



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 розроблення програмного забезпечення»*  
**Тема: «Основи проектування»**

Виконала:  
Студент групи ІА-31  
Самелюк А.С.

Перевірив:  
Мягкий М.Ю.

**Тема:** Основи проектування.

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

**Посилання на репозиторій з проєктом:**

[https://github.com/Sameliuk/OnlineDiary\\_trpz/tree/main](https://github.com/Sameliuk/OnlineDiary_trpz/tree/main)

### **Хід роботи**

1. Ознайомитись з короткими теоретичними відомостями.
2. Проаналізувати тему та спроектувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.

Онлайн-щоденник — це веб-застосунок, який дозволяє користувачу створювати, редагувати та організовувати особисті записи.

Основні функції:

- створення, редагування, видалення нотаток;
- додавання тегів до відповідних записів;
- сортування записів за тегами чи датами;
- пошук за ключовим словом;
- email-нагадування про важливі записи.

**Актори:**

- Користувач
- Система сповіщень

**Варіанти використання:**

#### **1. Управління обліковим записом**

- Реєстрація користувача — створення нового облікового запису.
- Авторизація користувача — вхід до системи за допомогою логіна та пароля.
- Вихід із системи — завершення роботи користувача.

#### **2. Робота з нотатками**

- Створення нотатки — користувач створює новий запис.

- Редагування нотатки – внесення змін у вже існуючу нотатку.
- Видалення нотатки – повне видалення запису з бази даних.
- Перегляд нотаток – відображення списку всіх нотаток із можливістю фільтрації.

### **3. Організація записів**

- Додавання тегів – створення тегів для нотаток.
- Прив'язка тегів до нотаток – організація записів за категоріями.
- Сортування за тегами та датою – швидка навігація між нотатками.

### **4. Пошук інформації**

- Пошук за ключовим словом заголовку – знаходження нотаток у заголовку чи тексті.

### **5. Нагадування та сповіщення**

- Створення нагадування – встановлення дати й часу сповіщення для нотатки.
- Отримання нагадування на email – система автоматично надсилає повідомлення через NotificationService.
- Скасування нагадування – користувач може видалити або деактивувати нагадування.

### **Зв'язки:**

- Користувач → Реєстрація
- Користувач → Авторизація
- Користувач → Вихід із системи
- Користувач → Створення нотатки
- Користувач → Редагування нотатки
- Користувач → Видалення нотатки
- Користувач → Перегляд нотаток
- Користувач → Додавання тегів
- Користувач → Прив'язка тегів до нотаток
- Користувач → Сортування за тегами та датою
- Користувач → Пошук за ключовим словом

- Користувач → Пошук за тегом
- Користувач → Створення нагадування
- Користувач → Скасування нагадування
- Система → Отримання нагадування на email ← Email-сервіс.

