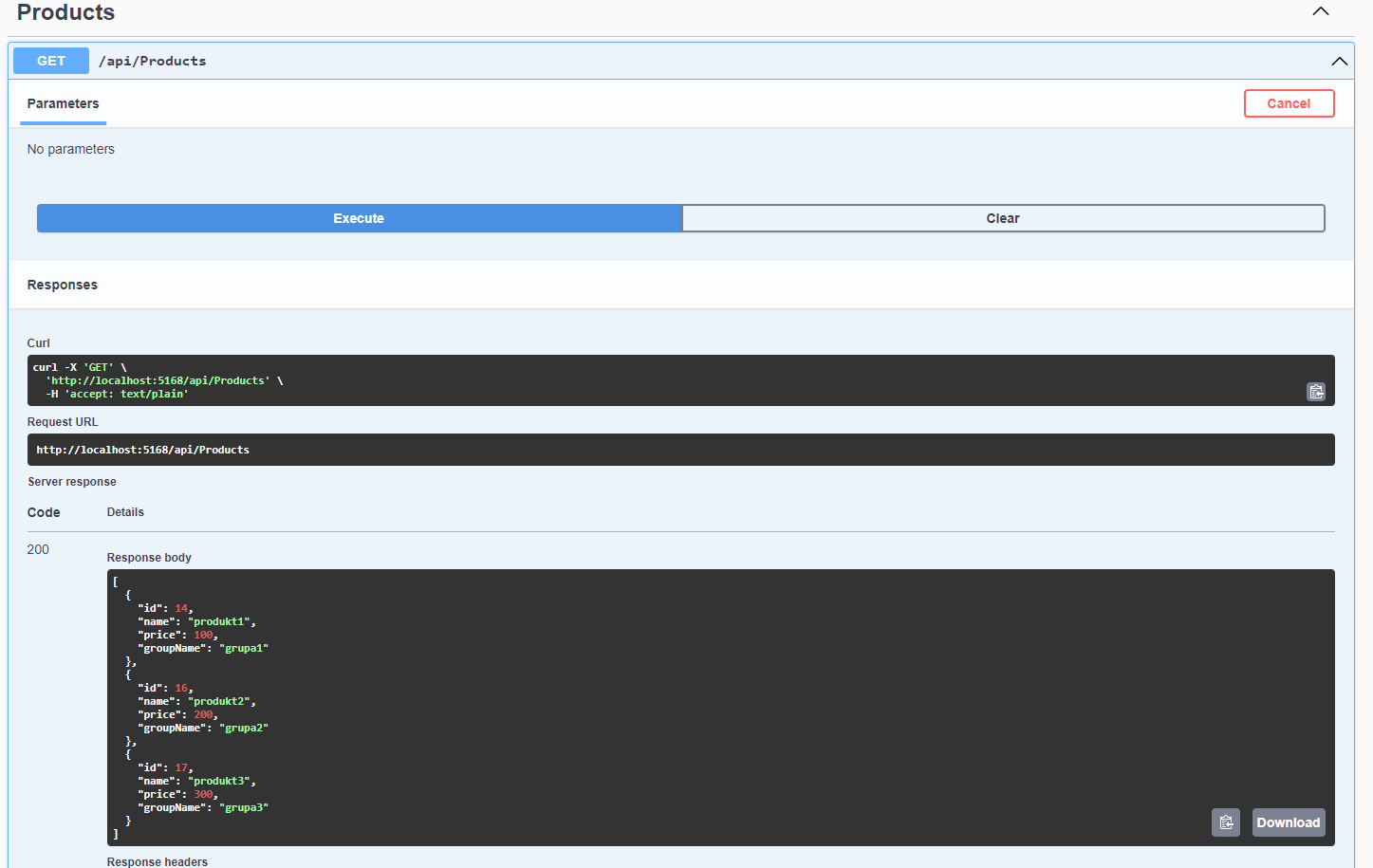
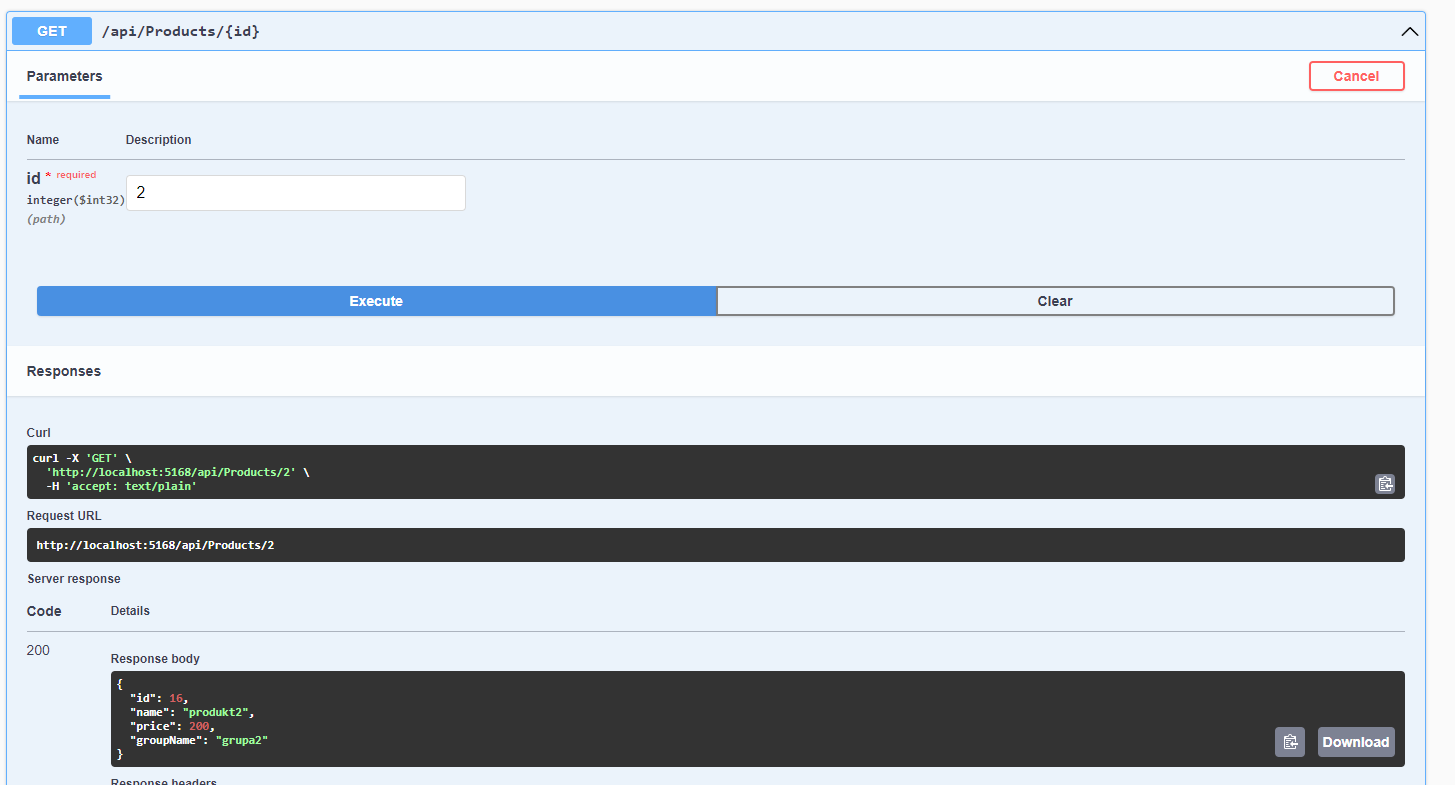
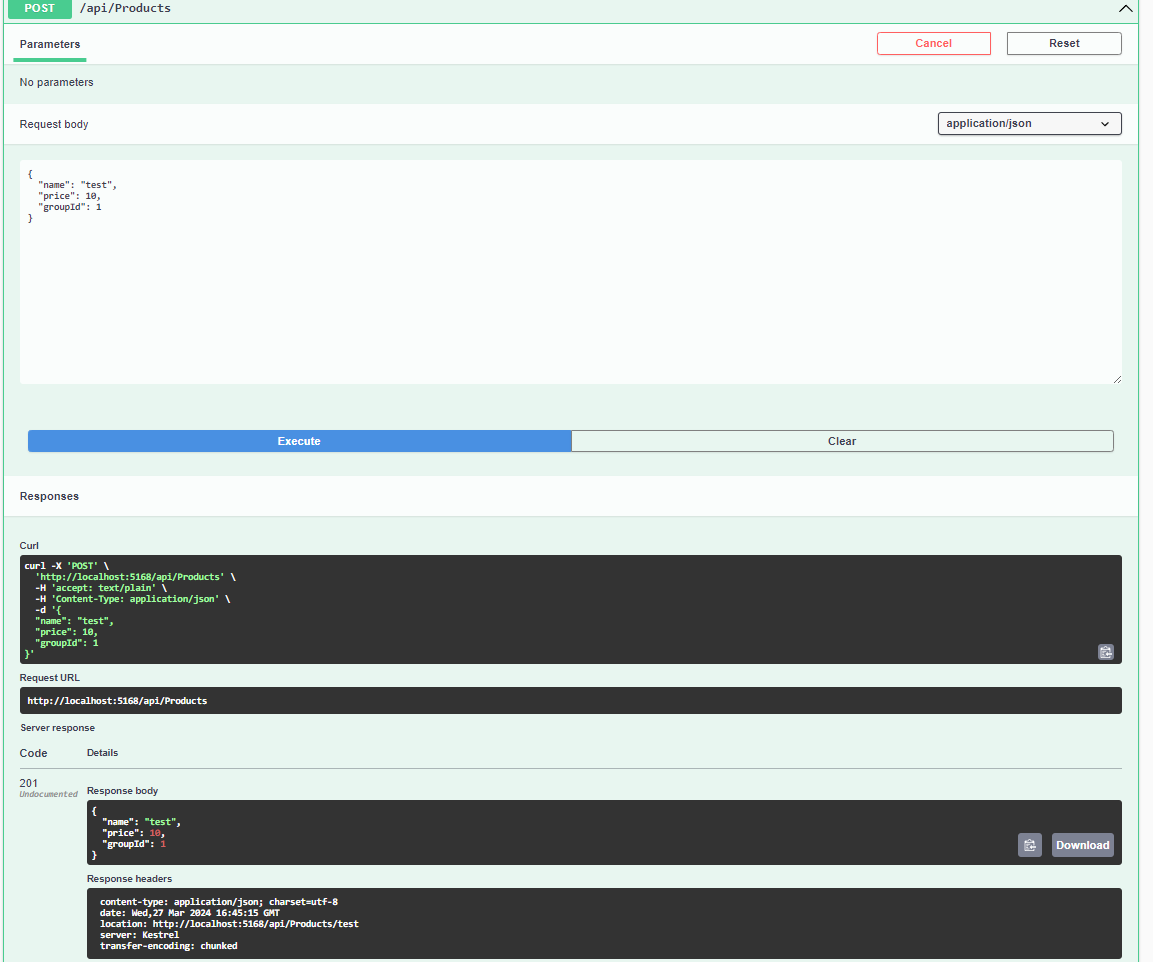
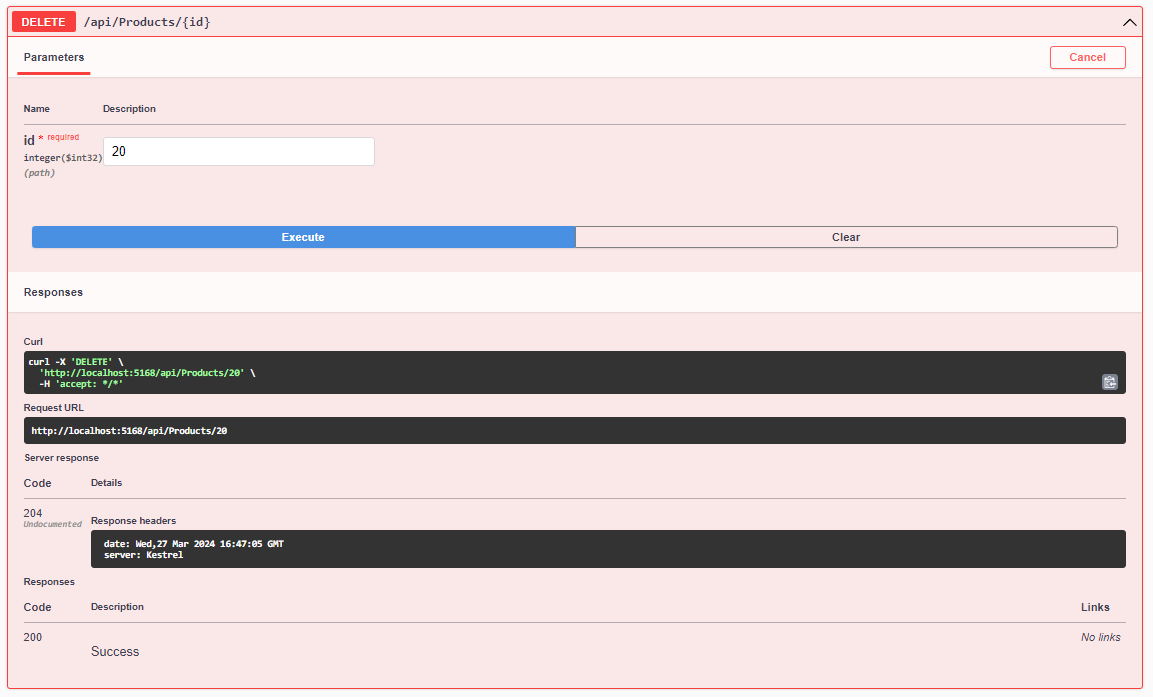
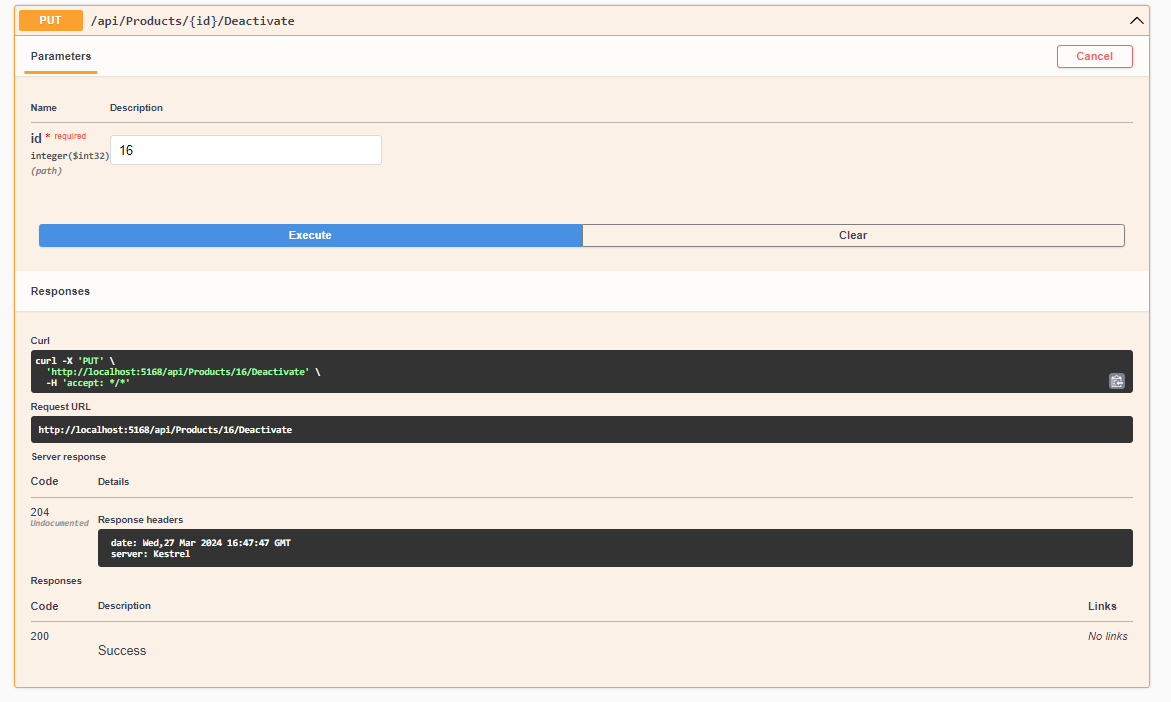
EF  
Products

