

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І.І. МЕЧНИКОВА
ФАКУЛЬТЕТ МАТЕМАТИКИ, ФІЗИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

ПОЯСНЮВАЛЬНА ЗАПИСКА
до курсового проекту
з дисципліни «Організація баз даних»
на тему
«Інформаційна система для співробітників
фітнес-клубу»

студентки III курсу

групи _____

спеціальності «Комп'ютерна інженерія»

Воронич Марії Сергіївни

Керівник: _____

Захищено « ____ » _____ 201__ р.

з оцінкою _____

Комісія:

_____ (ПІБ) _____ (Підпис)

_____ (ПІБ) _____ (Підпис)

_____ (ПІБ) _____ (Підпис)

АНОТАЦІЯ

У курсовій роботі розроблюється інформаційна система для співробітників фітнес-клубу.

Мета цієї роботи – розробка інформаційної системи для предметної області фітнес-клуб. Відмінною рисою цієї системи є організація обліку співробітників, трудових договорів, оформлених під час найму співробітника, надання необхідної інформації співробітникам. Система передбачає реалізацію всіх операцій, які можуть знадобитися тренеру фітнес-клубу для оформлення абонементів та створення розкладу спортивних занять. Також, система дає можливість адміністраторам фітнес-клубу вести контроль за тренерами, їх діяльністю, модифікувати данні про фітнес-клуб.

Результатом курсової роботи є інформаційна система із зручним інтерфейсом користувача, який дозволяє інтуїтивно зрозуміло контролювати всі процеси, що відбуваються у системі «фітнес-клуб».

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ.....	4
ВСТУП	5
1 ПОСТАНОВКА ЗАВДАННЯ	7
2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ	8
3 ІНФОРМАЦІЙНЕ МОДЕЛЮВАННЯ.....	11
4 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	19
5 СТВОРЕННЯ БАЗИ ДАНИХ.....	20
6 ЗАПИТИ ДО БД ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ	23
7 РОЗПОДІЛ РОЛЕЙ В БАЗІ ДАНИХ	26
8 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ.....	29
9 ІНСТРУКЦІЯ КОРИСТУВАЧА	31
ВИСНОВОК	38
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	39
ДОДАТОК А. СПИСОК ЗАДАЧ КОРИСТУВАЧІВ	40
ДОДАТОК Б. ЗАПИТИ НА СТВОРЕННЯ БАЗИ ДАНИХ	42
ДОДАТОК В. ЗАПИТИ ДО БАЗИ ДАНИХ.....	45
ДОДАТОК Г. ВИХІДНИЙ КОД ОСНОВНИХ КЛАСІВ.....	53

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

MVP – Model-View-Presenter

UI – User Interface

ПрО – предметна область

ІС – інформаційна система

ВСТУП

Фітнес-клуби є одними із найбільш попитних організацій в умовах міста. Вони забезпечують гарну фізичну підготовку для їхніх клієнтів, покращення загального самопочуття, коригування маси тіла, якісної підготовки до спортивних змагань та реабілітації після важких травм або інших недоліків здоров'я. Також, при оптимальній організації процесу найму співробітників, обліку абонементів та створення розкладу спортивних занять, фітнес-клуби можуть приносити прибуток. Саме тому особливо актуальним є створення такого інструменту, який допоміг би мінімізувати витрати часу та робочих ресурсів, полегшити роботу таких організацій.

Для того, щоб створити оптимальні умови роботи фітнес-клубу, доречно використати інформаційну систему для організації фітнес-клубу, яка дозволить адміністраторам подібних організацій здійснювати контроль за своїми співробітниками та клієнтами у зручній формі та додатково реалізує для співробітників можливість вести облік необхідних даних у інтуїтивно зрозумілому вигляді.

Для зручної реалізації і впровадження подібної інформаційної системи найбільше підходить візуалізований інтерфейс користувача для контролю даних про розклад спортивних занять, тренерів фітнес-клубу, оформлених абонементів, який дозволяє підтримувати порядок в організаційній і фінансовій частині фітнес-клубу.

Мета цієї роботи – розробка інформаційної системи для предметної області фітнес-клуб.

Для досягнення поставленої мети сформовані такі завдання:

- 1) виконати аналіз предметної області «Інформаційна система для співробітників фітнес-клубу»;
- 2) визначити вимоги до інформаційної системи;
- 3) розробити архітектуру системи;
- 4) вибрати засоби реалізації;

5) розробити бібліотеку класів, що реалізують доступ до даних стосовно розкладу занять, тренерів, клієнтів і абонементів - деякий "чорний ящик", що надає розробникам всі необхідні засоби для роботи з даними і приховує реалізацію бази і обміну даними;

6) реалізувати інтерфейс, який дасть можливість працівникам зручного оформлення співробітників на роботу, надання належних послуг клієнтам;

7) реалізувати клієнтську програму, яка, крім перегляду, надасть можливості редагування даних у фітнес-клубі - можливі абонементи, послуги; а також додавання і видалення особистої інформації та облікових записів співробітників фітнес-клубу;

8) забезпечити розмежування доступу з боку різних категорій користувачів і захист від несанкціонованого доступу.

1 ПОСТАНОВКА ЗАВДАННЯ

Система надає кожному типу користувачів свій набір функцій, що дозволяє проводити маніпуляції з даними, доступними тренеру фітнес-клубу та його адміністратору, переглядати записи, редагувати, створювати і видаляти їх відповідно до прав кожного користувача:

1) тренер фітнес-клубу – має права на перегляд персональної інформації клієнтів; має право реєструвати нового клієнта, оформлювати для нього абонемент, створювати новий розклад занять для клієнтів;

2) адміністратор фітнес-клубу – може оформлювати нового співробітника на роботу, а також звільнити співробітника; має можливість додати нову послугу, тип абонементу та приміщення.

Таким чином, кожен користувач даної інформаційної системи має свою роль і програма надає можливість реалізації своєї функціональності відповідно до призначеної користувачеві роллю.

Список задач користувачів можна переглянути в Додатку А.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

При розподілі інформаційної системи на ланки, була обрана дворівнева архітектура.

Дворівнева архітектура дозволяє ізолювати рівні один від одного в достатній мірі, щоб підвищити надійність і модульованість системи. Рівні цієї архітектури включають в себе:

1) клієнт (шар клієнта) — це інтерфейсний (зазвичай графічний) компонент комплексу, надається кінцевому користувачу;

2) сервер баз даних (шар даних) забезпечує зберігання даних і виноситься на окремий рівень, реалізується, як правило, засобами систем управління базами даних.

Також, така архітектура дозволяє підвищити незалежність системи від помилок сервера баз даних. Інформаційна система для співробітників фітнес-клубу, як правило, підключається по локальних мережах, в межах одного фітнес-клубу. При проблемах з одним терміналом в системі, забезпеченим «товстим» клієнтом, інші термінали функціонуватимуть. Кожна робоча станція з програмою буде автономною, незалежною від інших робочих станцій. Завдяки такому підходу будь-які проблеми можуть виникати тільки в тому випадку, якщо стався збій в роботі бази даних. На відміну від трирівневої архітектури, дворівнева дозволяє уникнути таких неприємностей, як можливі атаки сервера зловмисниками або зайве навантаження на сервер при обслуговуванні системи.

Грунтуючись на обраній архітектурі, необхідно підібрати шаблон проектування інтерфейсу користувача. Для дворівневої архітектури найбільше підходить шаблон MVP, який і застосовується для поділу інформаційної системи на шари.

MVP — шаблон проектування інтерфейсу користувача, розроблений для полегшення автоматичного модульного тестування і поліпшення

розподілу відповідальності в презентаційній логіці (відділення бізнес-логіки від відображення):

- Модель (англ. Model) — зберігає в собі всю бізнес-логіку, при необхідності отримує дані зі сховища.
- Вид (англ. View) — реалізує відображення даних (з Model), звертається до Presenter за оновленнями.
- Ведучий (англ. Presenter) — реалізує взаємодію між Model і View.

Слабкозв'язані системи у більшості випадків простіше і надійніше будувати за допомогою MVP, так як цей шаблон набагато краще підходить для швидкого і не витратного переходу від одного інтерфейсу до іншого з мінімальними змінами в Model і без змін в Presenter. MVP також використовується при умовах, коли програма має бути побудовано на основі компонентного фреймворка (докладніше в розділі «Вибір програмного забезпечення»). При цьому немає необхідності постійно створювати Presenter наново (рис. 2.1).

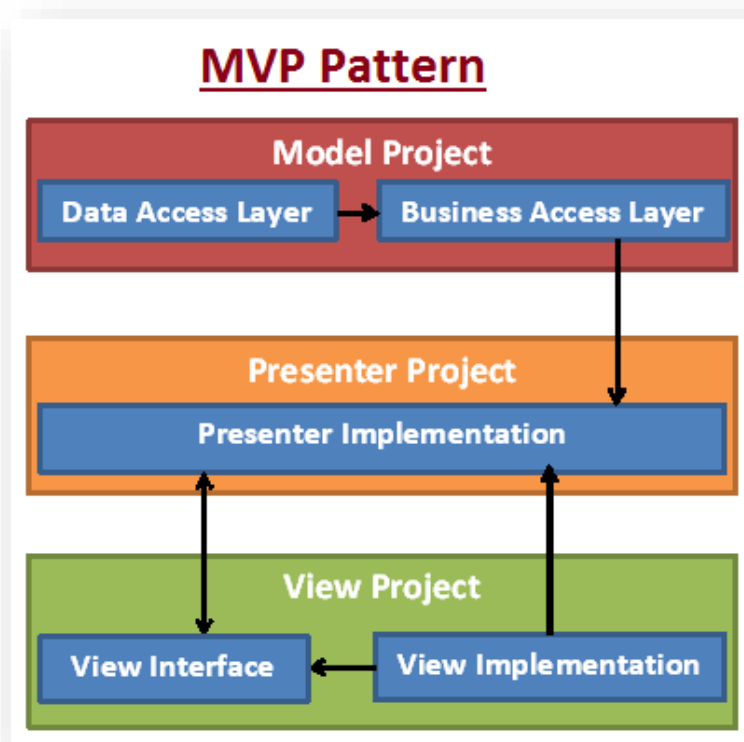


Рисунок 2.1 – Діаграма складових частин шаблону MVP

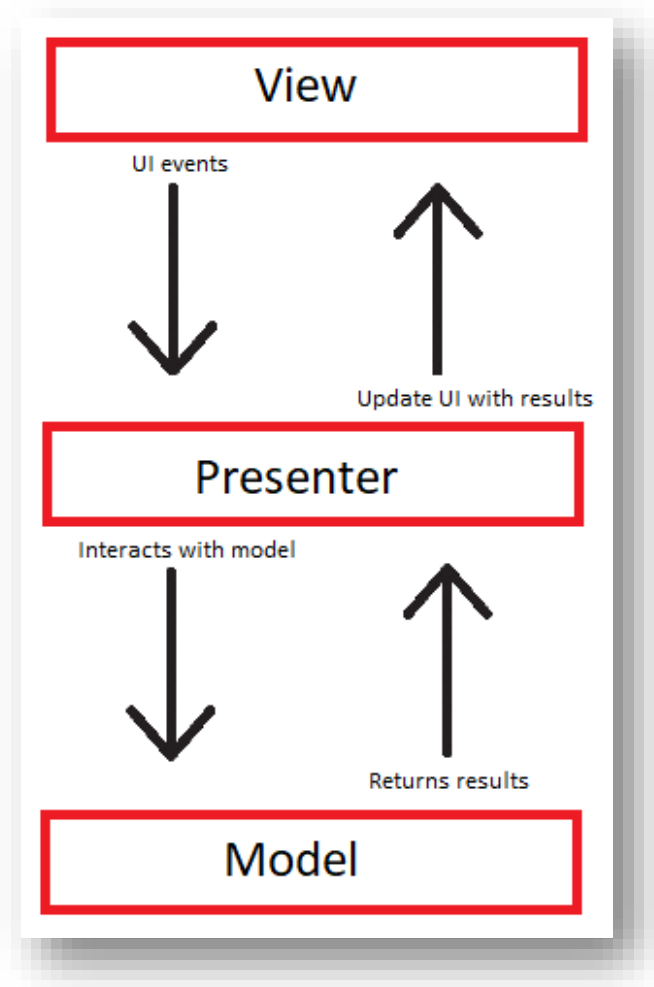


Рисунок 2.2 – Діаграма взаємодії складових частин шаблону MVP

Мета MVP - зробити Views використовуваними наново. Для цього кожен View реалізує певний інтерфейс і реалізує механізм подій для зворотного зв'язку з Presenter'ом (рис. 2.2).