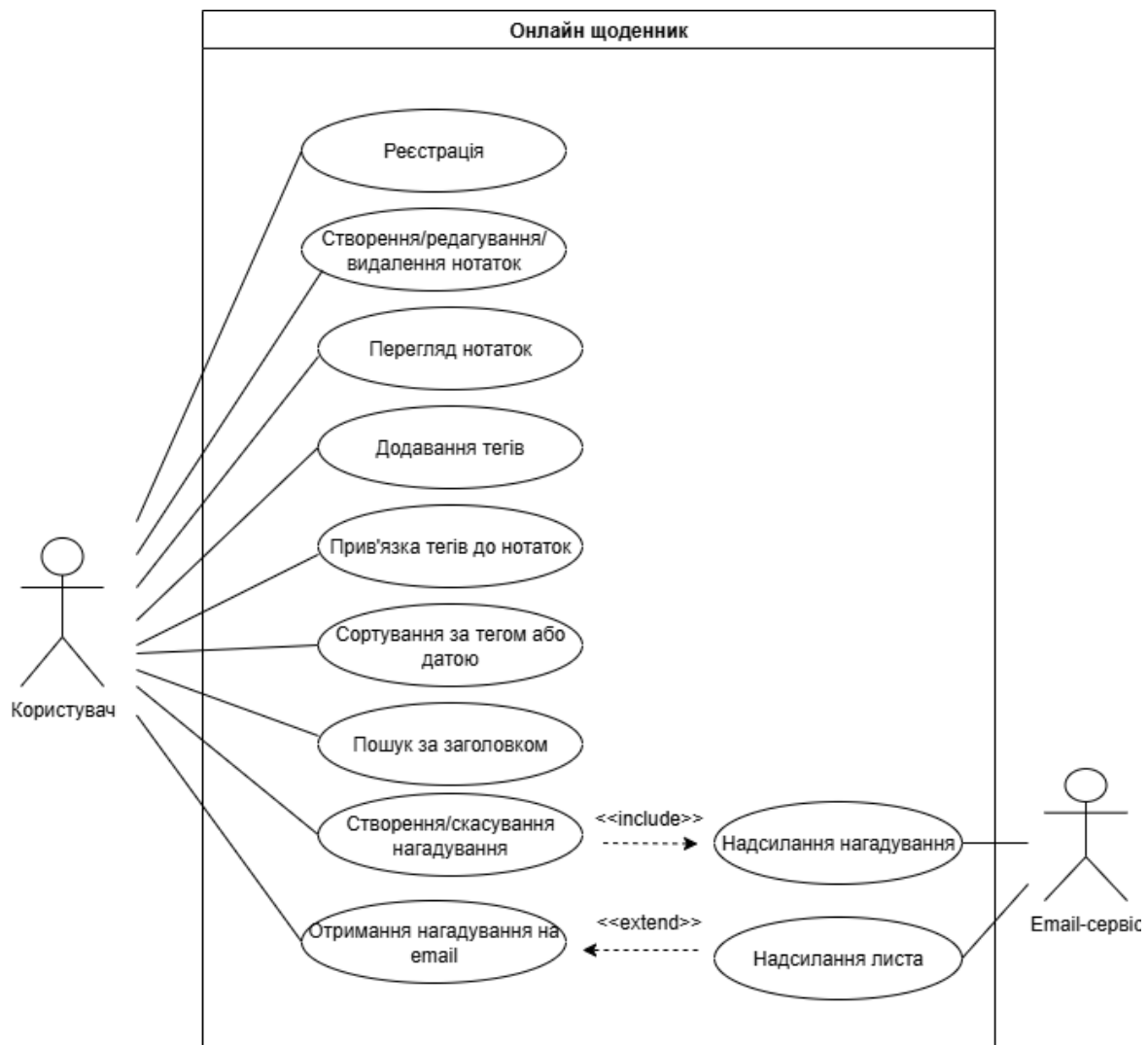


Рис.1 - Діаграму варіантів використання

3. Спроектувати діаграму класів предметної області.

