



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 №9**  
із дисципліни «*Технології розроблення програмного забезпечення*»  
**Тема: «Взаємодія компонентів системи»**

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

Перевірив:  
Мякий М.Ю.

**Тема:** Взаємодія компонентів системи.

**Тема проєкту:** Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Service oriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

**Посилання на репозиторій з проєктом та звітами:**

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

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

## **Хід роботи**

1. Ознайомитись з короткими теоретичними відомостями.

### **Клієнт-серверна архітектура**

Клієнт-серверна архітектура — це модель, у якій клієнт відповідає за взаємодію з користувачем, а сервер — за зберігання й обробку даних. Тонкий клієнт передає більшість операцій на сервер і лише відображає результати (наприклад, вебзастосунки). Його перевага — просте оновлення, адже зміни виконуються лише на сервері. Товстий клієнт виконує більшу частину логіки на своїй стороні, що зменшує навантаження на сервер і дозволяє працювати офлайн (мобільні та десктопні застосунки — Evernote, Viber, Outlook тощо). SPA (Single Page Application) — проміжний варіант: логіка виконується на клієнті, але робота можлива лише при наявності зв'язку з сервером. Типова структура клієнт-серверної системи має три рівні: клієнтський (інтерфейс і взаємодія), спільний (middleware) і серверний (бізнес-логіка та робота з даними).

### **Peer-to-Peer архітектура**

P2P — децентралізована модель, у якій кожен вузол одночасно є клієнтом і сервером. Усі учасники рівноправні й обмінюються ресурсами без центрального сервера. Основні принципи: децентралізація, рівноправність вузлів і розподіл ресурсів. Приклади: BitTorrent, блокчейн, Skype, Zoom, розподілені обчислення (BOINC).

Недоліки: складність забезпечення безпеки, синхронізації й пошуку даних у великих мережах.

### Сервіс-орієнтована архітектура (SOA)

SOA — модульний підхід до створення системи як набору незалежних сервісів зі стандартизованими інтерфейсами, що взаємодіють через HTTP, SOAP або REST. Сервіси виконують конкретні бізнес-функції й обмінюються повідомленнями, не маючи спільної бази даних. Можуть бути обгортками для старих систем і реєструються в сервісному каталозі. Часто використовують Enterprise Service Bus (ESB) для взаємодії між сервісами. SOA стала основою для розвитку мікросервісної архітектури.

### Мікросервісна архітектура

Мікросервісна архітектура — це створення додатків як набору незалежних малих сервісів, що взаємодіють через HTTP, WebSockets або AMQP. Кожен мікросервіс має власну бізнес-логіку, життєвий цикл і може розгортатися автономно. За визначенням з книги О'Рейлі, мікросервіс — це незалежний компонент із чіткими межами, що спілкується через повідомлення. Переваги: гнучкість, масштабованість і легке супроводження великих систем.

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

NoteController.cs\* NotebookService.cs Create.cshtml Index.cshtml NotebookRepository.cs NotebookCont

