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 розробки програмного забезпечення*»

Тема: «Взаємодія компонентів системи»

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.

Репозиторій: https://github.com/StaticReadonly/kpi-trpz-labs-klim

Мета: Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Service oriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

Теоретичні відомості:

Клієнт-серверні додатки являють собою найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовується для зберігання і обробки даних).

Розрізняють тонкі клієнти і товсті клієнти.

Тонкий клієнт – клієнт, який повністю всі операції (або більшість, пов'язаних з логікою роботи програми) передає для обробки на сервер, а сам зберігає лише візуальне уявлення одержуваних від сервера відповідей. Грубо кажучи, тонкий клієнт – набір форм відображення і канал зв'язку з сервером. Прикладом тонкого клієнта є класичні Web-застосунки. У такому варіанті використання майже все навантаження лягає на сервер або групу серверів. Перевагою таких моделей є простота розгортання, тому що оновлювати потрібно лише сервери і в результаті клієнти з наступними запитами автоматично будуть працювати з оновленою системою.

Товстий клієнт – антипод тонкого клієнта, більшість логіки обробки даних містить на стороні клієнта. Це сильно розвантажує сервер. Сервер в таких випадках зазвичай працює лише як точка доступу до деякого іншого ресурсу (наприклад, бази даних) або сполучна ланка з іншими клієнтськими комп'ютерами. Перевагою такого підходу є менші вимоги до серверної частини. Також перевагою, при певному підході до реалізації, є можливість працювати клієнтам без тимчасового доступу до серверу. Прикладом товстого клієнта можна назвати мобільні застосунки, або

десктоп застосунки. Наприклад, Evernote, Viber, MS Outlook, комп’ютерні антивіруси, ігри, що потребують інсталяції (The Sims, GTA, …) та інші.

Проміжним варіантом можна назвати SPA (Single Page Application) – це товсті Web-клієнти, які при старті кожен раз завантажуються з сервера, а надалі працюють з сервером через web-API. З одного боку більшу частину логіки вони відпрацьовують на клієнтський стороні, за рахунок чого зменшується серверне навантаження. Також оновлення простіше ніж для товстих клієнтів. Але такі застосунки не працюють, якщо сервер не доступний.

Клієнтська частина містить візуальне відображення і логіку обробки дії користувача; код для встановлення сеансу зв'язку з сервером і виконання відповідних викликів.

Серверна частина містить основну логіку роботи програми (бізнес-логіку) або ту її частину, яка відповідає зберіганню або обміну даними між клієнтом і сервером або клієнтами.

Peer-to-Peer (P2P) архітектура – це модель мережевої взаємодії, в якій кожен вузол (комп'ютер або пристрій) є одночасно клієнтом і сервером. У цій архітектурі всі вузли мають рівні права та можливості для обміну даними, ресурсами або виконання завдань. На відміну від клієнт-серверної моделі, де є чітке розділення на клієнти і сервери, P2P-мережа дозволяє учасникам взаємодіяти безпосередньо, без необхідності в централізованому сервері. Основними принципами P2P-архітектури є:

- Децентралізація – відсутність центрального сервера, що зменшує залежність від одного вузла, підвищуючи стійкість мережі до збоїв і атак.
- Рівноправність вузлів – кожен вузол може виконувати одночасно функції клієнта (отримувати ресурси) і сервера (надавати ресурси).
- Розподіл ресурсів – вузли надають доступ до своїх власних ресурсів, таких як обчислювальна потужність, дисковий простір або файли.

Основними сферами де peer-to-peer архітектура знайшла широке застосування є файлообмінники (BitTorrent), криптовалюти та інші блокчейн технології, інтернет телефонія та відеоконференції (Skype, Zoom), розподілені обчислення (SETI@home, BOINC).

До основних проблемних зон можна віднести безпеку, синхронізацію даних та пошук ресурсів. Через централізацію складно контролювати дані,

які передаються. Ефективність пошуку даних знижується зі збільшенням кількості вузлів у мережі і для підвищення ефективності пошуку потрібно застосовувати спеціальні алгоритми.

Сервіс-орієнтована архітектура (SOA, англ. service-oriented architecture) – модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабо пов’язаних сервісів або служб, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами. Історично сервіс-орієнтована архітектура з'явилася як альтернатива монолітній архітектурі, в які вся система розроблялася та розгорталася як одне ціле.

Програмні комплекси, розроблені відповідно до сервіс-орієнтованою архітектурою, зазвичай реалізуються як набір веб-служб (або веб-сервісів), які, як правило, взаємодіють по HTTP з використанням SOAP або REST. Ці служби надають певні бізнес-функції, наприклад, отримання інформації про наявність матеріалів на складі.

