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 розробки програмного забезпечення*»

Тема: «Основи проектування розгортання»

Виконав:
Студент групи ІА-31
Клим’юк В.Л.

Перевірив:
Мягкий М.Ю.

Київ-2025

Тема: Система запису на прийом до лікаря (strategy, adapter, observer, facade, visitor, client-server).
Система дозволяє пацієнтам шукати та записуватись на прийоми до лікарів, де їм виноситься висновок за результатами прийому. Також наявні функції відміни прийому та додавання відгуків про лікарів.

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.

Теоретичні відомості:

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).

Між вузлами можуть стояти зв'язки, які зазвичай зображують у вигляді прямої лінії. Як і на інших діаграмах, у зв'язків можуть бути атрибути множинності. Вузли можуть містити артефакти (artifacts), які є фізичним уособленням програмного забезпечення; зазвичай це файли. Перелік артефактів усередині вузла вказує на те, що на даному вузлі артефакт розгортається в систему, що запускається. Артефакти можна зображати у вигляді прямокутників класів або перераховувати їхні імена всередині вузла. Артефакти часто є реалізацією компонентів. Це можна показати, задавши значення-мітки всередині прямокутників артефактів.

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. Коли на діаграмі представляють фізичне розбиття, то в такому разі на діаграмі компонентів показують компоненти та залежності між ними. Залежності показують, що класи в з одного компонента використовують класи з іншого компонента. Фізична модель використовується для розуміння які компоненти повинні бути зібрані в інсталяційний пакет. Також така діаграма показує зміни в якому компоненті будуть впливати на інші компоненти.

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
- Об’єкти або класи
- Повідомлення
- Активності
- Контрольні структури

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.

Хід роботи:

Діаграма розгортання:

На діаграмі зображено сервер, запущений на машині з Windows OS. Код серверу виконується в середовищі .NET Core CLR. Сервер та база даних комунікують через frontend/backend protocol від PostgreSQL. Клієнти комунікують з сервером через http протокол і використовують для цього веб-браузер.

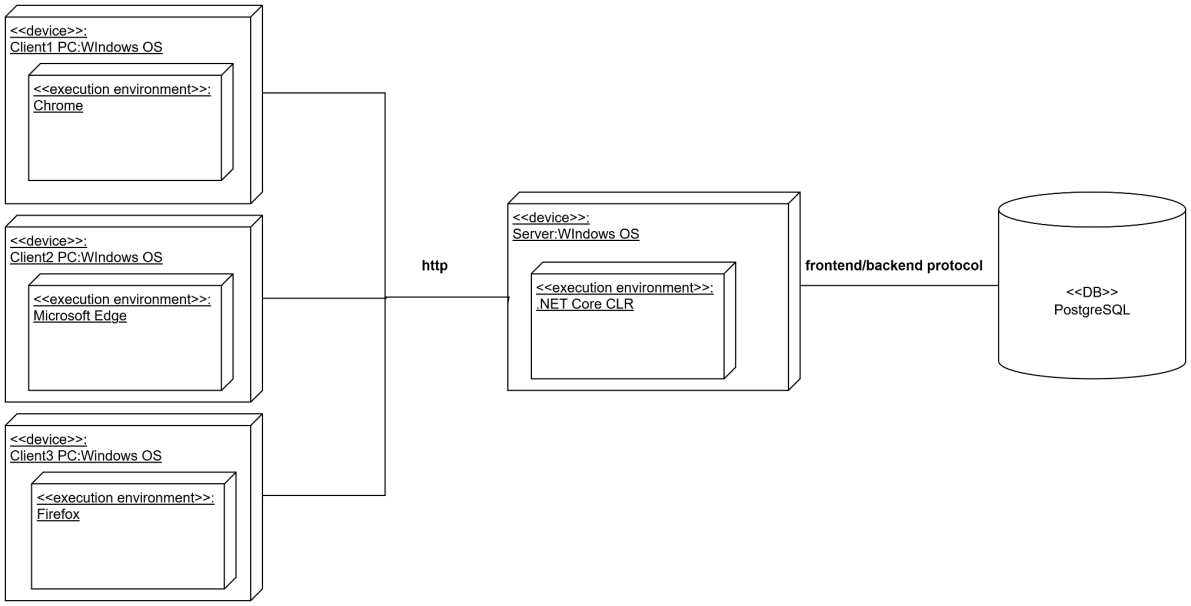

Рис.1 - Діаграма розгортання

Діаграма компонентів:

На діаграмі зображено клієнтську та серверну частину системи. Клієнтська частина складається з браузера, хтмл-сторінок та файлів бібліотеки

Bootstrap. Серверна частина складається з виконуваного файлу, основної бібліотеки програми BookingClinic.dll, бібліотеки для розробки веб-додатків Microsoft.NET.Sdk.Web, бібліотек для роботи з базою даних: Microsoft.EntityFrameworkCore.dll та Npgsql.dll, а також додаткових бібліотек для роботи з даними: Mapster.dll та FluentValidation.dll.

