Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра Програмної Інженерії

ЗВІТ

з дисципліни «Архітектура програмного забезпечення»

з лабораторної роботи №2

|  |  |
| --- | --- |
| Виконав | Перевірив: |
| ст. гр. ПЗПІ-20-7 | Старш. викл. кафедри ПІ |
| Крупчак Євгеній | 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 С# 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.

A screenshot of a computer program

Description automatically generated with low confidence

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.

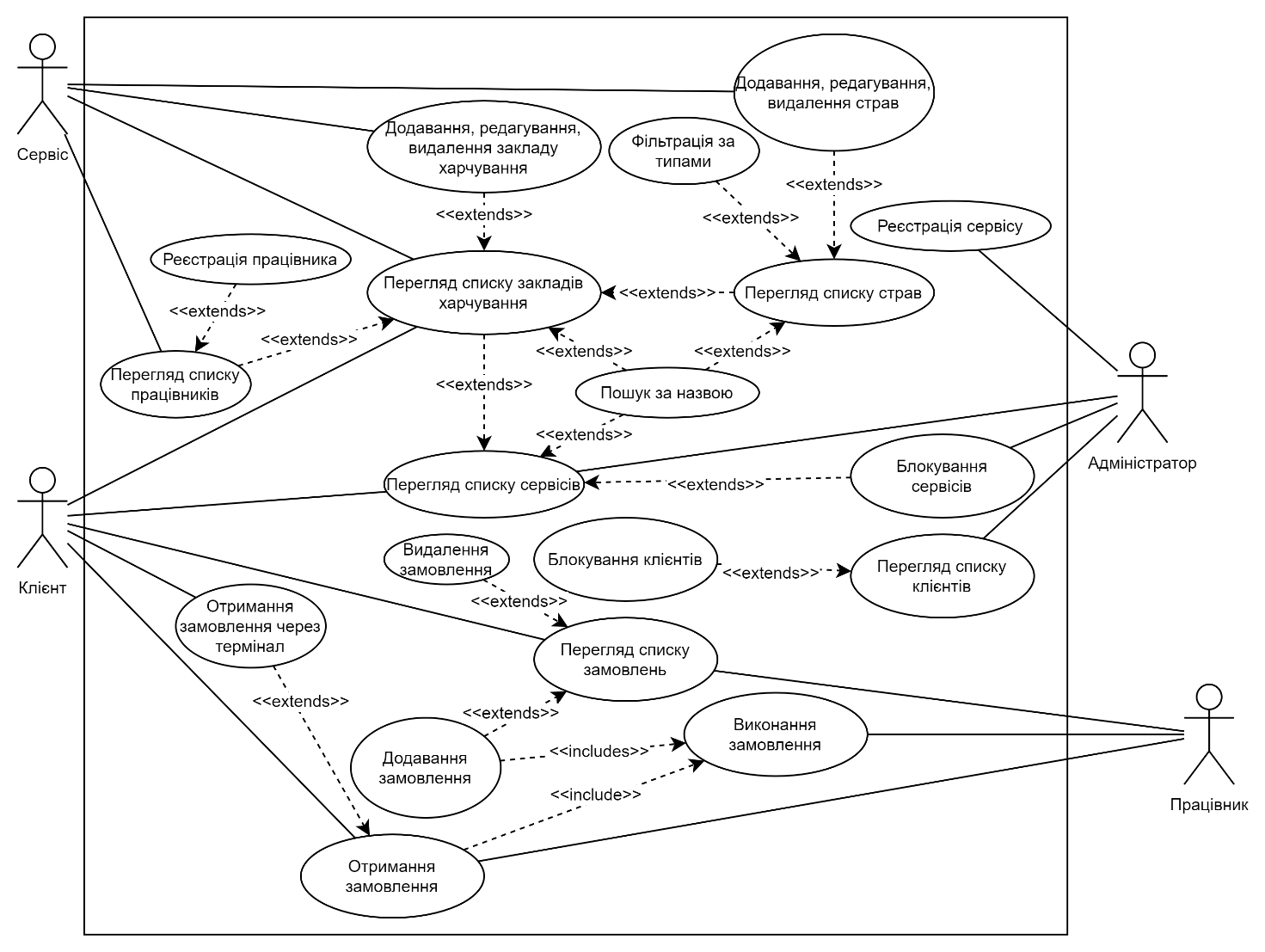


Рисунок 2 – Діаграма прецендентів серверної частини

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.