Сервіси взаємодіють між собою тільки за рахунок обміну повідомленнями, без створення спеціальних інтеграцій для доступу до однієї інформації, наприклад, до однієї бази даних. Сервіси також можуть бути реалізовані як обгортки навколо застарілої системи. Це робиться для зменшення вартості переробки системи, а також спрощення інтеграції існуючих монолітних систем в нову архітектуру

Сама назва дає зрозуміти, що мікросервісна архітектура є підходом до створення серверного додатку як набору малих служб. Це означає, що архітектура мікро-сервісів головним чином орієнтована на серверну частину, не дивлячись на те, що цей підхід так само використовується для зовнішнього інтерфейсу, де кожна служба виконується в своєму процесі і взаємодіє з іншими службами за такими протоколами, як HTTP/HTTPS, WebSockets чи AMQP. Кожен мікросервіс реалізує специфічні можливості в предметній області і свою бізнес-логіку в рамках конкретного обмеженого контексту, повинна розроблятись автономно і розвертатись незалежно.

Мікросервіси забезпечують чудові можливості супроводження у величезних комплексних системах з високою масштабуємістю за рахунок створення додатків, заснованих на множині незалежно розгортуючих служб з автономними життєвими циклами.

Завдання:

- Ознайомитись з короткими теоретичними відомостями.
- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
- Реалізувати функціонал для роботи в розподіленому оточенні відповідно до обраної теми.
- Реалізувати взаємодію розподілених частин:
- Для клієнт-серверних варіантів: реалізація клієнтської і серверної частини додатків, а також загальної частини (middleware); зв'язок клієнтської і серверної частин за допомогою WCF, TcpClient, .NETRemoting на розсуд виконавця.
- Для однорангових мереж: реалізація взаємодії клієнтських додатків за допомогою WCF Peer to peer channel.
- Для SOA додатків: реалізація сервісу, що надає послуги клієнтським застосуванням; викладання сервісу в хмару або підняття у вигляді Web Service на локальній машині; використання токенів для передачі даних про автентифікації, двостороннє шифрування.
- 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.

Хід роботи:

Клієнтська частина складається з html-сторінок, що повертаються сервером та мають необхідні елементи для того щоб робити запити на сервер. За допомогою рендерингу сторінок браузер вбудовує в сторінку дані моделей відповідно до запиту на повертає клієнту. Клієнт є тонким, оскільки не реалізує ніякої обробки, а лише надсилає запити серверу.

Серверна частина побудована за допомогою ASP.NET Core. При отриманні запиту серверна частина серіалізує його в об’єкт класу HTTPContext, який містить дані запиту включаючи тіло, заголовки, строку запиту та інше.

Спочатку дані запиту проходять проміжну частину Middleware, яка бере на себе завдання автентифікації, авторизації, маршрутизації. Після шару Middleware дані запиту потрапляють у клас-контролер, який асоційований зі шляхом поточного HTTP-запиту. Контролерів багато, і кожний з них займається обробкою логічно пов’язаних запитів із застосуванням сервісів та репозиторіїв. Наприклад, AdminController займається обробкою запитів зі сторінки адміністратора. Коли контролер обробив запит, він надсилає відповідь у вигляді cshtml файла з даними, які перетворюються в готову html-сторінку та відправляються клієнту.

Типовий процес обробки запиту від клієнта в моєму додатку виглядає так:
Новий запит -> Controller -> Service -> Repository.
Тобто контролер приймає запит, викликає потрібний сервіс, сервіс викликає потрібний репозиторій, репозиторій фіксує зміни в базі якщо вони є.

Діаграма класів:

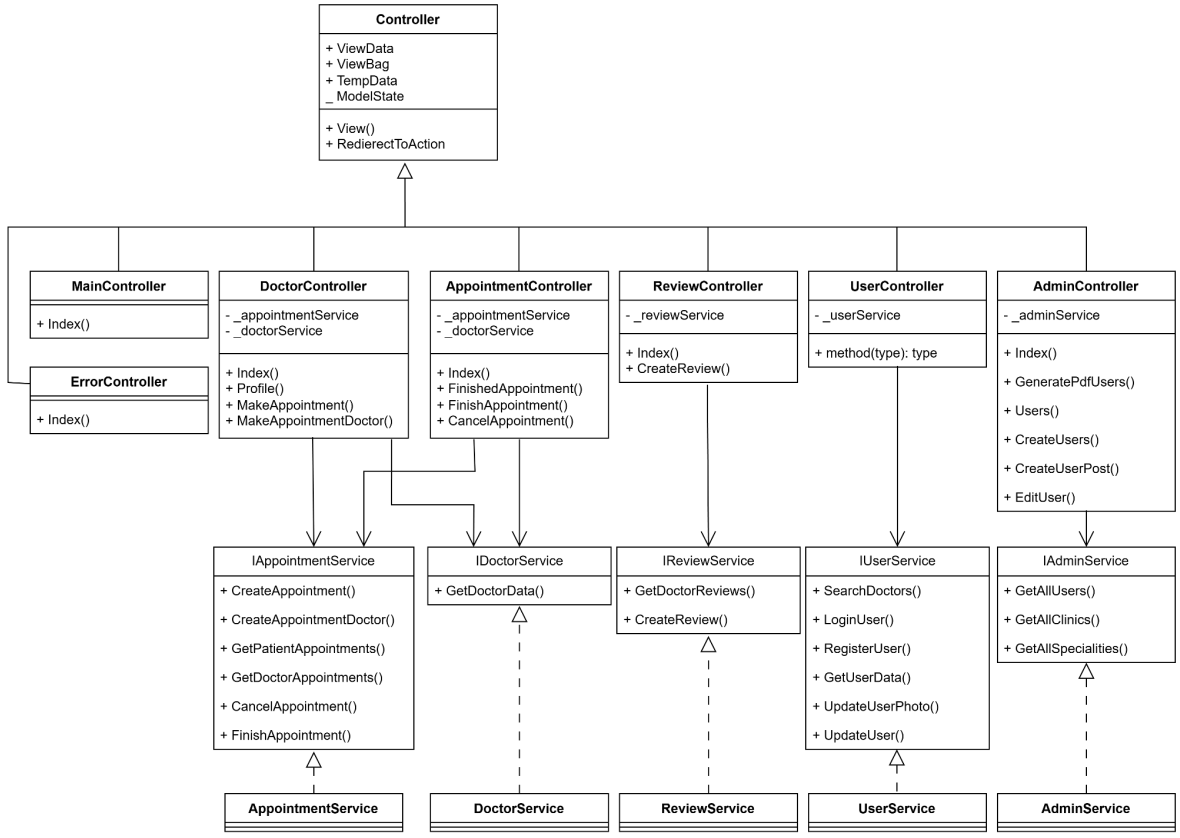

Рис.1 - Діаграма класів контролерів та сервісів

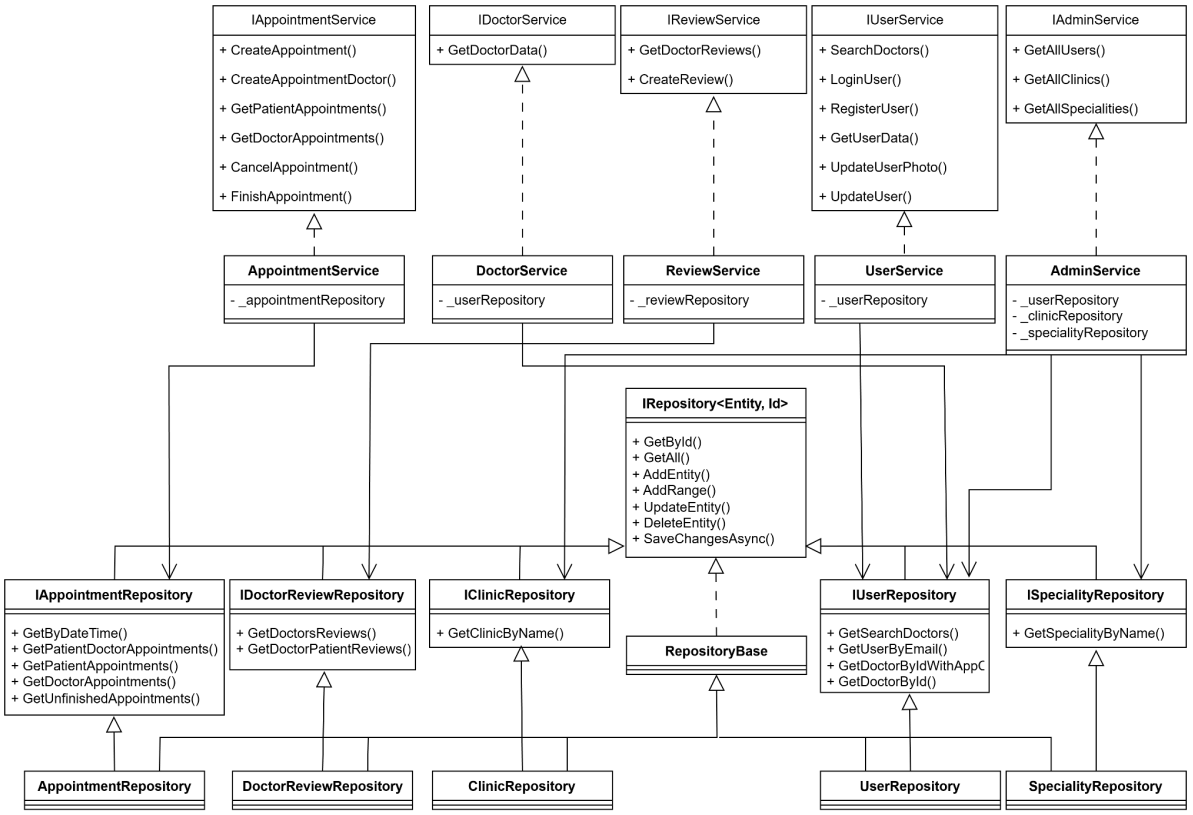

Рис.2 - Діаграма класів сервісів та репозиторіїв

На цих діаграмах:

Controller - базовий клас контролера.
ErrorController, MainController, DoctorController, AppointmentController, ReviewController, UserController, AdminController - контролери, які обробляють певні запити. Кожний метод контролера прив’язаний до певного шляху та методу http-запиту. Деякі контролери або їх методи захищені авторизацією за допомогою кукі.
IAppointmentService, IDoctorService, IReviewService, IUserService, IAdminService - інтерфейси сервісів для роботи з моделям для сутностей записів, користувачів, відгуків тощо.
AppointmentService, DoctorService, ReviewService, UserService, AdminService - конкретні класи-реалізації інтерфейсів сервісів. Використовують інтерфейси репозиторіїв для виконання операцій над сутностями в бд.
IRepository - базовий інтерфейс репозиторія для доступу до даних.
RepositoryBase - абстрактний клас базового репозиторія.

