Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 3

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Основи проектування розгортання»

«Менеджер завдань»

Виконав:
студент групи - ІА-32
Воробйов Кирило Андрійович

Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Основи проектування розгортання.

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.

Тема проєкту: Менеджер завдань (To Do Manager)

Патерни: Strategy, Command, Observer, Mediator, Composite, Client-Server

Опис: Додаток повинен мати можливість створювати, редагувати та видаляти завдання, забезпечувати гнучку фільтрацію та сортування (Strategy), та обробляти дії користувача (Command). Система має підтримувати ієрархічну структуру (завдання/підзадачі або проєкт/завдання) (Composite), забезпечувати автоматичне оновлення візуальних компонентів при зміні даних (Observer) та керувати взаємодією елементів інтерфейсу (Mediator). Зберігання та синхронізація завдань реалізується через клієнт-серверний зв'язок (Client-Server).

Зміст

Теоретичні відомості

Діаграми компонентів — показують модулі (компоненти), їхні залежності і артефакти (.jar, таблиці, html). Використовуються для логічного/фізичного поділу системи.

Діаграма компонентів UML є представленням проєктованої системи, розбитої на окремі модулі. Залежно від способу поділу на модулі розрізняють три види діаграм компонентів:

- логічні;
- фізичні;
- виконувані.

Коли використовують логічне розбиття на компоненти, то у такому разі проєктовану систему віртуально уявляють як набір самостійних, автономних модулів (компонентів), що взаємодіють між собою.

Діаграми розгортання — показують фізичні вузли (devices) і середовища виконання (execution environments), на яких розгортаються артефакти; зв’язки зазначають протоколи (HTTP, SQL/ODBC).

Діаграми розгортання представляють фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення.

Головними елементами діаграми є вузли, пов'язані інформаційними шляхами. Вузол (node) – це те, що може містити програмне забезпечення. Вузли бувають двох типів. Пристрій (device) – це фізичне обладнання: комп'ютер або пристрій, пов'язаний із системою.

Середовище виконання (execution environment) – це програмне забезпечення, яке саме може включати інше програмне забезпечення, наприклад операційну систему або процес-контейнер (наприклад, вебсервер).

Діаграми послідовностей — моделюють часову послідовність повідомлень між акторами/об’єктами (HTTP POST/GET: Browser – Server DB – Browser).

Діаграма послідовностей (Sequence Diagram) – це один із типів діаграм у моделюванні UML (Unified Modeling Language), який використовується для моделювання взаємодії між об'єктами системи у певній послідовності часу. Вона відображає, як об'єкти обмінюються повідомленнями, показуючи порядок і логіку виконання операцій. Діаграма складається з таких основних елементів:

Актори (Actors): Зазвичай позначаються піктограмами або назвами. Це користувачі чи інші системи, які взаємодіють із системою. Актори можуть бути

Об'єкти або класи: Розміщуються горизонтально на діаграмі. Вони позначаються прямокутниками з іменем об'єкта або класу під прямокутником.

Кожен об'єкт має «життєвий цикл», який представлений вертикальною пунктирною лінією (лінія життя).

Повідомлення: Це лінії зі стрілками, які з'єднують об'єкти. Вони показують передачу повідомлень чи виклик методів. Стрілка може бути синхронною (звичайна стрілка) або асинхронною (лінія з відкритим трикутником) та з пунктирною лінією, що показує повернення результату.

Активності: Вказують періоди, протягом яких об'єкт виконує певну дію. На діаграмі це позначається прямокутником, накладеним на лінію життя.

Контрольні структури: Використовуються для відображення умов, циклів або альтернативних сценаріїв. Наприклад, блоки "alt" (альтернатива) або "loop" (цикл).

Хід роботи

Завдання:

- Ознайомитись з короткими теоретичними відомостями.
- Проаналізувати діаграми створені в попередній лабораторній роботі а також тему системи та спроєктувати діаграму розгортання використання відповідно до обраної теми лабораторного циклу.
- Розробити діаграму компонентів для проєктованої системи.
- Розробити діаграму розгортання для проєктованої системи.
- Розробити як мінімум дві діаграми послідовностей для сценаріїв прописаних в попередній лабораторній роботі.
- На основі спроєктованих 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).
- 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

Діаграма розгортання системи (Фінальний вигляд)