OnlineDiaryApp

```

1 using Microsoft.AspNetCore.Mvc;
2 using OnlineDiaryApp.Models;
3 using OnlineDiaryApp.Services;
4 using OnlineDiaryApp.Utilities;
5
6 namespace OnlineDiaryApp.Controllers
7 {
8     1 reference
9     public class NoteController : Controller
10     {
11         private readonly NoteService _noteService;
12         private readonly ReminderService _reminderService;
13         private readonly TagService _tagService;
14         private readonly FileService _fileService;
15         private readonly NotebookService _notebookService;
16
17         0 references
18         public NoteController(
19             NoteService noteService,
20             ReminderService reminderService,
21             TagService tagService,
22             FileService fileService,
23             NotebookService notebookService)
24         {
25             _noteService = noteService;
26             _reminderService = reminderService;
27             _tagService = tagService;
28             _fileService = fileService;
29             _notebookService = notebookService;
30
31         // GET: /Note/
32         0 references
33         public async Task<IActionResult> Index(string? sortBy, string? tag)
34         {
35             var userId = GetUserId();
36             if (userId == null) return RedirectToAction("Login", "User");
37
38             ISortStrategy? strategy = null;
39             if (!string.IsNullOrEmpty(sortBy))
40             {
41                 strategy = sortBy.ToLower() switch
42                 {
43                     "date" => new SortByDateStrategy(),
44                     "tag" when !string.IsNullOrEmpty(tag) => new SortByTagStrategy(tag),
45                     "title" => new SortByTitleStrategy(),
46                     _ => null
47                 };
48             }
49
50             var notes = await _noteService.GetAllNotesByUserAsync(userId.Value, strategy);
51             ViewBag.SortBy = sortBy;
52             ViewBag.SelectedTag = tag;
53             ViewBag.Tags = await _tagService.GetAllTagsAsync(userId.Value);
54             ViewBag.Notebooks = await _notebookService.GetAllNotebooksAsync(userId.Value);
55
56             return View(notes);
57         }
58
59         [HttpGet]
60         0 references
61         public async Task<IActionResult> Create(int notebookId)
62         {
63             var userId = GetUserId();
64             if (userId == null) return RedirectToAction("Login", "User");
65
66             ViewBag.Tags = await _tagService.GetAllTagsAsync(userId.Value);
67             ViewBag.NotebookId = notebookId;
68             ViewBag.GoogleClientId = Environment.GetEnvironmentVariable("GOOGLE_CLIENT_ID");
69             ViewBag.GoogleApiKey = Environment.GetEnvironmentVariable("GOOGLE_API_KEY");
70             return View();
71         }
72
73         [HttpPost]
74         0 references
75         public async Task<IActionResult> Create(
76             string title,
77             string content,
78             int notebookId,
79             List<int>? tagIds,
80             DateTime? reminderDate,
81             List<string>? GoogleDriveLinks,
82             List<string>? GoogleDriveNames,
83             IFormFile? voiceNote)
84         {
85             var userId = int.Parse(HttpContext.Session.GetString("UserId") ?? "0");
86
87             var notebook = await _notebookService.GetNotebookByIdAsync(notebookId);
88             if (notebook == null)
89                 return BadRequest("Блокнот не найден");
90
91             var note = await _noteService.CreateNoteAsync(title, content, userId, notebookId, tagIds ?? new List<int>());
92         }
93     }
94 }

```

Код 1.1 – NoteController(Controllers)

```

88     var note = await _noteService.CreateNoteAsync(title, content, userId, notebookId, tagIds ?? new List<int>());
89
90     if (GoogleDriveLinks != null && GoogleDriveNames != null)
91     {
92         for (int i = 0; i < GoogleDriveLinks.Count; i++)
93         {
94             var link = GoogleDriveLinks[i];
95             var name = GoogleDriveNames.Count > i ? GoogleDriveNames[i] : "Google Drive file";
96             await _fileService.AddLinkFileAsync(note.Id, name, link);
97         }
98     }
99
100     if (voiceNote != null && voiceNote.Length > 0)
101     {
102         var fileName = Path.GetFileName(voiceNote.FileName);
103         var filePath = Path.Combine("wwwroot", "VoiceNotes", fileName);
104
105         using (var stream = new FileStream(filePath, FileMode.Create))
106         {
107             await voiceNote.CopyToAsync(stream);
108         }
109
110         await _fileService.AddVoiceFileAsync(note.Id, fileName, "/VoiceNotes/" + fileName);
111     }
112
113     await _noteService.UpdateNoteAsync(note, tagIds ?? new List<int>(), reminderDate);
114
115     return RedirectToAction("ListByNotebook", new { notebookId });
116 }
117
118 0 references
119 public async Task<IActionResult> ListByNotebook(int notebookId)
120 {
121     var userId = GetUserId();
122     if (userId == null) return RedirectToAction("Login", "User");
123
124     var notes = await _noteService.GetNotesByNotebookAsync(notebookId);
125     var notebook = await _notebookService.GetNotebookByIdAsync(notebookId);
126
127     ViewBag.Notebook = notebook;
128     ViewBag.Tags = await _tagService.GetAllTagsAsync(userId.Value);
129
130     return View("Index", notes);
131 }
132
133 // GET: /Note/Edit/5
134 0 references
135 public async Task<IActionResult> Edit(int id)
136 {
137     var note = await _noteService.GetNoteByIdAsync(id);
138     if (note == null) return NotFound();
139
140     var userId = GetUserId();
141     if (userId == null) return RedirectToAction("Login", "User");
142
143     ViewBag.Tags = await _tagService.GetAllTagsAsync(userId.Value);
144     ViewBag.Reminder = await _reminderService.GetReminderByNoteIdAsync(id);
145     ViewBag.GoogleClientId = Environment.GetEnvironmentVariable("GOOGLE_CLIENT_ID");
146     ViewBag.GoogleApiKey = Environment.GetEnvironmentVariable("GOOGLE_API_KEY");
147
148     return View(note);
149 }
150
151 // POST: /Note/Edit/5
152 [HttpPost]
153 0 references
154 public async Task<IActionResult> Edit(
155     int id,
156     string title,
157     string content,
158     List<int>? tagIds,
159     DateTime? remindAt,
160     List<string>? GoogleDriveLinks,
161     List<string>? GoogleDriveNames,
162     List<int>? DeletedFileIds,
163     IFormFile? voiceNote)
164 {
165     var note = await _noteService.GetNoteByIdAsync(id);
166     if (note == null) return NotFound();
167
168     note.Title = title;
169     note.Content = content;
170
171     if (DeletedFileIds != null)
172     {
173         foreach (var fileId in DeletedFileIds)
174             await _fileService.DeleteFileAsync(fileId);
175     }
176
177     if (GoogleDriveLinks != null && GoogleDriveNames != null)

