МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
[НАЗВА УНІВЕРСИТЕТУ]

Факультет [назва факультету]
Кафедра [назва кафедри]

КУРСОВА РОБОТА

з дисципліни «[Назва дисципліни]»

на тему:

«ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ АТЕЛЬЄ»

Виконав:
студент групи [номер групи]
Б студента

Перевірив:
Б викладача
сада, науковий ступінь

Місто, 2025

Зміст

1 ВСТУП

1.1 Актуальність теми

Сучасний розвиток інформаційних технологій створює нові можливості для автоматизації бізнес-процесів у різних галузях економіки. Ательє як підприємство, що надає послуги з пошиття та ремонту одягу, потребує ефективної системи управління для підвищення якості обслуговування клієнтів та оптимізації виробничих процесів.

Актуальність розробки інформаційної системи для ательє обумовлена наступними факторами:

- необхідність автоматизації обліку замовлень та клієнтів;
- потреба у контролі виконання замовлень та завантаженості працівників;
- важливість ведення складського обліку матеріалів;
- необхідність формування звітності для аналізу діяльності підприємства;
- підвищення конкурентоспроможності за рахунок покращення якості обслуговування.

1.2 Мета та завдання курсової роботи

Мета роботи: розробка інформаційної системи управління ательє для автоматизації основних бізнес-процесів підприємства.

Завдання роботи:

1. Проаналізувати предметну область та виявити основні бізнес-процеси ательє;
2. Визначити функціональні вимоги до інформаційної системи;
3. Розробити діаграми прецедентів для визначення взаємодії користувачів з системою;
4. Створити діаграму класів для опису структури системи;
5. Розробити модель бази даних;

6. Описати архітектуру системи;

7. Розробити діаграми діяльності для ключових бізнес-процесів.

1.3 Об'єкт та предмет дослідження

Об'єкт дослідження: бізнес-процеси ательє з пошиття та ремонту одягу.

Предмет дослідження: методи та засоби автоматизації управління діяльністю ательє.

2 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Опис предметної області

Ательє — це підприємство, що надає послуги з пошиття одягу на замовлення, ремонту та підгонки одягу за розміром клієнта. Основними процесами ательє є:

- **Прийом замовлень** — консультація клієнта, зняття мірок, вибір тканини та фасону, визначення вартості та термінів виконання;
- **Виробництво** — розкрій тканини, пошиття виробу, примірка та коригування;
- **Видача замовлень** — перевірка якості, розрахунок з клієнтом, видача готового виробу;
- **Складський облік** — облік тканин, фурнітури та інших матеріалів;
- **Управління персоналом** — розподіл замовлень між кравцями, облік робочого часу.

2.2 Основні сутності предметної області

1. **Клієнт** — особа, яка замовляє послуги ательє;
2. **Замовлення** — конкретне замовлення клієнта на виготовлення або ремонт виробу;
3. **Працівник** — співробітник ательє (кравець, закрійник, адміністратор);
4. **Послуга** — вид роботи (пошиття сукні, ремонт брюк, підгонка піджака тощо);
5. **Матеріал** — тканина, фурнітура та інші матеріали;
6. **Мірки** — індивідуальні параметри клієнта.

2.3 Функціональні вимоги до системи

Інформаційна система управління ательє повинна забезпечувати:

1. **Управління клієнтами:**
    - реєстрація нових клієнтів;
    - зберігання контактної інформації;
    - зберігання мірок клієнтів;
    - історія замовлень клієнта.
2. **Управління замовленнями:**
    - створення нових замовлень;
    - відстеження статусу виконання;
    - призначення виконавців;
    - розрахунок вартості;
    - контроль термінів виконання.
3. **Управління матеріалами:**
    - облік наявності матеріалів на складі;
    - облік витрачання матеріалів на замовлення;
    - контроль залишків та формування замовлень постачальникам.
4. **Управління персоналом:**
    - облік працівників;
    - розподіл замовлень між виконавцями;
    - контроль завантаженості.
5. **Звітність:**
    - звіт по виконаних замовленнях;
    - фінансовий звіт;
    - звіт по завантаженості працівників;
    - звіт по залишках матеріалів.

3 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Діаграма прецедентів

Діаграма прецедентів (use case diagram) відображає взаємодію користувачів з системою та основні функції, які вона надає.

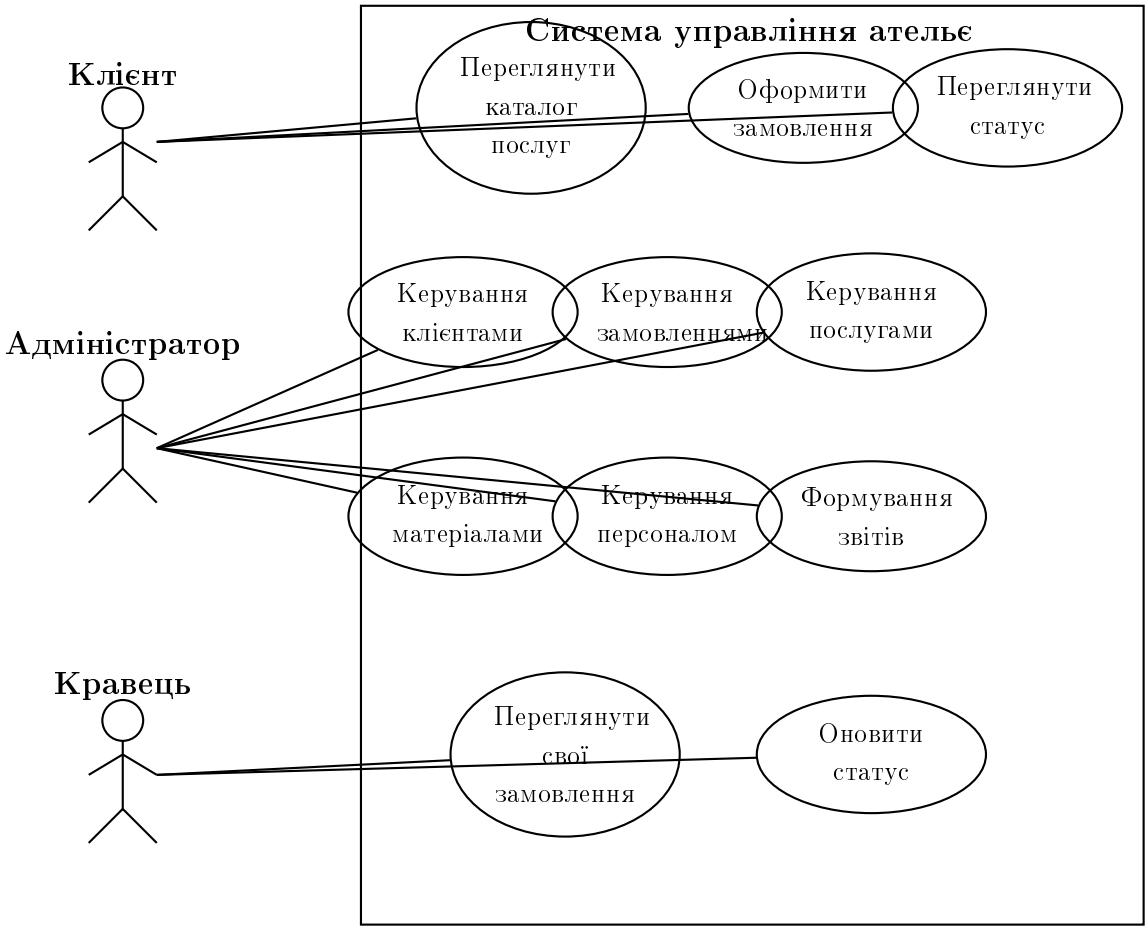

Рис. 1: Діаграма прецедентів інформаційної системи ательє

Опис основних прецедентів:

- **Оформити замовлення** — клієнт або адміністратор створює нове замовлення, вказуючи необхідні послуги, матеріали та терміни виконання;
- **Керування замовленнями** — адміністратор може створювати, редагувати та видаляти замовлення, призначати виконавців;
- **Оновити статус** — кравець змінює статус замовлення по мірі виконання роботи;
- **Формування звітів** — адміністратор формує різні види звітів для аналізу діяльності.

3.2 Діаграма класів

Діаграма класів відображає структуру системи, основні класи та зв'язки між ними.

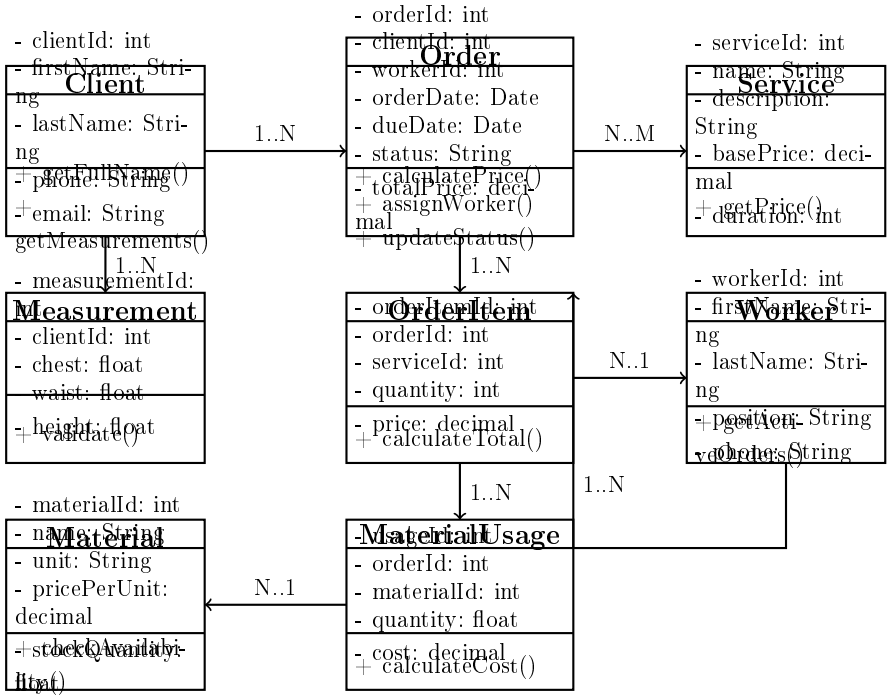

Рис. 2: Діаграма класів інформаційної системи

3.3 Модель бази даних

Концептуальна модель бази даних представлена у вигляді ER-діаграми, яка відображає основні сутності та зв'язки між ними.

