Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота №2

із дисципліни «*Технології розробки програмного забезпечення*»

Тема: «Основи проектування»

Виконав:
Студент групи ІА-31
Клим’юк В.Л.

Перевірив:
Мягкий М.Ю.

Київ-2025

Тема: Система запису на прийом до лікаря (strategy, adapter, observer, facade, visitor, client-server).
Система дозволяє пацієнтам шукати та записуватись на прийоми до лікарів, де їм виноситься висновок за результатами прийому. Також наявні функції відміни прийому та додавання відгуків про лікарів.

Мета: Вибрати зручну систему побудови UML-діаграм та навчитися будувати діаграми варіантів використання для системи що проектується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.

Теоретичні відомості:

Мова UML є загальноцільовою мовою візуального моделювання, яка розроблена для специфікації, візуалізації, проєктування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Мова UML є досить строгим та потужним засобом моделювання, який може бути ефективно використаний для побудови концептуальних, логічних та графічних 18 моделей складних систем різного цільового призначення. Ця мова увібрала в себе найкращі якості та досвід методів програмної інженерії, які з успіхом використовувалися протягом останніх років при моделюванні великих та складних систем.

Діаграма – це графічне уявлення сукупності елементів моделі у формі зв'язкового графа, вершинам і ребрам (дугам) якого приписується певна семантика. Нотація канонічних діаграм є основним засобом розробки моделей мовою UML.

У нотації мови UML визначено такі види діаграм:

- варіантів використання (use case diagram);
- класів (class diagram);
- кооперації (collaboration diagram);
- послідовності (sequence diagram);
- станів (statechart diagram);
- діяльності (activity diagram);
- компонентів (component diagram);
- розгортання (deployment diagram).

Діаграма варіантів використання (Use-Cases Diagram) – це UML діаграма за допомогою якої у графічному вигляді можна зобразити вимоги

до системи, що розробляється. Діаграма варіантів використання – це вихідна концептуальна модель проєктованої системи, вона не описує внутрішню побудову системи.

Діаграма використання складається з:

- Акторів - будь-які об'єкти, суб'єкти чи системи, що взаємодіють з модельованою бізнес-системою ззовні для досягнення своїх цілей або вирішення певних завдань.
- Варіантів використання - служать для опису служб, які система надає актору.
- Відношень: асоціація, узагальнення, залежність, включення, розширення.

Діаграми класів використовуються при моделюванні програмних систем найчастіше. Вони є однією із форм статичного опису системи з погляду її проєктування, показуючи її структуру. Діаграма класів не відображає динамічної поведінки об'єктів зображених на ній класів. На діаграмах класів показуються класи, інтерфейси та відносини між ними.

Діаграма класів містить у собі класи, їхні методи та атрибути, зв’язки. Методи та атрибути мають 4 модифікатори доступу: public, package, protected, private. Зв’язки налічують у собі асоціацію, агрегацію, композицію, успадкування тощо.

Завдання:

- Ознайомитись з короткими теоретичними відомостями.
- Проаналізувати тему та спроєктувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.
- Спроєктувати діаграму класів предметної області.
- Вибрати 3 варіанти використання та написати за ними сценарії використання.
- На основі спроєктованої діаграми класів предметної області розробити основні класи та структуру бази даних системи. Класи даних повинні реалізувати шаблон Repository для взаємодії з базою даних.
- Нарисувати діаграму класів для реалізованої частини системи.
- Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму варіантів використання відповідно, діаграму

класів системи, вихідні коди класів системи, а також зображення структури бази даних

Хід роботи:

Діаграма варіантів використання

Актори:
- Користувач
- Пацієнт
- Лікар
- Адміністратор

Варіанти використання:
- Реєстрація
- Авторизація
- Перегляд лікарів
- Створити відгук про лікаря
- Запис на прийом
- Відміна прийому
- Перегляд прийомів
- Закінчення прийому
- Запис пацієнта на наступний прийом
- Створення/видалення/зміна прийомів
- Створення/видалення/зміна лікарів
- Створення/видалення/зміна клінік
- Створення/видалення/зміна спеціальностей

Зв’язки:
- Користувач - Реєстрація
- Користувач - Авторизація
- Користувач - Перегляд лікарів
- Пацієнт - Створити відгук про лікаря
- Створити відгук про лікаря - (include) - Перегляд лікарів
- Пацієнт - Запис на прийом
- Запис на прийом - (include) - Перегляд лікарів
- Пацієнт - відміна прийому