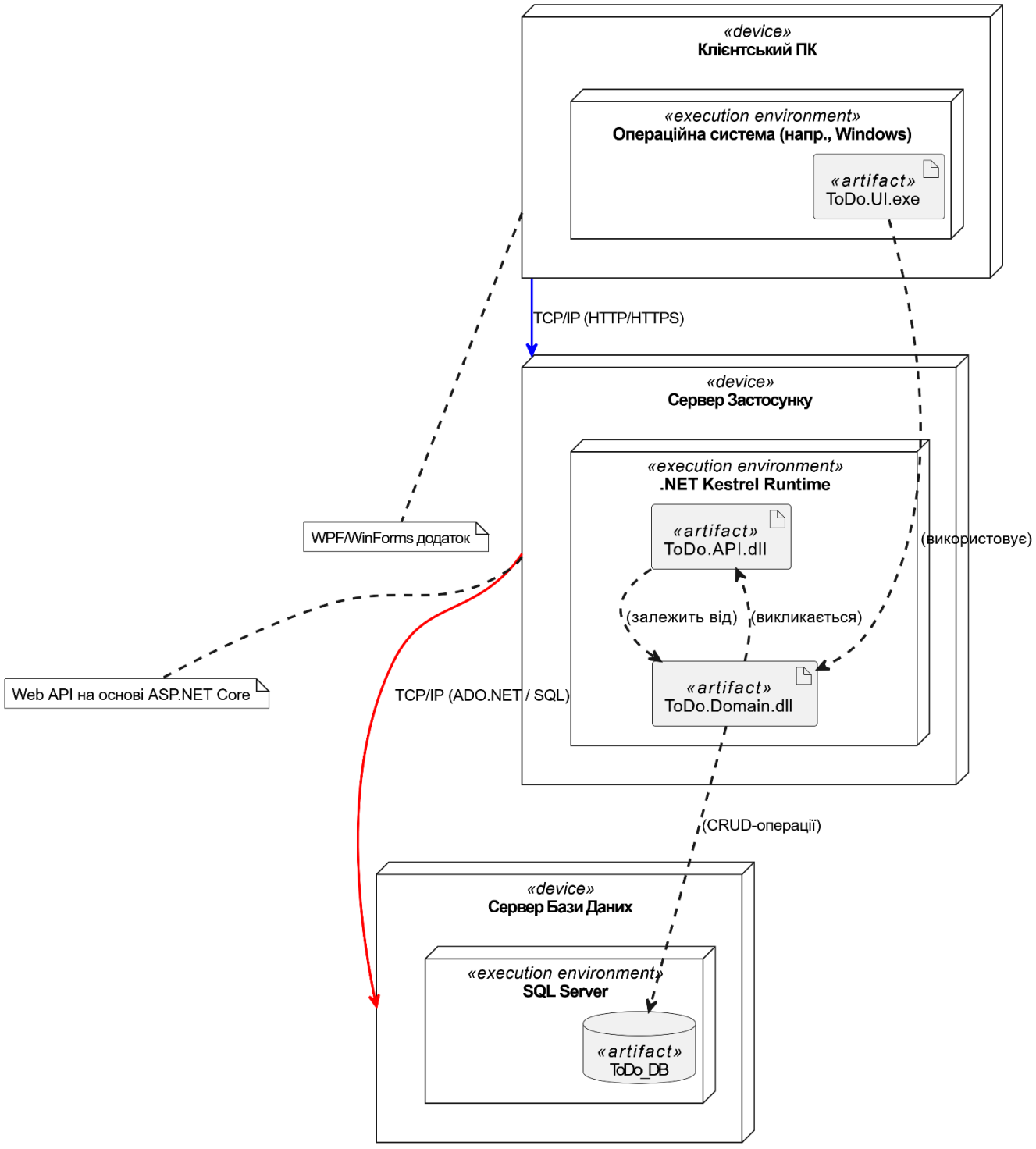


Рис. 1. Діаграма розгортання системи
(фінальний вигляд)

Діаграма розгортання демонструє фінальну 3-рівневу архітектуру системи. Клієнтський ПК містить виконуваний файл ToDo.UI.exe (WPF/WinForms), який взаємодіє з **Сервером Застосунку** через протоколи TCP/IP (зазвичай HTTP/HTTPS). Сервер застосунку (API) містить бізнес-логіку (ToDo.Domain.dll) та взаємодіє із **Сервером Бази Даних** через протокол SQL/ADO.NET для зберігання даних.

Діаграма розгортання системи (На момент 3-ї лаби)

