НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

Розрахунково-графічна робота

з дисципліни

«Бази даних та засоби управління»

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

Виконала: студентка групи КВ-11
Пильова Діана

Telegram: @smesharik29

Київ – 2023

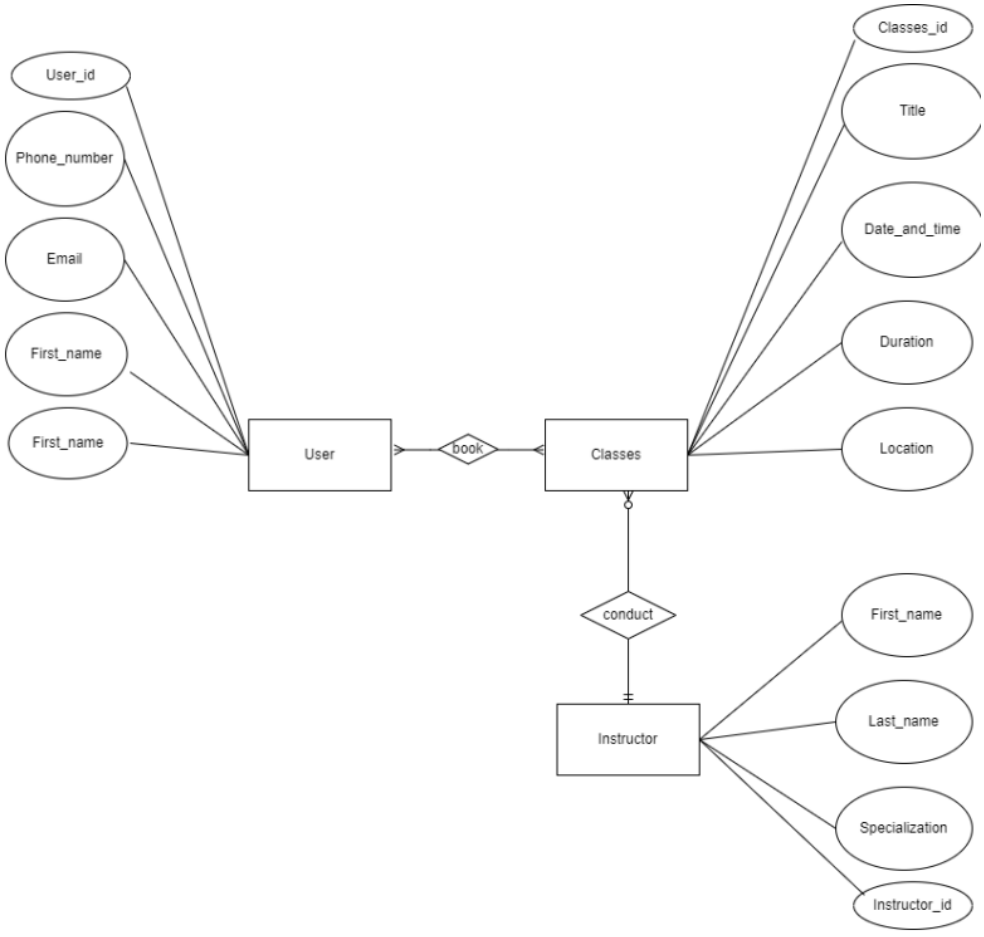
Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

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).

Посилання на репозиторій в гітхаб: https://github.com/pylova/DataBase

ER-діаграма, побудована за нотацією Чена



Сутності з описом призначення

Для побудови бази даних предметної області було виділено декілька сутностей:

1. Заняття (Classes) з атрибутами: назва заняття, дата і час проведення, тривалість, місце проведення, індифікаційний номер.

2. Користувач (User) з атрибутами: ім'я, прізвище, електронна адреса, номер телефону, індифікаційний номер.

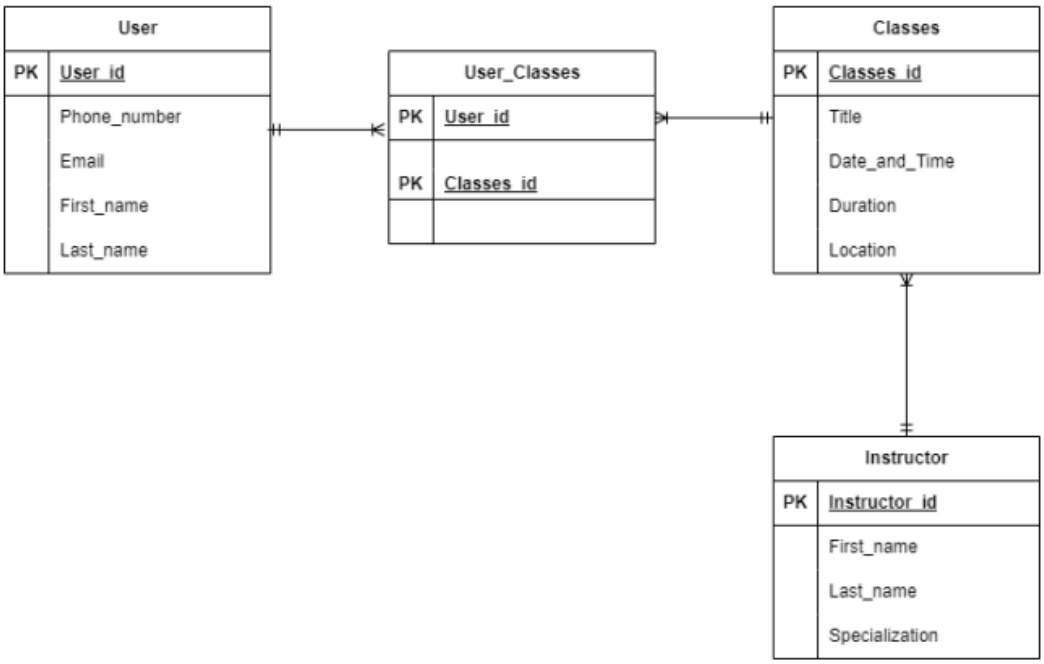
3. Тренер (Instructor) з атрибутами: ім'я, прізвище, спеціалізація, індифікаційний номер.

