



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 №3
із дисципліни *«Технології розроблення програмного забезпечення»*
Тема: «Основи проектування розгортання»

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

Перевірив:
Мякий М.Ю.

Тема: Основи проектування розгортання.

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

Мета: Навчитися проєктувати діаграми розгортання та компонентів для системи що проєктується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.

Посилання на репозиторій з проєктом:

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

Хід роботи

1. Ознайомитись з короткими теоретичними відомостями.
2. Проаналізувати діаграми створені в попередній лабораторній роботі, а також тему системи та спроектувати діаграму розгортання використання відповідно до обраної теми лабораторного циклу.

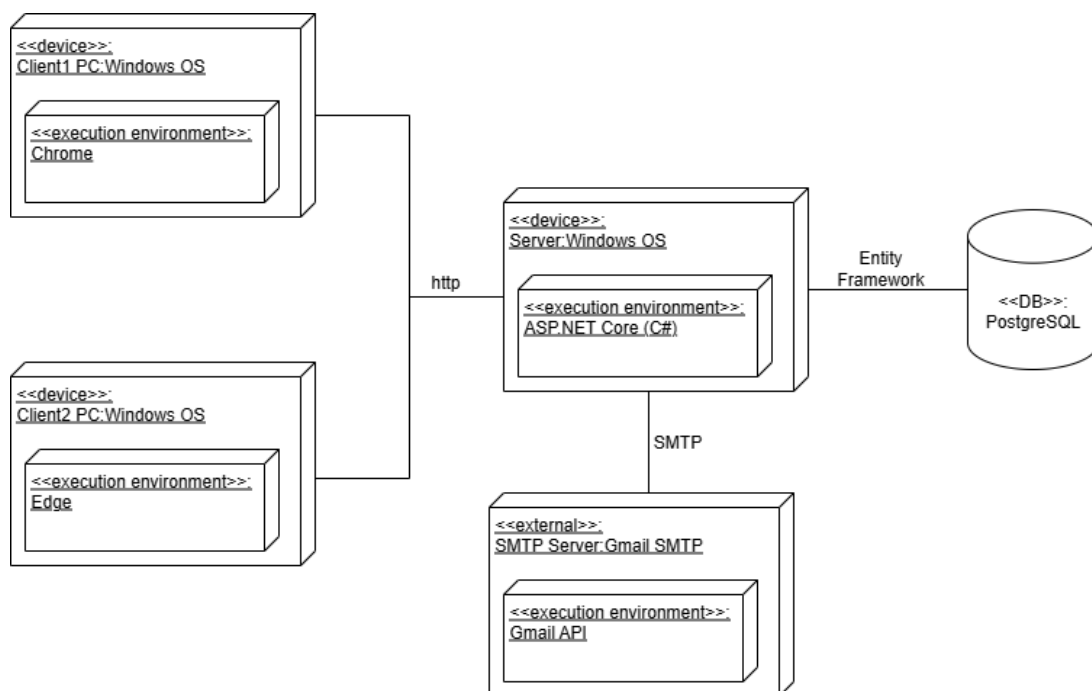


Рис.1 – Діаграма розгортання

На діаграмі розгортання зображено інфраструктуру веб-застосунку «OnlineDiary», де клієнти працюють на операційній системі Windows та взаємодіють із системою через браузер Chrome та Edge. Веб-застосунок розгорнутий на сервері під керуванням Windows OS і виконується у середовищі ASP.NET Core (C#). Для доступу до бази даних PostgreSQL застосунок використовує ORM-технологію Entity Framework. Для надсилання email-нагадувань система взаємодіє з зовнішнім SMTP-сервером (Gmail SMTP), використовуючи протокол SMTP.

3. Розробити діаграму компонентів для проєктованої системи.