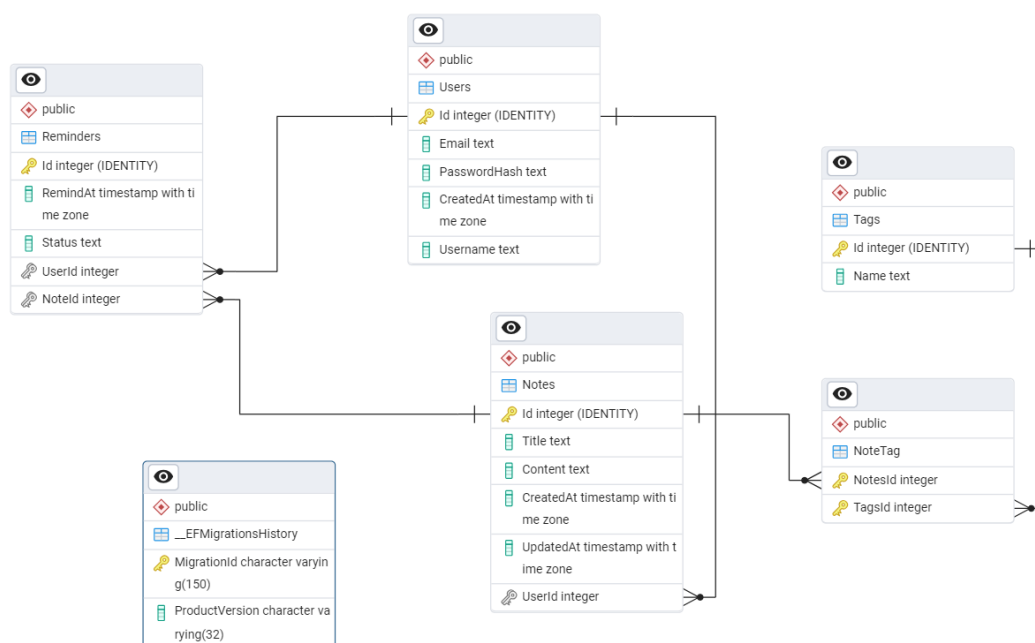
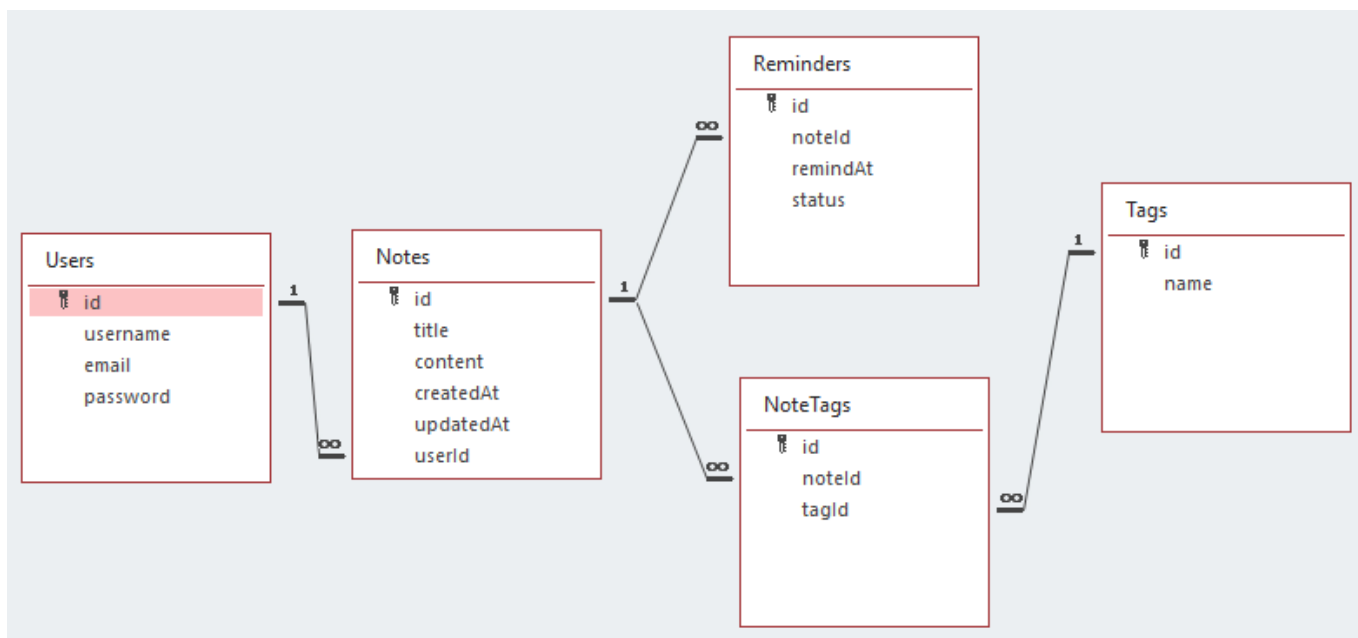


Рис.2 – Структура бази даних(MS Access та PostgreSQL)

Users		
	Имя поля	Тип данных
	id	Счетчик
	username	Короткий текст
	email	Короткий текст
	password	Короткий текст

Рис.3 – Опис таблиці Користувачі

Notes		
	Имя поля	Тип данных
id		Счетчик
title		Короткий текст
content		Длинный текст
createdAt		Дата и время
updatedAt		Дата и время
userId		Числовой

