Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Online Food Ordering System

Semestrálny projekt z ASwS 2023

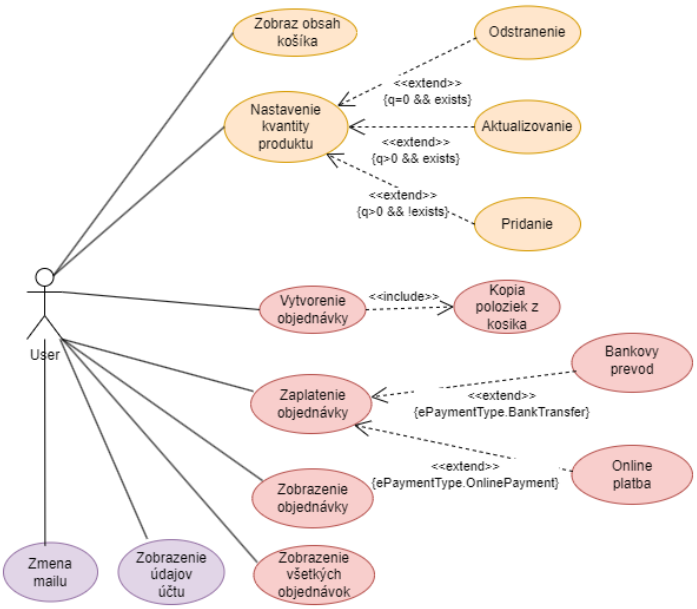
12.6.2023 Patrik Hampel

Úvod

Systém slúži na objednávanie produktov (jedla). Zákazníci majú košíky, do ktorých môžu pridávať a odstraňovať produkty. Z košíka je možné vytvoriť objednávku, zaplatiť za ňu a sledovať jej status.

Exceptions ktoré sú thrownuté v kóde sa s pomocou middleware pošlú ako error message v jsone.

Use case



Use Case: Nastavenie kvantity produktu
Actor: User
Priority: Must have
Pre-condition: prihlásený používateľ, existujúci produkt
Post-condition: aktualizovaná položka v košíku
Basic path:

1. Používateľ sa prihlási
2. Používateľ si vyberie produkt
3. Používateľ nastaví kvantitu produktu

Alternative path:
3a.1. Používateľ zvolí >0 pre produkt, ktorý nie je v košíku
3a.2. Produkt sa pridá do košíka
3b.1. Používateľ zvolí >0 pre produkt, ktoré je v košíku
3b.2. Kvantita produktu v košíku sa aktualizuje
3c.1. Používateľ zvolí 0 pre produkt, ktorý je v košíku
3c.2. Produkt sa odstráni z košíka

Use Case: Platba objednávky
Actor: User
Priority: Must have
Pre-condition: prihlásený používateľ, existujúca objednávka
Post-condition: zaplatená objednávka
Basic path:

1. Používateľ sa prihlási
2. Používateľ si vyberie objednávku

3. Používateľ si zvolí typ platby
4. Status objednávky je aktualizovaný na zaplatený a poslaná notifikácia

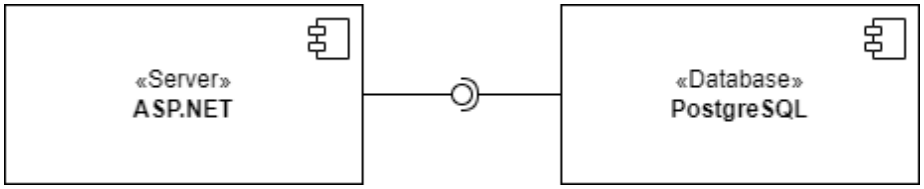
Alternative path:

3a. Platba bankovým prevodom

3b. Online platba

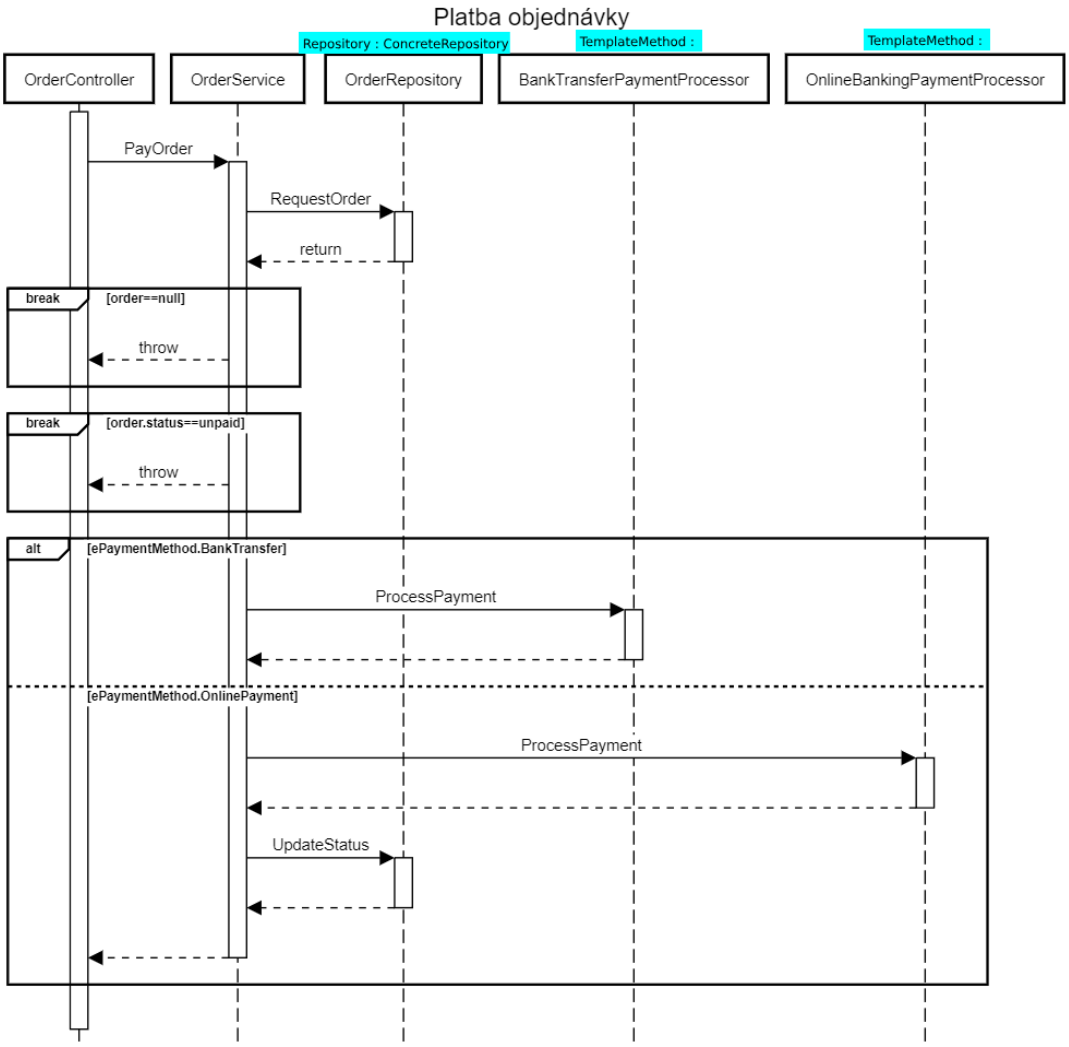
Architektúra systému

Na zariadení (v mojom prípade Windows, no .NET Core a Docker bežia na Linuxe aj MacOS) beží .NET Core backend server a Postgres DB je deploynutá cez Docker.

