



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 №4
із дисципліни «*Технології розроблення програмного забезпечення*»
Тема: «Вступ до паттернів проектування»

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

Перевірив:
Мякий М.Ю.

Тема: Вступ до паттернів 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 «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

Посилання на репозиторій з проєктом:

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

Хід роботи

1. Ознайомитись з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

В даній частині було реалізовану логіку з тегами та нагадуваннями. Було реалізовано відповідні сервіси та контролери для виконання певного функціоналу.

TagService.cs
NoteController.cs
NoteService.cs
notes.css

OnlineDiaryApp
OnlineDiaryAp

```

1  using OnlineDiaryApp.Models;
2  using OnlineDiaryApp.Repositories.Interfaces;
3
4  namespace OnlineDiaryApp.Services
5  {
6      6 references
7      public class TagService
8      {
9
10         0 references
11         public TagService(ITagRepository tagRepository)
12         {
13             _tagRepository = tagRepository;
14         }
15
16         4 references
17         public async Task<IEnumerable<Tag>> GetAllTagsAsync(int userId)
18         {
19             var allTags = await _tagRepository.GetAllAsync(userId);
20             return allTags.Where(t => t.UserId == userId);
21         }
22
23         0 references
24         public async Task<Tag?> GetTagByIdAsync(int id)
25         {
26             return await _tagRepository.GetByIdAsync(id);
27         }
28
29         1 reference
30         public async Task<Tag> CreateTagAsync(string name, int userId)
31         {
32             var tag = new Tag
33             {
34                 Name = name,
35                 UserId = userId
36             };
37             await _tagRepository.AddAsync(tag);
38             await _tagRepository.SaveChangesAsync();
39             return tag;
40         }
41
42         0 references
43         public async Task<IEnumerable<Tag>> GetTagsByUserAsync(int userId)
44         {
45             var tags = await _tagRepository.GetAllAsync(userId);
46             return tags.Where(t => t.UserId == userId);
47         }
48
49         1 reference
50         public async Task DeleteTagAsync(int id)
51         {
52             await _tagRepository.DeleteAsync(id);
53             await _tagRepository.SaveChangesAsync();
54         }
55     }
56

```

Код 1 – TagService (services)

```

1  using Microsoft.AspNetCore.Mvc;
2  using OnlineDiaryApp.Services;
3
4  namespace OnlineDiaryApp.Controllers
5  {
6      1 reference
7      public class TagController : Controller
8      {
9          private readonly TagService _tagService;
10
11          0 references
12          public TagController(TagService tagService)
13          {
14              _tagService = tagService;
15          }
16
17          0 references
18          public async Task<IActionResult> Index()
19          {
20              var userIdString = HttpContext.Session.GetString("UserId");
21              if (!int.TryParse(userIdString, out int userId))
22                  return RedirectToAction("Login", "User");
23
24              var tags = await _tagService.GetAllTagsAsync(userId);
25              return View(tags);
26          }
27
28          0 references
29          public IActionResult Create()
30          {
31              return View();
32          }
33
34          [HttpPost]
35          0 references
36          public async Task<IActionResult> Create(string name)
37          {
38              var userIdString = HttpContext.Session.GetString("UserId");
39              if (!int.TryParse(userIdString, out int userId))
40                  return RedirectToAction("Login", "User");
41
42              if (string.IsNullOrWhiteSpace(name))
43              {
44                  ModelState.AddModelError("", "Назва тегу обов'язкова");
45                  return View();
46              }
47
48              await _tagService.CreateTagAsync(name, userId);
49              return RedirectToAction("Index");
50          }
51
52          0 references
53          public async Task<IActionResult> Delete(int id)
54          {
55              await _tagService.DeleteTagAsync(id);
56              return RedirectToAction("Index");
57          }
58      }
59  }