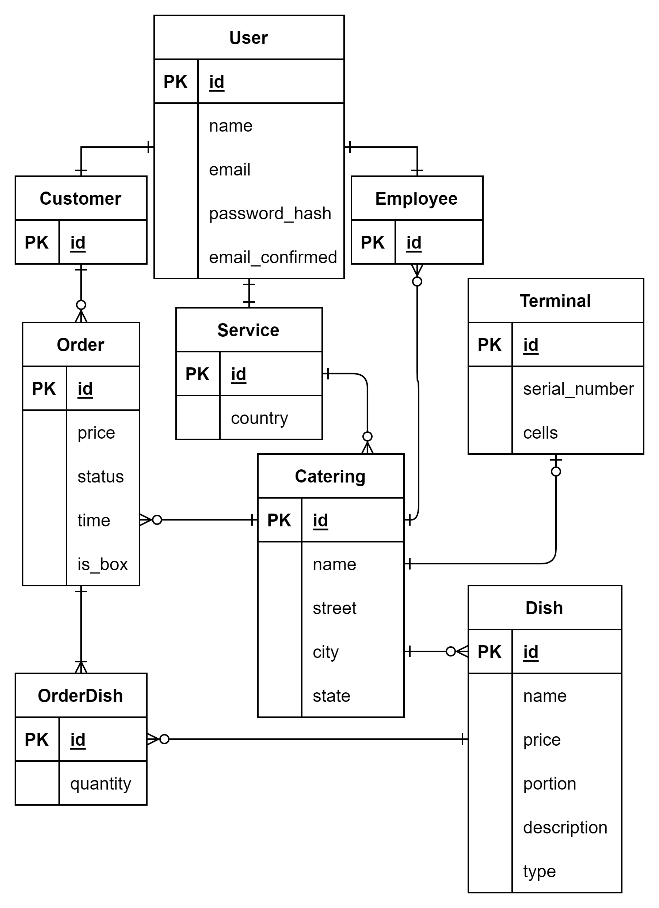


Рисунок 3 – ER-діаграма бази даних

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.