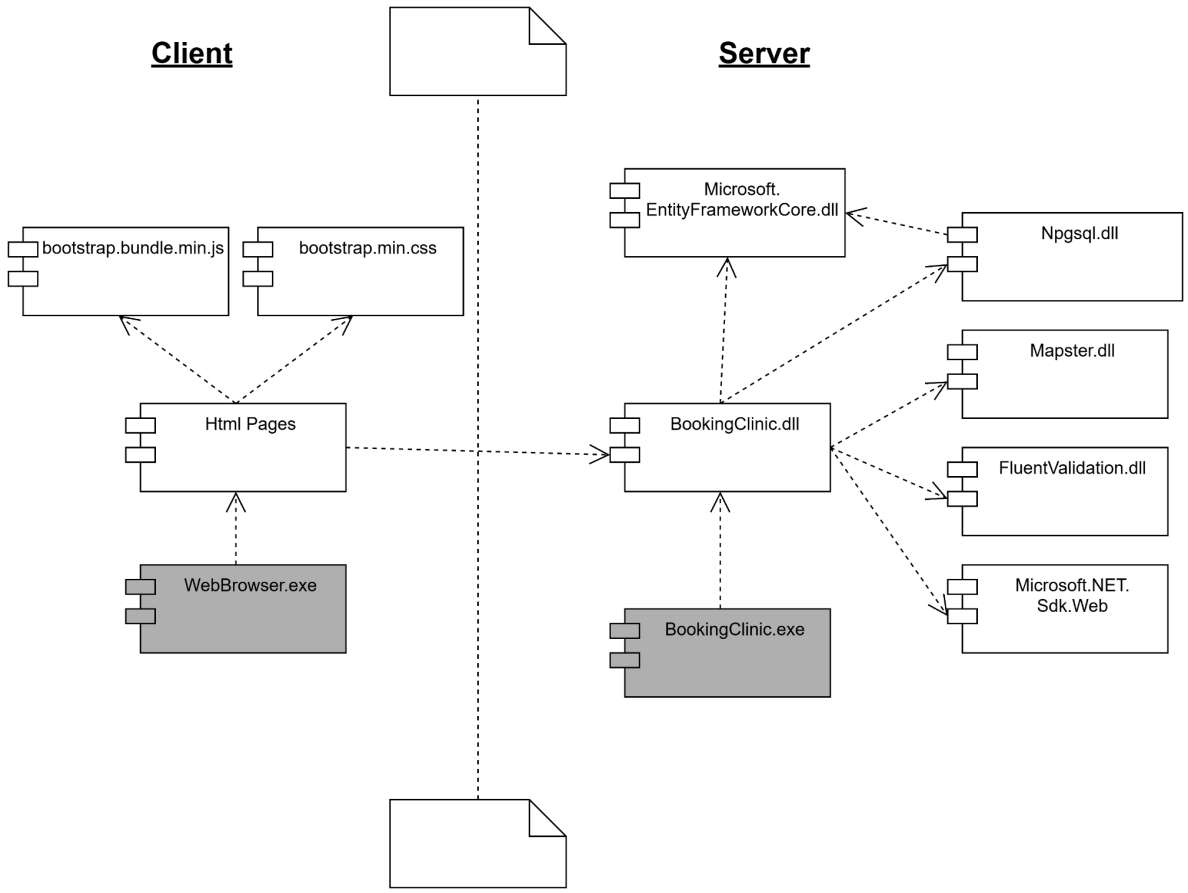

Рис.2 - Діаграма компонентів

Діаграми послідовностей:

Сценарій 1. Запис на прийом:

Передумови: Пацієнт авторизований у системі, лікар існує та має вільні місця для запису на прийом.
Постумови: Створюється запис на прийом до обраного лікаря.
Взаємодіючі сторони: Пацієнт, Система запису пацієнтів на прийом.
Короткий опис: Цей варіант використання визначає запис пацієнта на прийом.
Основний потік подій:

Цей варіант використання запускається коли користувач хоче записатись на прийом до лікаря.

1. Користувач шукає лікаря.
2. Користувач переходить на профіль лікаря.
3. Користувач обирає доступні час та дату запису.
4. Користувач підтверджує дію запису.

Винятки:

Виняток №1: Місце було зайняте. Якщо обране місце виявилось зайнятим, то користувач отримує помилку і повертається до початку основного потоку.

Примітки: Відсутні.