```

Код 2 – TagController (controllers)

```
ReminderController.cs  X  ReminderService.cs  TagService.cs  NoteService.c
OnlineDiaryApp
OnlineDiaryApp.Contr

1  using Microsoft.AspNetCore.Mvc;
2  using OnlineDiaryApp.Services;
3
4  namespace OnlineDiaryApp.Controllers
5  {
6      1 reference
7      public class ReminderController : Controller
8      {
9          private readonly ReminderService _reminderService;
10
11      0 references
12      public ReminderController(ReminderService reminderService)
13      {
14          _reminderService = reminderService;
15
16      0 references
17      public IActionResult Create(int noteId)
18      {
19          ViewBag.NoteId = noteId;
20          return View();
21      }
22
23      [HttpPost]
24      0 references
25      public async Task<IActionResult> Create(int noteId, DateTime remindAt)
26      {
27          var userIdString = HttpContext.Session.GetString("UserId");
28          if (!int.TryParse(userIdString, out int userId))
29          {
30              return RedirectToAction("Login", "User");
31          }
32
33          await _reminderService.CreateReminderAsync(noteId, remindAt, userId);
34
35          return RedirectToAction("Details", "Note", new { id = noteId });
36      }
37  }
```

Код 3 – ReminderController (controllers)

ReminderService.cs	TagService.cs	NoteController.cs	NoteService.cs	Index.cshtml	Remind
--------------------	---------------	-------------------	----------------	--------------	--------

OnlineDiaryApp
OnlineDiaryApp.Services.ReminderService

```

1  using OnlineDiaryApp.Models;
2
3  namespace OnlineDiaryApp.Services
4  {
5      7 references
6      public class ReminderService
7      {
8          private readonly IReminderRepository _reminderRepository;
9
10         0 references
11         public ReminderService(IReminderRepository reminderRepository)
12         {
13             _reminderRepository = reminderRepository;
14
15         3 references
16         public async Task CreateReminderAsync(int noteId, DateTime remindAt, int userId)
17         {
18             var utcRemindAt = DateTime.SpecifyKind(remindAt, DateTimeKind.Utc);
19
20             var reminder = new Reminder
21             {
22                 NoteId = noteId,
23                 UserId = userId,
24                 RemindAt = utcRemindAt,
25                 Status = "active"
26             };
27
28             await _reminderRepository.AddAsync(reminder);
29             await _reminderRepository.SaveChangesAsync();
30
31         1 reference
32         public async Task<IEnumerable<Reminder>> GetAllRemindersAsync()
33         {
34             return await _reminderRepository.GetAllAsync();
35
36         2 references
37         public async Task<Reminder?> GetReminderByNoteIdAsync(int noteId)
38         {
39             return await _reminderRepository.GetByNoteIdAsync(noteId);
40
41         2 references
42         public async Task UpdateReminderAsync(Reminder reminder, DateTime? newRemindAt = null, string? newStatus = null)
43         {
44             if (newRemindAt.HasValue)
45                 reminder.RemindAt = DateTime.SpecifyKind(newRemindAt.Value, DateTimeKind.Utc);
46
47             if (!string.IsNullOrEmpty(newStatus))
48                 reminder.Status = newStatus;
49
50             await _reminderRepository.UpdateAsync(reminder);
51             await _reminderRepository.SaveChangesAsync();
52
53         0 references
54         public async Task DeleteReminderAsync(int reminderId)
55         {
56             var reminder = await _reminderRepository.GetByIdAsync(reminderId);
57             if (reminder != null)
58             {
59                 await _reminderRepository.DeleteAsync(reminder.Id);
60                 await _reminderRepository.SaveChangesAsync();
61             }
62         }
63     }

```

Код 4 – ReminderService (services)

3. Реалізувати один з розглянутих шаблонів за обраною темою.