Опис зв’язків між сутностями предметної області

Сутність "Заняття" встановлює зв'язок багато-до-багатьох (M:N) з сутністю "Користувач", оскільки кожне заняття може мати багатьох користувачів, які на нього записані, і кожен користувач може бути записаний на багато занять.

Сутність "Заняття" має зв'язок один-до-багатьох (1:N) з сутністю "Тренер", оскільки кожне заняття має одного тренера, який його проводить, і один тренер може проводити багато занять.

Схема бази даних у графічному вигляді



Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.10.

Середовище розробки програмного забезпечення – PyCharm Community Edition.

Бібліотека взаємодії з PostgreSQL - psycopg2.

```
Menu:
1. Add row
2. Generating `randomized` data (only for "Classes")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice:
```

```
Tables:
1. Users
2. Classes
3. Instructors
4. User Classes
5. Back to menu
Select table:
```

Меню складається з 7 пунктів:

Перший пункт – **Add row** служить для додавання рядка в таблицю. Після вибору цього пункту, потрібно обрати таблицю, для якої буде виконана ця операція, після чого, треба ввести дані для кожного атрибуту таблиці, щоб додати новий рядок.

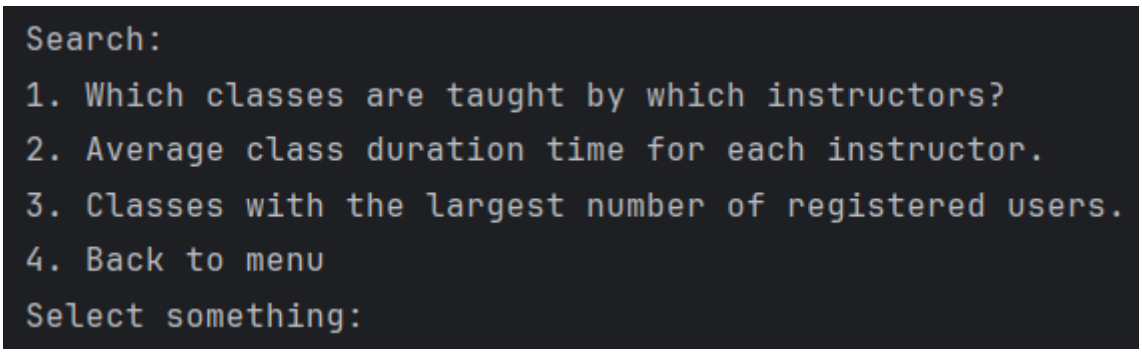
Другий пункт – **Generating `randomized` data**. Для цього пункту було обрано таблицю Classes. Цей пункт створений для додавання «рандомізованих» даних. Потрібно ввести число полів, яке ми хочемо додати.

Третій пункт – **Show table** служить для показу таблиці. Перед виведенням, користувач обирає, яку саме таблицю потрібно вивести. Після цього на екрані виводяться всі поля обраної таблиці БД.

Четвертий пункт – **Update row** використовується для редагування полів по id у таблицях. Спочатку потрібно обрати, для якої таблиці буде відбуватися, після чого потрібно ввести id поля, яке потрібно змінити. Залишається ввести нові дані для кожного атрибуту таблиці.

П’ятий пункт – **Delete row** служить для видалення рядку по id у таблицях. Спочатку потрібно обрати, для якої таблиці буде відбуватися видалення рядка, після чого користувач вводить id рядка, який потрібно видалити.

Шостий пункт – **Search** створений для пошуку за атрибутами з декількох таблиць. Пропонується 4 варіанти вибору:



1. Which classes are taught by which instructors? Які заняття проводять які інструктори?
   При виборі цього пункту, буде виведена таблиця з прізвищами та іменами інструкторів та назви класів, які вони викладають.
2. Average class duration time for each instructor. Середня тривалість заняття для кожного викладача.
   При виборі цього пункту, буде виведена таблиця з id викладача, його прізвищем та іменем, а також середня тривалість занять у цього викладача.
3. Classes with the largest number of registered users. Класи з найбільшою кількістю підписаних юзерів.
   При виборі цього пункту, буде виведена таблиця з назвою класу, а також кількість юзерів, що обрали цей клас.
4. Back to menu. Повернення до основного меню.

Також можна побачити час виконання запиту у мілісекундах, після виведення даних.

Сьомий пункт – **Exit** служить для виходу з програми.