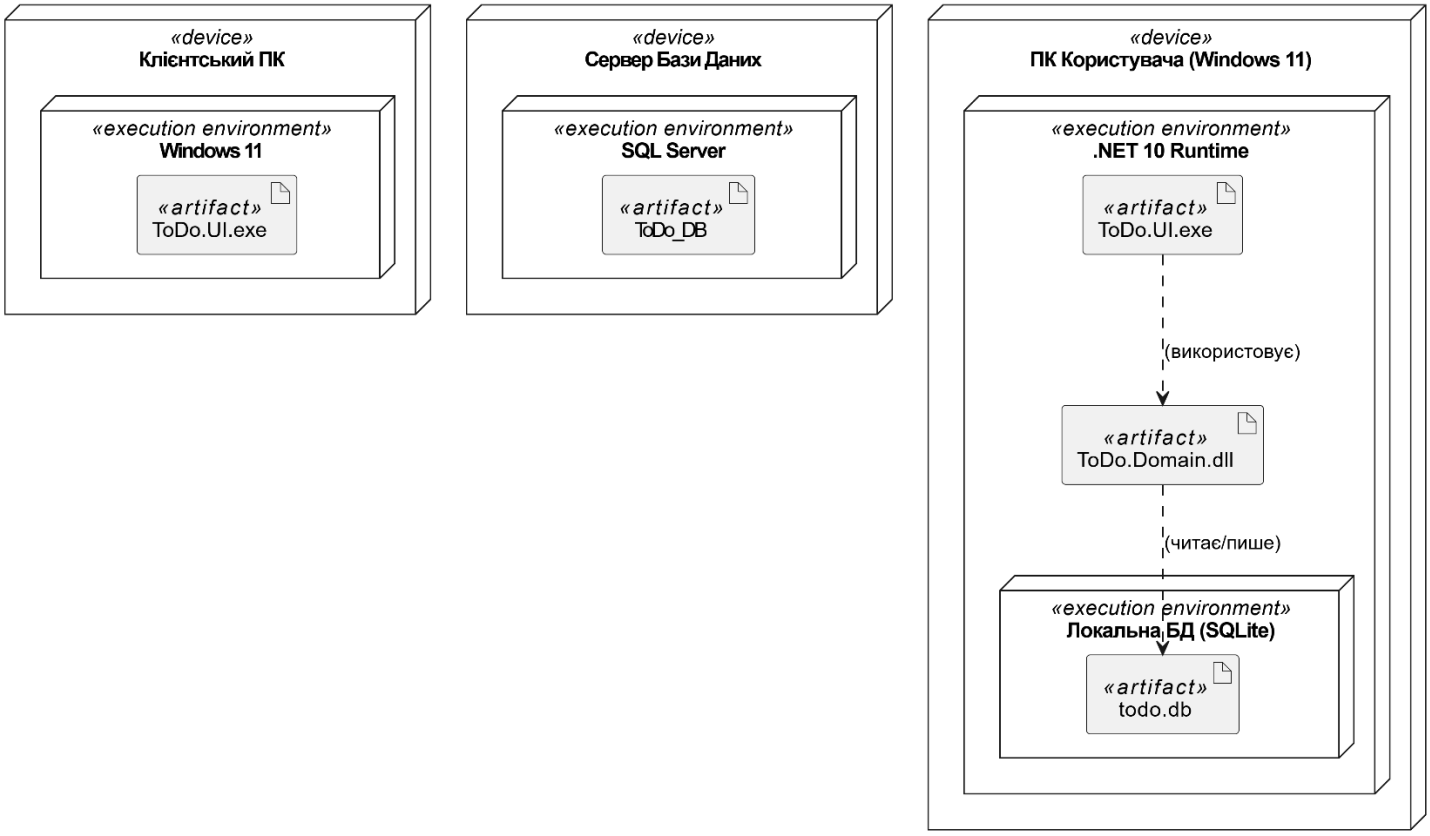


Рис. 2. Діаграма розгортання системи (на момент 2-ї лаби)

ПК Користувача (device): Це фізичний комп'ютер, на якому працює користувач.

.NET 8 Runtime (execution environment): Це середовище виконання, необхідне для запуску C# додатку.

Артефакти (artifacts): Це наші файли:

- ToDo.UI.exe: Виконуваний файл візуальної частини (форми).
- ToDo.Domain.dll: Наша бібліотека класів з ЛР 2 (моделі, репозиторій).
- todo.db: Файл бази даних SQLite.

Діаграма компонентів

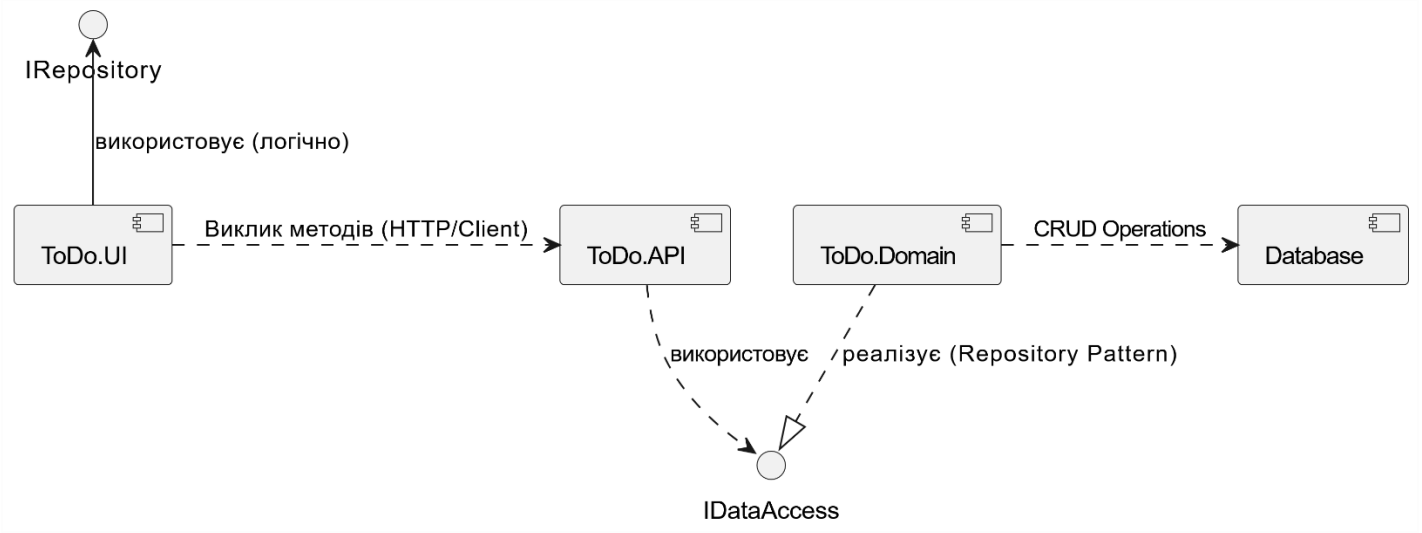


Рис. 2. Діаграма компонентів

Діаграма компонентів демонструє логічну структуру системи, розділяючи її на три основні функціональні компоненти, що взаємодіють через чітко визначені інтерфейси, відповідно до трирівневої архітектури (3-Tier Architecture):

- Presentation Tier (Рівень подання) – ToDo.UI: Відповідає за відображення даних та обробку введення користувача. Він використовує (залежить від) інтерфейс IRepository, не знаючи, яка конкретна реалізація знаходиться за ним.