MVC і MVP-парадигми дуже схожі один на одного, але їх застосування залежить від умов використання. Для MVC - це там, де Presenter оновлюється кожного разу по якій-небудь події, а для MVP, коли Presenter не потрібно створювати щоразу.

Так як в розроблювальній програмі Presenter не повинен створюватися кожного разу при будь-якому впливі користувача на UI, перевага була віддана паттерну MVP. Створивши Presenter один раз, система зберігає його і має можливість використовувати його в подальшій роботі.

3 ІНФОРМАЦІЙНЕ МОДЕЛЮВАННЯ

Перший етап проектування полягає у створенні схеми БД, яка включає визначення сутностей і існуючих між ними зв'язків і атрибутів. Результатом даного проектування є схема БД, представлена у вигляді ER-діаграми (Entity-Relationship), на якій відображаються основні сутності предметної області, їх атрибути та зв'язки. Схема БД створюється на основі функціональних вимог користувачів і абсолютно не залежить від будь-яких особливостей фізичної реалізації інформаційної системи, таких як тип обраної СУБД або мову програмування.

Послідовність етапів створення схеми бази даних:

- 1) визначення сутностей;
- 2) визначення атрибутів сутностей;
- 3) визначення первинних і альтернативних ключів.

У цій роботі поставлена задача побудови ER-діаграми для фітнес-клубу. Виходячи з аналізу ПрО, для функціонування інформаційної системи необхідні наступні сутності, які мають первинний ключ і атрибути:

- 1) Співробітник – це персонал фітнес-клубу, який буде використовувати ресурс для контролю та перегляду даних, створення розкладу спортивних занять, оформлення абонементів;
- 2) Клієнт – це користувач фітнес-клубу, інформація якого буде використана даною інформаційною системою для створення розкладів спортивних занять та оформлення абонементів;
- 3) Посада – це трудова роль, яку виконує співробітник;
- 4) Послуга – це сервіс, надаваний тренером фітнес-клубу;
- 5) Абонемент – це документ, що надає доступ клієнту до послуг фітнес-клубу на певний період за певну суму.

У даній ПрО можна виділити наступні зв'язки між сутностями:

- 1) так як співробітник може працювати на декількох посадах, а на одній посаді може працювати багато співробітників, то між ними встановлюється зв'язок «багато-до-багатьох»;
- 2) так як співробітник може надавати декілька послуг, то між ними встановлюється зв'язок «один-до-багатьох»;
- 3) так як клієнт може скористатися декількома послугами, а послуга може бути використана багатьма клієнтами, то між ними встановлюється зв'язок «багато-до багатьох»;
- 4) так як клієнт може мати декілька видів абонементів, а один вид абонементу може належати до багатьох клієнтів, то між ними встановлюється зв'язок «багато-до-багатьох»;
- 5) так як в одному приміщенні можуть займатися спортом декілька клієнтів, а один клієнт може займатися в декількох приміщеннях, то між ними встановлюється зв'язок «багато-до-багатьох».

Наступним кроком ми повинні формалізувати зв'язки між сутностями. Так як між співробітником та посадою встановився n -арний зв'язок, а в реляційних моделях цей зв'язок не можна представити у вигляді двох таблиць, то варто провести формалізацію. Для формалізації n -арного зв'язку необхідно $n+1$ відношення: по одному для кожної сутності, що беруть участь у зв'язку, і одне для самого зв'язку. Причому відношення для зв'язку, точніше, його схема повинна включати первинні ключі всіх інших n відношень [1]. Тому була створена таблиця «Договір», яка містить первинні ключі сутностей Співробітник та Посада. Аналогічна операція відбувається із зв'язком між сутностями Клієнт та Послуга. Як наслідок, отримуємо нову сутність Розклад.

Після формалізації була отримана наступна схема, представлена на рис 3.1.

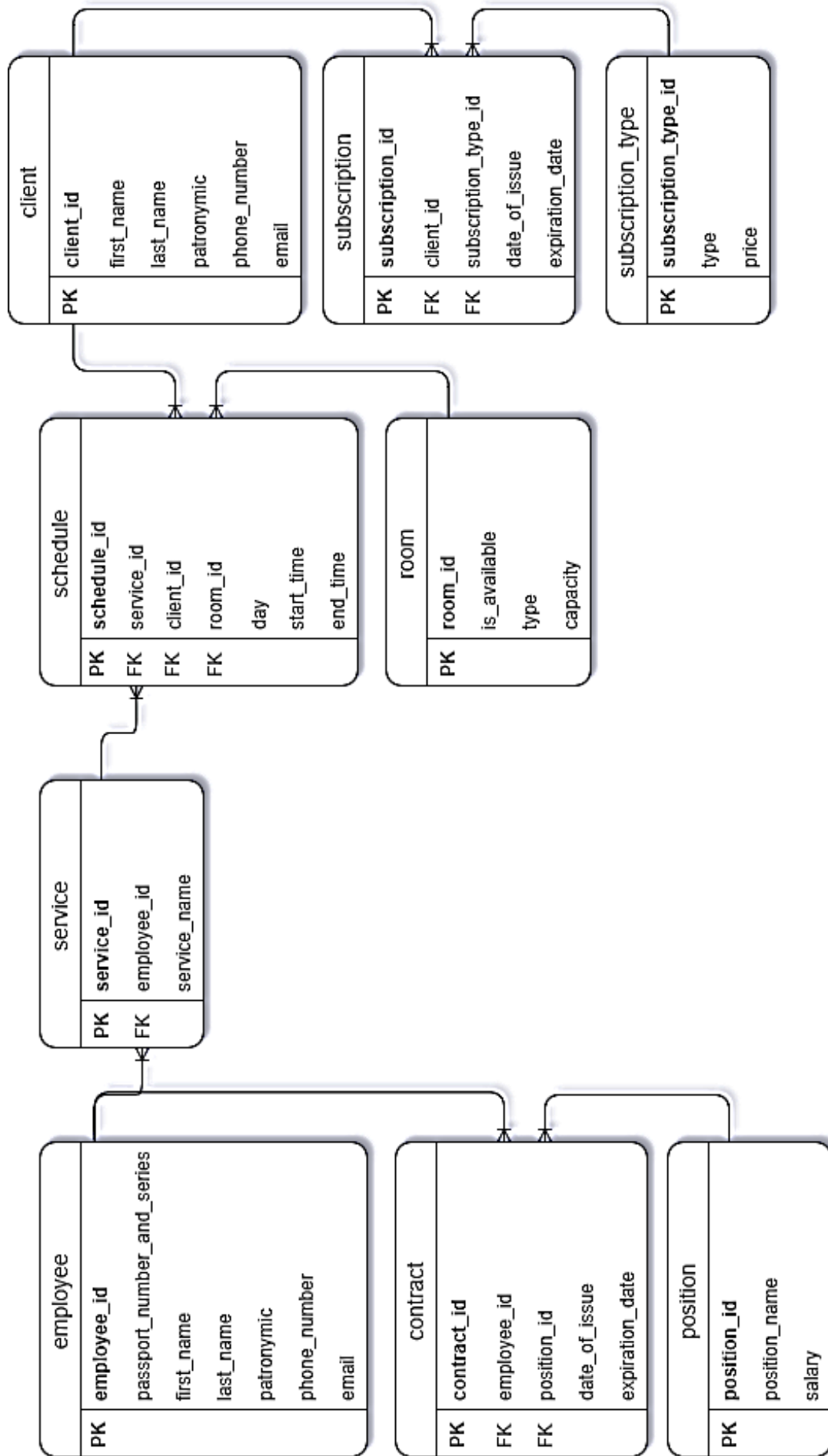


Рисунок 3.1 – ER-діаграма ІС

Виходячи з аналізу предметної області, для функціонування інформаційної системи необхідні наступні сутності та їх властивості:

Таблиця 3.1 – Опис сутностей

Ім'я атрибута	Тип	Призначення	Унікальність	Обмеження
Співробітник (employee)				
employee_id	цілочисельний	ідентифікатор співробітника	+	не пусте, первинний ключ
passport_number_and_series	текст	номер паспорту та серійний номер	+	не пусте
first_name	текст	ім'я		не пусте
last_name	текст	прізвище		не пусте
patronymic	текст	по-батькові		не пусте
phone_number	текст	номер телефону	+	не пусте
email	текст	електронна пошта	+	не пусте
Договір (contract)				
contract_id	цілочисельний	номер договору	+	не пусте, первинний ключ
employee_id	цілочисельний	номер співробітника		не пусте, зовнішній ключ на поле employee_id у таблиці employee

Продовження таблиці 3.1

Ім'я атрибута	Тип	Призначення	Унікальність	Обмеження
position_id	цілочисельний	номер посади		не пусте, зовнішній ключ на поле position_id у таблиці position
date_of_issue	дата	дата оформлення		не пусте
expiration_date	дата	дата кінця терміну придатності		не пусте, більше за дату оформлення
Посада (position)				
position_id	цілочисельний	номер посади	+	не пусте, первинний ключ
position_name	текст	назва посади	+	не пусте
salary	цілочисельний	розмір заробітної плати		не пусте, більше за нуль
Послуга (service)				
service_id	цілочисельний	номер послуги	+	не пусте, первинний ключ
employee_id	цілочисельний	номер співробітника		не пусте, зовнішній ключ на поле employee_id у таблиці employee
service_name	текст	назва послуги		не пусте

Продовження таблиці 3.1

Ім'я атрибута	Тип	Призначення	Унікальність	Обмеження
Розклад занять (schedule)				
schedule_id	цілочисельний	номер розкладу	+	не пусте, первинний ключ
service_id	цілочисельний	номер послуги		не пусте, зовнішній ключ на поле service_id у таблиці service
client_id	цілочисельний	номер клієнта		не пусте, зовнішній ключ на поле client_id у таблиці client
room_id	цілочисельний	номер приміщення		не пусте, зовнішній ключ на поле room_id у таблиці room
day	текст	день тижня, коли проводиться заняття		не пусте
start_time	час	час початку заняття		не пусте
end_time	час	час кінця заняття		не пусте, час кінця заняття більше ніж час початку заняття
Приміщення (room)				
room_id	цілочисельний	номер приміщення	+	не пусте, первинний ключ

Продовження таблиці 3.1

Ім'я атрибута	Тип	Призначення	Унікальність	Обмеження
is_available	логічний	чи є приміщення доступним		не пусте, true за умовчуванням
type	текст	тип приміщення	+	не пусте
capacity	цілочисельний	місткість приміщення		не пусте
Клієнт (client)				
client_id	цілочисельний	номер клієнта	+	не пусте, первинний ключ
first_name	текст	ім'я		не пусте
last_name	текст	прізвище		не пусте
patronymic	текст	по-батькові		не пусте
phone_number	текст	номер телефону	+	не пусте
email	текст	електронна пошта	+	не пусте
Абонемент (subscription)				
subscription_id	цілочисельний	номер абонементу	+	не пусте, первинний ключ
client_id	цілочисельний	номер клієнта		не пусте, зовнішній ключ на поле client_id у таблиці client
subscription_type_id	цілочисельний	номер типу абонемента		не пусте, зовнішній ключ на поле subscription_type_id у таблиці subscription_type

Продовження таблиці 3.1

Ім'я атрибута	Тип	Призначення	Унікальність	Обмеження
date_of_issue	дата	дата оформлення		не пусте
expiration_date	дата	дата кінця терміну придатності		не пусте, більше за дату оформлення
Тип абонементу (subscription_type)				
subscription_type_id	цілочисельний	номер типу абонемента	+	не пусте, первинний ключ
type	текст	тип абонемента	+	не пусте
price	цілочисельний	ціна		не пусте, більше за нуль

4 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В якості мови реалізації програми були обрані наступні інструменти:

мова програмування - C #. Завдяки високій інтеграції ООП, модульності і гнучкості, C # є одним з найбільш зручних мов для розробки десктопних додатків;

середовище розробки - Visual Studio 2019 Community Edition;

модулі інтерфейсу: Windows Forms - зручність в простоті створення дизайну і підключення функціональності.

