



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 №8
із дисципліни «*Технології розроблення програмного забезпечення*»
Тема: «Патерни проектування»

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

Перевірив:
Мякий М.Ю.

Тема: Патерни проектування.

Тема проєкту: Онлайн-щоденник (strategy, adapter, observer, facade, composite, client-server). Веб-застосунок дозволяє користувачу створювати, редагувати та організовувати особисті записи з можливістю пошуку, сортування та додавання тегів, а також отримання email-нагадувань.

Мета: Вивчити структуру шаблонів «Composite», «Flyweight» (Пристосуванець), «Interpreter», «Visitor» та навчитися застосовувати їх в реалізації програмної системи.

Посилання на репозиторій з проєктом та звітами:

https://github.com/Sameliuk/OnlineDiary_trpz/tree/main

https://github.com/Sameliuk/OnlineDiary_trpz/tree/reports

Хід роботи

1. Ознайомитись з короткими теоретичними відомостями.

Шаблон Composite дозволяє створювати деревоподібні структури «частина—ціле», уніфіковано обробляючи як окремі об'єкти, так і складені з вкладеними елементами. Композит містить колекцію дочірніх об'єктів і виконує операції рекурсивно. Застосовується для графічних форм, ієрархій завдань у проєктах тощо. Переваги: простота роботи з ієрархіями, гнучкість, легке додавання/видалення елементів. Недоліки: складність початкової реалізації, потребує добре спроектованого інтерфейсу.

Шаблон Flyweight використовується для зменшення кількості об'єктів через спільне використання. Важливо розрізняти внутрішній стан (дані об'єкта) і зовнішній стан (контекст використання). Підходить для графічних примітивів, текстових символів та інших однакових об'єктів. Переваги: економія пам'яті. Недоліки: додаткові обчислення контексту, ускладнення коду через введення додаткових класів.

Шаблон Interpreter забезпечує подання граматики та інтерпретацію мови через абстрактне синтаксичне дерево. Кожен вираз (термінальний або нетермінальний) інтерпретується з контекстом, дочірні вирази обробляються рекурсивно. Використовується для пошуку рядків, скриптових мов, простих граматик. Переваги: легко розширювати граматику і змінювати обчислення виразів. Недоліки: складність супроводу великої граматики.

Шаблон Visitor дозволяє визначати операції над елементами без зміни їх структури, розділяючи дані та логіку. Клас відвідувача реалізує операції для кожного типу елемента. Підходить для онлайн-корзин, компіляторів та інших ієрархій об'єктів, де потрібно додавати нові операції без зміни елементів. Переваги: легко додавати нові операції, підтримка різних алгоритмів роботи з об'єктами. Недоліки: складно додавати нові елементи в ієрархію, необхідно оновлювати всіх відвідувачів.

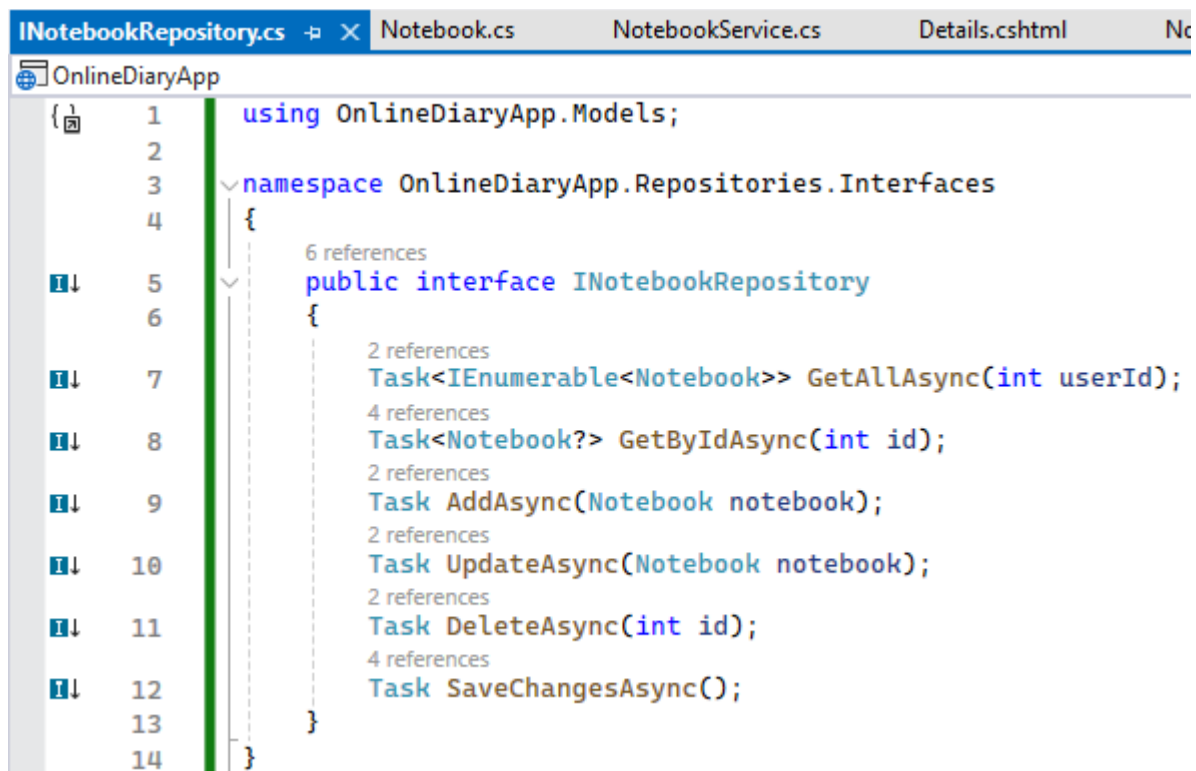
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