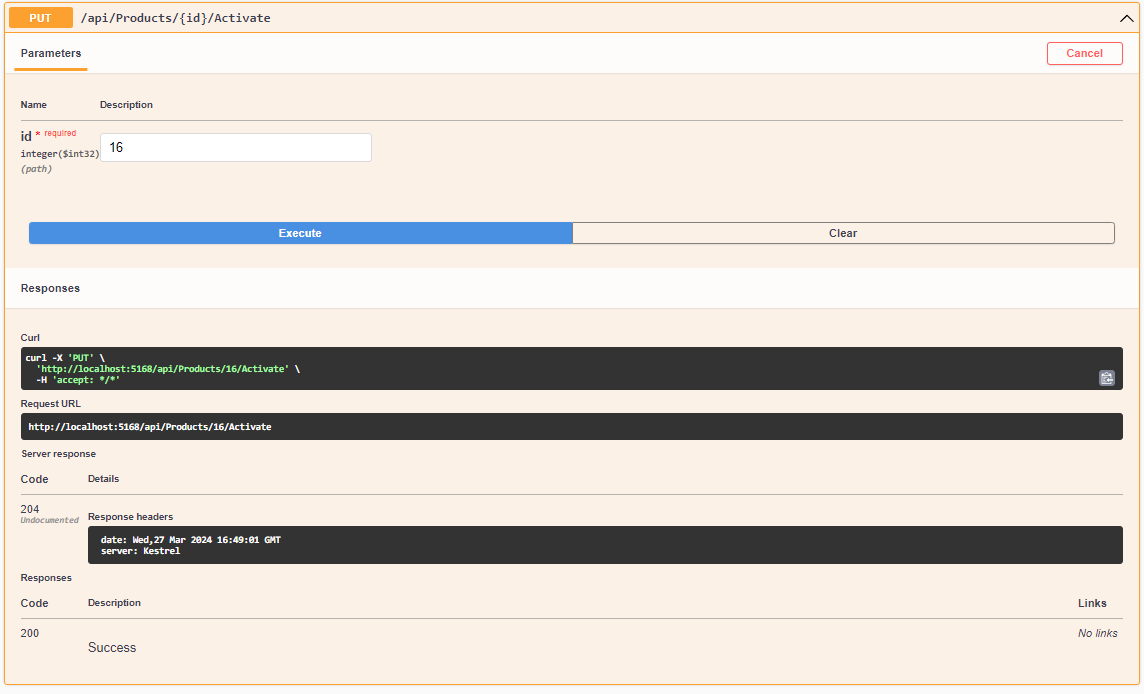


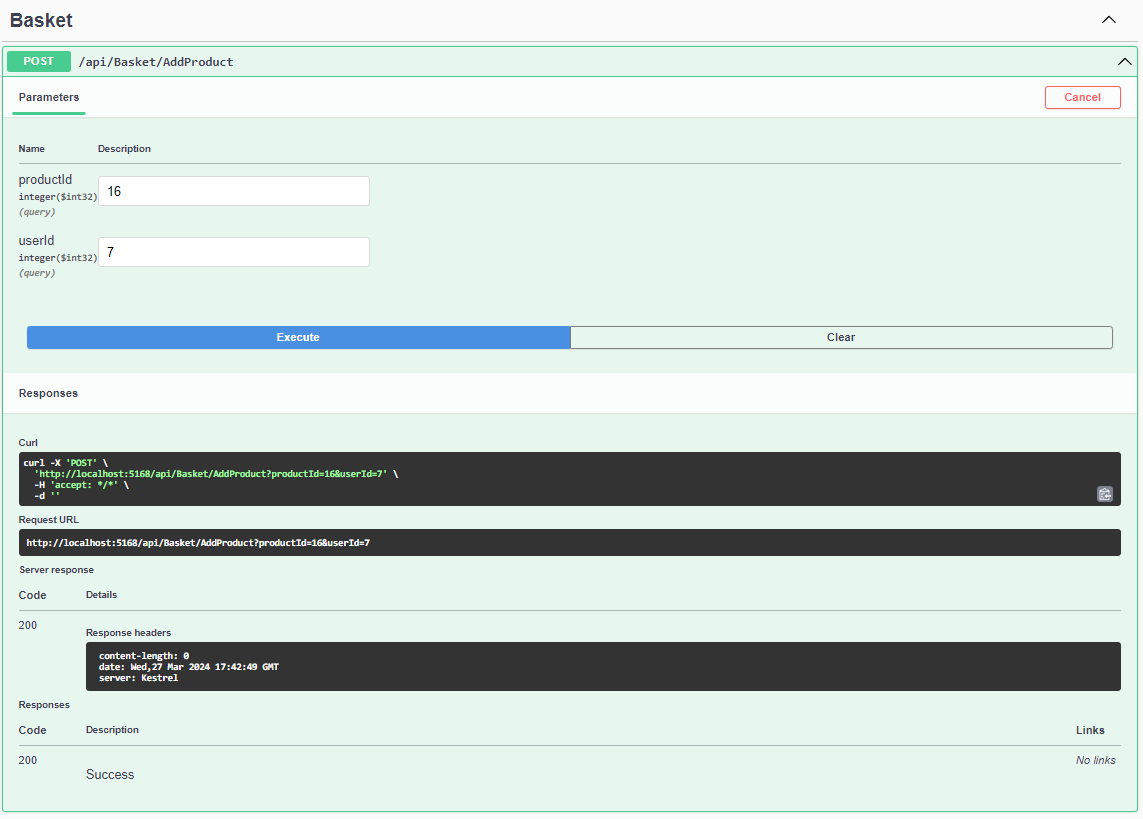
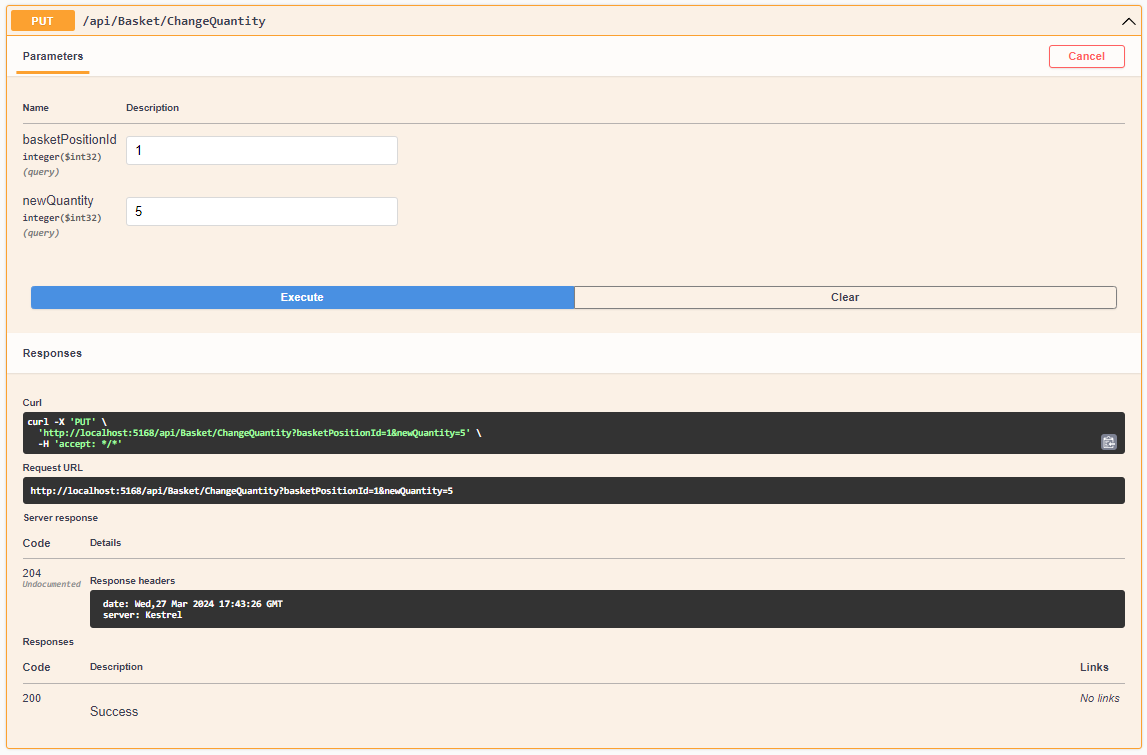
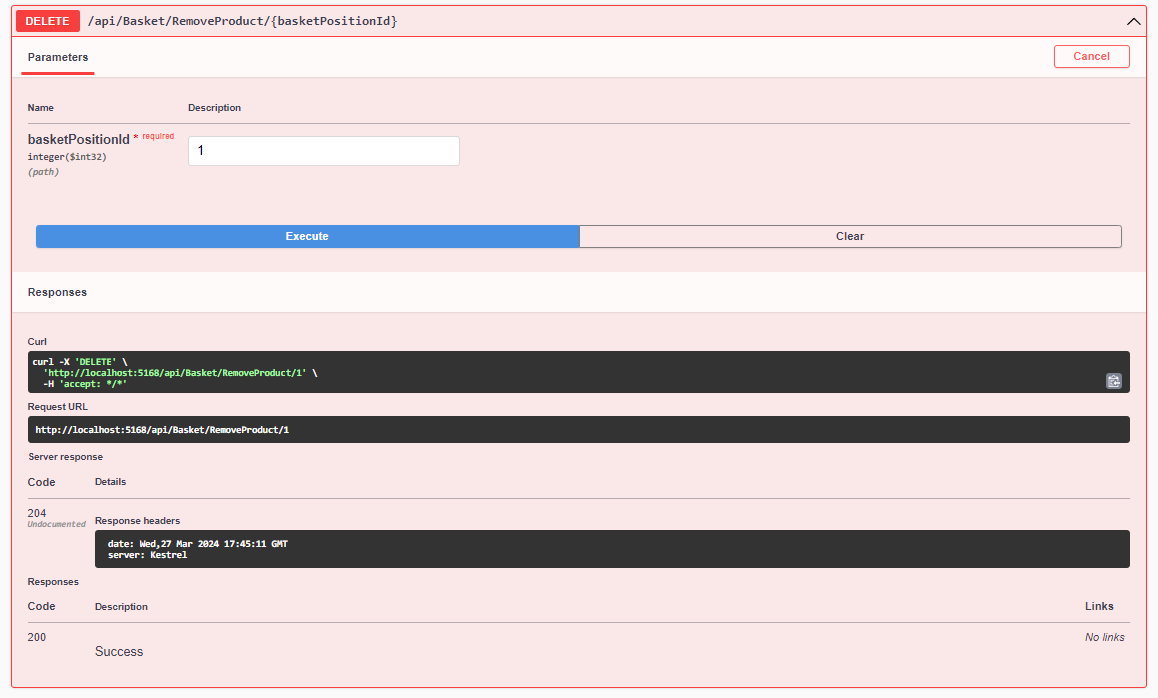