В якості СУБД був обраний PostgreSQL. Такий вибір пов'язаний з рядом переваг PostgreSQL перед іншими СУБД:

PostgreSQL не просто реляційна, а об'єктно-реляційна СУБД. Це дає їй вагомим переваги над іншими SQL базами даних з відкритим вихідним кодом;

PostgreSQL надає надійні вбудовані оператори та функції. Крім того, він дозволяє створювати власні оператори і функції (включаючи агрегати), а також процедури, що і тригери;

для PostgreSQL добре опрацьованості документація, що дозволяє вирішувати виникаючі проблеми з деякими зручністю;

PostgreSQL - безкоштовна СУБД з відкритим кодом;

існує безліч API під різні платформи.

Під час розробки використовувалися також наступні засоби програмування:

Npgsql - бібліотека класів, необхідна для роботи з СУБД PostgreSQL.

5 СТВОРЕННЯ БАЗИ ДАНИХ

Детально розглянемо запит на створення однієї з таблиць БД – contract.

Даний запит відповідає за створення таблиці і її атрибутів.

```
create table "contract"
(
    "contract_id" serial,
    "employee_id" int not null,
    "position_id" int not null,
    "date_of_issue" date not null,
    "expiration_date" date not null check("expiration_date" >
"date_of_issue"),

    primary key("contract_id"),
    foreign key("employee_id") references "employee" on update
cascade on delete cascade,
    foreign key("position_id") references "position" on update
cascade
);
```

Створення таблиці відбувається за допомогою команди CREATE TABLE. Поле contract_id є первинним ключем; поле employee_id є зовнішнім ключем на таблицю employee; поле position_id є зовнішнім ключем на таблицю position. Поле CHECK є полем для перевірки, яку вказують в круглих дужках. Запис ON UPDATE CASCADE означає, що при оновленні первинного ключа цей зовнішній ключ також придбає нове значення, рівне новому значенню первинного ключа інших таблиць. ON DELETE CASCADE - при видаленні запису з таблиці employee всі кортежі, які посилалися на видалений запис, також каскадно видаляться. Всі інші приклади створення таблиць наведені в Додатку Б.

Розглянемо створення однієї зі збережених процедур:

```
create or replace function subscribe_client
(
    _client_id int,
    _subscription_type varchar,
    _date_of_issue date,
    _expiration_date date
)
returns void
```

```

as $$
begin
    insert into
    "subscription" ("client_id", "subscription_type_id", "date_of_issue",
    "expiration_date")
        values (_client_id, (select "subscription_type_id"
                                from "subscription_type"
                                where
                                "type"=_subscription_type), _date_of_issue, _expiration_date);
end $$
language plpgsql;

```

Створення процедури відбувається за допомогою CREATE OR REPLACE FUNCTION. Як параметри передаються необхідні дані для оформлення абонементу. Процедура виробляє команду INSERT, створюючи записи в таблиці subscription.

Розглянемо створення одного з представлень:

```

create or replace view "employee_info"
as
select
    e."employee_id",
    concat(e."first_name", ' ', e."patronymic", ' ',
e."last_name") as "name",
    e."phone_number",
    e."passport_number_and_series",
    e."email",
    p."position_name",
    s."service_name",
    c."date_of_issue",
    c."expiration_date"
from
    "employee" e
join
    "contract" c using("employee_id")
left join
    "service" s using("employee_id")
join
    "position" p using("position_id");

```

Створення представлення відбувається за допомогою CREATE OR REPLACE VIEW. Представлення виводить необхідні поля з чотирьох різних таблиць: з employee – номер співробітника, ПІБ, номер телефону, номер паспорта та серію, електронну пошту; з position – назву посади; з service –

назву послуги, що надає співробітник, з contract – дату оформлення трудового договору та дату завершення терміну дії трудового договору.

Розглянемо створення одного з тригерів та тригерних процедур:

```
create or replace function check_time() returns trigger as $$
begin
    if exists
        (select *
         from "schedule"
         where "service_id" = new.service_id -- Selection of the
same service on the same day.
         and "day" = new.day
         and ("start_time" <= new.end_time -- If time periods
overlap
         and "end_time" >= new.start_time))
    then
        raise exception 'Overlaps with existing time.';
    else
        return new;
    end if;
end $$
language plpgsql;

create trigger check_time_trigger before insert on "schedule"
for each row execute procedure check_time();
```

Створення тригера відбувається за допомогою команди CREATE TRIGGER. Спрацьовування тригера призначено на момент перед введенням даних (BEFORE INSERT) у таблицю schedule (ON "schedule"). Тригер спрацює для кожного рядка введених даних FOR EACH ROW, викликавши функцію, що перевірює, чи не збігається час проведення заняття із іншими розкладами занять. Всі інші приклади наведені в Додатку В.

6 ЗАПИТИ ДО БД ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

Розглянемо кілька типів запитів, які можуть з'являтися в базі даних:

Вибірка даних

Розберемо приклад вибірки даних за допомогою уявлення `schedule_info`:

У додатку з `ScheduleModel` за допомогою `NpgsqlCommand` передається запит до бази даних:

```
select * from schedule_info order by schedule_id;
```

У базі даних, в свою чергу, викликається представлення `schedule_info`, що відповідає за відображення даних з таблиць бази даних:

```
create view "schedule_info"
as
select
    sc.schedule_id,
    sr.service_name,
    sr.employee_id,
    sc.client_id,
    r.type as "room_type",
    sc.day,
    sc.start_time,
    sc.end_time
from
    "schedule" sc
join
    "client" c using("client_id")
join
    "service" sr using("service_id")
join
    "room" r using("room_id");
```

Введення даних

Розглянемо введення даних за допомогою збереженої процедури `create_employee`

У додатку з `EmployeeModel` за допомогою `NpgsqlCommand` передається запит до бази даних з урахуванням параметрів, лічених з `TextBox` та `ComboBox`:

```
create or replace function create_employee
(
```

Передані параметри:

```
  _passport_number_and_series varchar,
  _first_name varchar,
  _last_name varchar,
  _patronymic varchar,
  _phone_number varchar,
  _email varchar,
  _position_name varchar,
  _date_of_issue date,
  _expiration_date date
```

```
)
```

Процедура не повертає значення:

```
returns void
as $$
```

Декларативний блок для оголошення змінної new_employee_id:

```
declare
  new_employee_id int;
```

Початок тіла процедури:

```
begin
```

Запит на введення нового кортежу у таблицю employee:

```
  insert into "employee" ("passport_number_and_series",
    "first_name", "last_name", "patronymic", "phone_number",
    "email")
    values (_passport_number_and_series, _first_name,
    _last_name, _patronymic, _phone_number, _email);
```

Ініціалізація змінної new_employee_id:

```
  new_employee_id = (select max("employee_id") from
    "employee");
```

Запит на введення нового кортежу у таблицю contract:

```
  insert into "contract" ("employee_id", "position_id",
    "date_of_issue", "expiration_date")
    values (new_employee_id,
      (select "position_id" from "position" where
        "position_name" = _position_name),
      _date_of_issue, _expiration_date);
```

Кінець тіла процедури:

```
end $$
language plpgsql;
```


Видалення даних

Розглянемо видалення даних за допомогою запиту

У додатку з ScheduleModel за допомогою NpgsqlCommand передається запит до бази даних з урахуванням параметрів, лічених з TextBox:

```
delete from "schedule"
```

Розклад занять видається за номером, який передається з контейнера

```
where "schedule_id" = _schedule_id;
```

Модифікація даних

Розглянемо введення даних за допомогою запиту

У додатку з ContractModel за допомогою NpgsqlCommand передається запит до бази даних з урахуванням параметрів, лічених з TextBox:

```
update "contract"
```

Встановлюється поточна дата завершення терміну дії трудового договору за встановленим номером співробітника, наданим с контейнера:

```
set "expiration_date" = current_date  
where "employee_id" = _employee_id;
```

Всі інші приклади наведені в Додатку В.

7 РОЗПОДІЛ РОЛЕЙ В БАЗІ ДАНИХ

Як вже було згадано вище, в даній інформаційній системі присутні дві ролі. Можна їх розглянути більш докладно з точки зору прав на таблиці бази даних:

Адміністратор – має можливість виконувати всі CRUD операції над сутностями Співробітник, Трудовий Договір та Посада; має право на запис, редагування та видалення із сутностей Приміщення, Послуга та Тип Абонементу;

Тренер – має можливість виконувати всі CRUD операції над сутностями Розклад, Клієнт, Абонемент; має право на читання сутностей Приміщення, Тип Абонементу та Послуга.

Розглянемо приклад розподілу ролей та прав доступу у базі даних:

Створюємо роль `trainers`, що відповідає тренерам:

```
create role trainers;
```

Надаємо тренерам право на під'єднання до бази даних:

```
grant connect on database fitness_club to trainers;
```

Надаємо тренерам права доступу до схеми `public`:

```
grant usage on schema public to trainers;
```

Надаємо тренерам всі привілеї на всі таблиці у схемі:

```
grant all privileges on all tables in schema public to  
trainers;
```

Знімемо з тренерів привілеї на запис, видалення та модифікацію таблиць `room`, `subscription_type` та `service`:

```
revoke insert, delete, update on "room",  
"subscription_type", "service" from trainers;
```

Розглянемо запит на створення користувача `masha_trainer` з паролем `trainer` у ролі тренера:

```
create user masha_trainer with encrypted password 'trainer'  
in role trainers;
```

Всі інші запити на створення ролей та розподіл прав можна подивитися у Додатку В.

Нижче, в таблиці 7.1, наведена інформація про права кожної ролі на кожній таблиці, де рядки - таблиця в БД, а в осередках – наявні операції CRUD.

Таблиця 7.1 - Права ролей на таблиці БД

Таблиця/Роль	Адміністратор	Тренер
employee	CRUD	-
contract	CRUD	-
position	CRUD	-
service	CUD	R
schedule	-	CRUD
room	CUD	R
client	-	CRUD
subscription	-	CRUD
subscription_type	CUD	R

Розглянемо процес аутентифікації користувача:

```
private void btnSignIn_Click_1(object sender, EventArgs e)
{
    ...
    Login(TextBoxLogin.Text, TextBoxPassword.Text);
    ...
}
public void Login(string login, string password)
{
    factory = new Factory("127.0.0.1", "5433", login, password,
        "fitness_club");
}
public Factory(string server, string port, string user, string
    pass, string dbname)
{
    string ConnectionString = $"Server={server}; Port={port};
    User Id={user}; Password={pass}; Database={dbname}";
    npgsqlConnection = new NpgsqlConnection(ConnectionString);
    connection = new Connection(npgsqlConnection);
    OpenConnection();
    ...
}
public NpgsqlConnection npgsqlConnection { get; }
```

```

public Connection(NpgsqlConnection connection)
{
    npgsqlConnection = connection;
}
public void OpenConnection()
{
    ...
    npgsqlConnection.Open();
    ...
}

```

Після введення користувачем логіну та пароллю у TextBoxLogin та TextBoxPassword відповідно та натиску кнопки sign in, викликається функція btnSignIn_Click_1, у якій значення цих елементів управління передаються як параметри до функції Login. Функція Login ініціалізує об'єкт factory за допомогою конструктору класу Factory, де у якості параметрів передається IP-адреса серверу, номер порту, логін та пароль користувача, що були передані функції Login, та назва бази даних. Конструктор Factory у свою чергу створює строку з'єднання ConnectionString, де {server} - IP-адреса серверу, {user} – логін користувача, {pass} - пароль користувача, {dbname} – назва БД. ConnectionString застосовується у якості параметру для ініціалізації об'єкту npgsqlConnection за допомогою конструктора класу NpgsqlConnection. Згодом об'єкт connection ініціалізується конструктором класу Connection, де у якості параметру передається об'єкт npgsqlConnection. У тілі конструктора Connection ініціалізується об'єкт npgsqlConnection значенням параметру connection. Потім у конструкторі Factory викликається функція OpenConnection, що від об'єкту npgsqlConnection викликає функцію Open().