- Business/Data Access Tier (Рівень логіки та доступу) – ToDo.Repositories / ToDo.Data: Включає ядро бізнес-логіки. Компонент ToDo.Repositories надає інтерфейс IRepository (який реалізує GenericRepository) і використовує компонент ToDo.Data (який містить ToDoDbContext) для роботи з Entity Framework.

- Data Tier (Рівень даних) – ToDo_DB: Представляє базу даних, яка забезпечує стійке зберігання інформації (таблиці Tasks, Projects, Users).

Зв'язки ілюструють слабку зв'язаність: UI залежить лише від абстракції (IRepository), а не від конкретного типу бази даних.

Діаграма послідовності

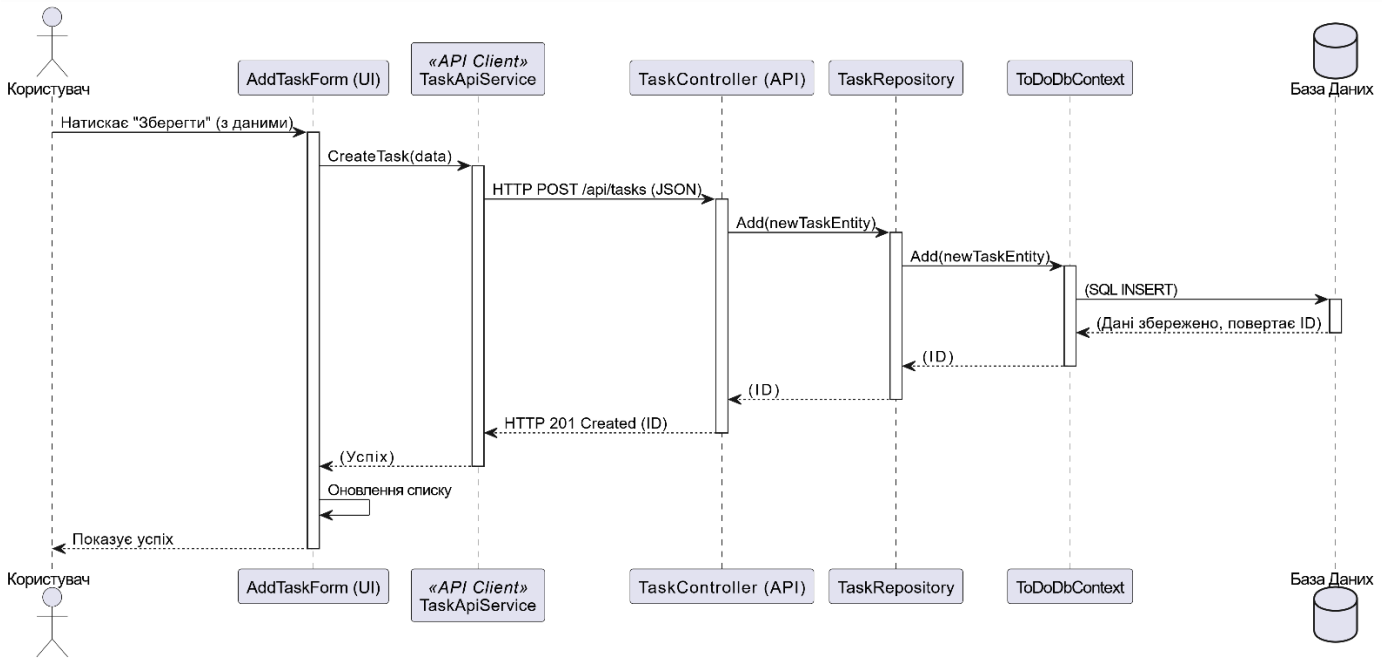


Рис. 3. Діаграма послідовності «Створення нового завдання»