```

## Код 1.2 – NoteController(Controllers)

```

176     if (GoogleDriveLinks != null && GoogleDriveNames != null)
177     {
178         for (int i = 0; i < GoogleDriveLinks.Count; i++)
179         {
180             var link = GoogleDriveLinks[i];
181             var name = GoogleDriveNames.Count > i ? GoogleDriveNames[i] : "Google Drive file";
182             await _fileService.AddLinkFileAsync(note.Id, name, link);
183         }
184     }
185
186     if (voiceNote != null && voiceNote.Length > 0)
187     {
188         var fileName = Path.GetFileName(voiceNote.FileName);
189         var filePath = Path.Combine("wwwroot", "VoiceNotes", fileName);
190
191         using (var stream = new FileStream(filePath, FileMode.Create))
192         {
193             await voiceNote.CopyToAsync(stream);
194         }
195
196         await _fileService.AddVoiceFileAsync(note.Id, fileName, "/VoiceNotes/" + fileName);
197     }
198
199     await _noteService.UpdateNoteAsync(note, tagIds ?? new List<int>(), remindAt);
200
201     return RedirectToAction("ListByNotebook", "Note", new { notebookId = note.NotebookId });
202 }
203
204 // GET: /Note/Delete/5
205 0 references
206 public async Task<IActionResult> Delete(int id)
207 {
208     var reminder = await _reminderService.GetReminderByNoteIdAsync(id);
209     if (reminder != null)
210         await _reminderService.DeleteReminderAsync(reminder.Id);
211
212     await _noteService.DeleteNoteAsync(id);
213     return RedirectToAction("Index");
214 }
215
216 // GET: /Note/Details/5
217 0 references
218 public async Task<IActionResult> Details(int id)
219 {
220     var note = await _noteService.GetNoteByIdAsync(id);
221     if (note == null) return NotFound();
222
223     ViewBag.Reminder = await _reminderService.GetReminderByNoteIdAsync(id);
224
225     note.Files = (await _fileService.GetFilesByNoteIdAsync(note.Id)).ToList();
226
227     return View(note);
228 }
229
230 [HttpGet]
231 0 references
232 public async Task<IActionResult> ExportToPdf(int id)
233 {
234     var note = await _noteService.GetNoteByIdAsync(id);
235     if (note == null)
236         return NotFound();
237
238     var pdfBytes = PdfGenerator.GenerateNotePdf(note.Title, note.Content);
239
240     var fileName = $"{note.Title}.pdf";
241     return File(pdfBytes, "application/pdf", fileName);
242 }
243
244 // GET: /Note/Search
245 0 references
246 public async Task<IActionResult> Search(string keyword)
247 {
248     var userId = GetUserId();
249     if (userId == null) return View(new List<Note>());
250
251     var notes = await _noteService.SearchByTitleAsync(keyword);
252     notes = notes.Where(n => n.UserId == userId.Value);
253     return View("Index", notes);
254 }
255
256 5 references
257 private int? GetUserId()
258 {
259     var userIdString = HttpContext.Session.GetString("UserId");
260     if (!int.TryParse(userIdString, out int userId)) return null;
261     return userId;
262 }