В даній частині було реалізовану логіку з блокнотами та групуванням нотаток у блокноти. Було реалізовано відповідні сервіси, контролери та інтерфейси для виконання певного функціоналу.

```
1 namespace OnlineDiaryApp.Models
2 {
3     31 references
4     public class Notebook
5     {
6         9 references
7         public int Id { get; set; }
8         14 references
9         public string Name { get; set; } = null!;
10        7 references
11        public string Description { get; set; }
12        2 references
13        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
14        1 reference
15        public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;
16
17        5 references
18        public int UserId { get; set; }
19        1 reference
20        public User User { get; set; }
21
22        3 references
23        public ICollection<Note> Notes { get; set; } = new List<Note>();
24    }
25 }
```

Рис.1 – Notebook (Model)

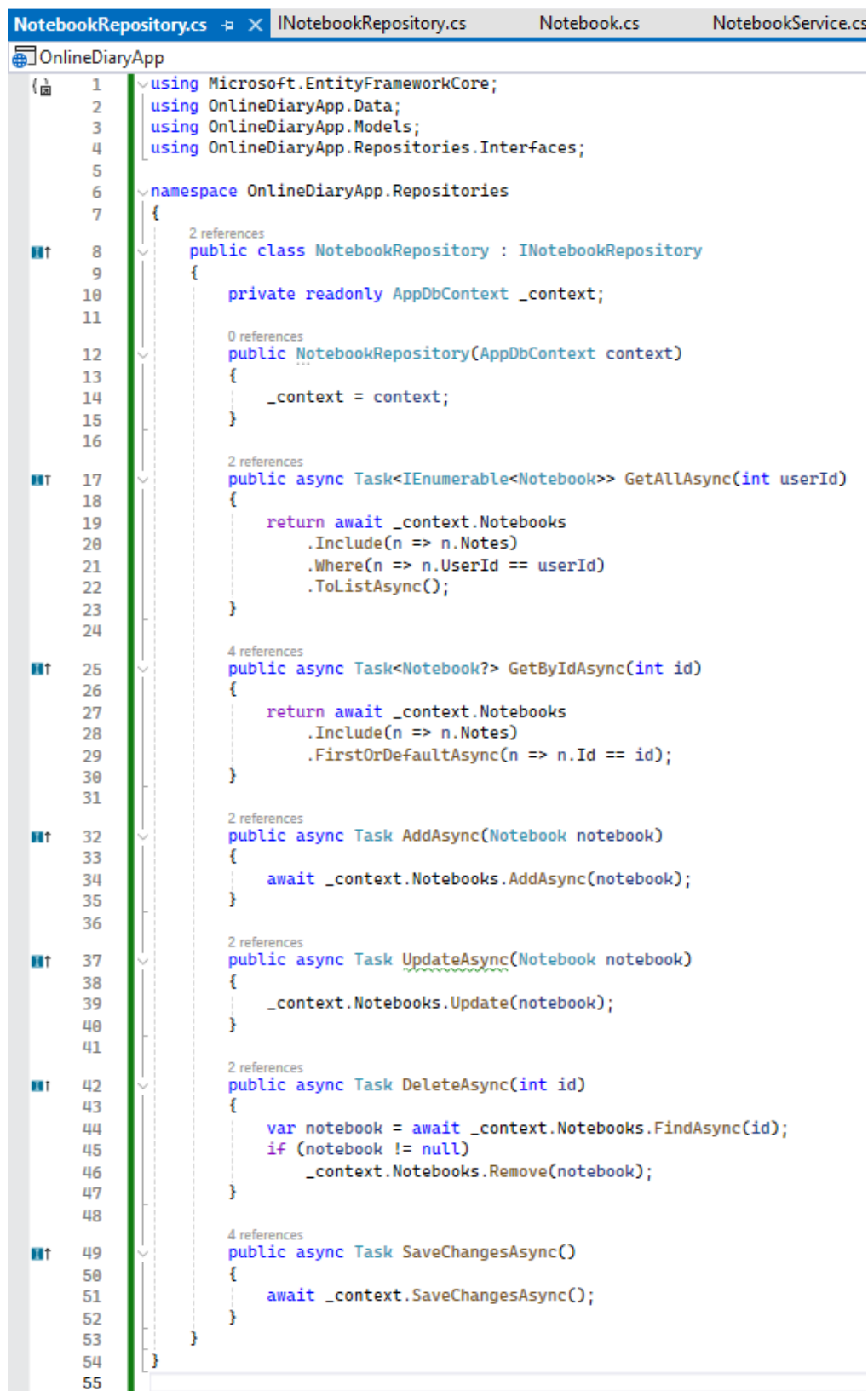
Notebook представляє блокнот користувача, який містить нотатки, та служить для організації та структуризації інформації в системі онлайн-щоденника. Вона інкапсулює дані про блокнот і забезпечує зв'язки з іншими сутностями, такими як користувач, нотатки, теги чи нагадування.



```
INotebookRepository.cs Notebook.cs NotebookService.cs Details.cshtml Nc
OnlineDiaryApp
1 using OnlineDiaryApp.Models;
2
3 namespace OnlineDiaryApp.Repositories.Interfaces
4 {
5     public interface INotebookRepository
6     {
7         Task<IEnumerable<Notebook>> GetAllAsync(int userId);
8         Task<Notebook?> GetByIdAsync(int id);
9         Task AddAsync(Notebook notebook);
10        Task UpdateAsync(Notebook notebook);
11        Task DeleteAsync(int id);
12        Task SaveChangesAsync();
13    }
14 }
```