Завдання 1

Додавання рядка

Таблиця “User” до:

	User_id [PK] integer	Phone_number character varying (15)	Email character varying (255)	First_name character varying (255)	Last_name character varying (255)
1	1	0997328921	qkwk@gmail.com	Steven	Doyle
2	2	0962683804	hdjf@gmail.com	Niall	James
3	3	555	test	test	test
4	4	0116745491	O	VS	MO
5	5	0339144684	2f8@example.com	A	TL
6	7	0997328921	one@gmail.com	Petro	Peter
7	8	0962683804	nastya@gmail.com	Nastya	Turenko
8	9	0912345678	katya@gmail.com	Katya	Maistrenko
9	10	0987654321	mykola@gmail.com	Mykola	Koval
10	11	0985555555	lev@gmail.com	Lev	Dan
11	12	0976543210	dima@gmail.com	Dima	Bogdan
12	13	0981111111	masha@gmail.com	Masha	Teteruk

```
Menu:
1. Add row
2. Generating `randomized` data (only for "Classes")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 1
```

```
Tables:
1. Users
2. Classes
3. Instructors
4. User Classes
5. Back to menu
Select table: 1
```

```
Adding user:
Enter user_id: 14
Enter phone number: 0637441228
Enter email: newyear2024@gmail.com
Enter first name: Name1
Enter last name: Surname1
User added successfully!
```

Таблиця “User” після:

	User_id [PK] integer	Phone_number character varying (15)	Email character varying (255)	First_name character varying (255)	Last_name character varying (255)
1	1	0997328921	qkwk@gmail.com	Steven	Doyle
2	2	0962683804	hdjf@gmail.com	Niall	James
3	3	555	test	test	test
4	4	0116745491	O	VS	MO
5	5	0339144684	2f8@example.com	A	TL
6	7	0997328921	one@gmail.com	Petro	Peter
7	8	0962683804	nastya@gmail.com	Nastya	Turenko
8	9	0912345678	katya@gmail.com	Katya	Maistrenko
9	10	0987654321	mykola@gmail.com	Mykola	Koval
10	11	0985555555	lev@gmail.com	Lev	Dan
11	12	0976543210	dima@gmail.com	Dima	Bogdan
12	13	0981111111	masha@gmail.com	Masha	Teteruk
13	14	0637441228	newyear2024@gmail.com	Name1	Surname1

Перегляд таблиці

Таблиця “Instructor”:

	Instructor_id [PK] integer	First_name character varying (255)	Last_name character varying (255)	Classes_id integer
1	1	Conor	Smith	1
2	3	name2	lastname2	1
3	4	a	a	1
4	5	Conor	Smith	5
5	6	Sophie	Lee	10
6	7	Michael	Johnson	5
7	8	Emma	White	5
8	9	Alex	Taylor	1
9	10	Grace	Brown	1
10	11	Daniel	Miller	2
11	12	Olivia	Wilson	5
12	13	William	Clark	2
13	14	Sophia	Moore	2

```
Menu:
1. Add row
2. Generating `randomized` data (only for "Classes")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 3
```

```
Tables:
1. Users
2. Classes
3. Instructors
4. User Classes
5. Back to menu
Select table: 3
```

```
Instructors:
Instructor_id: 1, First_name: Conor, Last_name: Smith, Specialization: 1
Instructor_id: 3, First_name: name2, Last_name: lastname2, Specialization: 1
Instructor_id: 4, First_name: a, Last_name: a, Specialization: 1
Instructor_id: 5, First_name: Conor, Last_name: Smith, Specialization: 5
Instructor_id: 6, First_name: Sophie, Last_name: Lee, Specialization: 10
Instructor_id: 7, First_name: Michael, Last_name: Johnson, Specialization: 5
Instructor_id: 8, First_name: Emma, Last_name: White, Specialization: 5
Instructor_id: 9, First_name: Alex, Last_name: Taylor, Specialization: 1
Instructor_id: 10, First_name: Grace, Last_name: Brown, Specialization: 1
Instructor_id: 11, First_name: Daniel, Last_name: Miller, Specialization: 2
Instructor_id: 12, First_name: Olivia, Last_name: Wilson, Specialization: 5
Instructor_id: 13, First_name: William, Last_name: Clark, Specialization: 2
Instructor_id: 14, First_name: Sophia, Last_name: Moore, Specialization: 2
```

Оновлення рядка

Таблиця “Instructor” до:

	Instructor_id [PK] integer	First_name character varying (255)	Last_name character varying (255)	Classes_id integer
2	3	name2	lastname2	1
3	4	a	a	1
4	5	Conor	Smith	5
5	6	Sophie	Lee	10
6	7	Michael	Johnson	5
7	8	Emma	White	5
8	9	Alex	Taylor	1
9	10	Grace	Brown	1
10	11	Daniel	Miller	2
11	12	Olivia	Wilson	5
12	13	William	Clark	2
13	14	Sophia	Moore	2
14	15	Test1	Test1	777

```
Menu:
1. Add row
2. Generating `randomized` data (only for "Classes")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 4
```

```
Tables:
1. Users
2. Classes
3. Instructors
4. User Classes
5. Back to menu
Select table: 3
```

```
Updating instructor:
Enter instructor ID: 15
Enter instructor_id: 15
Enter first name: Test2
Enter last name: Test2
Enter class_id: 999
Instructor updated successfully!
```

Таблиця “Instructor” після:

	Instructor_id [PK] integer	First_name character varying (255)	Last_name character varying (255)	Classes_id integer
2	3	name2	lastname2	1
3	4	a	a	1
4	5	Conor	Smith	5
5	6	Sophie	Lee	10
6	7	Michael	Johnson	5
7	8	Emma	White	5
8	9	Alex	Taylor	1
9	10	Grace	Brown	1
10	11	Daniel	Miller	2
11	12	Olivia	Wilson	5
12	13	William	Clark	2
13	14	Sophia	Moore	2
14	15	Test2	Test2	999

Видалення рядка

Таблиця “User_Classes” до:

	User_id [PK] integer	Classes_id [PK] integer
1	1	1
2	1	2
3	5	5

```
Menu:
1. Add row
2. Generating `randomized` data (only for "Classes")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 5
```

```
Tables:
1. Users
2. Classes
3. Instructors
4. User Classes
5. Back to menu
Select table: 4
```

```
Deleting user class:
Enter user ID: 5
User Class deleted successfully!
```

Таблиця “User_Classes” після:

	User_id [PK] integer	Classes_id [PK] integer
1	1	1
2	1	2

Вихід

```
Menu:
1. Add row
2. Generating `randomized` data (only for "Classes")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 7

Process finished with exit code 0
```

Завдання 2

Генерування «рандомізованих» даних

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

```
WITH max_id AS (
    SELECT COALESCE(MAX("Classes_id"), 0) FROM public."Classes"
)
INSERT INTO public."Classes" ("Classes_id", "Title", "Date_and_Time", "Duration", "Location")
SELECT (SELECT * FROM max_id) + row_number() OVER () as "Classes_id",
       chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int),
       current_date + (random() * 30)::integer * '1 day'::interval,
       random() * 100,
       chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int)
FROM generate_series(1, %s);
```

%s - підставляється число, яке вводиться користувачем, скільки рядків хоче додати користувач в таблицю.

Таблиця “Classes” до:

	Classes_id [PK] integer	Title character varying (255)	Date_and_Time date	Duration integer	Location character varying (255)
1	100013	JV	2024-01-03	7	BA
2	100012	YE	2023-12-27	68	NS
3	100011	KM	2024-01-05	66	EJ
4	100010	NT	2024-01-16	49	PN
5	100009	PV	2023-12-26	9	RM
6	100008	OP	2024-01-13	1	TA
7	100007	TF	2023-12-28	69	ME
8	100006	XV	2023-12-21	30	MU
9	100005	PK	2024-01-02	41	YX
10	100004	DU	2024-01-05	99	DG
11	100003	UN	2024-01-14	89	GQ
12	100002	MI	2024-01-12	14	FJ
13	100001	ND	2024-01-04	50	XS

	Classes_id [PK] integer	Title character varying (255)	Date_and_Time date	Duration integer	Location character varying (255)
27	100000	CM	2024-01-17	39	QI
28	99999	RF	2024-01-01	24	SI
29	99998	GH	2024-01-05	55	DD
30	99997	HB	2023-12-26	98	KG
31	99996	VJ	2024-01-11	40	AR
32	99995	AJ	2024-01-18	48	YV
33	99994	QQ	2023-12-27	49	GW
34	99993	GR	2024-01-17	23	KJ
35	99992	IH	2024-01-15	15	DA
36	99991	CU	2024-01-03	40	XO
37	99990	QO	2024-01-05	3	SB
38	99989	QC	2023-12-27	96	TY
39	99988	LG	2024-01-08	95	UH
40	99987	JJ	2023-12-22	32	MS
41	99986	NJ	2024-01-16	55	LX
42	99985	YS	2024-01-12	82	ST
43	99984	FQ	2023-12-25	44	MD
44	99983	JW	2023-12-30	94	VX
45	99982	VJ	2023-12-30	80	DL
46	99981	GK	2024-01-09	48	VS
47	99980	SD	2023-12-30	53	OA
48	99979	MB	2024-01-01	4	EC

	Classes_id [PK] integer	Title character varying (255)	Date_and_Time date	Duration integer	Location character varying (255)
49	99978	FY	2024-01-17	33	WK
50	99977	QI	2024-01-14	4	PW
51	99976	EQ	2024-01-13	14	WY
52	99975	CB	2024-01-09	30	GJ
53	99974	CS	2024-01-03	83	YV
54	99973	QP	2023-12-30	70	IB
55	99972	OT	2024-01-02	55	YH
56	99971	BN	2023-12-31	17	QC
57	99970	SF	2023-12-20	50	CY
58	99969	IT	2023-12-26	9	TX
59	99968	LK	2024-01-16	86	WL
60	99967	CQ	2024-01-17	69	ML
61	99966	UU	2023-12-22	59	BS
62	99965	AN	2024-01-09	21	TG
63	99964	RO	2024-01-02	3	IF
64	99963	FB	2024-01-12	65	XA
65	99962	FO	2024-01-07	31	UO
66	99961	XC	2023-12-22	34	QH
67	99960	BS	2024-01-14	95	KV
68	99959	GW	2024-01-10	89	EN
69	99958	NB	2024-01-04	90	LR
70	99957	MW	2024-01-09	89	AY

```
Menu:
1. Add row
2. Generating `randomized` data (only for "Classes")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 2
```

```
Tables:
1. Users
2. Classes
3. Instructors
4. User Classes
5. Back to menu
Select table: 2
```

```
Adding random classes:
Enter the number: 10
Random fields added successfully!
```

Таблиця “Classes” після:

	Classes_id [PK] integer	Title character varying (255)	Date_and_Time date	Duration integer	Location character varying (255)
1	100023	UY	2024-01-16	33	SB
2	100022	MI	2023-12-29	99	SR
3	100021	XD	2024-01-01	48	LS
4	100020	CI	2023-12-24	53	OX
5	100019	PN	2023-12-26	32	LY
6	100018	MC	2024-01-10	34	MH
7	100017	TS	2023-12-31	14	AS
8	100016	PD	2024-01-09	12	GS
9	100015	OD	2023-12-21	64	HH
10	100014	KE	2023-12-27	49	ED
11	100013	JV	2024-01-03	7	BA
12	100012	YE	2023-12-27	68	NS
13	100011	KM	2024-01-05	66	EJ