Для даного проекту було обрано патерн Strategy (Стратегія). Шаблон реалізований таким чином:

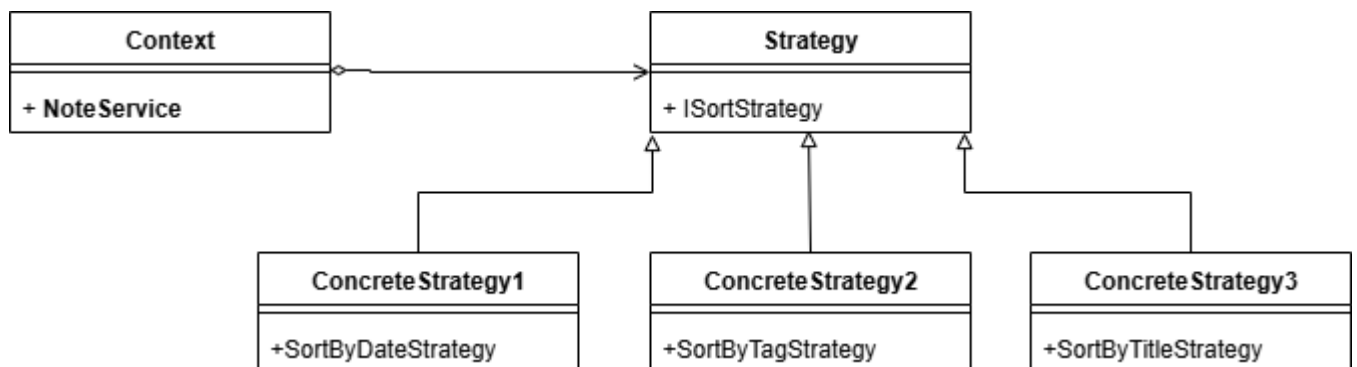


Рисунок 1 - Структура патерну Стратегія

- ISortStrategy – це інтерфейс, який описує метод Sort. Він задає контракт для всіх можливих стратегій сортування.
- SortByDate, SortByTag, SortByTitle – це конкретні реалізації стратегій. Кожен клас реалізує метод Sort по-своєму: один сортує записи за датою, інший – за тегами, ще один – за назвою.
- NoteService – це контекст, який працює з вибраною стратегією. Він не знає деталей реалізації, а просто викликає метод Sort у поточної стратегії.
- Note – модель, яка представляє дані, з якими працюють стратегії (назва, дата, тег).

В результаті:

Мажна легко змінювати алгоритм сортування, не змінюючи код самого NoteOrganizer. Це і є суть Стратегії – інкапсуляція алгоритмів у вигляді окремих класів та можливість динамічно їх змінювати.

4. Реалізувати не менше 3-х класів відповідно до обраної теми.