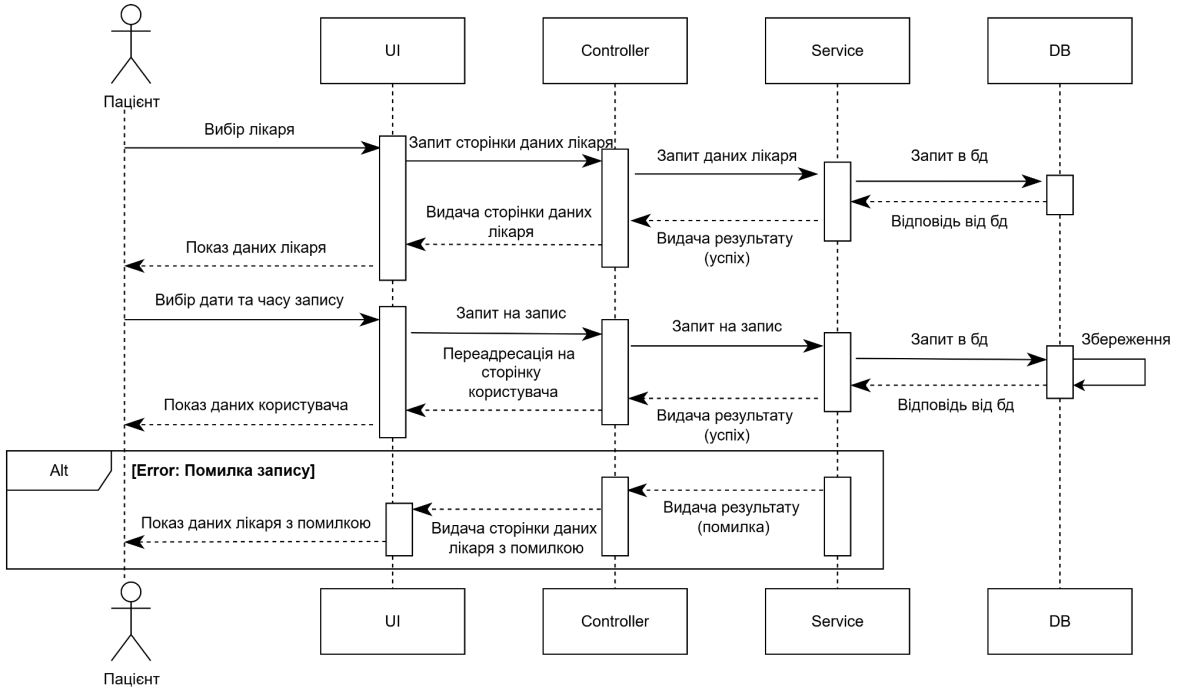

Рис.3 - Діаграма послідовностей: Запис на прийом

Сценарій 3. Створити відгук про лікаря:

Передумови: Пацієнт має хоча б один завершений прийом у лікаря.
Постумови: В разі успіху створюється відгук про обраного лікаря.
Взаємодіючі сторони: Пацієнт, Система запису пацієнтів на прийом.
Короткий опис: Цей варіант використання визначає як пацієнт пише відгук про лікаря.
Основний потік подій: Цей варіант використання запускається коли пацієнт відкриває список лікарів.

1. Пацієнт знаходить потрібного лікаря.
2. Пацієнт переходить у профіль лікаря.
3. Пацієнт натискає на кнопку додавання відгуку.
4. Пацієнт вводить текст та ставить оцінку.
5. Пацієнт підтверджує дію.

Винятки:

Виняток №1: Пацієнт не мав завершених прийомів у лікаря. Система повідомляє користувачу про неможливість залишити відгук.

Примітки: Відсутні.

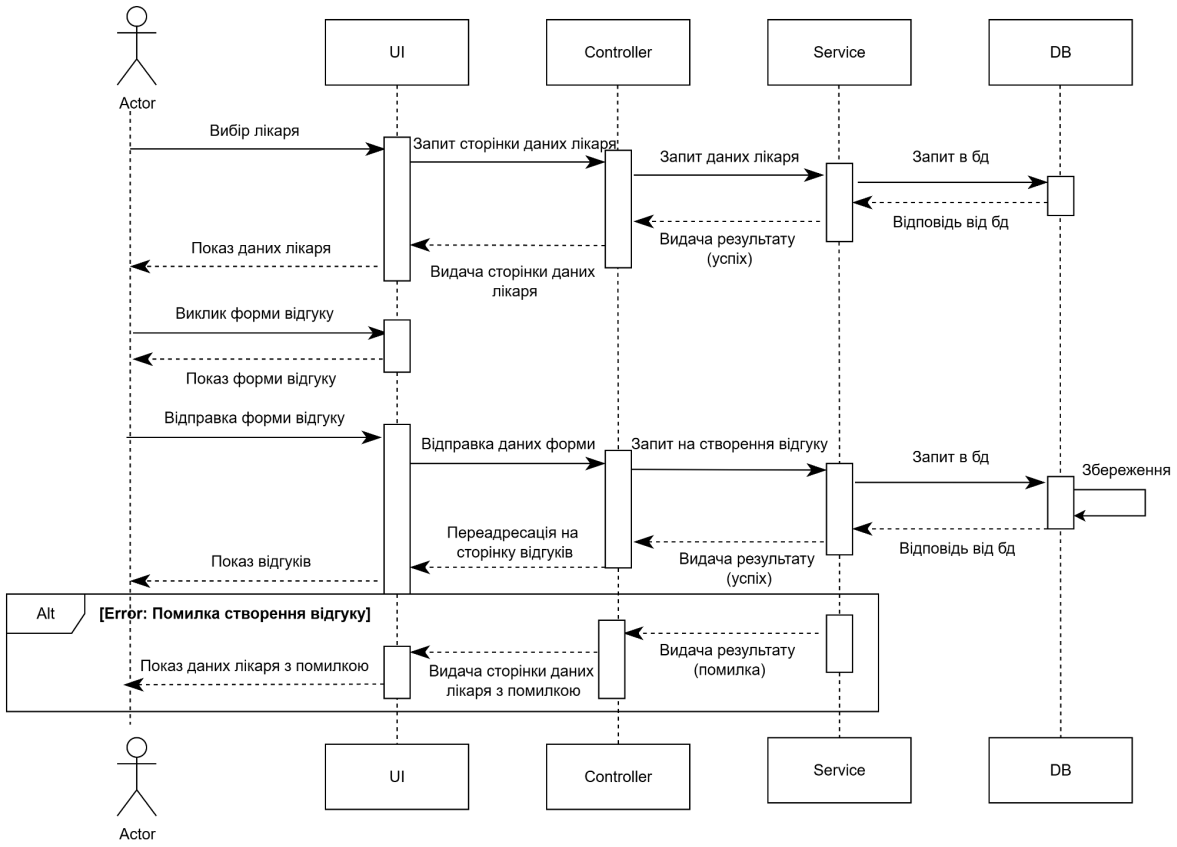

Рис.4 - Діаграма послідовностей: Створити відгук про лікаря

Реалізовані компоненти системи:

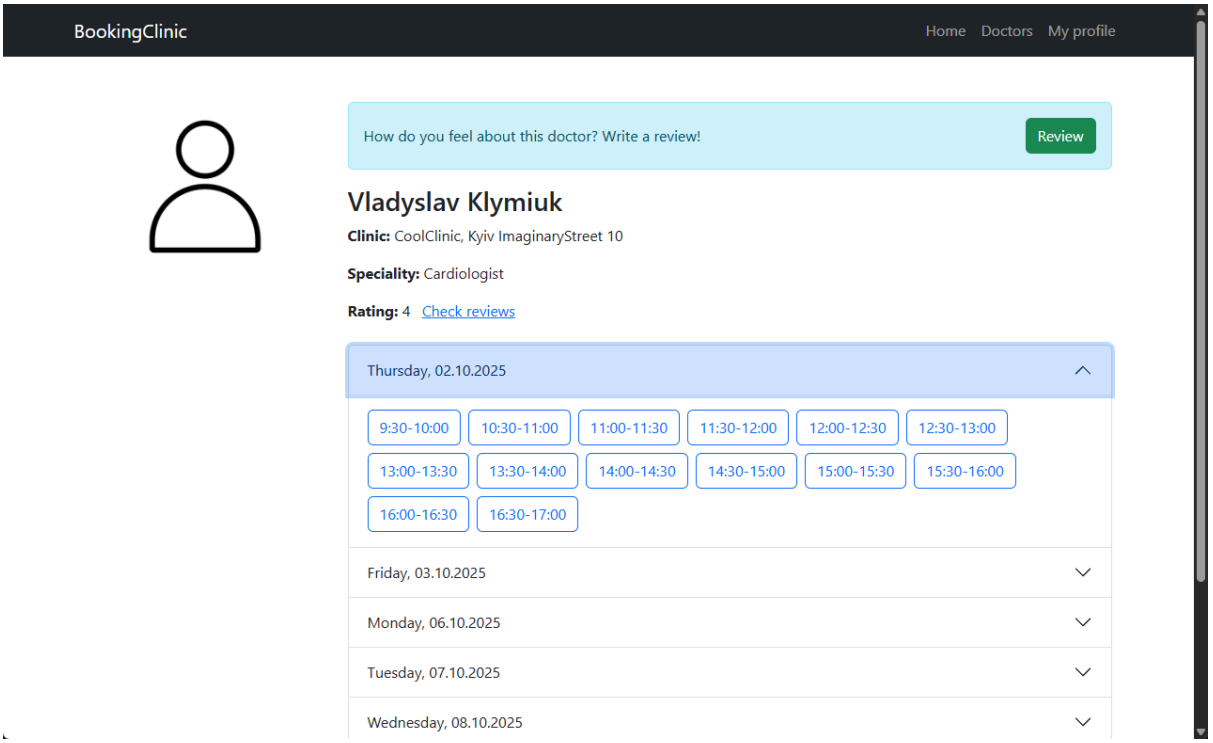

Рис. 5 - Сторінка профілю лікаря

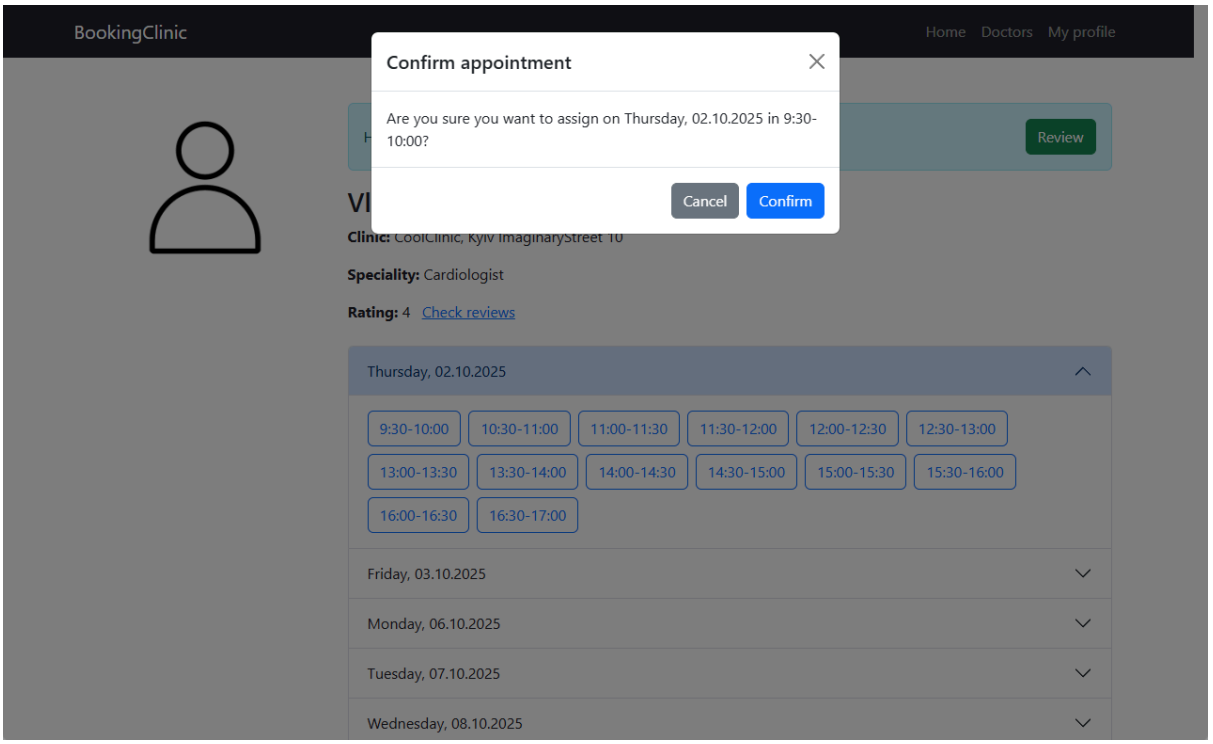

Рис.6 - Запис на прийом

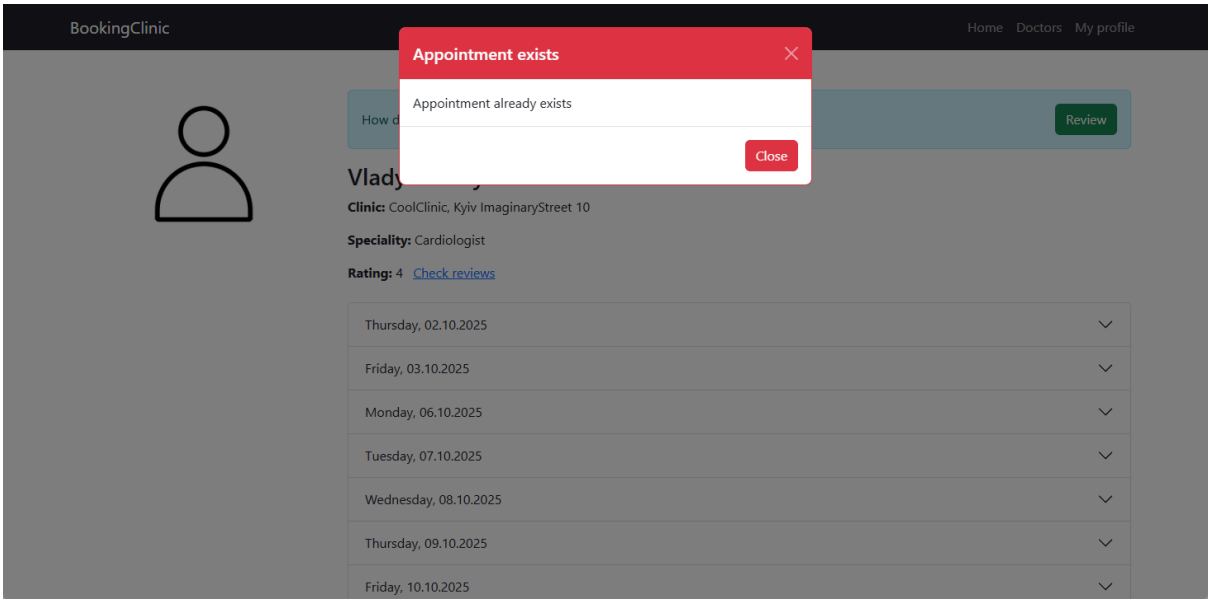

Рис.7 - Помилка запису на прийом

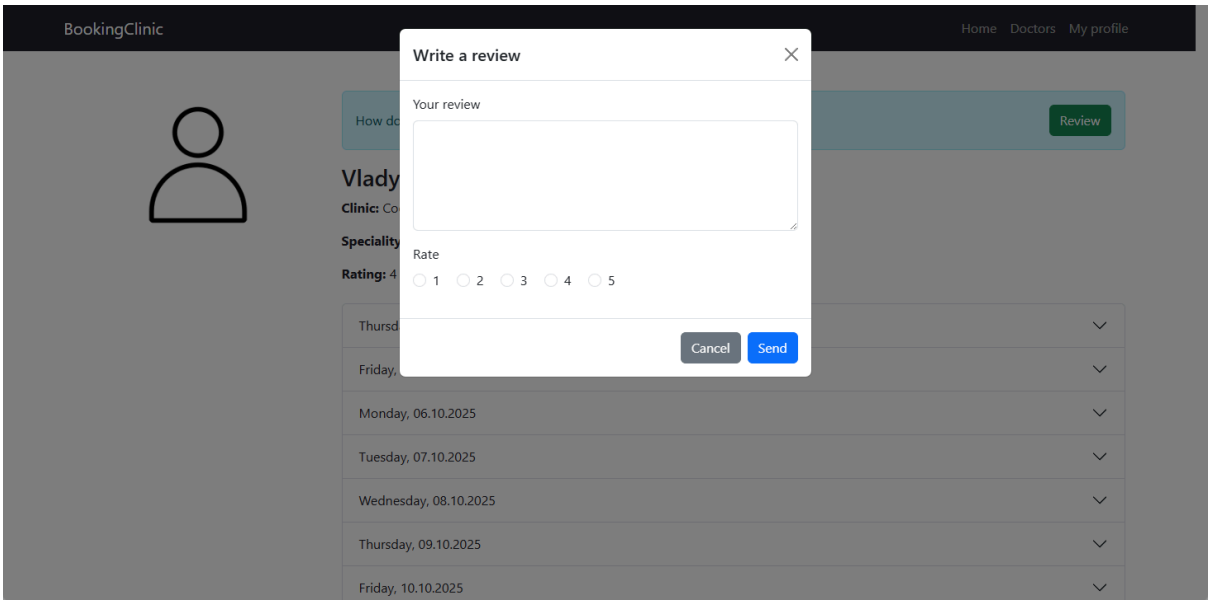

Рис.8 - Написання відгуку

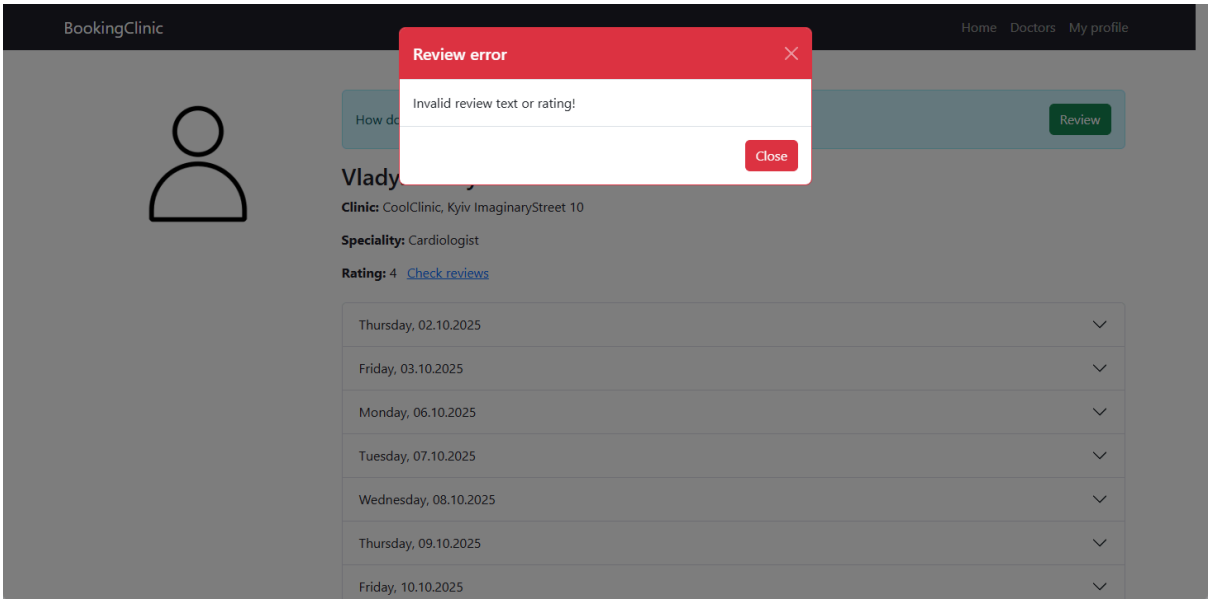

Рис.9 - Помилка написання відгуку

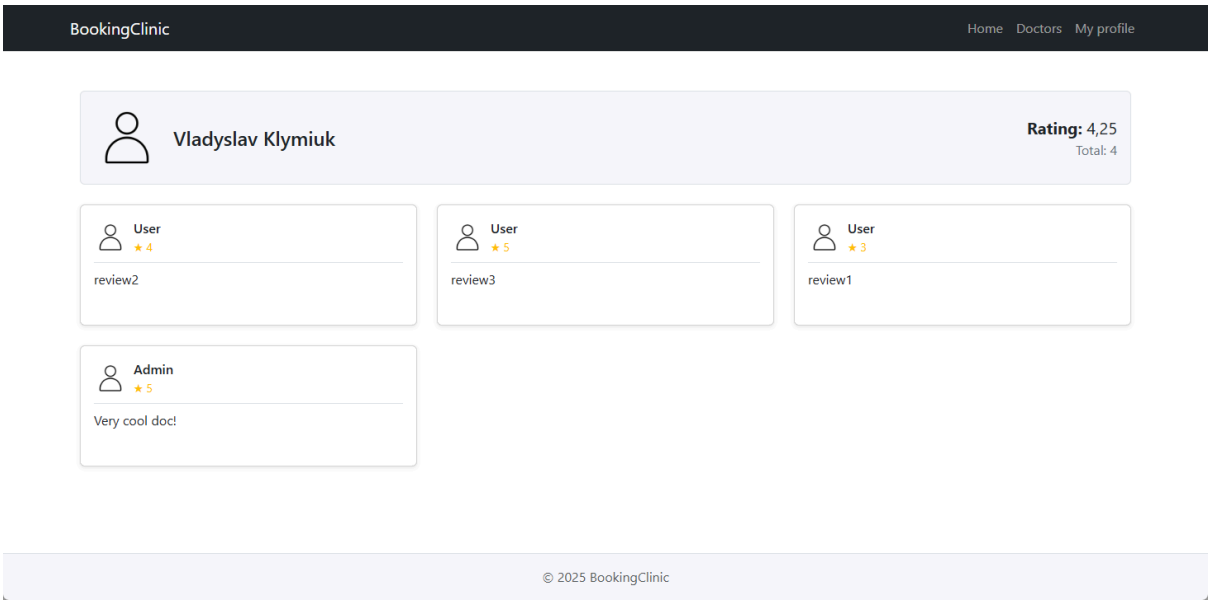

Рис.10 - Відгуки про лікаря

```
namespace BookingClinic.Controllers
{
    [Route("doctors")]
    Ссылок: 1
    public class DoctorController : Controller
    {
        private readonly IUserService _userService;
        private readonly ISpecialityService _specialityService;
