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:
Мягкий М.Ю.

Київ-2025

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/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.

Код 1.1 – NoteController(Controllers)

```
            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)

```
            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>(), remindAt);

            return RedirectToAction("ListByNotebook", "Note", new { notebookId = note.NotebookId });
        }


        // GET: /Note/Delete/5
        0 references
        public async Task<IActionResult> Delete(int id)
        {
            var reminder = await _reminderService.GetReminderByNoteIdAsync(id);
            if (reminder != null)
                await _reminderService.DeleteReminderAsync(reminder.Id);

            await _noteService.DeleteNoteAsync(id);
            return RedirectToAction("Index");
        }

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

            ViewBag.Reminder = await _reminderService.GetReminderByNoteIdAsync(id);

            note.Files = (await _fileService.GetFilesByNoteIdAsync(note.Id)).ToList();

            return View(note);
        }

        [HttpGet]
        0 references
        public async Task<IActionResult> ExportToPdf(int id)
        {
            var note = await _noteService.GetNoteByIdAsync(id);
            if (note == null)
                return NotFound();


            var pdfBytes = PdfGenerator.GenerateNotePdf(note.Title, note.Content);

            var fileName = $"{note.Title}.pdf";
            return File(pdfBytes, "application/pdf", fileName);
        }


        // GET: /Note/Search
        0 references
        public async Task<IActionResult> Search(string keyword)
        {
            var userId = GetUserId();
            if (userId == null) return View(new List<Note>());

            var notes = await _noteService.SearchByTitleAsync(keyword);
            notes = notes.Where(n => n.UserId == userId.Value);
            return View("Index", notes);
        }

        5 references
        private int? GetUserId()
        {
            var userIdString = HttpContext.Session.GetString("UserId");
            if (!int.TryParse(userIdString, out int userId)) return null;
            return userId;
        }
    }
}
```

Код 1.3 – NoteController(Controllers)

Create.cshtml* | 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()">🔗 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 () {
            // Теги
            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.

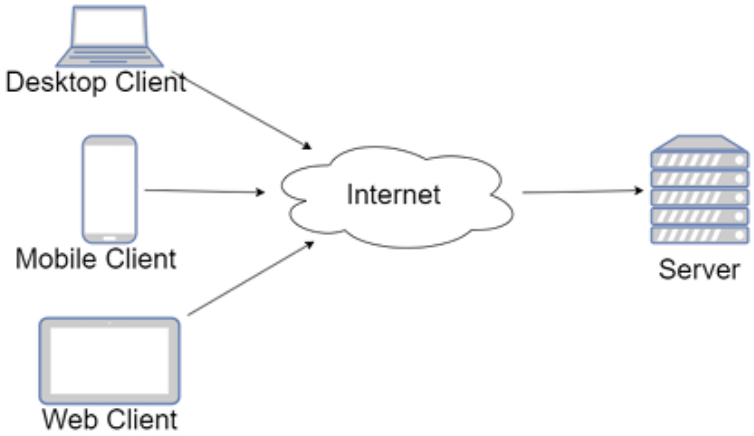


Рис.1 – Архітектура «клієнт-сервер»

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.

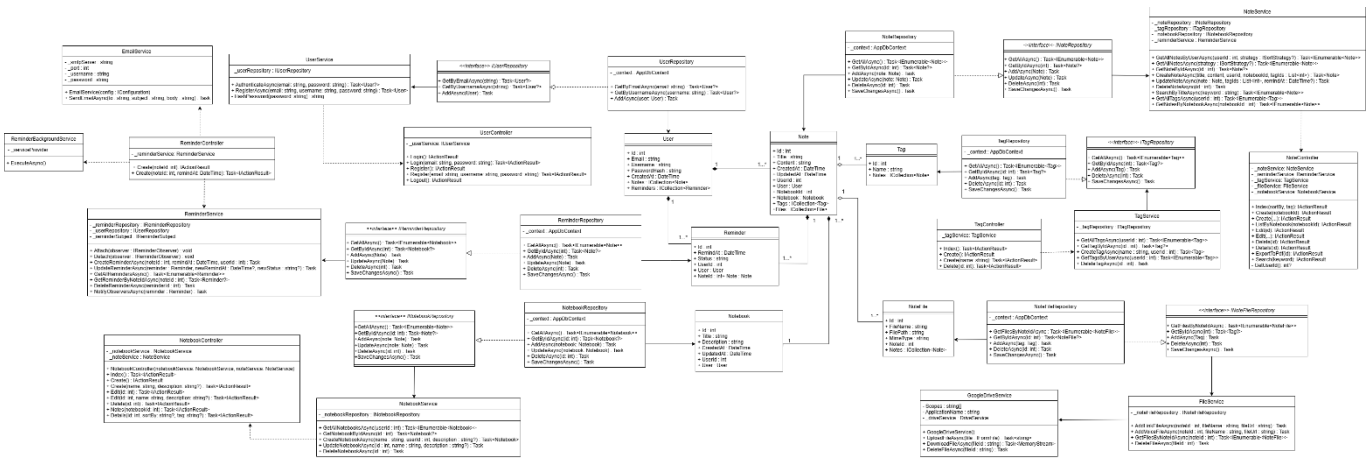


Рис.2 – Діаграма класів

4. Реалізувати взаємодію розподілених частин.

Index.cshtml* | NotebookRepository.cs | NotebookController.cs | NoteController.cs | Program.cs

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>
```