NoteService.cs* notes.css* note.css NoteRepository.cs Index.cshtml Index.csht

OnlineDiaryApp OnlineDiaryApp.Services.SortByDateStrategy

```

1  using OnlineDiaryApp.Models;
2  using OnlineDiaryApp.Repositories.Interfaces;
3
4
5  namespace OnlineDiaryApp.Services
6  {
7      6 references
8      public interface ISortStrategy
9      {
10         5 references
11         IEnumerable<Note> Sort(IEnumerable<Note> notes);
12     }
13
14     1 reference
15     public class SortByDateStrategy : ISortStrategy
16     {
17         3 references
18         public IEnumerable<Note> Sort(IEnumerable<Note> notes) =>
19             notes.OrderByDescending(n => n.CreatedAt);
20     }
21
22     2 references
23     public class SortByTagStrategy : ISortStrategy
24     {
25         private readonly string _tag;
26         1 reference
27         public SortByTagStrategy(string tag) => _tag = tag;
28
29         3 references
30         public IEnumerable<Note> Sort(IEnumerable<Note> notes) =>
31             notes.Where(n => n.Tags.Any(t => t.Name == _tag));
32     }
33
34     0 references
35     public class SortByTitleStrategy : ISortStrategy
36     {
37         3 references
38         public IEnumerable<Note> Sort(IEnumerable<Note> notes) =>
39             notes.OrderBy(n => n.Title);
40     }
41
42     6 references
43     public class NoteService
44     {
45         private readonly INoteRepository _noteRepository;
46         private readonly ITagRepository _tagRepository;
47
48         0 references
49         public NoteService(INoteRepository noteRepository, ITagRepository tagRepository)
50         {
51             _noteRepository = noteRepository;
52             _tagRepository = tagRepository;
53         }
54
55         1 reference
56         public async Task<IEnumerable<Note>> GetAllNotesByUserAsync(int userId, ISortStrategy? strategy = null)
57         {
58             var notes = (await _noteRepository.GetAllAsync())
59                 .Where(n => n.UserId == userId);
60
61             return strategy != null ? strategy.Sort(notes) : notes;
62         }
63
64         1 reference
65         public async Task<IEnumerable<Note>> GetAllNotesAsync(ISortStrategy? strategy = null)
66         {
67             var notes = await _noteRepository.GetAllAsync();
68             return strategy != null ? strategy.Sort(notes) : notes;
69         }
70
71         2 references
72         public async Task<Note?> GetNoteByIdAsync(int id) =>
73             await _noteRepository.GetByIdAsync(id);
74
75         1 reference
76         public async Task<Note> CreateNoteAsync(string title, string content, int userId, List<int> tagIds)
77         {
78             var tags = new List<Tag>();
79             foreach (var id in tagIds)
80             {
81                 var tag = await _tagRepository.GetByIdAsync(id);
82                 if (tag != null)
83                     tags.Add(tag);
84             }
85
86             var utcNow = DateTime.UtcNow;
87
88             var note = new Note
89             {
90                 Title = title,
91                 Content = content,
92                 UserId = userId,
93                 Tags = tags,
94                 CreatedAt = utcNow
95             };
96
97             await _noteRepository.AddAsync(note);
98             await _noteRepository.SaveChangesAsync();
99
100            return note;
101        }
102    }
  
```

Код 5.1 – NoteService (services)


```

86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120

1 reference
public async Task UpdateNoteAsync(Note note, List<int> tagIds)
{
    var tags = new List<Tag>();
    foreach (var id in tagIds)
    {
        var tag = await _tagRepository.GetByIdAsync(id);
        if (tag != null)
            tags.Add(tag);
    }

    note.Tags = tags;
    await _noteRepository.UpdateAsync(note);
    await _noteRepository.SaveChangesAsync();
}

1 reference
public async Task DeleteNoteAsync(int id)
{
    await _noteRepository.DeleteAsync(id);
    await _noteRepository.SaveChangesAsync();
}

1 reference
public async Task<IEnumerable<Note>> SearchByTitleAsync(string keyword)
{
    var notes = await _noteRepository.GetAllAsync();
    return notes.Where(n => n.Title.Contains(keyword, StringComparison.OrdinalIgnoreCase));
}

0 references
public async Task<IEnumerable<Tag>> GetAllTagsAsync(int userId) =>
(await _tagRepository.GetAllAsync(userId)).Where(t => t.UserId == userId);
}

```

Код 5.2 – NoteService (services)

NoteController.cs*
NoteService.cs*
notes.css*
NoteRepository.cs
Index.cshtml
Index.cshtml

