# How do I update aggregate child in DDD?

Asked 1 year, 2 months ago Active 1 year, 2 months ago Viewed 457 times

0

I am trying out DDD-style architecture with asp.net core but I'm having trouble understanding how to Update a aggregate roots child-entites after its created.

I have a order-class which has a read-only list of OrderLines. These orderlines can be updated or removed from the order. The order will always be edited in the frontend in a single transaction(post to API).

I have made methods for this on the order aggregate, but it feels like the logic is in the wrong place. Is this the correct way to update/delete child-entites in DDD?

Since the Order will come in to a API in a PUT-fashion I check if any items that are on the entity from the DB are not in the incoming data. If not, I remove them. If they are, I update them.

Does this belong in a service-class instead? What is the best practice in DDD?

```
public class Order : BaseEntity, IAggregateRoot
    {
        public Order(List<OrderItem> items, Address shippingAddress, int customerId)
        {
            ShipToAddress = shippingAddress ?? throw new Exception("Can not be null");
            _orderItems = items ?? throw new Exception("Can not be null");
```

```
            CustomerId = customerId;
            Status = OrderStatus.Processing;
        }

        private readonly List<OrderItem> _orderItems = new List<OrderItem>();

        public IReadOnlyCollection<OrderItem> OrderItems => _orderItems.AsReadOnly();

        public void AddOrUpdateOrderItem(ProductOrdered itemOrdered, decimal unitPrice,
int units, DateTime deliverDate, int orderItemId)
        {
            var existingOrderLine = _orderItems.Where(o => o.Id == orderItemId)
                .SingleOrDefault();

            if (existingOrderLine != null)
            {
                existingOrderLine.Update(itemOrdered, unitPrice, units, deliverDate);
            }
            else
            {
                //add validated new order item
                var orderItem = new OrderItem(itemOrdered, unitPrice, units,
deliverDate);
                _orderItems.Add(orderItem);
            }
        }
        public void RemoveOrderLines(List<int> orderItemIds)
        {
            foreach (var item in _orderItems.ToList())
            {
                var containsItem = orderItemIds.Any(newOrderLine => newOrderLine ==
item.Id);
                if (!containsItem)
                {
                    _orderItems.Remove(item);
                }
            }
        }
    }
```

OrderLine:

```
public class OrderItem : BaseEntity
    {
        public ProductOrdered ItemOrdered { get; private set; }
        public decimal UnitPrice { get; private set; }
        public int Units { get; private set; }
```

```
        public DateTime DeliveryDate { get; set; }
        public OrderLineStatus OrderLineStatus { get; set; }

        private OrderItem()
        {
            // required by EF
        }

        public OrderItem(ProductOrdered itemOrdered, decimal unitPrice, int units,
DateTime deliverDate)
        {
            ItemOrdered = itemOrdered;
            UnitPrice = unitPrice;
            Units = units;
            DeliveryDate = deliverDate;
            OrderLineStatus = OrderLineStatus.Waiting;
        }

        public void Update(ProductOrdered itemOrdered, decimal unitPrice, int units,
DateTime deliverDate)
        {
            ItemOrdered = itemOrdered;
            UnitPrice = unitPrice;
            Units = units;
            DeliveryDate = deliverDate;
        }
    }
```

c# asp.net-core domain-driven-design

edited Jan 25 '19 at 17:49

Ivan Stoev

133k ● 8 ● 136 ● 194

asked Jan 25 '19 at 17:25

klas mack

457 ● 6 ● 14

## 1 Answer

Active | Oldest | Votes

0

I have made methods for this on the order aggregate, but it feels like the logic is in the wrong place. Is this the correct way to update/delete child-entites in DDD? Yes. I just don't like that 2 separate usecases are defined in one. I would do Add in seaparate method and Update in another. Regarding to Put endpoint. Consider using Task-Based UI. It fits DDD much better than CrudBased put operation. So define commands - AddLineItemCommand, UpdateLineItemCommand, RemoveLineItemCommand and different POST operations for them. Treat each usecase differently.

answered Jan 31 '19 at 9:38

DmitriBodiu

**681** ● 4 ● 13