8 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Для реалізації інформаційної системи був написаний ряд класів, які вирішують різні завдання в залежності від шару, в якому вони знаходяться. Розглянемо основні класи на кожному з шарів:

1) Tables

Являє собою простір імен для таблиць-контейнерів бази даних. До Tables має доступ тільки шар Model. Таблиці, які знаходяться в Tables:

- а) employee;
- б) contract;
- в) position;
- г) service;
- д) schedule;
- е) room;
- ж) client;
- з) subscription;
- и) subscription_type.

2) Model

Являє собою групу таблиць-моделей для кожної з таблиць в Tables. У моделях формуються всі SQL-запити і відбувається маніпуляція з даними. Шар служить для інкапсуляції таблиць-сховищ.

Таблиці, які знаходяться в Repos:

- а) EmployeeModel;
- б) ContractModel;
- в) PositionModel;
- г) ServiceModel;
- д) ScheduleModel;
- е) RoomModel;
- ж) ClientModel;
- з) SubscriptionModel;

и) SubscriptionTypeModel.

3) Керівні класи:

а) Presenter - клас Presenter, відповідає за передавання даних і команд від View до Model і навпаки;

б) Factory - реалізація фабричного методу для зручної і масштабованої реалізації Presenter, огортає всі класи з Model;

в) SqlConnection - реалізація з'єднання додатку з базою даних за допомогою Npgsql.

Розглянемо докладніше частини MVP за допомогою діаграми класів (рис. 8.1):

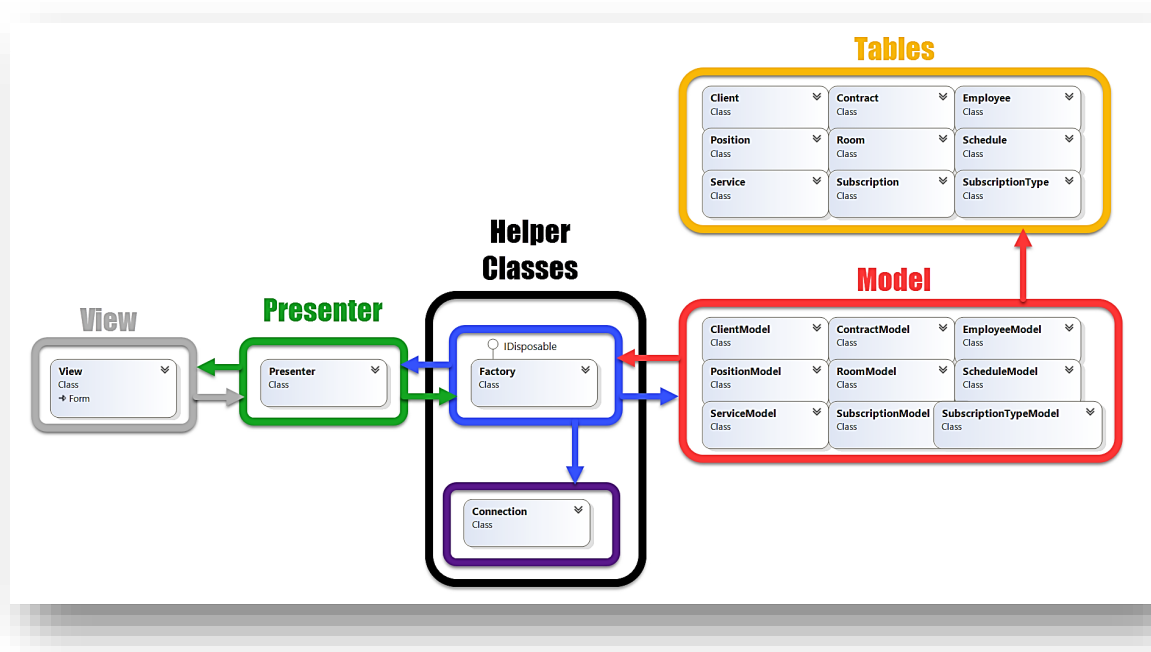


Рисунок 8.1 – Діаграма роботи патерна MVP

Програмний код класів наведено у Додатку Г.

9 ІНСТРУКЦІЯ КОРИСТУВАЧА

Під час запуску програми користувач бачить просте вікно, що містить два поля для введення - логін і пароль - а також кнопку підтвердження (рис. 9.1).

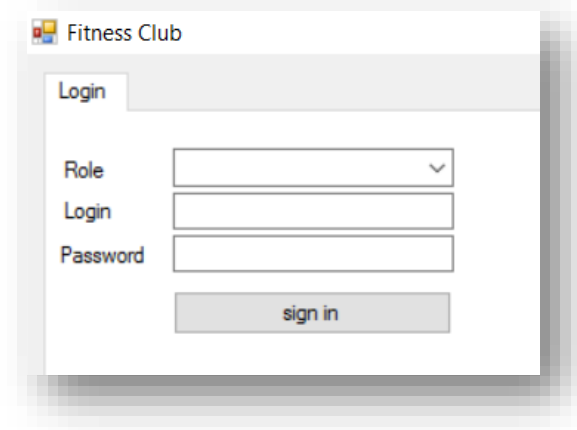


Рисунок 9.1 - Сторінка авторизації

Так як в додатку передбачено два види користувачів, кожен з користувачів буде володіти своїм набором функціоналу. Розглядається форма від імені адміністратора і від імені тренера (рис. 9.2-9.4).

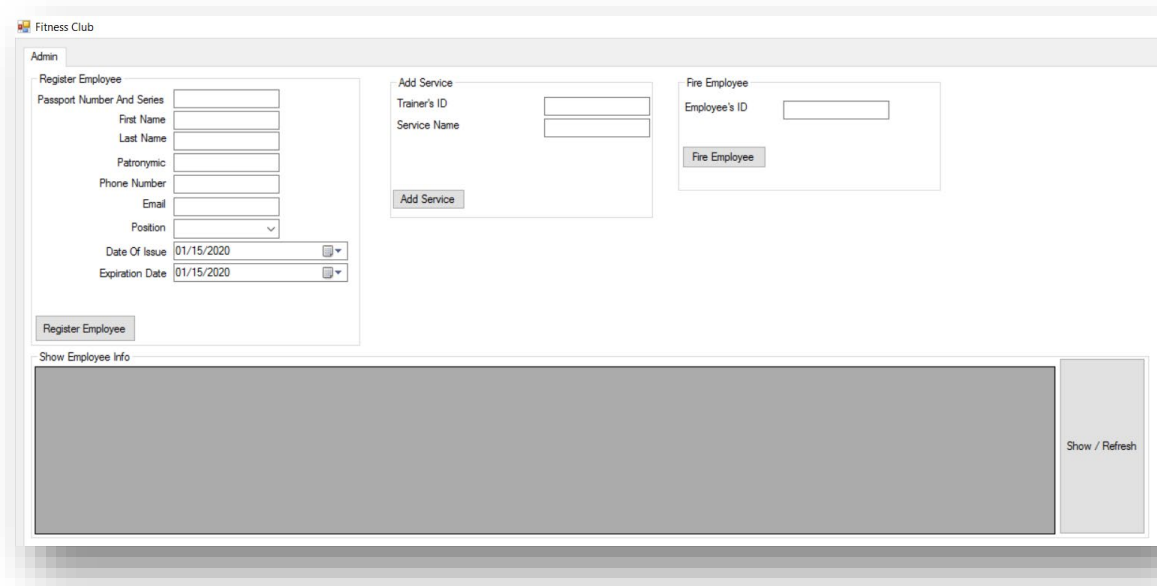


Рисунок 9.2 – Адміністратор

Fitness Club

Trainer **Client**

Create Schedule

Client's ID

Service ID

Room

Day

Start Time

End Time

Create Schedule

Show Service

Show Service

Delete Schedule

Schedule's ID

Delete

Show Schedule Info

Show / Refresh

Рисунок 9.3 – Тренер, вкладка «Trainer»

Fitness Club

Trainer **Client**

Register Client

First Name

Last Name

Patronymic

Phone Number

Email

Register Client

Subscribe Client

Client's ID

Subscription Type

Date Of Issue

Expiration Date

Subscribe Client

Renew Subscription

Subscription's ID

Expiration Date

Renew

Show Subscription Type Info

Show / Refresh

Show Client Info

Show / Refresh

Рисунок 9.4 – Тренер, вкладка «Client»

Розглянемо функціонал від імені адміністратора.

Подивимось результати відображення інформацій про співробітників (рис. 9.5):

Show Employee Info

Employee's ID	Name	Phone Number	Passport	Email	Position	Service	Date Of Issue	Expiration Date
1	Vasya Batkovich Poopkin	3333333333	FF333333	poopkin@email.c...	trainer	fitness	1/18/2020 12:00...	11/25/2020 12:00...
2	W W W	111	WW111111	W@	admin	swimming	1/15/2020 12:00...	1/19/2020 12:00...
*								

Show / Refresh

Рисунок 9.5 – Виводиться інформація про співробітників

Оформимо нового співробітника на роботу (рис. 9.6):

Register Employee

Passport Number And Series GR554432

First Name Sidor

Last Name Sidorov

Patronymic Sidorovych

Phone Number 5323242

Email sidorov@email.com

Position admin

Date Of Issue 01/15/2020

Expiration Date 07/10/2599

Register Employee

Рисунок 9.6 – Додання нового співробітника до бази

Переконаємося в тому, що інформація стосовно нового співробітника є в базі (рис. 9.7):

Show Employee Info

Employee's ID	Name	Phone Number	Passport	Email	Position	Service	Date Of Issue	Expiration Date
1	Vasya Batkovich Poopkin	3333333333	FF333333	poopkin@email.c...	trainer	fitness	1/18/2020 12:00...	11/25/2020 12:00...
2	W W W	111	WW111111	W@	admin	swimming	1/15/2020 12:00...	1/19/2020 12:00...
3	Sidor Sidorovych Sidorov	5323242	GR554432	sidorov@email.com	admin		1/15/2020 12:00...	7/10/2599 12:00...
*								

Show / Refresh

Рисунок 9.7 – Додання нового співробітника до бази

Додамо нову послугу (рис. 9.8):

Add Service

Trainer's ID 1

Service Name box

Add Service

Рисунок 9.8 – Додання нової послуги до бази

Звільнимо співробітника (рис. 9.9):

Fire Employee

Employee's ID 3

Fire Employee

Рисунок 9.9 – Звільнення співробітника

Розглянемо функціонал від імені тренера.

Перейдемо до вкладки «Client» та переглянемо інформацію про клієнта (рис. 10.1):

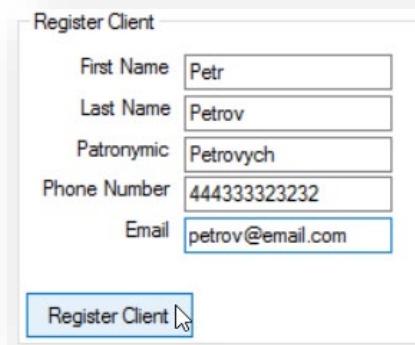
Show Client Info

	Client's ID	Name	Phone Number	Email	Sub ID	Type	Price	Date Of Issue	Expiration Date
►	1	Ivan Vasilevich Boonsha	222222222	boonsha@email...	1	lux	800	1/15/2020 12:00...	12/31/2398 12:0...
*									

Show / Refresh

Рисунок 10.1 – Виведення інформації про клієнтів

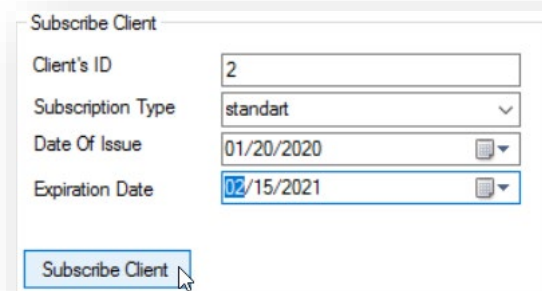
Зареєструємо нового клієнта (рис. 10.2):



A screenshot of a 'Register Client' form. It contains five input fields: 'First Name' with 'Petr', 'Last Name' with 'Petrov', 'Patronymic' with 'Petrovych', 'Phone Number' with '444333323232', and 'Email' with 'petrov@email.com'. A 'Register Client' button is at the bottom, with a mouse cursor hovering over it.