Рис.4 – Опис таблиці Нотатки

Tags		
	Имя поля	Тип данных
id		Счетчик
name		Короткий текст

Рис.5 – Опис таблиці Теги

NoteTags		
	Имя поля	Тип данных
id		Счетчик
noteId		Числовой
tagId		Числовой

Рис.6– Опис таблиці зв'язку Теги нотаток

Reminders		
	Имя поля	Тип данных
id		Счетчик
noteId		Числовой
remindAt		Дата и время
status		Короткий текст

Рис.7 – Опис таблиці зв'язку Нагадування

4. Вибрати 3 варіанти використання та написати за ними сценарії використання.

#### Сценарій 1 – Створення нотатки

Передумови: Користувач авторизований у системі.

Постумови: У разі успішного виконання нова нотатка збережена у базі даних.

Взаємодіючі сторони: Користувач, Система.

Короткий опис: Варіант використання дозволяє користувачу створювати нові нотатки в особистому щоденнику.

Основний потік подій:

1. Користувач відкриває форму створення нотатки.
2. Система відображає форму для введення даних.
3. Користувач вводить заголовок і текст нотатки.
4. Користувач додає теги (за потреби).
5. Користувач натискає кнопку «Зберегти».
6. Система перевіряє коректність введених даних.
7. Система зберігає нотатку у базі даних.

Винятки:

- Виняток №1: Обов'язкові поля (наприклад, заголовок) залишені порожніми. Система повідомляє користувача про помилку та пропонує виправити дані.

Примітки: Відсутні.

## **Сценарій 2 – Пошук нотатки за ключовим словом**

Передумови: У базі даних існують збережені нотатки.

Постумови: Користувач отримує список нотаток, що відповідають критерію пошуку.

Взаємодіючі сторони: Користувач, Система.

Короткий опис: Варіант використання дозволяє швидко знайти потрібну нотатку за введеним ключовим словом.

Основний потік подій:

1. Користувач вводить ключове слово у пошукове поле.
2. Користувач запускає пошук.
3. Система виконує пошук збігів у заголовках та тексті нотаток.
4. Система формує список результатів.
5. Система відображає список знайдених нотаток користувачу.

Винятки:

- Виняток №1: За заданим ключовим словом збігів не знайдено. Система повідомляє користувача, що результати відсутні.

Примітки: Можливе подальше уточнення пошуку за тегами чи датою.

### **Сценарій 3 – Нагадування про подію**

Передумови: Користувач має створену нотатку.

Постумови: У заданий час користувач отримує email-нагадування.

Взаємодіючі сторони: Користувач, Система, Email-сервіс.

Короткий опис: Варіант використання дозволяє користувачу встановити нагадування для нотатки та отримати повідомлення на email у потрібний час.

Основний потік подій:

1. Користувач відкриває нотатку та встановлює дату і час нагадування.
2. Система зберігає дані про нагадування у базі даних.
3. Коли настає вказаний час, система активує процес надсилення.
4. Система звертається до Email-сервісу для відправки повідомлення.
5. Email-сервіс надсилає повідомлення на пошту користувача.

Винятки:

- Виняток №1: Email-сервіс недоступний. Система реєструє помилку, може повторити спробу надсилення пізніше.

Примітки: Повідомлення може містити посилання для швидкого переходу до відповідної нотатки.

5. На основі спроектованої діаграми класів предметної області розробити основні класи та структуру бази даних системи. Класи даних повинні реалізувати шаблон Repository для взаємодії з базою даних.