```

### Код 1.3 – NoteController(Controllers)

```

Create.cshtml*  NoteController.cs*  NotebookService.cs  Create.cshtml  Index.cshtml  NotebookR
C# OnlineDiaryApp
@model OnlineDiaryApp.Models.Note
@{
    ViewData["Title"] = "Створити нотатку";
}

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

<div class="note-page">
    <div class="note-container">
        <h2>Створити нотатку</h2>

        <form asp-action="Create" method="post" enctype="multipart/form-data">
            <input type="hidden" name="notebookId" value="@ViewBag.NotebookId" />

            <div class="form-group">
                <label asp-for="Title">Заголовок</label>
                <input asp-for="Title" class="form-control" required />
            </div>

            <div class="form-group">
                <label asp-for="Content">Текст</label>

                <div class="toolbar">
                    <button type="button" onclick="format('bold')"><b>B</b></button>
                    <button type="button" onclick="format('italic')"><i>I</i></button>
                    <button type="button" onclick="format('underline')"><u>U</u></button>
                    <button type="button" onclick="format('insertUnorderedList')">• Список</button>
                    <button type="button" onclick="format('insertOrderedList')">1. Список</button>
                    <button type="button" onclick="addLink()">🔗 Посилання</button>
                </div>

                <div id="editor" contenteditable="true" class="editor"></div>

                <input type="hidden" asp-for="Content" id="Content" />
            </div>

            <div class="form-group">
                <label for="tagSelect">Тег</label>
                <div id="tagSelectContainer">
                    <select id="tagSelect" class="form-control">
                        <option value="">- Оберіть тег -</option>
                        @foreach (var tag in ViewBag.Tags)
                        {
                            <option value="@tag.Id">@tag.Name</option>
                        }
                    </select>

                    <div id="selectedTags" class="selected-tags"></div>
                </div>
            </div>

            <div class="form-group">
                <label>Нагадування</label>
                <input type="datetime-local" name="reminderDate" class="form-control" />
            </div>

            <div class="form-group">
                <label>Файли з Google Drive</label>
                <button type="button" id="pickFromDrive" class="btn btn-secondary">Вибрати з Google Drive</button>
                <div id="driveFilesContainer" class="mt-2"></div>
            </div>

            <div class="form-group">
                <label>Голосова нотатка</label>
                <div>
                    <button type="button" id="startRecording" class="btn btn-info">Записати</button>
                    <button type="button" id="stopRecording" class="btn btn-warning disabled">Зупинити</button>
                </div>
                <audio id="audioPlayback" controls style="display:none;"></audio>
                <input type="file" id="voiceNoteFile" name="voiceNote" accept="audio/*" hidden />
            </div>

            <div class="note-actions">
                <a asp-controller="Notebook" asp-action="Index" class="btn-cancel">Скасувати</a>
                <button type="submit" class="btn-save">Зберегти</button>
            </div>
        </form>
    </div>
</div>

<style>
</style>

