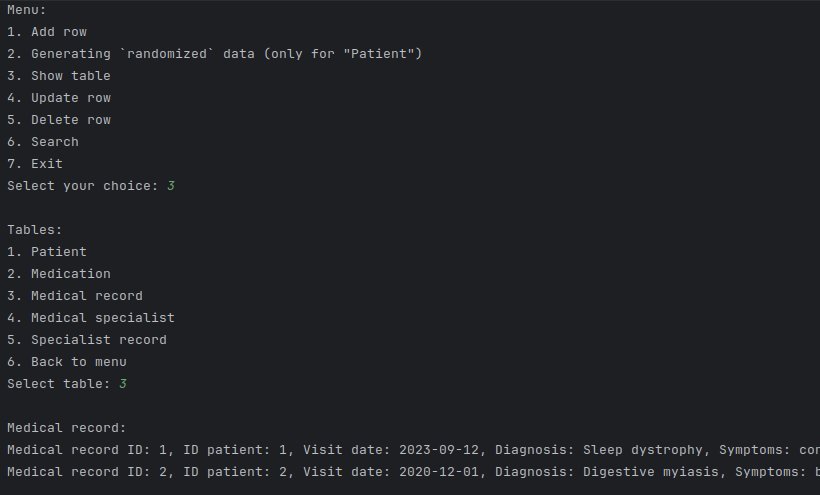
**

*Кінцевий стан таблиці*

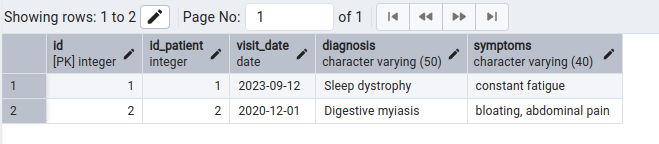
**

**Таблиця Medical Record**

*Перегляд таблиці medical record*

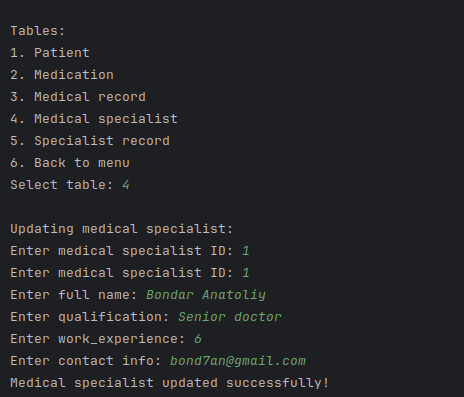
**

*Вигляд таблиці*

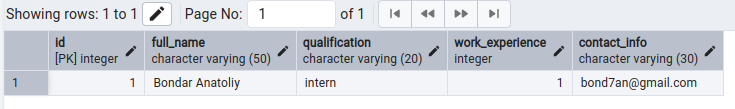
**

**Таблиця Medical Specialist**

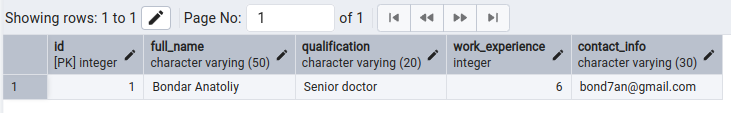
*Редагування рядка*

**

*Початковий стан таблиці*

**

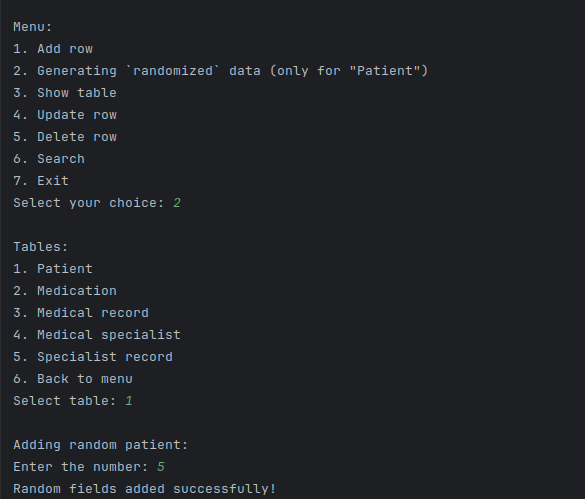
*Кінцевий стан таблиці*

**

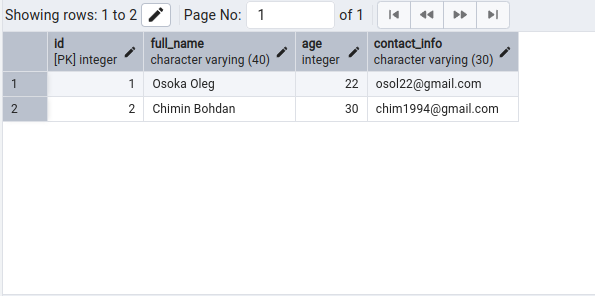
**Генерування «рандомізованих» даних Запит, що був використаний для генерування «рандомізованих» даних:**

