Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
Кафедра Програмної Інженерії

ЗВІТ

з дисципліни «Архітектура програмного забезпечення»

з лабораторної роботи №2

Виконав
ст. гр. ПЗПІ-20-7
Крупчак Євгеній

Перевірив:
Старш. викл. кафедри ПІ
Сокорчук І. П.

Харків 2022

ЛАБОРАТОРНА РОБОТА №2. РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ

1.1 Мета роботи

Метою першої лабораторної роботи є розробка серверної частини для проєкту за темою «Програмна система для автоматизації видачі боксів із їжею».

1.2 Хід роботи

Бекенд частина додатку написана на мові програмування C# за допомогою ASP.NET Core 7. У якості бази даних використано MS SQL Server. Для безпечності використовуються: CORS, система аутентифікації та авторизації на основі ASP.NET Identity Core. Також використовуються:

1) FluentResults для генерації відповідей від сервісів;
2) MailKit та MimeKit для відправлення електронних листів;
3) Mapster для маппінгу об’єктів;
4) MQTTnet для комунікації між сервером та IoT;
5) Entity Framework Core як ORM.

На рисунку 1 зображено діаграму розгортання.

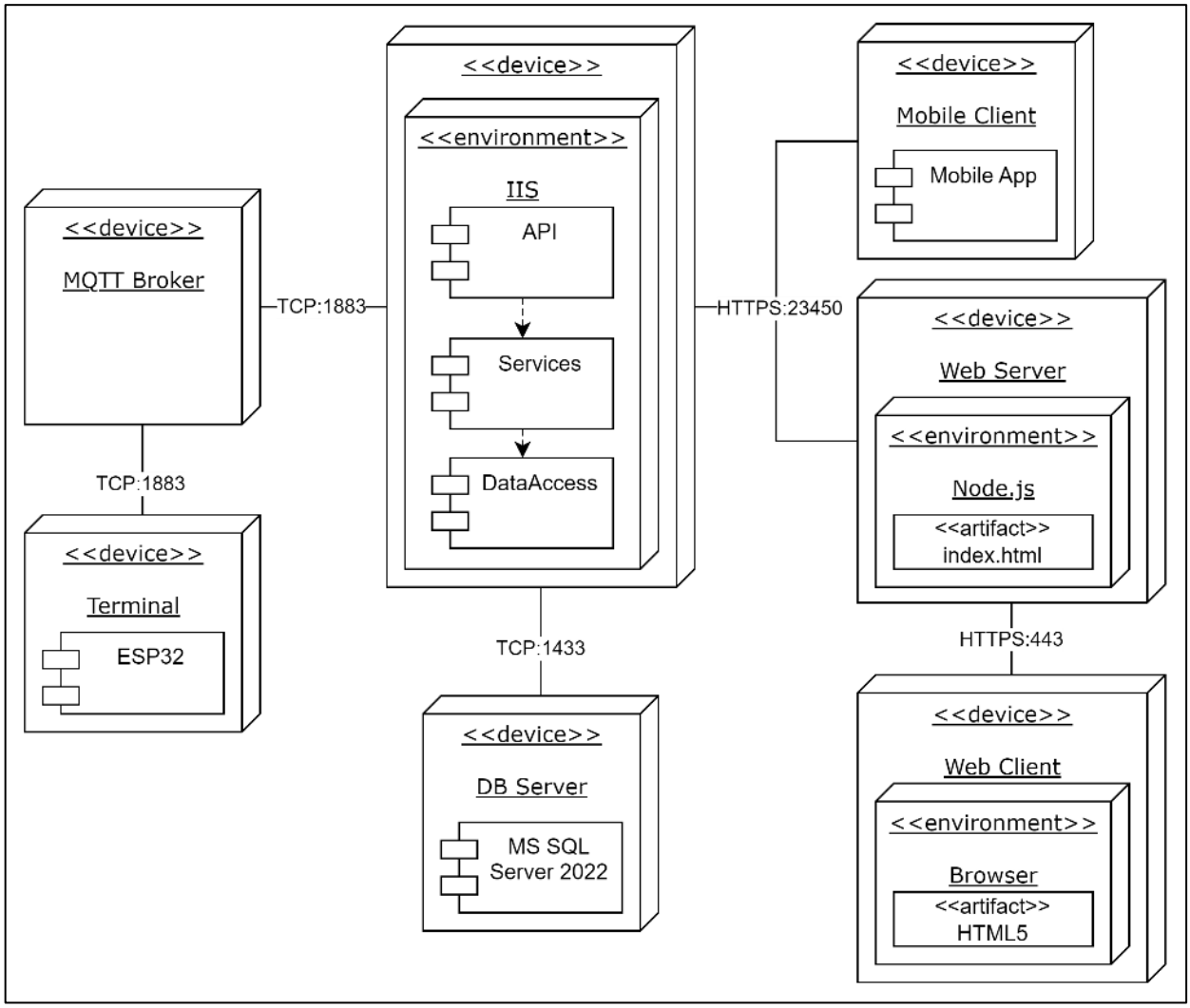

Рисунок 1 – Діаграма розгортання програмної системи

На рисунку 2 зображено діаграму прецедентів для загальної архітектури, на якій зображено відношення між акторами та прецедентами в системі.

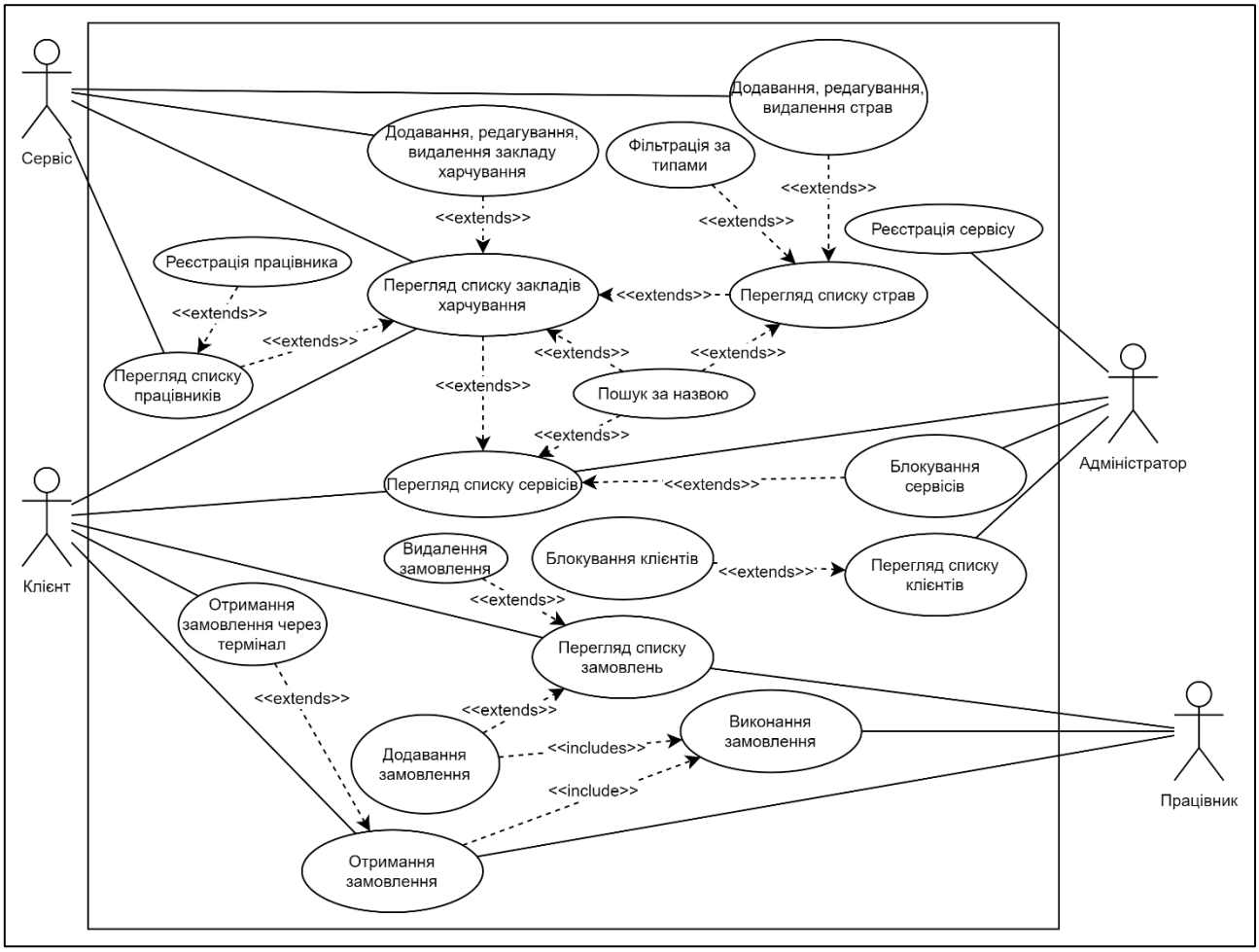

Рисунок 2 – Діаграма прецендентів серверної частини

REST API в даному проекті складається із 34 ендпоінтів, кожен з яких допомагає відображати інформацію на сторінках браузера/мобільного додатку або виконувати безпосередньо бізнес логіку системи. Специфікація кожного контроллеру наведена у додатку А.