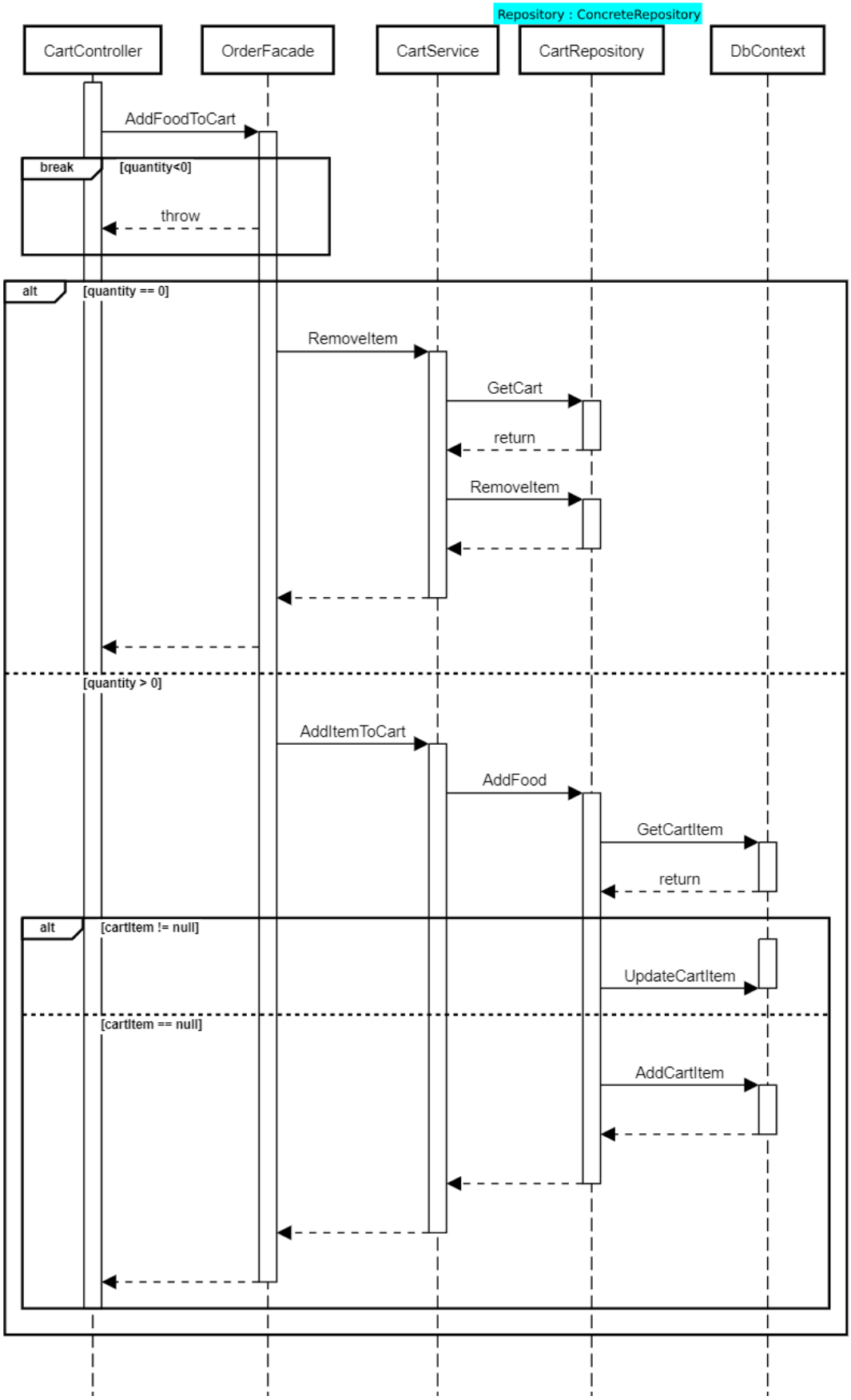


Analýza

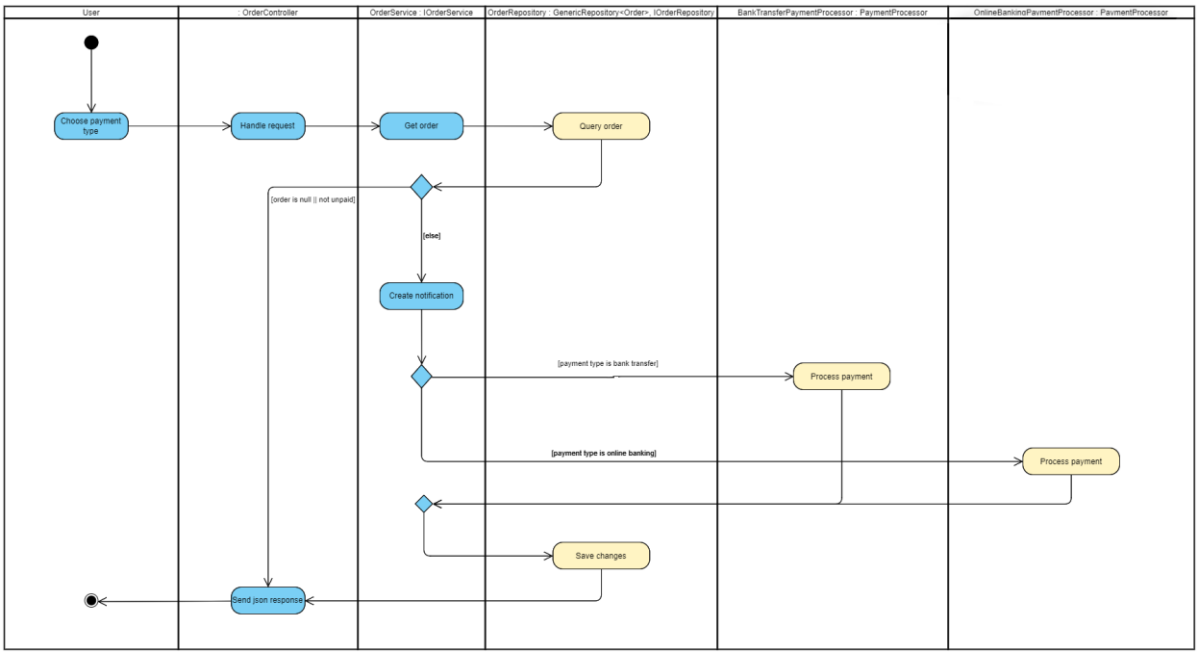
Sequence diagram



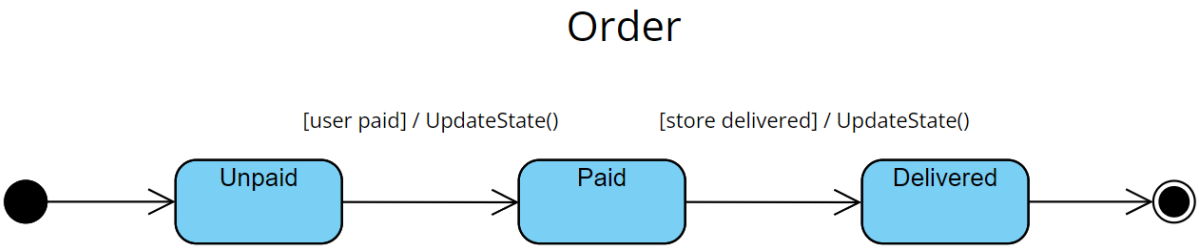
Aktualizácia košíka



Activity diagram

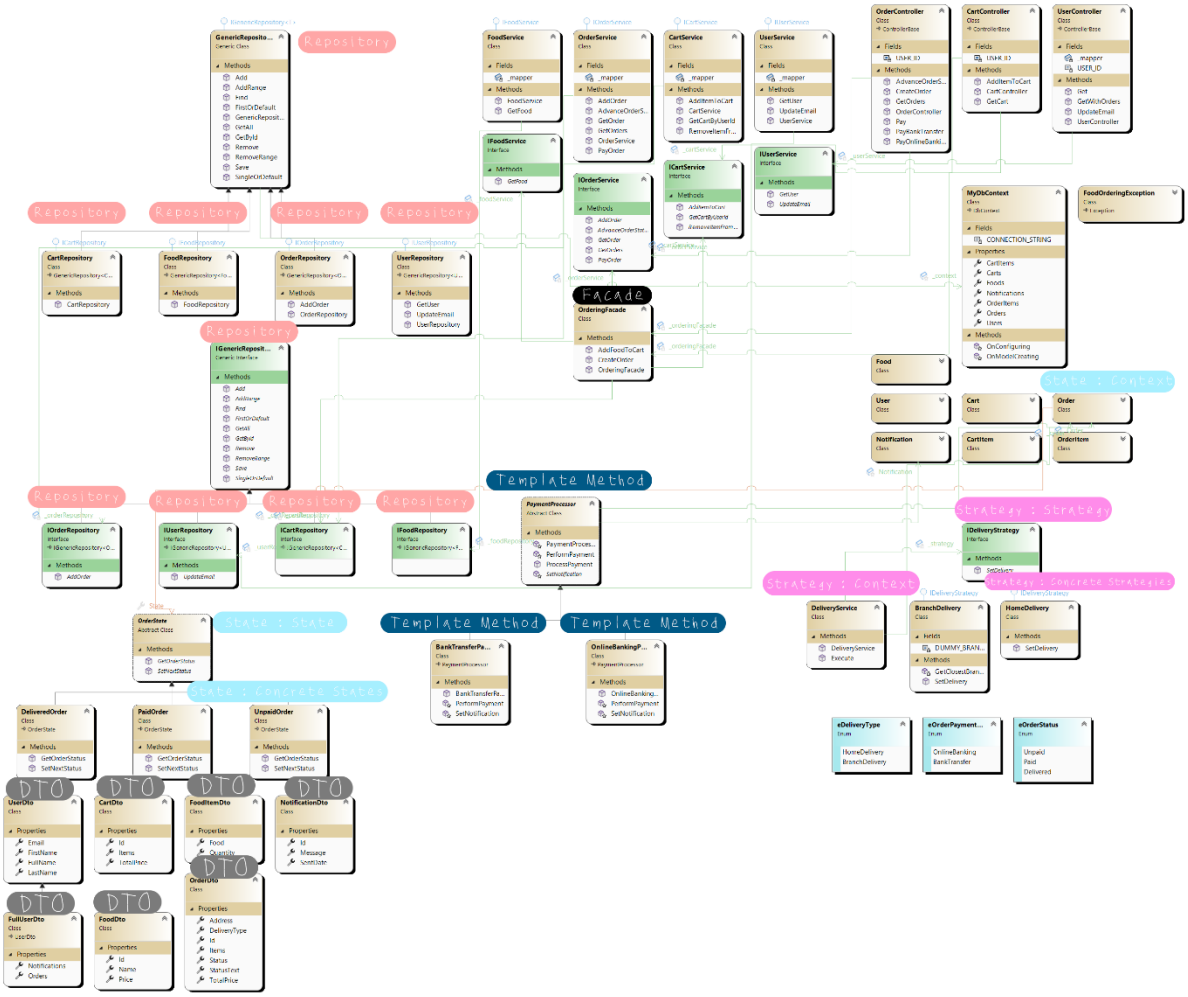


State diagram

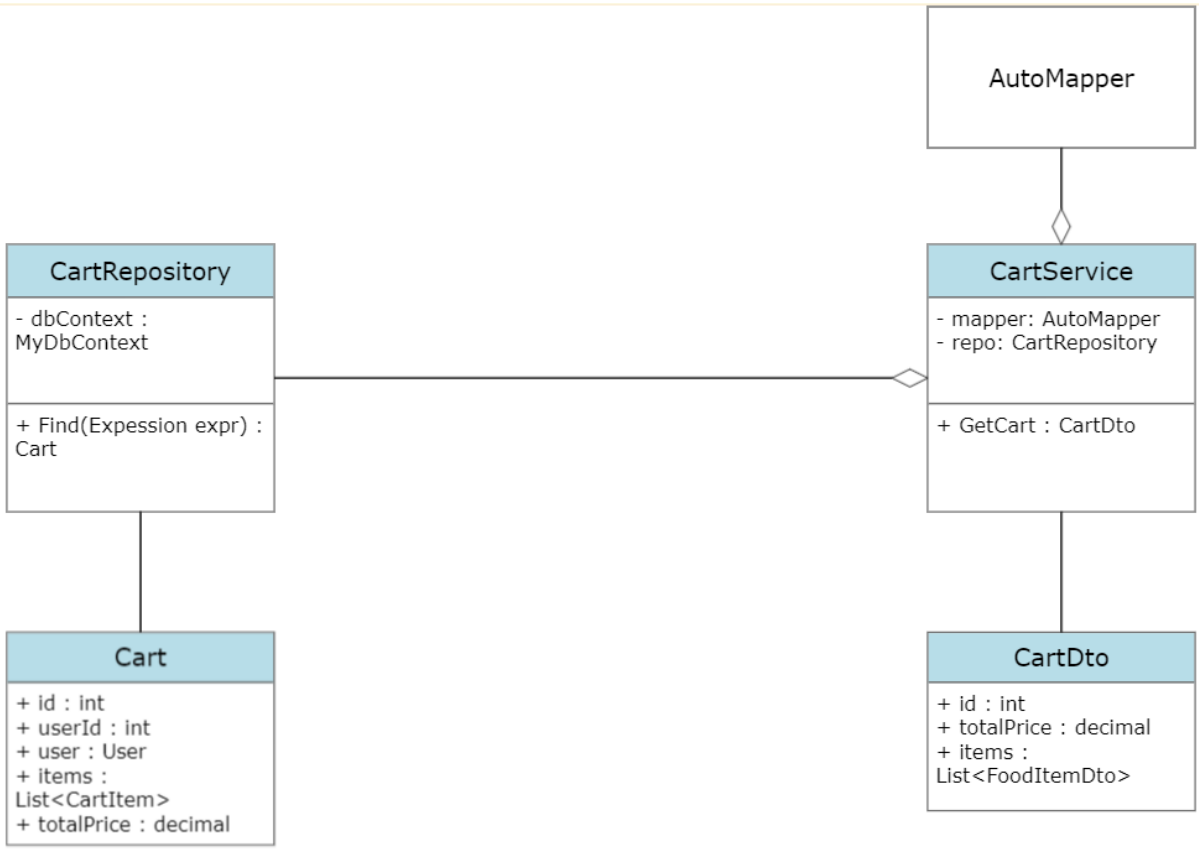


Návrh a implementácia, identifikácia vzorov

Class diagram



Class diagram pre vzor DTO



```
public class MappingProfile : Profile
{
    YelovSK
    public MappingProfile()
    {
        CreateMap<Order, OrderDto>();
        CreateMap<User, UserDto>();
        CreateMap<User, UserInfoDto>();
        CreateMap<UserDto, UserInfoDto>();
        CreateMap<Cart, CartDto>();
        CreateMap<CartItem, FoodItemDto>();
        CreateMap<OrderItem, FoodItemDto>();
        CreateMap<Food, FoodDto>();
    }
}
```

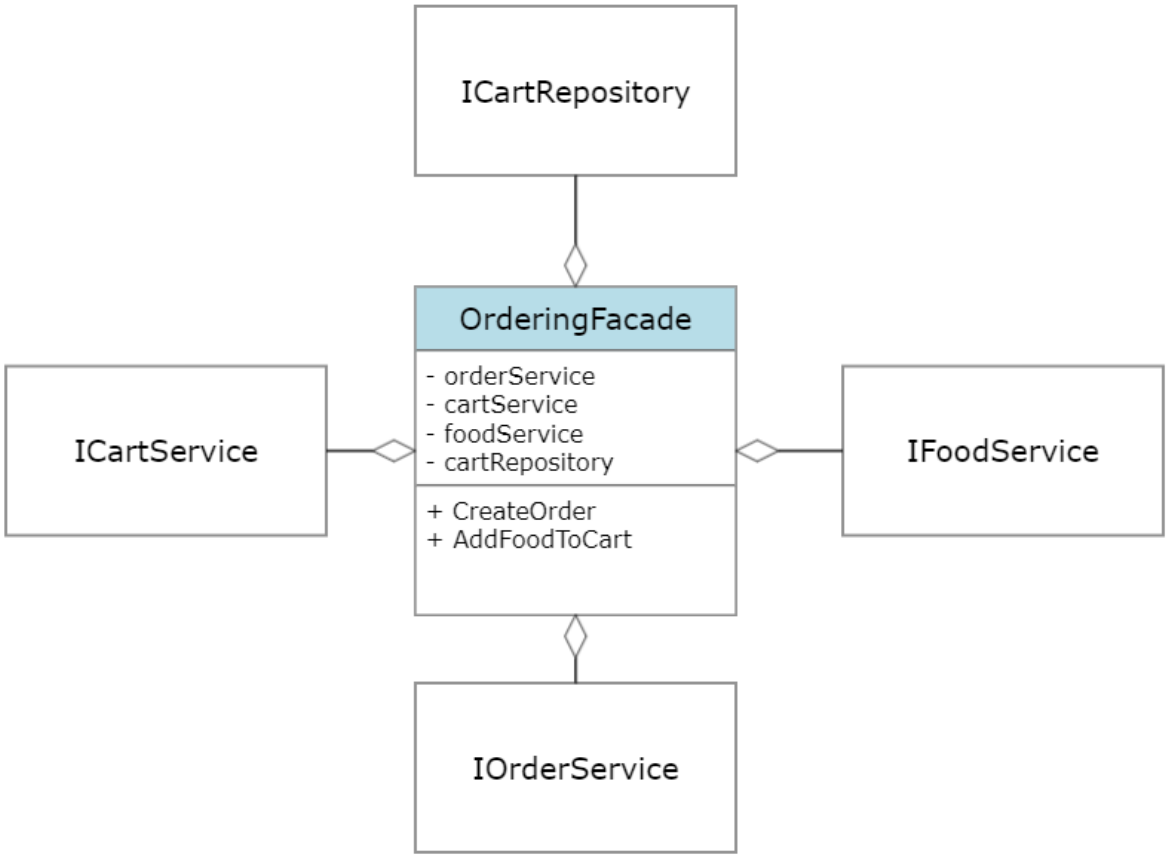
```
public class CartService : ICartService
{
    private readonly ICartRepository _cartRepository;
    private readonly IMapper _mapper;

    YelovSK
    public CartService(ICartRepository cartRepository, IMapper mapper)
    {
        _cartRepository = cartRepository;
        _mapper = mapper;
    }
```

```
public CartDto GetCartByUserId(int userId)
{
    var cart = _cartRepository.SingleOrDefault(expression: i:Cart => i.UserId == userId);

    if (cart == null)
    {
        throw new FoodOrderingException(message: "User not found");
    }

    return _mapper.Map<Cart, CartDto>(cart);
}
```

```
public class CartDto
{
    public int Id { get; set; }

    YelovSK
    public decimal TotalPrice => Items.Sum(f:FoodItemDto => f.Food.Price * f.Quantity);

    1 usage
    public List<FoodItemDto> Items { get; set; } = new();
}
```