Рис.2 – INotebookRepository (Repository.Interfaces)

INotebookRepository визначає контракт для роботи з блокнотами на рівні доступу до даних. Він інкапсулює операції з базою даних або іншими джерелами зберігання, дозволяючи контролерам та сервісам працювати з блокнотами, не знаючи деталей реалізації збереження даних.



```
1 using Microsoft.EntityFrameworkCore;
2 using OnlineDiaryApp.Data;
3 using OnlineDiaryApp.Models;
4 using OnlineDiaryApp.Repositories.Interfaces;
5
6 namespace OnlineDiaryApp.Repositories
7 {
8     public class NotebookRepository : INotebookRepository
9     {
10         private readonly AppDbContext _context;
11
12         public NotebookRepository(AppDbContext context)
13         {
14             _context = context;
15         }
16
17         public async Task<IEnumerable<Notebook>> GetAllAsync(int userId)
18         {
19             return await _context.Notebooks
20                 .Include(n => n.Notes)
21                 .Where(n => n.UserId == userId)
22                 .ToListAsync();
23         }
24
25         public async Task<Notebook?> GetByIdAsync(int id)
26         {
27             return await _context.Notebooks
28                 .Include(n => n.Notes)
29                 .FirstOrDefaultAsync(n => n.Id == id);
30         }
31
32         public async Task AddAsync(Notebook notebook)
33         {
34             await _context.Notebooks.AddAsync(notebook);
35         }
36
37         public async Task UpdateAsync(Notebook notebook)
38         {
39             _context.Notebooks.Update(notebook);
40         }
41
42         public async Task DeleteAsync(int id)
43         {
44             var notebook = await _context.Notebooks.FindAsync(id);
45             if (notebook != null)
46                 _context.Notebooks.Remove(notebook);
47         }
48
49         public async Task SaveChangesAsync()
50         {
51             await _context.SaveChangesAsync();
52         }
53     }
54 }
55
```

Рис.3 – NotebookRepository (Repository.Implementation)

NotebookRepository є конкретною реалізацією інтерфейсу INotebookRepository і відповідає за безпосередню роботу з базою даних для збереження, отримання,

оновлення та видалення блокнотів. Він інкапсулює всі деталі доступу до БД, використовуючи ORM (наприклад, Entity Framework), і надає стандартизовані методи для сервісів та контролерів.

```
1 using OnlineDiaryApp.Models;
2 using OnlineDiaryApp.Repositories.Interfaces;
3
4 namespace OnlineDiaryApp.Services
5 {
6     8 references
7     public class NotebookService
8     {
9         private readonly INotebookRepository _notebookRepository;
10
11         0 references
12         public NotebookService(INotebookRepository notebookRepository)
13         {
14             _notebookRepository = notebookRepository;
15         }
16
17         3 references
18         public async Task<IEnumerable<Notebook>> GetAllNotebooksAsync(int userId)
19         {
20             return await _notebookRepository.GetAllAsync(userId);
21         }
22
23         5 references
24         public async Task<Notebook?> GetNotebookByIdAsync(int id)
25         {
26             return await _notebookRepository.GetByIdAsync(id);
27         }
28
29         1 reference
30         public async Task<Notebook> CreateNotebookAsync(string name, int userId, string? description = null)
31         {
32             var notebook = new Notebook
33             {
34                 Name = name,
35                 Description = description,
36                 UserId = userId,
37                 CreatedAt = DateTime.UtcNow
38             };
39
40             await _notebookRepository.AddAsync(notebook);
41             await _notebookRepository.SaveChangesAsync();
42             return notebook;
43         }
44
45         1 reference
46         public async Task UpdateNotebookAsync(int id, string name, string? description)
47         {
48             var notebook = await _notebookRepository.GetByIdAsync(id);
49             if (notebook == null) return;
50
51             notebook.Name = name;
52             notebook.Description = description;
53             notebook.UpdatedAt = DateTime.UtcNow;
54
55             await _notebookRepository.UpdateAsync(notebook);
56             await _notebookRepository.SaveChangesAsync();
57         }
58
59         1 reference
60         public async Task DeleteNotebookAsync(int id)
61         {
62             await _notebookRepository.DeleteAsync(id);
63             await _notebookRepository.SaveChangesAsync();
64         }
65     }
66 }
```