        private readonly IClinicService _clinicService;
        private readonly IDoctorService _doctorService;
        private readonly IAppointmentService _appointmentService;
        private readonly IPaginationHelper<SearchDoctorResDto> _paginationHelper;
        private readonly IReviewsHelper _reviewsHelper;

        Ссылок: 0
        public DoctorController(
            IUserService userService,
            IPaginationHelper<SearchDoctorResDto> paginationHelper,
            ISpecialityService specialityService,
            IClinicService clinicService,
            IDoctorService doctorService,
            IAppointmentService appointmentService,
            IReviewsHelper reviewsHelper)...

        [HttpGet]
        Ссылок: 0
        public IActionResult Index([FromQuery] SearchDoctorDto dto, [FromQuery] int page, [FromQuery] string? orderBy)
        {
            var res = _userService.SearchDoctors(dto);
            var specialities = _specialityService.GetSpecialityNames();
            var clinics = _clinicService.GetClinicNames();

            if (specialities.IsSuccess)
            {
                ViewData["Specialities"] = specialities.Result;
            }

            if (clinics.IsSuccess)
            {
                ViewData["Clinics"] = clinics.Result;
            }

            ViewData["Ordering"] = orderBy;
            ViewData["Page"] = page;

            if (res.IsSuccess)
            {
```