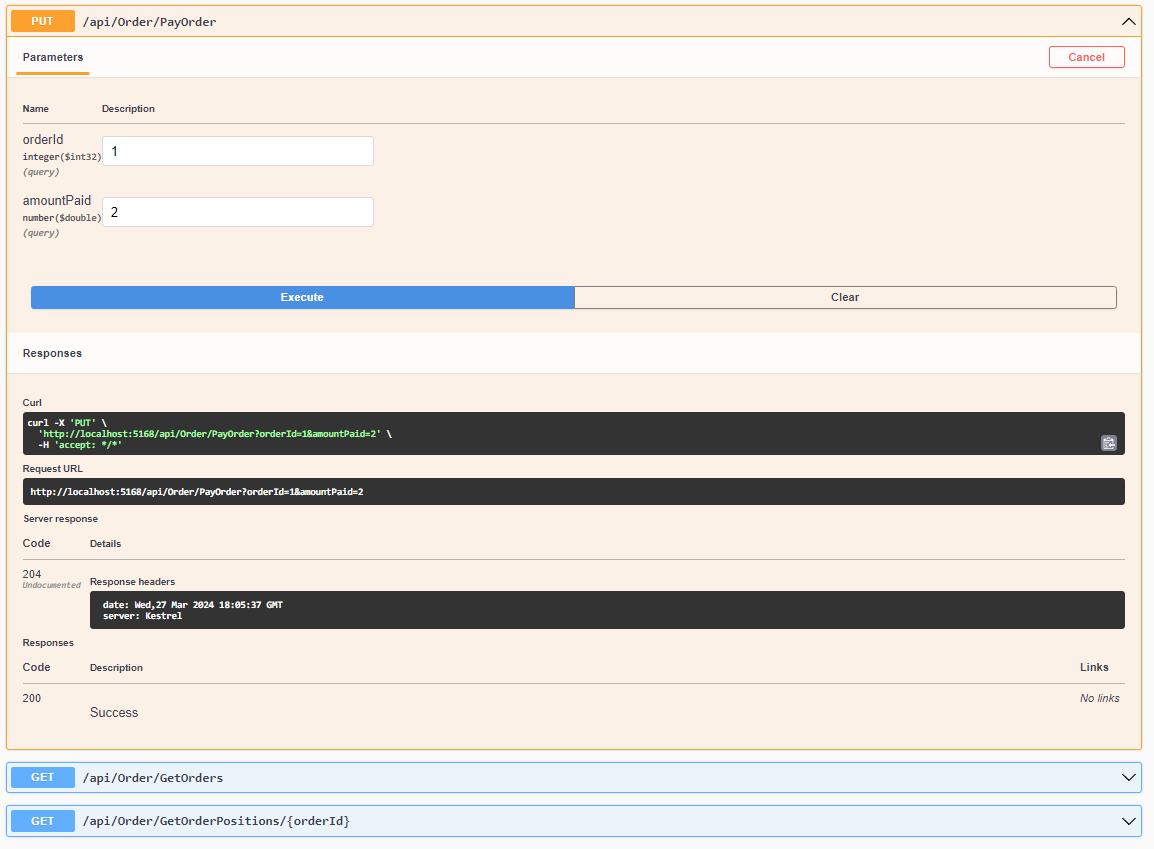
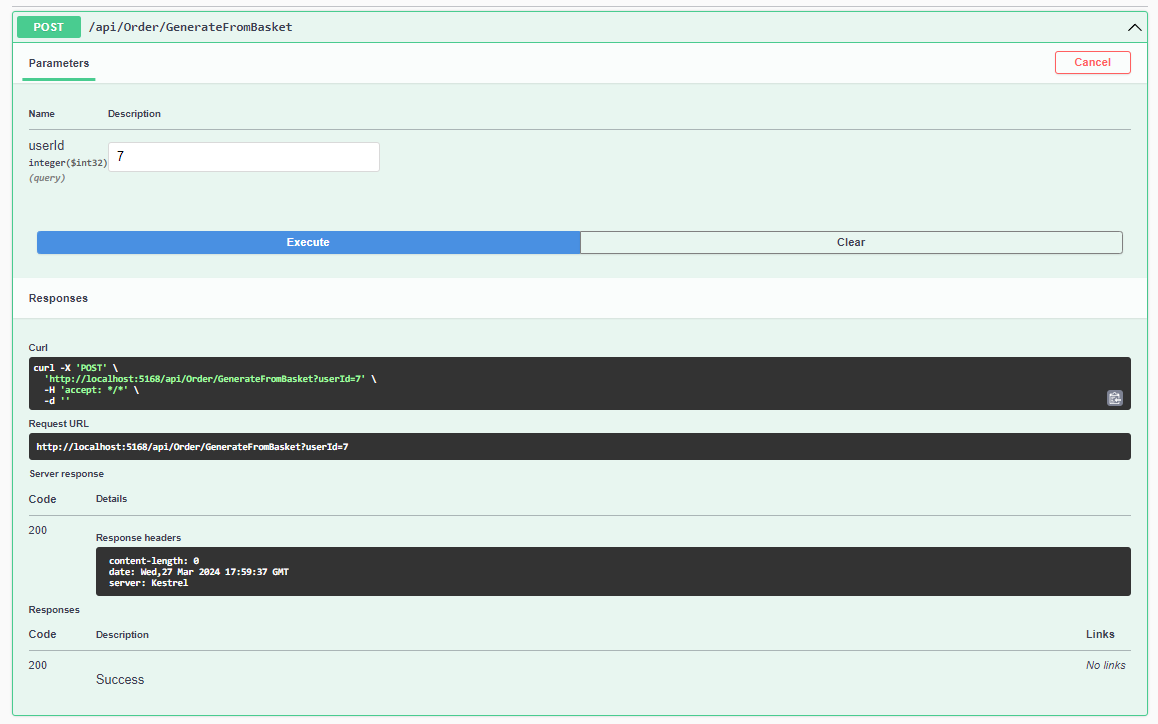
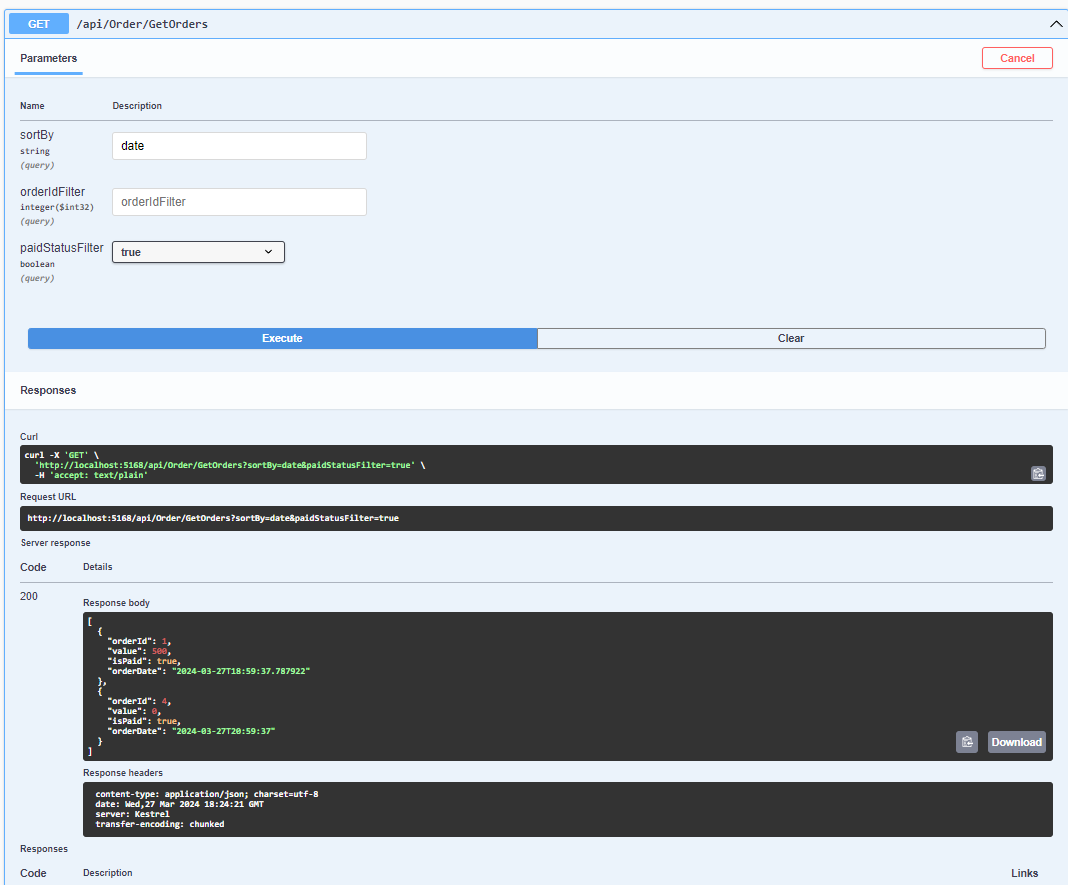