- Відміна прийому - (include) - Перегляд прийомів
- Пацієнт - Перегляд прийомів
- Лікар - Закінчення прийому
- Закінчення прийому - (include) - Перегляд прийомів
- Запис пацієнта на наступний прийом - (extend) - Закінчення прийому
- Адміністратор - Створення/видалення/зміна прийомів
- Створення/видалення/зміна прийомів - (include) - Перегляд прийомів
- Адміністратор - Створення/видалення/зміна лікарів
- Створення/видалення/зміна лікарів - (include) - Перегляд лікарів
- Адміністратор - Створення/видалення/зміна клінік
- Адміністратор - Створення/видалення/зміна спеціальностей

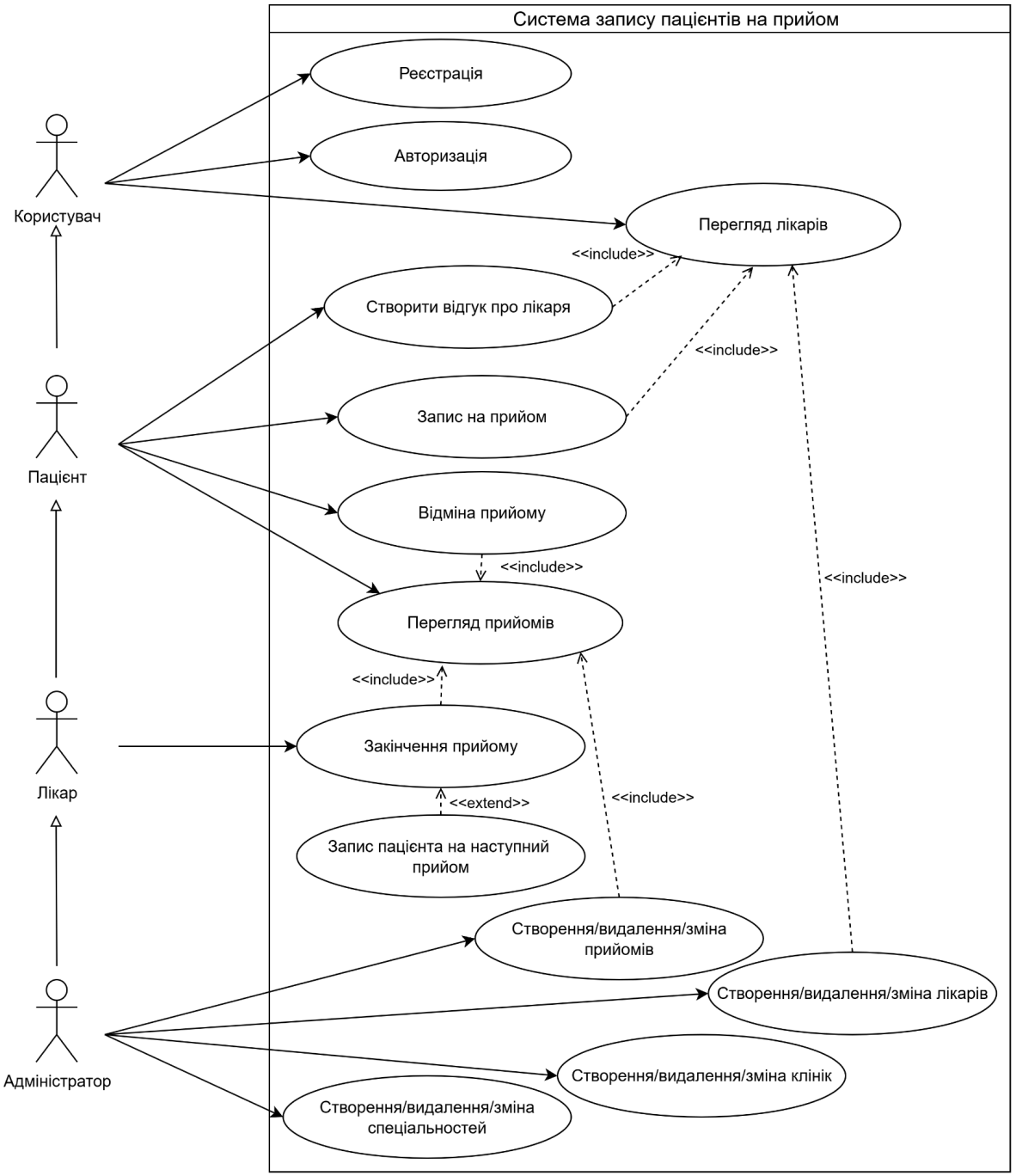

Рис. 1 - Діаграма використання

Сценарії використання:

1. **Сценарій використання “Запис на прийом”**

Передумови: Пацієнт авторизований у системі, лікар існує та має вільні місця для запису на прийом.

Постумови: Створюється запис на прийом до обраного лікаря.

Взаємодіючі сторони: Пацієнт, Система запису пацієнтів на прийом.

Короткий опис: Цей варіант використання визначає запис пацієнта на прийом.

Основний потік подій:

Цей варіант використання запускається коли користувач хоче записатись на прийом до лікаря.

1. Користувач шукає лікаря.
2. Користувач переходить на профіль лікаря.
3. Користувач обирає доступні час та дату запису.
4. Користувач підтверджує дію запису.

Винятки:

Виняток №1: Місце було зайняте. Якщо обране місце виявилось зайнятим, то користувач отримує помилку і повертається до початку основного потоку.

Примітки: Відсутні.

2. **Сценарій використання “Закінчення прийому”**

Передумови: Пацієнт записаний на прийом, який відбувається в день запуску варіанту використання.

Постумови: Прийом помічається як завершений, вноситься висновок про прийом.

Взаємодіючі сторони: Лікар, Система запису пацієнтів на прийом.

Короткий опис: Цей варіант використання визначає те як завершується прийом у лікаря.

Основний потік подій: Цей варіант використання запускається коли настає час прийому пацієнта.

1. Існує запис на прийом від пацієнта.

2. Настає час і дата прийому.
3. Лікар приймає пацієнта.
4. Лікар знаходить відповідний прийом.
5. Лікар заносить результати прийому.
6. Лікар натискає кнопку завершення прийому.

Винятки:

Виняток №1: Прийом не було завершено вчасно. Прийом помічається як скасований.

Примітки: Відсутні.

3. **Сценарій використання “Створити відгук про лікаря”**

Передумови: Пацієнт має хоча б один завершений прийом у лікаря.

Постумови: В разі успіху створюється відгук про обраного лікаря.

Взаємодіючі сторони: Пацієнт, Система запису пацієнтів на прийом.

Короткий опис: Цей варіант використання визначає як пацієнт пише відгук про лікаря.

Основний потік подій: Цей варіант використання запускається коли пацієнт відкриває список лікарів.

1. Пацієнт знаходить потрібного лікаря.
2. Пацієнт переходить у профіль лікаря.
3. Пацієнт натискає на кнопку додавання відгуку.
4. Пацієнт вводить текст та ставить оцінку.
5. Пацієнт підтверджує дію.

Винятки:

Виняток №1: Пацієнт не мав завершених прийомів у лікаря. Система повідомляє користувачу про неможливість залишити відгук.

Примітки: Відсутні.

Структура БД:

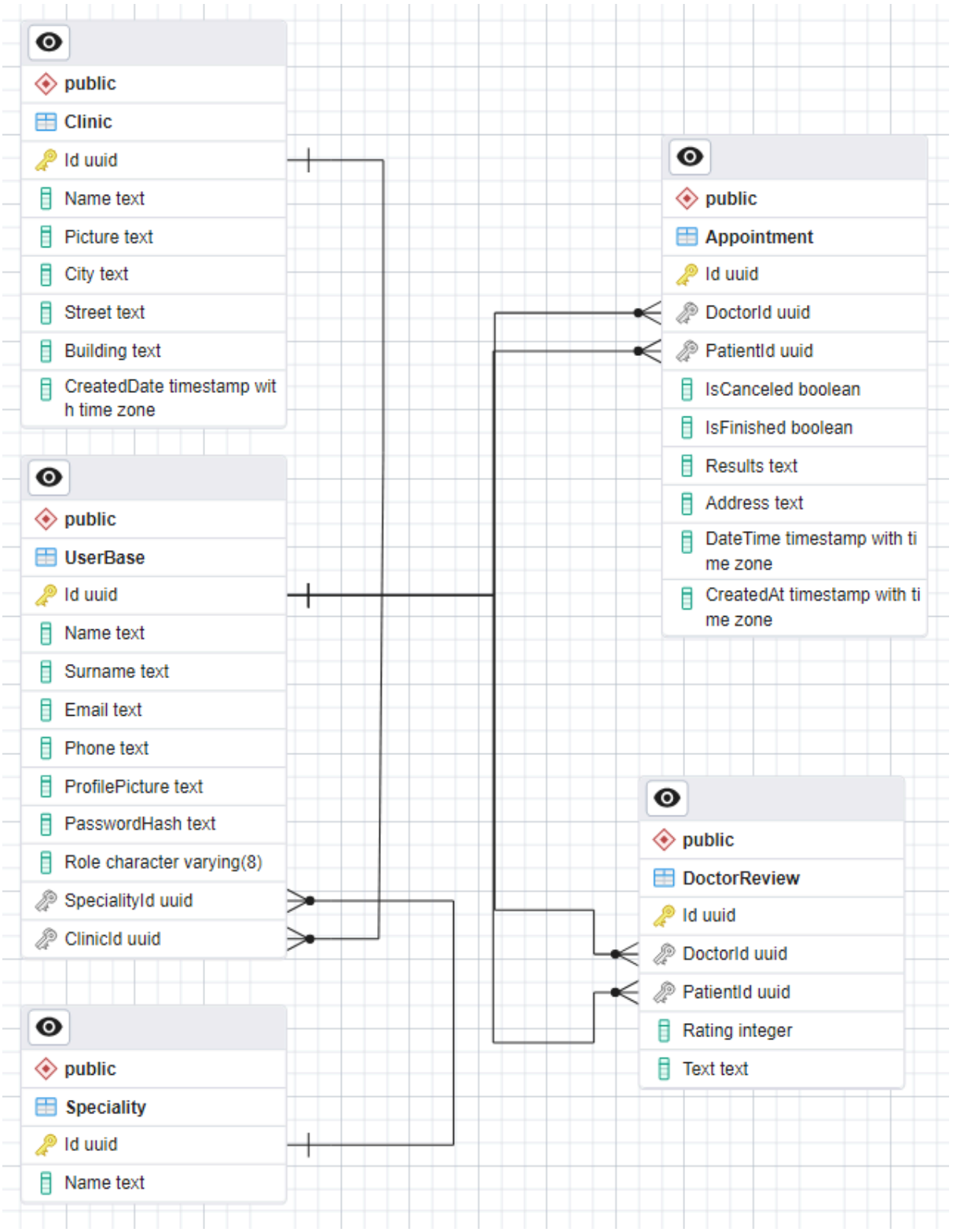

Рис. 2 - Структура БД

Діаграми класів:

Репозиторії:

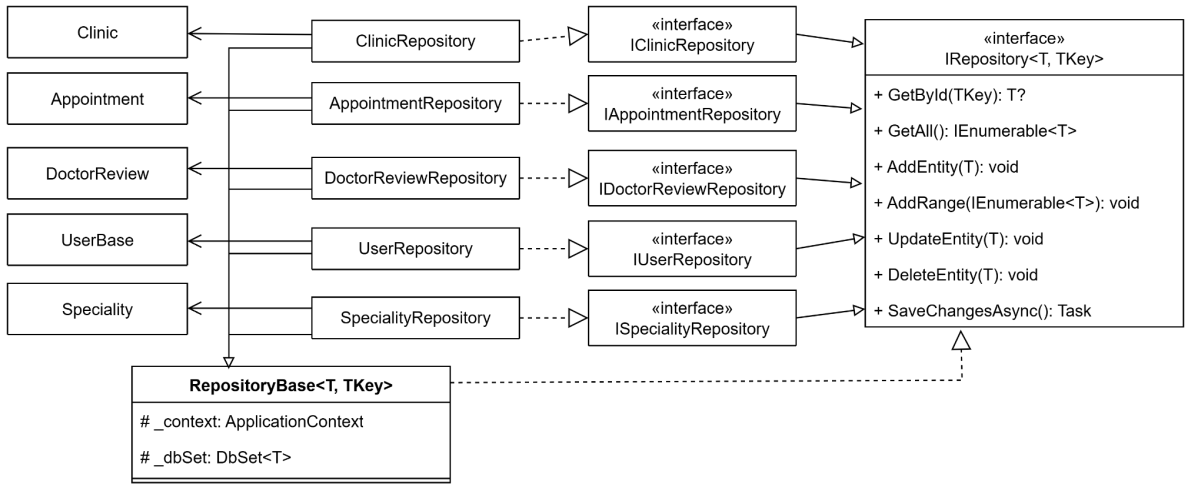

Рис. 3 - Діаграма класів репозиторіїв

Класи даних:

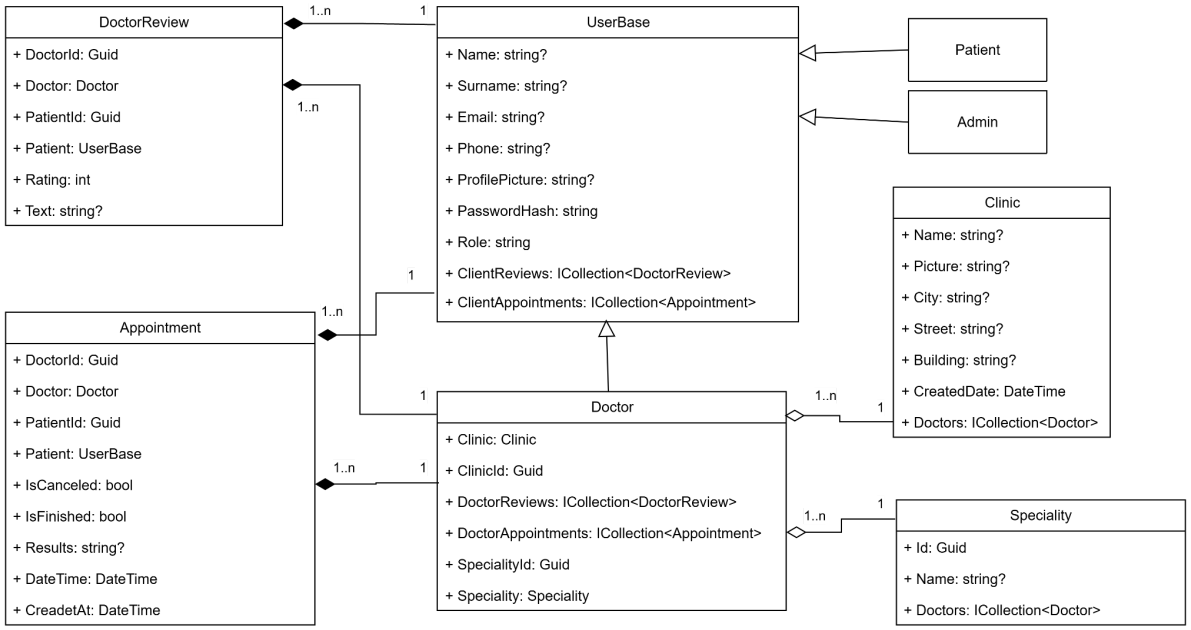

Рис. 4 - Діаграма класів даних

Вихідний код класів:

```
namespace ClinicBooking.Data.Repositories.Abstraction
{
    Ссылок: 8
    public interface IRepository<T, in TKey> where T : class
    {
        Ссылок: 1
        T? GetById(TKey key);
        Ссылок: 1
        IEnumerable<T> GetAll();
        Ссылок: 1
        void AddEntity(T entity);
        Ссылок: 1
        void AddRange(IEnumerable<T> entities);
        Ссылок: 1
        void UpdateEntity(T entity);
        Ссылок: 1
        void DeleteEntity(T entity);
        Ссылок: 1
        Task SaveChangesAsync();
    }
}
```

