



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

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

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

Перевірив:
Мягкий М.Ю.

Тема: Взаємодія компонентів системи.

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

```

Код 1.1 – NoteController(Controllers)

```

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
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176

    var note = await _noteService.CreateNoteAsync(title, content, userId, notebookId, tagIds ?? new List<int>());
    if (GoogleDriveLinks != null && GoogleDriveNames != null)
    {
        for (int i = 0; i < GoogleDriveLinks.Count; i++)
        {
            var link = GoogleDriveLinks[i];
            var name = GoogleDriveNames.Count > i ? GoogleDriveNames[i] : "Google Drive file";
            await _fileService.AddLinkFileAsync(note.Id, name, link);
        }
    }

    if (voiceNote != null && voiceNote.Length > 0)
    {
        var fileName = Path.GetFileName(voiceNote.FileName);
        var filePath = Path.Combine("wwwroot", "VoiceNotes", fileName);

        using (var stream = new FileStream(filePath, FileMode.Create))
        {
            await voiceNote.CopyToAsync(stream);
        }

        await _fileService.AddVoiceFileAsync(note.Id, fileName, "/VoiceNotes/" + fileName);
    }

    await _noteService.UpdateNoteAsync(note, tagIds ?? new List<int>(), reminderDate);
}

return RedirectToAction("ListByNotebook", new { notebookId });
}

0 references
public async Task<IActionResult> ListByNotebook(int notebookId)
{
    var userId = GetUserId();
    if (userId == null) return RedirectToAction("Login", "User");

    var notes = await _noteService.GetNotesByNotebookAsync(notebookId);
    var notebook = await _notebookService.GetNotebookByIdAsync(notebookId);

    ViewBag.Notebook = notebook;
    ViewBag.Tags = await _tagService.GetAllTagsAsync(userId.Value);

    return View("Index", notes);
}

// GET: /Note/Edit/5
0 references
public async Task<IActionResult> Edit(int id)
{
    var note = await _noteService.GetNoteByIdAsync(id);
    if (note == null) return NotFound();

    var userId = GetUserId();
    if (userId == null) return RedirectToAction("Login", "User");

    ViewBag.Tags = await _tagService.GetAllTagsAsync(userId.Value);
    ViewBag.Reminder = await _reminderService.GetReminderByNoteIdAsync(id);
    ViewBag.GoogleClientId = Environment.GetEnvironmentVariable("GOOGLE_CLIENT_ID");
    ViewBag.GoogleApiKey = Environment.GetEnvironmentVariable("GOOGLE_API_KEY");

    return View(note);
}

// POST: /Note/Edit/5
[HttpPost]
0 references
public async Task<IActionResult> Edit(
    int id,
    string title,
    string content,
    List<int>? tagIds,
    DateTime? remindAt,
    List<string>? GoogleDriveLinks,
    List<string>? GoogleDriveNames,
    List<int>? DeletedFileIds,
    IFormFile? voiceNote)
{
    var note = await _noteService.GetNoteByIdAsync(id);
    if (note == null) return NotFound();

    note.Title = title;
    note.Content = content;

    if (DeletedFileIds != null)
    {
        foreach (var fileId in DeletedFileIds)
            await _fileService.DeleteFileAsync(fileId);
    }

    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
205 // GET: /Note/Delete/5
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
217 // GET: /Note/Details/5
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 public async Task<IActionResult> ExportToPdf(int id)
232 {
233     var note = await _noteService.GetNoteByIdAsync(id);
234     if (note == null)
235         return NotFound();
236
237     var pdfBytes = PdfGenerator.GenerateNotePdf(note.Title, note.Content);
238
239     var fileName = $"{note.Title}.pdf";
240     return File(pdfBytes, "application/pdf", fileName);
241 }
242
243
244 // GET: /Note/Search
245 public async Task<IActionResult> Search(string keyword)
246 {
247     var userId = GetUserId();
248     if (userId == null) return View(new List<Note>());
249
250     var notes = await _noteService.SearchByTitleAsync(keyword);
251     notes = notes.Where(n => n.UserId == userId.Value);
252     return View("Index", notes);
253 }
254
255 private int? GetUserId()
256 {
257     var userIdString = HttpContext.Session.GetString("UserId");
258     if (!int.TryParse(userIdString, out int userId)) return null;
259     return userId;
260 }
261 }
262

```

Код 1.3 – NoteController(Controllers)

```

Create.cshtml*  X  NoteController.cs*  NotebookService.cs  Create.cshtml  Index.cshtml  NotebookR
[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()">&nbsp; 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">Теги</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 () {
        // Tag select
        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">x</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.

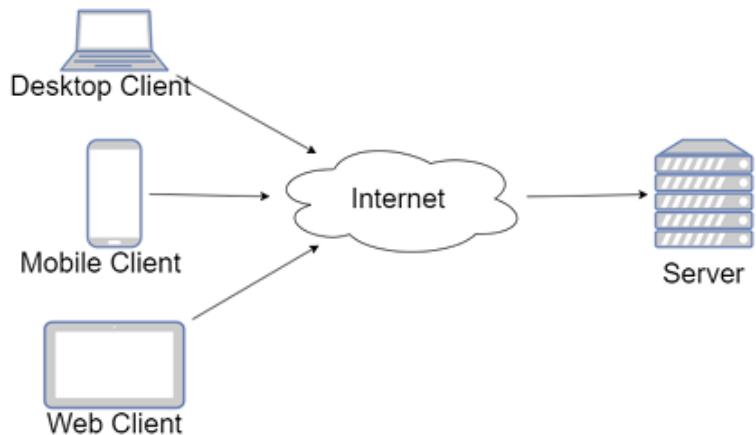


Рис.1 – Архітектура «клієнт-сервер»

У даному проекті 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 проекті 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:

- Користувач (клієнт) відкриває сторінку — браузер надсилає HTTP-запит до сервера.
- Контролер на сервері приймає запит і звертається до відповідного сервісу.
- Сервіс викликає метод репозиторію, який працює з базою даних.
- 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.