OnlineDiaryApp
OnlineDiaryApp.Controllers.NoteController

```

1  using Microsoft.AspNetCore.Mvc;
2  using OnlineDiaryApp.Models;
3  using OnlineDiaryApp.Services;
4
5  namespace OnlineDiaryApp.Controllers
6  {
7      1 reference
8      public class NoteController : Controller
9      {
10         private readonly NoteService _noteService;
11         private readonly ReminderService _reminderService;
12         private readonly TagService _tagService;
13
14         0 references
15         public NoteController(NoteService noteService, ReminderService reminderService, TagService tagService)
16         {
17             _noteService = noteService;
18             _reminderService = reminderService;
19             _tagService = tagService;
20         }
21
22         0 references
23         public async Task<IActionResult> Index(string? sortBy, string? tag)
24         {
25             var userIdString = HttpContext.Session.GetString("UserId");
26             if (!int.TryParse(userIdString, out int userId))
27                 return RedirectToAction("Login", "User");
28
29             ISortStrategy? strategy = null;
30             if (!string.IsNullOrEmpty(sortBy))
31             {
32                 strategy = sortBy.ToLower() switch
33                 {
34                     "date" => new SortByDateStrategy(),
35                     "tag" when !string.IsNullOrEmpty(tag) => new SortByTagStrategy(tag),
36                     _ => null
37                 };
38             }
39
40             var notes = await _noteService.GetAllNotesByUserAsync(userId, strategy);
41
42             ViewBag.SortBy = sortBy;
43             ViewBag.SelectedTag = tag;
44             ViewBag.Tags = await _tagService.GetAllTagsAsync(userId);
45
46             return View(notes);
47         }
48
49         0 references
50         public async Task<IActionResult> Create()
51         {
52             var userIdString = HttpContext.Session.GetString("UserId");
53             if (!int.TryParse(userIdString, out int userId))
54                 return RedirectToAction("Login", "User");
55
56             var tags = await _tagService.GetAllTagsAsync(userId);
57             ViewBag.Tags = tags ?? new List<Tag>();
58
59             return View();
60         }
61
62         [HttpPost]
63         0 references
64         public async Task<IActionResult> Create(string title, string content, List<int>? tagIds, DateTime? reminderDate)
65         {
66             var userIdString = HttpContext.Session.GetString("UserId");
67             if (!int.TryParse(userIdString, out int userId))
68                 return RedirectToAction("Login", "User");
69
70             var note = await _noteService.CreateNoteAsync(title, content, userId, tagIds ?? new List<int>());
71             if (reminderDate.HasValue)
72             {
73                 await _reminderService.CreateReminderAsync(note.Id, reminderDate.Value, userId);
74             }
75
76             return RedirectToAction("Index");
77         }
78
79         0 references
80         public async Task<IActionResult> Edit(int id)
81         {
82             var note = await _noteService.GetNoteByIdAsync(id);
83             if (note == null)
84                 return NotFound();
85
86             var userIdString = HttpContext.Session.GetString("UserId");
87             if (!int.TryParse(userIdString, out int userId))
88                 return RedirectToAction("Login", "User");
89
90             ViewBag.Tags = await _tagService.GetAllTagsAsync(userId) ?? new List<Tag>();
91             var reminder = await _reminderService.GetReminderByNoteIdAsync(id);
92             ViewBag.Reminder = reminder;
93
94             return View(note);
95         }
96     }

```

Код 6.1 – NoteController (controllers)

```

93
94 [HttpPost]
95 0 references
96 public async Task<IActionResult> Edit(int id, string title, string content, List<int>? tagIds, DateTime? reminderDate)
97 {
98     var note = await _noteService.GetNoteByIdAsync(id);
99     if (note == null)
100         return NotFound();
101
102     note.Title = title;
103     note.Content = content;
104
105     await _noteService.UpdateNoteAsync(note, tagIds ?? new List<int>());
106
107     var existingReminder = await _reminderService.GetReminderByNoteIdAsync(note.Id);
108     if (reminderDate.HasValue)
109     {
110         if (existingReminder != null)
111         {
112             await _reminderService.UpdateReminderAsync(existingReminder, reminderDate.Value);
113         }
114         else
115         {
116             await _reminderService.CreateReminderAsync(note.Id, reminderDate.Value, note.UserId);
117         }
118     }
119
120     return RedirectToAction("Index");
121 }
122
123 0 references
124 public async Task<IActionResult> Delete(int id)
125 {
126     await _noteService.DeleteNoteAsync(id);
127     return RedirectToAction("Index");
128 }
129
130 0 references
131 public async Task<IActionResult> Search(string keyword)
132 {
133     var userIdString = HttpContext.Session.GetString("UserId");
134     if (!int.TryParse(userIdString, out int userId))
135         return View(new List<Note>());
136
137     var notes = await _noteService.SearchByTitleAsync(keyword);
138     notes = notes.Where(n => n.UserId == userId);
139     return View("Index", notes);
140 }

```

Код 6.2 – NoteController (controllers)