INSERT INTO patient (id, full\_name, age, contact\_info) SELECT row\_number() OVER () + COALESCE((SELECT MAX(id) FROM patient), 0), chr(trunc(65 + random() \* 26)::int) || chr(trunc(65 + random() \* 26)::int) || ' ' || chr(trunc(65 + random() \* 26)::int) || chr(trunc(65 + random() \* 26)::int), trunc(18 + random() \* 62)::int, 'contact' || trunc(random() \* 1000)::int || '@example.com' FROM generate\_series(1, %s);

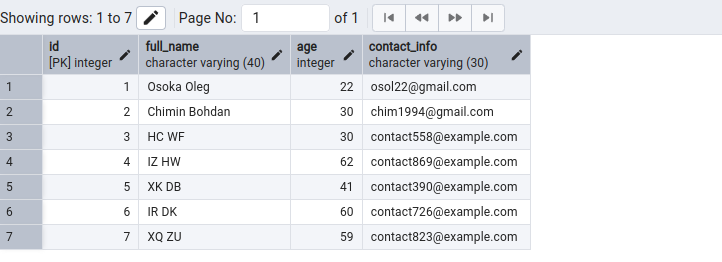
*Створення нових рядків в таблиці пацієнт*

**

*Початковий стан*

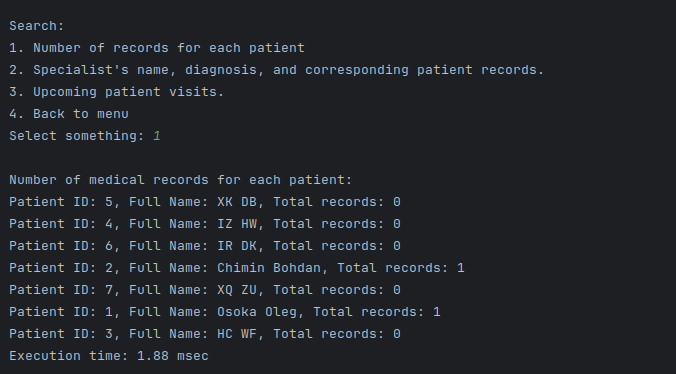
**

*Кінцевий стан*

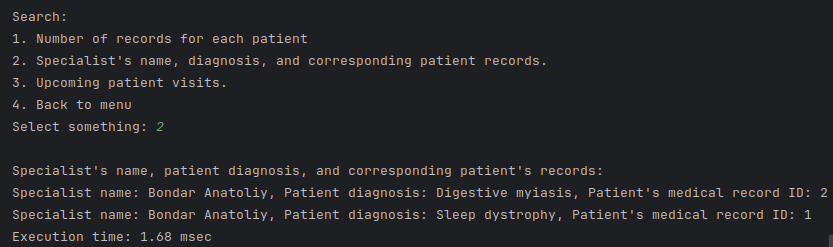
**

**Пошук даних**

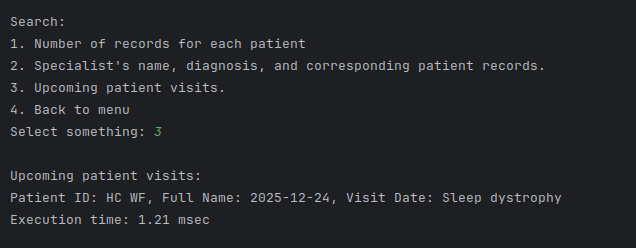
*Перший випадок пошуку*

**

*Другий випадок пошуку*

**

*Третій випадок пошуку*

**

**Шаблон MVC**

MVC визначає архітектурний шаблон програмування, який включає три основні компоненти: Модель (Model), Вид (View) та Контролер (Controller). Цей шаблон дозволяє розділити логічні частини програми, щоб полегшити розробку, управління та розуміння коду. Основні компоненти шаблону MVC:

Model – представляє клас, що описує логіку використовуваних даних. Клас реалізований у файлі model.py, у ньому відбуваються найважчі процеси (вставка, видалення, оновлення, пошук, рандомізація даних, звернення до бази даних) і після виконаної події відправляє результат до View.