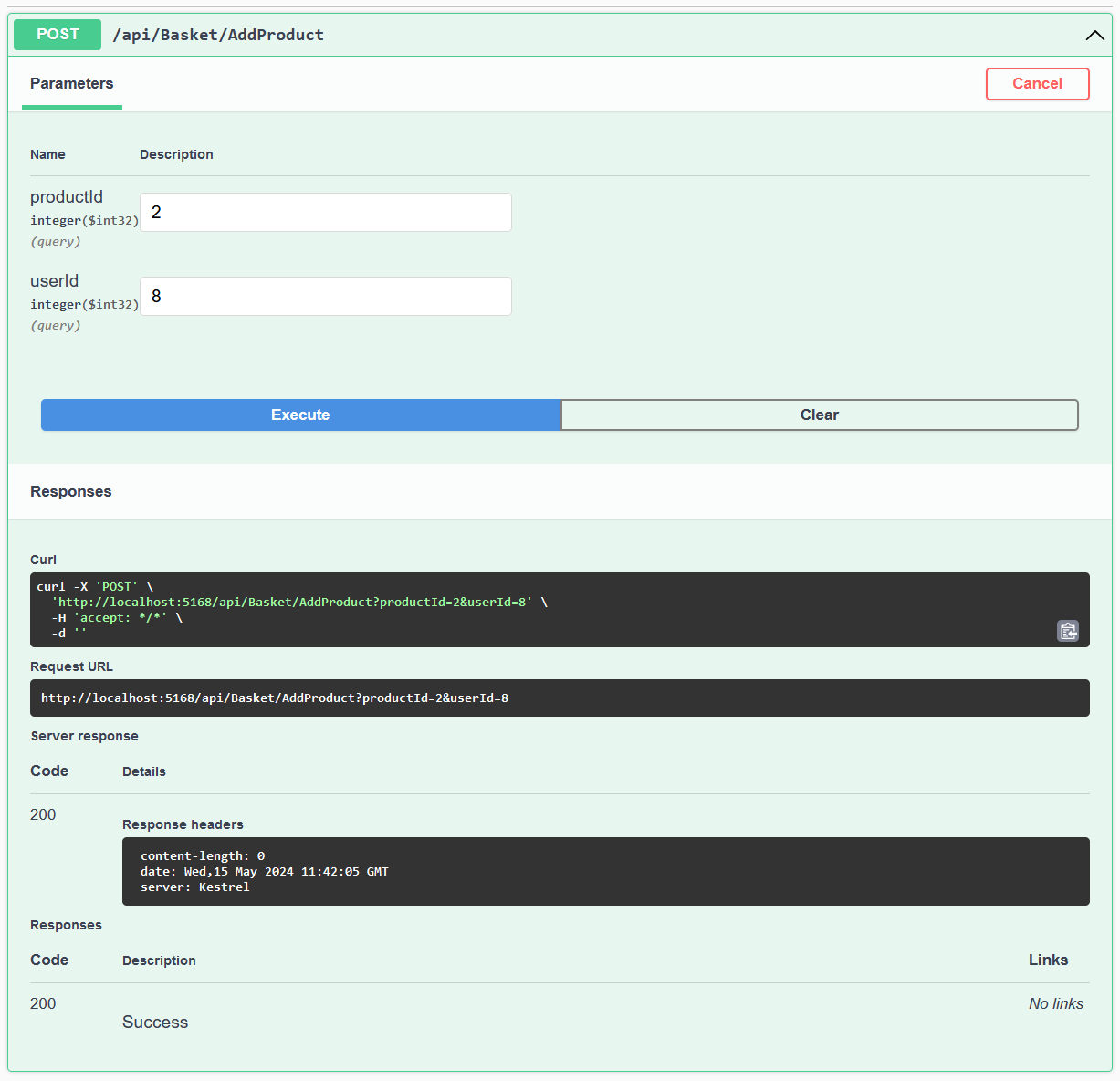
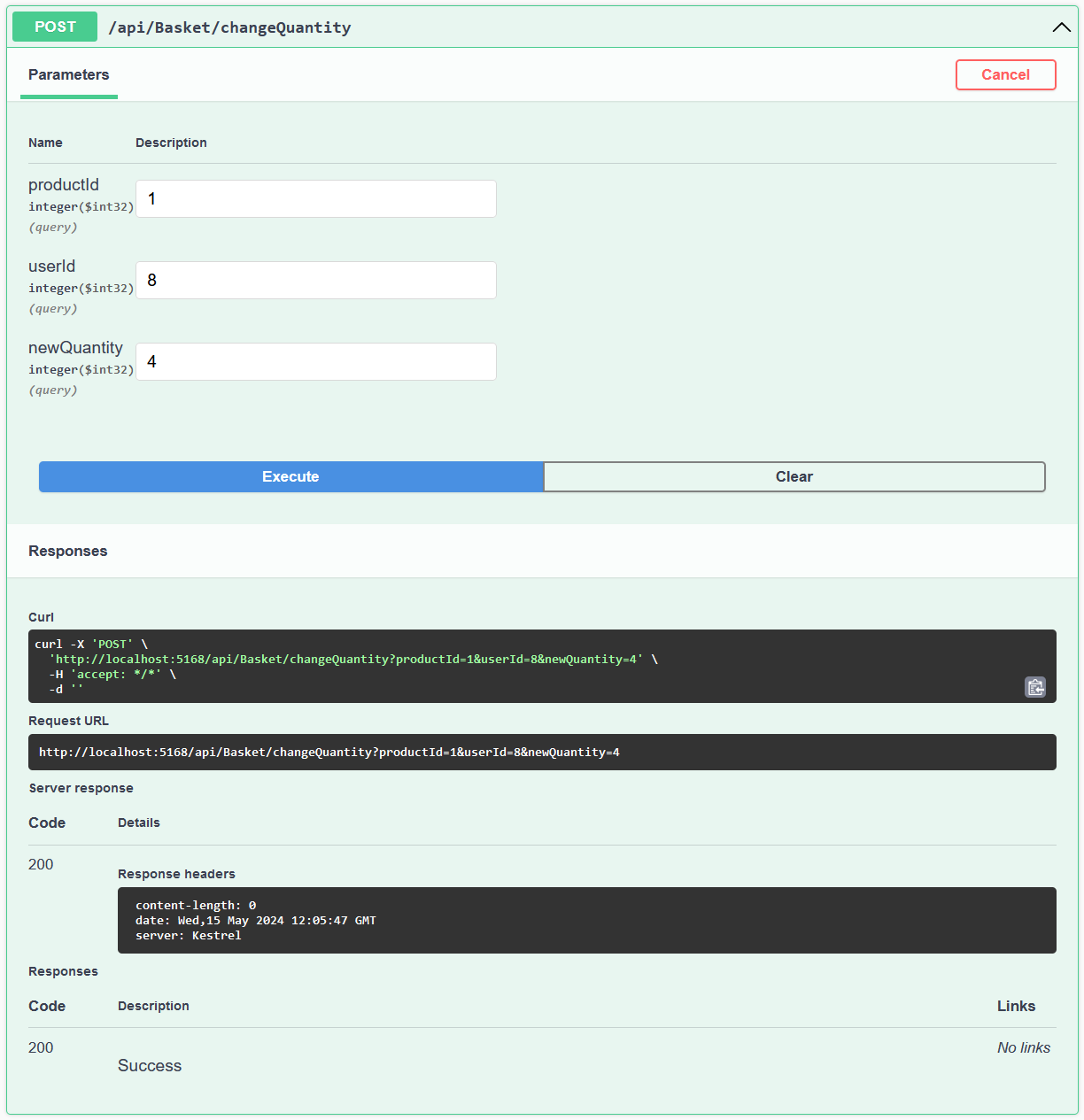