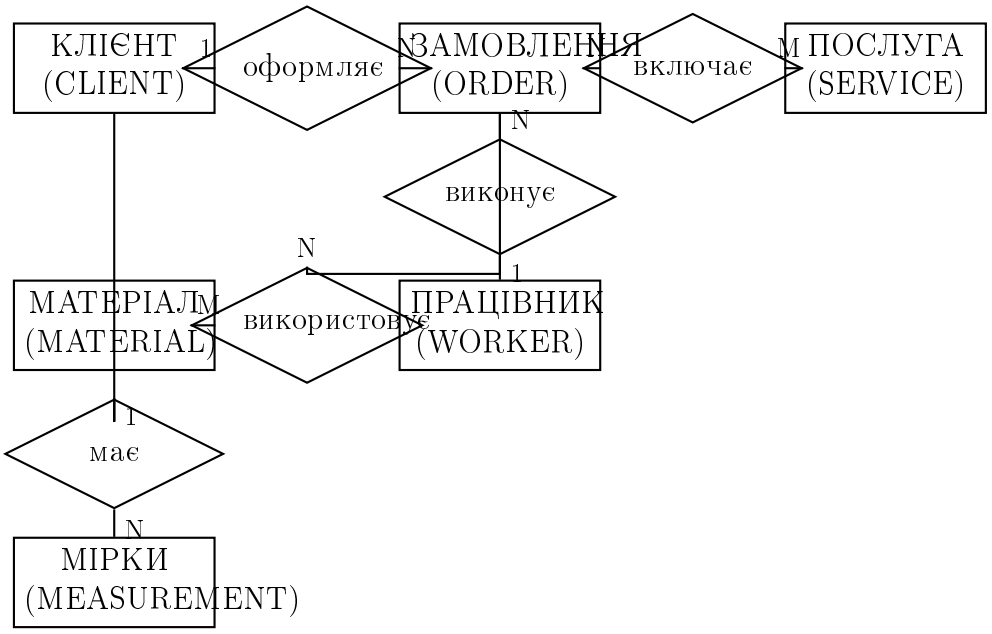

Рис. 3: ER-діаграма бази даних

3.3.1 Опис таблиць бази даних

Таблиця CLIENT (Клієнти):

- client_id (PK) — унікальний ідентифікатор клієнта
- first_name — ім'я
- last_name — прізвище
- phone — номер телефону
- email — електронна пошта
- registration_date — дата реєстрації
- notes — додаткові примітки

Таблиця MEASUREMENT (Мірки):

- measurement_id (PK) — унікальний ідентифікатор

- client_id (FK) — посилання на клієнта
- chest — обхват грудей
- waist — обхват талії
- hips — обхват стегон
- height — зріст
- measurement_date — дата зняття мірок

Таблиця WORKER (Працівники):

- worker_id (PK) — унікальний ідентифікатор
- first_name — ім'я
- last_name — прізвище
- position — посада
- phone — номер телефону
- hire_date — дата прийняття на роботу
- salary — заробітна плата

Таблиця SERVICE (Послуги):

- service_id (PK) — унікальний ідентифікатор
- name — назва послуги
- description — опис
- base_price — базова ціна
- duration_hours — тривалість виконання
- category — категорія послуги

Таблиця ORDER (Замовлення):

- order_id (PK) — унікальний ідентифікатор

- client_id (FK) — посилання на клієнта
- worker_id (FK) — посилання на виконавця
- order_date — дата оформлення
- due_date — термін виконання
- completion_date — дата завершення
- status — статус (новий, в роботі, готовий, виданий)
- total_price — загальна вартість
- notes — примітки

Таблиця ORDER_ITEM (Позиції замовлення):

- order_item_id (PK) — унікальний ідентифікатор
- order_id (FK) — посилання на замовлення
- service_id (FK) — посилання на послугу
- quantity — кількість
- price — ціна
- description — опис

Таблиця MATERIAL (Матеріали):

- material_id (PK) — унікальний ідентифікатор
- name — назва матеріалу
- unit — одиниця виміру
- price_per_unit — ціна за одиницю
- stock_quantity — кількість на складі
- min_stock_level — мінімальний рівень запасів
- supplier — постачальник

Таблиця MATERIAL_USAGE (Використання матеріалів):

- usage_id (PK) — унікальний ідентифікатор
- order_id (FK) — посилання на замовлення
- material_id (FK) — посилання на матеріал
- quantity — кількість використаного матеріалу
- cost — вартість
- usage_date — дата використання

3.4 Діаграма діяльності

Діаграма діяльності для процесу оформлення та виконання замовлення.

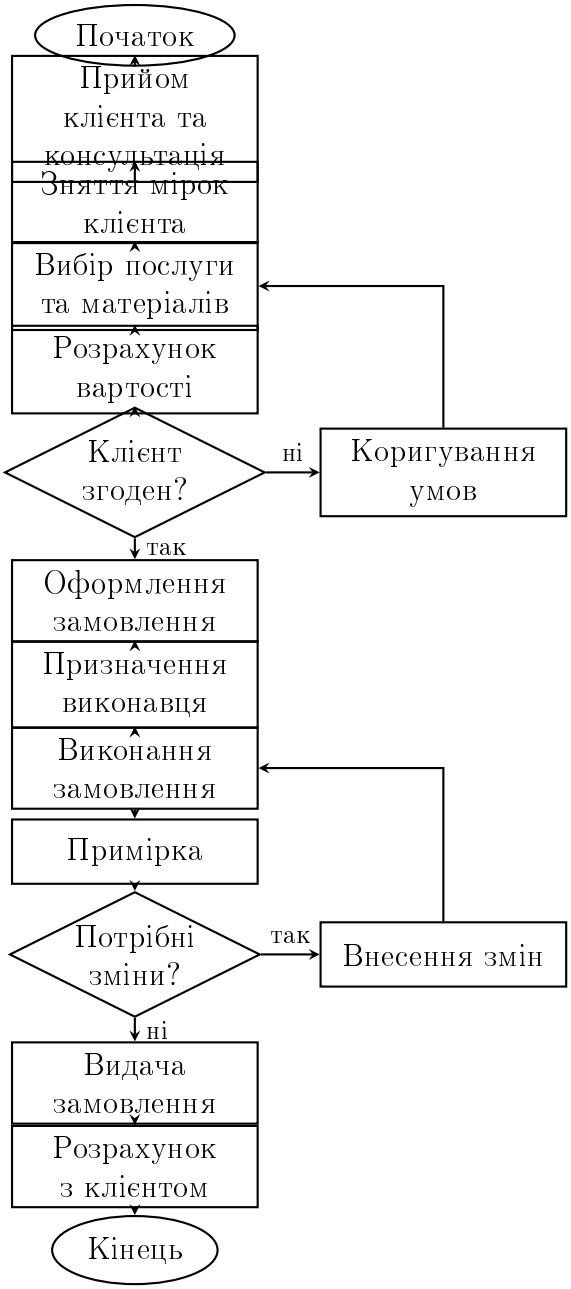

Рис. 4: Діаграма діяльності: процес оформлення та виконання замовлення

3.5 Архітектура системи

Інформаційна система управління ательє побудована за трирівневою архітектурою:

Рис. 5: Трирівнева архітектура системи

Опис рівнів:

1. **Рівень представлення:** забезпечує взаємодію користувача з системою через графічний інтерфейс. Може бути реалізований як веб-додаток або десктоп-програма.

2. **Рівень бізнес-логіки:** містить основну логіку роботи системи, обробляє запити користувачів, виконує валідацію даних, здійснює розрахунки та формує звіти.

3. **Рівень даних:** відповідає за зберігання та отримання даних з бази даних, забезпечує цілісність та безпеку інформації.

3.5.1 Діаграма компонентів

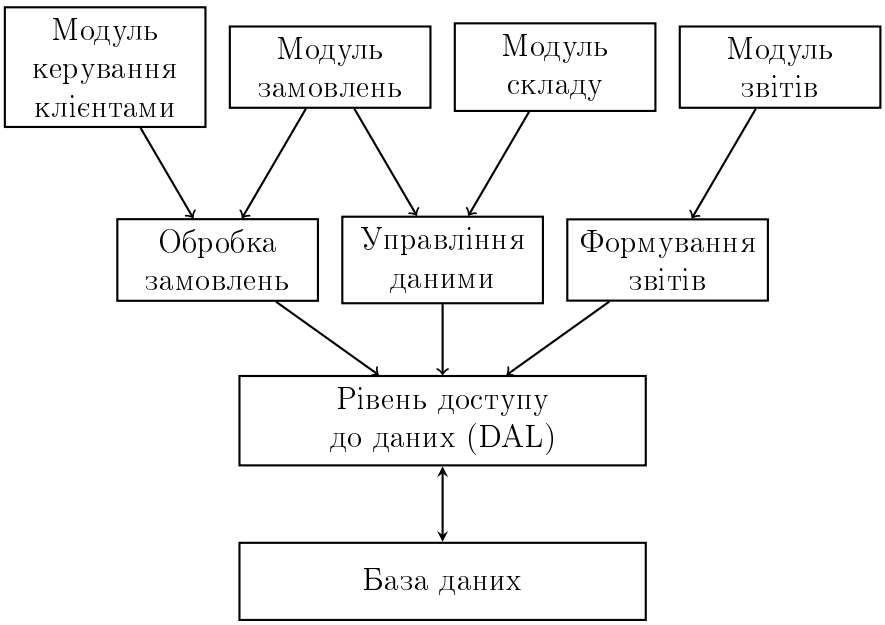

Рис. 6: Діаграма компонентів системи

3.6 Діаграма послідовності

Діаграма послідовності для процесу створення нового замовлення.

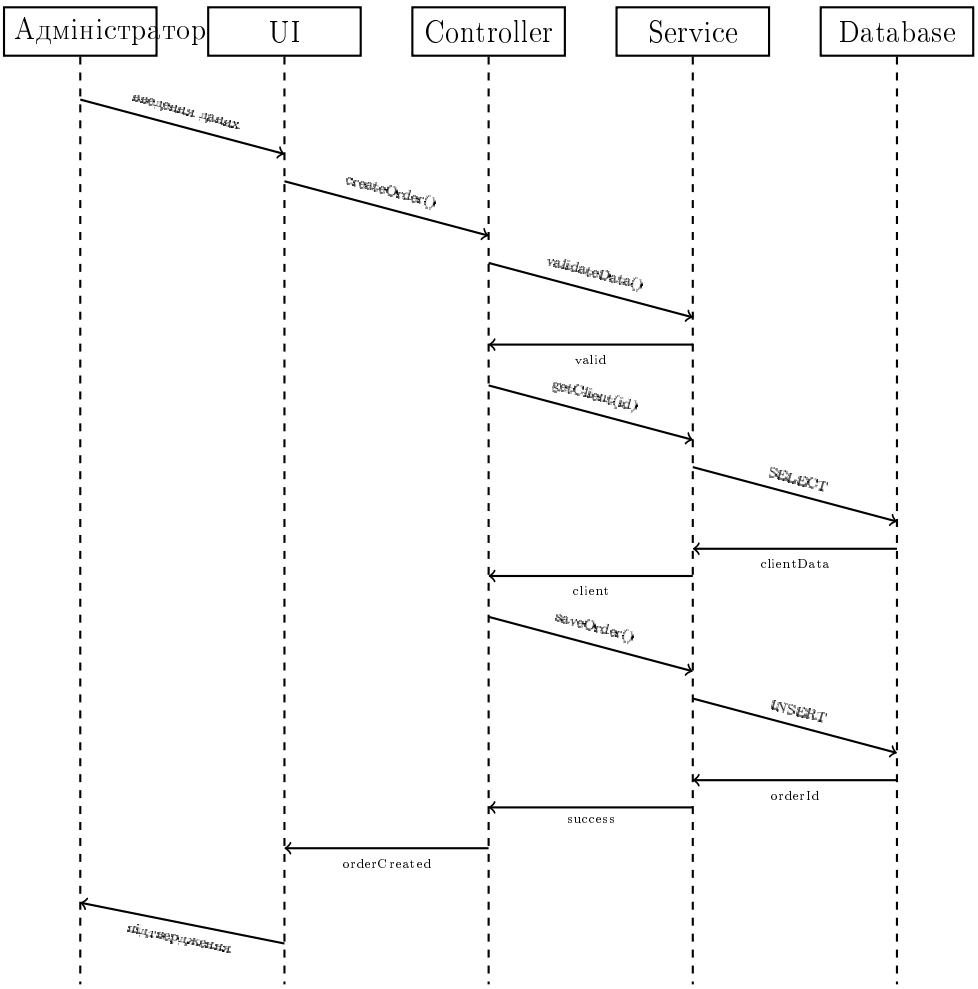

Рис. 7: Діаграма послідовності: створення замовлення

3.7 Діаграма станів

Діаграма станів для сутності "Замовлення"відображає можливі стани замовлення та переходи між ними.