Рис. 5 - Код інтерфейсу IRepository<T, TKey>

```
using ClinicBooking.Data.Entities.AppContext;
using Microsoft.EntityFrameworkCore;

namespace ClinicBooking.Data.Repositories.Abstraction
{
    Ссылок: 15
    public class RepositoryBase<T, TKey> : IRepository<T, TKey> where T : class
    {
        protected readonly ApplicationContext _context;
        protected readonly DbSet<T> _dbSet;

        Ссылок: 7
        public RepositoryBase(ApplicationContext context)
        {
            _context = context;
            _dbSet = _context.Set<T>();
        }

        Ссылок: 1
        public void AddEntity(T entity) => _dbSet.Add(entity);

        Ссылок: 1
        public void AddRange(IEnumerable<T> entities) => _dbSet.AddRange(entities);

        Ссылок: 1
        public void DeleteEntity(T entity) => _dbSet.Remove(entity);

        Ссылок: 1
        public IEnumerable<T> GetAll() => _dbSet.ToList();

        Ссылок: 1
        public T? GetById(TKey key) => _dbSet.Find(key);

        Ссылок: 1
        public Task SaveChangesAsync() => _context.SaveChangesAsync();

        Ссылок: 1
        public void UpdateEntity(T entity) => _dbSet.Update(entity);
    }
}
```

Рис. 6 - Код класу RepositoryBase<T, TKey>

```
using System.ComponentModel.DataAnnotations;

namespace BookingClinic.Data.Entities
{
    public class UserBase
    {
        [Key]
        public Guid Id { get; set; }
        public string Name { get; set; }
        public string Surname { get; set; }
        public string Email { get; set; }
        public string Phone { get; set; }
        public string? ProfilePicture { get; set; }
        public string PasswordHash { get; set; }
        public string Role { get; set; }
        public ICollection<Appointment> ClientAppointments { get; set; }
        public ICollection<DoctorReview> ClientReviews { get; set; }
    }
}
```

Рис. 7 - Код класу UserBase

```
using System.ComponentModel.DataAnnotations;

namespace BookingClinic.Data.Entities
{

    public class Clinic
    {
        [Key]

        public Guid Id { get; set; }

        public string Name { get; set; }

        public string? Picture { get; set; }

        public string City { get; set; }

        public string Street { get; set; }

        public string Building { get; set; }

        public DateTime CreatedDate { get; set; }

        public ICollection<Doctor> Doctors { get; set; }
    }
}
```

Рис. 8 - Код класу Clinic

```
using System.ComponentModel.DataAnnotations;

namespace BookingClinic.Data.Entities
{
    Ссылок: 7
    public class DoctorReview
    {
        [Key]
        Ссылок: 0
        public Guid Id { get; set; }
        Ссылок: 1
        public Guid DoctorId { get; set; }
        Ссылок: 1
        public Doctor Doctor { get; set; }
        Ссылок: 1
        public Guid PatientId { get; set; }
        Ссылок: 1
        public UserBase Patient { get; set; }
        Ссылок: 0
        public int Rating { get; set; }
        Ссылок: 0
        public string? Text { get; set; }
    }
}
```

Рис. 9 - Код класу DoctorReview

```
using System.ComponentModel.DataAnnotations;

namespace BookingClinic.Data.Entities
{
    Ссылок: 6
    public class Speciality
    {
        [Key]
        Ссылок: 0
        public Guid Id { get; set; }
        Ссылок: 0
        public string? Name { get; set; }
        Ссылок: 1
        public ICollection<Doctor> Doctors { get; set; }
    }
}
```

Рис. 10 - Код класу Speciality

```
using System.ComponentModel.DataAnnotations;

namespace BookingClinic.Data.Entities
{
    Ссылок: 7
    public class Appointment
    {
        [Key]
        Ссылок: 0
        public Guid Id { get; set; }
        Ссылок: 1
        public Guid DoctorId { get; set; }
        Ссылок: 1
        public Doctor Doctor { get; set; }
        Ссылок: 1
        public Guid PatientId { get; set; }
        Ссылок: 1
        public UserBase Patient { get; set; }
        Ссылок: 0
        public bool IsCanceled { get; set; }
        Ссылок: 0
        public bool IsFinished { get; set; }
        Ссылок: 0
        public string? Results { get; set; }
        Ссылок: 0
        public string Address { get; set; }
        Ссылок: 0
        public DateTime DateTime { get; set; }
        Ссылок: 0
        public DateTime CreatedAt { get; set; }
    }
}
```

Рис. 11 - Код класу Appointment

Контрольні питання:

1. Що таке UML?