На рисунку 3 зображено ER-модель даних.

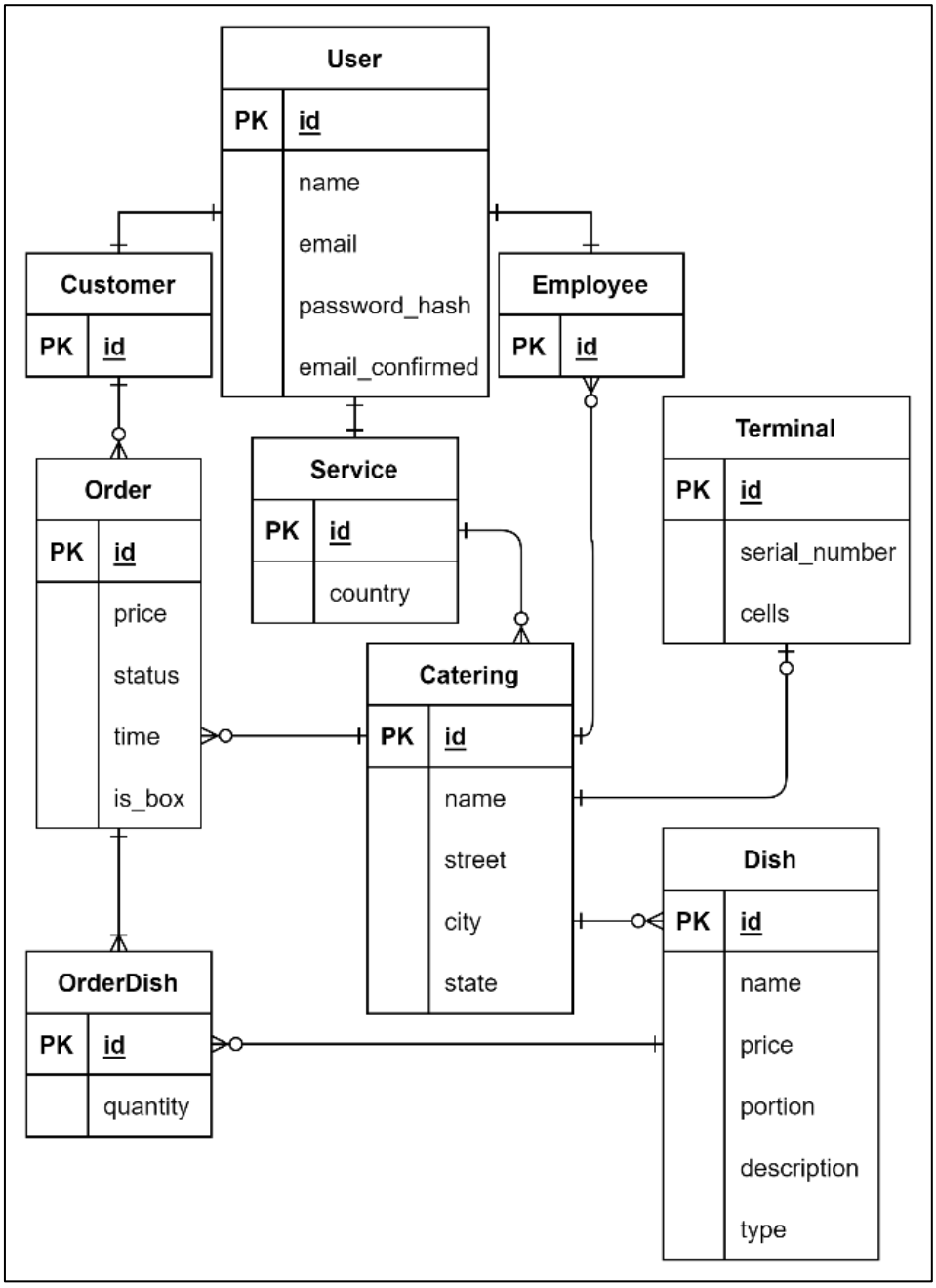

Рисунок 3 – ER-діаграма бази даних

Для роботи була обрана база даних MS SQL Server. Сутність «User» зв’язана з сутностями «Customer», «Service» та «Employee» зв’язком один до одного. Сутність «Customer» пов’язана із сутністю «Order» зв’язком один до багатьох. Сутність «Service» пов’язана із сутністю «Catering» зв’язком один до багатьох. Сутність «Catering» зв’язана з сутностями «Order», «Employee» та

«Dish» зв’язком один до багатьох і з сутністю «Terminal» зв’язком один до одного. Сутність «Order» та сутність « Dish» пов’язані зв’язком багато до багатьох через проміжну таблицю «OrderDish». Робота з базою даних реалізована за домомогою Entity Framework Core. Приклад реалізації наведено у додатку Б.