Опис станів:

- **Новий** — замовлення створено, але ще не прийнято в роботу;
- **Прийнято в роботу** — замовлення призначено виконавцю;
- **В процесі виконання** — виконавець працює над замовленням;
- **Примірка** — клієнт проводить примірку виробу;
- **Готовий** — виріб готовий до видачі;
- **Виданий** — замовлення видано клієнту;
- **Скасовано** — замовлення скасовано клієнтом або адміністратором.

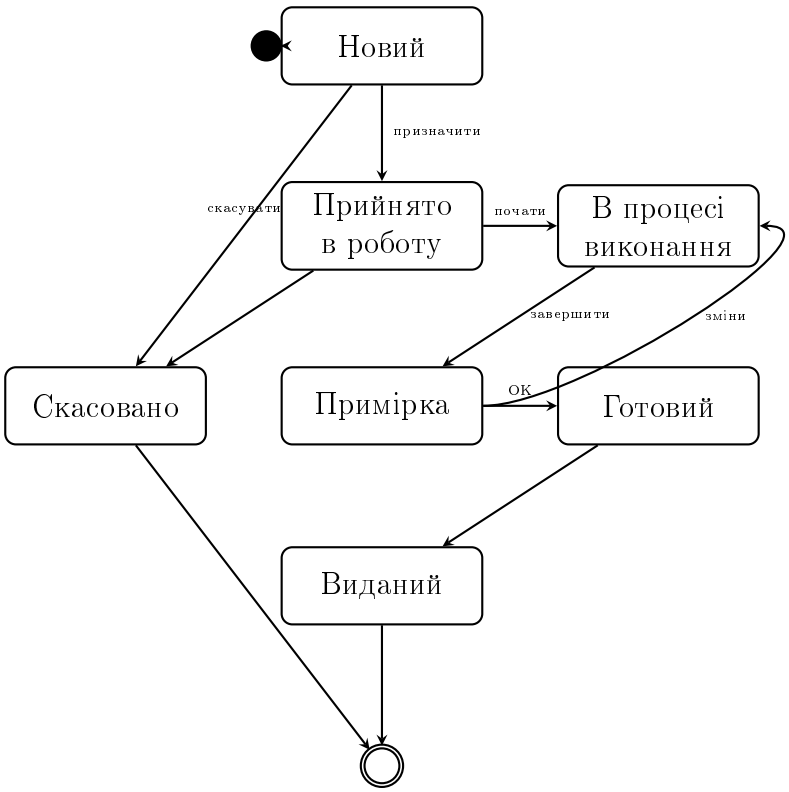

Рис. 8: Діаграма станів замовлення

4 РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Технології та інструменти

Для реалізації інформаційної системи управління ательє пропонується використовувати наступний стек технологій:

Backend (серверна частина):

- Мова програмування: C# (.NET 8)
- Фреймворк: ASP.NET Core 8.0
- ORM: Entity Framework Core 8.0
- База даних: SQL Server 2022 / PostgreSQL 16
- Архітектурний патерн: Clean Architecture
- Dependency Injection: вбудований DI контейнер .NET

Frontend (клієнтська частина):

- HTML5, CSS3, JavaScript

- Фреймворк: React / Angular / Vue.js
- UI-бібліотека: Material-UI / Bootstrap

Інструменти розробки:

- Система контролю версій: Git
- IDE: Visual Studio 2022 / JetBrains Rider
- Інструменти для тестування: xUnit, NUnit, Moq
- Документування API: Swagger / OpenAPI 3.0
- Управління пакетами: NuGet
- CI/CD: GitHub Actions, Azure DevOps

4.2 Структура проекту

Приклад структури проекту для веб-додатку:

```
AtelierManagementSystem/
 src/
    AtelierManagementSystem.Api/
       Controllers/          # API контролери
       Middleware/           # Middleware компоненти
       Filters/              # Фільтри запитів
       Program.cs            # Точка входу
    AtelierManagementSystem.Core/
       Entities/             # Сутності предметної області
       Interfaces/           # Інтерфейси
       Services/             # Бізнес-логіка
       Specifications/       # Специфікації запитів
    AtelierManagementSystem.Infrastructure/
       Data/                 # DbContext, міграції
       Repositories/         # Репозиторії
       Identity/             # Автентифікація
    AtelierManagementSystem.Shared/
```

```
        DTOs/                   # Data Transfer Objects
        Constants/              # Константи
tests/
   UnitTests/
   IntegrationTests/
AtelierManagementSystem.sln   # Solution файл
```

4.3 SQL-скрипти створення таблиць

Лістинг 1: Створення таблиць бази даних

```
-- Tablycya klientiv
CREATE TABLE client (
    client_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    phone VARCHAR(20),
    email VARCHAR(100),
    registration_date DATE DEFAULT CURRENT_DATE,
    notes TEXT
);

-- Tablycya mirok
CREATE TABLE measurement (
    measurement_id SERIAL PRIMARY KEY,
    client_id INTEGER REFERENCES client(client_id),
    chest DECIMAL(5,2),
    waist DECIMAL(5,2),
    hips DECIMAL(5,2),
    height DECIMAL(5,2),
    measurement_date DATE DEFAULT CURRENT_DATE
);

-- Tablycya pratsivnykiv
CREATE TABLE worker (
    worker_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    position VARCHAR(50),
    phone VARCHAR(20),
    hire_date DATE,
    salary DECIMAL(10,2)
);

-- Tablycya posluh
CREATE TABLE service (
    service_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
```

```
    base_price DECIMAL(10,2),
    duration_hours INTEGER,
    category VARCHAR(50)
);
```

Лістинг 2: Продовження створення таблиць

```
-- Tablycya zamovlen
CREATE TABLE "order" (
    order_id SERIAL PRIMARY KEY,
    client_id INTEGER REFERENCES client(client_id),
    worker_id INTEGER REFERENCES worker(worker_id),
    order_date DATE DEFAULT CURRENT_DATE,
    due_date DATE,
    completion_date DATE,
    status VARCHAR(20) DEFAULT 'new',
    total_price DECIMAL(10,2),
    notes TEXT
);

-- Tablycya pozytsiy zamovlennya
CREATE TABLE order_item (
    order_item_id SERIAL PRIMARY KEY,
    order_id INTEGER REFERENCES "order"(order_id),
    service_id INTEGER REFERENCES service(service_id),
    quantity INTEGER DEFAULT 1,
    price DECIMAL(10,2),
    description TEXT
);

-- Tablycya materialiv
CREATE TABLE material (
    material_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    unit VARCHAR(20),
    price_per_unit DECIMAL(10,2),
    stock_quantity DECIMAL(10,2),
    min_stock_level DECIMAL(10,2),
    supplier VARCHAR(100)
);

-- Tablycya vykorystannya materialiv
CREATE TABLE material_usage (
    usage_id SERIAL PRIMARY KEY,
    order_id INTEGER REFERENCES "order"(order_id),
    material_id INTEGER REFERENCES material(material_id),
    quantity DECIMAL(10,2),
```

```
    cost DECIMAL(10,2),
    usage_date DATE DEFAULT CURRENT_DATE
);
```

4.4 Приклади коду

Клас Order (C#):

Лістинг 3: Клас Order на C#

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace AtelierManagementSystem.Core.Entities
{
    public class Order
    {
        public int OrderId { get; set; }
        public int ClientId { get; set; }
        public int? WorkerId { get; set; }
        public DateTime OrderDate { get; set; }
        public DateTime DueDate { get; set; }
        public DateTime? CompletionDate { get; set; }
        public OrderStatus Status { get; set; }
        public decimal TotalPrice { get; set; }
        public string Notes { get; set; }

        // Navigation properties
        public virtual Client Client { get; set; }
        public virtual Worker Worker { get; set; }
        public virtual ICollection<OrderItem> OrderItems { get;
           set; }
        public virtual ICollection<MaterialUsage> MaterialUsages
           { get; set; }

        public Order()
        {
            OrderItems = new List<OrderItem>();
            MaterialUsages = new List<MaterialUsage>();
            OrderDate = DateTime.Now;
            Status = OrderStatus.New;
        }

        public decimal CalculateTotalPrice()
        {
            decimal itemsTotal = OrderItems?.Sum(i => i.Price * i
```

```
            .Quantity) ?? 0;
        decimal materialsTotal = MaterialUsages?.Sum(m => m.
            Cost) ?? 0;
        return itemsTotal + materialsTotal;
    }

    public void UpdateStatus(OrderStatus newStatus)
    {
        if (Status == OrderStatus.Completed || Status ==
            OrderStatus.Cancelled)
        {
            throw new InvalidOperationException(
                "Cannot␣change␣status␣of␣completed␣or␣
                    cancelled␣order");
        }

        Status = newStatus;

        if (newStatus == OrderStatus.Completed)
        {
            CompletionDate = DateTime.Now;
        }
    }

    public bool IsOverdue()
    {
        return DateTime.Now > DueDate &&
               Status != OrderStatus.Completed &&
               Status != OrderStatus.Cancelled;
    }
}

public enum OrderStatus
{
    New = 0,
    Accepted = 1,
    InProgress = 2,
    Fitting = 3,
    Ready = 4,
    Completed = 5,
    Cancelled = 6
```

```
    }
}
```

4.5 Реалізація репозиторіїв на C#

Для доступу до даних використовується патерн Repository з використанням Entity Framework Core.

Лістинг 4: Інтерфейс репозиторію

```
namespace AtelierManagementSystem.Core.Interfaces
{
    public interface IRepository<T> where T : class
    {
        Task<T> GetByIdAsync(int id);
        Task<IEnumerable<T>> GetAllAsync();
        Task<IEnumerable<T>> FindAsync(Expression<Func<T, bool>>
           predicate);
        Task AddAsync(T entity);
        Task UpdateAsync(T entity);
        Task DeleteAsync(int id);
        Task<int> SaveChangesAsync();
    }

    public interface IOrderRepository : IRepository<Order>
    {
        Task<IEnumerable<Order>> GetOrdersByClientIdAsync(int
           clientId);
        Task<IEnumerable<Order>> GetOrdersByWorkerIdAsync(int
           workerId);
        Task<IEnumerable<Order>> GetOrdersByStatusAsync(
           OrderStatus status);
        Task<IEnumerable<Order>> GetOverdueOrdersAsync();
    }
}
```

Лістинг 5: Реалізація репозиторію замовлень