IAppointmentRepository, IDoctorReviewRepository, IClinicRepository, IUserRepository, ISpecialityRepository - інтерфейси репозиторіїв для конкретних сутностей.
AppointmentRepository, DoctorReviewRepository, ClinicRepository, UserRepository, SpecialityRepository - репозиторії для конкретних сутностей.

Приклад коду для обробки запитів пов’язаних з лікарями:

```
using BookingClinic.Application.Interfaces.Services;
using BookingClinic.Application.Interfaces.Helpers;
using BookingClinic.Application.Data.Doctor;
using BookingClinic.Application.Data.Appointment;
using BookingClinic.Application.Common;
using FluentValidation.AspNetCore;
using FluentValidation.Results;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Text.Json;

namespace BookingClinic.Controllers
{
    [Route("doctors")]
    Ссылок: 1
    public class DoctorController : Controller
    {
        private readonly IDoctorService _doctorService;
        private readonly IAppointmentService _appointmentService;
        private readonly IReviewsHelper _reviewsHelper;
        private readonly ISearchDoctorFacade _searchDoctorFacade;

        Ссылок: 0
        public DoctorController(
            IDoctorService doctorService,
            IAppointmentService appointmentService,
            IReviewsHelper reviewsHelper,
            ISearchDoctorFacade searchDoctorFacade)
        {
            _doctorService = doctorService;
            _appointmentService = appointmentService;
            _reviewsHelper = reviewsHelper;
            _searchDoctorFacade = searchDoctorFacade;
        }

        [HttpGet]
        Ссылок: 0
        public IActionResult Index([FromQuery] SearchDoctorDto dto, [FromQuery] int page)
        {
            var res = _searchDoctorFacade.SearchForDoctors(dto, page);

            ViewData["Specialities"] = res.Specialities;
            ViewData["Clinics"] = res.Clinics;
            ViewData["Sortings"] = res.Sortings;
            ViewData["Page"] = res.Page;
            ViewData["Pages"] = res.Pages;
```

Рис.3 - Клас DoctorController

```
            ViewData["Doctors"] = res.Doctors;
            ViewData["Errors"] = res.Errors;

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

        [HttpPost]
        [Authorize(AuthorizationPolicies.AuthorizedUserOnlyPolicy)]
```

Рис.4 - Клас DoctorController

```
85          [HttpPost]
86          [Authorize(AuthorizationPolicies.AuthorizedUserOnlyPolicy)]
            Ссылок: 0
87          public async Task<IActionResult> MakeAppointment(
88              [FromForm] MakeAppointmentDto dto)
89          {
90              var res = await _appointmentService.CreateAppointment(dto, User);
91
92              if (!res.IsSuccess)
93              {
94                  TempData["Errors"] = JsonSerializer.Serialize(res.Errors);
95                  return RedirectToAction("Profile", new { id = dto.DoctorId });
96              }
97              return RedirectToAction("Index", "User");
98          }
99
100         [HttpPost("docApp")]
101         [Authorize(AuthorizationPolicies.DoctorOnlyPolicy)]
            Ссылок: 0
102         public async Task<IActionResult> MakeAppointmentDoctor(
103             [FromForm] MakeAppointmentDocDto dto)
104         {
105             var res = await _appointmentService.CreateAppointmentDoctor(dto, User);
106
107             if (res.IsSuccess)
108             {
109                 return RedirectToAction("FinishedAppointment", "Appointment", new { dto.PatientId });
110             }
111             else
112             {
113                 TempData["Errors"] = JsonSerializer.Serialize(res.Errors);
114                 return RedirectToAction("Index", "Appointment");
115             }
116         }
117     }
118 }
119
```

Рис.5 - Клас DoctorController

```
1   using BookingClinic.Application.Common;
2   using BookingClinic.Application.Data.Doctor;
3
4   namespace BookingClinic.Application.Interfaces.Services
5   {
6       public interface IDoctorService
7       {
8           ServiceResult<DoctorDataDto> GetDoctorData(Guid doctorId);
9       }
10  }
11
```