На рисунку 4 зображено діаграму компонентів.

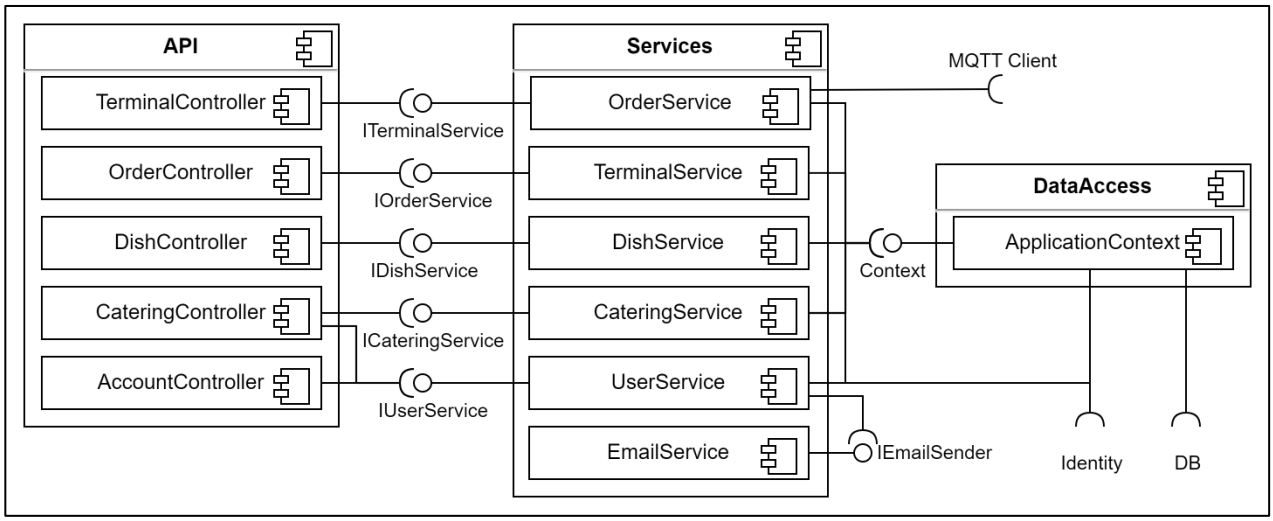

Рисунок 4 – Діаграма компонентів серверної частини

Компонент «API» являє собою шар представлення, та має у собі набір контроллерів, що оброблюють запити до серверу. Там же і налаштовується рівень доступу до ендпоінтів. Приклад контроллеру наведено у додатку В.

Компонент «Services» являє собою шар бізнес-логіки. Приклад класу бізнес-логіки наведено у додатку Г.

Компонент «» являє собою шар доступу до данних. Його код наведено у додатку Б.

1.3 Висновки

У данній лабораторній роботі було розроблено серверну частину програмної системи для проєкту за темою «Програмна система для автоматизації видачі боксів із їжею».

Посилання на папку "DEMO" на Google Drive: https://drive.google.com/drive/folders/1sBPljA5AFl66BZejiDRfry6l0LPELsV7?usp=share_link

ДОДАТОК А

REST-специфікація

Таблиця А.1 – Маршрут /api/Account/

Дія	Метод	Параметри	Тіло	Відповіді
Login	POST		email, password	200, 400
Register	POST	callbackUrl	name, emai, password	201, 400
Register /Service	POST	callbackUrl	name, email, password, country	201, 400
Register /Employee	POST	callbackUrl	name, email, password, cateringId	201, 400
Logout	DELETE			200, 401
Info	GET			200, 401
Role	GET			200, 401
ConfirmEmail	GET	id, code		200, 400, 404
Delete/{id?}	DELETE			200, 400-404
ChangeName /{id?}	PATCH		name	200, 400-404
ForgotPassword	POST	callbackUrl	email	200, 404
ResetPassword	PATCH		email, password, code confirmPassword	200, 400, 404
ChangePassword	PATCH		newPassword oldPassword	200, 400-403
GoogleAuth	GET	returnUrl		302
GetEmployees	GET	cateringId, page, query		200, 401, 403
GetServices	GET	country, page, query		200, 401, 403
GetCustomers	GET	page, query		200, 401, 403
Block, Unblock	PATCH	id		200, 401-404