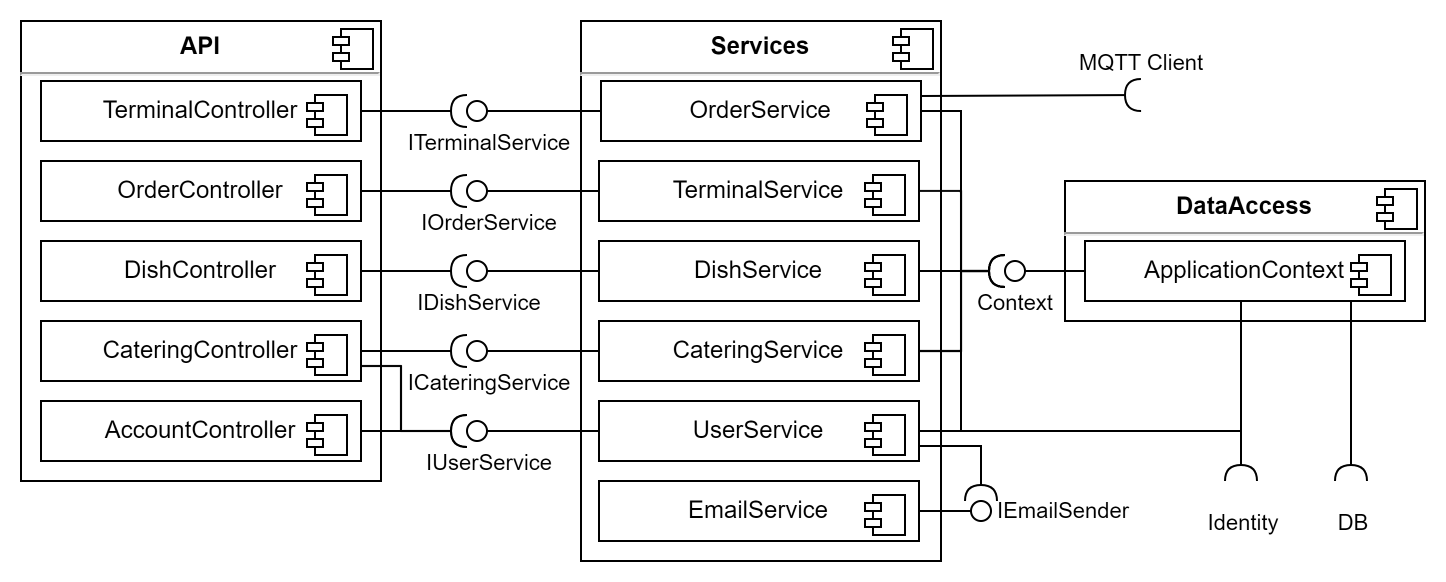


Рисунок 4 – Діаграма компонентів серверної частини

Компонент «АРІ» 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 A**

REST-специфікація

Таблиця A.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 A.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 A.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 A.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 A.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)

.HasForeignKey(o => o.CateringId)

.OnDelete(DeleteBehavior.Cascade);

builder.Entity<Catering>()

.HasOne(c => c.Terminal)

.WithOne(t => t.Catering)

.HasForeignKey<Terminal>(t => t.Id)

.OnDelete(DeleteBehavior.Cascade);

builder.Entity<Terminal>()

.HasKey(t => t.Id);

builder.Entity<Terminal>()

.Property(t => t.Id)

.ValueGeneratedNever();

builder.Entity<Order>()

.HasMany(o => o.OrderDishes)

.WithOne(od => od.Order)

.HasForeignKey(od => od.OrderId)

.OnDelete(DeleteBehavior.Cascade);

builder.Entity<OrderDish>()

.HasKey(od => new { od.OrderId, od.DishId });

builder.Entity<OrderDish>()

.HasOne(od => od.Dish)

.WithMany(o => o.OrderDishes)

.HasForeignKey(od => od.DishId)

.OnDelete(DeleteBehavior.NoAction);

builder.Entity<Terminal>()

.Property(t => t.Cells)

.HasConversion(c => string.Join('|', c),

c => c.Split('|', StringSplitOptions.None));

base.OnModelCreating(builder);

}

}

**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)

{

var result = await orderService.DoAsync(User, id);

return HandleResult(result);

}

[HttpPatch(Routes.Action)]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status403Forbidden)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public async virtual Task<IActionResult> ReceiveAsync(int id)

{

var result = await orderService.ReceiveAsync(User, id);

return HandleResult(result);

}

[HttpDelete]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

[ProducesResponseType(StatusCodes.Status403Forbidden)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public async virtual Task<IActionResult> DeleteAsync([Required] int id)

{

var result = await orderService.DeleteAsync(User, id);

return HandleResult(result);

}

[HttpGet(Routes.Action)]

[ProducesResponseType(StatusCodes.Status200OK)]

public async Task<ActionResult<PagedArrayModel<ServiceModel>>> ServicesAsync(int page = 1, string query = "", string? country = null) =>

await orderService.GetServicesAsync(page, query, country);