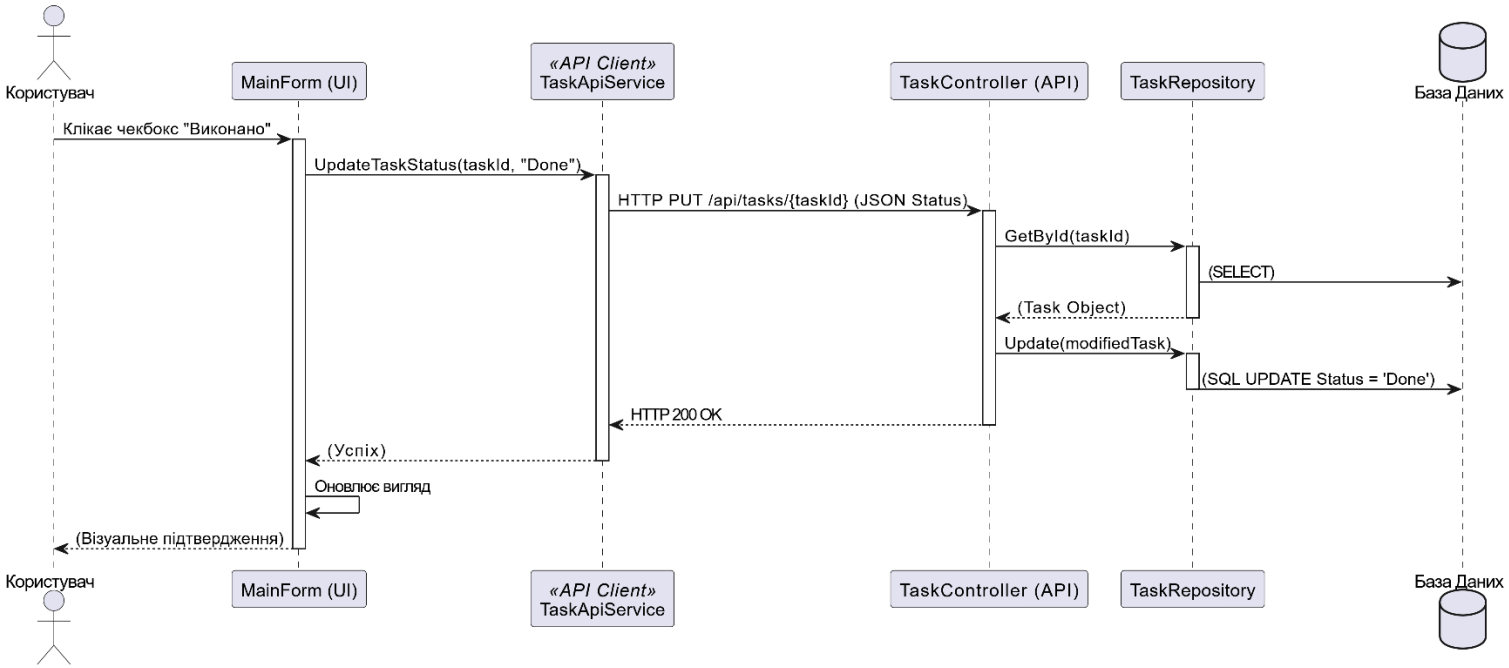


Рис. 3. Діаграма послідовності «Відзначити завдання як виконане»

Діаграма послідовностей описує часову послідовність взаємодії об'єктів для реалізації варіанта використання **«Створення нового завдання»**.

Вона включає такі об'єкти (лінії життя):

- **:Користувач** (Actor): Ініціює процес.
- **:MainForm** (Presentation): Головне вікно, яке керує процесом.
- **:AddTaskForm** (Presentation): Вікно для введення даних завдання.
- **:TaskRepository** (Business/Data Access): Репозиторій, що інкапсулює логіку роботи із сутністю Task.
- **:ToDoDbContext** (Data Access): Об'єкт Entity Framework, відповідальний за взаємодію з БД.

Основна послідовність:

1. :Користувач натискає кнопку **"Додати завдання"**.
2. :MainForm створює та показує :AddTaskForm.
3. Після введення даних, :AddTaskForm повертає результат (**OK**) та дані в :MainForm.
4. :MainForm створює об'єкт :Task та викликає **Add(newTask)** у :TaskRepository.
5. :TaskRepository використовує **:ToDoDbContext** для додавання сутності в набір даних і викликає **SaveChanges()**, що призводить до фізичного запису даних у базу.
6. Після успішного збереження :MainForm викликає **LoadTasks()** для оновлення списку.

Аналогічний опис складається і для другої діаграми «Відзначити як виконане», де замість Add викликається метод Update.

Код програми

GenericRepository

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using ToDo.Data;

namespace ToDo.Repositories
{
   public class GenericRepository<T> : IRepository<T> where T : class
   {
      private readonly ToDoDbContext _context;
      private readonly DbSet<T> _dbSet;

      public GenericRepository(ToDoDbContext context)
      {
         _context = context ?? throw new
ArgumentNullException(nameof(context));
         _dbSet = _context.Set<T>();
      }

      public T? GetById(int id)
      {
         return _dbSet.Find(id);
      }

      public IEnumerable<T> GetAll()
      {
         return _dbSet.ToList();
      }

      public void Add(T entity)
      {
         _dbSet.Add(entity);
         _context.SaveChanges();
      }

      public void Update(T entity)
```

```
        {
            _dbSet.Update(entity);
            _context.SaveChanges();
        }

        public void Delete(int id)
        {
            var entity = GetById(id);
            if (entity != null)
            {
                _dbSet.Remove(entity);
                _context.SaveChanges();
            }
        }
    }
}
```

IRepository

```
using System.Collections.Generic;

namespace ToDo.Repositories
{
    public interface IRepository<T> where T : class
    {
        T? GetById(int id);

        IEnumerable<T> GetAll();
        void Add(T entity);
        void Update(T entity);
        void Delete(int id);
    }
}
```

AddTaskForm