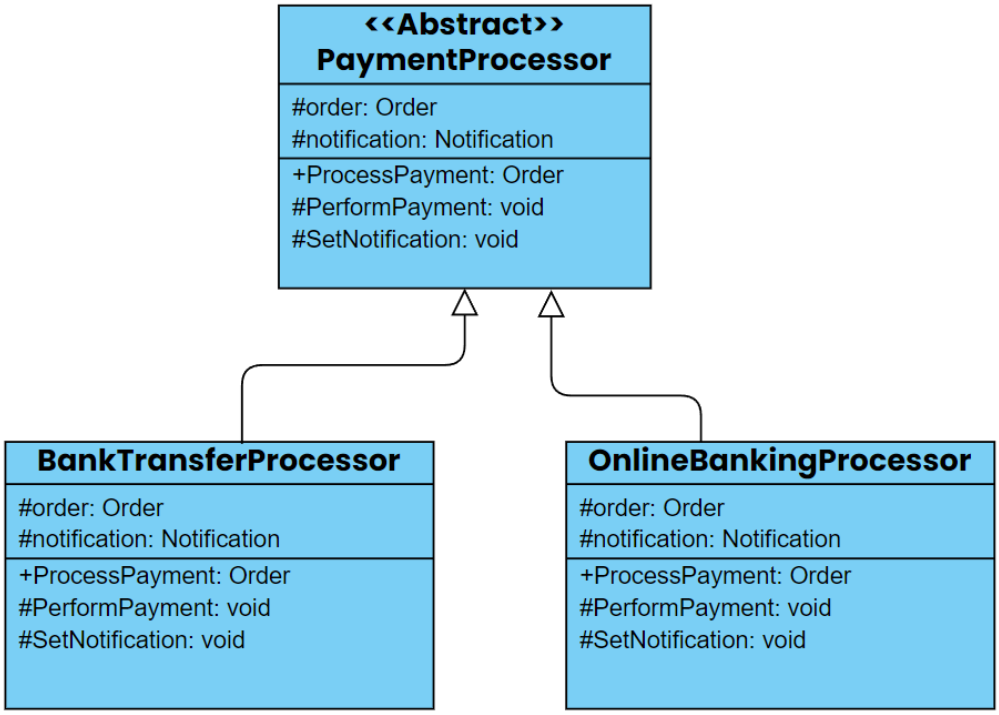
```
public class Cart
{
    2 usages
    public int Id { get; set; }

    [Required]
    [ForeignKey(nameof(User))]
    4 usages
    public int UserId { get; set; }

    2 usages
    public User User { get; set; }

    4 usages
    public ICollection<CartItem> Items { get; set; }

    YelovSK
    public decimal TotalPrice => Items.Sum(f:CartItem => f.Food.Price * f.Quantity);
}
```

Facade




```
public class OrderingFacade
{
    private readonly IOrderService _orderService;
    private readonly ICartService _cartService;
    private readonly IFoodService _foodService;
    private readonly ICartRepository _cartRepository;

    YelovSK
    public OrderingFacade(IOrderService orderService, ICartService cartService, IFoodService foodService, ICartRepository cartRepository)
    {
        _orderService = orderService;
        _cartService = cartService;
        _foodService = foodService;
        _cartRepository = cartRepository;
    }

    1 usage  YelovSK
    public OrderDto CreateOrder(int userId){...}

    1 usage  YelovSK
    public void AddFoodToCart(int userId, int foodId, int quantity){...}
}
```