}

**ДОДАТОК Г**

Програмний код класу бізнес-логіки замовлень

public class OrderService : IOrderService

{

private readonly ApplicationContext context;

private readonly UserManager<User> userManager;

private readonly IConfiguration configuration;

private readonly IMqttClient client;

public OrderService(ApplicationContext context, UserManager<User> userManager, IConfiguration configuration, IMqttClient client)

{

this.context = context;

this.userManager = userManager;

this.configuration = configuration;

this.client = client;

}

public async Task<PagedArrayModel<OrderModel>> GetAsync(ClaimsPrincipal principal, int page, bool? isBox, DateTime startTime, DateTime endTime)

{

var user = await userManager.GetUserAsync(principal);

Expression<Func<Order, bool>> predicate = user switch

{

Employee employee => x => x.CateringId == employee.CateringId,

null => throw new NullReferenceException(),

\_ => x => x.CustomerId == user.Id,

};

var query = context.Set<Order>()

.Where(o => o.Time <= endTime

&& o.Time >= startTime)

.Where(predicate);

if (isBox is not null)

query = query.Where(x => x.IsBox == isBox);

var entities = await query.OrderByDescending(x => x.Time)

.Skip((page - 1) \* Utils.ItemsPerPage)

.Take(Utils.ItemsPerPage)

.ToListAsync();

var models = entities.Adapt<List<OrderModel>>();

return new PagedArrayModel<OrderModel>(models, query.Count());

}

public async Task<Result<OrderModel>> AddAsync(ClaimsPrincipal principal, OrderModel model)

{

using var transaction = context.Database.BeginTransaction();

model.Status = OrderStatuses.Undone;

var b = model.OrderDishes.All(od => context.Set<Dish>().Any(d => d.Id == od.DishId));

if (!b)

return Result.Fail(Errors.InvalidDishes);

var entity = model.Adapt<Order>();

var user = await userManager.GetUserAsync(principal);

switch (user)

{

case Customer customer:

entity.CustomerId = user.Id;

break;

case Employee employee:

entity.CateringId = employee.CateringId;

break;

default:

throw new ArgumentException(string.Empty, nameof(principal));

}

var proxy = context.Set<Order>().CreateProxy();

context.Entry(proxy).CurrentValues.SetValues(entity);

await context.AddAsync(proxy);

await context.SaveChangesAsync();

var orderDishes = model.OrderDishes.Adapt<List<OrderDish>>();

orderDishes.ForEach(od =>

{

od.OrderId = proxy.Id;

var dish = context.Set<Dish>().Find(od.DishId);

od.Dish = dish!;

});

await context.AddRangeAsync(orderDishes);

if (proxy.IsBox)

{

var terminal = proxy.Catering.Terminal;

var index = Array.IndexOf(terminal.Cells, string.Empty);

if (index == -1)

return Result.Fail("No cells available");

terminal.Cells[index] = proxy.Id.ToString();

context.Update(terminal);

}

proxy.Price = orderDishes.Sum(od => od.Quantity \* od.Dish.Price);

context.Update(proxy);

await context.SaveChangesAsync();

transaction.Commit();

var response = proxy.Adapt<OrderModel>();

return Result.Ok(response);

}

public async Task<Result> DeleteAsync(ClaimsPrincipal principal, int id)

{

var order = await context.FindAsync<Order>(id);

if (order is null)

return Result.Fail(Errors.NotFound);

var user = await userManager.GetUserAsync(principal);

if (order.GetOwnerId() != user!.Id)

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}")

.WithPayload(cellId.ToString())

.Build();

await client.PublishAsync(message);

RemoveFromCell(order);

}

order.Status = newStatus;

context.Update(order);

await context.SaveChangesAsync();

return Result.Ok();

}

private void RemoveFromCell(Order order)

{

var terminal = order.Catering.Terminal;

var index = Array.IndexOf(terminal.Cells, order.Id.ToString());

terminal.Cells[index] = string.Empty;

context.Update(terminal);

}

}