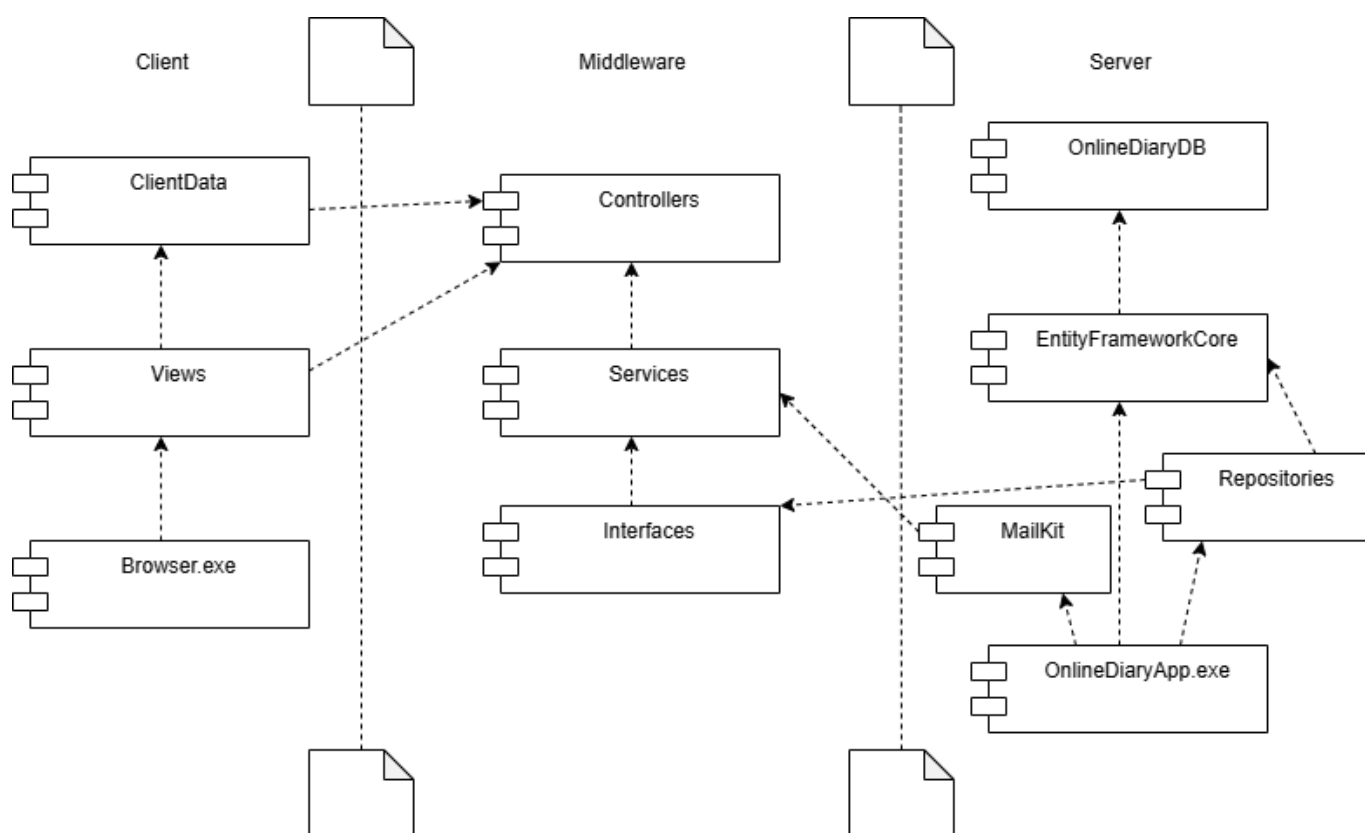


Рис.2 – Діаграма компонентів

На діаграмі зображено основні компоненти програмної системи. У лівій частині подані клієнтські файли запуску (.exe) та бібліотеки (.dll), які відповідають за інтерфейс користувача і взаємодію з веб-додатком. У правій частині показано серверні компоненти (.exe та .dll), які реалізують бізнес-логіку, роботу з базою даних через Entity Framework та інтеграцію з зовнішнім email-сервісом. У центральній частині розташовані модулі middleware, які містять загальні інтерфейси,

контракти API та сервіси, що забезпечують узгоджену взаємодію між клієнтом і сервером. Таким чином, клієнтська і серверна частини системи обмінюються інформацією через спільний набір інтерфейсів і даних, визначених у middleware.

4. Розробити як мінімум дві діаграми послідовностей для сценаріїв прописаних в попередній лабораторній роботі.