```

Код 2.1 – Create(Views.Note)



```

@section Scripts {
<script src="https://accounts.google.com/gsi/client" async defer></script>
<script src="https://apis.google.com/js/api.js"></script>
<script>

const CLIENT_ID = "@ViewBag.GoogleClientId";
const API_KEY = "@ViewBag.GoogleApiKey";
const SCOPES = 'https://www.googleapis.com/auth/drive.readonly';

let tokenClient;
let accessToken = null;

function gapiLoaded() {
    gapi.load('client:picker', initializePicker);
}

async function initializePicker() {
    await gapi.client.init({ apiKey: API_KEY });
    tokenClient = google.accounts.oauth2.initTokenClient({
        client_id: CLIENT_ID,
        scope: SCOPES,
        callback: (tokenResponse) => {
            accessToken = tokenResponse.access_token;
            createPicker();
        },
    });
}

function createPicker() {
    if (!accessToken) return;
    const view = new google.picker.View(google.picker.ViewId.DOCS);
    const picker = new google.picker.PickerBuilder()
        .addView(view)
        .setOAuthToken(accessToken)
        .setDeveloperKey(API_KEY)
        .setCallback(pickerCallback)
        .build();
    picker.setVisible(true);
}

function pickerCallback(data) {
    if (data.action === google.picker.Action.PICKED) {
        const filesContainer = document.getElementById("driveFilesContainer");
        data.docs.forEach(file => {
            const fileUrl = `https://drive.google.com/file/d/${file.id}/view`;
            const fileName = file.name;
            const div = document.createElement("div");
            div.innerHTML = `<a href="${fileUrl}" target="_blank">${file.name}</a>
                <input type="hidden" name="GoogleDriveLinks" value="${fileUrl}" />
                <input type="hidden" name="GoogleDriveNames" value="${fileName}" />`;
            filesContainer.appendChild(div);
        });
    }
}

document.getElementById('pickFromDrive').addEventListener('click', () => {
    if (!tokenClient) return;
    tokenClient.requestAccessToken();
});

window.addEventListener('load', gapiLoaded);

document.addEventListener('DOMContentLoaded', function () {
    // Tern
    const tagSelect = document.getElementById('tagSelect');
    const selectedTagsContainer = document.getElementById('selectedTags');

    tagSelect.addEventListener('change', function () {
        const selectedValue = this.value;
        const selectedText = this.options[this.selectedIndex].text;
        if (!selectedValue) return;
        if (selectedTagsContainer.querySelector(`[data-id="${selectedValue}"]`)) { this.value = ""; return; }

        const tagEl = document.createElement('span');
        tagEl.className = 'tag-item';
        tagEl.dataset.id = selectedValue;
        tagEl.innerHTML = `${selectedText} <button type="button" class="remove-tag"></button>`;

        const hiddenInput = document.createElement('input');
        hiddenInput.type = 'hidden';
        hiddenInput.name = 'tagIds';
        hiddenInput.value = selectedValue;

        selectedTagsContainer.appendChild(tagEl);
        selectedTagsContainer.appendChild(hiddenInput);

        this.value = "";
    });

    selectedTagsContainer.addEventListener('click', function (e) {
        if (e.target.classList.contains('remove-tag')) {
            const tagEl = e.target.closest('.tag-item');
            const tagId = tagEl.dataset.id;
            tagEl.remove();
            selectedTagsContainer.querySelector(`input[value="${tagId}"]`)?.remove();
        }
    });
});

```

Код 2.2 – Create(Views.Note)



```

document.querySelector("form").addEventListener("submit", function () {
    document.getElementById("hiddenContent").value = document.getElementById("editor").innerHTML;
});

function format(command) {
    document.execCommand(command, false, null);
}

function addLink() {
    const url = prompt("Введіть URL посилання:");
    if (url) document.execCommand("createLink", false, url);
}

document.querySelector('form').addEventListener('submit', function () {
    document.getElementById('Content').value = document.getElementById('editor').innerHTML;
});

let mediaRecorder;
let audioChunks = [];

const startBtn = document.getElementById("startRecording");
const stopBtn = document.getElementById("stopRecording");
const audioPlayback = document.getElementById("audioPlayback");
const voiceNoteFile = document.getElementById("voiceNoteFile");

startBtn.addEventListener("click", async () => {
    const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
    mediaRecorder = new MediaRecorder(stream);
    audioChunks = [];

    mediaRecorder.ondataavailable = e => audioChunks.push(e.data);
    mediaRecorder.onstop = e => {
        const audioBlob = new Blob(audioChunks, { type: 'audio/webm' });
        const audioUrl = URL.createObjectURL(audioBlob);
        audioPlayback.src = audioUrl;
        audioPlayback.style.display = "block";

        const file = new File([audioBlob], "VoiceNote.webm", { type: "audio/webm" });
        const dataTransfer = new DataTransfer();
        dataTransfer.items.add(file);
        voiceNoteFile.files = dataTransfer.files;
    };

    mediaRecorder.start();
    startBtn.disabled = true;
    stopBtn.disabled = false;
});

stopBtn.addEventListener("click", () => {
    mediaRecorder.stop();
    startBtn.disabled = false;
    stopBtn.disabled = true;
});
</script>

```

Код 2.3 – Create(Views.Note)

3. Реалізувати функціонал для роботи в розподіленому оточенні відповідно до обраної теми.