Рис.11 - Код класу DoctorController

```
        if (res.IsSuccess)
        {
            var doctors = _paginationHelper.Paginate(res.Result, page, 5, out var pages);

            ViewData["Pages"] = pages;
            ViewData["Doctors"] = doctors;

            return View(dto);
        }
        else
        {
            ViewData["Page"] = 1;
            ViewData["Pages"] = new List<int> { 1 };
            ViewData["Errors"] = res.Errors;
            ViewData["Doctors"] = new List<SearchDoctorResDto>() { };

            return View(dto);
        }
    }

    [HttpGet("{id:guid}")]
    Ссылок: 0
    public IActionResult Profile([FromRoute] Guid id)
    {
        if (TempData["Errors"] != null)
        {
            ViewData["Errors"] = JsonSerializer.Deserialize<List<ServiceError>>(TempData["Errors"].ToString());
        }

        if (TempData["ReviewErrors"] != null)
        {
            List<ValidationFailure> failures =
                JsonSerializer.Deserialize<List<ValidationFailure>>(TempData["ReviewErrors"].ToString());

            var valRes = new ValidationResult(failures);
            valRes.AddToModelState(ModelState);
        }

        var res = _doctorService.GetDoctorData(id);

        if (res.IsSuccess)
        {
            if (User.IsInRole("Patient") || User.IsInRole("Admin"))
            {
                res.Result!.CanWriteReview = _reviewsHelper.CanUserWriteReview(id, User);
            }

            return View(res.Result);
```

Рис.12 - Код класу DoctorController

```
            }
            else
            {
                ViewData["Errors"] = res.Errors;
                return View();
            }
        }

        [HttpPost]
        [Authorize("AuthUser")]
        Ссылок: 0
        public async Task<IActionResult> MakeAppointment(
            [FromForm] MakeAppointmentDto dto)
        {
            var res = await _appointmentService.CreateAppointment(dto, User);

            if (!res.IsSuccess)
            {
                TempData["Errors"] = JsonSerializer.Serialize(res.Errors);
                return RedirectToAction("Profile", new { id = dto.DoctorId });
            }
            return RedirectToAction("Index", "User");
        }
    }
}
```

Рис.13 - Код класу DoctorController

```
namespace BookingClinic.Controllers
{
    [Route("reviews")]
    Ссылок: 1
    public class ReviewController : Controller
    {
        private readonly IReviewService _reviewService;
        private readonly IValidator<AddReviewDto> _addReviewValidator;

        Ссылок: 0
        public ReviewController(
            IReviewService reviewService,
            IValidator<AddReviewDto> addReviewValidator)
        {
            _reviewService = reviewService;
            _addReviewValidator = addReviewValidator;
        }

        [HttpGet("{id:guid}")]
        Ссылок: 0
        public IActionResult Index([FromRoute] Guid id)
        {
            var res = _reviewService.GetDoctorReviews(id);

            if (res.IsSuccess)
            {
                return View(res.Result);
            }
            else
            {
                ViewData["Errors"] = res.Errors;
                return View();
            }
        }

        [HttpPost]
        [Authorize("AuthUser")]
        Ссылок: 0
        public async Task<IActionResult> CreateReview([FromForm] AddReviewDto dto)
        {
            var validationRes = _addReviewValidator.Validate(dto);

            if (!validationRes.IsValid)
            {
                TempData["Errors"] = JsonSerializer.Serialize(
                    new List<ServiceError>() { ServiceError.InvalidReviewData()});
                return RedirectToAction("Profile", "Doctor", new { id = dto.DoctorId});
```

Рис.14 - Код класу ReviewController

```
            }

            var res = await _reviewService.CreateReview(dto, User);

            if (res.IsSuccess)
            {
                return RedirectToAction("Index", new {id = dto.DoctorId});
            }
            else
            {
                TempData["Errors"] = JsonSerializer.Serialize(res.Errors);
                return RedirectToAction("Profile", "Doctor", new { id = dto.DoctorId });
            }
        }
    }
}
```

Рис.15 - Код класу ReviewController

```
namespace BookingClinic.Services.Appointment
{
    Ссылок: 2
    public class AppointmentService : IAppointmentService
    {
        private readonly IAppointmentRepository _appointmentRepository;
        private readonly IUserRepository _userRepository;

        Ссылок: 0
        public AppointmentService(
            IAppointmentRepository appointmentRepository,
            IUserRepository userRepository)
        {
            _appointmentRepository = appointmentRepository;
            _userRepository = userRepository;
        }

        Ссылок: 2
        public async Task<ServiceResult<object>> CreateAppointment(MakeAppointmentDto dto, ClaimsPrincipal principal)
        {
            var idClaim = principal.FindFirst(c => c.Type == ClaimTypes.NameIdentifier);
            var id = Guid.Parse(idClaim.Value);

            var doctor = _userRepository.GetDoctorById(dto.DoctorId);

            if (doctor == null)
            {
                return ServiceResult<object>.Failure(
                    new List<ServiceError>() { ServiceError.DoctorNotFound() });
            }

            var clinic = doctor.Clinic;

            var times = dto.AppointmentTime.Split('-');
            var hoursMinutes = times[0].Split(':');
            var hours = int.Parse(hoursMinutes[0]);
            var minutes = int.Parse(hoursMinutes[1]);
            DateTime dateTime = DateTime.ParseExact(dto.AppointmentDay.Split(',')[1].Trim(), "dd.MM.yyyy", CultureInfo.InvariantCulture);
            dateTime = DateTime.SpecifyKind(dateTime.AddHours(hours).AddMinutes(minutes), DateTimeKind.Utc);

            var app = _appointmentRepository.GetByDateTime(dateTime);
```