Рис.6 - Інтерфейс IDoctorService

```
using BookingClinic.Application.Common;
using BookingClinic.Application.Data.Doctor;
using BookingClinic.Application.Interfaces.Repositories;
using BookingClinic.Application.Interfaces.Services;
using BookingClinic.Domain.Interfaces;
using Mapster;

namespace BookingClinic.Application.Services
{
    public class DoctorService : IDoctorService
    {
        private readonly IUserRepository _userRepository;
        private readonly IAppointmentDomainService _appointmentDomainService;


        public DoctorService(
            IUserRepository userRepository,
            IAppointmentDomainService appointmentDomainService)
        {
            _userRepository = userRepository;
            _appointmentDomainService = appointmentDomainService;
        }

        Ссылок: 3
        public ServiceResult<DoctorDataDto> GetDoctorData(Guid doctorId)
        {
            var doctor = _userRepository.GetDoctorByIdWithAppClinicSpeciality(doctorId);

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

            var res = doctor.Adapt<DoctorDataDto>();

            res.Appointments = _appointmentDomainService.GetAppointments(doctor);
            res.Rating = doctor.DoctorReviews.Select(r => r.Rating).DefaultIfEmpty(0).Average();

            return ServiceResult<DoctorDataDto>.Success(res);
        }
    }
}

```

Рис.7 - Клас DoctorService

```
namespace BookingClinic.Application.Interfaces.Repositories
{
    public interface IRepository<T, in TKey> where T : class
    {
        T? GetById(TKey key);
        IEnumerable<T> GetAll();
        void AddEntity(T entity);
        void AddRange(IEnumerable<T> entities);
        void UpdateEntity(T entity);
        void DeleteEntity(T entity);
        Task SaveChangesAsync();
    }
}
```

Рис.8 - Інтерфейс IRepository

```
using BookingClinic.Application.Interfaces.Repositories;
using BookingClinic.Infrastructure.AppContext;
using Microsoft.EntityFrameworkCore;

namespace BookingClinic.Infrastructure.Repositories
{
    public class RepositoryBase<T, TKey> : IRepository<T, TKey> where T : class
    {
        protected readonly ApplicationContext _context;
        protected readonly DbSet<T> _dbSet;

        public RepositoryBase(ApplicationContext context)
        {
            _context = context;
            _dbSet = _context.Set<T>();
        }

        public void AddEntity(T entity) => _dbSet.Add(entity);

        public void AddRange(IEnumerable<T> entities) => _dbSet.AddRange(entities);

        public void DeleteEntity(T entity) => _dbSet.Remove(entity);

        public IEnumerable<T> GetAll() => _dbSet.ToList();

        public T? GetById(TKey key) => _dbSet.Find(key);

        public Task SaveChangesAsync() => _context.SaveChangesAsync();

        public void UpdateEntity(T entity) => _dbSet.Update(entity);
    }
}
```

Рис.9 - Клас RepositoryBase

```
1   using BookingClinic.Domain.Entities;
2
3   namespace BookingClinic.Application.Interfaces.Repositories
4   {
5       public interface IUserRepository : IRepository<UserBase, Guid>
6       {
7           IEnumerable<UserBase> GetVisitorAdmins();
8           IEnumerable<UserBase> GetVisitorDoctors();
9           IEnumerable<UserBase> GetVisitorPatients();
            Ссылок: 2
10          IEnumerable<Doctor> GetSearchDoctors();
            Ссылок: 4
11          UserBase? GetUserByEmail(string email);
            Ссылок: 2
12          Doctor? GetDoctorByIdWithAppClinicSpeciality(Guid id);
            Ссылок: 3
13          Doctor? GetDoctorById(Guid id);
14      }
15  }
16
```

Рис.10 - Інтерфейс IUserRepository

```
using BookingClinic.Application.Interfaces.Repositories;
using BookingClinic.Domain.Entities;
using BookingClinic.Infrastructure.AppContext;
using Microsoft.EntityFrameworkCore;

namespace BookingClinic.Infrastructure.Repositories
{

    public class UserRepository : RepositoryBase<UserBase, Guid>, IUserRepository
    {

        public UserRepository(ApplicationContext context)
            : base(context)
        {
        }


        public IEnumerable<UserBase> GetVisitorAdmins() =>
            _dbSet.OfType<Admin>()
            .Include(a => a.ClientAppointments)
            .Include(a => a.ClientReviews)
            .ToList();

        Ссылок: 2
        public IEnumerable<UserBase> GetVisitorDoctors() =>
            _dbSet.OfType<Doctor>().Include(d => d.DoctorAppointments)
            .Include(d => d.DoctorReviews)
            .Include(d => d.Clinic)
            .Include(d => d.Speciality)
            .ToList();

        Ссылок: 2
        public IEnumerable<UserBase> GetVisitorPatients() =>
            _dbSet.OfType<Patient>().Include(p => p.ClientAppointments)
            .Include(p => p.ClientReviews)
            .ToList();

        Ссылок: 3
        public Doctor? GetDoctorById(Guid id) =>
            _dbSet.OfType<Doctor>().Include(d => d.Clinic).FirstOrDefault(d => d.Id == id);

        Ссылок: 2
        public Doctor? GetDoctorByIdWithAppClinicSpeciality(Guid id) =>
            _dbSet.OfType<Doctor>().Include(d => d.DoctorAppointments)
                                   .Include(d => d.Clinic)
                                   .Include(d => d.Speciality)
                                   .Include(d => d.DoctorReviews)
```