View – це консольний інтерфейс, з яким взаємодіє користувач.

Відповідає за введення/виведення даних. У програмі це реалізовано за

допомогою файлу view.py (клас View та клас Menu).

Controller – забезпечує зв’язок між користувачем і системою, поданням і сховищем даних. Він отримує введені користувачем дані і обробляє їх. У програмі це реалізовано у файлі controller.py Використовуючи шаблон Model-View-Controller (MVC), розробники можуть розділити програмний код на логічно пов’язані компоненти, що сприяє полегшенню розуміння, тестуванню та зміні програми. Це особливо важливо для великих проектів, де організованість і підтримка коду відіграють ключову роль.

**Код програми**

***main.py***

from Controller import Controller

if \_\_name\_\_ == "\_\_main\_\_":

controller = Controller()

controller.run()

***model.py***

import psycopg2

class Model:

def \_\_init\_\_(self):

self.conn = psycopg2.connect(

dbname='postgres',

user='postgres',

password='qwerty',

host='localhost',

port=5432

)

def add\_patient(self, id, full\_name, age, contact\_info):

c = self.conn.cursor()

c.execute('INSERT INTO patient(id, full\_name, age, contact\_info) VALUES(%s, %s, %s, %s);',

(id, full\_name, age, contact\_info))

self.conn.commit()

def add\_medication(self, id, id\_patient, name, dosage\_form, side\_effect):

c = self.conn.cursor()

c.execute('INSERT INTO medication(id, id\_patient, name, dosage\_form, side\_effect) VALUES(%s, %s, %s, %s, %s);',

(id, id\_patient, name, dosage\_form, side\_effect))

self.conn.commit()

def add\_medical\_record(self, id, id\_patient, visit\_date, diagnosis, symptoms):

c = self.conn.cursor()

c.execute('INSERT INTO medical\_record(id, id\_patient, visit\_date, diagnosis, symptoms) VALUES(%s, %s, %s, %s, %s);',

(id, id\_patient, visit\_date, diagnosis, symptoms))

self.conn.commit()

def add\_medical\_specialist(self, id, full\_name, qualification, work\_experience, contact\_info):

c = self.conn.cursor()

c.execute('INSERT INTO medical\_specialist(id, full\_name, qualification, work\_experience, contact\_info) VALUES(%s, %s, %s, %s, %s);',

(id, full\_name, qualification, work\_experience, contact\_info))

self.conn.commit()

def add\_specialist\_record(self, id, id\_specialist, id\_record):

c = self.conn.cursor()

c.execute('INSERT INTO specialist\_record(id, id\_specialist, id\_record) VALUES(%s, %s, %s);',

(id, id\_specialist, id\_record))

self.conn.commit()

def get\_patient(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM patient;')

return c.fetchall()

def get\_medication(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM medication;')

return c.fetchall()

def get\_medical\_record(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM medical\_record;')

return c.fetchall()

def get\_medical\_specialist(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM medical\_specialist;')

return c.fetchall()

def get\_specialist\_record(self):

c = self.conn.cursor()

c.execute('SELECT \* FROM specialist\_record;')

return c.fetchall()

def update\_patient(self, id, full\_name, age, contact\_info, id\_new):

c = self.conn.cursor()

c.execute('UPDATE patient SET id=%s, full\_name=%s, age=%s, contact\_info=%s WHERE id=%s',

(id, full\_name, age, contact\_info, id\_new))

self.conn.commit()

def update\_medication(self, id, id\_patient, name, dosage\_form, side\_effect, id\_new):

c = self.conn.cursor()

c.execute('UPDATE medication SET id=%s, id\_patient=%s, name=%s, dosage\_form=%s, side\_effect=%s WHERE id=%s',

(id, id\_patient, name, dosage\_form, side\_effect, id\_new))

self.conn.commit()

def update\_medical\_record(self, id, id\_patient, visit\_date, diagnosis, symptoms, id\_new):

c = self.conn.cursor()

c.execute('UPDATE medical\_record SET id=%s, id\_patient=%s, visit\_date=%s, diagnosis=%s, symptoms=%s WHERE id=%s',