Рисунок 10.2 – Реєстрація нового клієнта

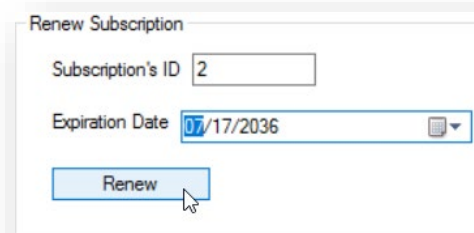
Оформлюємо абонемент для клієнта (рис. 10.3):



A screenshot of a 'Subscribe Client' form. It contains four input fields: 'Client's ID' with '2', 'Subscription Type' with 'standart' (dropdown), 'Date Of Issue' with '01/20/2020' (calendar icon), and 'Expiration Date' with '02/15/2021' (calendar icon). A 'Subscribe Client' button is at the bottom, with a mouse cursor hovering over it.

Рисунок 10.3 – Оформлення абонементу

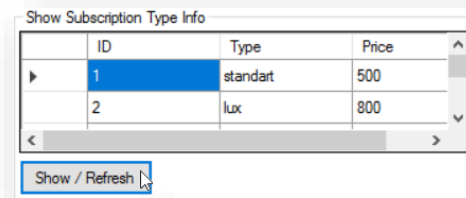
Подовжуємо термін дії абонементу (рис. 10.4):



A screenshot of a 'Renew Subscription' form. It contains two input fields: 'Subscription's ID' with '2' and 'Expiration Date' with '07/17/2036' (calendar icon). A 'Renew' button is at the bottom, with a mouse cursor hovering over it.

Рисунок 10.4 – Подовження терміну дії абонементу

Переглядаємо інформацію щодо наявних типів абонементів (рис. 10.5):

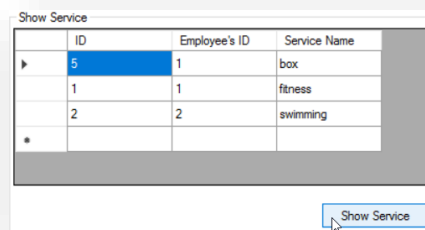


ID	Type	Price
1	standart	500
2	lux	800

Show / Refresh

Рисунок 10.5 – Виведення інформації про наявні типи абонементів

Перейдемо до вкладки «Trainer» та переглянемо інформацію стосовно наявних послуг (рис. 10.6):

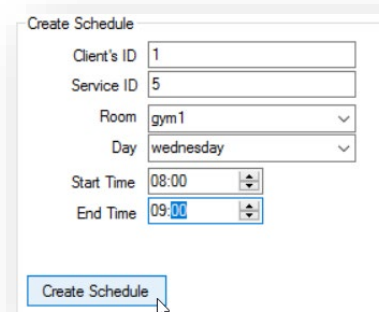


ID	Employee's ID	Service Name
5	1	box
1	1	fitness
2	2	swimming

Show Service

Рисунок 10.6 – Виведення інформації про наявні послуги

Створимо новий розклад занять (рис. 10.7):



Create Schedule

Client's ID: 1

Service ID: 5

Room: gym1

Day: wednesday

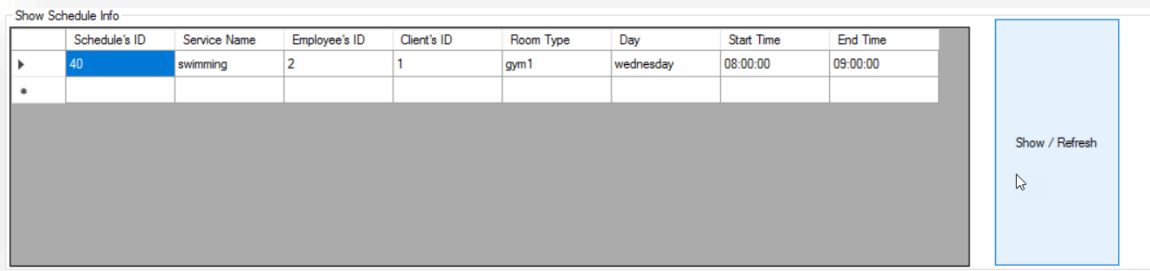
Start Time: 08:00

End Time: 09:00

Create Schedule

Рисунок 10.7 – Створення нового розкладу занять

Переглянемо інформацію про розклади занять (рис. 10.8):

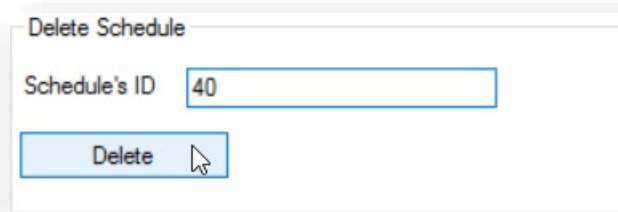


	Schedule's ID	Service Name	Employee's ID	Client's ID	Room Type	Day	Start Time	End Time
►	40	swimming	2	1	gym1	wednesday	08:00:00	09:00:00
*								

Show / Refresh

Рисунок 10.8 – Виведення інформації щодо розкладів занять.

Видалимо розклад занять (рис. 10.9):



Delete Schedule

Schedule's ID

Delete

Рисунок 10.9 – Видалення розкладу занять.

ВИСНОВОК

В результаті аналізу предметної області, проведеного при виконанні курсової роботи, сформульовані цілі створення інформаційної системи для співробітників фітнес-клубу, визначено перелік її користувачів (адміністратор, тренер) і завдань, які вони вирішують в аналізованій предметної області.

Визначено вимоги до збережених даних і спроектована база даних з 9 таблиць для використання в інформаційній системі.

Для створення інформаційної системи обрані двухзвенная архітектура клієнт - сервер бази даних, шаблон проектування MVP, СУБД PostgreSQL, мова програмування C# і WinForms для розробки привабливого користувацького інтерфейсу.

Створено додаток, який дає можливість зручно маніпулювати даними предметної області. Крім того, розмежований доступ до додатка з боку різних категорій користувачів шляхом використання механізму ролей і привілеїв і реалізований захист від несанкціонованого доступу шляхом використання механізму аутентифікації і авторизації.

За рахунок використання в створеній інформаційній системі дволанкової архітектури клієнт-сервер була досягнута стабільність і надійність системи. Завдяки використанню шаблону проектування MVP досягнута висока модульність системи.

Надалі, спроектована інформаційна система може бути доповнена великою кількістю функцій, може придбати більш масштабне значення - в перспективі розробки є ідея про створення подібної інформаційної системи не тільки для окремого фітнес-клубу, а й для мережі фітнес-клубів, а також для різних мереж фітнес-клубів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1) Малахов Є.В. Організація баз даних: конспект лекцій. – О.: ОНУ, 2018–194 с.
- 2) Блог .NET [Електроний ресурс] devblogs.microsoft.com/dotnet/ - 01.02.2019
- 3) Документація PostgreSQL [Електроний ресурс] - postgrespro.ru/docs/postgresql/9.6 – 01.02.2019
- 4) Model View Presenter pattern in C# | technical-recipes.com [Електроний ресурс] - technical-recipes.com/2015/the-model-view-presenter-pattern-in-c-a-minimalist-implementation - 01.02.2019

ДОДАТОК А. СПИСОК ЗАДАЧ КОРИСТУВАЧІВ

Користувачі		
Назва функції	Вхідні дані	Вихідні дані
Адміністратор		
Оформити співробітника на роботу	Номер паспорту та серія, ПІБ номер телефону, електрона пошта, посада, дата оформлення трудового договору, дата завершення терміну дії трудового договору	Новий співробітник, новий трудовий договір
Звільнити співробітника	Номер співробітника	Результат виконання операції або помилка, якщо запит не вдалося виконати
Додати нову послугу	Номер співробітника, назва послуги	Нова послуга
Переглянути інформацію про наявних співробітників	Відсутні	Перелік працівників
Тренер		
Створити розклад спортивних занять	Номер клієнта, номер послуги, приміщення, день тижня, час початку заняття, час завершення заняття	Новий розклад
Переглянути інформацію щодо наявних послуг	Відсутні	Перелік наявних послуг

Зареєструвати нового клієнта	ПІБ, номер телефону, електрона пошта	Новый клієнт
Переглянути інформацію щодо наявних розкладів спортивних занять	Відсутні	Перелік наявних розкладів спортивних занять
Оформити абонемент для клієнта	Номер клієнта, тип абонементу, дата оформлення, дата завершення терміну дії	Новий абонемент
Подовжити термін дії абонементу	Номер абонементу, нова дата завершення терміну дії	Результат виконання операції або помилка, якщо запит не вдалося виконати
Переглянути інформацію щодо наявних типів абонементів	Відсутні	Перелік доступних типів абонементів
Переглянути інформацію стосовно наявних клієнтів	Відсутні	Перелік клієнтів
Видалити розклад заняття	Номер розкладу	Результат виконання операції або помилка, якщо запит не вдалося виконати

ДОДАТОК Б. ЗАПИТИ НА СТВОРЕННЯ БАЗИ ДАНИХ

```
create table "employee"
(
    "employee_id" serial,
    "passport_number_and_series" varchar not null,
    "first_name" varchar not null,
    "last_name" varchar not null,
    "patronymic" varchar not null,
    "phone_number" varchar not null,
    "email" varchar,

    primary key("employee_id"),
    unique("passport_number_and_series")
);
create table "position"
(
    "position_id" serial,
    "position_name" varchar not null,
    "salary" int not null check("salary" > 0),

    primary key("position_id"),
    unique("position_name")
);
create table "contract"
(
    "contract_id" serial,
    "employee_id" int not null,
    "position_id" int not null,
    "date_of_issue" date not null,
    "expiration_date" date not null check("expiration_date" >
"date_of_issue"),

    primary key("contract_id"),
    foreign key("employee_id") references "employee" on update
cascade on delete cascade,
    foreign key("position_id") references "position" on update
cascade
);

create table "client"
(
    "client_id" serial,
    "first_name" varchar not null,
    "last_name" varchar not null,
```

```

    "patronymic" varchar not null,
    "phone_number" varchar not null,
    "email" varchar,

    primary key("client_id"),
    unique("phone_number")
);
create table "subscription_type"
(
    "subscription_type_id" serial,
    "type" varchar not null,
    "price" int not null check("price" > 0),

    primary key("subscription_type_id"),
    unique("type")
);
create table "subscription"
(
    "subscription_id" serial,
    "client_id" int not null,
    "subscription_type_id" int not null,
    "date_of_issue" date not null,
    "expiration_date" date not null check("expiration_date" >
"date_of_issue"),

    primary key("subscription_id"),
    foreign key("client_id") references "client" on update
cascade on delete cascade,
    foreign key("subscription_type_id") references
"subscription_type" on update cascade
);
create table "service"
(
    "service_id" serial,
    "employee_id" int not null,
    "service_name" varchar not null,

    primary key("service_id"),
    unique("employee_id", "service_name"),
    foreign key("employee_id") references "employee" on update
cascade on delete cascade
);
create table "room"
(
    "room_id" serial,

```

```

    "type" varchar not null,
    "is_available" bool not null default true,
    "capacity" int not null,

    primary key("room_id"),
    unique("type")
);
create table "schedule"
(
    "schedule_id" serial,
    "service_id" int not null,
    "client_id" int not null,
    "room_id" int not null,
    "day" varchar not null check("day" in('monday', 'tuesday',
'wednesday', 'thursday', 'friday', 'saturday', 'sunday')),
    "start_time" time not null,
    "end_time" time not null check("end_time" > "start_time"),

    primary key("schedule_id"),
    foreign key("service_id") references "service" on update
cascade,
    foreign key("client_id") references "client" on delete
cascade,
    foreign key("room_id") references "room" on update cascade
);