Таблиця А.2 – Маршрут /api/Terminal/

Метод	Параметри	Тіло	Відповіді
POST		serialNumber, id, cellCount	201, 401-404
PUT	id	serialNumber, id, cellCount	200, 401-404
DELETE	id		200, 401-404

Таблиця А.3 – Маршрут /api/Catering/

Дія	Метод	Параметри	Тіло	Відповіді
	GET	page, query		200, 401, 403
	POST		name, street, city, state	201, 401-404
	PUT	id	name, street, city, state	200, 401-404
	DELETE	id		200, 401-404
{serviceId}	GET	page, query		200, 401, 403

Таблиця А.4 – Маршрут /api/Dish/

Дія	Метод	Параметри	Тіло	Відповіді
	GET	page, query, cateringId, tags		200, 401, 403
	POST		price, portion, name, description, type, cateringId	201, 401-404
	PUT	id	price, portion, name, description, type, cateringId	200, 401-404
	DELETE	id		200, 401-404
Types	GET			200, 401, 403

Таблиця А.5 – Маршрут /api/Order/

Дія	Метод	Параметри	Тіло	Відповіді
	GET	page, startTime, endTime, isBox		200, 401, 403
	POST		time, isBox, cateringId, orderDishes	201, 401-404
	DELETE	id		200, 401-404
Optimal	GET	cateringId, maxPrice, types		200, 401-404
Do	PATCH	id		200, 401-404
Receive	PATCH	id		200, 401-404
Services	GET	page, query, country		200, 401, 403

ДОДАТОК Б

Програмний код контексту доступу до БД

```
public class ApplicationContext : IdentityDbContext<User>
{
  public ApplicationContext(DbContextOptions<ApplicationContext> options)
    : base(options) {}
  public DbSet<Customer> Customers { get; set; } = null!;
  public DbSet<Employee> Employees { get; set; } = null!;
  public DbSet<Service> Services { get; set; } = null!;
  public DbSet<Catering> Caterings { get; set; } = null!;
  public DbSet<Dish> Dishes { get; set; } = null!;
  public DbSet<Order> Orders { get; set; } = null!;
  public DbSet<OrderDish> OrderDishes { get; set; } = null!;
  public DbSet<Terminal> Terminals { get; set; } = null!;
  protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder) => optionsBuilder.UseLazyLoadingProxies();
  protected override void OnModelCreating(ModelBuilder builder)
  {
    builder.Entity<User>()
         .UseTptMappingStrategy();
    builder.Entity<Order>()
         .HasOne(o => o.Customer)
         .WithMany(c => c.Orders)
         .HasForeignKey(o => o.CustomerId)
         .OnDelete(DeleteBehavior.NoAction);
    builder.Entity<Service>()
         .HasMany(s => s.Caterings)
         .WithOne(o => o.Service)
         .HasForeignKey(o => o.ServiceId)
         .OnDelete(DeleteBehavior.Cascade);
    builder.Entity<Catering>()
         .HasMany(c => c.Employees)
         .WithOne(e => e.Catering)
         .HasForeignKey(e => e.CateringId)
         .OnDelete(DeleteBehavior.Cascade);
    builder.Entity<Employee>()
         .HasOne(e => e.Catering)
         .WithMany(c => c.Employees)
         .HasForeignKey(e => e.CateringId)
         .OnDelete(DeleteBehavior.NoAction);
    builder.Entity<Catering>()
         .HasMany(c => c.Dishes)
         .WithOne(o => o.Catering)
         .HasForeignKey(o => o.CateringId)
         .OnDelete(DeleteBehavior.Cascade);
    builder.Entity<Catering>()
         .HasMany(c => c.Orders)
         .WithOne(o => o.Catering)
```

```
47            .HasForeignKey(o => o.CateringId)
48            .OnDelete(DeleteBehavior.Cascade);
49        builder.Entity<Catering>()
50            .HasOne(c => c.Terminal)
51            .WithOne(t => t.Catering)
52            .HasForeignKey<Terminal>(t => t.Id)
53            .OnDelete(DeleteBehavior.Cascade);
54        builder.Entity<Terminal>()
55            .HasKey(t => t.Id);
56        builder.Entity<Terminal>()
57            .Property(t => t.Id)
58            .ValueGeneratedNever();
59        builder.Entity<Order>()
60            .HasMany(o => o.OrderDishes)
61            .WithOne(od => od.Order)
62            .HasForeignKey(od => od.OrderId)
63            .OnDelete(DeleteBehavior.Cascade);
64        builder.Entity<OrderDish>()
65            .HasKey(od => new { od.OrderId, od.DishId });
66        builder.Entity<OrderDish>()
67            .HasOne(od => od.Dish)
68            .WithMany(o => o.OrderDishes)
69            .HasForeignKey(od => od.DishId)
70            .OnDelete(DeleteBehavior.NoAction);
71        builder.Entity<Terminal>()
72            .Property(t => t.Cells)
73            .HasConversion(c => string.Join('|', c),
74                  c => c.Split('|', StringSplitOptions.None));
75        base.OnModelCreating(builder);
76    }
77  }
```