Код реалізованих класів:



```

Note.cs  x  NoteRepository.cs  ReminderController.cs  202509171643...TimezoneV2.cs  2025
OnlineDiaryApp  OnlineDiaryApp.Models.Note
1  namespace OnlineDiaryApp.Models
2  {
3      42 references
4      public class Note
5      {
6          9 references
7          public int Id { get; set; }
8          9 references
9          public string Title { get; set; } = null!;
10         6 references
11         public string Content { get; set; } = null!;
12         5 references
13         public DateTime CreatedAt { get; set; }
14         1 reference
15         public DateTime UpdatedAt { get; set; }
16
17         5 references
18         public int UserId { get; set; }
19         0 references
20         public User User { get; set; }
21
22         12 references
23         public ICollection<Tag> Tags { get; set; } = new List<Tag>();
24         0 references
25         public Reminder? Reminder { get; set; }
26     }
27 }

```

Код 1 – Note.cs (Models)

```

Reminder.cs  x  Note.cs  NoteRepository.cs  ReminderController.cs
OnlineDiaryApp  OnlineDiaryApp.Models
1  namespace OnlineDiaryApp.Models
2  {
3      18 references
4      public class Reminder
5      {
6          2 references
7          public int Id { get; set; }
8          3 references
9          public DateTime RemindAt { get; set; }
10         3 references
11         public string Status { get; set; } = "active";
12
13         1 reference
14         public int UserId { get; set; }
15         4 references
16         public User User { get; set; }
17
18         2 references
19         public int NoteId { get; set; }
20         5 references
21         public Note Note { get; set; }
22     }
23 }

```

Код 2 –Reminder.cs (Models)

```
Tag.cs  Reminder.cs  Note.cs  NoteRepository.cs  ReminderController.cs  Privacy.cshtml
OnlineDiaryApp
namespace OnlineDiaryApp.Models
{
    19 references
    public class Tag
    {
        2 references
        public int Id { get; set; }
        5 references
        public string Name { get; set; } = null!;

        2 references
        public ICollection<Note> Notes { get; set; } = new List<Note>();
    }
}
```

Код 3 –Tag.cs (Models)

```
User.cs  Tag.cs  Reminder.cs  Note.cs  NoteRepository.cs  ReminderController.cs  Privacy.cshtml
OnlineDiaryApp
namespace OnlineDiaryApp.Models
{
    17 references
    public class User
    {
        2 references
        public int Id { get; set; }
        6 references
        public string Email { get; set; } = null!;

        6 references
        public string Username { get; set; } = null!;
        2 references
        public string PasswordHash { get; set; } = null!;
        1 reference
        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        0 references
        public ICollection<Note> Notes { get; set; } = new List<Note>();
        0 references
        public ICollection<Reminder> Reminders { get; set; } = new List<Reminder>();
    }
}
```

Код 4 –User.cs (Models)

```
INoteRepository.cs  AppDbContext.cs  NoteRepository.cs  User.cs
OnlineDiaryApp
{
    1  using OnlineDiaryApp.Models;
    2
    3  namespace OnlineDiaryApp.Repositories.Interfaces
    4  {
    5      4 references
    6      public interface INoteRepository
    7      {
    8          4 references
    9          Task<IEnumerable<Note>> GetAllAsync();
    10         2 references
    11         Task<Note?> GetByIdAsync(int id);
    12         2 references
    13         Task AddAsync(Note note);
    14         2 references
    15         Task UpdateAsync(Note note);
    16         2 references
    17         Task DeleteAsync(int id);
    18         4 references
    19         Task SaveChangesAsync();
    20     }
    21 }
```

Код 5 – INoteRepository.cs (Repositories.Interfaces)

```
IReminderRepository.cs  INoteRepository.cs  AppDbContext.cs  Note
OnlineDiaryApp
{
    1  using OnlineDiaryApp.Models;
    2
    3      4 references
    4      public interface IReminderRepository
    5      {
    6          2 references
    7          Task<IEnumerable<Reminder>> GetAllAsync();
    8          2 references
    9          Task<Reminder?> GetByIdAsync(int id);
    10         2 references
    11         Task<Reminder?> GetByNoteIdAsync(int noteId);
    12         2 references
    13         Task AddAsync(Reminder reminder);
    14         2 references
    15         Task UpdateAsync(Reminder reminder);
    16         2 references
    17         Task DeleteAsync(int id);
    18         4 references
    19         Task SaveChangesAsync();
    20     }
    21 }
```