Рис.16 - Код класу AppointmentService

```
            var app = _appointmentRepository.GetByDateTime(dateTime);

            if (app != null)
            {
                return ServiceResult<object>.Failure(
                    new List<ServiceError>() { ServiceError.AppointmentAlreadyExists() });
            }

            var appointment = new BookingClinic.Data.Entities.Appointment()
            {
                Id = Guid.NewGuid(),
                PatientId = id,
                DoctorId = dto.DoctorId,
                CreatedAt = DateTime.UtcNow,
                DateTime = dateTime,
                Address = $"{clinic.Name}, {clinic.City} {clinic.Street} {clinic.Building}"
            };

            _appointmentRepository.AddEntity(appointment);

            try
            {
                await _appointmentRepository.SaveChangesAsync();

                return ServiceResult<object>.Success(null);
            }
            catch (Exception)
            {
                return ServiceResult<object>.Failure(
                    new List<ServiceError>() { ServiceError.UnexpectedError() });
            }
        }
    }
}
```

Рис.17 - Код класу AppointmentService

```
namespace BookingClinic.Services.Review
{
    Ссылок: 2
    public class ReviewService : IReviewService
    {
        private readonly IDoctorReviewRepository _reviewsRepository;
        private readonly IUserRepository _usersRepository;

        Ссылок: 0
        public ReviewService(
            IDoctorReviewRepository reviewsRepository,
            IUserRepository usersRepository)
        {
            _reviewsRepository = reviewsRepository;
            _usersRepository = usersRepository;
        }

        Ссылок: 2
        public async Task<ServiceResult<object>> CreateReview(AddReviewDto dto, ClaimsPrincipal principal)
        {
            var userIdClaim = principal.FindFirst(c => c.Type == ClaimTypes.NameIdentifier);
            var userId = Guid.Parse(userIdClaim!.Value);

            DoctorReview rev = new()
            {
                Id = Guid.NewGuid(),
                DoctorId = dto.DoctorId,
                PatientId = userId,
                Rating = dto.Rating,
                Text = dto.Text
            };

            _reviewsRepository.AddEntity(rev);

            try
            {
                await _reviewsRepository.SaveChangesAsync();

                return ServiceResult<object>.Success(null);
            }
            catch (Exception)
            {
                return ServiceResult<object>.Failure(
                    new List<ServiceError>() { ServiceError.UnexpectedError() });
            }
        }
```

Рис.18 - Код класу ReviewService

```
        Ссылок: 2
        public ServiceResult<DoctorReviewsDto> GetDoctorReviews(Guid doctorId)
        {
            var doctor = _usersRepository.GetById(doctorId);

            if (doctor == null)
            {
                return ServiceResult<DoctorReviewsDto>.Failure(
                    new List<ServiceError>() { ServiceError.DoctorNotFound() });
            }

            var reviews = _reviewsRepository.GetDoctorsReviews(doctorId);

            var res = doctor.Adapt<DoctorReviewsDto>();
            res.Reviews = reviews.Adapt<IEnumerable<ReviewDataDto>>();
            res.Rating = res.Reviews.Select(r => r.Rating).DefaultIfEmpty(0).Average();

            return ServiceResult<DoctorReviewsDto>.Success(res);
        }
    }
}
```

Рис.19 - Код класу ReviewService

Контрольні питання:

1. Що собою становить діаграма розгортання?

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

2. Які бувають види вузлів на діаграмі розгортання?

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).

3. Які бувають зв’язки на діаграмі розгортання?

Зв’язки між вузлами зображуються лініями, що можуть мати назви (наприклад, протоколи чи технології) та атрибути множинності.

4. Які елементи присутні на діаграмі компонентів?

Діаграма компонентів містить компоненти і залежності між ними.

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

Зв’язки показують, що класи в з одного компонента використовують класи з іншого компонента.

6. Які бувають види діаграм взаємодії?

Види діаграм взаємодії: діаграма послідовностей і діаграма комунікацій.

7. Для чого призначена діаграма послідовностей?

Діаграма послідовностей 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.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

Діаграма послідовностей складається з акторів, об’єктів, повідомлень, активності, контрольних структур.

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

Діаграми послідовностей деталізують сценарії варіантів використання, показуючи їхній перебіг у часі.

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

На діаграмі послідовностей застосовуються класи, що описуються на діаграмі класів, і зв’язки між ними.

Висновки:
При виконанні лабораторної роботи я ознайомився з ще 3 видами UML-діаграм: розгортання, компонентів та послідовностей. На основі обраної теми було розроблено всі три типи діаграм, що дало можливість наочно відобразити структуру системи, її логічні компоненти та взаємодію між ними. Виконання лабораторної роботи допомогло краще зрозуміти практичне призначення цих діаграм при проектуванні. Отримані знання допоможуть у подальшій розробці проекту.