```
using System;
using System.Windows.Forms;
using ToDo.Enums;
using ToDo.Models;

namespace ToDo.UI
```

```
{
    public partial class AddTaskForm : Form
    {
        public string TaskTitle { get; private set; }
        public PriorityEnum TaskPriority { get; private set; }
        public int ProjectId { get; private set; }

        public AddTaskForm(int projectId)
        {
            InitializeComponent();
            ProjectId = projectId;

            cmbPriority.DataSource = Enum.GetValues(typeof(PriorityEnum));
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrWhiteSpace(txtTaskTitle.Text))
            {
                MessageBox.Show("Назва завдання не може бути порожньою.");
                return;
            }

            TaskTitle = txtTaskTitle.Text;
            TaskPriority = (PriorityEnum)cmbPriority.SelectedItem;

            this.DialogResult = DialogResult.OK;
            this.Close();
        }

        private void btnCancel_Click(object sender, EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
            this.Close();
        }
    }
}
```

AddTaskForm.Designer

```
using System.ComponentModel;

namespace ToDo.UI;

partial class AddTaskForm
{
    private System.Windows.Forms.TextBox txtTaskTitle;
    private System.Windows.Forms.ComboBox cmbPriority;
    private System.Windows.Forms.Button btnSave;
    private System.Windows.Forms.Button btnCancel;

    /// <summary>
    /// Required designer variable.
    /// </summary>
    private IContainer components = null;

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }

        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
```

```
{
    this.txtTaskTitle = new System.Windows.Forms.TextBox();
    this.cmbPriority = new System.Windows.Forms.ComboBox();
    this.btnSave = new System.Windows.Forms.Button();
    this.btnCancel = new System.Windows.Forms.Button();
    this.SuspendLayout();

    //
    // txtTaskTitle
    //
    this.txtTaskTitle.Location = new System.Drawing.Point(12, 35);
    this.txtTaskTitle.Name = "txtTaskTitle";
    this.txtTaskTitle.Size = new System.Drawing.Size(260, 23);
    this.txtTaskTitle.TabIndex = 0;

    //
    // cmbPriority
    //
    this.cmbPriority.FormattingEnabled = true;
    this.cmbPriority.Location = new System.Drawing.Point(12, 70);
    this.cmbPriority.Name = "cmbPriority";
    this.cmbPriority.Size = new System.Drawing.Size(260, 23);
    this.cmbPriority.TabIndex = 1;

    //
    // btnSave
    //
    this.btnSave.Location = new System.Drawing.Point(12, 120);
    this.btnSave.Name = "btnSave";
    this.btnSave.Size = new System.Drawing.Size(120, 30);
    this.btnSave.TabIndex = 2;
    this.btnSave.Text = "Зберегти";
    this.btnSave.UseVisualStyleBackColor = true;
    this.btnSave.Click += new System.EventHandler(this.btnSave_Click);

    //
    // btnCancel
    //
    this.btnCancel.Location = new System.Drawing.Point(152, 120);
```

```
        this.btnCancel.Name = "btnCancel";
        this.btnCancel.Size = new System.Drawing.Size(120, 30);
        this.btnCancel.TabIndex = 3;
        this.btnCancel.Text = "Скасувати";
        this.btnCancel.UseVisualStyleBackColor = true;
        this.btnCancel.Click += new System.EventHandler(this.btnCancel_Click);

        //
        // AddTaskForm
        //
        this.ClientSize = new System.Drawing.Size(284, 161);
        this.Controls.Add(this.btnCancel);
        this.Controls.Add(this.btnSave);
        this.Controls.Add(this.cmbPriority);
        this.Controls.Add(this.txtTaskTitle);
        this.Name = "AddTaskForm";
        this.Text = "Додати Завдання";
        this.ResumeLayout(false);
        this.PerformLayout();
    }

    #endregion
}
```

MainForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using ToDo.Models;
using ToDo.Repositories;
using ToDo.Enums;
using Task = ToDo.Models.Task;
```

```
namespace ToDo.UI
{
   public partial class MainForm : Form
   {
      private readonly IRepository<Task> _taskRepository;
      private readonly IRepository<Project> _projectRepository;
      private readonly IRepository<User> _userRepository;

      public MainForm(IRepository<Task> taskRepository,
                      IRepository<Project> projectRepository,
                      IRepository<User> userRepository)
      {
         InitializeComponent();

         _taskRepository = taskRepository;
         _projectRepository = projectRepository;
         _userRepository = userRepository;

         LoadTasks();
      }

      private void LoadTasks()
      {
         tasksListBox.Items.Clear();

         var tasks = _taskRepository.GetAll();
         foreach (var task in tasks)
         {
            tasksListBox.Items.Add($"[{task.Status}] {task.Title} (Пріоритет:
{task.Priority})");
         }
      }

      private void btnAddTask_Click(object sender, EventArgs e)
      {
         var defaultUser = _userRepository.GetAll().FirstOrDefault();

         if (defaultUser == null)
         {
```

```
            MessageBox.Show("Критична помилка: Не знайдено користувача для збереження даних.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }

        var defaultProject = _projectRepository.GetAll().FirstOrDefault(p => p.UserId == defaultUser.UserId);

        if (defaultProject == null)
        {
            defaultProject = new Project
            {
                Title = "Загальний Проєкт",
                UserId = defaultUser.UserId,
                CreationDate = DateTime.Now
            };
            _projectRepository.Add(defaultProject);
        }

        using (var addTaskForm = new AddTaskForm(defaultProject.ProjectId))
        {
            if (addTaskForm.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    var newTask = new Task
                    {
                        Title = addTaskForm.TaskTitle!,
                        Description = string.Empty,
                        Priority = addTaskForm.TaskPriority,
                        ProjectId = defaultProject.ProjectId,
                        Status = StatusEnum.New,
                        DueDate = DateTime.Now.AddDays(1)
                    };

                    _taskRepository.Add(newTask);
```

```
                LoadTasks();
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Помилка збереження завдання:
{ex.Message}\n\n" +
                                (ex.InnerException != null ?
ex.InnerException.Message : ""),
                                "Помилка БД", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
    }
  }
}
```