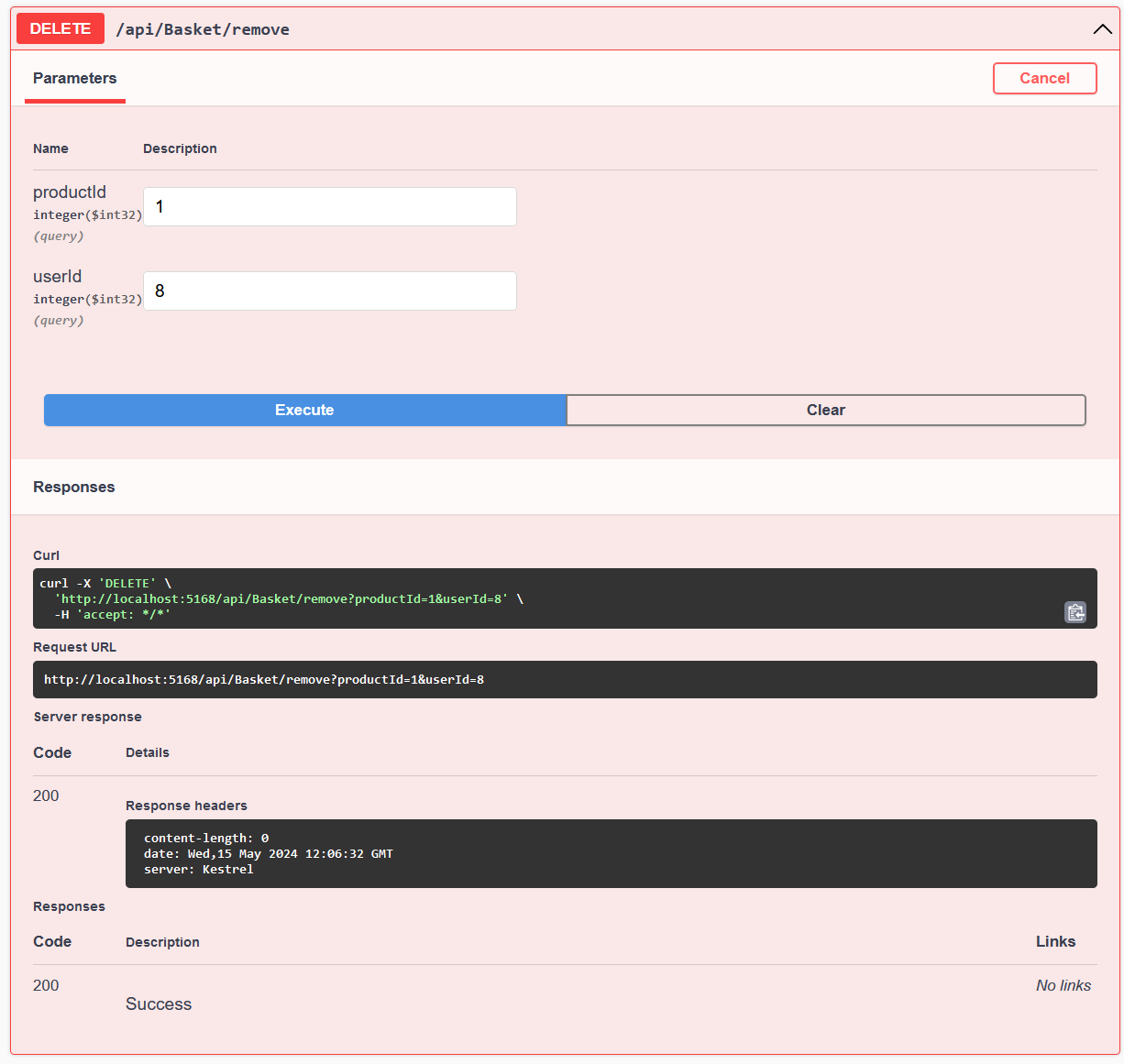


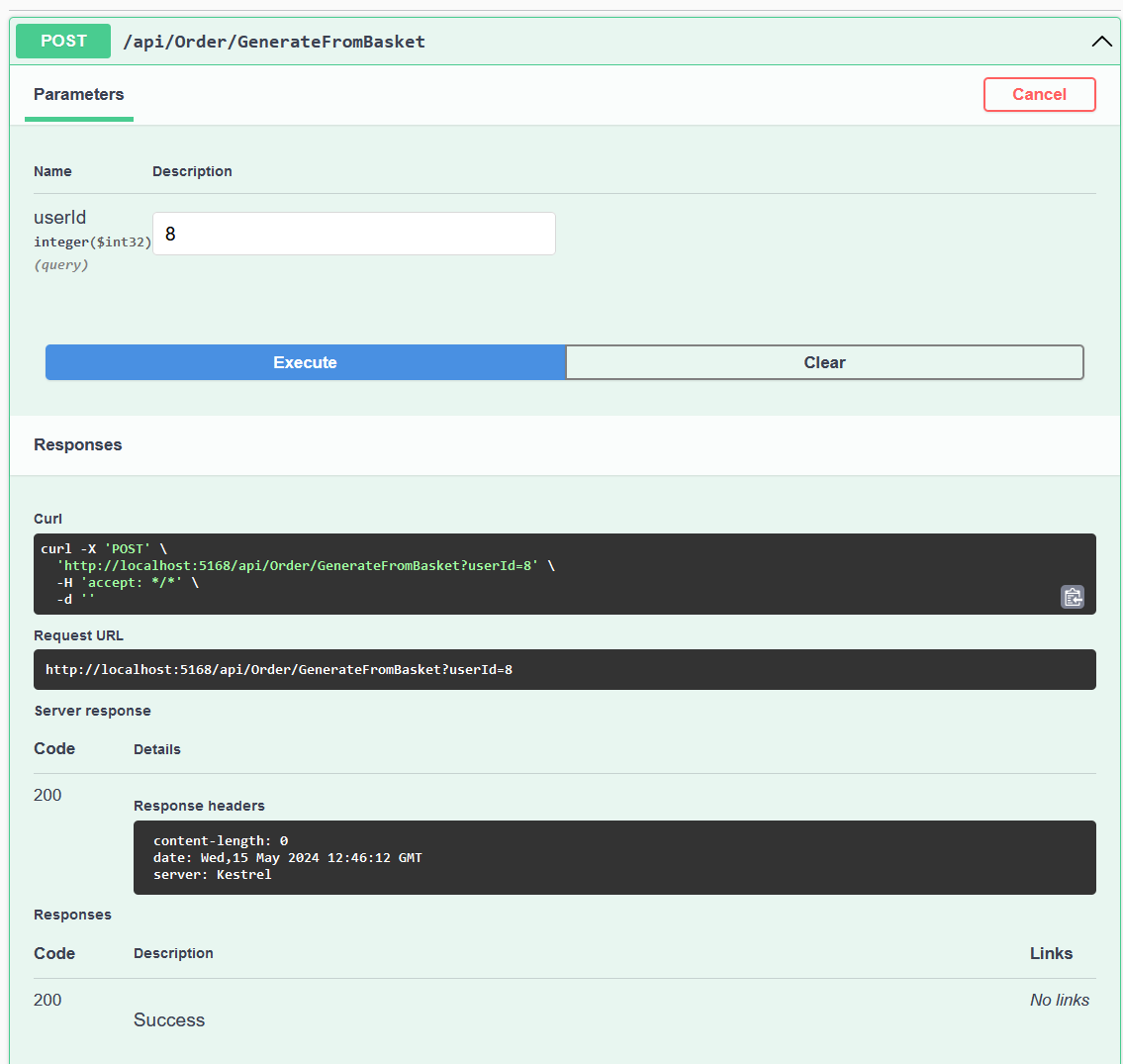
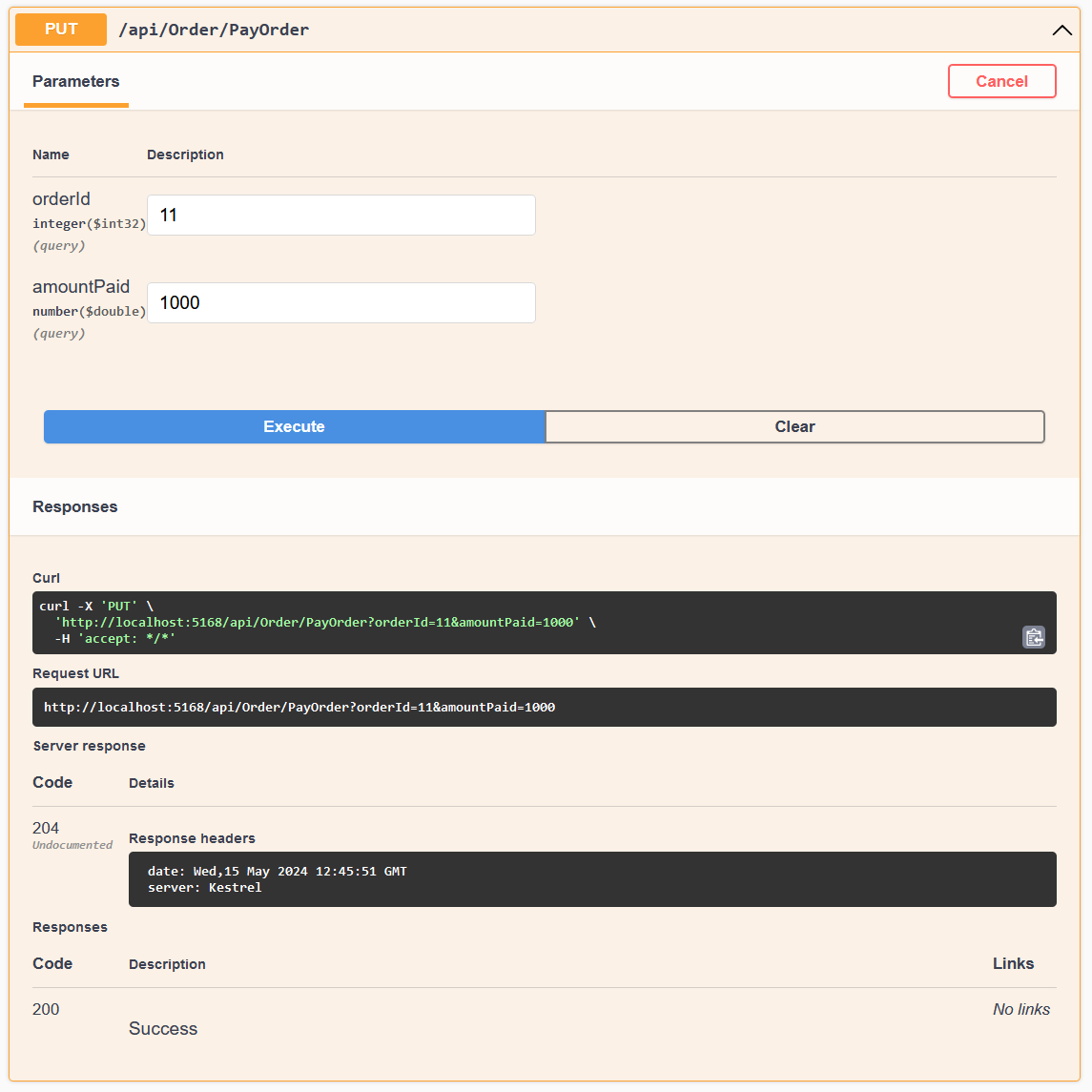