```
using Microsoft.EntityFrameworkCore;

namespace AtelierManagementSystem.Infrastructure.Repositories
{
    public class OrderRepository : IOrderRepository
    {
        private readonly AtelierDbContext _context;

        public OrderRepository(AtelierDbContext context)
```

```
        {
            _context = context;
        }

        public async Task<Order> GetByIdAsync(int id)
        {
            return await _context.Orders
                .Include(o => o.Client)
                .Include(o => o.Worker)
                .Include(o => o.OrderItems)
                    .ThenInclude(oi => oi.Service)
                .Include(o => o.MaterialUsages)
                    .ThenInclude(mu => mu.Material)
                .FirstOrDefaultAsync(o => o.OrderId == id);
        }

        public async Task<IEnumerable<Order>>
           GetOrdersByStatusAsync(
            OrderStatus status)
        {
            return await _context.Orders
                .Include(o => o.Client)
                .Include(o => o.Worker)
                .Where(o => o.Status == status)
                .OrderBy(o => o.DueDate)
                .ToListAsync();
        }

        public async Task<IEnumerable<Order>>
           GetOverdueOrdersAsync()
        {
            return await _context.Orders
                .Include(o => o.Client)
                .Where(o => o.DueDate < DateTime.Now &&
                            o.Status != OrderStatus.Completed &&
                            o.Status != OrderStatus.Cancelled)
                .ToListAsync();
        }

        public async Task AddAsync(Order entity)
        {
```

```
            await _context.Orders.AddAsync(entity);
            await _context.SaveChangesAsync();
        }
    }
}
```

4.6 Конфігурація Entity Framework Core

Лістинг 6: DbContext для системи ательє

```
using Microsoft.EntityFrameworkCore;

namespace AtelierManagementSystem.Infrastructure.Data
{
    public class AtelierDbContext : DbContext
    {
        public AtelierDbContext(DbContextOptions<AtelierDbContext
           > options)
            : base(options)
        {
        }

        public DbSet<Client> Clients { get; set; }
        public DbSet<Measurement> Measurements { get; set; }
        public DbSet<Worker> Workers { get; set; }
        public DbSet<Service> Services { get; set; }
        public DbSet<Order> Orders { get; set; }
        public DbSet<OrderItem> OrderItems { get; set; }
        public DbSet<Material> Materials { get; set; }
        public DbSet<MaterialUsage> MaterialUsages { get; set; }

        protected override void OnModelCreating(ModelBuilder
           modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            // Client configuration
            modelBuilder.Entity<Client>(entity =>
            {
                entity.HasKey(e => e.ClientId);
                entity.Property(e => e.FirstName).IsRequired().
                   HasMaxLength(50);
                entity.Property(e => e.LastName).IsRequired().
                   HasMaxLength(50);
                entity.Property(e => e.Phone).HasMaxLength(20);
                entity.Property(e => e.Email).HasMaxLength(100);
                entity.HasIndex(e => e.Phone);
            });
```

```
            // Order configuration
            modelBuilder.Entity<Order>(entity =>
            {
                entity.HasKey(e => e.OrderId);
                entity.Property(e => e.TotalPrice).HasPrecision
                   (10, 2);
                entity.Property(e => e.Status)
                    .HasConversion<string>()
                    .HasMaxLength(20);

                entity.HasOne(e => e.Client)
                    .WithMany(c => c.Orders)
                    .HasForeignKey(e => e.ClientId)
                    .OnDelete(DeleteBehavior.Restrict);

                entity.HasOne(e => e.Worker)
                    .WithMany(w => w.Orders)
                    .HasForeignKey(e => e.WorkerId)
                    .OnDelete(DeleteBehavior.SetNull);
            });

            // Material configuration
            modelBuilder.Entity<Material>(entity =>
            {
                entity.HasKey(e => e.MaterialId);
                entity.Property(e => e.PricePerUnit).HasPrecision
                   (10, 2);
                entity.Property(e => e.StockQuantity).
                   HasPrecision(10, 2);
            });
        }
    }
}
```

4.7 API контролери на ASP.NET Core

Лістинг 7: OrdersController

```
using Microsoft.AspNetCore.Mvc;
using AtelierManagementSystem.Core.Interfaces;
using AtelierManagementSystem.Shared.DTOs;

namespace AtelierManagementSystem.Api.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class OrdersController : ControllerBase
    {
        private readonly IOrderService _orderService;
        private readonly ILogger<OrdersController> _logger;

        public OrdersController(
            IOrderService orderService,
            ILogger<OrdersController> logger)
        {
            _orderService = orderService;
            _logger = logger;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<OrderDto>>>
           GetOrders(
            [FromQuery] OrderStatus? status = null)
        {
            try
            {
                var orders = status.HasValue
                    ? await _orderService.GetOrdersByStatusAsync(
                       status.Value)
                    : await _orderService.GetAllOrdersAsync();

                return Ok(orders);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error retrieving orders");
```

```
            return StatusCode(500, "Internal␣server␣error");
        }
    }

    [HttpGet("{id}")]
    public async Task<ActionResult<OrderDto>> GetOrder(int id
       )
    {
        var order = await _orderService.GetOrderByIdAsync(id)
           ;

        if (order == null)
        {
            return NotFound($"Order␣with␣ID␣{id}␣not␣found");
        }

        return Ok(order);
    }

    [HttpPost]
    public async Task<ActionResult<OrderDto>> CreateOrder(
        CreateOrderDto createOrderDto)
    {
        try
        {
            var order = await _orderService.CreateOrderAsync(
               createOrderDto);
            return CreatedAtAction(
                nameof(GetOrder),
                new { id = order.OrderId },
                order);
        }
        catch (InvalidOperationException ex)
        {
            return BadRequest(ex.Message);
        }
    }

    [HttpPut("{id}/status")]
    public async Task<IActionResult> UpdateOrderStatus(
        int id,
```

```
            [FromBody] UpdateOrderStatusDto updateDto)
        {
            try
            {
                await _orderService.UpdateOrderStatusAsync(id,
                    updateDto.Status);
                return NoContent();
            }
            catch (InvalidOperationException ex)
            {
                return BadRequest(ex.Message);
            }
        }
    }
}
```

4.8 Додаткові діаграми системи

На наступних сторінках представлені детальні діаграми різних аспектів системи управління ательє.

2 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Опис предметної області

Ательє — це підприємство, що надає послуги з пошиття одягу на замовлення, ремонту та підгонки одягу за розміром клієнта. Основними процесами ательє є:

- **Прийом замовлень** — консультація клієнта, зняття мірок, вибір тканини та фасону, визначення вартості та термінів виконання;
- **Виробництво** — розкрій тканини, пошиття виробу, примірка та коригування;
- **Видача замовлень** — перевірка якості, розрахунок з клієнтом, видача готового виробу;
- **Складський облік** — облік тканин, фурнітури та інших матеріалів;
- **Управління персоналом** — розподіл замовлень між кравцями, облік робочого часу.

2.2 Основні сутності предметної області

1. **Клієнт** — особа, яка замовляє послуги ательє;
2. **Замовлення** — конкретне замовлення клієнта на виготовлення або ремонт виробу;
3. **Працівник** — співробітник ательє (кравець, закрійник, адміністратор);
4. **Послуга** — вид роботи (пошиття сукні, ремонт брюк, підгонка піджака тощо);
5. **Матеріал** — тканина, фурнітура та інші матеріали;
6. **Мірки** — індивідуальні параметри клієнта.

5

Рис. 9: Діаграма варіантів використання системи

2.3 Функціональні вимоги до системи

Інформаційна система управління ательє повинна забезпечувати:

1. **Управління клієнтами:**
   - реєстрація нових клієнтів;
   - зберігання контактної інформації;
   - зберігання мірок клієнтів;
   - історія замовлень клієнта.
2. **Управління замовленнями:**
   - створення нових замовлень;
   - відстеження статусу виконання;
   - призначення виконавців;
   - розрахунок вартості;
   - контроль термінів виконання.
3. **Управління матеріалами:**
   - облік наявності матеріалів на складі;
   - облік витрачання матеріалів на замовлення;
   - контроль залишків та формування замовлень постачальникам.
4. **Управління персоналом:**
   - облік працівників;
   - розподіл замовлень між виконавцями;
   - контроль завантаженості.
5. **Звітність:**
   - звіт по виконаних замовленнях;
   - фінансовий звіт;
   - звіт по завантаженості працівників;
   - звіт по залишках матеріалів.

Рис. 10: Діаграма взаємодії компонентів

3 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Діаграма прецедентів

Діаграма прецедентів (use case diagram) відображає взаємодію користувачів з системою та основні функції, які вона надає.

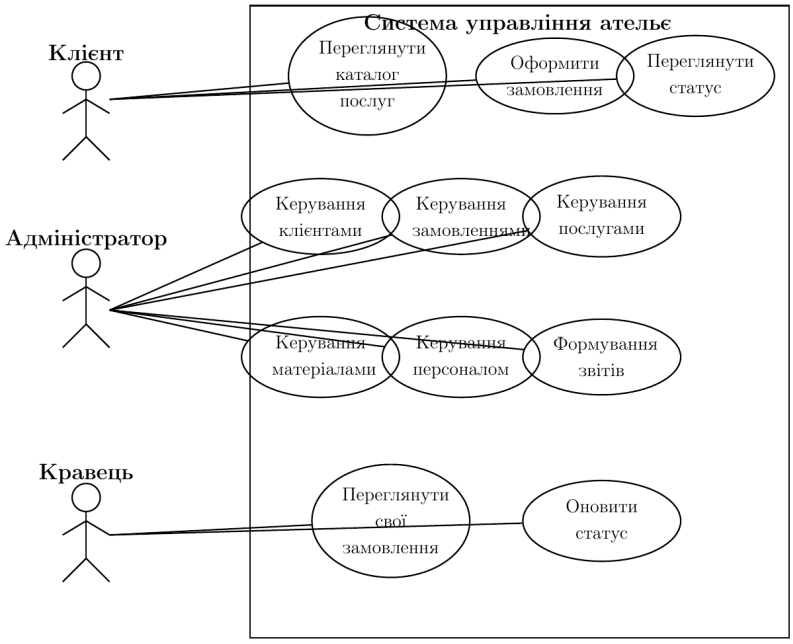

Рис. 1: Діаграма прецедентів інформаційної системи ательє

Опис основних прецедентів:

- **Оформити замовлення** — клієнт або адміністратор створює нове замовлення, вказуючи необхідні послуги, матеріали та терміни виконання;
- **Керування замовленнями** — адміністратор може створювати, редагувати та видаляти замовлення, призначати виконавців;
- **Оновити статус** — кравець змінює статус замовлення по мірі виконання роботи;
- **Формування звітів** — адміністратор формує різні види звітів для аналізу діяльності.

Рис. 11: Діаграма потоків даних

5 SQL ЗАПИТИ ТА СЕРВІСНА ЛОГІКА

5.1 SQL запити для роботи з базою даних

Система використовує складні SQL запити для отримання аналітичної інформації.

3.2 Діаграма класів

Діаграма класів відображає структуру системи, основні класи та зв'язки між ними.

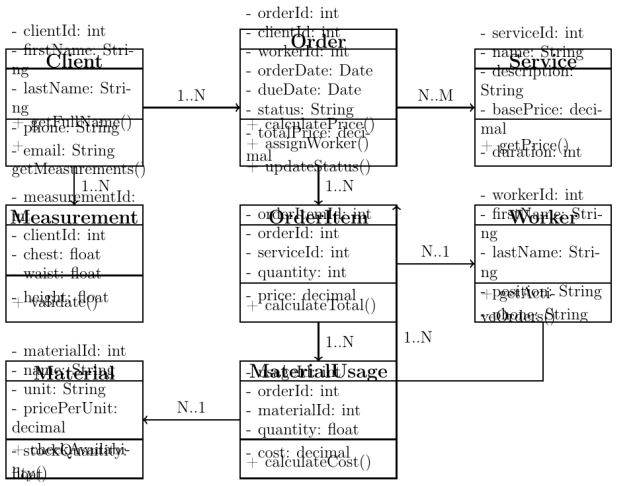

Рис. 2: Діаграма класів інформаційної системи

Рис. 12: Діаграма бізнес-процесів

Лістинг 8: Запит для звіту по виконаних замовленнях