MainForm.Designer

```
namespace ToDo.UI;

partial class MainForm
{
    private System.Windows.Forms.ListBox tasksListBox;
    private System.Windows.Forms.Button btnAddTask;

    /// <summary>
    ///  Required designer variable.
    /// </summary>
    private System.ComponentModel.IContainer components = null;

    /// <summary>
    ///  Clean up any resources being used.
    /// </summary>
    /// <param name="disposing">true if managed resources should be
disposed; otherwise, false.</param>
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
```

```
        }

        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    /// <summary>
    ///  Required method for Designer support - do not modify
    ///  the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent()
    {
        this.tasksListBox = new System.Windows.Forms.ListBox();
        this.btnAddTask = new System.Windows.Forms.Button();
        this.SuspendLayout();

        //
        // tasksListBox
        //
        this.tasksListBox.Location = new System.Drawing.Point(12, 12);
        this.tasksListBox.Name = "tasksListBox";
        this.tasksListBox.Size = new System.Drawing.Size(400, 300);
        this.tasksListBox.TabIndex = 0;

        //
        // btnAddTask
        //
        this.btnAddTask.Location = new System.Drawing.Point(12, 320);
        this.btnAddTask.Name = "btnAddTask";
        this.btnAddTask.Size = new System.Drawing.Size(400, 30);
        this.btnAddTask.TabIndex = 1;
        this.btnAddTask.Text = "Додати завдання";
        this.btnAddTask.UseVisualStyleBackColor = true;
        this.btnAddTask.Click += new
System.EventHandler(this.btnAddTask_Click);

        //
        // MainForm
```

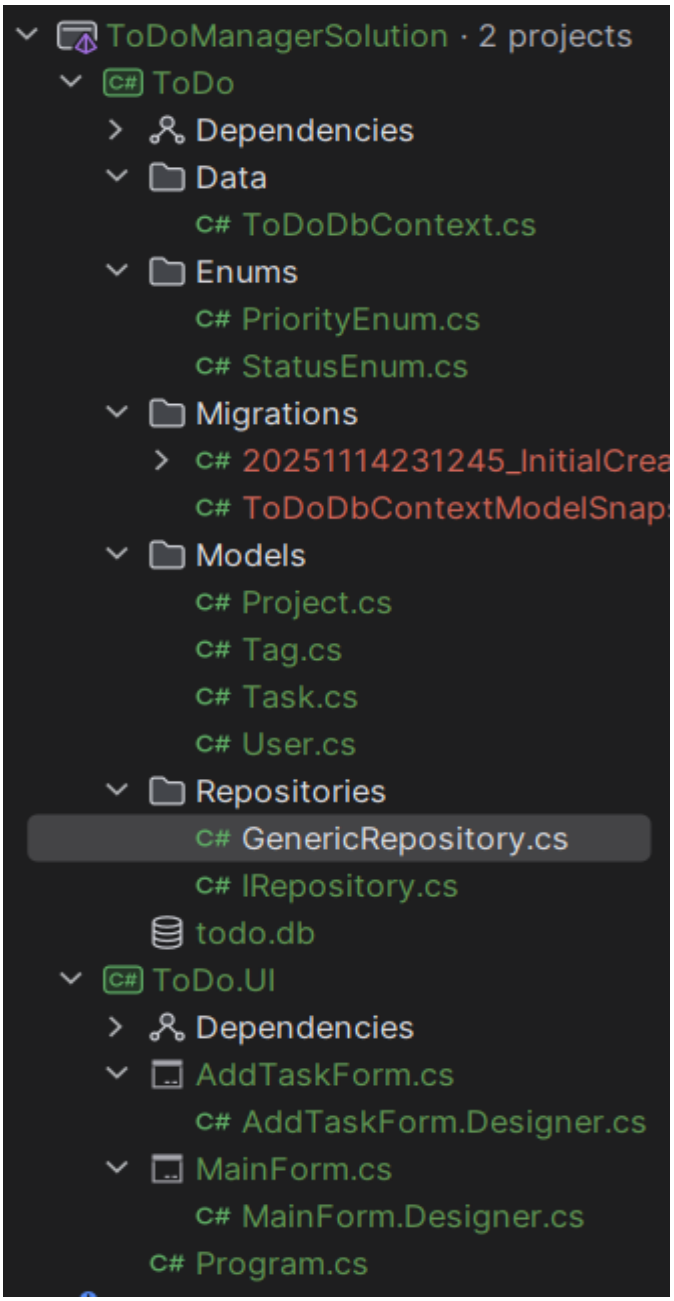
```
        //
        this.ClientSize = new System.Drawing.Size(424, 361);
        this.Controls.Add(this.btnAddTask);
        this.Controls.Add(this.tasksListBox);
        this.Text = "Менеджер Завдань (ToDo)";
        this.ResumeLayout(false);
        this.PerformLayout();
    }

    #endregion
}
```