```

ДОДАТОК В. ЗАПИТИ ДО БАЗИ ДАНИХ

Оформити співробітника на роботу

```

create or replace function create_employee
(
    _passport_number_and_series varchar,
    _first_name varchar,
    _last_name varchar,
    _patronymic varchar,
    _phone_number varchar,
    _email varchar,
    _position_name varchar,
    _date_of_issue date,
    _expiration_date date
)
returns void
as $$
declare
    new_employee_id int;
begin
    insert into "employee" ("passport_number_and_series",
"first_name", "last_name", "patronymic", "phone_number",
"email")
        values (_passport_number_and_series, _first_name,
_last_name, _patronymic, _phone_number, _email);

    new_employee_id = (select max("employee_id") from
"employee");

    insert into "contract" ("employee_id", "position_id",
"date_of_issue", "expiration_date")
        values (new_employee_id,
(select "position_id" from "position" where
"position_name" = _position_name),
_date_of_issue, _expiration_date);
end $$
language plpgsql;

```

Додати нову послугу

```

create or replace function create_service
(
    _employee_id int,
    _service_name varchar
)
returns void
as $$
begin
    insert into "service"("employee_id", "service_name")
values(_employee_id, _service_name);

```

```
end $$
language plpgsql;
```

Звільнити співробітника

```
create or replace function fire_employee
(
    _employee_id int
)
returns void
as $$
begin
    update "contract"
    set "expiration_date" = current_date
    where "employee_id" = _employee_id;
end $$
language plpgsql;
```

Зареєструвати нового клієнта

```
create or replace function create_client
(
    _first_name varchar,
    _last_name varchar,
    _patronymic varchar,
    _phone_number varchar,
    _email varchar
)
returns void
as $$
begin
    insert into "client" ("first_name", "last_name",
"patronymic", "phone_number", "email")
    values (_first_name, _last_name, _patronymic,
_phone_number, _email);
end $$
language plpgsql;
```

Подовжити термін дії абонементу

```
create or replace function renew_subscription
(
    _subscription_id int,
    _expiration_date date
)
returns void
as $$
begin
    update "subscription"
    set "expiration_date" = _expiration_date
    where "subscription_id" = _subscription_id;
```

```
end $$
language plpgsql;
```

Створити розклад спортивних занять

```
create or replace function create_schedule
(
    _service_id int,
    _client_id int,
    _room_type varchar,
    _day varchar,
    _start_time time,
    _end_time time
)
returns void
as $$
begin
    insert into "schedule" ("service_id", "client_id",
"room_id", "day", "start_time", "end_time")
        values (_service_id, _client_id, (select "room_id"
from "room" where "type" = _room_type), _day, _start_time,
_end_time);
end $$
language plpgsql;
```

Оформити абонемент для клієнта

```
create or replace function subscribe_client
(
    _client_id int,
    _subscription_type varchar,
    _date_of_issue date,
    _expiration_date date
)
returns void
as $$
begin
    insert into
"subscription"("client_id","subscription_type_id","date_of_issue
","expiration_date")
        values(_client_id, (select "subscription_type_id"
                             from "subscription_type"
                             where
"type"=_subscription_type),_date_of_issue,_expiration_date);
end $$
language plpgsql;
```

Видалити розклад заняття

```
create or replace function delete_schedule
(
```

```

        _schedule_id int
    )
    returns void
    as $$
    begin
        delete from "schedule"
        where "schedule_id" = _schedule_id;
    end $$
    language plpgsql;

```

Переглянути інформацію стосовно наявних клієнтів

```

create or replace view "client_info"
as
    select
        c.client_id,
        concat(c."first_name", ' ', c."patronymic", ' ',
c."last_name") as "name",
        c.phone_number,
        c.email,
        s.subscription_id,
        st.type,
        st.price,
        s.date_of_issue,
        s.expiration_date
    from
        "client" c
    left join
        "subscription" s using("client_id")
    left join
        "subscription_type" st
using("subscription_type_id");

```

Переглянути інформацію про наявних співробітників

```

create or replace view "employee_info"
as
    select
        e."employee_id",
        concat(e."first_name", ' ', e."patronymic", ' ',
e."last_name") as "name",
        e."phone_number",
        e."passport_number_and_series",
        e."email",
        p."position_name",
        s."service_name",
        c."date_of_issue",
        c."expiration_date"
    from
        "employee" e
    join
        "contract" c using("employee_id")

```



```

left join
    "service" s using("employee_id")
join
    "position" p using("position_id");

```

Переглянути інформацію стосовно наявних розкладів спортивних занять

```

create view "schedule_info"
as
    select
        sc.schedule_id,
        sr.service_name,
        sr.employee_id,
        sc.client_id,
        r.type as "room_type",
        sc.day,
        sc.start_time,
        sc.end_time
    from
        "schedule" sc
    join
        "client" c using("client_id")
    join
        "service" sr using("service_id")
    join
        "room" r using("room_id");

```

Запобігти спробі поставити одного тренера на кілька занять одночасно

```

create or replace function check_time() returns trigger as
$$
begin
    if exists
        (select *
         from "schedule"
         where "service_id" = new.service_id -- Selection
of the same service on the same day.
         and "day" = new.day
         and ("start_time" <= new.end_time -- If time
periods overlap
         and "end_time" >= new.start_time))
    then
        raise exception 'Overlaps with existing time.';
    else
        return new;
    end if;
end $$
language plpgsql;

```

```

create trigger check_time_trigger before insert on
"schedule"
for each row execute procedure check_time();

```

Запобігти спробі створити розклад спортивних занять у переповненому приміщенні

```

create or replace function check_room_capacity() returns
trigger as $$
declare
    new_rec_capacity int;
    occupancy int;
begin
    new_rec_capacity = (select "capacity" from "room" where
"room_id" = new.room_id);
    occupancy = (select count("schedule_id")
from "schedule"
where "room_id" = new.room_id
and "day" = new.day
-- Simultaneously
and ("start_time" <= new.end_time
and "end_time" >= new.start_time));

    if (occupancy < new_rec_capacity) -- Fits in
then
        return new;
    else
        raise exception 'The room is overcrowded.';
    end if;
end $$
language plpgsql;

create trigger check_room_capacity_trigger before insert on
"schedule"
for each row execute procedure check_room_capacity();

```

Запобігти спробі створити розклад спортивних занять у недоступному приміщенні

```

create or replace function check_room_availability()
returns trigger as $$
begin
    if ((select "is_available" from "room" where "room_id"
= new.room_id) = false) -- If the room is unavailable
then
        raise exception 'The room is unavailable.';
    else
        return new;
    end if;
end $$
language plpgsql;

```

```

create trigger check_room_availability_trigger before
insert on "schedule"
for each row execute procedure
check_room_availability();

```

Запобігти спробі створити послугу, де виконавцем є адміністратор, а не тренер

```

create or replace function prevent_service_creation()
returns trigger as $$
begin
    if
        ((select "position_name" from "employee_info"
where "employee_id" = new.employee_id) = 'admin')
    then
        raise exception 'Administrators cannot provide a
service.';
    end if;
    return new;
end $$
language plpgsql;

create trigger prevent_service_creation_trigger before
insert on "service"
for each row execute procedure
prevent_service_creation();

```

Запити на створення ролі адміністратора

```

create role administrators;
grant connect on database fitness_club to administrators;
grant usage on schema public to administrators;
grant all privileges on all tables in schema public to
administrators;
grant all privileges on all sequences in schema public to
administrators;
revoke all privileges on function "create_schedule",
"delete_schedule", "create_client", "subscribe_client",
"renew_subscription" from administrators;
revoke select on "room", "service", "schedule_info",
"client_info", "subscription_type" from administrators;
revoke all privileges on "client", "subscription",
"schedule" from administrators;

```

Запити на створення ролі тренера

```

create role trainers;
grant connect on database fitness_club to trainers;
grant usage on schema public to trainers;
grant all privileges on all tables in schema public to trainers;

```

```
grant all privileges on all sequences in schema public to
trainers;
revoke all privileges on function "create_employee",
"create_service", "fire_employee" from trainers;
revoke all privileges on "employee", "contract",
"employee_info", "position" from trainers;
revoke insert, delete, update on "room", "subscription_type",
"service" from trainers;
```

Запити на створення користувачів

```
create user masha_admin with encrypted password 'admin' in
role administrators;
create user masha_trainer with encrypted password 'trainer'
in role trainers;
```

ДОДАТОК Г. ВИХІДНИЙ КОД ОСНОВНИХ КЛАСІВ

Код, що відповідає частині Model

1) Співробітник

```
using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Windows.Forms;
using Npgsql;
using Fitness_Club.Helper_Classes;
using Fitness_Club.Helper_Classes.Tables;

namespace Fitness_Club.Model
{
    class EmployeeModel
    {
        private Connection connection;
        public EmployeeModel(Connection connection)
        {
            this.connection = connection;
        }

        public List<Employee> GetEmployees()
        {
            Employee employee;
            List<Employee> employees = new List<Employee>();
            try
            {
                string query = "select * from employee_info
order by employee_id;";
                NpgsqlCommand command =
                    new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
                NpgsqlDataReader dataReader =
command.ExecuteReader();

                foreach(DbDataRecord dbDataRecord in dataReader)
                {
                    employee = new Employee(
                        dbDataRecord["employee_id"].ToString(),
                        dbDataRecord["name"].ToString(),
                        dbDataRecord["phone_number"].ToString(),

                        dbDataRecord["passport_number_and_series"].ToString(),
                        dbDataRecord["email"].ToString(),

                        dbDataRecord["position_name"].ToString(),
                        dbDataRecord["service_name"].ToString(),

                        dbDataRecord["date_of_issue"].ToString(),
```

```

dbDataRecord["expiration_date"].ToString());
        employees.Add(employee);
    }
    dataReader.Close();
}
catch(PostgresException ex)
{
    MessageBox.Show("DB error. \n" +
Convert.ToString(ex));
}
return employees;
}

public void CreateEmployee (string passport, string
firstName, string lastName, string patronymic,
                        string phoneNumber, string
email, string positionName, string dateOfIssue,
                        string expirationDate)
{
    try
    {
        string query = $"select
create_employee('{passport}', '{firstName}', '{lastName}',
'{patronymic}', " +
                        $"'{phoneNumber}','{email}',
'{positionName}', '{dateOfIssue}', '{expirationDate}');"
        NpgsqlCommand command = new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
        try
        {
            command.ExecuteNonQuery(); // NonQuery means
we are not interested in returned value
        }
        catch (PostgresException ex)
        {
            MessageBox.Show("Check input values in
pg.\n" + ex.Message);
        }
    }
    catch(PostgresException ex)
    {
        MessageBox.Show("Check input parameters. \n" +
Convert.ToString(ex));
    }
}

public void CreateSchedule(string serviceID, string
clientId, string roomId, string day, string startTime, string
endTime)
{
    try
    {

```

```

        string query = $"select
create_schedule({serviceID}, {clientId}, {roomId}, {day},
'{startTime}', '{endTime}');"
        NpgsqlCommand command = new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
        try
        {
            command.ExecuteNonQuery(); // NonQuery means
we are not interested in returned value
        }
        catch (PostgresException ex)
        {
            MessageBox.Show("Check input values in
pg.\n" + ex.Message);
        }
    }
    catch (PostgresException ex)
    {
        MessageBox.Show("Check input parameters. \n" +
Convert.ToString(ex));
    }
}
}
}

```

2) Посада

```

using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Windows.Forms;
using Npgsql;
using Fitness_Club.Helper_Classes;
using Fitness_Club.Helper_Classes.Tables;

namespace Fitness_Club.Model
{
    class PositionModel
    {
        private Connection connection;
        public PositionModel(Connection connection)
        {
            this.connection = connection;
        }
        public List<Position> GetPositionNames()
        {
            Position position;
            List<Position> positions = new List<Position>();
            try
            {
                string QueryString =

```

```

        "select position_name from \"position\"";
        NpgsqlCommand Command =
            new NpgsqlCommand(QueryString,
connection.CreateConnection.npgsqlConnection);
        NpgsqlDataReader dataReader =
            Command.ExecuteReader();
        foreach (DbDataRecord dbDataRecord in
dataReader)
        {
            position = new Position(
dbDataRecord["position_name"].ToString());
            positions.Add(position);
        }
        dataReader.Close();
    }
    catch (PostgresException ex)
    {
        MessageBox.Show("DB error. \n" +
Convert.ToString(ex));
    }
    return positions;
}
}
}