Index.cshtml	ReminderController.cs	ReminderService.cs	TagService.cs	NoteService.cs
C# OnlineDiaryApp				
<pre> @model IEnumerable<OnlineDiaryApp.Models.Note> <link rel="stylesheet" href="~/css/notes.css" /> <h2 class="notes-header">Мої нотатки</h2> + Створити нову нотатку <!-- Панель сортування --> <div class="sort-panel"> <form method="get" asp-action="Index"> <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"])">Датум</option> <option value="tag" selected="@("tag" == Context.Request.Query["sortBy"])">Тег</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> <!-- Список нотаток --> <div class="notes-grid"> @foreach (var note in Model) { <div class="note-card"> <div class="note-title">@note.Title</div> <div class="note-date">@note.CreatedAt.ToString("g")</div> <div class="note-tags"> @foreach (var tag in note.Tags) { @tag.Name } </div> <div class="note-actions"> Редагувати Видалити </div> </div> } </div> </pre>				

Код 7 – Index (Views/Note)

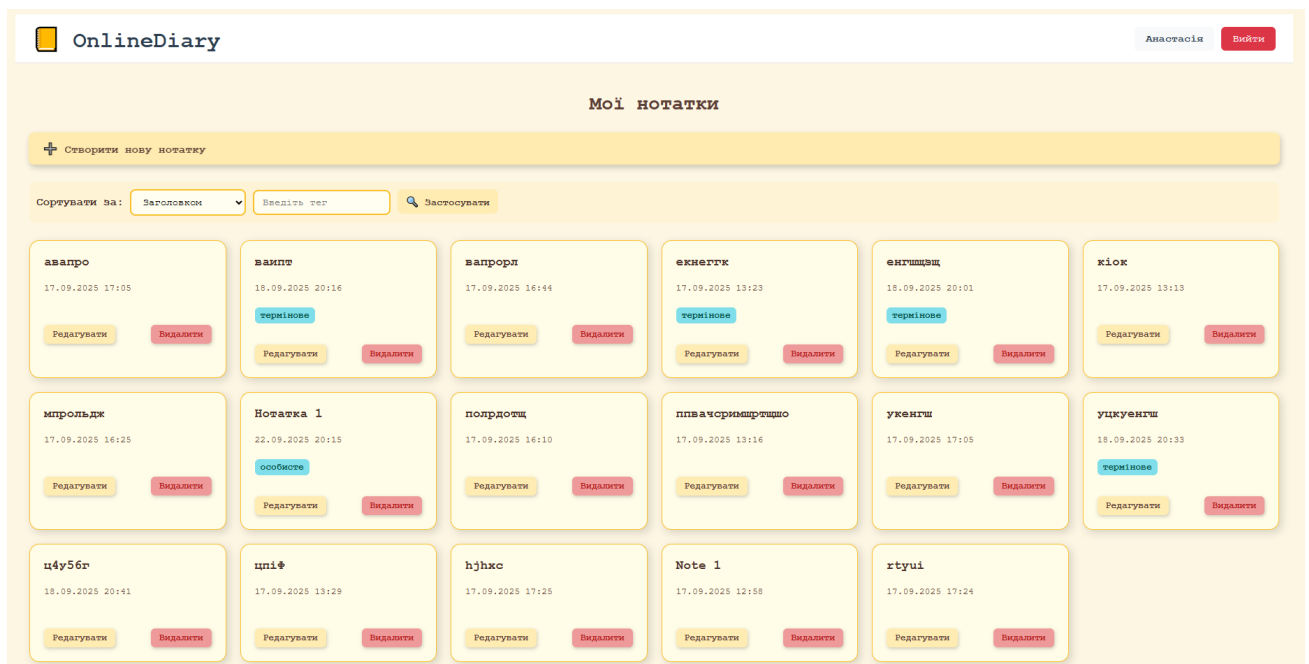


Рисунок 2 – Сортування за заголовком (в алфавітному порядку)