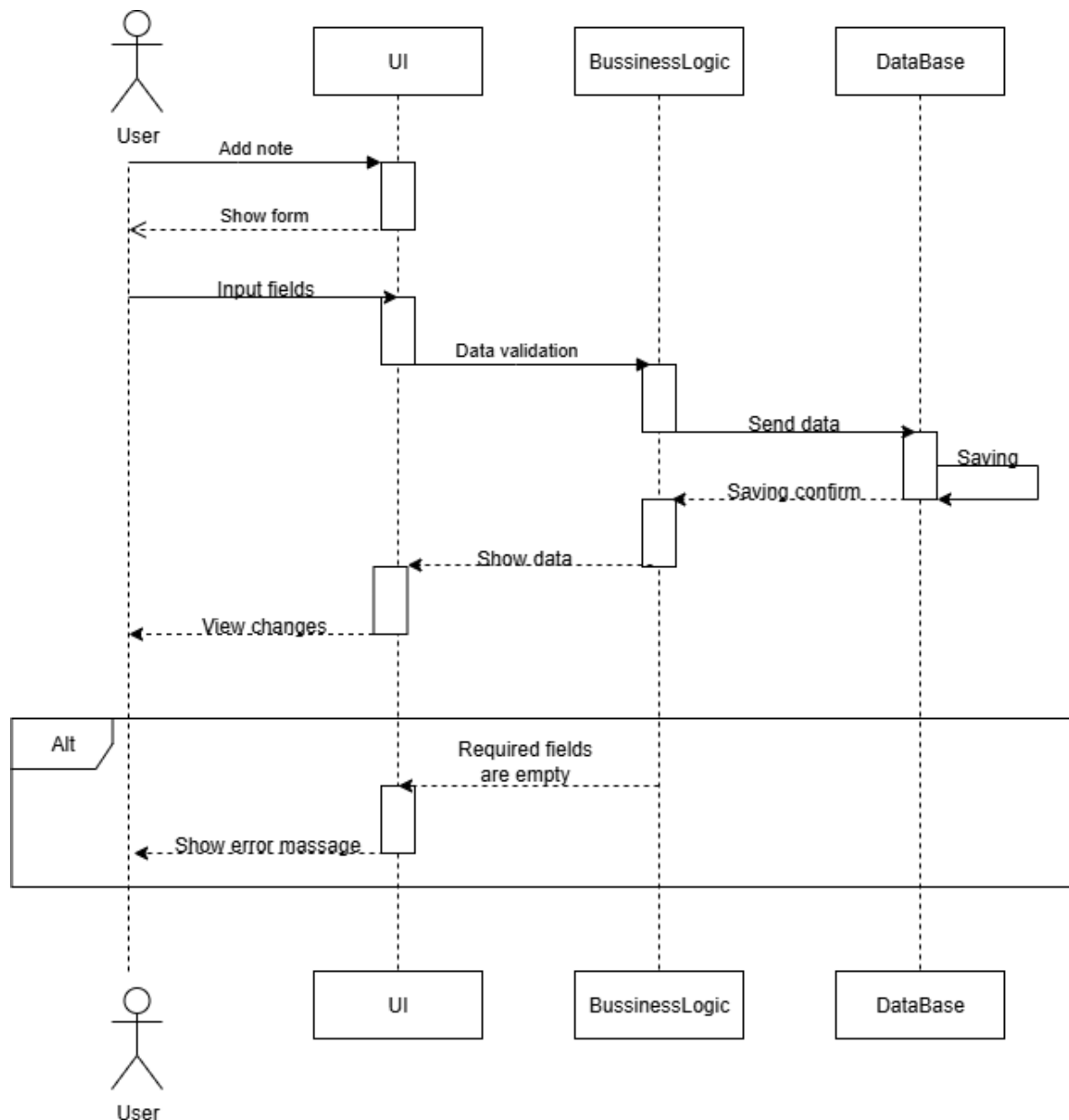


Рис.3 – Діаграма послідовності (сценарій 1)

Сценарій 1 – Створення нотатки

Передумови: Користувач авторизований у системі.

Постумови: У разі успішного виконання нова нотатка збережена у базі даних.

Взаємодіючі сторони: Користувач, Система.

Короткий опис: Варіант використання дозволяє користувачу створювати нові нотатки в особистому щоденнику.

Основний потік подій:

1. Користувач відкриває форму створення нотатки.
2. Система відображає форму для введення даних.
3. Користувач вводить заголовок і текст нотатки.
4. Користувач додає теги (за потреби).
5. Користувач натискає кнопку «Зберегти».
6. Система перевіряє коректність введених даних.
7. Система зберігає нотатку у базі даних.

Винятки:

- Виняток №1: Обов'язкові поля (наприклад, заголовок) залишені порожніми.
Система повідомляє користувача про помилку та пропонує виправити дані.

Примітки: Відсутні.