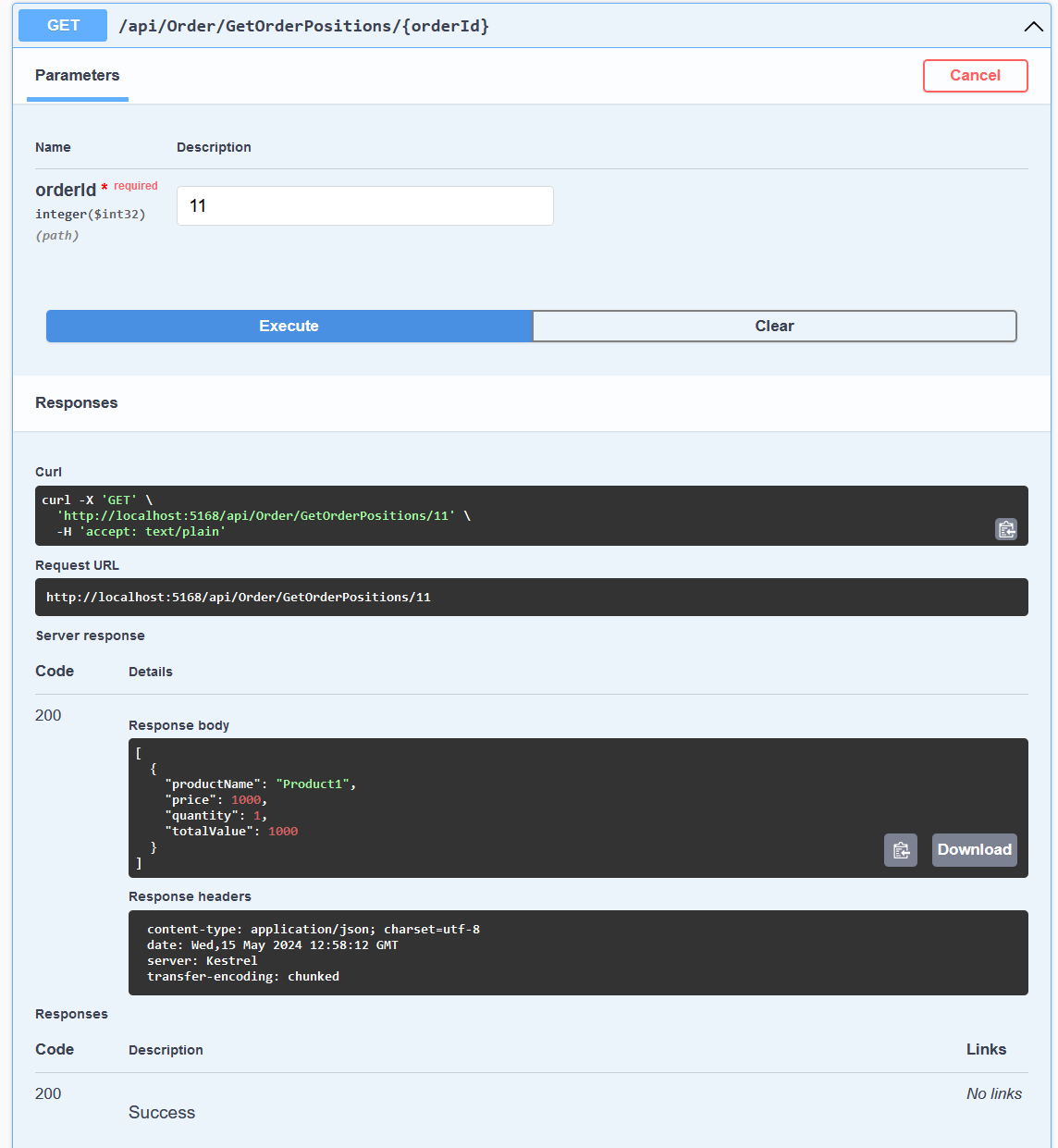


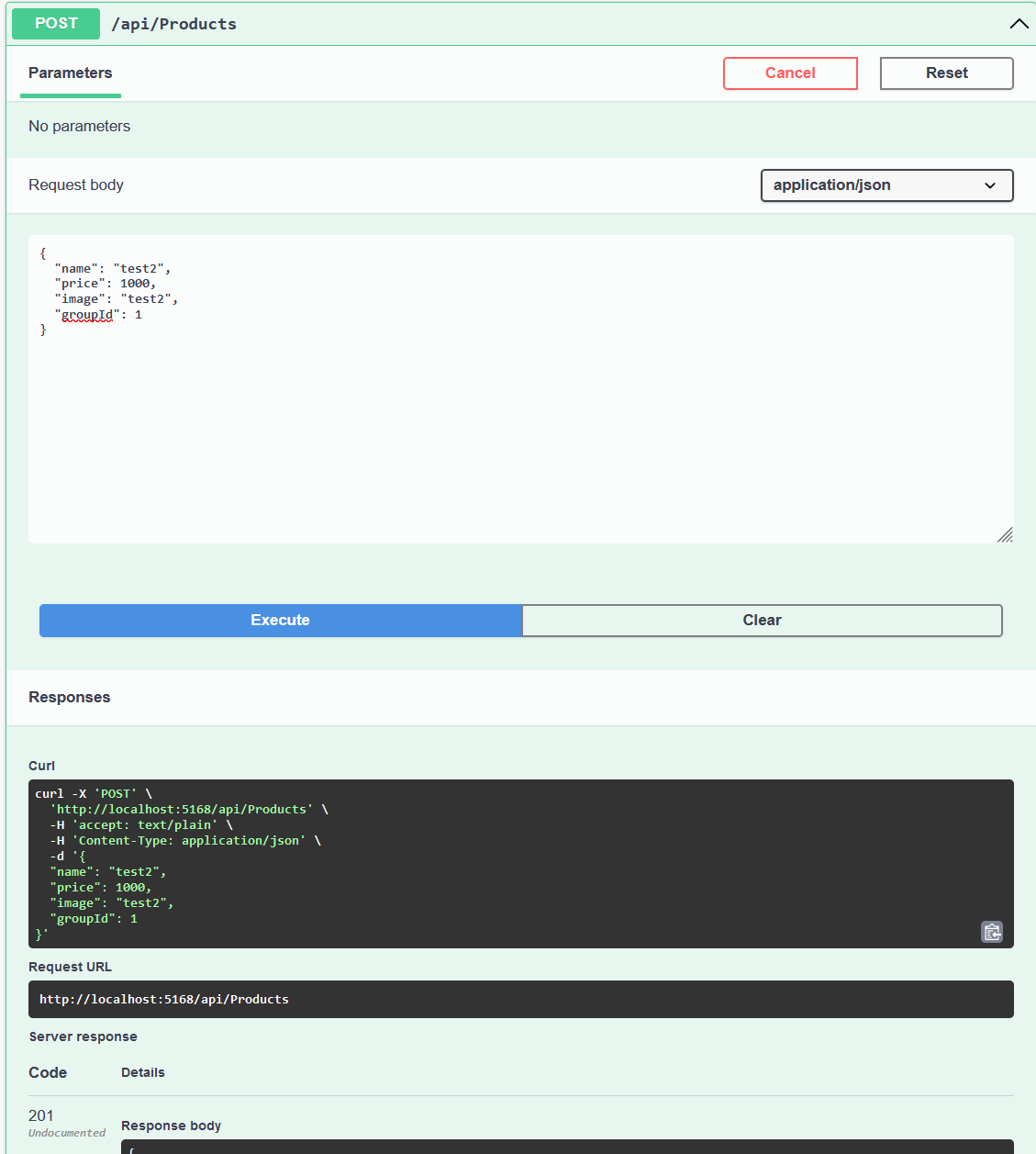
Basket  
  
  
  