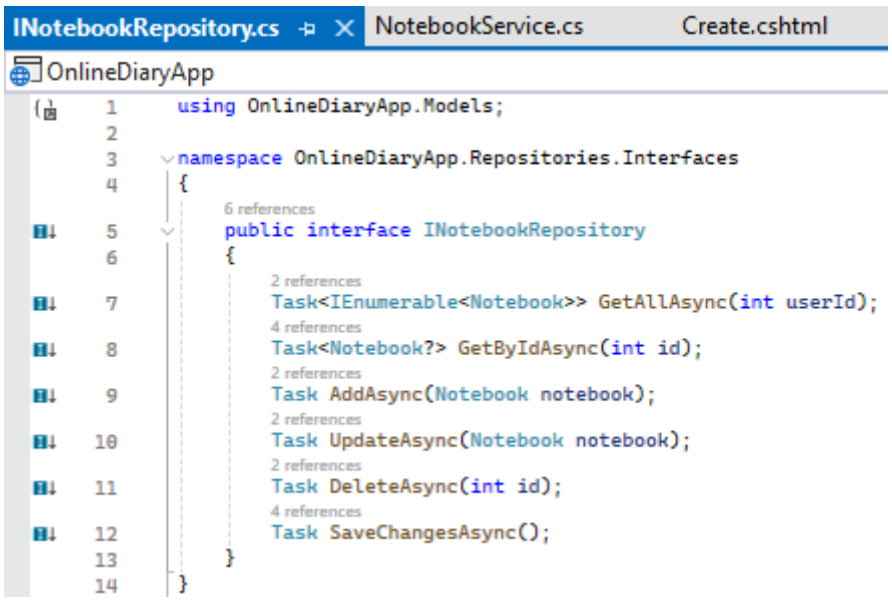
Код 3 – Index(Views.Notebook)

Create.cshtml | Index.cshtml* | NotebookRepository.cs | NotebookControll

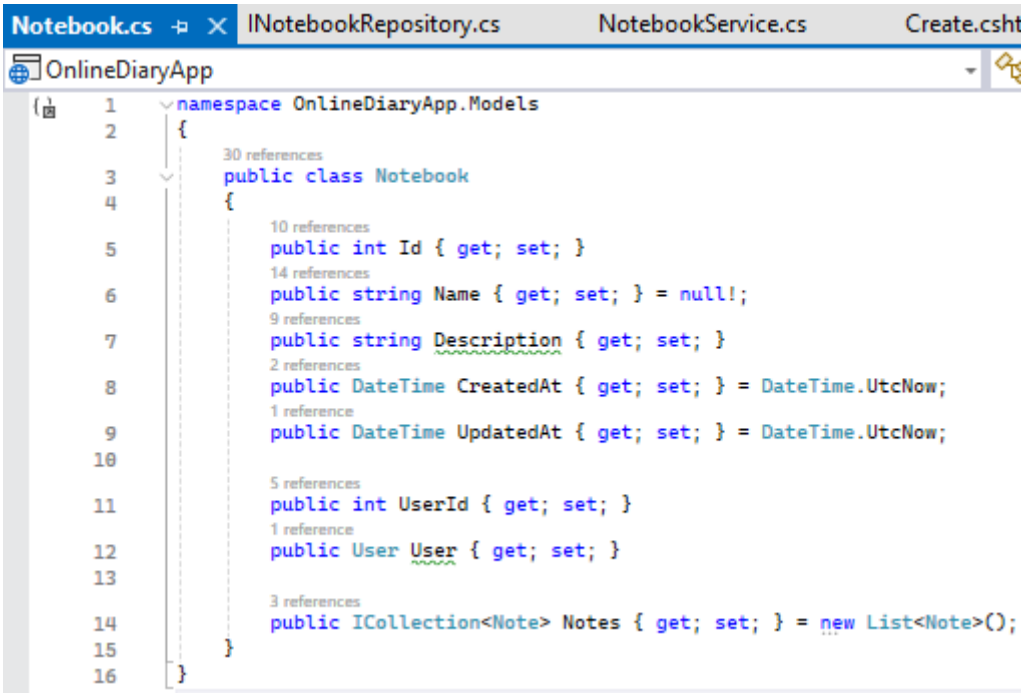
OnlineDiaryApp

```
@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)



Код 5 – INotebookRepository(Repoisitories.Interfaces)



Код 6 – Notebook(Models)

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

OnlineDiaryApp | OnlineDiaryApp.Services.NotebookServ