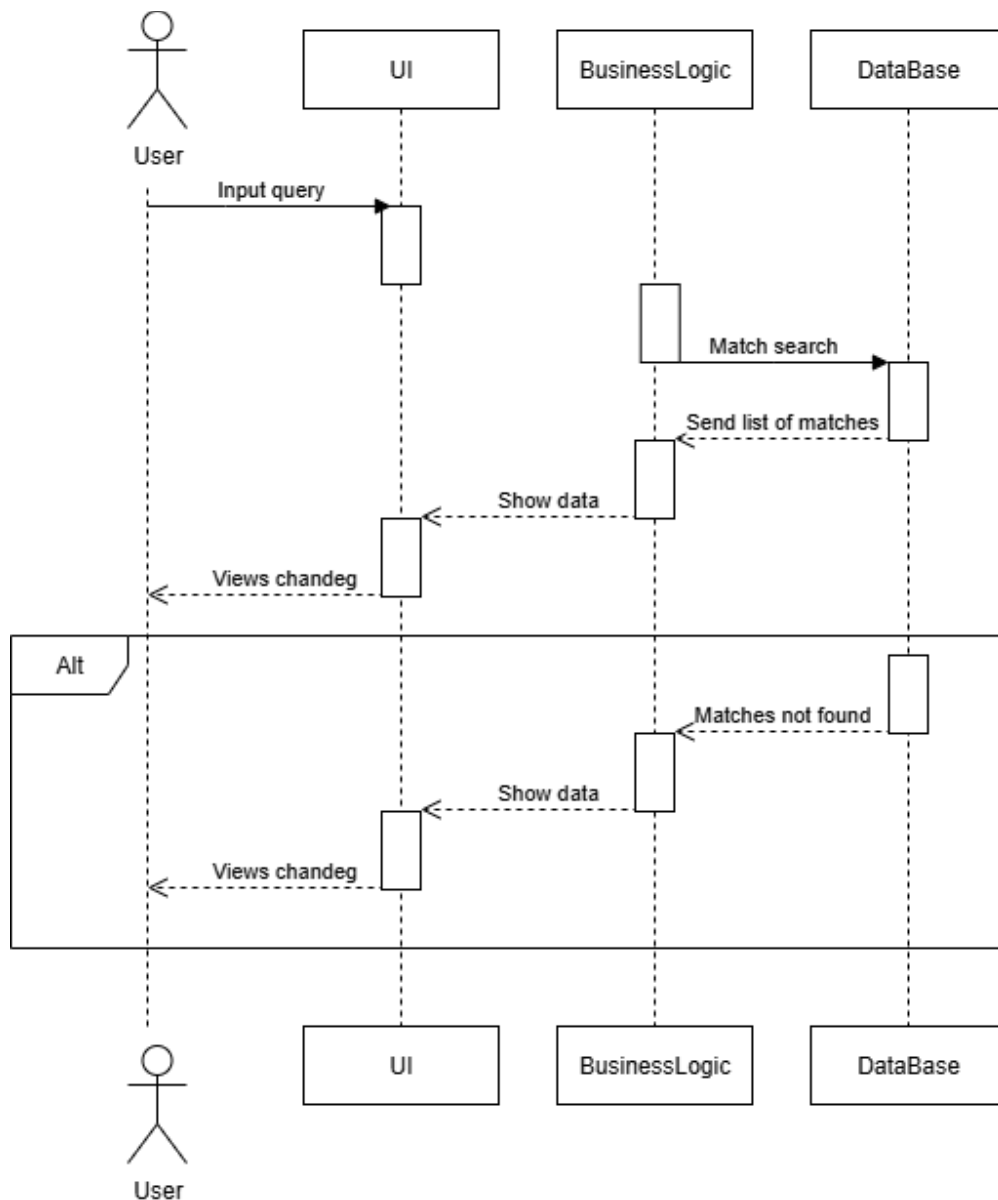


Рис.4 – Діаграма послідовності (сценарій 2)

Сценарій 2 – Пошук нотатки за ключовим словом

Передумови: У базі даних існують збережені нотатки.

Постумови: Користувач отримує список нотаток, що відповідають критерію пошуку.

Взаємодіючі сторони: Користувач, Система.

Короткий опис: Варіант використання дозволяє швидко знайти потрібну нотатку за введеним ключовим словом.

Основний потік подій:

1. Користувач вводить ключове слово у пошукове поле.

2. Користувач запускає пошук.
3. Система виконує пошук збігів у заголовках та тексті нотаток.
4. Система формує список результатів.
5. Система відображає список знайдених нотаток користувачу.

Винятки:

- Виняток №1: За заданим ключовим словом збігів не знайдено. Система повідомляє користувача, що результати відсутні.

Примітки: Можливе подальше уточнення пошуку за тегами чи датою.

5. На основі спроектованих діаграм розгортання та компонентів доопрацювати програмну частину системи. Реалізація системи, додатково до попередньої реалізації, повинна містити як мінімум дві візуальні форми. В системі вже повинен бути повністю реалізована архітектура (повний цикл роботи з даними від вводу на формі до збереження їх в БД і подальшій виборці з БД та відображенням на UI).

Реєстрація нового користувача в системі:

The image shows a registration form titled "Реєстрація" (Registration) on a light yellow background. It contains three input fields: "Email" with the value "sameliuk.anastasia@l111.kpi.ua", "Ім'я користувача" (Username) with the value "Anastasia", and "Пароль" (Password) with masked characters ".....". Below the fields is a green button labeled "Зареєструватися" (Register). At the bottom, there is a link that says "Вже є акаунт? [Увійти](#)" (Already have an account? [Login](#)).

Рис.5 – Форма реєстрації

	Id [PK] integer	Email text	PasswordHash text	CreatedAt timestamp with time zone	Username text
1	1	sameliuk.anastasia@gmail.c...	/qxT8INiQB...	2025-09-17 08:43:48.531162+00	Анастасія
2	2	sameliuk.anastasia@ill.kpi.ua	/qxT8INiQB...	2025-09-22 20:01:48.443598+00	Anastasia

Рис.6 – Додано користувач в таблиці Users в базі даних

Створення нотатки:

Створити нотатку

Заголовок

Текст

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Теги

термінове

особисте

особисте ×

Нагадування



Зберегти

Рис.7 – Форма створення нотатки



Рис.8 – Сторінка користувача з усіма його нотатками

	Id [PK] integer	Title text	Content text	CreatedAt timestamp with time zone	UpdatedAt timestamp with time zone	Userid integer
6	7	полрдотц	тпмлрицтшобдлът гшцзшо т гтшз	2025-09-17 16:10:58.542067+00	2025-09-17 16:10:58.542068+00	1
7	8	мпрольдз	вртчпте	2025-09-17 16:25:21.809022+00	2025-09-17 16:25:21.809023+00	1
8	9	вапрорл	іквеанпроло	2025-09-17 16:44:05.909585+00	2025-09-17 16:44:06.150533+00	1
9	10	авапро	арпол	2025-09-17 17:05:00.753936+00	2025-09-17 17:05:00.94786+00	1
10	11	укенгш	уцкенгшцз	2025-09-17 17:05:25.215023+00	2025-09-17 17:05:25.217784+00	1
11	12	rtyui	rtryuio	2025-09-17 17:24:51.650209+00	2025-09-17 17:24:51.801885+00	1
12	13	hjhxc	efreyr6ui6olgf	2025-09-17 17:25:22.906961+00	2025-09-17 17:25:22.910011+00	1
13	14	енгшцзц	цкеункгешнцгзшхц	2025-09-18 20:01:48.282695+00	2025-09-18 20:01:48.36814+00	1
14	15	ваипт	івавпарп	2025-09-18 20:16:36.982996+00	2025-09-18 20:16:37.10574+00	1
15	16	уцкуенгш	кенгшцз	2025-09-18 20:33:47.471041+00	2025-09-18 20:33:47.665094+00	1
16	17	ц4у56г	ц4ну5к	2025-09-18 20:41:17.56741+00	2025-09-18 20:41:17.651086+00	1
17	18	Нотатка 1	Lorem Ipsum is simply dummy text ...	2025-09-22 20:15:40.501675+00	2025-09-22 20:15:40.674654+00	1

Рис.9 – Додано нотатку в таблиці Notes в базі даних

	Id [PK] integer	RemindAt timestamp with time zone	Status text	Userid integer	Notelid integer
1	4	2025-09-26 23:15:00+00	active	1	7
2	5	2025-09-25 19:25:00+00	active	1	8
3	9	2025-09-17 23:25:00+00	sent	1	12
4	10	2025-09-17 16:26:00+00	sent	1	13
5	11	2025-09-19 23:01:00+00	sent	1	14
6	14	2025-09-18 23:42:00+00	sent	1	17
7	15	2025-09-24 22:15:00+00	active	1	18

Рис.10 – Додано нагадування до нотатки в таблиці Reminders в базі даних

UserController.cs* note.css Register.cshtml Index.cshtml notes.css auth

OnlineDiaryApp OnlineDiaryApp.Controllers.UserContr

```

1  using Microsoft.AspNetCore.Mvc;
2  using OnlineDiaryApp.Services;
3
4  namespace OnlineDiaryApp.Controllers
5  {
6      1 reference
7      public class UserController : Controller
8      {
9          private readonly IUserService _userService;
10
11          0 references
12          public UserController(IUserService userService)
13          {
14              _userService = userService;
15          }
16
17          [HttpGet]
18          0 references
19          public IActionResult Login()
20          {
21              return View();
22          }
23
24          // POST: Login
25          [HttpPost]
26          0 references
27          public async Task<IActionResult> Login(string email, string password)
28          {
29              var user = await _userService.AuthenticateAsync(email, password);
30              if (user == null)
31              {
32                  ViewBag.Error = "Невірний email або пароль";
33                  return View();
34              }
35
36              HttpContext.Session.SetString("UserId", user.Id.ToString());
37              HttpContext.Session.SetString("Username", user.Username);
38
39              return RedirectToAction("Index", "Home");
40          }
41
42          [HttpGet]
43          0 references
44          public IActionResult Register()
45          {
46              return View();
47          }
48
49          // POST: Register
50          [HttpPost]
51          0 references
52          public async Task<IActionResult> Register(string email, string username, string password)
53          {
54              try
55              {
56                  var user = await _userService.RegisterAsync(email, username, password);
57                  HttpContext.Session.SetString("UserId", user.Id.ToString());
58                  HttpContext.Session.SetString("Username", user.Username);
59
60                  return RedirectToAction("Index", "Home");
61              }
62              catch (Exception ex)
63              {
64                  ViewBag.Error = ex.Message;
65                  return View();
66              }
67          }
68
69          // Logout
70          0 references
71          public IActionResult Logout()
72          {
73              HttpContext.Session.Clear();
74              return RedirectToAction("Index", "Home");
75          }
76      }

```

Код 1 – UserController.cs (controllers)

NoteController.cs*
UserController.cs*
note.css
Register.cshtml
Index.cshtml

OnlineDiaryApp
OnlineDiaryApp.Controllers.NoteController

```

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