```

3) Трудовий договір

```

using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Windows.Forms;
using Npgsql;
using Fitness_Club.Helper_Classes;
using Fitness_Club.Helper_Classes.Tables;

namespace Fitness_Club.Model
{
    class ContractModel
    {
        private Connection connection;

        public ContractModel(Connection connection)
        {
            this.connection = connection;
        }
        public void FireEmployee(string employeeId)
        {
            try
            {

```



```

        string query = $"update \"contract\" set
expiration_date = current_date where employee_id =
{employeeId}";
        NpgsqlCommand command = new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
        try
        {
            command.ExecuteNonQuery(); // NonQuery means
we are not interested in returned value
        }
        catch (PostgresException ex)
        {
            MessageBox.Show("Check input values in
pg.\n" + ex.Message);
        }
    }
    catch (PostgresException ex)
    {
        MessageBox.Show("Check input parameters. \n" +
Convert.ToString(ex));
    }
}
}
}

```

4) Послуга

```

using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Windows.Forms;
using Npgsql;
using Fitness_Club.Helper_Classes;
using Fitness_Club.Helper_Classes.Tables;

namespace Fitness_Club.Model
{
    class ServiceModel
    {
        private Connection connection;
        public ServiceModel(Connection connection)
        {
            this.connection = connection;
        }
        public List<Service> GetServices()
        {
            Service service;
            List<Service> services = new List<Service>();
            try
            {
                string query = "select * from \"service\" order
by service_name;";
                NpgsqlCommand Command =

```

```

        new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
        NpgsqlDataReader dataReader =
Command.ExecuteReader();
        foreach (DbDataRecord dbDataRecord in
dataReader)
        {
            service = new
Service(dbDataRecord["service_id"].ToString(),
dbDataRecord["employee_id"].ToString(),
dbDataRecord["service_name"].ToString());
            services.Add(service);
        }
        dataReader.Close();
    }
    catch(PostgresException ex)
    {
        MessageBox.Show("DB error. \n" +
Convert.ToString(ex));
    }
    return services;
}

public void CreateService(string employeeId, string
serviceName)
{
    try
    {
        string query = $"select
create_service('{employeeId}', '{serviceName}')";
        NpgsqlCommand command = new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
        try
        {
            command.ExecuteNonQuery(); // NonQuery means
we are not interested in returned value
        }
        catch (PostgresException ex)
        {
            MessageBox.Show("Check input values in
pg.\n" + ex.Message);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Check input values.\n" +
ex.Message);
    }
}
}
}

```

5) Клієнт

```

using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Windows.Forms;
using Npgsql;
using Fitness_Club.Helper_Classes;
using Fitness_Club.Helper_Classes.Tables;

namespace Fitness_Club.Model
{
    class ClientModel
    {
        private Connection connection;
        public ClientModel(Connection connection)
        {
            this.connection = connection;
        }

        public List<Client> GetClients()
        {
            Client client;
            List<Client> clients = new List<Client>();
            try
            {
                string query = "select * from client_info order
by client_id;";
                NpgsqlCommand command = // Create the command
from the query and Connection
                new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
                NpgsqlDataReader dataReader =
command.ExecuteReader(); // Call the command, execute it at DB

                foreach (DbDataRecord dbDataRecord in dataReader)
                {
                    client = new Client(
                        dbDataRecord["client_id"].ToString(),
                        dbDataRecord["name"].ToString(),
                        dbDataRecord["phone_number"].ToString(),
                        dbDataRecord["email"].ToString(),

                        dbDataRecord["subscription_id"].ToString(),
                        dbDataRecord["type"].ToString(),
                        dbDataRecord["price"].ToString(),

                        dbDataRecord["date_of_issue"].ToString(),

                        dbDataRecord["expiration_date"].ToString());
                    clients.Add(client);
                }
                dataReader.Close();
            }
            catch { }
        }
    }
}

```

```

    }
    catch (PostgresException ex)
    {
        MessageBox.Show("DB error. \n" +
Convert.ToString(ex));
    }
    return clients;
}

public void CreateClient(string firstName, string
lastName, string patronymic, string phoneNumber, string email)
{
    try
    {
        string query = $"select
create_client('{firstName}', '{lastName}', '{patronymic}',
'{phoneNumber}', '{email}');";
        NpgsqlCommand command = new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
        try
        {
            command.ExecuteNonQuery(); // NonQuery means
we are not interested in returned value
        }
        catch (PostgresException ex)
        {
            MessageBox.Show("Check input values in
pg.\n" + ex.Message);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Check input values.\n" +
ex.Message);
    }
}
}
}

```

6) Абонемент

```

using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Windows.Forms;
using Npgsql;
using Fitness_Club.Helper_Classes;
using Fitness_Club.Helper_Classes.Tables;
namespace Fitness_Club.Model
{
    class SubscriptionModel

```

```

{
    private Connection connection;

    public SubscriptionModel(Connection connection)
    {
        this.connection = connection;
    }

    public void AddSubscription(string clientId, string
subscriptionTypeId, string dateOfIssue, string expirationDate)
    {
        try
        {
            string query = $"select
subscribe_client('{clientId}','{subscriptionTypeId}','{dateOfIss
ue}','{expirationDate}')";
            NpgsqlCommand command = new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
            try
            {
                command.ExecuteNonQuery(); // NonQuery means
we are not interested in returned value
            }
            catch (PostgresException ex)
            {
                MessageBox.Show("Check input values in
pg.\n" + ex.Message);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Check input values.\n" +
ex.Message);
        }
    }

    public void RenewSubscription(string subscriptionId,
string dateOfIssue)
    {
        try
        {
            string query = $"select
renew_subscription('{subscriptionId}','{dateOfIssue}')";
            NpgsqlCommand command = new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
            try
            {
                command.ExecuteNonQuery(); // NonQuery means
we are not interested in returned value
            }
            catch (PostgresException ex)
            {
                MessageBox.Show("Check input values in
pg.\n" + ex.Message);
            }
        }
    }
}

```

```

    }
    catch (Exception ex)
    {
        MessageBox.Show("Check input values.\n" +
ex.Message);
    }
}
}
}

```

7) Тип абонемента

```

using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Windows.Forms;
using Npgsql;
using Fitness_Club.Helper_Classes;
using Fitness_Club.Helper_Classes.Tables;

namespace Fitness_Club.Model
{
    class SubscriptionTypeModel
    {
        private Connection connection;

        public SubscriptionTypeModel(Connection connection)
        {
            this.connection = connection;
        }
        public List<SubscriptionType> GetSubscriptionTypes()
        {
            SubscriptionType subscriptionType;
            List<SubscriptionType> subscriptionTypes = new
List<SubscriptionType>();
            try
            {
                string query = "select * from
\"subscription_type\"";
                NpgsqlCommand Command =
                new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
                NpgsqlDataReader dataReader =
Command.ExecuteReader();
                foreach (DbDataRecord dbDataRecord in
dataReader)
                {
                    subscriptionType = new
SubscriptionType(dbDataRecord["subscription_type_id"].ToString(),
,
                    dbDataRecord["type"].ToString(),
dbDataRecord["price"].ToString());

```

```

        subscriptionTypes.Add(subscriptionType);
    }
    dataReader.Close();
}
catch (PostgresException ex)
{
    MessageBox.Show("DB error. \n" +
Convert.ToString(ex));
}
return subscriptionTypes;
}
}
}

```

8) Розклад спортивних занять

```

using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Windows.Forms;
using Npgsql;
using Fitness_Club.Helper_Classes;
using Fitness_Club.Helper_Classes.Tables;
namespace Fitness_Club.Model
{
    class ScheduleModel
    {
        private Connection connection;

        public ScheduleModel(Connection connection)
        {
            this.connection = connection;
        }
        public List<Schedule> GetSchedules()
        {
            Schedule schedule;
            List<Schedule> schedules = new List<Schedule>();
            try
            {
                string query = "select * from schedule_info
order by schedule_id;";
                NpgsqlCommand command =
                    new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
                NpgsqlDataReader dataReader =
command.ExecuteReader();

                foreach (DbDataRecord dbDataRecord in
dataReader)
                {
                    schedule = new Schedule(
                        dbDataRecord["schedule_id"].ToString(),

```

```

        dbDataRecord["service_name"].ToString(),
        dbDataRecord["employee_id"].ToString(),
        dbDataRecord["client_id"].ToString(),
        dbDataRecord["room_type"].ToString(),
        dbDataRecord["day"].ToString(),
        dbDataRecord["start_time"].ToString(),
        dbDataRecord["end_time"].ToString());
        schedules.Add(schedule);
    }
    dataReader.Close();
}
catch (PostgresException ex)
{
    MessageBox.Show("DB error. \n" +
Convert.ToString(ex));
}
return schedules;
}

public void CreateSchedule(string serviceId, string
clientId, string roomName, string day, string startTime,
string endTime)
{
    try
    {
        string query = $"select
create_schedule({serviceId}, {clientId}, '{roomName}', '{day}',
" +
                    $"'{startTime}', '{endTime}');"
        NpgsqlCommand command = new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
        try
        {
            command.ExecuteNonQuery(); // NonQuery means
we are not interested in returned value
        }
        catch (PostgresException ex)
        {
            MessageBox.Show("Check input values in
pg.\n" + ex.Message);
        }
    }
    catch (PostgresException ex)
    {
        MessageBox.Show("Check input parameters. \n" +
Convert.ToString(ex));
    }
}

public void DeleteSchedule(string scheduleId)
{
    try
    {
        string query = $"select
delete_schedule({scheduleId});";

```



```

        NpgsqlCommand command = new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
        try
        {
            command.ExecuteNonQuery(); // NonQuery means
we are not interested in returned value
        }
        catch (PostgresException ex)
        {
            MessageBox.Show("Check input values in
pg.\n" + ex.Message);
        }
    }
    catch (PostgresException ex)
    {
        MessageBox.Show("Check input parameters. \n" +
Convert.ToString(ex));
    }
}
}
}

```

9) Приміщення

```

using System;
using System.Collections.Generic;
using System.Data.Common;
using System.Windows.Forms;
using Npgsql;
using Fitness_Club.Helper_Classes;
using Fitness_Club.Helper_Classes.Tables;

namespace Fitness_Club.Model
{
    class RoomModel
    {
        private Connection connection;

        public RoomModel(Connection connection)
        {
            this.connection = connection;
        }

        public List<Room> GetRooms()
        {
            Room room;
            List<Room> rooms = new List<Room>();
            try
            {
                string query = "select * from room order by
room_id;";
                NpgsqlCommand command =

```

```

        new NpgsqlCommand(query,
connection.CreateConnection.npgsqlConnection);
        NpgsqlDataReader dataReader =
command.ExecuteReader();

        foreach (DbDataRecord dbDataRecord in
dataReader)
        {
            room = new Room(
                dbDataRecord["room_id"].ToString(),
                dbDataRecord["is_available"].ToString(),
                dbDataRecord["type"].ToString(),
                dbDataRecord["capacity"].ToString());
            rooms.Add(room);
        }
        dataReader.Close();
    }
    catch (PostgresException ex)
    {
        MessageBox.Show("DB error. \n" +
Convert.ToString(ex));
    }
    return rooms;
}
}
}

```

Код, що відповідає частині Presenter

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Fitness_Club.Model;
using Fitness_Club.Helper_Classes;
using System.Windows.Forms;

namespace Fitness_Club
{
    class Presenter
    {
        public void ShowClient(Factory factory, DataGridView
dgv)
        {
            foreach (var i in factory.clientModel.GetClients())
                dgv.Rows.Add(i.clientId, i.name, i.phoneNumber,
i.email, i.subscriptionId, i.type, i.price,
i.dateOfIssue, i.expirationDate);
        }
    }
}

```