Рис.4 – NotebookService (Services)

NotebookService є сервісним класом, який реалізує бізнес-логіку для роботи з блокнотами користувачів. Він інкапсулює взаємодію з репозиторієм NotebookRepository, забезпечуючи чистий та зручний інтерфейс для контролерів. Сервіс відповідає за перевірку даних, додаткову обробку та координацію операцій із блокнотами та, за потреби, із нотатками, що входять у блокнот.

3. Реалізувати один з розглянутих шаблонів за обраною темою.

Для даного проєкту було обрано патерн Composite. Шаблон реалізований таким чином:

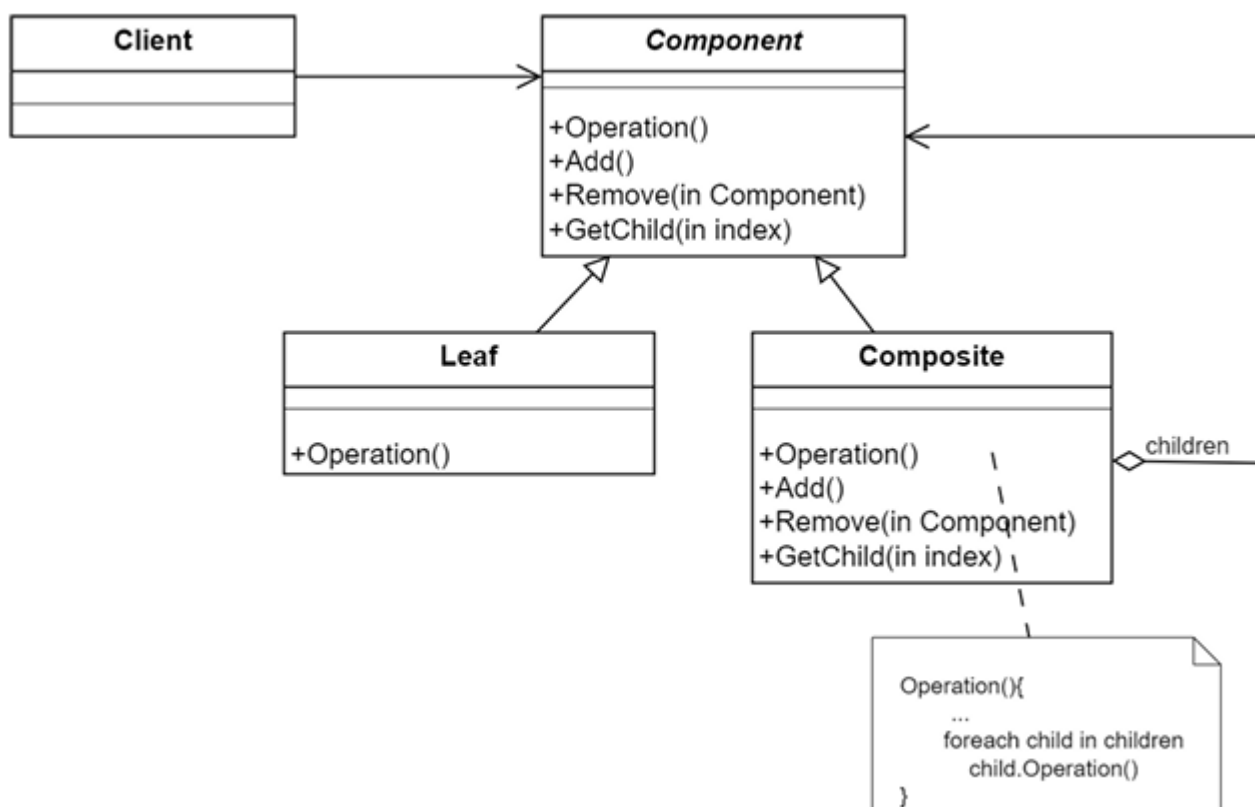


Рис.1 – Структура патерну Компонувальник

Component: INotebookComponent - це базовий інтерфейс для всіх компонентів, які можуть бути частиною структури блокнот–нотатка. Він визначає спільні операції для всіх елементів: Add(INotebookComponent component) — додає дочірній елемент (для Composite); Remove(INotebookComponent component) — видаляє дочірній елемент; GetChildren() — повертає список дочірніх елементів;

Leaf-елементи (нотатки) реалізують цей інтерфейс, але методи Add/Remove у них можуть бути пустими або кидати виключення, бо листи не містять дочірніх елементів.