  
  
Order

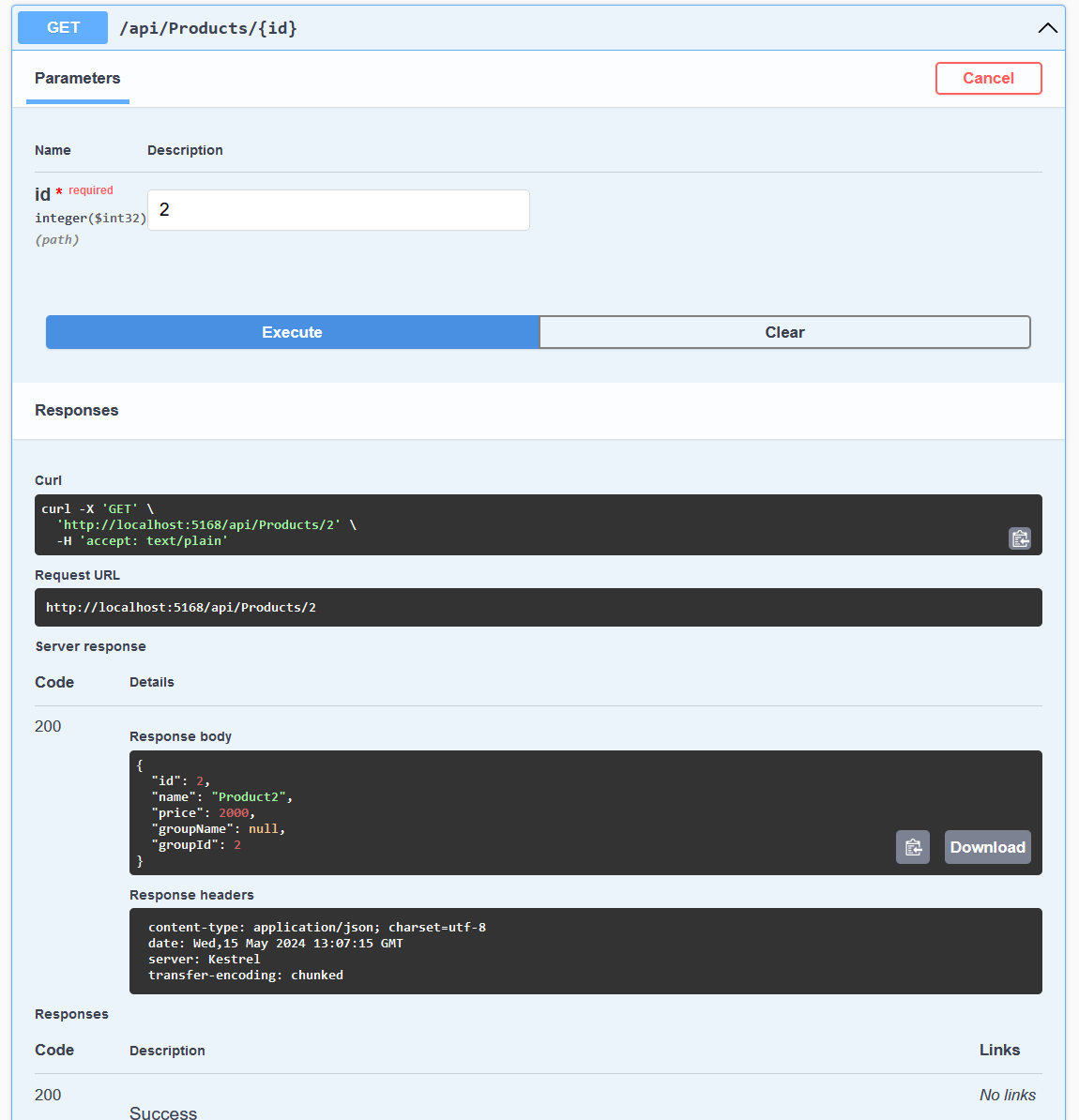
  
  
  
  
  