Завдання 3

Пошук даних

```
Menu:
1. Add row
2. Generating `randomized` data (only for "Classes")
3. Show table
4. Update row
5. Delete row
6. Search
7. Exit
Select your choice: 6
```

1.

```
Search:
1. Which classes are taught by which instructors?
2. Average class duration time for each instructor.
3. Classes with the largest number of registered users.
4. Back to menu
Select something: 1
```

```
Instructors-Classes:
Instructor name: Conor, Instructor surname: Smith, Class: cardio
Instructor name: name2, Instructor surname: lastname2, Class: cardio
Instructor name: a, Instructor surname: a, Class: cardio
Instructor name: Alex, Instructor surname: Taylor, Class: cardio
Instructor name: Grace, Instructor surname: Brown, Class: cardio
Instructor name: Test2, Instructor surname: Test2, Class: DA
Instructor name: Daniel, Instructor surname: Miller, Class: dances
Instructor name: William, Instructor surname: Clark, Class: dances
Instructor name: Sophia, Instructor surname: Moore, Class: dances
Instructor name: Sophie, Instructor surname: Lee, Class: GJ
Instructor name: Olivia, Instructor surname: Wilson, Class: KL
Instructor name: Emma, Instructor surname: White, Class: KL
Instructor name: Conor, Instructor surname: Smith, Class: KL
Instructor name: Michael, Instructor surname: Johnson, Class: KL
Execution time: 4.98 msec
```

2.

```
Search:
1. Which classes are taught by which instructors?
2. Average class duration time for each instructor.
3. Classes with the largest number of registered users.
4. Back to menu
Select something: 2
```

```
Average class duration time for each instructor:
Instructor ID: 1, Instructor name: Conor, Instructor surname: Smith, Average duration: 60.0000000000000000
Instructor ID: 3, Instructor name: name2, Instructor surname: lastname2, Average duration: 60.0000000000000000
Instructor ID: 4, Instructor name: a, Instructor surname: a, Average duration: 60.0000000000000000
Instructor ID: 5, Instructor name: Conor, Instructor surname: Smith, Average duration: 9.0000000000000000
Instructor ID: 6, Instructor name: Sophie, Instructor surname: Lee, Average duration: 31.0000000000000000
Instructor ID: 7, Instructor name: Michael, Instructor surname: Johnson, Average duration: 9.0000000000000000
Instructor ID: 8, Instructor name: Emma, Instructor surname: White, Average duration: 9.0000000000000000
Instructor ID: 9, Instructor name: Alex, Instructor surname: Taylor, Average duration: 60.0000000000000000
Instructor ID: 10, Instructor name: Grace, Instructor surname: Brown, Average duration: 60.0000000000000000
Instructor ID: 11, Instructor name: Daniel, Instructor surname: Miller, Average duration: 45.0000000000000000
Instructor ID: 12, Instructor name: Olivia, Instructor surname: Wilson, Average duration: 9.0000000000000000
Instructor ID: 13, Instructor name: William, Instructor surname: Clark, Average duration: 45.0000000000000000
Instructor ID: 14, Instructor name: Sophia, Instructor surname: Moore, Average duration: 45.0000000000000000
Instructor ID: 15, Instructor name: Test2, Instructor surname: Test2, Average duration: 11.0000000000000000
Execution time: 5.96 msec
```

3.

```
Search:
1. Which classes are taught by which instructors?
2. Average class duration time for each instructor.
3. Classes with the largest number of registered users.
4. Back to menu
Select something: 3
```

```
Classes with the largest number of registered users:
Title: cardio, User count: 1
Title: dances, User count: 1
Title: EN, User count: 0
Title: FG, User count: 0
Title: RU, User count: 0
Execution time: 38.90 msec
```

Завдання 4

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

Користуючись шаблоном MVC, розробники можуть розділити програмний код на логічно зв'язані компоненти, що полегшує розуміння, тестування та зміну програми. Це особливо корисно для великих проектів, де структурованість і підтримка коду грають важливу 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='Sport_Classes',
            user='postgres',
            password='1111',
            host='localhost',
            port=3000
        )

    def add_user(self, user_id, phone_number, email, first_name,
last_name):
        c = self.conn.cursor()
        c.execute('INSERT INTO public."User"("User_id", "Phone_number",
"Email", "First_name", "Last_name") VALUES(%s, %s, %s, %s, %s);', (user_id,
phone_number, email, first_name, last_name))
        self.conn.commit()

    def add_class(self, classes_id, title, date_and_Time, duration,
location):
        c = self.conn.cursor()
        c.execute('INSERT INTO public."Classes"("Classes_id", "Title",
"Date_and_Time", "Duration", "Location") VALUES(%s, %s, %s, %s, %s);',
(classes_id, title, date_and_Time, duration, location))
        self.conn.commit()

    def add_instructor(self, instructor_id, first_name, last_name,
classes_id):
        c = self.conn.cursor()
        c.execute('INSERT INTO public."Instructor"("Instructor_id",
"First_name", "Last_name", "Classes_id") VALUES(%s, %s, %s, %s);',
(instructor_id, first_name, last_name, classes_id))
        self.conn.commit()

    def add_user_class(self, user_id, classes_id):
        c = self.conn.cursor()
        c.execute('INSERT INTO public."User_Classes"("User_id",
"Classes_id") VALUES(%s, %s);', (user_id, classes_id))
        self.conn.commit()

    def get_users(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM public."User";')
        return c.fetchall()

    def get_classes(self):
        c = self.conn.cursor()
```

```
        c.execute('SELECT * FROM public."Classes";')
        return c.fetchall()