```

        public void ShowEmployee(Factory factory, DataGridView
dgv)
        {
            foreach (var i in
factory.employeeModel.GetEmployees())
                dgv.Rows.Add(i.employeeId, i.name,
i.phoneNumber, i.passport, i.email, i.positionName,
i.serviceName,
                                i.dateOfIssue, i.expirationDate);
        }
        public void ShowSchedule(Factory factory, DataGridView
dgv)
        {
            foreach (var i in
factory.scheduleModel.GetSchedules())
                dgv.Rows.Add(i.scheduleId, i.serviceName,
i.employeeId, i.clientId, i.roomType, i.day, i.startTime,
                                i.endTime);
        }
        public void ShowSubscriptionType(Factory factory,
DataGridView dgv)
        {
            foreach (var i in
factory.subscriptionTypeModel.GetSubscriptionTypes())
                dgv.Rows.Add(i.subscriptionTypeId, i.type,
i.price);
        }
        public void ShowService(Factory factory, DataGridView
dgv)
        {
            foreach (var i in
factory.serviceModel.GetServices())
                dgv.Rows.Add(i.serviceId, i.employeeId,
i.serviceName);
        }
        public void RegisterClient(Factory factory, string
firstName, string lastName, string patronymic,
                                string phoneNumber, string
email)
        {
            factory.clientModel.CreateClient(firstName,
lastName, patronymic, phoneNumber, email);
        }
        public void RegisterEmployee(Factory factory, string
passport, string firstName, string lastName,
                                string patronymic, string
phoneNumber, string email, string positionName,
                                string dateOfIssue, string
expirationDate)
        {
            factory.employeeModel.CreateEmployee(passport,
firstName, lastName, patronymic, phoneNumber, email,

```

```

                                                                    positionName,
dateOfIssue, expirationDate);
    }
    public void AddService(Factory factory, string
employeeId, string serviceName)
    {
        factory.serviceModel.CreateService(employeeId,
serviceName);
    }
    public void CreateSchedule(Factory factory, string
serviceId, string clientId, string roomName, string day, string
startTime,
                                string endTime)
    {
        factory.scheduleModel.CreateSchedule(serviceId,
clientId, roomName, day, startTime, endTime);
    }
    public void SubscribeClient(Factory factory, string
clientId, string subscriptionTypeId, string dateOfIssue, string
expirationDate)
    {
        factory.subscriptionModel.AddSubscription(clientId,
subscriptionTypeId, dateOfIssue, expirationDate);
    }
    public void FireEmployee(Factory factory, string
employeeId)
    {
        factory.contractModel.FireEmployee(employeeId);
    }
    public void RenewSubscription(Factory factory, string
subscriptionId, string dateOfIssue)
    {
        factory.subscriptionModel.RenewSubscription(subscriptionId,
dateOfIssue);
    }
    public void DeleteSchedule (Factory factory, string
scheduleId)
    {
        factory.scheduleModel.DeleteSchedule(scheduleId);
    }
    #region ComboBoxFill
    public void FillPositions(Factory factory, ComboBox
comboBox)
    {
        foreach (var i in
factory.positionModel.GetPositionNames())
            comboBox.Items.Add(i.Name);
    }

    public void FillServices(Factory factory, ComboBox
comboBox)
    {

```

```

        foreach (var i in
factory.serviceModel.GetServices())
            comboBox.Items.Add(i.serviceName);
    }

    public void FillRooms(Factory factory, ComboBox
comboBox)
    {
        foreach (var i in factory.roomModel.GetRooms())
            comboBox.Items.Add(i.type);
    }
    public void FillSubscriptionType(Factory factory,
ComboBox comboBox)
    {
        foreach (var i in
factory.subscriptionTypeModel.GetSubscriptionTypes())
            comboBox.Items.Add(i.type);
    }
    #endregion
}
}

```

Код, що відповідає частині View

```

using System;
using Npgsql;
using System.Windows.Forms;
using Fitness_Club.Helper_Classes;

namespace Fitness_Club
{
    public partial class View : Form
    {
        Factory factory;
        Presenter presenter = new Presenter();
        public View()
        {
            InitializeComponent();
            GetLoginTab();
        }

        public void Login(string login, string password)
        {
            factory = new Factory("127.0.0.1", "5433", login,
password, "fitness_club");
        }
        private void btnShowClient_Click(object sender,
EventArgs e)
        {
            dgvShowClient.Columns.Clear();
            dgvShowClient.Columns.Add("clientId", "Client's
ID");

```

```

        dgvShowClient.Columns.Add("name", "Name");
        dgvShowClient.Columns.Add("phoneNumber", "Phone
Number");
        dgvShowClient.Columns.Add("email", "Email");
        dgvShowClient.Columns.Add("subscriptionId", "Sub
ID");
        dgvShowClient.Columns.Add("type", "Type");
        dgvShowClient.Columns.Add("price", "Price");
        dgvShowClient.Columns.Add("dateOfIssue", "Date Of
Issue");
        dgvShowClient.Columns.Add("expirationDate",
"Expiration Date");

        presenter.ShowClient(factory, dgvShowClient);
    }

    private void btnRegisterClient_Click(object sender,
EventArgs e)
    {
        presenter.RegisterClient(factory,
textBoxClientFirstName.Text,
                                textBoxClientLastName.Text,
textBoxClientPatronymic.Text,
textBoxClientPhoneNumber.Text,
                                textBoxClientEmail.Text);
    }

    private void btnSignIn_Click_1(object sender, EventArgs
e)
    {
        try
        {
            Login(textBoxLogin.Text, textBoxPassword.Text);
            switch (comboBoxRole.SelectedItem)
            {
                case "trainer":
                    GetTrainerTab();
                    break;
                case "admin":
                    GetAdminTab();
                    break;
            }
            tabControl1.TabPages.Remove(loginTab);
        }
        catch (PostgresException pgex)
        {
            MessageBox.Show("Incorrect login or password.
\n" + Convert.ToString(pgex));
        }
    }

```

```

    }

    #region GetTabs
    private void GetAdminTab()
    {
        tabControl1.TabPages.Add(adminTab);
        presenter.FillPositions(factory,
comboBoxRegisterEmployeePosition);
    }
    private void GetTrainerTab()
    {
        tabControl1.TabPages.Add(trainerTab);
        tabControl1.TabPages.Add(clientTab);

        comboBoxCreateScheduleDay.Items.Add("monday");
        comboBoxCreateScheduleDay.Items.Add("tuesday");
        comboBoxCreateScheduleDay.Items.Add("wednesday");
        comboBoxCreateScheduleDay.Items.Add("thursday");
        comboBoxCreateScheduleDay.Items.Add("friday");
        comboBoxCreateScheduleDay.Items.Add("sunday");
        comboBoxCreateScheduleDay.Items.Add("saturday");

        presenter.FillRooms(factory,
comboBoxCreateScheduleRoom);
        presenter.FillSubscriptionType(factory,
comboBoxSubscribeClientSubscriptionType);
    }
    private void GetLoginTab()
    {
        // Filling of combobox with roles
        comboBoxRole.Items.Add("admin");
        comboBoxRole.Items.Add("trainer");

        // Leave only authentication tab
        tabControl1.TabPages.Remove(adminTab);
        tabControl1.TabPages.Remove(trainerTab);
        tabControl1.TabPages.Remove(clientTab);
    }
    #endregion

    private void btnRegisterEmployee_Click(object sender,
EventArgs e)
    {
        presenter.RegisterEmployee(factory,

textBoxRegisterEmployeePassport.Text,

textBoxRegisterEmployeeFirstName.Text,

textBoxRegisterEmployeeLastName.Text,

textBoxRegisterEmployeePatronymic.Text,

```

```

textBoxRegisterEmployeePhoneNumber.Text,

textBoxRegisterEmployeeEmail.Text,

comboBoxRegisterEmployeePosition.SelectedItem.ToString(),

dateTimePickerRegisterEmployeeDateOfIssue.Value.ToShortDateStrin
g(),

dateTimePickerRegisterEmployeeExpirationDate.Value.ToShortDateSt
ring());
    }

    private void btnShowEmployee_Click(object sender,
EventArgs e)
    {
        dgvShowEmployee.Columns.Clear();
        dgvShowEmployee.Columns.Add("employeeId",
"Employee's ID");
        dgvShowEmployee.Columns.Add("name", "Name");
        dgvShowEmployee.Columns.Add("phoneNumber", "Phone
Number");
        dgvShowEmployee.Columns.Add("passport", "Passport");
        dgvShowEmployee.Columns.Add("email", "Email");
        dgvShowEmployee.Columns.Add("positionName",
"Position");
        dgvShowEmployee.Columns.Add("serviceName",
"Service");
        dgvShowEmployee.Columns.Add("dateOfIssue", "Date Of
Issue");
        dgvShowEmployee.Columns.Add("expirationDate",
"Expiration Date");

        presenter.ShowEmployee(factory, dgvShowEmployee);
    }

    private void btnAddService_Click(object sender,
EventArgs e)
    {
        presenter.AddService(factory,
textBoxAddServiceEmployeeId.Text,
textBoxAddServiceServiceName.Text);
    }

    private void btnFireEmployee_Click(object sender,
EventArgs e)
    {
        presenter.FireEmployee(factory,
textBoxFireEmployeeID.Text);
    }

```



```

        private void btnShowSchedule_Click(object sender,
EventArgs e)
        {
            dataGridViewShowShedule.Columns.Clear();
            dataGridViewShowShedule.Columns.Add("scheduleId",
"Schedule's ID");
            dataGridViewShowShedule.Columns.Add("serviceName",
"Service Name");
            dataGridViewShowShedule.Columns.Add("employeeId",
"Employee's ID");
            dataGridViewShowShedule.Columns.Add("clientId",
"Client's ID");
            dataGridViewShowShedule.Columns.Add("roomType",
"Room Type");
            dataGridViewShowShedule.Columns.Add("day", "Day");
            dataGridViewShowShedule.Columns.Add("startTime",
"Start Time");
            dataGridViewShowShedule.Columns.Add("endTime", "End
Time");

            presenter.ShowSchedule(factory,
dataGridViewShowShedule);
        }

        private void btnCreateSchedule_Click(object sender,
EventArgs e)
        {
            presenter.CreateSchedule(factory,

textBoxCreateScheduleService.Text,

textBoxCreateScheduleClientID.Text,

comboBoxCreateScheduleRoom.SelectedItem.ToString(),

comboBoxCreateScheduleDay.SelectedItem.ToString(),

dateTimePickerCreateScheduleStartTime.Value.ToShortTimeString(),

dateTimePickerCreateScheduleEndTime.Value.ToShortTimeString());
        }

        private void btnSubscribeClient_Click(object sender,
EventArgs e)
        {
            presenter.SubscribeClient(factory,
textBoxSubscribeClientClientID.Text,

comboBoxSubscribeClientSubscriptionType.SelectedItem.ToString(),

dateTimePickerSubscribeClientDateOfIssue.Value.ToShortDateString()
),

```

```

dateTimePickerSubscribeClientExpirationDate.Value.ToShortDateStr
ing());
    }

    private void btnRenewSubscription_Click(object sender,
EventArgs e)
    {
        presenter.RenewSubscription(factory,
textBoxRenewSubscriptionSubscriptionID.Text,

dateTimePickerRenewSubscriptionExpirationDate.Value.ToShortDates
tring());
    }

    private void btnShowSubscriptionType_Click(object
sender, EventArgs e)
    {
        dataGridViewSubType.Columns.Clear();

dataGridViewSubType.Columns.Add("subscriptionTypeId", "ID");
        dataGridViewSubType.Columns.Add("type", "Type");
        dataGridViewSubType.Columns.Add("price", "Price");

        presenter.ShowSubscriptionType(factory,
dataGridViewSubType);
    }

    private void btnShowService_Click(object sender,
EventArgs e)
    {
        dataGridViewShowService.Columns.Clear();
        dataGridViewShowService.Columns.Add("serviceId",
"ID");
        dataGridViewShowService.Columns.Add("employeeId",
"Employee's ID");
        dataGridViewShowService.Columns.Add("serviceName",
"Service Name");

        presenter.ShowService(factory,
dataGridViewShowService);
    }

    private void btnDeleteSchedule_Click(object sender,
EventArgs e)
    {
        presenter.DeleteSchedule(factory,
textBoxDeleteScheduleScheduleID.Text);
    }
}

```