```
-- Zvit po vykonanykh zamovlennyakh za period
SELECT
    o.order_id,
    CONCAT(c.first_name, ' ', c.last_name) AS client_name,
```

3.3 Модель бази даних

Концептуальна модель бази даних представлена у вигляді ER-діаграми, яка відображає основні сутності та зв'язки між ними.

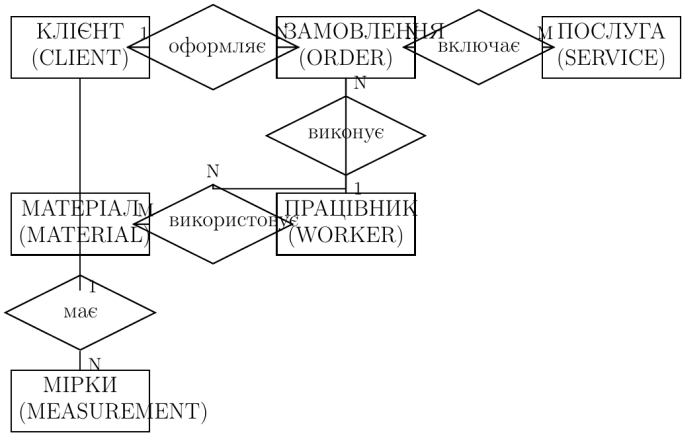

Рис. 3: ER-діаграма бази даних

3.3.1 Опис таблиць бази даних

Таблиця CLIENT (Клієнти):

- client_id (PK) — унікальний ідентифікатор клієнта
- first_name — ім'я
- last_name — прізвище
- phone — номер телефону
- email — електронна пошта
- registration_date — дата реєстрації
- notes — додаткові примітки

Таблиця MEASUREMENT (Мірки):

- measurement_id (PK) — унікальний ідентифікатор

Рис. 13: Діаграма архітектури додатку

```
CONCAT(w.first_name, ' ', w.last_name) AS worker_name,
o.order_date,
o.completion_date,
o.total_price,
STRING_AGG(s.name, ', ') AS services
```

- client_id (FK) — посилання на клієнта
- chest — обхват грудей
- waist — обхват талії
- hips — обхват стегон
- height — зріст
- measurement_date — дата зняття мірок

Таблиця WORKER (Працівники):

- worker_id (PK) — унікальний ідентифікатор
- first_name — ім'я
- last_name — прізвище
- position — посада
- phone — номер телефону
- hire_date — дата прийняття на роботу
- salary — заробітна плата

Таблиця SERVICE (Послуги):

- service_id (PK) — унікальний ідентифікатор
- name — назва послуги
- description — опис
- base_price — базова ціна
- duration_hours — тривалість виконання
- category — категорія послуги

Таблиця ORDER (Замовлення):

- order_id (PK) — унікальний ідентифікатор

Рис. 14: Діаграма безпеки системи

```
FROM "order" o
INNER JOIN client c ON o.client_id = c.client_id
LEFT JOIN worker w ON o.worker_id = w.worker_id
INNER JOIN order_item oi ON o.order_id = oi.order_id
INNER JOIN service s ON oi.service_id = s.service_id
```

- client_id (FK) — посилання на клієнта
- worker_id (FK) — посилання на виконавця
- order_date — дата оформлення
- due_date — термін виконання
- completion_date — дата завершення
- status — статус (новий, в роботі, готовий, виданий)
- total_price — загальна вартість
- notes — примітки

Таблиця ORDER_ITEM (Позиції замовлення):

- order_item_id (PK) — унікальний ідентифікатор
- order_id (FK) — посилання на замовлення
- service_id (FK) — посилання на послугу
- quantity — кількість
- price — ціна
- description — опис

Таблиця MATERIAL (Матеріали):

- material_id (PK) — унікальний ідентифікатор
- name — назва матеріалу
- unit — одиниця виміру
- price_per_unit — ціна за одиницю
- stock_quantity — кількість на складі
- min_stock_level — мінімальний рівень запасів
- supplier — постачальник

11

Рис. 15: Діаграма інтеграції з зовнішніми системами

```
WHERE o.status = 'Completed'
    AND o.completion_date BETWEEN '2025-01-01' AND '2025-12-31'
GROUP BY o.order_id, c.first_name, c.last_name,
         w.first_name, w.last_name
ORDER BY o.completion_date DESC;
```

Таблиця MATERIAL_USAGE (Використання матеріалів):

- usage_id (PK) — унікальний ідентифікатор
- order_id (FK) — посилання на замовлення
- material_id (FK) — посилання на матеріал
- quantity — кількість використаного матеріалу
- cost — вартість
- usage_date — дата використання

12

Рис. 16: Діаграма управління даними

Лістинг 9: Запит для аналізу продуктивності працівників

```
-- Analiz produktyvnosti pratsivnykiv
SELECT
    w.worker_id,
    CONCAT(w.first_name, ' ', w.last_name) AS worker_name,
```

3.4 Діаграма діяльності

Діаграма діяльності для процесу оформлення та виконання замовлення.

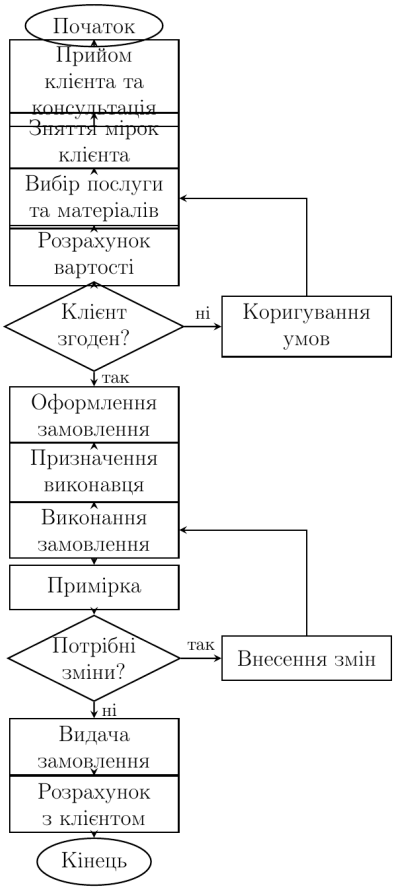

Рис. 4: Діаграма діяльності: процес оформлення та виконання замовлення

Рис. 17: Діаграма звітності системи

```
w.position,
COUNT(o.order_id) AS total_orders,
SUM(CASE WHEN o.status = 'Completed' THEN 1 ELSE 0 END)
    AS completed_orders,
SUM(CASE WHEN o.status = 'Completed' THEN o.total_price ELSE
```

3.5 Архітектура системи

Інформаційна система управління ательє побудована за трирівневою архітектурою:

Рис. 5: Трирівнева архітектура системи

Опис рівнів:

1. **Рівень представлення:** забезпечує взаємодію користувача з системою через графічний інтерфейс. Може бути реалізований як веб-додаток або десктоп-програма.
2. **Рівень бізнес-логіки:** містить основну логіку роботи системи, обробляє запити користувачів, виконує валідацію даних, здійснює розрахунки та формує звіти.
3. **Рівень даних:** відповідає за зберігання та отримання даних з бази даних, забезпечує цілісність та безпеку інформації.

3.5.1 Діаграма компонентів

Рис. 18: Діаграма моніторингу та логування

```
    0 END)
      AS total_revenue,
AVG(CASE WHEN o.status = 'Completed'
     THEN EXTRACT(DAY FROM (o.completion_date - o.order_date))
     ELSE NULL END) AS avg_completion_days
```

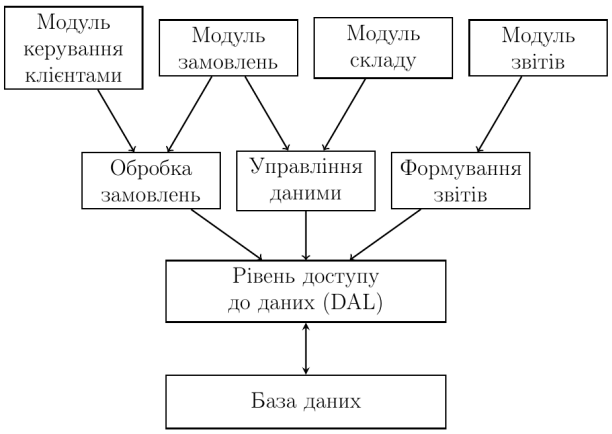

Рис. 6: Діаграма компонентів системи

3.6 Діаграма послідовності

Діаграма послідовності для процесу створення нового замовлення.

Рис. 19: Діаграма масштабування системи

```
FROM worker w
LEFT JOIN "order" o ON w.worker_id = o.worker_id
WHERE o.order_date >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY w.worker_id, w.first_name, w.last_name, w.position
ORDER BY total_revenue DESC;
```

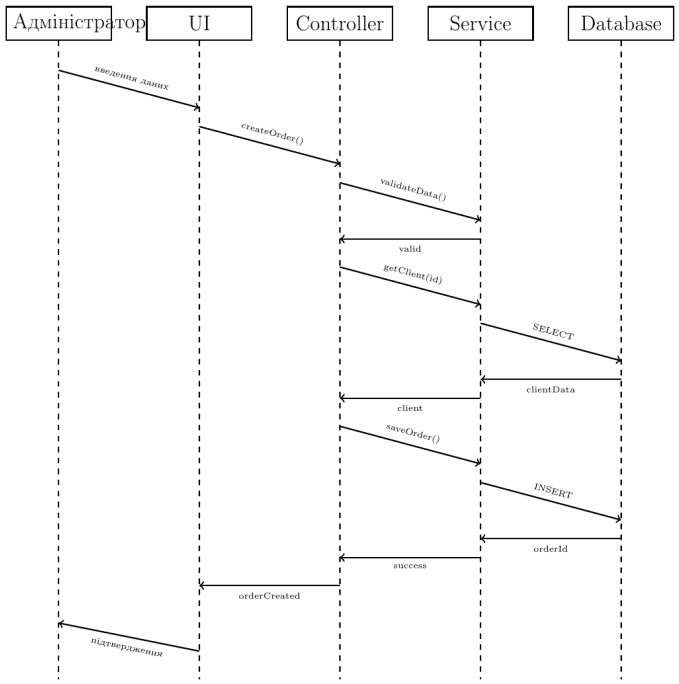

Рис. 7: Діаграма послідовності: створення замовлення

3.7 Діаграма станів

Діаграма станів для сутності "Замовлення"відображає можливі стани замовлення та переходи між ними.

Опис станів:

- **Новий** — замовлення створено, але ще не прийнято в роботу;
- **Прийнято в роботу** — замовлення призначено виконавцю;
- **В процесі виконання** — виконавець працює над замовленням;
- **Примірка** — клієнт проводить примірку виробу;
- **Готовий** — виріб готовий до видачі;
- **Виданий** — замовлення видано клієнту;
- **Скасовано** — замовлення скасовано клієнтом або адміністратором.

Рис. 20: Діаграма резервного копіювання

Лістинг 10: Запит для контролю залишків матеріалів