Рис.11 - Клас UserRepository

```
        public Doctor? GetDoctorByIdWithAppClinicSpeciality(Guid id) =>
            _dbSet.OfType<Doctor>().Include(d => d.DoctorAppointments)
                                   .Include(d => d.Clinic)
                                   .Include(d => d.Speciality)
                                   .Include(d => d.DoctorReviews)
                                   .FirstOrDefault(d => d.Id == id);

        Ссылок: 2
        public IEnumerable<Doctor> GetSearchDoctors() =>
            _dbSet.OfType<Doctor>().Include(d => d.Speciality).Include(d => d.Clinic).ToList();

        Ссылок: 4
        public UserBase? GetUserByEmail(string email) =>
            _dbSet.FirstOrDefault(u => u.Email == email);
    }
}
```

Рис.12 - Клас UserRepository

```
@using BookingClinic.Application.Data.Doctor
@model SearchDoctorDto;
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@{
    ViewData["Title"] = "Doctors";
    var data = (IEnumerable<SearchDoctorResDto>)ViewData["Doctors"];
    var pages = (IEnumerable<int>)ViewData["Pages"];
    var currentPage = (int)ViewData["Page"];
    var specialities = (IEnumerable<string>)ViewData["Specialities"];
    var clinics = (IEnumerable<string>)ViewData["Clinics"];
    var queryString = Context.Request.QueryString.Value;
    var sortings = (List<string>)ViewData["Sortings"];

    if (!string.IsNullOrEmpty(queryString))
    {
        var vals = queryString.Split('&');
        var filterVals = vals.Where(val => !val.Contains("page="));
        queryString = string.Join('&', filterVals);
    }
}

<form class="row g-2 align-items-center mb-4" method="get">
    <div class="col-md-4">
        <input type="text" class="form-control" asp-for="Query" placeholder="Name and surname" />
    </div>
    <div class="col-md-2">
        <select class="form-select" asp-for="Speciality">
            <option value="">Speciality</option>
            @foreach (var s in specialities)
            {
                <option value="@s">@s</option>
            }
        </select>
    </div>
    <div class="col-md-2">
        <select class="form-select" asp-for="Clinic">
            <option value="">Clinic</option>
            @foreach (var c in clinics)
            {
                <option value="@c">@c</option>
            }
        </select>
    </div>
    <div class="col-md-2">
        <select class="form-select" asp-for="OrderBy">
            <option value="">Sort by</option>
            @foreach (var s in sortings)
```

Рис.13 - Файл Index.cshtml

```
                <option value="">Sort by</option>
                @foreach (var s in sortings)
                {
                    <option value="@s">@s</option>
                }
            </select>
        </div>
        <div class="col-md-2">
            <button type="submit" class="btn btn-primary w-100">
                <img src="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/icons/search.svg" alt="Search" width="20" height="20" />
            </button>
        </div>
    </form>
    <div class="row row-cols-1 gy-4">
        @foreach (var doc in data)
        {
            <div class="col">
                <div class="card p-3 flex-row align-items-center">
                    <div class="me-4">
                        <img src="@(doc.ProfilePicture == null ? Url.Content("~/profiles/default/defaultUser.png") : Url.Content("~/prof
                             alt="Doctor"
                             class="rounded-circle"
                             style="width: 96px; height: 96px; object-fit: cover;" />
                    </div>
                    <div class="flex-grow-1">
                        <div><strong>Name:</strong> @doc.Name</div>
                        <div><strong>Surname:</strong> @doc.Surname</div>
                        <div><strong>Clinic:</strong> @doc.Clinic</div>
                        <div><strong>Speciality:</strong> @doc.Speciality</div>
                        <a class="btn btn-outline-primary mt-2" href="~/doctors/@doc.Id">Go to doctor</a>
                    </div>
                </div>
            </div>
        }
    </div>
    <nav class="mt-4">
        <ul class="pagination justify-content-center">
            @foreach (var p in pages)
            {
                var href = string.Empty;

                if (string.IsNullOrEmpty(queryString))
                {
                    href = $"./doctors?page={p}";
                }
                else
```

Рис.14 - Файл Index.cshtml

```
                }
                else
                {
                    href = $"./doctors{queryString}&page={p}";
                }
                @* if (!string.IsNullOrWhiteSpace(Model.OrderBy))
                {
                    href += $"&ordering={Model.OrderBy}";
                } *@
                <li class="page-item @(p == currentPage ? "active" : "")">
                    <a class="page-link" href="@href">@p</a>
                </li>
            }
        </ul>
    </nav>
```