```
using OnlineDiaryApp.Models;
using OnlineDiaryApp.Repositories.Interfaces;

namespace OnlineDiaryApp.Services
{
    8 references
    public class NotebookService
    {
        private readonly INotebookRepository _notebookRepository;

        0 references
        public NotebookService(INotebookRepository notebookRepository)
        {
            _notebookRepository = notebookRepository;
        }

        3 references
        public async Task<IEnumerable<Notebook>> GetAllNotebooksAsync(int userId)
        {
            return await _notebookRepository.GetAllAsync(userId);
        }

        5 references
        public async Task<Notebook?> GetNotebookByIdAsync(int id)
        {
            return await _notebookRepository.GetByIdAsync(id);
        }

        1 reference
        public async Task<Notebook> CreateNotebookAsync(string name, int userId, string? description = null)
        {
            var notebook = new Notebook
            {
                Name = name,
                Description = description,
                UserId = userId,
                CreatedAt = DateTime.UtcNow
            };

            await _notebookRepository.AddAsync(notebook);
            await _notebookRepository.SaveChangesAsync();
            return notebook;
        }

        1 reference
        public async Task UpdateNotebookAsync(int id, string name, string? description)
        {
            var notebook = await _notebookRepository.GetByIdAsync(id);
            if (notebook == null) return;

            notebook.Name = name;
            notebook.Description = description;
            notebook.UpdatedAt = DateTime.UtcNow;

            await _notebookRepository.UpdateAsync(notebook);
            await _notebookRepository.SaveChangesAsync();
        }

        1 reference
        public async Task DeleteNotebookAsync(int id)
        {
            await _notebookRepository.DeleteAsync(id);
            await _notebookRepository.SaveChangesAsync();
        }
    }
}
```

Код 7 – NotebookService(Services)

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

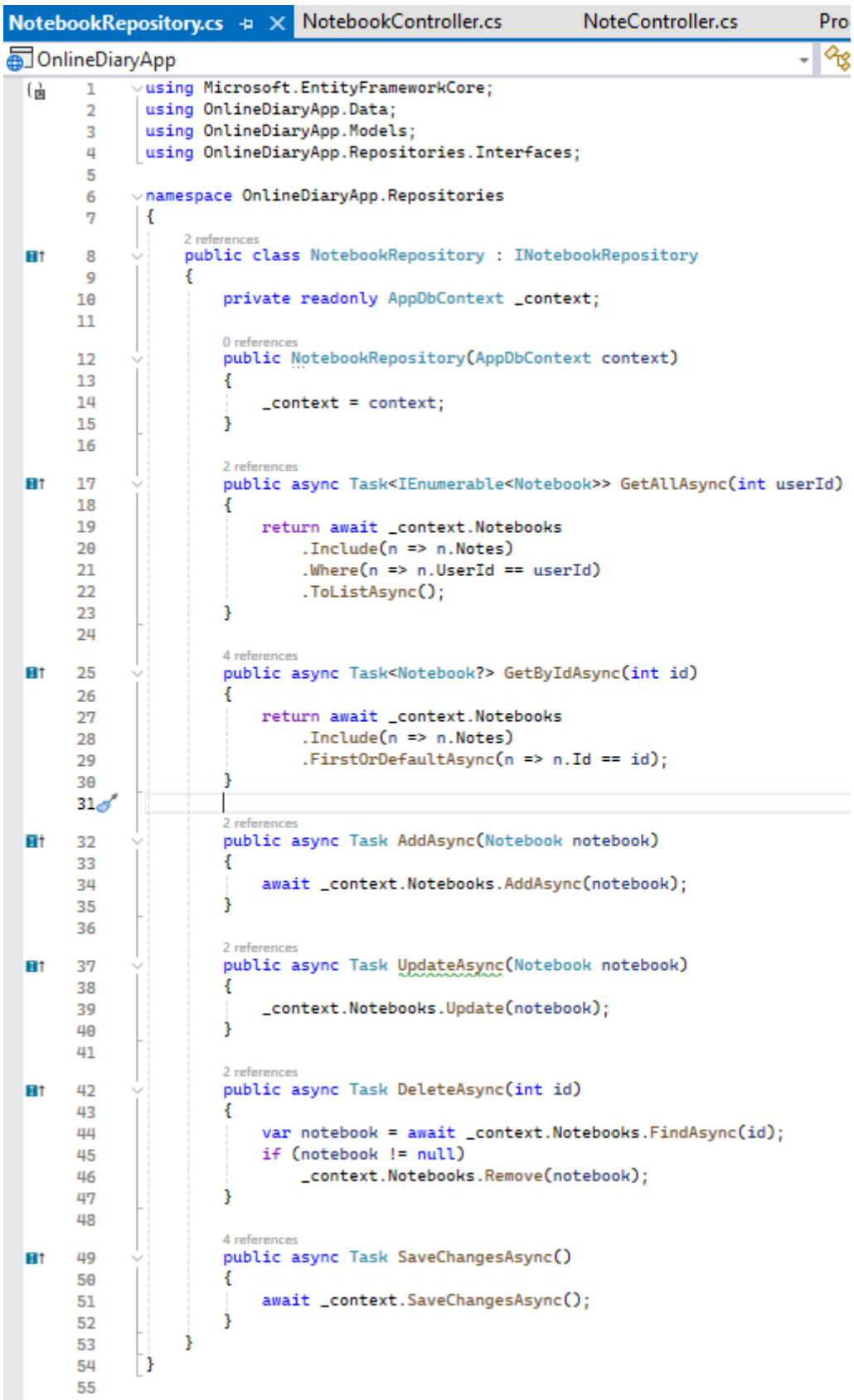
OnlineDiaryApp | OnlineDiaryApp.Contr