Код 6 – IReminderRepository.cs (Repositories.Interfaces)

```
ITagRepository.cs*  ReminderRepository.cs*  UserRepository.cs*
OnlineDiaryApp
1  using OnlineDiaryApp.Models;
2
3  namespace OnlineDiaryApp.Repositories.Interfaces
4  {
5      public interface ITagRepository
6      {
7          Task<IEnumerable<Tag>> GetAllAsync();
8          Task<Tag?> GetByIdAsync(int id);
9          Task AddAsync(Tag tag);
10         Task DeleteAsync(int id);
11         Task SaveChangesAsync();
12     }
13 }
```

Код 7 – ITagRepository.cs (Repositories.Interfaces)

```
IUserRepository.cs  ITagRepository.cs  IReminderRepository.cs  INoteRepository.cs
OnlineDiaryApp
1  using OnlineDiaryApp.Models;
2  using System.Threading.Tasks;
3
4  namespace OnlineDiaryApp.Repositories.Interfaces
5  {
6      public interface IUserRepository
7      {
8          Task<User?> GetByEmailAsync(string email);
9          Task<User?> GetByUsernameAsync(string username);
10         Task AddAsync(User user);
11     }
12 }
13
```

Код 8 – IUserRepository.cs (Repositories.Interfaces)

```
NoteRepository.cs x | UserRepository.cs | TagRepository.cs | ReminderRepository.cs | NoteRepository.cs | AppDbContext.cs
OnlineDiaryApp | OnlineDiaryApp.Repositories.Implementation.NoteRepository | UpdateAs
1 using Microsoft.EntityFrameworkCore;
2 using OnlineDiaryApp.Data;
3 using OnlineDiaryApp.Models;
4 using OnlineDiaryApp.Repositories.Interfaces;
5
6 namespace OnlineDiaryApp.Repositories.Implementation
7 {
8     2 references
9     public class NoteRepository : INoteRepository
10     {
11         private readonly AppDbContext _context;
12
13         0 references
14         public NoteRepository(AppDbContext context)
15         {
16             _context = context;
17         }
18
19         4 references
20         public async Task<IEnumerable<Note>> GetAllAsync()
21         {
22             return await _context.Notes.Include(n => n.Tags).ToListAsync();
23         }
24
25         2 references
26         public async Task<Note?> GetByIdAsync(int id)
27         {
28             return await _context.Notes.Include(n => n.Tags).FirstOrDefaultAsync(n => n.Id == id);
29         }
30
31         2 references
32         public async Task AddAsync(Note note)
33         {
34             await _context.Notes.AddAsync(note);
35         }
36     }
37 }
```

```
31
32     2 references
33     public async Task UpdateAsync(Note note)
34     {
35         _context.Notes.Update(note);
36     }
37
38     2 references
39     public async Task DeleteAsync(int id)
40     {
41         var note = await _context.Notes.FindAsync(id);
42         if (note != null)
43             _context.Notes.Remove(note);
44     }
45
46     4 references
47     public async Task SaveChangesAsync()
48     {
49         await _context.SaveChangesAsync();
50     }
51 }
```

Код 9 – NoteRepository.cs (Repositories.Implementations)

```

ReminderRepository.cs*  UserRepository.cs*  TagRepository.cs*  NoteRepository.cs
OnlineDiaryApp  ReminderRepository
1  using Microsoft.EntityFrameworkCore;
2  using OnlineDiaryApp.Data;
3  using OnlineDiaryApp.Models;
4  public class ReminderRepository : IReminderRepository
5  {
6      private readonly AppDbContext _context;
7
8      public ReminderRepository(AppDbContext context)
9      {
10         _context = context;
11     }
12
13     public async Task<IEnumerable<Reminder>> GetAllAsync()
14     {
15         return await _context.Reminders
16             .Include(r => r.User)
17             .Include(r => r.Note)
18             .ToListAsync();
19     }
20
21     public async Task<Reminder?> GetByIdAsync(int id)
22     {
23         return await _context.Reminders
24             .Include(r => r.User)
25             .Include(r => r.Note)
26             .FirstOrDefaultAsync(r => r.Id == id);
27     }