Template method

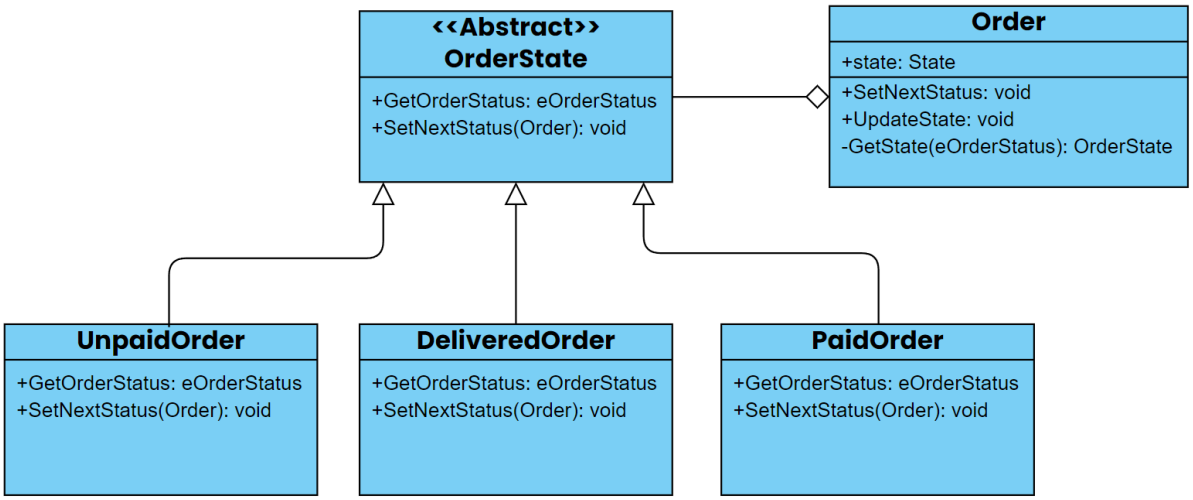


```
public abstract class PaymentProcessor
{
    protected readonly Order Order;
    protected readonly Notification Notification;

    2 usages  YelovSK
    protected PaymentProcessor(Order order, Notification notification){...}

    1 usage  YelovSK
    public Order ProcessPayment()
    {
        PerformPayment();
        SetNotification();
        return Order;
    }

    3 usages  2 overrides  YelovSK
    protected virtual void PerformPayment()
    {
        if (Order.Status ≠ eOrderStatus.Unpaid)
        {
            throw new FoodOrderingException( message: "Order is not in unpaid state");
        }
    }

    1 usage  2 overrides  YelovSK
    protected abstract void SetNotification();
}
```

```
public class OnlineBankingPaymentProcessor : PaymentProcessor
{
    1 usage  YelovSK
    public OnlineBankingPaymentProcessor(Order order, Notification notification) : base(order, notification)
    {
    }

    0+3 usages  YelovSK
    protected override void PerformPayment()
    {
        base.PerformPayment();

        Order.Status = eOrderStatus.Paid;
    }

    0+1 usages  YelovSK
    protected override void SetNotification()
    {
        Notification.Message = $"Performed online banking payment for order {Order.Id}";
        Order.User.Notifications.Add(Notification);
    }
}
```

```
public class BankTransferPaymentProcessor : PaymentProcessor
{
    1 usage  YelovSK
    public BankTransferPaymentProcessor(Order order, Notification notification) : base(order, notification)
    {
    }

    0+3 usages  YelovSK
    protected override void PerformPayment()
    {
        base.PerformPayment();

        Order.Status = eOrderStatus.Paid;
    }

    0+1 usages  YelovSK
    protected override void SetNotification()
    {
        Notification.Message = $"Performed bank transfer payment for order {Order.Id}";
        Order.User.Notifications.Add(Notification);
    }
}
```

State


<<Abstract>>
OrderState
+GetOrderStatus: eOrderStatus
+SetNextStatus(Order): void
Order
+state: State
+SetNextStatus: void
+UpdateState: void
-GetState(eOrderStatus): OrderState
UnpaidOrder
+GetOrderStatus: eOrderStatus
+SetNextStatus(Order): void
DeliveredOrder
+GetOrderStatus: eOrderStatus
+SetNextStatus(Order): void
PaidOrder
+GetOrderStatus: eOrderStatus
+SetNextStatus(Order): void

```
public partial class Order
{
    [NotMapped]
    6 usages
    public OrderState? State { get; set; }

    1 usage   YelovSK
    public void SetNextStatus()
    {
        State ??= GetState(Status);

        State.SetNextStatus( context: this);
    }

    1 usage   YelovSK
    public void UpdateState()
    {
        State ??= GetState(Status);

        Status = State.GetOrderStatus();
    }

    2 usages   YelovSK
    private static OrderState GetState(eOrderStatus status)
    {
        return status switch
        {
            eOrderStatus.Unpaid    => new UnpaidOrder(),
            eOrderStatus.Paid      => new PaidOrder(),
            eOrderStatus.Delivered => new DeliveredOrder(),
            _                      => throw new FoodOrderingException( message: "Invalid status");
        };
    }
}
```

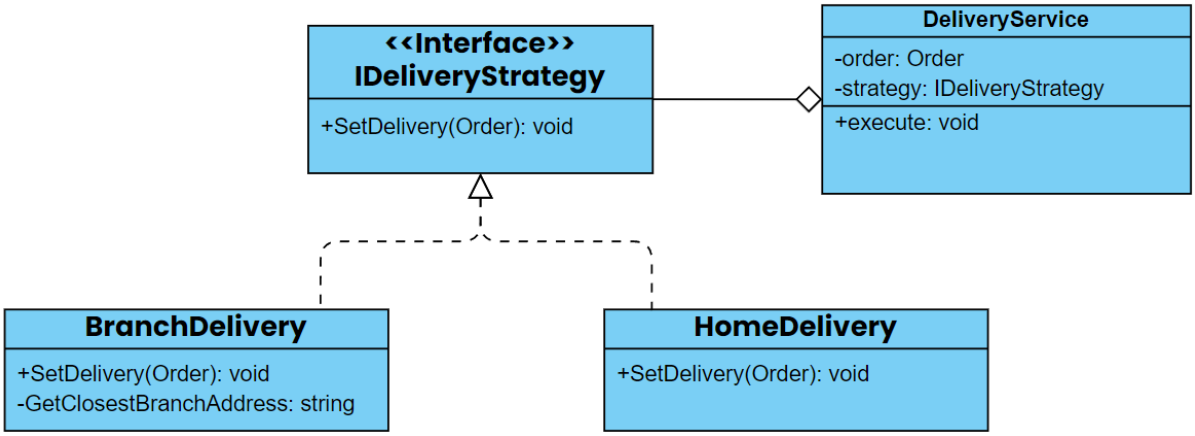
```
public abstract class OrderState
{
    1 usage  3 overrides  YelovSK
    public abstract eOrderStatus GetOrderStatus();

    1 usage  3 overrides  YelovSK
    public abstract void SetNextStatus(Order context);
}
```

```
public class DeliveredOrder : OrderState
{
    0+1 usages  YelovSK
    public override eOrderStatus GetOrderStatus() => eOrderStatus.Delivered;

    0+1 usages  YelovSK
    public override void SetNextStatus(Order context)
    {
        throw new FoodOrderingException( message: "There is no next state for delivered order.");
    }
}
```

```
public class PaidOrder : OrderState
{
    0+1 usages  YelovSK
    public override eOrderStatus GetOrderStatus() => eOrderStatus.Paid;

    0+1 usages  YelovSK
    public override void SetNextStatus(Order context)
    {
        context.State = new DeliveredOrder();
    }
}
```

```
public class UnpaidOrder : OrderState
{
    0+1 usages   YelovSK
    public override eOrderStatus GetOrderStatus() => eOrderStatus.Unpaid;

    0+1 usages   YelovSK
    public override void SetNextStatus(Order context)
    {
        context.State = new PaidOrder();
    }
}
```