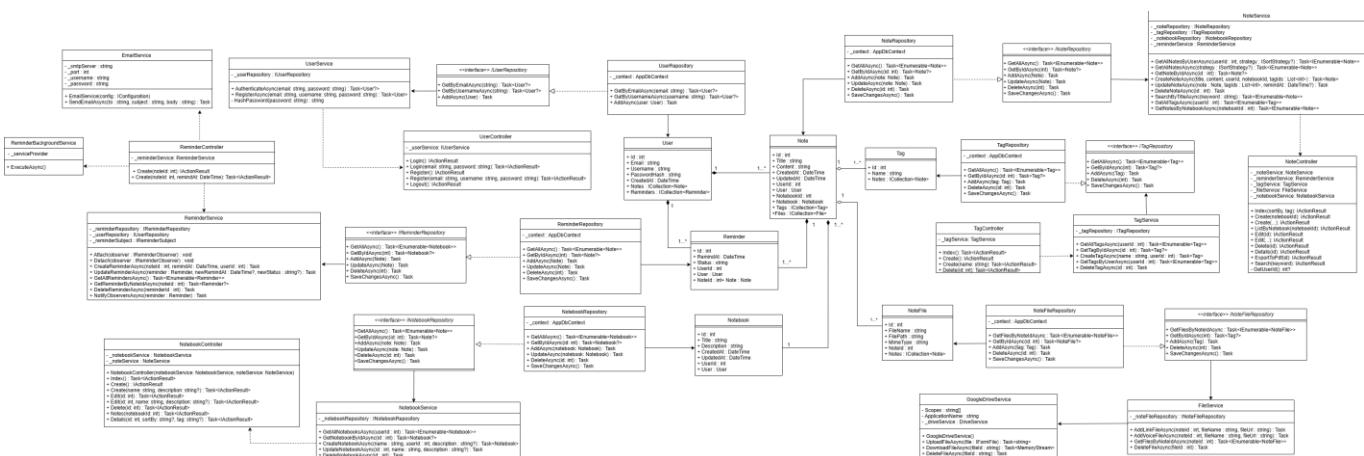


Рис.2 – Діаграма класів

4. Реалізувати взаємодію розподілених частин.

```
Index.cshtml* NoteBookRepository.cs NoteBookController.cs NoteController.cs Program.cs .en
C# 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 NoteBookController.cs
C# 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)

The screenshot shows a code editor with the tab bar at the top containing 'INotebookRepository.cs', 'NotebookService.cs', and 'Create.cshtml'. The active tab is 'INotebookRepository.cs'. The code in the editor is:

```
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(Repositories.Interfaces)

The screenshot shows a code editor with the tab bar at the top containing 'Notebook.cs', 'INotebookRepository.cs', 'NotebookService.cs', and 'Create.cshtml'. The active tab is 'Notebook.cs'. The code in the editor is:

```
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         public int UserId { get; set; }
11         public User User { get; set; }
12         public ICollection<Note> Notes { get; set; } = new List<Note>();
13     }
14 }
```

Код 6 – Notebook(Models)

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

Код 7 – NotebookService(Services)

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

Код 8.1 – NotebookController(Controllers)

```
76
77     0 references
78     public async Task<IActionResult> Details(int id, string? sortBy, string? tag)
79     {
80         var notebook = await _notebookService.GetNotebookByIdAsync(id);
81         if (notebook == null) return NotFound();
82
83         var notes = await _noteService.GetNotesByNotebookAsync(id);
84
85         if (!string.IsNullOrEmpty(tag))
86             notes = notes.Where(n => n.Tags.Any(t => t.Name == tag)).ToList();
87
88         ISortStrategy? strategy = sortBy?.ToLower() switch
89         {
90             "date" => new SortByDateStrategy(),
91             "title" => new SortByTitleStrategy(),
92             "tag" when !string.IsNullOrEmpty(tag) => new SortByTagStrategy(tag),
93             _ => null
94         };
95
96         if (strategy != null)
97             notes = strategy.Sort(notes).ToList();
98
99         var notebookComposite = new NotebookComposite(notebook);
100        foreach (var note in notes)
101            notebookComposite.Add(new NoteLeaf(note));
102
103        return View(notebookComposite);
104    }
105
106
107 }
```

Код 8.2 – NotebookController(Controllers)

The screenshot shows a code editor with the tab bar at the top containing "NotebookRepository.cs", "NotebookController.cs", "NoteController.cs", and "Pro". The main area displays the "NotebookRepository.cs" file. The code is a C# class definition for "NotebookRepository" that implements the "INotebookRepository" interface. It uses Microsoft.EntityFrameworkCore and OnlineDiaryApp.Data namespaces. The class has a private readonly field "_context" of type AppDbContext. It includes methods for getting all notebooks for a user, getting a notebook by ID, adding a new notebook, updating an existing notebook, deleting a notebook, and saving changes. The code uses async/await and Entity Framework's .Include and .Where methods.

```
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 }
```

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 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. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура — це модель організації комп’ютерних систем, де клієнти (зазвичай користувачькі програми або пристрой) 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.
- Стандартизовани 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 стандартизовани 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, підвищуючи 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 проекту.