ДОДАТОК В

Програмний код контроллеру замовлень

```
[ApiController]
[Route(Routes.CrudRoute)]
[Authorize(Roles = Roles.CustomerEmployee)]
[ProducesResponseType(StatusCodes.Status401Unauthorized)]
public class OrderController : BaseController
{
  private readonly IOrderService orderService;
  public OrderController(IOrderService orderService) =>
    this.orderService = orderService;
  [HttpGet]
  [ProducesResponseType(StatusCodes.Status200OK)]
  public async Task<ActionResult<PagedArrayModel<OrderModel>>>
GetAsync(DateTime? startTime, DateTime? endTime, int page = 1, bool?
isBox = null) =>
    await orderService.GetAsync(User, page, isBox, startTime ??
DateTime.MinValue, endTime ?? DateTime.MaxValue);
  [HttpGet(Routes.Action)]
  [ProducesResponseType(StatusCodes.Status200OK)]
  [ProducesResponseType(StatusCodes.Status404NotFound)]
  public ActionResult<List<OrderDishModel>> Optimal(int cateringId,
                            int maxPrice,
                            [FromQuery] Dictionary<string, int> types)
  {
    types.Remove("maxPrice");
    types.Remove("cateringId");
    var result = orderService.GetOptimal(cateringId, maxPrice, types);
    return HandleResult(result);
  }
  [HttpPost]
  [ProducesResponseType(StatusCodes.Status201Created)]
  [ProducesResponseType(StatusCodes.Status400BadRequest)]
  [ProducesResponseType(StatusCodes.Status403Forbidden)]
  [ProducesResponseType(StatusCodes.Status404NotFound)]
  public async virtual Task<ActionResult<OrderModel>> AddAsync([FromBody]
OrderRequest request)
  {
    var model = request.Adapt<OrderModel>();
    var result = await orderService.AddAsync(User, model);
    return HandleCreatedResult(result);
  }
  [HttpPatch(Routes.Action)]
  [ProducesResponseType(StatusCodes.Status200OK)]
  [ProducesResponseType(StatusCodes.Status400BadRequest)]
  [ProducesResponseType(StatusCodes.Status403Forbidden)]
  [ProducesResponseType(StatusCodes.Status404NotFound)]
  public async virtual Task<IActionResult> DoAsync([Required] int id)
```

```
47      {
48        var result = await orderService.DoAsync(User, id);
49        return HandleResult(result);
50      }
51      [HttpPatch(Routes.Action)]
52      [ProducesResponseType(StatusCodes.Status200OK)]
53      [ProducesResponseType(StatusCodes.Status400BadRequest)]
54      [ProducesResponseType(StatusCodes.Status403Forbidden)]
55      [ProducesResponseType(StatusCodes.Status404NotFound)]
56      public async virtual Task<IActionResult> ReceiveAsync(int id)
57      {
58        var result = await orderService.ReceiveAsync(User, id);
59        return HandleResult(result);
60      }
61      [HttpDelete]
62      [ProducesResponseType(StatusCodes.Status200OK)]
63      [ProducesResponseType(StatusCodes.Status400BadRequest)]
64      [ProducesResponseType(StatusCodes.Status403Forbidden)]
65      [ProducesResponseType(StatusCodes.Status404NotFound)]
66      public async virtual Task<IActionResult> DeleteAsync([Required] int id)
67      {
68        var result = await orderService.DeleteAsync(User, id);
69        return HandleResult(result);
70      }
71      [HttpGet(Routes.Action)]
72      [ProducesResponseType(StatusCodes.Status200OK)]
73      public async Task<ActionResult<PagedArrayModel<ServiceModel>>>
74  ServicesAsync(int page = 1, string query = "", string? country = null) =>
75        await orderService.GetServicesAsync(page, query, country);
76  }
```