Leaf: NoteLeaf представляє кінцевий елемент структури — окрему нотатку. Він містить посилання на модель Note. Реалізує інтерфейс INotebookComponent, але GetChildren() повертає порожню колекцію, бо Leaf-елемент не може містити інші елементи.

Composite: NotebookComposite представляє складовий елемент — блокнот, який може містити нотатки (NoteLeaf). Містить список дочірніх компонентів (_children). Реалізує інтерфейс INotebookComponent, дозволяючи додавати, видаляти та отримувати дочірні елементи. Через Composite можна обійтися одним інтерфейсом для роботи як з окремими нотатками, так і з цілими блокнотами.

Client: NotebookController / Views. Контролер створює NotebookComposite, додає в нього всі нотатки користувача як NoteLeaf. Представлення (View) використовує Composite для відображення всіх нотаток блокнота у вигляді карток. Завдяки патерну Composite контролер не потребує окремої логіки для роботи з блокнотами або нотатками — всі елементи обробляються через один інтерфейс.

Переваги реалізації:

1. Уніфікація роботи з елементами — контролер і View працюють з INotebookComponent, не знаючи, це блокнот чи нотатка.
2. Гнучкість — легко додати нові типи елементів, наприклад підблокноти, не змінюючи існуючий код клієнта.
3. Просте відображення — Composite дозволяє зручно рендерити структуру в Razor View як ієрархію елементів.
4. Реалізувати не менше 3-х класів відповідно до обраної теми.

```

1 namespace OnlineDiaryApp.Composite
2 {
3     11 references
4     public interface INotebookComponent
5     {
6         3 references
7         void Display(int depth);
8         2 references
9         IEnumerable<INotebookComponent> GetChildren();
10 }
11 }

```

Код 5 – INotebookComponent (Composite)

```

1 using OnlineDiaryApp.Models;
2
3 namespace OnlineDiaryApp.Composite
4 {
5     4 references
6     public class NotebookComposite : INotebookComponent
7     {
8         private readonly List<INotebookComponent> _children = new List<INotebookComponent>();
9
10        4 references
11        public Notebook Notebook { get; }
12
13        3 references
14        public NotebookComposite(Notebook notebook)
15        {
16            Notebook = notebook;
17        }
18
19        3 references
20        public void Add(INotebookComponent component)
21        {
22            _children.Add(component);
23        }
24
25        0 references
26        public void Remove(INotebookComponent component)
27        {
28            _children.Remove(component);
29        }
30
31        1 reference
32        public IEnumerable<INotebookComponent> GetChildren()
33        {
34            return _children;
35        }
36
37        2 references
38        public void Display(int depth)
39        {
40            System.Console.WriteLine(new string('-', depth) + Notebook.Name);
41            foreach (var child in _children)
42            {
43                child.Display(depth + 2);
44            }
45        }
46
47        0 references
48        public string Name => Notebook.Name;
49
50        0 references
51        public int Id => Notebook.Id;
52    }
53 }

```

Код 6 – NotebookComposite (Composite)

NotebookController.cs*	NoteLeaf.cs*	NotebookComposite.cs*	INotebookComponent.cs
------------------------	--------------	-----------------------	-----------------------

```

OnlineDiaryApp
1  using Microsoft.AspNetCore.Mvc;
2  using OnlineDiaryApp.Services;
3  using OnlineDiaryApp.Composite;
4
5
6  namespace OnlineDiaryApp.Controllers
7  {
8      1 reference
9      public class NotebookController : Controller
10     {
11         private readonly NotebookService _notebookService;
12         private readonly NoteService _noteService;
13
14         0 references
15         public NotebookController(NotebookService notebookService, NoteService noteService)
16         {
17             _notebookService = notebookService;
18             _noteService = noteService;
19         }
20
21         0 references
22         public async Task<IActionResult> Index()
23         {
24             var userIdString = HttpContext.Session.GetString("UserId");
25             if (!int.TryParse(userIdString, out int userId))
26                 return RedirectToAction("Login", "User");
27
28             var notebooks = await _notebookService.GetAllNotebooksAsync(userId);
29             return View(notebooks);
30         }
31
32         [HttpGet]
33         0 references
34         public IActionResult Create()
35         {
36             return View();
37         }
38
39         [HttpPost]
40         0 references
41         public async Task<IActionResult> Create(string name, string? description)
42         {
43             var userIdString = HttpContext.Session.GetString("UserId");
44             if (!int.TryParse(userIdString, out int userId))
45                 return RedirectToAction("Login", "User");
46
47             await _notebookService.CreateNotebookAsync(name, userId, description);
48             return RedirectToAction("Index");
49         }
50
51         [HttpGet]
52         0 references
53         public async Task<IActionResult> Edit(int id)
54         {
55             var notebook = await _notebookService.GetNotebookByIdAsync(id);
56             if (notebook == null) return NotFound();
57             return View(notebook);
58         }
59
60         [HttpPost]
61         0 references
62         public async Task<IActionResult> Edit(int id, string name, string? description)
63         {
64             await _notebookService.UpdateNotebookAsync(id, name, description);
65             return RedirectToAction("Index");
66         }
67
68         0 references
69         public async Task<IActionResult> Delete(int id)
70         {
71             await _notebookService.DeleteNotebookAsync(id);
72             return RedirectToAction("Index");
73         }
74
75         0 references
76         public async Task<IActionResult> Notes(int notebookId)
77         {
78             var notebook = await _notebookService.GetNotebookByIdAsync(notebookId);
79             if (notebook == null) return NotFound();
80
81             var notes = await _noteService.GetNotesByNotebookAsync(notebook.Id);
82             ViewBag.Notebook = notebook;
83             return View(notes);
84         }
85     }