(id, id\_patient, visit\_date, diagnosis, symptoms, id\_new))

self.conn.commit()

def update\_medical\_specialist(self, id, full\_name, qualification, work\_experience, contact\_info, id\_new):

c = self.conn.cursor()

c.execute('UPDATE medical\_specialist SET id=%s, full\_name=%s, qualification=%s, work\_experience=%s, contact\_info=%s WHERE id=%s',

(id, full\_name, qualification, work\_experience, contact\_info, id\_new))

self.conn.commit()

def update\_specialist\_record(self, id, id\_specialist, id\_record, id\_new):

c = self.conn.cursor()

c.execute('UPDATE specialist\_record SET id=%s, id\_specialist=%s, id\_record=%s WHERE id=%s',

(id, id\_specialist, id\_record, id\_new))

self.conn.commit()

def delete\_patient(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM patient WHERE "id"=%s', (id,))

self.conn.commit()

def delete\_medication(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM medication WHERE "id"=%s', (id,))

self.conn.commit()

def delete\_medical\_record(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM medical\_record WHERE "id"=%s', (id,))

self.conn.commit()

def delete\_medical\_specialist(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM medical\_specialist WHERE "id"=%s', (id,))

self.conn.commit()

def delete\_specialist\_record(self, id):

c = self.conn.cursor()

c.execute('DELETE FROM specialist\_record WHERE "id"=%s', (id,))

self.conn.commit()

def get\_number\_of\_records\_per\_patient(self):

c = self.conn.cursor()

c.execute('SELECT p.id AS patient\_id, p.full\_name, COUNT(mr.id) AS total\_records '

'FROM patient p '

'LEFT JOIN medical\_record mr ON p.id = mr.id\_patient '

'GROUP BY p.id, p.full\_name;')

return c.fetchall()

def get\_specialist\_record\_patient(self):

c = self.conn.cursor()

c.execute(

'SELECT DISTINCT mr.id AS record\_id, mr.diagnosis, ms.full\_name AS specialist\_name, p.full\_name AS patient\_name '

'FROM medical\_record mr '

'JOIN specialist\_record sr ON mr.id = sr.id\_record '

'JOIN medical\_specialist ms ON sr.id\_specialist = ms.id '

'JOIN patient p ON mr.id\_patient = p.id;')

return c.fetchall()

def get\_upcoming\_patient\_visits(self):

c = self.conn.cursor()

c.execute('SELECT p.full\_name AS patient\_name, mr.visit\_date, mr.diagnosis '

'FROM medical\_record mr '

'JOIN patient p ON mr.id\_patient = p.id '

'WHERE mr.visit\_date >= CURRENT\_DATE '

'ORDER BY mr.visit\_date LIMIT 5;')

return c.fetchall()

def add\_random\_fields(self, number):

c = self.conn.cursor()

c.execute("""

INSERT INTO patient (id, full\_name, age, contact\_info)

SELECT

row\_number() OVER () + COALESCE((SELECT MAX(id) FROM patient), 0),

chr(trunc(65 + random() \* 26)::int) || chr(trunc(65 + random() \* 26)::int) || ' ' ||

chr(trunc(65 + random() \* 26)::int) || chr(trunc(65 + random() \* 26)::int),

trunc(18 + random() \* 62)::int,

'contact' || trunc(random() \* 1000)::int || '@example.com'

FROM generate\_series(1, %s);

""", (number,))

self.conn.commit()

***view.py***

from datetime import datetime

class View:

def show\_menu(self):

self.show\_message("\nMenu:")

self.show\_message("1. Add row")

self.show\_message('2. Generating `randomized` data (only for "Patient")')

self.show\_message("3. Show table")

self.show\_message("4. Update row")

self.show\_message("5. Delete row")

self.show\_message("6. Search")

self.show\_message("7. Exit")

choice = input("Select your choice: ")

return choice

def show\_tables(self):

self.show\_message("\nTables:")

self.show\_message("1. Patient")

self.show\_message("2. Medication")

self.show\_message("3. Medical record")

self.show\_message("4. Medical specialist")

self.show\_message("5. Specialist record")

self.show\_message("6. Back to menu")

table = input("Select table: ")

return table

def show\_search(self):

self.show\_message("\nSearch:")

self.show\_message("1. Number of records for each patient")

self.show\_message("2. Specialist's name, diagnosis, and corresponding patient records.")