Strategy



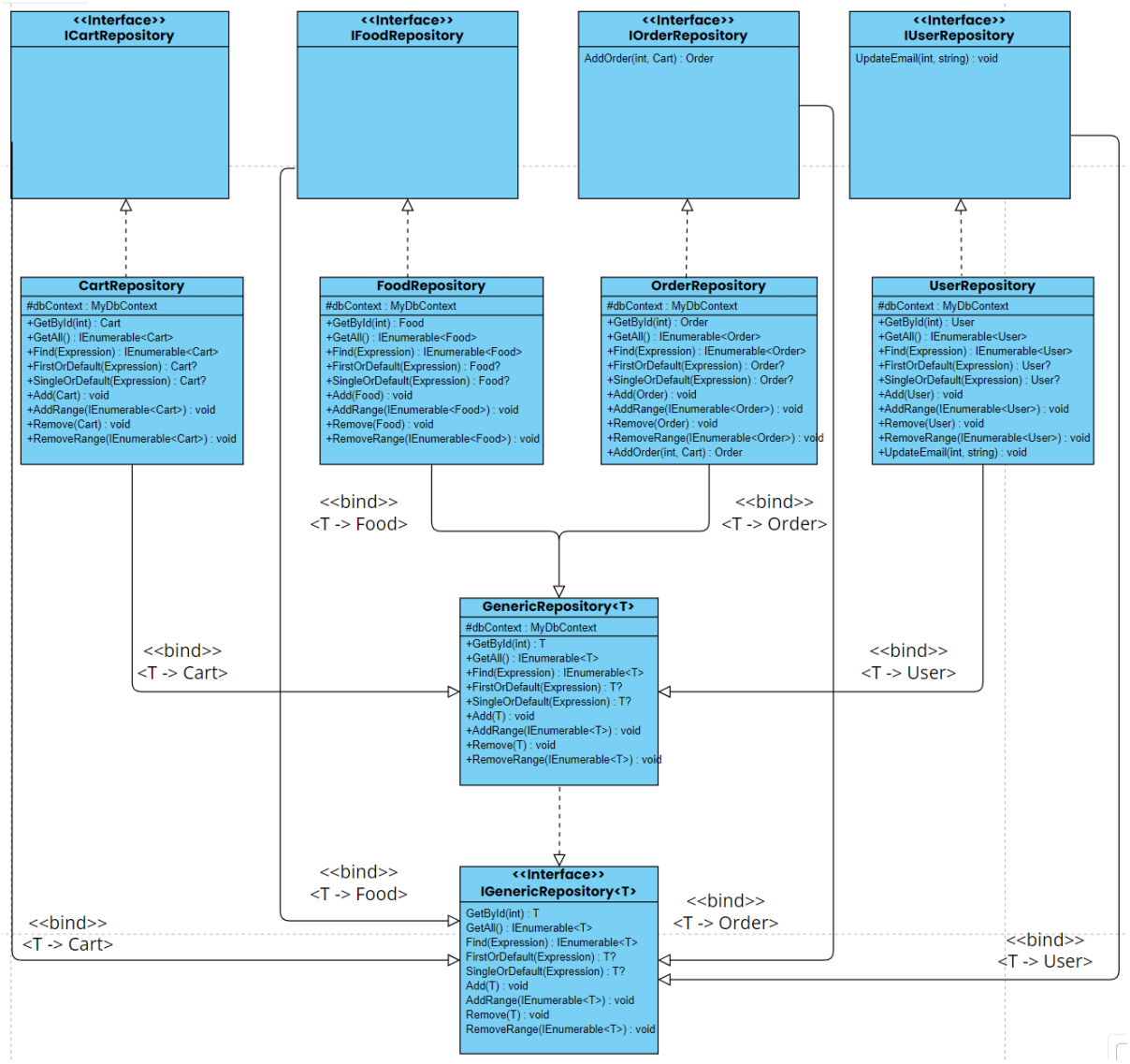
```
public class DeliveryService
{
    private readonly Order _order;
    private readonly IDeliveryStrategy _strategy;

    1 usage  YelovSK *
    public DeliveryService(Order order, IDeliveryStrategy strategy)
    {
        _order = order;
        _strategy = strategy;
    }

    1 usage  YelovSK *
    public void Execute()
    {
        _strategy.SetDelivery(_order);
    }
}
```

```
public interface IDeliveryStrategy
{
    1 usage  2 implementations  YelovSK
    void SetDelivery(Order order);
}
```

```
public class BranchDelivery : IDeliveryStrategy
{
    private const string DUMMY_BRANCH_ADDRESS = "Restaurant 322, Uganda";

    0+1 usages   YelovSK *
    public void SetDelivery(Order order)
    {
        order.DeliveryType = eDeliveryType.BranchDelivery;
        order.Address = GetClosestBranchAddress();
        order.User.Notifications.Add( item: new Notification
        {
            UserId = order.User.Id,
            SentDate = DateTime.UtcNow,
            Message = $"Delivery of order {order.Id} is set to {DUMMY_BRANCH_ADDRESS}",
        });
    }

    1 usage   YelovSK *
    private string GetClosestBranchAddress()
    {
        // Um, imagine some logic here
        return DUMMY_BRANCH_ADDRESS;
    }
}
```

```
public class HomeDelivery : IDeliveryStrategy
{
    0+1 usages   YelovSK
    public void SetDelivery(Order order)
    {
        order.DeliveryType = eDeliveryType.HomeDelivery;
        order.Address = order.User.Address;
        order.User.Notifications.Add( item: new Notification
        {
            UserId = order.User.Id,
            SentDate = DateTime.UtcNow,
            Message = $"Delivery of order {order.Id} is set to home delivery",
        });
    }
}
```