```

```

32     public async Task<Reminder?> GetByNoteIdAsync(int noteId)
33     {
34         return await _context.Reminders
35             .FirstOrDefaultAsync(r => r.NoteId == noteId);
36     }
37
38     public async Task AddAsync(Reminder reminder)
39     {
40         await _context.Reminders.AddAsync(reminder);
41     }
42
43     public async Task UpdateAsync(Reminder reminder)
44     {
45         _context.Reminders.Update(reminder);
46     }
47
48     public async Task DeleteAsync(int id)
49     {
50         var reminder = await _context.Reminders.FindAsync(id);
51         if (reminder != null)
52         {
53             _context.Reminders.Remove(reminder);
54         }
55     }
56
57     public async Task SaveChangesAsync()
58     {
59         await _context.SaveChangesAsync();
60     }
61 }

```

Код 10 – ReminderRepository.cs (Repositories.Implementations)

TagRepository.cs\*    ReminderRepository.cs\*    NoteRepository.cs    IUserRepository.cs

OnlineDiaryApp    OnlineDiaryApp.Repositories

```

1  using Microsoft.EntityFrameworkCore;
2  using OnlineDiaryApp.Data;
3  using OnlineDiaryApp.Models;
4  using OnlineDiaryApp.Repositories.Interfaces;
5  using System;
6
7  namespace OnlineDiaryApp.Repositories.Implementation
8  {
9      2 references
10     public class TagRepository : ITagRepository
11     {
12
13         0 references
14         public TagRepository(AppDbContext context)
15         {
16             _context = context;
17         }
18
19         3 references
20         public async Task<IEnumerable<Tag>> GetAllAsync()
21         {
22             return await _context.Tags.ToListAsync();
23         }
24
25         4 references
26         public async Task<Tag?> GetByIdAsync(int id)
27         {
28             return await _context.Tags.FindAsync(id);
29         }
30
31         2 references
32         public async Task AddAsync(Tag tag)
33         {
34             await _context.Tags.AddAsync(tag);
35         }
36
37         2 references
38         public async Task DeleteAsync(int id)
39         {
40             var tag = await _context.Tags.FindAsync(id);
41             if (tag != null)
42                 _context.Tags.Remove(tag);
43         }
44
45         3 references
46         public async Task SaveChangesAsync()
47         {
48             await _context.SaveChangesAsync();
49         }
50     }

```

Код 11 – TagRepository.cs (Repositories.Implementations)





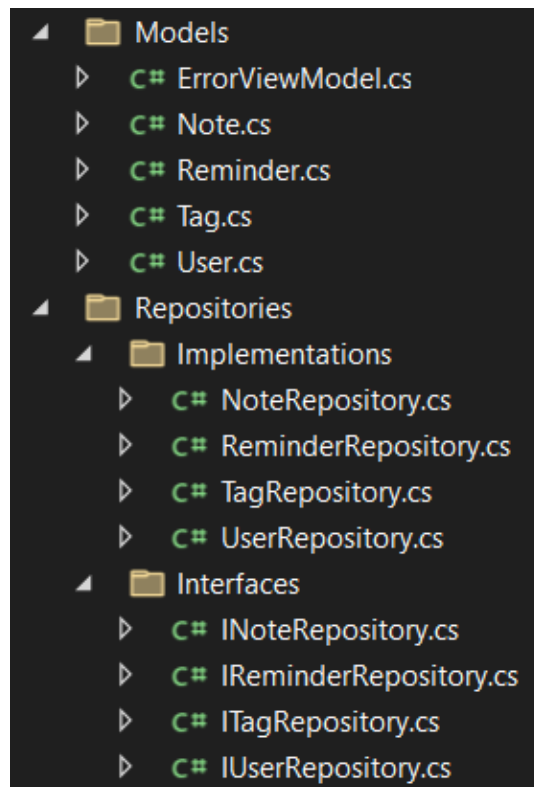


Рис. 9 – Реалізовані класи

7. 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.