```

Код 2.1 – NoteController.cs (controllers)

```

78
79 [HttpPost]
80 0 references
81 public async Task<IActionResult> Edit(int id, string title, string content, List<int>? tagIds, DateTime? reminderDate)
82 {
83     var note = await _noteService.GetNoteByIdAsync(id);
84     if (note == null)
85         return NotFound();
86
87     note.Title = title;
88     note.Content = content;
89
90     await _noteService.UpdateNoteAsync(note, tagIds ?? new List<int>());
91
92     var existingReminder = await _reminderService.GetReminderByNoteIdAsync(note.Id);
93
94     if (reminderDate.HasValue)
95     {
96         if (existingReminder != null)
97         {
98             await _reminderService.UpdateReminderAsync(existingReminder, reminderDate.Value);
99         }
100         else
101         {
102             await _reminderService.CreateReminderAsync(note.Id, reminderDate.Value, note.UserId);
103         }
104     }
105
106     return RedirectToAction("Index");
107 }
108
109 0 references
110 public async Task<IActionResult> Delete(int id)
111 {
112     await _noteService.DeleteNoteAsync(id);
113     return RedirectToAction("Index");
114 }
115
116 0 references
117 public async Task<IActionResult> Search(string keyword)
118 {
119     var userIdString = HttpContext.Session.GetString("UserId");
120     if (!int.TryParse(userIdString, out int userId))
121         return View(new List<Note>());
122
123     var notes = await _noteService.SearchByTitleAsync(keyword);
124     notes = notes.Where(n => n.UserId == userId);
125     return View("Index", notes);
126 }

```

Код 2.2 – NoteController.cs (controllers)

```
ReminderController.cs*  NoteController.cs*  UserController.cs*  note.css
OnlineDiaryApp
OnlineDiaryApp.Controllers

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

Код 3 – ReminderController.cs (controllers)

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

Код 4 – TagController.cs (controllers)

NoteService.cs* TagController.cs* ReminderController.cs* NoteController.cs*

OnlineDiaryApp

OnlineDiaryApp.Services.ISortStrateg

```

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

```

Код 5.1 – NoteService.cs (services)

```

67
68
69
70
71
72
73
74
75
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
108
109
110
111
112
113
114
115
116

    var note = new Note
    {
        Title = title,
        Content = content,
        UserId = userId,
        Tags = tags,
        CreatedAt = utcNow
    };

    await _noteRepository.AddAsync(note);
    await _noteRepository.SaveChangesAsync();

    return note;
}

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

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

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

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

0 references
public async Task<IEnumerable<Tag>> GetAllTagsAsync()
{
    return await _tagRepository.GetAllAsync();
}
}

```

Код 5.2 – NoteService.cs (services)

UserService.cs* × NoteService.cs* TagController.cs* ReminderController.cs*

OnlineDiaryApp OnlineDiaryApp.Service

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

using OnlineDiaryApp.Models;

using OnlineDiaryApp.Repositories.Interfaces;

using System.Security.Cryptography;

using System.Text;

namespace OnlineDiaryApp.Services

{

4 references

public interface IUserService

{

2 references

Task<User?> AuthenticateAsync(string email, string password);

2 references

Task<User> RegisterAsync(string email, string username, string password);

}

2 references

public class UserService : IUserService

{

private readonly IUserRepository _userRepository;

0 references

public UserService(IUserRepository userRepository)

{

_userRepository = userRepository;

}

// Login

2 references

public async Task<User?> AuthenticateAsync(string email, string password)

{

var user = await _userRepository.GetByEmailAsync(email);

if (user == null) return null;

var hash = HashPassword(password);

return user.PasswordHash == hash ? user : null;

}

// Register

2 references

public async Task<User> RegisterAsync(string email, string username, string password)

{

var existingUser = await _userRepository.GetByEmailAsync(email);

if (existingUser != null)

throw new Exception("Користувач з таким email вже існує");

var user = new User

{

Email = email,

Username = username,

PasswordHash = HashPassword(password),

CreatedAt = DateTime.UtcNow

};

await _userRepository.AddAsync(user);

return user;

}

// Hash function

2 references

private string HashPassword(string password)

{

using var sha = SHA256.Create();

var bytes = sha.ComputeHash(Encoding.UTF8.GetBytes(password));

return Convert.ToBase64String(bytes);

}

}

Код 6 – UserService.cs (services)

ReminderService.cs*
UserService.cs*
NoteService.cs*
TagController.cs*
ReminderCont

OnlineDiaryApp
OnlineDiaryApp.Services.ReminderService