self.show\_message("3. Upcoming patient visits.")

self.show\_message("4. Back to menu")

choice = input("Select something: ")

return choice

def show\_patient(self, patient):

print("\nPatient:")

for pat in patient:

print(f"Patient ID: {pat[0]}, Full name: {pat[1]}, Age: {pat[2]}, Contact info: {pat[3]}")

def show\_medication(self, medication):

print("\nMedication:")

for med in medication:

print(f"Medication ID: {med[0]}, Name: {med[1]}, Dosage form: {med[2]}, Side effect: {med[3]}")

def show\_medical\_record(self, medical\_record):

print("\nMedical record:")

for record in medical\_record:

print(f"Medical record ID: {record[0]}, ID patient: {record[1]}, Visit date: {record[2]}, Diagnosis: {record[3]}, Symptoms: {record[4]}")

def show\_medical\_specialist(self, medical\_specialist):

print("\nMedical specialist:")

for specialist in medical\_specialist:

print(f"Medical specialist ID: {specialist[0]}, Full name: {specialist[1]}, Qualification: {specialist[2]}, Work experience: {specialist[3]}, Contact info: {specialist[4]}")

def show\_specialist\_record(self, specialist\_record):

print("\nSpecialist record:")

for specialist in specialist\_record:

print(f"Specialist record ID: {specialist[0]}, Specialist ID: {specialist[1]}, Record ID: {specialist[2]}")

def get\_patient\_input(self):

while True:

try:

id = input("Enter patient ID: ")

if id.strip():

id = int(id)

break

else:

print("Patient ID cannot be empty.")

except ValueError:

print("It must be a number.")

while True:

try:

full\_name = input("Enter full name: ")

if full\_name.strip():

break

else:

print("Full name cannot be empty.")

except ValueError:

print("It must be a string.")

while True:

try:

age = input("Enter age: ")

if age.strip():

age = int(age)

break

else:

print("Age cannot be empty.")

except ValueError:

print("It must be a number.")

while True:

try:

contact\_info = input("Enter contact info: ")

if contact\_info.strip():

break

else:

print("Contact info cannot be empty.")

except ValueError:

print("It must be a string.")

return id, full\_name, age, contact\_info

def get\_medication\_input(self):

while True:

try:

id = input("Enter Medication ID: ")

if id.strip():

id = int(id)

break

else:

print("Medication ID cannot be empty.")

except ValueError:

print("It must be a number.")

while True:

try:

id\_patient = input("Enter id patient: ")

if id\_patient.strip():

id\_patient = int(id\_patient)

break

else:

print("ID patient cannot be empty.")

except ValueError:

print("It must be a number.")

while True:

try:

name = input("Enter name: ")

if name.strip():

break

else:

print("Name cannot be empty.")

except ValueError:

print("It must be a string.")

while True:

try:

dosage\_form = input("Enter dosage form: ")

if dosage\_form.strip():

break

else:

print("Dosage form cannot be empty.")

except ValueError:

print("It must be a string.")

while True:

try:

side\_effect = input("Enter side effect: ")

if side\_effect.strip():

break

else:

print("Side effect cannot be empty.")

except ValueError:

print("It must be a string.")

return id, id\_patient, name, dosage\_form, side\_effect

def get\_medical\_record\_input(self):

while True:

try:

id = input("Enter medical record ID: ")

if id.strip():

id = int(id)

break

else:

print("Medical record ID cannot be empty.")

except ValueError:

print("It must be a number.")

while True:

try:

id\_patient = input("Enter Patient ID: ")

if id\_patient.strip():

id\_patient = int(id\_patient)

break

else:

print("Patient ID cannot be empty.")

except ValueError:

print("It must be a number.")

while True:

try:

visit\_date = input("Enter state visit date (YYYY-MM-DD): ")

if visit\_date.strip():

datetime.strptime(visit\_date, "%Y-%m-%d")

break

else:

print("State visit date cannot be empty.")

except ValueError:

print("Invalid date format. Please use YYYY-MM-DD.")

while True:

try:

diagnosis = input("Enter diagnosis: ")

if diagnosis.strip():

break

else:

print("Diagnosis cannot be empty.")

except ValueError:

print("It must be a string.")