Repository


<<Interface>>
ICartRepository
<<Interface>>
IFoodRepository
<<Interface>>
IOrderRepository
AddOrder(int, Cart) : Order
<<Interface>>
IUserRepository
UpdateEmail(int, string) : void
CartRepository
#dbContext : MyDbContext
+GetById(int) : Cart
+GetAll() : IEnumerable<Cart>
+Find(Expression) : IEnumerable<Cart>
+FirstOrDefault(Expression) : Cart?
+SingleOrDefault(Expression) : Cart?
+Add(Cart) : void
+AddRange(IEnumerable<Cart>) : void
+Remove(Cart) : void
+RemoveRange(IEnumerable<Cart>) : void
FoodRepository
#dbContext : MyDbContext
+GetById(int) : Food
+GetAll() : IEnumerable<Food>
+Find(Expression) : IEnumerable<Food>
+FirstOrDefault(Expression) : Food?
+SingleOrDefault(Expression) : Food?
+Add(Food) : void
+AddRange(IEnumerable<Food>) : void
+Remove(Food) : void
+RemoveRange(IEnumerable<Food>) : void
OrderRepository
#dbContext : MyDbContext
+GetById(int) : Order
+GetAll() : IEnumerable<Order>
+Find(Expression) : IEnumerable<Order>
+FirstOrDefault(Expression) : Order?
+SingleOrDefault(Expression) : Order?
+Add(Order) : void
+AddRange(IEnumerable<Order>) : void
+Remove(Order) : void
+RemoveRange(IEnumerable<Order>) : void
+AddOrder(int, Cart) : Order
UserRepository
#dbContext : MyDbContext
+GetById(int) : User
+GetAll() : IEnumerable<User>
+Find(Expression) : IEnumerable<User>
+FirstOrDefault(Expression) : User?
+SingleOrDefault(Expression) : User?
+Add(User) : void
+AddRange(IEnumerable<User>) : void
+Remove(User) : void
+RemoveRange(IEnumerable<User>) : void
+UpdateEmail(int, string) : void
<<bind>>
<T -> Food>
<<bind>>
<T -> Order>
GenericRepository<T>
#dbContext : MyDbContext
+GetById(int) : T
+GetAll() : IEnumerable<T>
+Find(Expression) : IEnumerable<T>
+FirstOrDefault(Expression) : T?
+SingleOrDefault(Expression) : T?
+Add(T) : void
+AddRange(IEnumerable<T>) : void
+Remove(T) : void
+RemoveRange(IEnumerable<T>) : void
<<bind>>
<T -> Cart>
<<bind>>
<T -> User>
<<bind>>
<T -> Food>
<<Interface>>
IGenericRepository<T>
GetById(int) : T
GetAll() : IEnumerable<T>
Find(Expression) : IEnumerable<T>
FirstOrDefault(Expression) : T?
SingleOrDefault(Expression) : T?
Add(T) : void
AddRange(IEnumerable<T>) : void
Remove(T) : void
RemoveRange(IEnumerable<T>) : void
<<bind>>
<T -> Order>
<<bind>>
<T -> Cart>
<<bind>>
<T -> User>

```
public interface IGenericRepository<T> where T : class
{
    6 usages  1 implementation  YelovSK
    T? GetById(int id);
    1 implementation  YelovSK
    IEnumerable<T> GetAll();
    1 usage  1 implementation  YelovSK
    IEnumerable<T> Find(Expression<Func<T, bool>> expression);
    1 implementation  YelovSK
    T? FirstOrDefault(Expression<Func<T, bool>> expression);
    4 usages  1 implementation  YelovSK
    T? SingleOrDefault(Expression<Func<T, bool>> expression);
    1 implementation  YelovSK
    void Add(T entity);
    1 implementation  YelovSK
    void AddRange(IEnumerable<T> entities);
    1 implementation  YelovSK
    void Remove(T entity);
    1 implementation  YelovSK
    void RemoveRange(IEnumerable<T> entities);

    6 usages  1 implementation  YelovSK
    void Save();
}
```

```
public class GenericRepository<T> : IGenericRepository<T> where T : class
{
    protected readonly MyDbContext _context;

    4 usages  YelovSK
    public GenericRepository(MyDbContext context)
    {
        _context = context;
    }

    0+4 usages  YelovSK
    public T? SingleOrDefault(Expression<Func<T, bool>> expression)
    {
        return _context.Set<T>().SingleOrDefault(expression);
    }

    YelovSK
    public void Add(T entity)
    {
        _context.Set<T>().Add(entity);
    }

    YelovSK
    public void AddRange(IEnumerable<T> entities)
    {
        _context.Set<T>().AddRange(entities);
    }

    0+1 usages  YelovSK
    public IEnumerable<T> Find(Expression<Func<T, bool>> expression)
    {
        return _context.Set<T>().Where(expression);
    }
```

```
public interface IOrderRepository : IGenericRepository<Order>
{
    1 usage  1 implementation  YelovSK
    Order AddOrder(int userId, Cart cart);
}
```

```
public class OrderRepository : GenericRepository<Order>, IOrderRepository
{
    YelovSK
    public OrderRepository(MyDbContext context) : base(context)
    {
    }

    0+1 usages   YelovSK *
    public Order AddOrder(int userId, Cart cart)
    {
        var order = new Order
        {
            UserId = userId,
            Status = eOrderStatus.Unpaid,
            Address = string.Empty,
            DeliveryType = 0,
        };

        _context.Orders.Add(order);
        Save();

        // Map cart items to order items
        foreach (var item in cart.Items)
        {
            _context.OrderItems.Add( entity: new OrderItem
            {
                OrderId = order.Id,
                FoodId = item.FoodId,
                Quantity = item.Quantity,
            });
        }

        return order;
    }
}
```