```

1  using OnlineDiaryApp.Models;
2
3  namespace OnlineDiaryApp.Services
4  {
5      7 references
6      public class ReminderService
7      {
8          private readonly IReminderRepository _reminderRepository;
9
10         0 references
11         public ReminderService(IReminderRepository reminderRepository)
12         {
13             _reminderRepository = reminderRepository;
14         }
15
16         3 references
17         public async Task CreateReminderAsync(int noteId, DateTime remindAt, int userId)
18         {
19             var utcRemindAt = DateTime.SpecifyKind(remindAt, DateTimeKind.Utc);
20
21             var reminder = new Reminder
22             {
23                 NoteId = noteId,
24                 UserId = userId,
25                 RemindAt = utcRemindAt,
26                 Status = "active"
27             };
28
29             await _reminderRepository.AddAsync(reminder);
30             await _reminderRepository.SaveChangesAsync();
31         }
32
33         1 reference
34         public async Task<IEnumerable<Reminder>> GetAllRemindersAsync()
35         {
36             return await _reminderRepository.GetAllAsync();
37         }
38
39         2 references
40         public async Task<Reminder?> GetReminderByNoteIdAsync(int noteId)
41         {
42             return await _reminderRepository.GetByNoteIdAsync(noteId);
43         }
44
45         2 references
46         public async Task UpdateReminderAsync(Reminder reminder, DateTime? newRemindAt = null, string? newStatus = null)
47         {
48             if (newRemindAt.HasValue)
49                 reminder.RemindAt = DateTime.SpecifyKind(newRemindAt.Value, DateTimeKind.Utc);
50
51             if (!string.IsNullOrEmpty(newStatus))
52                 reminder.Status = newStatus;
53
54             await _reminderRepository.UpdateAsync(reminder);
55             await _reminderRepository.SaveChangesAsync();
56         }
57
58         0 references
59         public async Task DeleteReminderAsync(int reminderId)
60         {
61             var reminder = await _reminderRepository.GetByIdAsync(reminderId);
62             if (reminder != null)
63             {
64                 await _reminderRepository.DeleteAsync(reminder.Id);
65                 await _reminderRepository.SaveChangesAsync();
66             }
67         }
68     }
69 }

```

Код 7 – ReminderService.cs (services)

```

Create.cshtml  -> X  ReminderService.cs*  UserService.cs*  NoteService
C# OnlineDiaryApp  AspNetCore.View

@{
    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">
            <div class="form-group">
                <label>Заголовок</label>
                <input type="text" name="title" class="form-control" required />
            </div>

            <div class="form-group">
                <label>Текст</label>
                <textarea name="content" class="form-control" required></textarea>
            </div>

            <div class="form-group">
                <label>Тег</label>
                <select id="tagSelect" name="tagIds" class="form-control" multiple>
                    @if (ViewBag.Tags != null)
                    {
                        foreach (var tag in ViewBag.Tags)
                        {
                            <option value="@tag.Id">@tag.Name</option>
                        }
                    }
                </select>
                <div id="selectedTags" class="selected-tags"></div>
            </div>

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

            <button type="submit" class="btn-save">Зберегти</button>
        </form>
    </div>
</div>

<script>
    const tagSelect = document.getElementById("tagSelect");
    const selectedTagsContainer = document.getElementById("selectedTags");

    function renderSelectedTags() {
        selectedTagsContainer.innerHTML = "";

        [...tagSelect.selectedOptions].forEach(option => {
            const chip = document.createElement("div");
            chip.classList.add("tag-chip");
            chip.textContent = option.text;

            const removeBtn = document.createElement("span");
            removeBtn.classList.add("remove");
            removeBtn.innerHTML = "&times;";

            removeBtn.addEventListener("click", () => {
                option.selected = false;
                renderSelectedTags();
            });

            chip.appendChild(removeBtn);
            selectedTagsContainer.appendChild(chip);
        });
    }

    tagSelect.addEventListener("change", renderSelectedTags);
</script>

```

Код 8 – Create.cshtml (Views.Note)

```

Index.cshtml  Create.cshtml  ReminderService.cs*  UserService.cs*  NoteService.cs*
OnlineDiaryApp
@model IEnumerable<OnlineDiaryApp.Models.Note>

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

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

<a class="btn-create" asp-action="Create">+ Створити нову нотатку</a>

<div class="notes-grid">
    @foreach (var note in Model)
    {
        <div class="note-card">
            <div class="note-title">@note.Title</div>
            <div class="note-date">@note.CreatedAt.ToString("g")</div>

            <div class="note-tags">
                @foreach (var tag in note.Tags)
                {
                    <span class="note-tag">@tag.Name</span>
                }
            </div>

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

```

Код 9 – Index.cshtml (Views.Note)

```

Login.cshtml  Index.cshtml  Create.cshtml  ReminderService.cs*  UserService.cs*  NoteSe
OnlineDiaryApp
@{
    ViewData["Title"] = "Login";
}

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

<div class="auth-page">
    <div class="auth-container login-container">
        <h2>Увійти</h2>

        <form asp-action="Login" method="post">
            <div class="form-group">
                <label for="email">Email</label>
                <input type="email" class="form-control" id="email" name="email" required />
            </div>
            <div class="form-group">
                <label for="password">Пароль</label>
                <input type="password" class="form-control" id="password" name="password" required />
            </div>
            <button type="submit" class="btn-note">Увійти</button>
        </form>

        <p>
            Ще немає акаунту?
            <a asp-controller="User" asp-action="Register">Зареєструватися</a>
        </p>

        @if (!ViewData.ModelState.IsValid)
        {
            <div class="text-danger">
                @foreach (var error in ViewData.ModelState.Values.SelectMany(v => v.Errors))
                {
                    <p>@error.ErrorMessage</p>
                }
            </div>
        }
    </div>
</div>

```

Код 10 – Login.cshtml (Views.User)

```
Register.cshtml Login.cshtml Index.cshtml Create.cshtml ReminderService.cs*
C# OnlineDiaryApp
@model OnlineDiaryApp.Models.User