**Висновки:** Виконання лабораторної роботи дозволило ознайомитися з основними методами об'єктно-орієнтованого моделювання системи за допомогою UML. Було створено діаграму варіантів використання, яка наочно відображає взаємодію користувачів із системою та ключові функції, що вона надає. За допомогою діаграми класів було показано основні класи системи, їх атрибути та методи, а також зв'язки між класами, що дозволяє зрозуміти структурну організацію програми та логіку взаємодії об'єктів.

#### **Відповіді на контрольні питання:**

1. Що таке UML?

UML (Unified Modeling Language) — це уніфікована мова моделювання, яка використовується для графічного представлення, документування та

проектування програмних систем. Вона допомагає описати структуру, поведінку та взаємодію компонентів системи.

## 2. Що таке діаграма класів UML?

Діаграма класів UML — це статична діаграма, яка показує класи системи, їх атрибути, методи та зв'язки між класами. Вона використовується для моделювання структури об'єктно-орієнтованих програм.

## 3. Які діаграми UML називають канонічними?

Канонічні (основні) діаграми UML — це ті, що найчастіше використовуються для опису системи:

- Діаграма класів
- Діаграма варіантів використання (Use Case)
- Діаграма послідовності (Sequence)
- Діаграма станів (State)
- Діаграма активностей (Activity)

## 4. Що таке діаграма варіантів використання?

Діаграма варіантів використання (Use Case Diagram) показує взаємодію користувачів (акторів) з системою через варіанти використання. Вона відображає, хто і як використовує систему, без деталізації внутрішньої реалізації.

## 5. Що таке варіант використання?

Варіант використання (Use Case) — це послідовність дій, які система виконує для досягнення певної мети користувача. Наприклад, “Реєстрація користувача” або “Надіслати повідомлення”.

## 6. Які відношення можуть бути відображені на діаграмі використання?

На діаграмі варіантів використання можуть бути:

- Association (асоціація) — зв'язок між актором і варіантом використання.

- Include (включення) — один варіант використання завжди виконує інший.
- Extend (розширення) — додатковий варіант, який може виконуватися у певних умовах.
- Generalization (успадкування) — актор або варіант використання наслідує інший.

#### 7. Що таке сценарій?

Сценарій — це конкретна реалізація варіанту використання, тобто послідовність кроків, які відбуваються під час виконання дії користувачем та системою.

#### 8. Що таке діаграма класів?

Діаграма класів — це графічне представлення класів, їх атрибутів і методів, а також зв'язків між класами. Вона описує структуру системи.

#### 9. Які зв'язки між класами ви знаєте?

Основні типи зв'язків:

- Асоціація (Association) — загальний зв'язок між класами.
- Агрегація (Aggregation) — «має»; клас складається з інших, але частини можуть існувати окремо.
- Композиція (Composition) — сильніша агрегація; частини не можуть існувати без цілого.
- Успадкування (Generalization) — один клас наслідує властивості іншого.
- Залежність (Dependency) — один клас використовує інший тимчасово.

#### 10. Чим відрізняється композиція від агрегації?

- Агрегація: частини можуть існувати без цілого.
- Композиція: частини не можуть існувати без цілого; знищення цілого знищує частини.

#### 11. Чим відрізняється агрегація від композиції на діаграмах класів?

- Агрегація позначається порожнім ромбом на стороні цілого.

- Композиція позначається заповненим ромбом на стороні цілого.

## 12. Що являють собою нормальні форми баз даних?

Нормальні форми — це правила організації таблиць БД для уникнення надлишковості та аномалій при оновленні даних. Основні:

- 1НФ — всі атрибути атомарні.
- 2НФ — всі неключові атрибути залежать від усього первинного ключа.
- 3НФ — немає транзитивних залежностей між неключовими атрибутами.

## 13. Що таке фізична і логічна модель БД?

- Логічна модель — структура даних з таблицями, полями та зв'язками, незалежно від СУБД.
- Фізична модель — конкретна реалізація БД у СУБД, включає типи даних, індекси, обмеження, фізичне зберігання.

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

Кожна таблиця БД часто відповідає класу у програмі, а рядки таблиці — об'єктам класу. Поля таблиці відображаються як атрибути класу, а зв'язки між таблицями як зв'язки між класами.