```
-- Kontrol zalyshkiv materialiv
SELECT
    m.material_id,
    m.name,
```

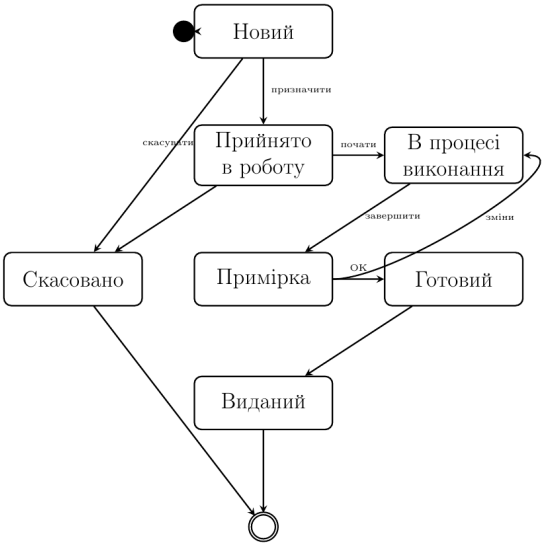

Рис. 8: Діаграма станів замовлення

4 РЕАЛІЗАЦІЯ СИСТЕМИ

4.1 Технології та інструменти

Для реалізації інформаційної системи управління ательє пропонується використовувати наступний стек технологій:

Backend (серверна частина):

- Мова програмування: Java / Python / C#
- Фреймворк: Spring Boot / Django / ASP.NET Core
- ORM: Hibernate / Django ORM / Entity Framework
- База даних: PostgreSQL / MySQL

Frontend (клієнтська частина):

- HTML5, CSS3, JavaScript
- Фреймворк: React / Angular / Vue.js
- UI-бібліотека: Material-UI / Bootstrap

Рис. 21: Діаграма продуктивності системи

```
m.unit,
m.stock_quantity,
m.min_stock_level,
m.price_per_unit,
CASE
```

```
        WHEN m.stock_quantity < m.min_stock_level
        THEN 'CRITICAL'
        WHEN m.stock_quantity < m.min_stock_level * 1.5
        THEN 'LOW'
        ELSE 'OK'
    END AS stock_status,
    (m.min_stock_level * 2 - m.stock_quantity)
        AS suggested_order_quantity,
    m.supplier
FROM material m
WHERE m.stock_quantity <= m.min_stock_level * 1.5
ORDER BY stock_status DESC, m.stock_quantity ASC;
```

Лістинг 11: Запит для фінансового звіту

```
-- Finansovyy zvit za period
WITH monthly_revenue AS (
    SELECT
        DATE_TRUNC('month', o.completion_date) AS month,
        SUM(o.total_price) AS revenue,
        COUNT(o.order_id) AS order_count
    FROM "order" o
    WHERE o.status = 'Completed'
        AND o.completion_date >= CURRENT_DATE - INTERVAL '12␣
           months'
    GROUP BY DATE_TRUNC('month', o.completion_date)
),
monthly_materials AS (
    SELECT
        DATE_TRUNC('month', mu.usage_date) AS month,
        SUM(mu.cost) AS material_cost
    FROM material_usage mu
    WHERE mu.usage_date >= CURRENT_DATE - INTERVAL '12␣months'
    GROUP BY DATE_TRUNC('month', mu.usage_date)
)
SELECT
    mr.month,
    mr.order_count,
    mr.revenue,
    COALESCE(mm.material_cost, 0) AS material_cost,
    (mr.revenue - COALESCE(mm.material_cost, 0)) AS profit
FROM monthly_revenue mr
```

```
LEFT JOIN monthly_materials mm ON mr.month = mm.month
ORDER BY mr.month DESC;
```

5.2 Сервісний шар на C#

Бізнес-логіка системи реалізована у сервісному шарі з використанням принципів SOLID.

Лістинг 12: OrderService з бізнес-логікою

```
namespace AtelierManagementSystem.Core.Services
{
    public class OrderService : IOrderService
    {
        private readonly IOrderRepository _orderRepository;
        private readonly IClientRepository _clientRepository;
        private readonly IMaterialRepository _materialRepository;
        private readonly ILogger<OrderService> _logger;

        public OrderService(
            IOrderRepository orderRepository,
            IClientRepository clientRepository,
            IMaterialRepository materialRepository,
            ILogger<OrderService> logger)
        {
            _orderRepository = orderRepository;
            _clientRepository = clientRepository;
            _materialRepository = materialRepository;
            _logger = logger;
        }

        public async Task<OrderDto> CreateOrderAsync(
           CreateOrderDto dto)
        {
            // Validate client exists
            var client = await _clientRepository.GetByIdAsync(dto
               .ClientId);
            if (client == null)
            {
                throw new InvalidOperationException(
                    $"Client␣with␣ID␣{dto.ClientId}␣not␣found");
            }

            // Check material availability
            foreach (var materialItem in dto.Materials)
            {
```

```
            var material = await _materialRepository
                .GetByIdAsync(materialItem.MaterialId);

            if (material == null)
            {
                throw new InvalidOperationException(
                    $"Material {materialItem.MaterialId} not
                        found");
            }

            if (material.StockQuantity < materialItem.
                Quantity)
            {
                throw new InvalidOperationException(
                    $"Insufficient stock for {material.Name}"
                        );
            }
        }

        // Create and configure order
        var order = new Order
        {
            ClientId = dto.ClientId,
            DueDate = dto.DueDate,
            Notes = dto.Notes,
            Status = OrderStatus.New
        };

        // Add order items and materials
        foreach (var item in dto.OrderItems)
        {
            order.OrderItems.Add(new OrderItem
            {
                ServiceId = item.ServiceId,
                Quantity = item.Quantity,
                Price = item.Price
            });
        }

        order.TotalPrice = order.CalculateTotalPrice();
        await _orderRepository.AddAsync(order);
```

```
            _logger.LogInformation(
                "Order␣{OrderId}␣created", order.OrderId);

            return MapToDto(order);
        }
    }
}
```

6 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ

6.1 Plan тестування

Тестування інформаційної системи включає наступні етапи:

1. **Модульне тестування (Unit Testing):**
    - Тестування окремих методів та функцій
    - Перевірка коректності обчислень
    - Валідація вхідних даних
2. **Інтеграційне тестування:**
    - Тестування взаємодії між модулями
    - Перевірка роботи з базою даних
    - Тестування API endpoints
3. **Системне тестування:**
    - Тестування повного функціоналу системи
    - Перевірка бізнес-процесів
    - Тестування продуктивності
4. **Приймальне тестування:**
    - Тестування користувачами
    - Перевірка відповідності вимогам
    - Виявлення зауважень

6.2 Приклади тестових сценаріїв

Тестовий сценарій 1: Створення нового замовлення

1. Адміністратор входить в систему
2. Вибирає клієнта зі списку або створює нового
3. Додає послуги до замовлення

4. Вказує необхідні матеріали

5. Система розраховує загальну вартість

6. Адміністратор вказує термін виконання

7. Система зберігає замовлення

8. Очікуваний результат: замовлення створено, статус "Новий"

Тестовий сценарій 2: Відстеження виконання замовлення

1. Кравець входить в систему

2. Переглядає список своїх замовлень

3. Вибирає замовлення в роботі

4. Оновлює статус на "В процесі виконання"

5. Після завершення змінює статус на "Готовий"

6. Очікуваний результат: статус оновлено, клієнт може бути повідомлений

6.3 Модульне тестування на C#

Для тестування використовуються фреймворки xUnit та Moq.

Лістинг 13: Тести для OrderService

```
using Xunit;
using Moq;
using FluentAssertions;

namespace AtelierManagementSystem.UnitTests.Services
{
    public class OrderServiceTests
    {
        private readonly Mock<IOrderRepository>
           _orderRepositoryMock;
        private readonly Mock<IClientRepository>
           _clientRepositoryMock;
        private readonly Mock<IMaterialRepository>
           _materialRepositoryMock;
        private readonly Mock<ILogger<OrderService>> _loggerMock;
        private readonly OrderService _sut;

        public OrderServiceTests()
        {
            _orderRepositoryMock = new Mock<IOrderRepository>();
            _clientRepositoryMock = new Mock<IClientRepository>()
               ;
            _materialRepositoryMock = new Mock<
               IMaterialRepository>();
            _loggerMock = new Mock<ILogger<OrderService>>();

            _sut = new OrderService(
                _orderRepositoryMock.Object,
                _clientRepositoryMock.Object,
                _materialRepositoryMock.Object,
                _loggerMock.Object);
        }

        [Fact]
        public async Task
           CreateOrderAsync_WithValidData_ShouldCreateOrder()
        {
```

```
        // Arrange
        var createDto = new CreateOrderDto
        {
            ClientId = 1,
            DueDate = DateTime.Now.AddDays(7),
            OrderItems = new List<CreateOrderItemDto>
            {
                new() { ServiceId = 1, Quantity = 1, Price =
                    100 }
            },
            Materials = new List<CreateMaterialUsageDto>()
        };

        var client = new Client
        {
            ClientId = 1,
            FirstName = "Ivan",
            LastName = "Petrov"
        };

        _clientRepositoryMock
            .Setup(r => r.GetByIdAsync(1))
            .ReturnsAsync(client);

        // Act
        var result = await _sut.CreateOrderAsync(createDto);

        // Assert
        result.Should().NotBeNull();
        result.ClientId.Should().Be(1);
        _orderRepositoryMock.Verify(
            r => r.AddAsync(It.IsAny<Order>()),
            Times.Once);
    }

    [Fact]
    public async Task
       CreateOrderAsync_WithInvalidClient_ShouldThrowException
       ()
    {
        // Arrange
```

```
            var createDto = new CreateOrderDto { ClientId = 999
                };

            _clientRepositoryMock
                .Setup(r => r.GetByIdAsync(999))
                .ReturnsAsync((Client)null);

            // Act & Assert
            await Assert.ThrowsAsync<InvalidOperationException>(
                () => _sut.CreateOrderAsync(createDto));
        }

        [Fact]
        public async Task UpdateOrderStatus_ShouldUpdateStatus()
        {
            // Arrange
            var order = new Order
            {
                OrderId = 1,
                Status = OrderStatus.New
            };

            _orderRepositoryMock
                .Setup(r => r.GetByIdAsync(1))
                .ReturnsAsync(order);

            // Act
            await _sut.UpdateOrderStatusAsync(1, OrderStatus.
                InProgress);

            // Assert
            order.Status.Should().Be(OrderStatus.InProgress);
            _orderRepositoryMock.Verify(
                r => r.UpdateAsync(order),
                Times.Once);
        }
    }
}
```

6.4 Інтеграційне тестування

Лістинг 14: Інтеграційні тести з базою даних