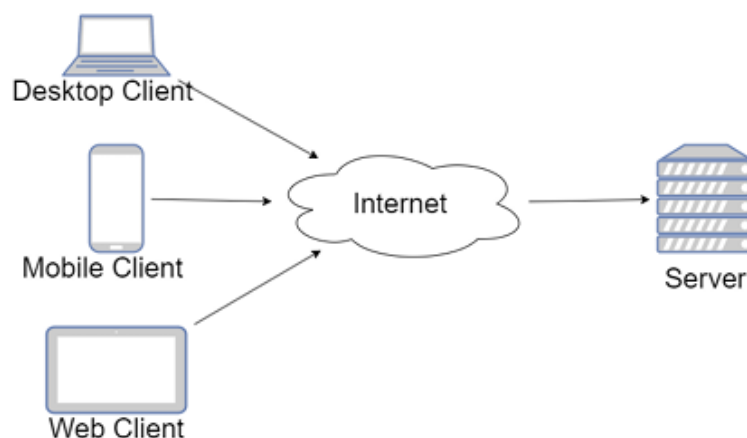


Рис.1 – Архітектура «клієнт-сервер»

У даному проєкті реалізовано класичну клієнт-серверну архітектуру, де клієнтом виступає вебінтерфейс користувача, а сервер забезпечує логіку обробки запитів, зберігання та доступ до даних.

### **Серверна частина**

Серверна частина побудована з використанням ASP.NET Core MVC, що виконує роль основного сервера застосунку. Вона реалізує:

- обробку HTTP-запитів клієнта через контролери (наприклад, NoteController, NotebookController, TagController);
- доступ до бази даних через репозиторії (NotebookRepository, NoteRepository), які використовують ORM Entity Framework для взаємодії з базою.

База даних розташована на сервері та зберігає інформацію про користувачів, нотатки, блокноти та теги. Сервер відповідає на запити клієнта, формує представлення та надсилає його у вигляді HTML-сторінки.

### **Клієнтська частина**

Клієнтська частина представлена у вигляді вебінтерфейсу, реалізованого за допомогою Razor-сторінок (.cshtml) і JavaScript. Вона відображає дані, отримані від сервера, та надсилає запити (через форми) для створення, редагування та видалення нотаток і блокнотів. Клієнт не має власної бізнес-логіки — він виконує роль тонкого клієнта, тобто лише взаємодіє із сервером та відображає результати користувачеві.

## **Middleware**

Посередницький шар у проєкті реалізовано у вигляді сервісів і моделей:

- Сервіси (NoteService, NotebookService) інкапсулюють бізнес-логіку;
- Інтерфейси репозиторіїв (INoteRepository, INotebookRepository) визначають контракти доступу до даних;
- Моделі (Note, Notebook, Tag) описують сутності системи, які спільно використовуються клієнтською і серверною частинами.

Цей шар виступає сполучною ланкою між контролерами (які обробляють запити клієнта) та базою даних.

## **Взаємодія клієнта і сервера**

Типова послідовність взаємодії виглядає так:

1. Користувач (клієнт) відкриває сторінку — браузер надсилає HTTP-запит до сервера.
2. Контролер на сервері приймає запит і звертається до відповідного сервісу.
3. Сервіс викликає метод репозиторію, який працює з базою даних.
4. Результат (наприклад, список нотаток) передається назад через сервіс і контролер до представлення (.cshtml), яке відображає його користувачу.

Таким чином, відбувається чітке розділення обов'язків між клієнтом, сервером та проміжним шаром логіки.

4. Реалізувати взаємодію розподілених частин.

```

Index.cshtml*  NotebookRepository.cs  NotebookController.cs  NoteController.cs  Program.cs  .en
OnlineDiaryApp
@model IEnumerable<OnlineDiaryApp.Models.Notebook>

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

<h2 class="notes-header">Мої блокноти</h2>

<a class="btn-create" asp-controller="Notebook" asp-action="Create">+ Створити новий блокнот</a>

<div class="notes-grid">
    @foreach (var nb in Model)
    {
        <div class="note-card">
            <a asp-controller="Notebook" asp-action="Details" asp-route-id="@nb.Id" class="note-card-link">
                <div class="note-content">
                    <div class="note-title">@nb.Name</div>
                    @if (!string.IsNullOrEmpty(nb.Description))
                    {
                        <div class="note-date">@nb.Description</div>
                    }
                </div>
            </a>

            <div class="note-actions">
                <a asp-controller="Notebook" asp-action="Edit" asp-route-id="@nb.Id" class="btn-edit">Редагувати</a>
                <a asp-controller="Notebook" asp-action="Delete" asp-route-id="@nb.Id" class="btn-delete">Видалити</a>
            </div>
        </div>
    }
</div>