ДОДАТОК Г

Програмний код класу бізнес-логіки замовлень

```
1   public class OrderService : IOrderService
2   {
3     private readonly ApplicationContext context;
4     private readonly UserManager<User> userManager;
5     private readonly IConfiguration configuration;
6     private readonly IMqttClient client;
7     public OrderService(ApplicationContext context, UserManager<User>
8   userManager, IConfiguration configuration, IMqttClient client)
9     {
10      this.context = context;
11      this.userManager = userManager;
12      this.configuration = configuration;
13      this.client = client;
14    }
15
16    public async Task<PagedArrayModel<OrderModel>> GetAsync(ClaimsPrincipal
17  principal, int page, bool? isBox, DateTime startTime, DateTime endTime)
18    {
19      var user = await userManager.GetUserAsync(principal);
20      Expression<Func<Order, bool>> predicate = user switch
21      {
22        Employee employee => x => x.CateringId == employee.CateringId,
23        null => throw new NullReferenceException(),
24        _ => x => x.CustomerId == user.Id,
25      };
26      var query = context.Set<Order>()
27                 .Where(o => o.Time <= endTime
28                       && o.Time >= startTime)
29                 .Where(predicate);
30      if (isBox is not null)
31        query = query.Where(x => x.IsBox == isBox);
32      var entities = await query.OrderByDescending(x => x.Time)
33                    .Skip((page - 1) * Utils.ItemsPerPage)
34                    .Take(Utils.ItemsPerPage)
35                    .ToListAsync();
36      var models = entities.Adapt<List<OrderModel>>();
37      return new PagedArrayModel<OrderModel>(models, query.Count());
38    }
39
40    public async Task<Result<OrderModel>> AddAsync(ClaimsPrincipal
41  principal, OrderModel model)
42    {
43      using var transaction = context.Database.BeginTransaction();
44      model.Status = OrderStatuses.Undone;
45      var b = model.OrderDishes.All(od => context.Set<Dish>().Any(d => d.Id
46  == od.DishId));
```

```
47      if (!b)
48        return Result.Fail(Errors.InvalidDishes);
49      var entity = model.Adapt<Order>();
50      var user = await userManager.GetUserAsync(principal);
51      switch (user)
52      {
53        case Customer customer:
54          entity.CustomerId = user.Id;
55          break;
56        case Employee employee:
57          entity.CateringId = employee.CateringId;
58          break;
59        default:
60          throw new ArgumentException(string.Empty, nameof(principal));
61
62      }
63      var proxy = context.Set<Order>().CreateProxy();
64      context.Entry(proxy).CurrentValues.SetValues(entity);
65      await context.AddAsync(proxy);
66      await context.SaveChangesAsync();
67      var orderDishes = model.OrderDishes.Adapt<List<OrderDish>>();
68      orderDishes.ForEach(od =>
69      {
70        od.OrderId = proxy.Id;
71        var dish = context.Set<Dish>().Find(od.DishId);
72        od.Dish = dish!;
73      });
74      await context.AddRangeAsync(orderDishes);
75      if (proxy.IsBox)
76      {
77        var terminal = proxy.Catering.Terminal;
78        var index = Array.IndexOf(terminal.Cells, string.Empty);
79        if (index == -1)
80          return Result.Fail("No cells available");
81        terminal.Cells[index] = proxy.Id.ToString();
82        context.Update(terminal);
83      }
84      proxy.Price = orderDishes.Sum(od => od.Quantity * od.Dish.Price);
85      context.Update(proxy);
86      await context.SaveChangesAsync();
87      transaction.Commit();
88      var response = proxy.Adapt<OrderModel>();
89      return Result.Ok(response);
90    }
91
92    public async Task<Result> DeleteAsync(ClaimsPrincipal principal, int
93  id)
94    {
95      var order = await context.FindAsync<Order>(id);
96      if (order is null)
97        return Result.Fail(Errors.NotFound);
98      var user = await userManager.GetUserAsync(principal);
99      if (order.GetOwnerId() != user!.Id)
100       return Result.Fail(Errors.Forbidden);
```

```
    if (user is Customer)
    {
      var timeoutMins =
double.Parse(configuration["Order:DeleteTimeout"]!);
      var timeout = TimeSpan.FromMinutes(timeoutMins);