```

Код 7.1 – NotebookController (Controllers)

```

76
77 public async Task<IActionResult> Details(int id, string? sortBy, string? tag)
78 {
79     var notebook = await _notebookService.GetNotebookByIdAsync(id);
80     if (notebook == null) return NotFound();
81
82     var notes = await _noteService.GetNotesByNotebookAsync(id);
83
84     if (!string.IsNullOrEmpty(tag))
85         notes = notes.Where(n => n.Tags.Any(t => t.Name == tag)).ToList();
86
87     ISortStrategy? strategy = sortBy?.ToLower() switch
88     {
89         "date" => new SortByDateStrategy(),
90         "title" => new SortByTitleStrategy(),
91         "tag" when !string.IsNullOrEmpty(tag) => new SortByTagStrategy(tag),
92         _ => null
93     };
94
95     if (strategy != null)
96         notes = strategy.Sort(notes).ToList();
97
98     var notebookComposite = new NotebookComposite(notebook);
99     foreach (var note in notes)
100         notebookComposite.Add(new NoteLeaf(note));
101
102     return View(notebookComposite);
103 }
104
105
106
107
108

```

Код 7.2 – NotebookController (Controllers)

```

NoteLeaf.cs* NotebookComposite.cs* INotebookComponent.cs Index.cshtml
OnlineDiaryApp
1 using OnlineDiaryApp.Models;
2
3 namespace OnlineDiaryApp.Composite
4 {
5     3 references
6     public class NoteLeaf : INotebookComponent
7     {
8         2 references
9         public Note Note { get; }
10
11         2 references
12         public NoteLeaf(Note note)
13         {
14             Note = note;
15
16         1 reference
17         public IEnumerable<INotebookComponent> GetChildren()
18         {
19             return new List<INotebookComponent>();
20
21         2 references
22         public void Display(int depth)
23         {
24             System.Console.WriteLine(new string('-', depth) + Note.Title);
25         }
26     }
27 }

```

Код 8 – NoteLeaf (Composite)

```

Details.cshtml  NotebookController.cs*  NoteLeaf.cs*  NotebookComposite.cs*  INotebookComponent.cs  In
OnlineDiaryApp

@model OnlineDiaryApp.Composite.NotebookComposite

<link rel="stylesheet" href="~/css/notes.css" />

<h2 class="notes-header">Блокнот: @Model.Name</h2>

<a class="btn-create" asp-controller="Note" asp-action="Create" asp-route-notebookId="@Model.Id">+ Створити нову нотатку</a>

<div class="sort-panel">
    <form method="get" asp-action="Details">
        <input type="hidden" name="id" value="@Model.Notebook.Id" />
        <label for="sortBy">Сортувати за:</label>
        <select id="sortBy" name="sortBy" onchange="this.form.submit()">
            <option value="">Без сортування</option>
            <option value="date" selected="@("date" == Context.Request.Query["sortBy"])">Датом</option>
            <option value="tag" selected="@("tag" == Context.Request.Query["sortBy"])">Тегом</option>
            <option value="title" selected="@("title" == Context.Request.Query["sortBy"])">Заголовком</option>
        </select>

        <input type="text" name="tag" placeholder="Введіть тег" value="@Context.Request.Query["tag"]" />
        <button type="submit">Застосувати</button>
    </form>
</div>

<div class="notes-grid">
    @foreach (var component in Model.GetChildren())
    {
        if (component is OnlineDiaryApp.Composite.NoteLeaf noteLeaf)
        {
            var note = noteLeaf.Note;
            <div class="note-card">
                <a asp-controller="Note" asp-action="Details" asp-route-id="@note.Id" class="note-card-link">
                    <div class="note-content">
                        <div class="note-title">@note.Title</div>
                        <div class="note-date">@note.CreatedAt.ToString("g")</div>

                        @if (note.Tags != null && note.Tags.Any())
                        {
                            <div class="note-tags">
                                @foreach (var tag in note.Tags)
                                {
                                    <span class="note-tag">@tag.Name</span>
                                }
                            </div>
                        }
                    </div>
                </a>
            </div>
        }
    }
</div>

```

Код 9 – Details (Views.Notebook)

5. Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

Висновки: У ході лабораторної роботи реалізовано патерн Composite для організації блокнотів і нотаток у веб-додатку. Це дозволило створити єдиний інтерфейс для роботи як з окремими нотатками, так і з блокнотами, що містять нотатки, спрощуючи обробку даних у контролерах і представленнях.