```
using Microsoft.EntityFrameworkCore;
using Xunit;

namespace AtelierManagementSystem.IntegrationTests
{
    public class OrderRepositoryIntegrationTests : IDisposable
    {
        private readonly AtelierDbContext _context;
        private readonly OrderRepository _repository;

        public OrderRepositoryIntegrationTests()
        {
            var options = new DbContextOptionsBuilder<
               AtelierDbContext>()
                .UseInMemoryDatabase(databaseName: "TestDb")
                .Options;

            _context = new AtelierDbContext(options);
            _repository = new OrderRepository(_context);

            SeedData();
        }

        private void SeedData()
        {
            var client = new Client
            {
                ClientId = 1,
                FirstName = "Test",
                LastName = "Client",
                Phone = "123456789"
            };

            var worker = new Worker
            {
                WorkerId = 1,
                FirstName = "Test",
                LastName = "Worker",
```

```
            Position = "Tailor"
        };

        var order = new Order
        {
            OrderId = 1,
            ClientId = 1,
            WorkerId = 1,
            OrderDate = DateTime.Now,
            DueDate = DateTime.Now.AddDays(5),
            Status = OrderStatus.New,
            TotalPrice = 500
        };

        _context.Clients.Add(client);
        _context.Workers.Add(worker);
        _context.Orders.Add(order);
        _context.SaveChanges();
    }

    [Fact]
    public async Task
       GetByIdAsync_ShouldReturnOrderWithRelations()
    {
        // Act
        var result = await _repository.GetByIdAsync(1);

        // Assert
        Assert.NotNull(result);
        Assert.Equal(1, result.OrderId);
        Assert.NotNull(result.Client);
        Assert.Equal("Test", result.Client.FirstName);
        Assert.NotNull(result.Worker);
    }

    [Fact]
    public async Task
       GetOrdersByStatusAsync_ShouldReturnFilteredOrders()
    {
        // Act
        var results = await _repository
```

```
                .GetOrdersByStatusAsync(OrderStatus.New);

            // Assert
            Assert.NotEmpty(results);
            Assert.All(results, o =>
                Assert.Equal(OrderStatus.New, o.Status));
        }

        public void Dispose()
        {
            _context.Database.EnsureDeleted();
            _context.Dispose();
        }
    }
}
```

6.5 Конфігурація та Dependency Injection

Лістинг 15: Конфігурація сервісів у Program.cs

```
using Microsoft.EntityFrameworkCore;
using AtelierManagementSystem.Infrastructure.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// Database configuration
builder.Services.AddDbContext<AtelierDbContext>(options =>
    options.UseSqlServer(
        builder.Configuration.GetConnectionString("
           DefaultConnection"),
        b => b.MigrationsAssembly("AtelierManagementSystem.
           Infrastructure")
    ));

// Register repositories
builder.Services.AddScoped<IOrderRepository, OrderRepository>();
builder.Services.AddScoped<IClientRepository, ClientRepository>()
   ;
builder.Services.AddScoped<IWorkerRepository, WorkerRepository>()
   ;
builder.Services.AddScoped<IMaterialRepository,
   MaterialRepository>();
builder.Services.AddScoped<IServiceRepository, ServiceRepository
   >();

// Register services
builder.Services.AddScoped<IOrderService, OrderService>();
builder.Services.AddScoped<IClientService, ClientService>();
builder.Services.AddScoped<IReportService, ReportService>();

// Add CORS
builder.Services.AddCors(options =>
{
```

```
    options.AddPolicy("AllowAll",
        policy => policy
            .AllowAnyOrigin()
            .AllowAnyMethod()
            .AllowAnyHeader());
});

// Add logging
builder.Services.AddLogging(config =>
{
    config.AddConsole();
    config.AddDebug();
});

var app = builder.Build();

// Configure the HTTP request pipeline
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseCors("AllowAll");
app.UseAuthorization();
app.MapControllers();

app.Run();
```

6.6 Конфігурація підключення до бази даних

Лістинг 16: appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=localhost;Database=AtelierDB;
      User Id=sa;Password=YourPassword;TrustServerCertificate=
         True"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning",
      "Microsoft.EntityFrameworkCore": "Information"
    }
  },
  "AllowedHosts": "*",
  "Jwt": {
    "Key": "YourSecretKeyForJWTAuthentication",
    "Issuer": "AtelierManagementSystem",
    "Audience": "AtelierUsers",
    "ExpiryInMinutes": 60
  },
  "EmailSettings": {
    "SmtpServer": "smtp.gmail.com",
    "SmtpPort": 587,
    "FromEmail": "noreply@atelier.com",
    "FromName": "Atelier Management System"
  }
}
```

6.7 Міграції бази даних

Entity Framework Core використовує міграції для управління схемою бази даних.

Лістинг 17: Початкова міграція

```
using Microsoft.EntityFrameworkCore.Migrations;

public partial class InitialCreate : Migration
{
```

```
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Clients",
            columns: table => new
            {
                ClientId = table.Column<int>(nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                FirstName = table.Column<string>(maxLength: 50,
                   nullable: false),
                LastName = table.Column<string>(maxLength: 50,
                   nullable: false),
                Phone = table.Column<string>(maxLength: 20,
                   nullable: true),
                Email = table.Column<string>(maxLength: 100,
                   nullable: true),
                RegistrationDate = table.Column<DateTime>(
                   nullable: false),
                Notes = table.Column<string>(nullable: true)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Clients", x => x.ClientId);
            });

        migrationBuilder.CreateIndex(
            name: "IX_Clients_Phone",
            table: "Clients",
            column: "Phone");
    }

    protected override void Down(MigrationBuilder
       migrationBuilder)
    {
        migrationBuilder.DropTable(name: "Clients");
    }
}
```

Команди для роботи з міграціями:

```
# Створення нової міграції
dotnet ef migrations add InitialCreate --project Infrastructure
```

```
# Застосування міграцій до бази даних
dotnet ef database update --project Infrastructure

# Відкат останньої міграції
dotnet ef migrations remove --project Infrastructure

# Створення SQL скрипту з міграцій
dotnet ef migrations script --project Infrastructure
```

6.8 Безпека та автентифікація

Лістинг 18: JWT автентифікація

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;

//    Program.cs              :
builder.Services.AddAuthentication(JwtBearerDefaults.
   AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new
           TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = builder.Configuration["Jwt:Issuer"],
            ValidAudience = builder.Configuration["Jwt:Audience"
               ],
            IssuerSigningKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(builder.Configuration["Jwt
                   :Key"]))
        };
    });

builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("AdminOnly", policy =>
        policy.RequireRole("Administrator"));
    options.AddPolicy("WorkerOrAdmin", policy =>
        policy.RequireRole("Administrator", "Worker"));
});
```

Лістинг 19: Сервіс генерації JWT токенів

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using Microsoft.IdentityModel.Tokens;
```

```
public class JwtTokenService
{
    private readonly IConfiguration _configuration;

    public JwtTokenService(IConfiguration configuration)
    {
        _configuration = configuration;
    }

    public string GenerateToken(User user)
    {
        var claims = new List<Claim>
        {
            new Claim(ClaimTypes.NameIdentifier, user.Id.ToString
               ()),
            new Claim(ClaimTypes.Name, user.Username),
            new Claim(ClaimTypes.Email, user.Email),
            new Claim(ClaimTypes.Role, user.Role)
        };

        var key = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));
        var credentials = new SigningCredentials(
            key, SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
            issuer: _configuration["Jwt:Issuer"],
            audience: _configuration["Jwt:Audience"],
            claims: claims,
            expires: DateTime.Now.AddMinutes(
                Convert.ToDouble(_configuration["Jwt:
                   ExpiryInMinutes"])),
            signingCredentials: credentials
        );

        return new JwtSecurityTokenHandler().WriteToken(token);
    }
}
```

6.9 Впровадження системи

Етапи впровадження:

1. **Підготовчий етап:**
    - Налаштування серверного обладнання
    - Встановлення програмного забезпечення
    - Створення бази даних
2. **Міграція даних:**
    - Перенесення існуючих даних про клієнтів
    - Імпорт каталогу послуг
    - Завантаження довідників
3. **Навчання персоналу:**
    - Проведення тренінгів для адміністраторів
    - Навчання кравців роботі з системою
    - Підготовка інструкцій користувача
4. **Пілотний запуск:**
    - Робота в тестовому режимі
    - Виявлення та усунення помилок
    - Збір зворотного зв'язку
5. **Промислова експлуатація:**
    - Повний перехід на нову систему
    - Технічна підтримка
    - Моніторинг роботи системи

7 ЗВІТНІСТЬ ТА МОНІТОРИНГ

7.1 Система звітів

Лістинг 20: Сервіс формування звітів

```
namespace AtelierManagementSystem.Core.Services
{
    public class ReportService : IReportService
    {
        private readonly AtelierDbContext _context;

        public async Task<FinancialReportDto>
           GenerateFinancialReportAsync(
            DateTime startDate, DateTime endDate)
        {
            var orders = await _context.Orders
                .Include(o => o.MaterialUsages)
                .Where(o => o.Status == OrderStatus.Completed &&
                            o.CompletionDate >= startDate &&
                            o.CompletionDate <= endDate)
                .ToListAsync();

            var totalRevenue = orders.Sum(o => o.TotalPrice);
            var totalMaterialCost = orders
                .SelectMany(o => o.MaterialUsages)
                .Sum(m => m.Cost);

            var ordersByMonth = orders
                .GroupBy(o => new {
                    o.CompletionDate.Value.Year,
                    o.CompletionDate.Value.Month
                })
                .Select(g => new MonthlyStats
                {
                    Year = g.Key.Year,
                    Month = g.Key.Month,
                    OrderCount = g.Count(),
                    Revenue = g.Sum(o => o.TotalPrice),
                    MaterialCost = g.SelectMany(o => o.
                       MaterialUsages)
                        .Sum(m => m.Cost)
```

```
            })
            .OrderBy(m => m.Year).ThenBy(m => m.Month)
            .ToList();

        return new FinancialReportDto
        {
            StartDate = startDate,
            EndDate = endDate,
            TotalOrders = orders.Count,
            TotalRevenue = totalRevenue,
            TotalMaterialCost = totalMaterialCost,
            NetProfit = totalRevenue - totalMaterialCost,
            MonthlyBreakdown = ordersByMonth
        };
    }

    public async Task<WorkerPerformanceReportDto>
        GenerateWorkerPerformanceReportAsync(int workerId,
           int days)
    {
        var startDate = DateTime.Now.AddDays(-days);

        var orders = await _context.Orders
            .Where(o => o.WorkerId == workerId &&
                        o.OrderDate >= startDate)
            .ToListAsync();

        var completedOrders = orders
            .Where(o => o.Status == OrderStatus.Completed)
            .ToList();

        var avgCompletionTime = completedOrders.Any()
            ? completedOrders.Average(o =>
                (o.CompletionDate.Value - o.OrderDate).
                   TotalDays)
            : 0;

        return new WorkerPerformanceReportDto
        {
            WorkerId = workerId,
            Period = days,
```

```
                TotalAssignedOrders = orders.Count,
                CompletedOrders = completedOrders.Count,
                InProgressOrders = orders.Count(o =>
                    o.Status == OrderStatus.InProgress),
                TotalRevenue = completedOrders.Sum(o => o.
                   TotalPrice),
                AverageCompletionDays = avgCompletionTime
            };
        }
    }
}
```