@{
    ViewData["Title"] = "Register";
}

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

<div class="auth-page">
    <div class="auth-container register-container">
        <h2>Рєєстрація</h2>

        <form asp-action="Register" method="post">
            <div class="form-group">
                <label for="email">Email</label>
                <input asp-for="Email" class="form-control" required />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label for="username">Ім'я користувача</label>
                <input asp-for="Username" class="form-control" required />
                <span asp-validation-for="Username" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label for="password">Пароль</label>
                <input type="password" class="form-control" name="password" required />
            </div>
            <button type="submit" class="btn-register">Зареєструватися</button>
        </form>

        <p>
            Вже є акаунт?
            <a asp-controller="User" asp-action="Login">Увійти</a>
        </p>
    </div>
</div>
```

Код 11 – Register.cshtml (Views.User)

- Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму розгортання з описом, діаграму компонентів системи з описом, діаграми послідовностей, а також вихідний код системи, який було додано в цій лабораторній роботі.

Висновки: У ході виконання лабораторної роботи було досягнуто поставлену мету: набуті навички проєктування діаграм розгортання та компонентів для розроблюваної системи, що дозволяє відобразити фізичну структуру системи, розміщення програмних модулів на вузлах та їх взаємозв'язки. Також було

опановано розробку діаграм взаємодії, зокрема діаграм послідовностей, на основі сценаріїв, створених у попередній лабораторній роботі. Це сприяло глибшому розумінню логіки взаємодії об'єктів системи та послідовності виконання операцій, що є важливим етапом у моделюванні програмних систем і підготовці до їхньої реалізації.

Відповіді на контрольні питання:

1. Що собою становить діаграма розгортання?

Діаграма розгортання (Deployment Diagram) показує фізичну архітектуру системи: як програмне забезпечення розміщується на апаратних вузлах, які зв'язки існують між вузлами, а також як компоненти системи розташовані на цих вузлах. Вона корисна для аналізу інфраструктури, серверів, пристроїв і мережевих з'єднань.

2. Які бувають види вузлів на діаграмі розгортання?

- Фізичні вузли (Node): апаратні пристрої або сервери.
- Програмні вузли: середовища виконання (наприклад, JVM, веб-сервер).
- Вузли баз даних: сервери баз даних або сховища.
- Компоненти всередині вузлів: програмні модулі або артефакти, що розміщені на фізичних вузлах.

3. Які бувають зв'язки на діаграмі розгортання?

- Зв'язок асоціації (Association): показує мережеве з'єднання між вузлами.
- Залежність (Dependency): вказує, що один вузол використовує інший.
- Комунікаційний канал (Communication path): фізичне з'єднання між вузлами для обміну даними.

4. Які елементи присутні на діаграмі компонентів?

- Компоненти (Component): незалежні модулі програмного забезпечення з чітко визначеними інтерфейсами.

- Інтерфейси (Interface): точки взаємодії компонентів.
- Пакети (Package): групування компонентів.
- Актори (Actor) або зовнішні системи: можуть бути присутніми для позначення джерел або отримувачів даних.

5. Що становлять собою зв'язки на діаграмі компонентів?

- Залежність (Dependency): один компонент залежить від іншого.
- Асоціація (Association): компоненти взаємодіють між собою.
- Реалізація інтерфейсу (Realization): компонент реалізує певний інтерфейс.
- Інформаційний потік (Information flow): показує рух даних між компонентами.

6. Які бувають види діаграм взаємодії?

- Діаграма послідовностей (Sequence Diagram).
- Діаграма комунікацій (Communication / Collaboration Diagram).
- Діаграма часових подій (Timing Diagram).
- Діаграма взаємодії (Interaction Overview Diagram).

7. Для чого призначена діаграма послідовностей?

Діаграма послідовностей описує динамічну взаємодію між об'єктами у вигляді послідовності повідомлень у часі. Вона використовується для аналізу логіки сценаріїв та послідовності виконання операцій у системі.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

- Об'єкти / Лайнери життя (Lifeline): учасники взаємодії.
- Повідомлення (Message): обмін інформацією між об'єктами.
- Активації (Activation): період, коли об'єкт активний.
- Умовні гілки / Цикли (alt, loop, opt): відображення логічних конструкцій.
- Примітки (Note): додаткова інформація про процеси.

9. Як діаграми послідовностей пов'язані з діаграмами варіантів використання?

Діаграми послідовностей деталізують сценарії, які описані у діаграмах варіантів використання (Use Case Diagram). Кожен варіант використання можна розкласти на кроки у вигляді повідомлень між об'єктами на діаграмі послідовностей.

10. Як діаграми послідовностей пов'язані з діаграмами класів?

Діаграми послідовностей показують об'єкти та їх взаємодію у часі, тоді як діаграми класів описують структуру цих об'єктів (класи, атрибути, методи). Іншими словами, діаграма класів визначає що існує, а діаграма послідовностей — як ці об'єкти взаємодіють.