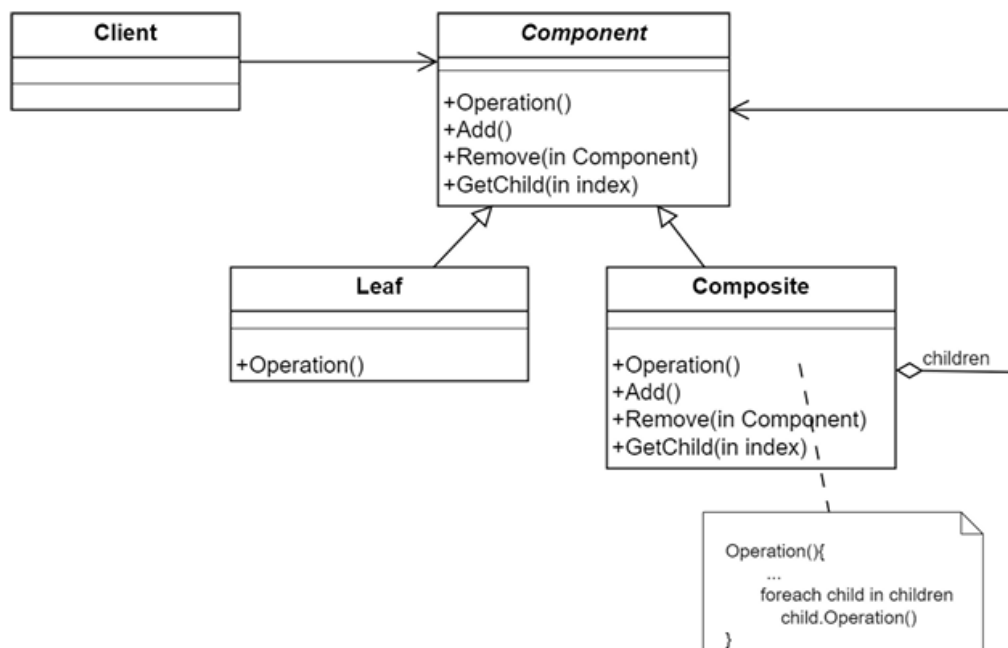
Відображення нотаток у вигляді карток і інтеграція кнопок створення стали більш зручними, а структура коду — гнучкою та масштабованою. Патерн Composite підвищив ефективність роботи з блокнотами та створив основу для подальшого розвитку додатку.

Відповіді на контрольні питання:

1. Призначення шаблону «Композит»

Дозволяє створювати деревоподібні структури «частина–ціле» та уніфіковано обробляти як окремі об'єкти, так і складені об'єкти з вкладеними елементами.

2. Структура шаблону «Композит»



3. Класи та взаємодія в «Композиті»

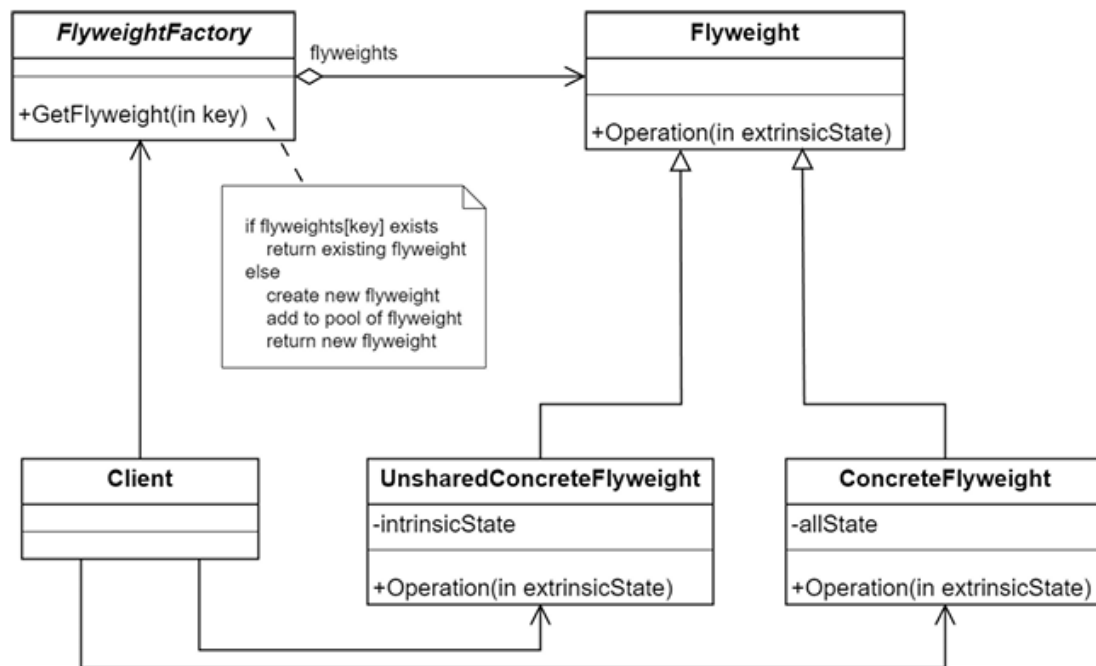
- **IComponent** – загальний інтерфейс для всіх об'єктів.
- **Leaf** – кінцевий об'єкт без дочірніх елементів.
- **Composite** – складений об'єкт, містить колекцію дочірніх об'єктів **IComponent**.