while True:

try:

symptoms = input("Enter symptoms: ")

if symptoms.strip():

break

else:

print("Symptoms cannot be empty.")

except ValueError:

print("It must be a string.")

return id, id\_patient, visit\_date, diagnosis, symptoms

def get\_medical\_specialist\_input(self):

while True:

try:

id = input("Enter medical specialist ID: ")

if id.strip():

id = int(id)

break

else:

print("Medical specialist ID cannot be empty.")

except ValueError:

print("It must be a number.")

while True:

try:

full\_name = input("Enter full name: ")

if full\_name.strip():

break

else:

print("Full name cannot be empty.")

except ValueError:

print("It must be a string.")

while True:

try:

qualification = input("Enter qualification: ")

if qualification.strip():

break

else:

print("Qualification cannot be empty.")

except ValueError:

print("It must be a string.")

while True:

try:

work\_experience = input("Enter work\_experience: ")

if work\_experience.strip():

work\_experience = int(work\_experience)

break

else:

print("Work\_experience cannot be empty.")

except ValueError:

print("It must be a number.")

while True:

try:

contact\_info = input("Enter contact info: ")

if contact\_info.strip():

break

else:

print("Contact info cannot be empty.")

except ValueError:

print("It must be a string.")

return id, full\_name, qualification, work\_experience, contact\_info

def get\_specialist\_record\_input(self):

while True:

try:

id = input("Enter specialist record ID: ")

if id.strip():

id = int(id)

break

else:

print("Specialist record ID cannot be empty.")

except ValueError:

print("It must be a number.")

while True:

try:

id\_specialist = input("Enter specialist ID: ")

if id\_specialist.strip():

id\_specialist = int(id\_specialist)

break

else:

print("Specialist ID cannot be empty.")

except ValueError:

print("It must be a number.")

while True:

try:

id\_record = input("Enter record ID: ")

if id\_record.strip():

id\_record = int(id\_record)

break

else:

print("Record ID cannot be empty.")

except ValueError:

print("It must be a number.")

return id, id\_specialist, id\_record

def get\_patient\_id(self):

while True:

try:

id = int(input("Enter patient ID: "))

break

except ValueError:

print("It must be a number.")

return id

def get\_medication\_id(self):

while True:

try:

id = int(input("Enter medication ID: "))

break

except ValueError:

print("It must be a number.")

return id

def get\_medical\_record\_id(self):

while True:

try:

id = int(input("Enter medical record ID: "))

break

except ValueError:

print("It must be a number.")

return id

def get\_medical\_specialist\_id(self):

while True:

try:

id = int(input("Enter medical specialist ID: "))

break

except ValueError:

print("It must be a number.")

return id

def get\_specialist\_record\_id(self):

while True:

try:

id = int(input("Enter specialist record ID: "))

break

except ValueError:

print("It must be a number.")

return id

def show\_number\_of\_records\_per\_patient(self, rows):

print("\nNumber of medical records for each patient:")

for row in rows:

print(f"Patient ID: {row[0]}, Full Name: {row[1]}, Total records: {row[2]}")

def show\_specialist\_record\_patient(self, rows):

print("\nSpecialist's name, patient diagnosis, and corresponding patient's records:")

for row in rows:

print(f"Specialist name: {row[2]}, Patient diagnosis: {row[1]}, Patient's medical record ID: {row[0]}")

def show\_upcoming\_patient\_visits(self, rows):

print("\nUpcoming patient visits:")

for row in rows:

print(f"Patient ID: {row[0]}, Full Name: {row[1]}, Visit Date: {row[2]}")

def show\_message(self, message):

print(message)

def get\_number(self):

while True:

try:

number = int(input("Enter the number: "))

break

except ValueError:

print("It must be a number.")

return number

***controller.py***

import time

from Model import Model

from View import View

class Controller:

def \_\_init\_\_(self):

self.model = Model()

self.view = View()

def run(self):

while True:

choice = self.view.show\_menu()

if choice == '7':

break

if choice == '6':

self.process\_search\_option()

elif choice in ['1', '2', '3', '4', '5']:

self.process\_menu\_choice(choice)

else:

self.view.show\_message("Wrong choice. Try again.")

def process\_menu\_choice(self, choice):

while True:

table = self.view.show\_tables()

if table == '6':

break

if choice == '1':

self.process\_add\_option(table)

elif choice == '2':

self.process\_add\_random\_option(table)

elif choice == '3':

self.process\_view\_option(table)

elif choice == '4':

self.process\_update\_option(table)

elif choice == '5':

self.process\_delete\_option(table)

else:

self.view.show\_message("Wrong choice. Try again.")