    def get_instructors(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM public."Instructor";')
        return c.fetchall()

    def get_user_classes(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM public."User_Classes";')
        return c.fetchall()

    def get_instructor_classes(self):
        c = self.conn.cursor()
        c.execute('SELECT "Instructor"."First_name" AS
"Instructor_FirstName", "Instructor"."Last_name" AS "Instructor_LastName",
"Classes"."Title" FROM "Instructor" JOIN "Classes" ON
"Instructor"."Classes_id" = "Classes"."Classes_id" ORDER BY
"Classes"."Title";')
        return c.fetchall()

    def get_average_class_duration(self):
        c = self.conn.cursor()
        c.execute('SELECT "Instructor"."Instructor_id",
"Instructor"."First_name", "Instructor"."Last_name",
AVG("Classes"."Duration") AS "Average_Duration" FROM "Instructor" JOIN
"Classes" ON "Instructor"."Classes_id" = "Classes"."Classes_id" GROUP BY
"Instructor"."Instructor_id", "Instructor"."First_name",
"Instructor"."Last_name";')
        return c.fetchall()

    def get_classes_with_number_of_users(self):
        c = self.conn.cursor()
        c.execute('SELECT "Classes"."Title",
COUNT("User_Classes"."User_id") AS "User_Count" FROM "Classes" LEFT JOIN
"User_Classes" ON "Classes"."Classes_id" = "User_Classes"."Classes_id"
GROUP BY "Classes"."Title" ORDER BY "User_Count" DESC LIMIT 5;')
        return c.fetchall()

    def update_user(self, user_id, phone_number, email, first_name,
last_name, id):
        c = self.conn.cursor()
        c.execute('UPDATE public."User" SET User_id=%s, Phone_number=%s,
Email=%s, First_name=%s, Last_name=%s WHERE User_id=%s', (user_id,
phone_number, email, first_name, last_name, id))
        self.conn.commit()

    def update_class(self, classes_id, title, date_time, duration,
location, id):
        c = self.conn.cursor()
        c.execute('UPDATE public."Classes" SET "Classes_id"=%s, "Title"=%s,
"Date_and_Time"=%s, "Duration"=%s, "Location"=%s WHERE "Classes_id"=%s',
(classes_id, title, date_time, duration, location, id))
        self.conn.commit()

    def update_instructor(self, instructor_id, first_name, last_name,
classes_id, id):
        c = self.conn.cursor()
        c.execute('UPDATE public."Instructor" SET "Instructor_id"=%s,
"First_name"=%s, "Last_name"=%s, "Classes_id"=%s WHERE "Instructor_id"=%s',
(instructor_id, first_name, last_name, classes_id, id))
        self.conn.commit()
```

```
    def update_user_class(self, user_id, classes_id, id):
        c = self.conn.cursor()
        c.execute('UPDATE public."User_Classes" SET "User_id"=%s,
"Classes_id"=%s WHERE "User_id"=%s', (user_id, classes_id, id))
        self.conn.commit()

    def delete_user(self, user_id):
        c = self.conn.cursor()
        c.execute('DELETE FROM public."User" WHERE "User_id"=%s',
(user_id,))
        self.conn.commit()

    def delete_class(self, class_id):
        c = self.conn.cursor()
        c.execute('DELETE FROM public."Classes" WHERE "Classes_id"=%s',
(class_id,))
        self.conn.commit()

    def delete_instructor(self, instructor_id):
        c = self.conn.cursor()
        c.execute('DELETE FROM public."Instructor" WHERE
"Instructor_id"=%s', (instructor_id,))
        self.conn.commit()

    def delete_user_class(self, user_id):
        c = self.conn.cursor()
        c.execute('DELETE FROM public."User_Classes" WHERE "User_id"=%s',
(user_id,))
        self.conn.commit()