```
using Microsoft.AspNetCore.Mvc;
using OnlineDiaryApp.Services;
using OnlineDiaryApp.Composite;


namespace OnlineDiaryApp.Controllers
{
    1 reference
    public class NotebookController : Controller
    {
        private readonly NotebookService _notebookService;
        private readonly NoteService _noteService;

        0 references
        public NotebookController(NotebookService notebookService, NoteService noteService)
        {
            _notebookService = notebookService;
            _noteService = noteService;
        }

        0 references
        public async Task<IActionResult> Index()
        {
            var userIdString = HttpContext.Session.GetString("UserId");
            if (!int.TryParse(userIdString, out int userId))
                return RedirectToAction("Login", "User");

            var notebooks = await _notebookService.GetAllNotebooksAsync(userId);
            return View(notebooks);
        }

        [HttpGet]
        0 references
        public IActionResult Create()
        {
            return View();
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> Create(string name, string? description)
        {
            var userIdString = HttpContext.Session.GetString("UserId");
            if (!int.TryParse(userIdString, out int userId))
                return RedirectToAction("Login", "User");

            await _notebookService.CreateNotebookAsync(name, userId, description);
            return RedirectToAction("Index");
        }

        [HttpGet]
        0 references
        public async Task<IActionResult> Edit(int id)
        {
            var notebook = await _notebookService.GetNotebookByIdAsync(id);
            if (notebook == null) return NotFound();
            return View(notebook);
        }

        [HttpPost]
        0 references
        public async Task<IActionResult> Edit(int id, string name, string? description)
        {
            await _notebookService.UpdateNotebookAsync(id, name, description);
            return RedirectToAction("Index");
        }

        0 references
        public async Task<IActionResult> Delete(int id)
        {
            await _notebookService.DeleteNotebookAsync(id);
            return RedirectToAction("Index");
        }

        0 references
        public async Task<IActionResult> Notes(int notebookId)
        {
            var notebook = await _notebookService.GetNotebookByIdAsync(notebookId);
            if (notebook == null) return NotFound();

            var notes = await _noteService.GetNotesByNotebookAsync(notebook.Id);
            ViewBag.Notebook = notebook;
            return View(notes);
        }
```

Код 8.1 – NotebookController(Controllers)

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

Код 8.2 – NotebookController(Controllers)



Код 9 – NotebookRepository(Repoisitories.Implementations)

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

Висновки: У ході виконання лабораторної роботи було реалізовано вебзастосунок із класичною клієнт-серверною архітектурою, що забезпечує чітке розділення обов'язків між різними частинами системи. Серверна частина, побудована на основі ASP.NET Core MVC, виконує роль основного обчислювального центру — обробляє HTTP-запити, взаємодіє з базою даних через ORM Entity Framework та формує відповіді для клієнта. Клієнтська частина реалізована у вигляді вебінтерфейсу на Razor-сторінках і виконує роль тонкого клієнта, відповідального лише за відображення даних і надсилання запитів.

Посередницький шар, що включає сервіси, моделі та інтерфейси репозиторіїв, забезпечує логічну ізоляцію між рівнями програми та спрощує підтримку й розширення коду. Завдяки цьому система стала структурованою, модульною та зрозумілою для подальшого розвитку.

Таким чином, у результаті роботи було створено функціональний клієнт-серверний застосунок із розмежованими рівнями логіки, що демонструє принципи побудови сучасних вебсистем на платформі .NET.

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.