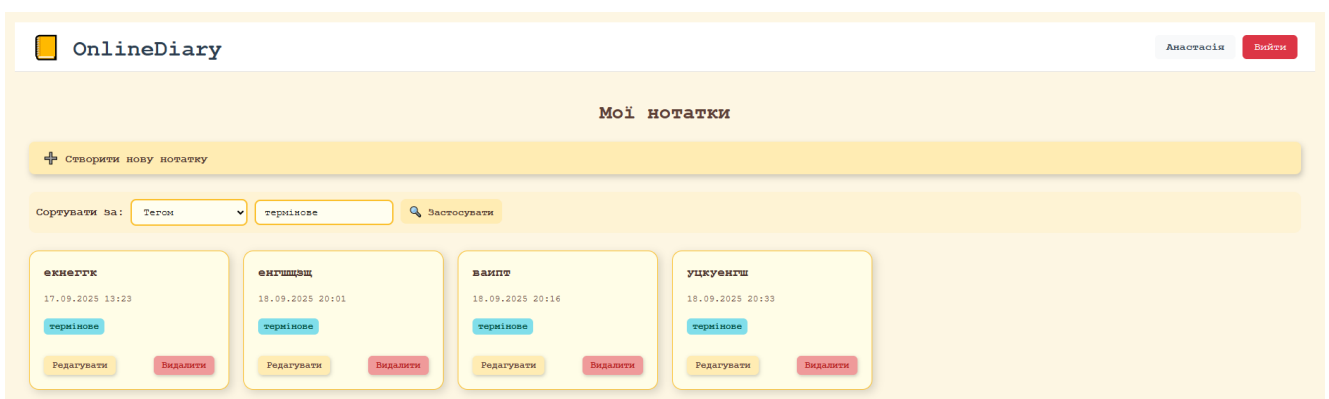


Рисунок 3 – Сортування за тегом

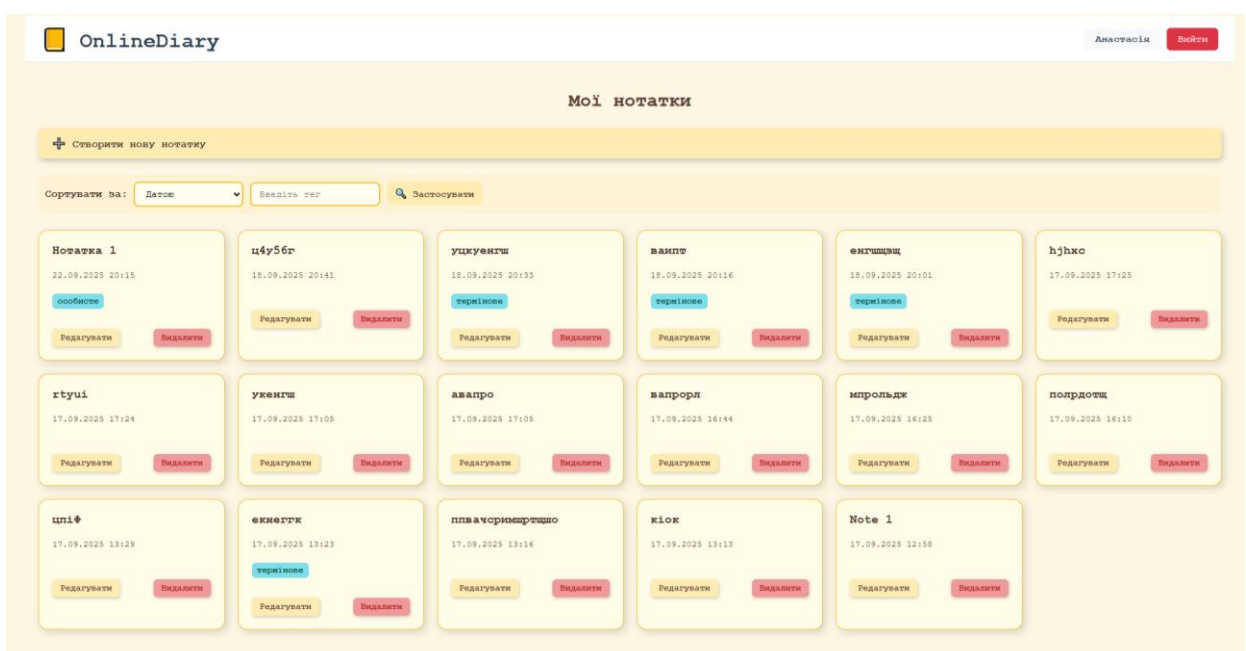


Рисунок 4 – Сортування за датою (від нових до давніх)