    def add_random_fields(self, number):
        c = self.conn.cursor()
        c.execute("""
            WITH max_id AS (
                SELECT COALESCE(MAX("Classes_id"), 0) FROM public."Classes"
            )
            INSERT INTO public."Classes" ("Classes_id", "Title",
"Date_and_Time", "Duration", "Location")
            SELECT (SELECT * FROM max_id) + row_number() OVER () as
"Classes_id",
                   chr(trunc(65 + random() * 25)::int) || chr(trunc(65 +
random() * 25)::int),
                   current_date + (random() * 30)::integer * '1
day'::interval,
                   random() * 100,
                   chr(trunc(65 + random() * 25)::int) || chr(trunc(65 +
random() * 25)::int)
            FROM generate_series(1, %s);
        """, (number,))
        self.conn.commit()
```

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 user:")
        self.add_user()
    elif table == '2':
        self.view.show_message("\nAdding class:")
        self.add_class()
    elif table == '3':
        self.view.show_message("\nAdding instructor:")
        self.add_instructor()
    elif table == '4':
        self.view.show_message("\nAdding user class:")
        self.add_user_class()
    elif table == '5':
        self.view.show_menu()
    else:
        self.view.show_message("Wrong choice. Try again.")

def process_add_random_option(self, table):
    if table == '2':
        self.view.show_message("\nAdding random classes:")
        self.add_random_fields()
    else:
        self.view.show_message("Wrong choice. Try again.")

def process_view_option(self, table):
    if table == '1':
        self.show_users()
    elif table == '2':
        self.show_classes()
    elif table == '3':
```

```
            self.show_instructors()
        elif table == '4':
            self.show_user_classes()
        elif table == '5':
            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 user:")
            self.update_user()
        elif table == '2':
            self.view.show_message("\nUpdating class:")
            self.update_class()
        elif table == '3':
            self.view.show_message("\nUpdating instructor:")
            self.update_instructor()
        elif table == '4':
            self.view.show_message("\nUpdating user class:")
            self.update_user_class()
        elif table == '5':
            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 user:")
            self.delete_user()
        elif table == '2':
            self.view.show_message("\nDeleting class:")
            self.delete_class()
        elif table == '3':
            self.view.show_message("\nDeleting instructor:")
            self.delete_instructor()
        elif table == '4':
            self.view.show_message("\nDeleting user class:")
            self.delete_user_class()
        else:
            self.view.show_message("Wrong choice. Try again.")

    def process_search_option(self):
        option = self.view.show_search()

        if option == '1':
            start_time = time.time()
            self.show_instructors_with_classes()
            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_average_class_duration()
            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_classes_with_number_of_users()
            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_user(self):
        try:
            user_id, phone_number, email, first_name, last_name =
self.view.get_user_input()
            self.model.add_user(user_id, phone_number, email, first_name,
last_name)
            self.view.show_message("User added successfully!")
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def add_class(self):
        try:
            class_id, title, date_time, duration, location =
self.view.get_class_input()
            self.model.add_class(class_id, title, date_time, duration,
location)
            self.view.show_message("Class added successfully!")
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def add_instructor(self):
        try:
            instructor_id, first_name, last_name, class_id =
self.view.get_instructor_input()
            self.model.add_instructor(instructor_id, first_name, last_name,
class_id)
            self.view.show_message("Instructor added successfully!")
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def add_user_class(self):
        try:
            user_id, classes_id = self.view.get_user_class_input()
            self.model.add_user_class(user_id, classes_id)
            self.view.show_message("User Class added successfully!")
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def show_users(self):
        try:
            users = self.model.get_users()
            self.view.show_users(users)
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def show_classes(self):
        try:
            classes = self.model.get_classes()
            self.view.show_classes(classes)
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def show_instructors(self):
        try:
            instructors = self.model.get_instructors()
            self.view.show_instructors(instructors)
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def show_user_classes(self):
        try:
            user_classes = self.model.get_user_classes()
```

```
            self.view.show_user_classes(user_classes)
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def show_instructors_with_classes(self):
        try:
            instructor_classes = self.model.get_instructor_classes()
            self.view.show_instructors_with_classes(instructor_classes)
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def show_average_class_duration(self):
        try:
            average_class_duration =
self.model.get_average_class_duration()
            self.view.show_average_class_duration(average_class_duration)
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def show_classes_with_number_of_users(self):
        try:
            rows = self.model.get_classes_with_number_of_users()
            self.view.show_classes_with_number_of_users(rows)
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def update_user(self):
        try:
            id = self.view.get_user_id()
            user_id, phone_number, email, first_name, last_name =
self.view.get_user_input()
            self.model.update_user(user_id, phone_number, email,
first_name, last_name, id)
            self.view.show_message("User updated successfully!")
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def update_instructor(self):
        try:
            id = self.view.get_instructor_id()
            instructor_id, first_name, last_name, class_id =
self.view.get_instructor_input()
            self.model.update_instructor(instructor_id, first_name,
last_name, class_id, id)
            self.view.show_message("Instructor updated successfully!")
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def update_class(self):
        try:
            id = self.view.get_class_id()
            classes_id, title, date_time, duration, location =
self.view.get_class_input()
            self.model.update_class(classes_id, title, date_time, duration,
location, id)
            self.view.show_message("Class updated successfully!")
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def update_user_class(self):
        try:
            id = self.view.get_user_id()
            user_id, class_id = self.view.get_user_class_input()
            self.model.update_user_class(user_id, class_id, id)
```

```
            self.view.show_message("User Class updated successfully!")
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def delete_user(self):
        try:
            user_id = self.view.get_user_id()
            self.model.delete_user(user_id)
            self.view.show_message("User deleted successfully!")
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def delete_class(self):
        try:
            class_id = self.view.get_class_id()
            self.model.delete_class(class_id)
            self.view.show_message("Class deleted successfully!")
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def delete_instructor(self):
        try:
            instructor_id = self.view.get_instructor_id()
            self.model.delete_instructor(instructor_id)
            self.view.show_message("Instructor deleted successfully!")
        except Exception as e:
            self.view.show_message(f"Something went wrong: {e}")

    def delete_user_class(self):
        try:
            user_id = self.view.get_user_id()
            self.model.delete_user_class(user_id)
            self.view.show_message("User Class deleted successfully!")
        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}")
```

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
"Classes")')
        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. Users")
        self.show_message("2. Classes")
        self.show_message("3. Instructors")
        self.show_message("4. User Classes")
        self.show_message("5. Back to menu")
        table = input("Select table: ")
        return table

    def show_search(self):
        self.show_message("\nSearch:")
        self.show_message("1. Which classes are taught by which
instructors?")
        self.show_message("2. Average class duration time for each
instructor.")
        self.show_message("3. Classes with the largest number of registered
users.")
        self.show_message("4. Back to menu")
        choice = input("Select something: ")
        return choice

    def show_users(self, users):
        print("\nUsers:")
        for user in users:
            print(f"User_id: {user[0]}, Phone_number: {user[1]}, Email:
{user[2]}, First_name: {user[3]}, Last_name: {user[4]}")


    def show_classes(self, classes):
        print("\nClasses:")
        for clas in classes:
            print(f"Class_id: {clas[0]}, Title: {clas[1]}, Date_and_Time:
{clas[2]}, Duration: {clas[3]}, Location: {clas[4]}")