7.2 Логування та моніторинг

Лістинг 21: Конфігурація Serilog

```
using Serilog;
using Serilog.Events;

//    Program.cs
Log.Logger = new LoggerConfiguration()
    .MinimumLevel.Information()
    .MinimumLevel.Override("Microsoft", LogEventLevel.Warning)
    .MinimumLevel.Override("Microsoft.EntityFrameworkCore",
        LogEventLevel.Warning)
    .Enrich.FromLogContext()
    .Enrich.WithProperty("Application", "AtelierManagementSystem"
      )
    .WriteTo.Console()
    .WriteTo.File(
        "logs/atelier-.log",
        rollingInterval: RollingInterval.Day,
        outputTemplate:
            "{Timestamp:yyyy-MM-dd␣HH:mm:ss.fff␣zzz}␣" +
            "[{Level:u3}]␣{Message:lj}{NewLine}{Exception}")
    .WriteTo.Seq("http://localhost:5341")
    .CreateLogger();

builder.Host.UseSerilog();
```

Лістинг 22: Middleware для логування запитів

```
public class RequestLoggingMiddleware
{
    private readonly RequestDelegate _next;
    private readonly ILogger<RequestLoggingMiddleware> _logger;

    public RequestLoggingMiddleware(
        RequestDelegate next,
        ILogger<RequestLoggingMiddleware> logger)
    {
        _next = next;
        _logger = logger;
    }
```

```
    public async Task InvokeAsync(HttpContext context)
    {
        var sw = Stopwatch.StartNew();

        try
        {
            await _next(context);
            sw.Stop();

            _logger.LogInformation(
                "HTTP {Method} {Path} responded {StatusCode} in {
                   Elapsed}ms",
                context.Request.Method,
                context.Request.Path,
                context.Response.StatusCode,
                sw.ElapsedMilliseconds);
        }
        catch (Exception ex)
        {
            sw.Stop();

            _logger.LogError(ex,
                "HTTP {Method} {Path} failed after {Elapsed}ms",
                context.Request.Method,
                context.Request.Path,
                sw.ElapsedMilliseconds);

            throw;
        }
    }
}
```

7.3 Обробка помилок

Лістинг 23: Global Exception Handler

```
public class GlobalExceptionHandler : IExceptionHandler
{
    private readonly ILogger<GlobalExceptionHandler> _logger;

    public GlobalExceptionHandler(ILogger<GlobalExceptionHandler>
        logger)
    {
        _logger = logger;
    }

    public async ValueTask<bool> TryHandleAsync(
        HttpContext httpContext,
        Exception exception,
        CancellationToken cancellationToken)
    {
        _logger.LogError(exception,
            "An␣unhandled␣exception␣occurred:␣{Message}",
            exception.Message);

        var problemDetails = new ProblemDetails
        {
            Status = StatusCodes.Status500InternalServerError,
            Title = "An␣error␣occurred",
            Type = "https://tools.ietf.org/html/rfc7231#section
               -6.6.1"
        };

        switch (exception)
        {
            case InvalidOperationException:
                problemDetails.Status = StatusCodes.
                   Status400BadRequest;
                problemDetails.Title = "Invalid␣operation";
                problemDetails.Detail = exception.Message;
                break;

            case KeyNotFoundException:
```

```
                problemDetails.Status = StatusCodes.
                    Status404NotFound;
                problemDetails.Title = "Resource not found";
                problemDetails.Detail = exception.Message;
                break;

            default:
                problemDetails.Detail =
                    "An unexpected error occurred. Please try
                        again later.";
                break;
        }

        httpContext.Response.StatusCode = problemDetails.Status.
            Value;
        await httpContext.Response
            .WriteAsJsonAsync(problemDetails, cancellationToken);

        return true;
    }
}
```

7.4 Кешування даних

Лістинг 24: Використання Redis для кешування

```
using StackExchange.Redis;

public class CacheService : ICacheService
{
    private readonly IConnectionMultiplexer _redis;
    private readonly IDatabase _database;
    private readonly ILogger<CacheService> _logger;

    public CacheService(
        IConnectionMultiplexer redis,
        ILogger<CacheService> logger)
    {
        _redis = redis;
        _database = redis.GetDatabase();
        _logger = logger;
    }

    public async Task<T> GetOrSetAsync<T>(
        string key,
        Func<Task<T>> factory,
        TimeSpan expiry)
    {
        var cachedValue = await _database.StringGetAsync(key);

        if (!cachedValue.IsNullOrEmpty)
        {
            _logger.LogInformation("Cache hit for key: {Key}",
               key);
            return JsonSerializer.Deserialize<T>(cachedValue);
        }

        _logger.LogInformation("Cache miss for key: {Key}", key);
        var value = await factory();

        var serialized = JsonSerializer.Serialize(value);
        await _database.StringSetAsync(key, serialized, expiry);

        return value;
```

```
    }

    public async Task RemoveAsync(string key)
    {
        await _database.KeyDeleteAsync(key);
        _logger.LogInformation("Removed cache key: {Key}", key);
    }
}

//
public async Task<IEnumerable<ServiceDto>> GetAllServicesAsync()
{
    return await _cacheService.GetOrSetAsync(
        "services:all",
        async () =>
        {
            var services = await _serviceRepository.GetAllAsync()
                ;
            return _mapper.Map<IEnumerable<ServiceDto>>(services)
                ;
        },
        TimeSpan.FromHours(1));
}
```

8 ВИСНОВКИ

В результаті виконання курсової роботи була розроблена інформаційна система управління ательє, яка дозволяє автоматизувати основні бізнес-процеси підприємства.

Основні результати роботи:

1. Проведено аналіз предметної області та виявлено основні бізнес-процеси ательє, що потребують автоматизації.
2. Визначено функціональні вимоги до інформаційної системи, які включають управління клієнтами, замовленнями, матеріалами, персоналом та формування звітності.
3. Розроблено діаграму прецедентів, що відображає взаємодію трьох типів користувачів з системою: клієнта, адміністратора та кравця.
4. Створено діаграму класів, яка описує структуру системи та зв'язки між основними сутностями.
5. Спроектовано модель бази даних, що складається з 8 таблиць та забезпечує зберігання всієї необхідної інформації.
6. Розроблено діаграму діяльності для процесу оформлення та виконання замовлення, що демонструє послідовність дій.
7. Описано трирівневу архітектуру системи, яка забезпечує розділення відповідальності та масштабованість.
8. Розроблено діаграму станів для сутності "Замовлення що визначає життєвий цикл замовлення.
9. Підготовлено SQL-скрипти для створення таблиць бази даних та приклади коду основних класів.

Переваги розробленої системи:

- Автоматизація обліку клієнтів та їх замовлень
- Спрощення процесу оформлення та відстеження замовлень

- Контроль витрачання матеріалів та складських залишків
- Ефективний розподіл навантаження між працівниками
- Можливість формування аналітичних звітів
- Покращення якості обслуговування клієнтів
- Підвищення продуктивності роботи ательє

Перспективи подальшого розвитку:

- Додавання мобільного додатку для клієнтів
- Інтеграція з онлайн-оплатою
- Впровадження системи лояльності для постійних клієнтів
- Додавання модуля для роботи з постачальниками
- Інтеграція з бухгалтерськими системами
- Впровадження аналітики на основі машинного навчання для прогнозування попиту

Розроблена інформаційна система є повноцінним інструментом для управління ательє та може бути впроваджена на підприємствах різного масштабу.

9 СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2020. — 368 с.

2. Фаулер М. UML. Основы. — 3-е изд. — СПб.: Символ-Плюс, 2019. — 192 с.

3. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. — 3-е изд. — М.: Вильямс, 2018. — 1440 с.

4. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. — 2-е изд. — М.: ДМК Пресс, 2019. — 496 с.

5. Вендров А.М. Проектирование программного обеспечения экономических информационных систем. — М.: Финансы и статистика, 2018. — 544 с.

6. Troelsen Э., Джепикс Ф. Язык программирования C# 10 и платформа .NET 6. — 10-е изд. — М.: Диалектика, 2022. — 1440 с.

7. Смит Дж. Entity Framework Core в действии. — 2-е изд. — СПб.: Питер, 2021. — 528 с.

8. Фримен А. ASP.NET Core 6. Разработка приложений. — 7-е изд. — СПб.: БХВ-Петербург, 2022. — 1056 с.

9. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. — К.: Держстандарт України, 1995.

10. Офіційна документація Microsoft .NET [Електронний ресурс]. — Режим доступу: https://docs.microsoft.com/dotnet/

11. Офіційна документація Entity Framework Core [Електронний ресурс]. — Режим доступу: https://docs.microsoft.com/ef/core/

12. Офіційна документація ASP.NET Core [Електронний ресурс]. — Режим доступу: https://docs.microsoft.com/aspnet/core/

13. Офіційна документація SQL Server [Електронний ресурс]. — Режим доступу: https://docs.microsoft.com/sql/

14. Microsoft C# Programming Guide [Електронний ресурс]. — Режим доступу: https://docs.microsoft.com/dotnet/csharp/

15. Sommerville I. Software Engineering. — 10th ed. — Pearson, 2019. — 816 p.

16. Pressman R.S., Maxim B.R. Software Engineering: A Practitioner's Approach. — 9th ed. — McGraw-Hill Education, 2020. — 976 p.

17. Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. — Prentice Hall, 2017. — 432 p.

18. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. — Addison-Wesley, 2003. — 560 p.

А ДОДАТОК А. Діаграма розгортання

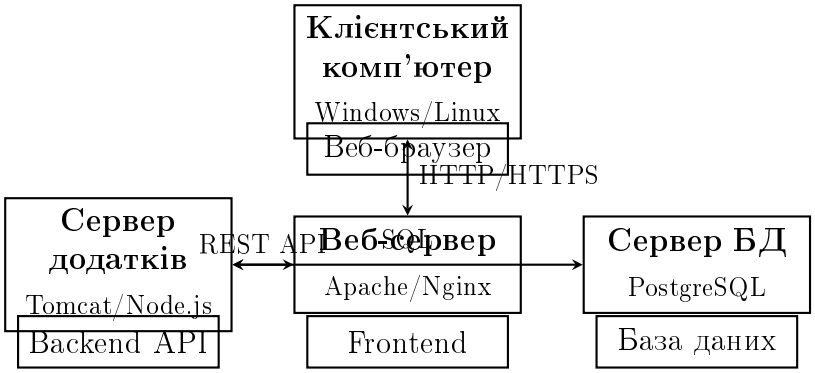

Рис. 22: Діаграма розгортання системи

Б ДОДАТОК Б. Словник термінів

- **Ательє** — підприємство, що надає послуги з пошиття та ремонту одягу
- **API** (Application Programming Interface) — інтерфейс програмування додатків
- **Backend** — серверна частина додатку
- **Frontend** — клієнтська частина додатку
- **CRUD** — Create, Read, Update, Delete (базові операції з даними)
- **ER-діаграма** — діаграма "сутність-зв'язок"
- **ORM** (Object-Relational Mapping) — об'єктно-реляційне відображення
- **REST** (Representational State Transfer) — архітектурний стиль для веб-сервісів
- **SQL** (Structured Query Language) — мова структурованих запитів
- **UML** (Unified Modeling Language) — уніфікована мова моделювання