def process\_add\_option(self, table):

if table == '1':

self.view.show\_message("\nAdding patient:")

self.add\_patient()

elif table == '2':

self.view.show\_message("\nAdding medication:")

self.add\_medication()

elif table == '3':

self.view.show\_message("\nAdding medical record:")

self.add\_medical\_record()

elif table == '4':

self.view.show\_message("\nAdding medical specialist task:")

self.add\_medical\_specialist()

elif table == '5':

self.view.show\_message("\nAdding specialist record task:")

self.add\_specialist\_record()

elif table == '6':

self.view.show\_menu()

else:

self.view.show\_message("Wrong choice. Try again.")

def process\_add\_random\_option(self, table):

if table == '1':

self.view.show\_message("\nAdding random patient:")

self.add\_random\_fields()

else:

self.view.show\_message("Wrong choice. Try again.")

def process\_view\_option(self, table):

if table == '1':

self.show\_patient()

elif table == '2':

self.show\_medication()

elif table == '3':

self.show\_medical\_record()

elif table == '4':

self.show\_medical\_specialist()

elif table == '5':

self.show\_specialist\_record()

elif table == '6':

self.view.show\_menu()

else:

self.view.show\_message("Wrong choice. Try again.")

def process\_update\_option(self, table):

if table == '1':

self.view.show\_message("\nUpdating patient:")

self.update\_patient()

elif table == '2':

self.view.show\_message("\nUpdating medication:")

self.update\_medication()

elif table == '3':

self.view.show\_message("\nUpdating medical record:")

self.update\_medical\_record()

elif table == '4':

self.view.show\_message("\nUpdating medical specialist:")

self.update\_medical\_specialist()

elif table == '5':

self.view.show\_message("\nUpdating specialist record:")

self.update\_specialist\_record()

elif table == '6':

self.view.show\_menu()

else:

self.view.show\_message("Wrong choice. Try again.")

def process\_delete\_option(self, table):

if table == '1':

self.view.show\_message("\nDeleting patient:")

self.delete\_patient()

elif table == '2':

self.view.show\_message("\nDeleting medication:")

self.delete\_medication()

elif table == '3':

self.view.show\_message("\nDeleting medical record:")

self.delete\_medical\_record()

elif table == '4':

self.view.show\_message("\nDeleting medical specialist:")

self.delete\_medical\_specialist()

elif table == '5':

self.view.show\_message("\nDeleting specialist record:")

self.delete\_specialist\_record()

else:

self.view.show\_message("Wrong choice. Try again.")

import time

def process\_search\_option(self):

option = self.view.show\_search()

if option == '1':

start\_time = time.time()

self.show\_number\_of\_records\_per\_patient()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

print(f"Execution time: {elapsed\_time:.2f} msec")

elif option == '2':

start\_time = time.time()

self.show\_specialist\_record\_patient()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

print(f"Execution time: {elapsed\_time:.2f} msec")

elif option == '3':

start\_time = time.time()

self.show\_upcoming\_patient\_visits()

end\_time = time.time()

elapsed\_time = (end\_time - start\_time) \* 1000

print(f"Execution time: {elapsed\_time:.2f} msec")

else:

self.view.show\_menu()

def add\_patient(self):

try:

id, full\_name, age, contact\_info = self.view.get\_patient\_input()

self.model.add\_patient(id, full\_name, age, contact\_info)

self.view.show\_message("Patient added successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def add\_medication(self):

try:

id, id\_patient, name, dosage\_form, side\_effect = self.view.get\_medication\_input()

self.model.add\_medication(id, id\_patient, name, dosage\_form, side\_effect)

self.view.show\_message("Medication added successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def add\_medical\_record(self):

try:

id, id\_patient, visit\_date, diagnosis, symptoms = self.view.get\_medical\_record\_input()

self.model.add\_medical\_record(id, id\_patient, visit\_date, diagnosis, symptoms)

self.view.show\_message("Medical record added successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def add\_medical\_specialist(self):

try:

id, full\_name, qualification, work\_experience, contact\_info = self.view.get\_medical\_specialist\_input()

self.model.add\_medical\_specialist(id, full\_name, qualification, work\_experience, contact\_info)

self.view.show\_message("Medical specialist added successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def add\_specialist\_record(self):

try:

id, id\_specialist, id\_record = self.view.get\_specialist\_record\_input()

self.model.add\_specialist\_record(id, id\_specialist, id\_record)

self.view.show\_message("Specialist record added successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def show\_patient(self):

try:

patient = self.model.get\_patient()

self.view.show\_patient(patient)