Рис.15 - Файл Index.cshtml

```cshtml
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@using BookingClinic.Application
@using BookingClinic.Application.Data.Doctor
@model DoctorDataDto
@{
    ViewData["Title"] = "Doctor profile";
    var errors = ViewData["Errors"];
}

<form id="appointmentForm" method="post" asp-controller="Doctor" asp-action="MakeAppointment" class="d-none">
    <input type="hidden" name="doctorId" id="doctorIdInput" />
    <input type="hidden" name="appointmentDay" id="appointmentDayInput" />
    <input type="hidden" name="appointmentTime" id="appointmentTimeInput" />
</form>

<!-- Review modal -->
<div class="modal fade" id="reviewModal" tabindex="-1" aria-labelledby="reviewModalLabel" aria-hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <form method="post" asp-controller="Review" asp-action="CreateReview">
                <div class="modal-header">
                    <h5 class="modal-title" id="reviewModalLabel">Write a review</h5>
                    <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
                </div>
                <div class="modal-body">
                    <input type="hidden" name="DoctorId" value="@Model.Id" />
                    <div class="mb-3">
                        <label for="reviewText" class="form-label">Your review</label>
                        <textarea class="form-control" id="reviewText" name="Text" rows="5" required></textarea>
                    </div>
                    <span class="invalid-feedback"></span>
                    <div class="mb-3">
                        <label class="form-label">Rate</label>
                        <div>
                            @for (int i = 1; i <= 5; i++)
                            {
                                <div class="form-check form-check-inline">
                                    <input class="form-check-input" type="radio" name="Rating" id="rating-@i" value="@i" required>
                                    <label class="form-check-label" for="rating-@i">@i</label>
                                </div>
                            }
                        </div>
                    </div>
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancel</button>
```

Рис.16 - Файл Profile.cshtml

```
                        <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancel</button>
                        <button type="submit" class="btn btn-primary">Send</button>
                    </div>
                </form>
            </div>
        </div>
    </div>

    <!-- Confirm appointment modal -->
    <div class="modal fade" id="confirmModal" tabindex="-1" aria-labelledby="confirmModalLabel" aria-hidden="true">
        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header">
                    <h5 class="modal-title" id="confirmModalLabel">Confirm appointment</h5>
                    <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
                </div>
                <div class="modal-body" id="confirmModalBody">
                </div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancel</button>
                    <button type="button" class="btn btn-primary" id="confirmModalOk">Confirm</button>
                </div>
            </div>
        </div>
    </div>

    <!-- Main -->
    <div class="container mt-4">
        <div class="row">
            <div class="col-md-3 text-center">
                @if (Model.ProfilePicture == null)
                {
                    <img src="~/profiles/default/defaultUser.png" class="img-fluid rounded-circle" style="max-width: 180px;" />
                }
                else
                {
                    <img src="~/profile/users/@Model.ProfilePicture" class="img-fluid rounded-circle" style="max-width: 180px;" />
                }
            </div>
            <div class="col-md-9">

                @if (Model.CanWriteReview)
                {
                    <div class="alert alert-info d-flex align-items-center justify-content-between mb-3">
                        <span>How do you feel about this doctor? Write a review!</span>
                        <button type="button" class="btn btn-success ms-3" data-bs-toggle="modal" data-bs-target="#reviewModal">
                            Review
                        </button>
                    </div>
                }
                <h3>@Model.Name @Model.Surname</h3>
                <p><strong>Clinic:</strong> @Model.Clinic</p>
                <p><strong>Speciality:</strong> @Model.Speciality</p>
                @if (Model.Rating < 0.1)
                {
                    <p>No rating</p>
                }
                else
                {
                    <p>
                        <strong>Rating:</strong> @Model.Rating
                        <a asp-controller="Review" asp-action="Index" asp-route-id="@Model.Id" class="ms-2">
                            Check reviews
                        </a>
                    </p>
                }

                <div class="accordion mt-4" id="appointmentsAccordion">
                    @for (int i = 0; i < Model.Appointments.Count; i++)
                    {
                        var day = Model.Appointments[i];
                        var dayId = $"day{i}";
                        <div class="accordion-item">
                            <h2 class="accordion-header" id="heading-@dayId">
                                <button class="accordion-button collapsed" type="button" data-bs-toggle="collapse" data-bs-target="#collaps
                                    @day.Item1
                                </button>
                            </h2>
                            <div id="collapse-@dayId" class="accordion-collapse collapse" aria-labelledby="heading-@dayId" data-bs-parent="
                                <div class="accordion-body">
                                    <div class="d-flex flex-wrap gap-2">
                                        @foreach (var time in day.Item2)
                                        {
                                            <button type="button" class="btn btn-outline-primary"
                                                    onclick="confirmAppointment('@day.Item1', '@time', '@Model.Id')">
                                                @time
                                            </button>
                                        }
                                    </div>
                                </div>
```