UML (Unified Modeling Language) - це загальноцільова мова візуального моделювання, яка застосовується для специфікації, візуалізації, проектування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Ця мова є потужним засобом

моделювання для побудови різних моделей складних систем, увібравши в себе найкращі якості та досвід методів програмної інженерії

2. Що таке діаграма класів UML?

Діаграма класів показує класи, інтерфейси, їхні атрибути та методи, а також зв’язки між ними. Вона є однією з форм статичного опису з погляду проектування програмної системи, показуючи її структуру.

3. Які діаграми UML називають канонічними?

У нотації мови UML визначено наступні діаграми:

- Класів
- Варіантів використання
- Кооперації
- Послідовності
- Станів
- Діяльності
- Компонентів
- Розгортання

4. Що таке діаграма варіантів використання?

Діаграма варіантів використання - це діаграма, що показує те як актори взаємодіють з системою через варіанти використання без деталізації внутрішньої структури. Діаграми використання призначені для формулювання загальних вимоги до функціональної поведінки проектованої системи і створення основи для виконання аналізу, проектування, розробки та тестування.

5. Що таке варіант використання?

Варіант використання описує служби, які система надає актору, і позначається на діаграмі варіантів використання еліпсом.

6. Які відношення можуть бути відображені на діаграмі використання?

На діаграмі використання можна зобразити відношення:

- Асоціації - ставлення між актором та варіантом використання.
- Узагальнення - показує що нащадок успадковує атрибути у батьківського елементу.
- Залежності - показує що зміна одного елемента призводить до зміни будь-якого іншого елемента.
- Включення - показує що деякий варіант використання містить поведінку що визначена в іншому варіанті.
- Розширення - показує, що варіант використання розширює базову послідовність дій та вставляє власну послідовність.

7. Що таке сценарій?

Сценарій використання - це покроковий текстовий опис того що відбувається при взаємодії актора з варіантом використання.

8. Що таке діаграма класів?

Діаграма класів показує класи, інтерфейси, їхні атрибути та методи, а також зв’язки між ними. Вона є однією з форм статичного опису з погляду проектування програмної системи, показуючи її структуру.

9. Які зв’язки між класами ви знаєте?

Між класами бувають зв’язки:

- Асоціації - показують загальні зв’язки між класами.
- Наслідування - показують те що один клас наслідує властивості іншого класу.
- Агрегації - показують що клас складається з інших класів, які можуть існувати окремо.
- Композиції - показують що клас складається з інших класів, які не можуть існувати окремо.

10. Чим відрізняється композиція від агрегації?

Агрегація показує що складові частини класу можуть існувати без нього, а композиція - ні.

11. Чим відрізняються зв'язки типу агрегації від зв'язків композиції на діаграмах класів?

На діаграмі класів агрегація позначається пустим ромбом, а композиції - заповненим.

12. Що являють собою нормальні форми баз даних?

Нормальні форми дозволяють привести БД до виду, що забезпечує мінімальну логічну надмірність і не зменшує продуктивність роботи або обсягу даних. Серед нормальних форм існують:

- Перша. Кожен атрибут містить лише одне значення.
- Друга - Всі неключові атрибути залежать від первинного ключа.
- Третя - Відсутні транзитивні функціональні залежності неключових атрибутів від ключових.
- Посилена третя. Кожна нетривіальна і неприведена зліва функціональна залежність має в якості свого детермінанта певний потенційний ключ

13. Що таке фізична модель бази даних? Логічна?

Логічна модель бази даних описує структуру даних (таблиці, стовпці, відношення) та правила для управління ними, незалежно від конкретної СУБД.
Фізична модель бази даних описує, як саме дані будуть зберігатися на дисках, залежно від обраної СУБД та апаратного забезпечення.

14. Який взаємозв'язок між таблицями БД та програмними класами?

У базі даних кожна таблиця відповідає певному класу, а її атрибути - певним полям. Кожний рядок у таблиці це клас зі своїми даними.

Висновки:
Я ознайомився з поняттям UML і діаграмами використання та класів. У даній роботі я створив діаграму використання, яка описує те як різні актори взаємодіють з моєю системою, і описав 3 сценарії використання.

Також створив діаграму класів з класами даних для системи та класами репозиторіїв для роботи з класами даних. Додатково додав діаграму, що описує зв’язки між сутностями в базі даних. Вивчене дозволяє приступити до подальших етапів проектування системи.