    def show_instructors(self, instructors):
        print("\nInstructors:")
        for instructor in instructors:
            print(f"Instructor_id: {instructor[0]}, First_name:
{instructor[1]}, Last_name: {instructor[2]}, Specialization:
{instructor[3]}")

    def show_user_classes(self, user_classes):
        print("\nUser Classes:")
        for user_class in user_classes:
            print(f"User_id: {user_class[0]}, Classes_id: {user_class[1]}")

    def show_instructors_with_classes(self, rows):
        print("\nInstructors-Classes:")
        for row in rows:
            print(f"Instructor name: {row[0]}, Instructor surname:
{row[1]}, Class: {row[2]}")

    def show_average_class_duration(self, rows):
        print("\nAverage class duration time for each instructor:")
        for row in rows:
            print(
                f"Instructor ID: {row[0]}, Instructor name: {row[1]},
Instructor surname: {row[2]}, Average duration: {row[3]}")

    def show_classes_with_number_of_users(self, rows):
        print("\nClasses with the largest number of registered users:")
```

```
        for row in rows:
            print(f"Title: {row[0]}, User count: {row[1]}")

    def get_user_input(self):
        while True:
            try:
                user_id = input("Enter user_id: ")
                if user_id.strip():
                    user_id = int(user_id)
                    break
                else:
                    print("User_id cannot be empty.")
            except ValueError:
                print("It must be a number.")
        while True:
            try:
                phone_number = input("Enter phone number: ")
                if phone_number.strip():
                    break
                else:
                    print("Phone number cannot be empty.")
            except ValueError:
                print("It must be a string.")
        while True:
            try:
                email = input("Enter email: ")
                if email.strip():
                    break
                else:
                    print("Email cannot be empty.")
            except ValueError:
                print("It must be a string.")
        while True:
            try:
                first_name = input("Enter first name: ")
                if first_name.strip():
                    break
                else:
                    print("First name cannot be empty.")
            except ValueError:
                print("It must be a string.")
        while True:
            try:
                last_name = input("Enter last name: ")
                if last_name.strip():
                    break
                else:
                    print("Last name cannot be empty.")
            except ValueError:
                print("It must be a string.")
        return user_id, phone_number, email, first_name, last_name

    def get_class_input(self):
        while True:
            try:
                class_id = input("Enter class_id: ")
                if class_id.strip():
                    class_id = int(class_id)
                    break
                else:
                    print("Class_id cannot be empty.")
            except ValueError:
                print("It must be a number.")
        while True:
```

```
            try:
                title = input("Enter title: ")
                if title.strip():
                    break
                else:
                    print("Title cannot be empty.")
            except ValueError:
                print("It must be a string.")
        while True:
            try:
                date_time = input("Enter date and time (YYYY-MM-DD HH:MM):
")
                if date_time.strip():
                    datetime.strptime(date_time, "%Y-%m-%d %H:%M")
                    break
                else:
                    print("Date and time cannot be empty.")
            except ValueError:
                print("Invalid date format. Please use YYYY-MM-DD HH:MM.")
        while True:
            try:
                duration = input("Enter duration: ")
                if duration.strip():
                    duration = int(duration)
                    break
                else:
                    print("Duration cannot be empty.")
            except ValueError:
                print("It must be a number.")
        while True:
            try:
                location = input("Enter location: ")
                if location.strip():
                    break
                else:
                    print("Location cannot be empty.")
            except ValueError:
                print("It must be a string.")
        return class_id, title, date_time, duration, location


    def get_instructor_input(self):
        while True:
            try:
                instructor_id = input("Enter instructor_id: ")
                if instructor_id.strip():
                    instructor_id = int(instructor_id)
                    break
                else:
                    print("Instructor_id cannot be empty.")
            except ValueError:
                print("It must be a number.")
        while True:
            try:
                first_name = input("Enter first name: ")
                if first_name.strip():
                    break
                else:
                    print("First name cannot be empty.")
            except ValueError:
                print("It must be a string.")
        while True:
            try:
                last_name = input("Enter last name: ")
```

```
                if last_name.strip():
                    break
                else:
                    print("Last name cannot be empty.")
            except ValueError:
                print("It must be a string.")
        while True:
            try:
                class_id = input("Enter class_id: ")
                if class_id.strip():
                    class_id = int(class_id)
                    break
                else:
                    print("Class_id cannot be empty.")
            except ValueError:
                print("It must be a number.")
        return instructor_id, first_name, last_name, class_id


    def get_user_class_input(self):
        while True:
            try:
                user_id = input("Enter user_id: ")
                if user_id.strip():
                    user_id = int(user_id)
                    break
                else:
                    print("User_id cannot be empty.")
            except ValueError:
                print("It must be a number.")
        while True:
            try:
                class_id = input("Enter class_id: ")
                if class_id.strip():
                    class_id = int(class_id)
                    break
                else:
                    print("Class_id cannot be empty.")
            except ValueError:
                print("It must be a number.")
        return user_id, class_id

    def get_user_id(self):
        while True:
            try:
                id = int(input("Enter user ID: "))
                break
            except ValueError:
                print("It must be a number.")
        return id


    def get_instructor_id(self):
        while True:
            try:
                id = int(input("Enter instructor ID: "))
                break
            except ValueError:
                print("It must be a number.")
        return id


    def get_class_id(self):
        while True:
```

```
            try:
                id = int(input("Enter class ID: "))
                break
            except ValueError:
                print("It must be a number.")
        return id

    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
```