Рис.17 - Файл Profile.cshtml

Рис.18 - Файл Profile.cshtml

```
                                </div>
                            </div>
                        }
                    </div>
                </div>
            </div>
        </div>

        <!-- Appointment and eror modals script -->
        <script>
            let appointmentData = {};

            function confirmAppointment(day, time, doctorId) {
                appointmentData = { day, time, doctorId };
                document.getElementById('confirmModalBody').innerText =
                    `Are you sure you want to assign on ${day} in ${time}?`;
                let modal = new bootstrap.Modal(document.getElementById('confirmModal'));
                modal.show();
            }

             document.addEventListener('DOMContentLoaded', function () {
                 document.getElementById('confirmModalOk').addEventListener('click', function () {
                     document.getElementById('doctorIdInput').value = appointmentData.doctorId;
                     document.getElementById('appointmentDayInput').value = appointmentData.day;
                     document.getElementById('appointmentTimeInput').value = appointmentData.time;
                     document.getElementById('appointmentForm').submit();
                 });
             });
        </script>
```

Рис.19 - Файл Profile.cshtml

Висновки:

У цій лабораторній роботі я ознайомився з різними типами взаємодії додатків та реалізував свій додаток за клієнт-серверною архітектурою. Клієнтом виступає інтерфейс у веб-браузері, який складається з html-сторінок і взаємодіє з сервером завдяки надсиланню HTTP запитів. Клієнт є тонким, оскільки не має якоїсь складної логіки обробки даних на своїй стороні. Клієнт надсилає лише пусті запити або запити з даними форм, сервер надсилає відповіді з html-сторінками. Вивчене розширило мої знання в патернах проектування та архітектурах систем.

Контрольні питання:

1. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура виділяє два види додатків: клієнт та сервер. Клієнт представляє додаток користувачеві та дозволяє робити запити на сервер, який надає відповіді та реалізує бізнес-логіку. Клієнти бувають

тонкими (всі операції обробки передають на сервер) та товстими (реалізує більшу частину обробки даних на стороні клієнта).

2. Розкажіть про сервіс-орієнтовану архітектуру.

Це архітектура, заснована на використанні розподілених, слабо пов'язаних сервісів або служб, оснащених стандартизованими інтерфейсами та протоколами для взаємодії. Сервіси взаємодіють між собою тільки за рахунок обміну повідомленнями, без створення спеціальних інтеграцій для доступу до однієї інформації, наприклад, до однієї бази даних.

3. Якими принципами керується SOA?

SOA керується такими принципами:
Слабка зв'язність - сервіси слабко пов'язані між собою.
Чітко визначена взаємодія - сервіси взаємодіють по чітко визначеним протоколам.
Повторне використання - сервіси можна повторно використати в різних системах.
Автономність сервісів - сервіси працюють незалежно та контролюють свій життєвий цикл.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси взаємодіють по стандартизованих протоколах (http, smtp, ftp тощо) із використанням SOAP або REST інтерфейсів. Повідомлення між сервісами мають стандартні формати, наприклад json або xml.

5. Як розробники взнають про існуючі сервіси і як робити до них запити?

Дізнатись про існуючий сервіс можна:

- Ручним пошуком у централізованому реєстрі сервісів.
- Динамічним виявленням сервіса у реєстрі.

Для того щоб зробити запит до сервісу потрібно знати його контракт. Найпоширенішими є SOAP та REST. Знаючи контракт розробник бачить якого формату запити потрібно надсилати: які необхідні параметри, які можливі помилки, які операції підтримуються.

6. У чому полягають переваги та недоліки клієнт-серверної моделі?

Перевагою є простота розгортання, тому що оновлювати потрібно лише сервери і в результаті клієнти з наступними запитами автоматично будуть працювати з оновленою системою. Також централізовано зберігаються дані і реалізується безпека, сервери можна масштабувати.
Недоліками є залежність системи від сервера, потреба у постійному з’єднанні, підвищеному навантаженні на сервер.

7. У чому полягають переваги та недоліки однорангової моделі взаємодії?

Переваги:
Немає центрального сервера, кожен учасник рівноправний.
Краще масштабування.
Вища стійкість до збої через відсутність вузького місця у вигляді сервера.
Нижча вартість через відсутність потужного сервера.
Недоліки:
Складніше управляти безпекою.
Можливі проблеми з синхронізацією даних.
Кожен учасник мусить управляти ресурсами.

8. Що таке мікросервісна архітектура?

Вона є підходом до створення серверного додатку як набору малих служб. Більше орієнтована на серверну частину, де кожна служба виконується у своєму процесі і взаємодіє з іншими службами за такими протоколами, як HTTP/HTTPS, WebSockets чи AMQP.

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?

Для обміну даними в мікросервісній архітектурі застосовуються протоколи HTTP/HTTPS (REST), gRPC, GraphQL, WebSocket, AMQP, MQTT та інші.

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, коли ми в проєкті між шаром веб-контролерів та шаром доступу до даних реалізуємо шар бізнес-логіки у вигляді сервісів?

Ні, бо це проста реалізація багатошарової архітектури певного додатку. При сервіс-орієнтованій архітектурі кожний сервіс виступає окремим додатком, який працює незалежно і може бути розгорнутим на окремому сервері, а також взаємодіє через мережу.