Program

```
using System;
using System.Windows.Forms;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using ToDo.Data;
using ToDo.Models;
using ToDo.Repositories;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using Task = ToDo.Models.Task;

namespace ToDo.UI
{
    static class Program
    {
        public static IServiceProvider? Services { get; private set; }

        [STAThread]
        static void Main()
        {
            Application.SetHighDpiMode(HighDpiMode.SystemAware);
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            var host = CreateHostBuilder().Build();
            Services = host.Services;
```

```
        using (var scope = Services.CreateScope())
        {
            var dbContext =
scope.ServiceProvider.GetRequiredService<ToDoDbContext>();
            try
            {
                dbContext.Database.Migrate();
                EnsureDataExists(dbContext);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Критична помилка ініціалізації бази
даних: {ex.Message}",
                                "Помилка БД", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return;
            }
        }

        Application.Run(Services.GetRequiredService<MainForm>());
    }

    static IHostBuilder CreateHostBuilder() =>
        Host.CreateDefaultBuilder()
            .ConfigureServices((context, services) =>
            {
                // Реєструємо DbContext
                services.AddDbContext<ToDoDbContext>();

                // 2. Реєструємо Репозиторії
                services.AddTransient<IRepository<Task>,
GenericRepository<Task>>();
                services.AddTransient<IRepository<Project>,
GenericRepository<Project>>();
                services.AddTransient<IRepository<User>,
GenericRepository<User>>();

                // 3. Реєструємо наші Форми
                services.AddTransient<MainForm>();
```

```
            });

        private static void EnsureDataExists(ToDoDbContext context)
        {
            if (!context.Users.Any())
            {
                context.Users.Add(new User
                {
                    Username = "default_user",
                    PasswordHash = "hashedpassword"
                });
                context.SaveChanges();
            }

            var defaultUser = context.Users.First();
            if (!context.Projects.Any())
            {
                context.Projects.Add(new Project
                {
                    Title = "Загальний Проєкт",
                    CreationDate = DateTime.Now,
                    UserId = defaultUser.UserId
                });
                context.SaveChanges();
            }
        }
    }
}
```



Висновки

Висновки: під час виконання лабораторної роботи ми навчилися проєктувати діаграми розгортання та компонентів, що визначають логічну структуру та фізичне розміщення цільової Client-Server архітектури системи. На основі сценаріїв, розроблених у попередній роботі, ми створили діаграми послідовностей, деталізуючи динамічну взаємодію об'єктів. Реалізована частина системи була доопрацьована шляхом додавання візуальних форм та повного циклу роботи з даними (UI → Repository → БД), що підтверджує функціональну готовність спроєктованої архітектури.

Питання до лабораторної роботи

1. Що собою становить діаграма розгортання?

Діаграма розгортання (Deployment Diagram) у UML показує фізичне розміщення апаратних вузлів (серверів, клієнтів) і компонентів системи на цих вузлах. Вона відображає, як програмне забезпечення реалізується на апаратних елементах.

2. Які бувають види вузлів на діаграмі розгортання?

Фізичні вузли (Node): реальні апаратні пристрої (сервер, комп'ютер, мобільний пристрій).
Вузли виконання (Execution Environment): середовище, у якому запускаються компоненти (операційна система, віртуальна машина, контейнер).

3. Які бувають зв'язки на діаграмі розгортання?

Ассоціація між вузлами (Communication Path): показує можливість обміну даними між вузлами.

Залежність (Dependency): вказує, що один вузол або компонент залежить від іншого.

4. Які елементи присутні на діаграмі компонентів?

- Компоненти (Component): самостійні частини ПЗ.
- Інтерфейси (Interface): порти, через які компоненти взаємодіють.
- Пакети (Package): групування компонентів.
- Зв'язки (Dependency, Association): взаємозв'язки між компонентами.

5. Що становлять собою зв'язки на діаграмі компонентів? Зв'язки відображають залежності між компонентами: хто на кого покладається, хто надає чи використовує інтерфейс іншого компонента.

6. Які бувають види діаграм взаємодії?

Діаграма послідовностей (Sequence Diagram) – показує порядок повідомлень між об’єктами.

Діаграма комунікацій (Communication Diagram) – показує зв’язки між об’єктами та обмін повідомленнями.

Діаграма часу (Timing Diagram) – показує зміни станів об’єктів у часі.

Діаграма взаємодії (Interaction Overview Diagram) – поєднує елементи кількох діаграм взаємодії.

7. Для чого призначена діаграма послідовностей?

Вона описує порядок і часову послідовність взаємодії між об’єктами системи для реалізації певного сценарію чи варіанту використання.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

- Об’єкти/Актори (Lifelines)
- Повідомлення (Messages) – виклики методів між об’єктами
- Активності (Activation bars) – час виконання дії об’єкта
- Події створення/знищення об’єктів

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

Кожна діаграма послідовностей реалізує сценарій певного варіанту використання, деталізуючи, як актори взаємодіють із системою крок за кроком.

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

Повідомлення на діаграмі послідовностей зазвичай відповідають методам класів, а об’єкти на діаграмі є екземплярами класів із діаграми класів.