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def show\_medication(self):

try:

medication = self.model.get\_medication()

self.view.show\_medication(medication)

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def show\_medical\_record(self):

try:

medical\_record = self.model.get\_medical\_record()

self.view.show\_medical\_record(medical\_record)

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def show\_medical\_specialist(self):

try:

medical\_specialist = self.model.get\_medical\_specialist()

self.view.show\_medical\_specialist(medical\_specialist)

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def show\_specialist\_record(self):

try:

specialist\_record = self.model.get\_specialist\_record()

self.view.show\_specialist\_record(specialist\_record)

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def update\_patient(self):

try:

id\_new = self.view.get\_patient\_id()

id, full\_name, age, contact\_info = self.view.get\_patient\_input()

self.model.update\_patient(id, full\_name, age, contact\_info, id\_new)

self.view.show\_message("Patient updated successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def update\_medication(self):

try:

id\_new = self.view.get\_medication\_id()

id, id\_patient, name, dosage\_form, side\_effect = self.view.get\_medication\_input()

self.model.update\_medication(id, id\_patient, name, dosage\_form, side\_effect, id\_new)

self.view.show\_message("Medication updated successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def update\_medical\_record(self):

try:

id\_new = self.view.get\_medical\_record\_id()

id, id\_patient, visit\_date, diagnosis, symptoms = self.view.get\_medical\_record\_input()

self.model.update\_medical\_record(id, id\_patient, visit\_date, diagnosis, symptoms, id\_new)

self.view.show\_message("Medical record updated successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def update\_medical\_specialist(self):

try:

id\_new = self.view.get\_medical\_specialist\_id()

id, full\_name, qualification, work\_experience, contact\_info = self.view.get\_medical\_specialist\_input()

self.model.update\_medical\_specialist(id, full\_name, qualification, work\_experience, contact\_info, id\_new)

self.view.show\_message("Medical specialist updated successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def update\_specialist\_record(self):

try:

id\_new = self.view.get\_specialist\_record\_id()

id, id\_specialist, id\_record = self.view.get\_specialist\_record\_input()

self.model.update\_specialist\_record(id, id\_specialist, id\_record, id\_new)

self.view.show\_message("Specialist record updated successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def delete\_patient(self):

try:

id = self.view.get\_patient\_id()

self.model.delete\_patient(id)

self.view.show\_message("Patient deleted successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def delete\_medication(self):

try:

id = self.view.get\_medication\_id()

self.model.delete\_medication(id)

self.view.show\_message("Medication deleted successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def delete\_medical\_record(self):

try:

id = self.view.get\_medical\_record\_id()

self.model.delete\_medical\_record(id)

self.view.show\_message("Medical record deleted successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def delete\_medical\_specialist(self):

try:

id = self.view.get\_medical\_specialist\_id()

self.model.delete\_medical\_specialist(id)

self.view.show\_message("Medical specialist deleted successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def delete\_specialist\_record(self):

try:

id = self.view.get\_specialist\_record\_id()

self.model.delete\_specialist\_record(id)

self.view.show\_message("Specialist record deleted successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def show\_number\_of\_records\_per\_patient(self):

try:

records\_per\_patient = self.model.get\_number\_of\_records\_per\_patient()

self.view.show\_number\_of\_records\_per\_patient(records\_per\_patient)

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def show\_specialist\_record\_patient(self):

try:

specialist\_record\_patient = self.model.get\_specialist\_record\_patient()

self.view.show\_specialist\_record\_patient(specialist\_record\_patient)

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def show\_upcoming\_patient\_visits(self):

try:

upcoming\_patient\_visits = self.model.get\_upcoming\_patient\_visits()

self.view.show\_upcoming\_patient\_visits(upcoming\_patient\_visits)

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")

def add\_random\_fields(self):

try:

number = self.view.get\_number()

self.model.add\_random\_fields(number)

self.view.show\_message("Random fields added successfully!")

except Exception as e:

self.view.show\_message(f"Something went wrong: {e}")