Взаємодія: клієнт працює з об'єктами через **IComponent**, **Composite** рекурсивно виконує операції над дочірніми елементами.

4. Призначення шаблону «Легковаговик» (Flyweight)

Зменшення кількості об'єктів за рахунок спільного використання, розділення внутрішнього стану (дані об'єкта) і зовнішнього стану (контекст використання).

5. Структура шаблону «Легковаговик»



6. Класи та взаємодія в «Легковаговику»

- Flyweight – інтерфейс або абстрактний клас, який визначає операції з внутрішнім станом.
- ConcreteFlyweight – реалізація Flyweight, зберігає внутрішній стан.
- FlyweightFactory – створює і керує спільними об'єктами.

Взаємодія: клієнт отримує Flyweight через фабрику та надає зовнішній стан при виконанні операцій.

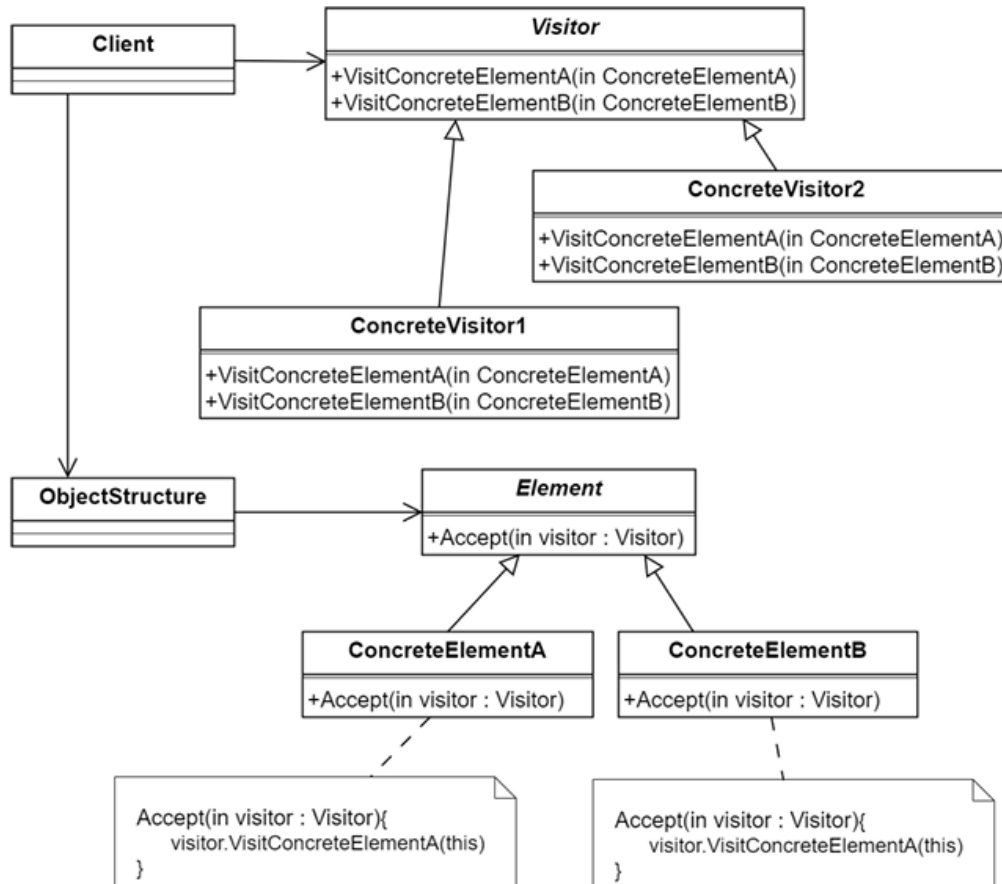
7. Призначення шаблону «Інтерпретатор»

Використовується для подання граматики мови та інтерпретації виразів через абстрактне синтаксичне дерево, де кожен вузол (термінальний або нетермінальний) обробляється рекурсивно у контексті.

8. Призначення шаблону «Відвідувач»

Дозволяє визначати операції над елементами об'єктної структури без зміни їх класів, відокремлюючи логіку від даних.

9. Структура шаблону «Відвідувач»



10. Класи та взаємодія в «Відвідувачі»

- **Visitor** – інтерфейс, визначає операції для кожного типу елемента.
- **ConcreteVisitor** – реалізує конкретні операції для елементів.
- **Element** – інтерфейс для елементів, які можуть приймати відвідувача.
- **ConcreteElement** – реалізація елементів, викликають метод відвідувача відповідно до свого типу.

Взаємодія: клієнт передає відвідувача об'єктам, елементи викликають відповідний метод відвідувача, що дозволяє виконувати операції без зміни самих елементів.