```

Код 3 – Index(Views.Notebook)

```

Create.cshtml  Index.cshtml*  NotebookRepository.cs  NotebookControll
OnlineDiaryApp
@model OnlineDiaryApp.Models.Notebook

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

<div class="page-wrapper">
    <div class="form-container">
        <h2>Створити новий блокнот</h2>

        <form asp-action="Create" method="post">
            <div class="form-group">
                <label asp-for="Name">Назва</label>
                <input asp-for="Name" class="form-control" />
                <span asp-validation-for="Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Description">Опис</label>
                <textarea asp-for="Description" class="form-control"></textarea>
            </div>
            <div class="note-actions">
                <a asp-action="Index" class="btn-cancel">Скасувати</a>
                <button type="submit" class="btn-save">Зберегти</button>
            </div>
        </form>
    </div>
</div>

```

Код 4 – Create(Views.Notebook)

```

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 }

```

Код 5 – INotebookRepository(Repoisitories.Interfaces)

```

1  namespace OnlineDiaryApp.Models
2  {
3      public class Notebook
4      {
5          public int Id { get; set; }
6          public string Name { get; set; } = null!;
7          public string Description { get; set; }
8          public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
9          public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;
10
11         public int UserId { get; set; }
12         public User User { get; set; }
13
14         public ICollection<Note> Notes { get; set; } = new List<Note>();
15     }
16 }

```

Код 6 – Notebook(Models)

NotebookService.cs   Create.cshtml   Index.cshtml\*   NotebookRepository.cs   NotebookController.cs

OnlineDiaryApp   OnlineDiaryApp.Services.NotebookServ

```

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

```

Код 7 – NotebookService(Services)

NotebookController.cs	NoteController.cs	Program.cs	.env	Tag.cs	Reminder
-----------------------	-------------------	------------	------	--------	----------

```

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 8.1 – NotebookController(Controllers)



```

76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107

```

```

0 references
public async Task<IActionResult> Details(int id, string? sortBy, string? tag)
{
    var notebook = await _notebookService.GetNotebookByIdAsync(id);
    if (notebook == null) return NotFound();

    var notes = await _noteService.GetNotesByNotebookAsync(id);

    if (!string.IsNullOrEmpty(tag))
        notes = notes.Where(n => n.Tags.Any(t => t.Name == tag)).ToList();

    ISortStrategy? strategy = sortBy?.ToLower() switch
    {
        "date" => new SortByDateStrategy(),
        "title" => new SortByTitleStrategy(),
        "tag" when !string.IsNullOrEmpty(tag) => new SortByTagStrategy(tag),
        _ => null
    };

    if (strategy != null)
        notes = strategy.Sort(notes).ToList();

    var notebookComposite = new NotebookComposite(notebook);
    foreach (var note in notes)
        notebookComposite.Add(new NoteLeaf(note));

    return View(notebookComposite);
}
}

```

Код 8.2 – NotebookController(Controllers)

NotebookRepository.cs
NotebookController.cs
NoteController.cs
Pro

OnlineDiaryApp