      if (order.Time - timeout < DateTime.UtcNow)
        return Result.Fail(Errors.Forbidden);
    }
    if (order.IsBox)
      RemoveFromCell(order);
    context.Remove(order);
    await context.SaveChangesAsync();
    return Result.Ok();
  }

  public Result<List<OrderDishModel>> GetOptimal(int cateringId, decimal
maxPrice, Dictionary<string, int> types)
  {
    try
    {
      static decimal GoalFunc(TagDish td) => td.Dishes.First().Price *
td.Type.Value;
      List<TagDish> tagDishes = new();
      List<TagDish> invariant = new();
      foreach (var type in types)
      {
        var tagDish = new TagDish(type)
        {
          Dishes = context.Dishes.Where(d => d.CateringId == cateringId)
                       .Where(d => d.Type == type.Key)
                       .OrderByDescending(d => d.Price)
        };
        tagDishes.Add(tagDish);
      }
      while (tagDishes.Sum(GoalFunc) > maxPrice)
      {
        tagDishes = tagDishes.OrderByDescending(GoalFunc).ToList();
        var first = tagDishes.First();
        if (first.Dishes.Count() == 1)
        {
          invariant.Add(first);
          maxPrice -= GoalFunc(first);
          tagDishes.RemoveAt(0);
        }
        else
          first.Dishes = first.Dishes.Skip(1).OrderByDescending(d =>
d.Price);
      }
      invariant.AddRange(tagDishes);
      if (maxPrice < 0)
        return Result.Fail(Errors.NotFound);
      return invariant.Select(i =>
      {
        Dish dish = i.Dishes.First();
```

```
          return new OrderDishModel { OrderId = 0, DishId = dish.Id, Dish =
dish.Adapt<DishModel>(), Quantity = i.Type.Value };
        }).ToList();
      }
      catch (InvalidOperationException)
      {
        return Result.Fail(Errors.NotFound);
      }
    }
    internal record TagDish(KeyValuePair<string, int> Type)
    {
      public IOrderedQueryable<Dish> Dishes { get; set; } = null!;
    }

    public async Task<PagedArrayModel<ServiceModel>> GetServicesAsync(int
page, string query, string? country)
    {
      var enumerable = context.Set<Service>().Where(x =>
x.Name.Contains(query));
      if (country is not null)
        enumerable = enumerable.Where(x => x.Country == country);
      var entities = await enumerable.OrderByDescending(x => x.Name)
                      .Skip((page - 1) * Utils.ItemsPerPage)
                      .Take(Utils.ItemsPerPage)
                      .ToListAsync();
      var models = entities.Adapt<List<ServiceModel>>();
      return new PagedArrayModel<ServiceModel>(models, enumerable.Count());
    }

    public async Task<Result> DoAsync(ClaimsPrincipal principal, int id) =>
      await SetStatus(principal, id, OrderStatuses.Undone,
OrderStatuses.Done);

    public async Task<Result> ReceiveAsync(ClaimsPrincipal principal, int
id) =>
      await SetStatus(principal, id, OrderStatuses.Done,
OrderStatuses.Received);

    private async Task<Result> SetStatus(ClaimsPrincipal principal, int id,
string oldStatus, string newStatus)
    {
      var order = await context.FindAsync<Order>(id);
      if (order is null)
        return Result.Fail(Errors.NotFound);
      var userId = userManager.GetUserId(principal);
      if (order.GetOwnerId() != userId || order.Status != oldStatus)
        return Result.Fail(Errors.Forbidden);
      if (order.IsBox && newStatus == OrderStatuses.Received)
      {
        var terminal = order.Catering.Terminal;
        var cellId = Array.IndexOf(terminal.Cells, order.Id.ToString());
        var prefix = configuration["Mqtt:Prefix"];
        var message = new MqttApplicationMessageBuilder()
        .WithTopic($"{prefix}/{terminal.SerialNumber}")
```

```
209         .WithPayload(cellId.ToString())
210         .Build();
211         await client.PublishAsync(message);
212         RemoveFromCell(order);
213       }
214       order.Status = newStatus;
215       context.Update(order);
216       await context.SaveChangesAsync();
217       return Result.Ok();
218     }
219
220     private void RemoveFromCell(Order order)
221     {
222       var terminal = order.Catering.Terminal;
223       var index = Array.IndexOf(terminal.Cells, order.Id.ToString());
224       terminal.Cells[index] = string.Empty;
225       context.Update(terminal);
226     }
227   }
```