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 виконання лабораторної роботи було розглянуто базові шаблони проєктування, зокрема Singleton, Iterator, Proxy, State та Strategy. Отримані знання дали можливість зрозуміти, яку роль відіграють патерни у побудові гнучких і масштабованих програмних систем. На прикладі веб-застосунку «Онлайн-щоденник» було реалізовано шаблон Strategy для організації гнучкого механізму сортування нотаток за різними критеріями (датою, тегами тощо). Це дало змогу розділити бізнес-логіку і алгоритми сортування, а також легко додавати нові стратегії без змін у вже написаному коді. Таким чином, робота дозволила закріпити теоретичні знання про шаблони проєктування та на практиці переконатися в їх корисності для створення якісного програмного забезпечення.

Відповіді на контрольні питання:

1. Що таке шаблон проєктування?

Шаблон проєктування — це типове рішення поширеної задачі проєктування програмного забезпечення, перевірене практикою.

2. Навіщо використовувати шаблони проєктування?

Вони роблять код зрозумілішим, зручнішим для повторного використання, легшим у підтримці та розширенні.

3. Яке призначення шаблону «Стратегія»?

Дозволяє визначати сімейство алгоритмів, інкапсулювати кожен із них та робити їх взаємозамінними під час виконання.

4. Нарисуйте структуру шаблону «Стратегія».

(Тут можна вставити діаграму, як у тебе вище: Context → Strategy → ConcreteStrategy1/2).

5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

- Context – клас, який використовує стратегію.
- Strategy – інтерфейс для всіх стратегій.
- ConcreteStrategy – конкретні реалізації алгоритмів.

Context тримає посилання на Strategy і виконує метод вибраної реалізації.

6. Яке призначення шаблону «Стан»?

Дозволяє об'єкту змінювати свою поведінку залежно від внутрішнього стану, ніби він змінює свій клас.

7. Нарисуйте структуру шаблону «Стан».

(Діаграма: Context → State → ConcreteStateA / ConcreteStateB).

8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

- Context – клас, який зберігає поточний стан.
- State – інтерфейс станів.
- ConcreteState – реалізації різних станів.

Context делегує виконання поточному стану, який може змінити його на інший.

8. Яке призначення шаблону «Ітератор»?

Дозволяє послідовно перебирати елементи колекції, не розкриваючи її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор».

(Діаграма: Iterator ↔ ConcreteIterator ↔ Aggregate ↔ ConcreteAggregate).

11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?

- Iterator – інтерфейс для доступу до елементів.
- ConcreteIterator – конкретна реалізація.

- Aggregate – інтерфейс колекції.
- ConcreteAggregate – конкретна колекція, яка створює ітератор.

12. В чому полягає ідея шаблону «Одинак»?

Забезпечити існування лише одного екземпляра класу та надати глобальну точку доступу до нього.

13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Бо він порушує принципи ООП: ускладнює тестування, створює приховані залежності та глобальний стан.

14. Яке призначення шаблону «Проксі»?

Контролює доступ до іншого об'єкта, виступаючи його замісником (наприклад, ліниве завантаження, кешування, контроль доступу).

15. Нарисуйте структуру шаблону «Проксі».

(Діаграма: Client → Proxy → RealSubject ← Interface).

16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

- Subject – спільний інтерфейс.
- RealSubject – реальний об'єкт.
- Proxy – замісник, який містить посилання на RealSubject і контролює звернення клієнта.