```

1  using Microsoft.EntityFrameworkCore;
2  using OnlineDiaryApp.Data;
3  using OnlineDiaryApp.Models;
4  using OnlineDiaryApp.Repositories.Interfaces;
5
6  namespace OnlineDiaryApp.Repositories
7  {
8      2 references
9      public class NotebookRepository : INotebookRepository
10     {
11         private readonly AppDbContext _context;
12
13         0 references
14         public NotebookRepository(AppDbContext context)
15         {
16             _context = context;
17         }
18
19         2 references
20         public async Task<IEnumerable<Notebook>> GetAllAsync(int userId)
21         {
22             return await _context.Notebooks
23                 .Include(n => n.Notes)
24                 .Where(n => n.UserId == userId)
25                 .ToListAsync();
26         }
27
28         4 references
29         public async Task<Notebook?> GetByIdAsync(int id)
30         {
31             return await _context.Notebooks
32                 .Include(n => n.Notes)
33                 .FirstOrDefaultAsync(n => n.Id == id);
34         }
35
36         2 references
37         public async Task AddAsync(Notebook notebook)
38         {
39             await _context.Notebooks.AddAsync(notebook);
40         }
41
42         2 references
43         public async Task UpdateAsync(Notebook notebook)
44         {
45             _context.Notebooks.Update(notebook);
46         }
47
48         2 references
49         public async Task DeleteAsync(int id)
50         {
51             var notebook = await _context.Notebooks.FindAsync(id);
52             if (notebook != null)
53                 _context.Notebooks.Remove(notebook);
54         }
55
56         4 references
57         public async Task SaveChangesAsync()
58         {
59             await _context.SaveChangesAsync();
60         }
61     }
62 }

```

Код 9 – NotebookRepository(Repositories.Implementations)

5. Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє спроектовану архітектуру. Навести фрагменти програмного коду, які є суттєвими для відображення реалізованої архітектури.

**Висновки:** У ході лабораторної роботи реалізовано патерн Composite для організації блоктів 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. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура — це модель організації комп'ютерних систем, де клієнти (зазвичай користувацькі програми або пристрої) роблять запити на сервер, який обробляє ці запити і повертає результати. Клієнт відповідає за інтерфейс користувача та ініціацію запитів, сервер — за обробку даних, зберігання та логіку.

2. Розкажіть про сервіс-орієнтовану архітектуру (SOA).

SOA — це архітектурний підхід, де функціональність програми реалізована у вигляді сервісів — автономних компонентів, що виконують конкретні бізнес-завдання. Кожен сервіс має чіткий інтерфейс і взаємодіє з іншими через стандартизовані протоколи (наприклад, HTTP, SOAP, REST).

3. Якими принципами керується SOA?

Основні принципи SOA:

- Автономність сервісів — сервіси працюють незалежно.
- Стандартизовані інтерфейси — сервіси взаємодіють через визначені API.
- Повторне використання — сервіси можна використовувати в різних системах.
- Легко інтегрувати — сервіси повинні легко поєднуватись у складні процеси.
- Слабке зв'язування (loose coupling) — зміни в одному сервісі мінімально впливають на інші.

#### 4. Як між собою взаємодіють сервіси в SOA?

Сервіси взаємодіють через стандартизовані повідомлення або API, наприклад через SOAP або REST. Кожен сервіс публікує свій інтерфейс (WSDL, OpenAPI), і інші сервіси можуть викликати його методи, обмінюючись даними у формі XML, JSON або інших форматах.

#### 5. Як розробники дізнаються про існуючі сервіси і як робити до них запити?

- Реєстр сервісів (Service Registry) — централізована база, де зареєстровані всі сервіси та їхні інтерфейси.
- Документація API (наприклад OpenAPI/Swagger).
- Запити здійснюються через стандартні протоколи (HTTP, SOAP, REST) за адресою сервісу і з використанням описаних методів та форматів даних.

#### 5. У чому полягають переваги та недоліки клієнт-серверної моделі?

Переваги:

- Центральне зберігання даних, легший контроль безпеки.
- Легко масштабувати сервери.
- Клієнти можуть бути простими, вся логіка на сервері.

Недоліки:

- Сервер може стати вузьким місцем (single point of failure).
- Високі вимоги до потужності сервера при великій кількості клієнтів.
- Залежність клієнта від сервера — без доступу до сервера система не працює.

7. У чому полягають переваги та недоліки однорангової (peer-to-peer) моделі взаємодії?

Переваги:

- Відсутність централізованого сервера, підвищена стійкість до відмов.
- Можливість прямого обміну ресурсами між учасниками.
- Масштабування «горизонтальне» — додаючи вузли, підвищуєш потужність.

Недоліки:

- Складніше забезпечувати безпеку та контроль доступу.
- Кожен вузол відповідає за управління ресурсами.
- Важче координувати оновлення і синхронізацію даних.

8. Що таке мікросервісна архітектура?

Мікросервісна архітектура — це підхід, де додаток складається з малих, незалежних сервісів, кожен з яких реалізує одну бізнес-функцію і може розгортатися окремо. Вона є розвитком SOA, але з більш дрібними і автономними компонентами.

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?

- HTTP/HTTPS + REST
- gRPC
- SOAP
- Message brokers: RabbitMQ, Kafka, MQTT для асинхронної взаємодії
- WebSockets для реального часу

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, якщо у проєкті між веб-контролерами та шаром доступу до даних реалізуємо шар бізнес-логіки у вигляді сервісів?

Це не повноцінна SOA, а локальне використання сервісів всередині одного додатку. SOA передбачає автономні, незалежні сервіси, доступні через

стандартизовані протоколи для інших систем. Твій підхід — це архітектурний патерн «сервісний шар» (Service Layer), який організовує бізнес-логіку всередині одного проєкту.