DB  
Basket  
  


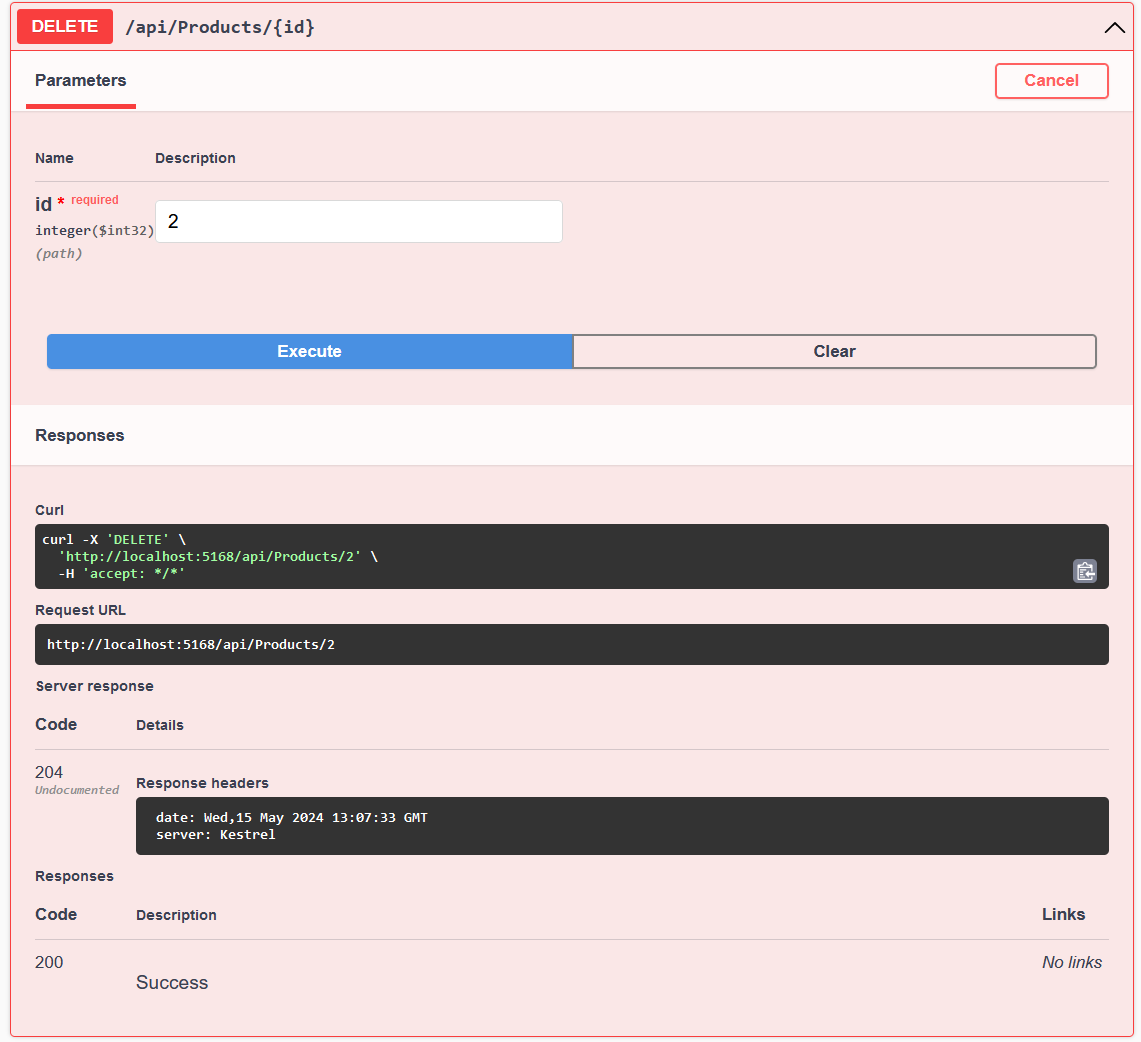


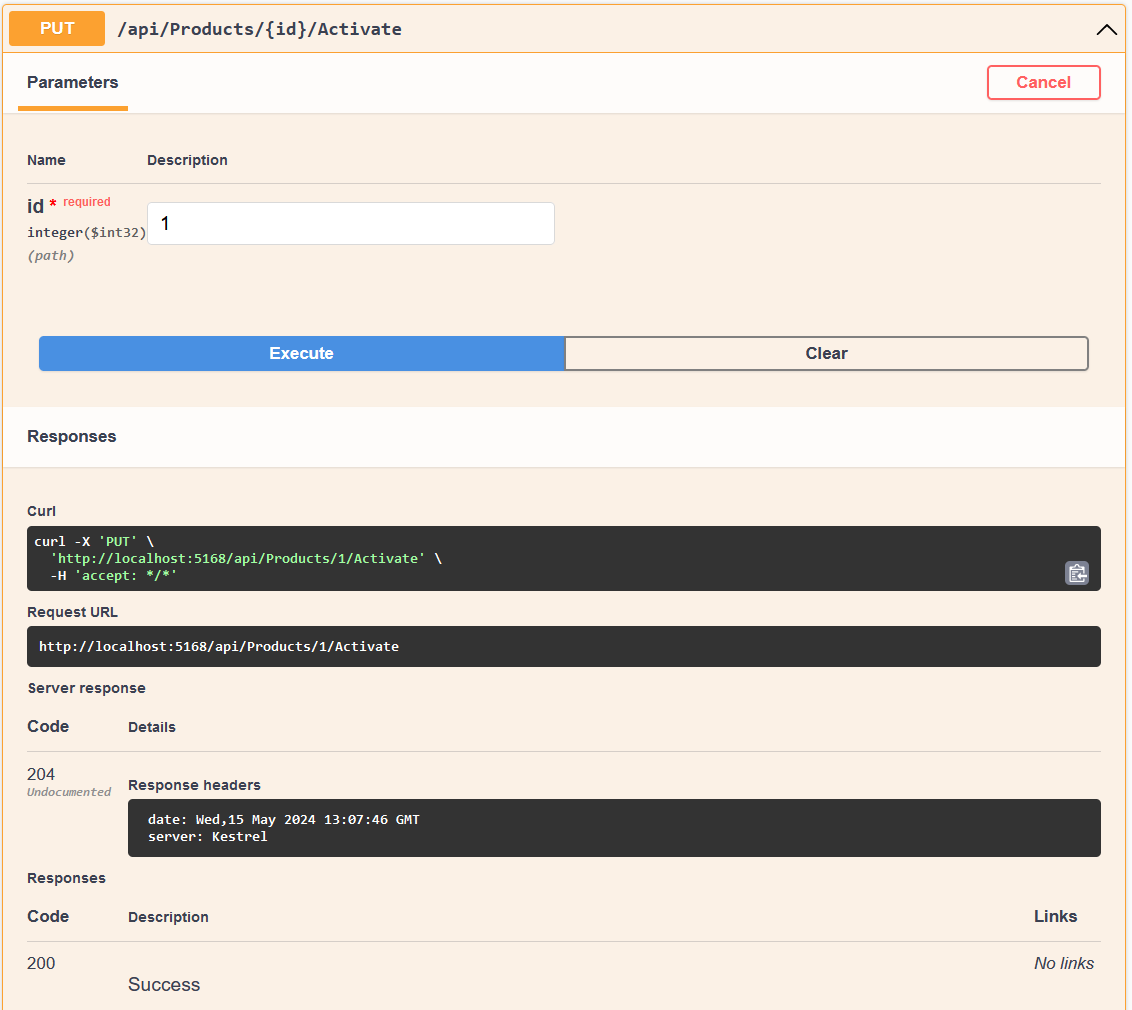
Order  
  


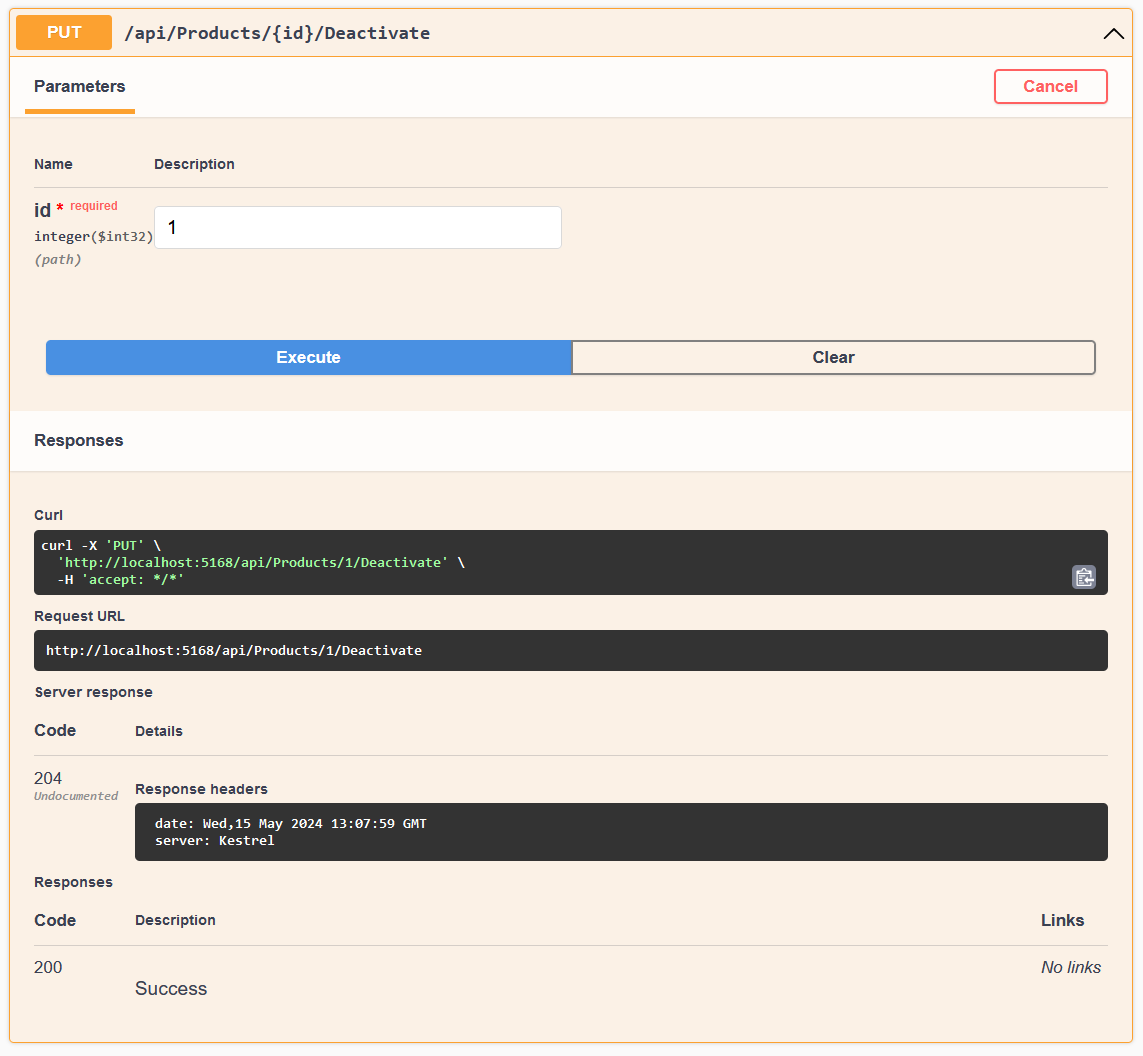










  
  
Porównanie:  
  
EF oferuje wyższy poziom abstrakcji i automatyzacji, co przyspiesza rozwój i zmniejsza ryzyko błędów. Jest idealny dla aplikacji, gdzie czas wdrożenia i łatwość utrzymania są kluczowe. Z drugiej strony, metoda DB zapewnia pełną kontrolę nad zapytaniami SQL i potencjalnie lepszą wydajność, co jest istotne w przypadku aplikacji o dużych wymaganiach wydajnościowych lub specyficznych potrzebach bazy danych. Wybór pomiędzy EF a ADO.NET zależy od specyficznych wymagań projektu, zasobów zespołu i priorytetów związanych z wydajnością, łatwością utrzymania i elastycznością.

SQL:  
  
CREATE PROCEDURE ActivateProduct

@ProductId INT

AS

BEGIN

UPDATE Products

SET IsActive = 1

WHERE Id = @ProductId;

END  
  
CREATE PROCEDURE dbo.AddProduct

@Name NVARCHAR(100),

@Price DECIMAL(18, 2),

@Image NVARCHAR(255),

@GroupId INT

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO Products (Name, Price, Image, GroupId, IsActive)

VALUES (@Name, @Price, @Image, @GroupId, 1);

END;  
  
CREATE PROCEDURE AddProductToBasket

@ProductId INT,

@UserId INT

AS

BEGIN

DECLARE @IsActive BIT;

-- Sprawdź, czy produkt jest aktywny

SELECT @IsActive = IsActive

FROM Products

WHERE Id = @ProductId;

IF @IsActive = 0

BEGIN

RAISERROR('Product is not active and cannot be added to the basket.', 16, 1);

RETURN;

END

INSERT INTO BasketPositions (ProductID, UserID, Amount)

VALUES (@ProductId, @UserId, 1);

END

CREATE PROCEDURE ChangeBasketPositionQuantity

@ProductId INT,

@UserId INT,

@NewQuantity INT

AS

BEGIN

IF @NewQuantity <= 0

BEGIN

RAISERROR('Quantity must be greater than 0.', 16, 1);

RETURN;

END

UPDATE BasketPositions

SET Amount = @NewQuantity

WHERE ProductId = @ProductId and UserId = @UserId;

END

CREATE PROCEDURE DeactivateProduct

@ProductId INT

AS

BEGIN

IF EXISTS (

SELECT 1

FROM OrderPositions od

INNER JOIN Orders o ON od.OrderId = o.Id

WHERE od.ProductId = @ProductId AND o.IsPayed = 0

)

BEGIN

RAISERROR('Product is associated with unpaid orders and cannot be deactivated.', 16, 1);

RETURN;

END

UPDATE Products

SET IsActive = 0

WHERE Id = @ProductId;

END

CREATE PROCEDURE GenerateOrderFromBasket

@UserId INT

AS

BEGIN

DECLARE @NewOrderId INT;

INSERT INTO Orders (UserId, DateTime, IsPayed)

VALUES (@UserId, GETDATE(), 0);

SET @NewOrderId = IDENT\_CURRENT('Orders')

INSERT INTO dbo.OrderPositions(OrderId, ProductId, Price, Amount)

SELECT OrderId = @NewOrderId, ProductId, p.Price, Amount

FROM BasketPositions inner join Products p on p.Id = ProductId

WHERE UserId = @UserId

DELETE FROM BasketPositions

WHERE UserId = @UserId;

Select \* from dbo.OrderPositions where OrderId = @NewOrderId

END

CREATE PROCEDURE GetOrderPositions

@OrderId INT

AS

BEGIN

SELECT

p.Name AS ProductName,

op.Price,

op.Amount AS Quantity,

(op.Price \* op.Amount) AS TotalValue

FROM

OrderPositions op

INNER JOIN

Products p ON op.ProductId = p.Id

WHERE

op.OrderId = @OrderId;

END

GO

CREATE PROCEDURE dbo.GetOrders

AS

BEGIN

SELECT o.Id AS OrderId, o.UserId, o.DateTime AS Date,

CAST(SUM(op.Amount \* op.Price) AS DECIMAL(18, 2)) AS Total, o.isPayed AS Paid

FROM Orders o

JOIN OrderPositions op ON o.Id = op.OrderId

GROUP BY o.Id, o.UserId, o.DateTime, o.isPayed;

END

CREATE PROCEDURE dbo.GetProducts

@IncludeInactive BIT

AS

BEGIN

SELECT

Id,

Name,

Price,

GroupId

FROM

Products

WHERE

IsActive = 1 OR @IncludeInactive = 1

END

CREATE PROCEDURE PayOrder

@OrderId INT,

@AmountPaid DECIMAL(18, 2)

AS

BEGIN

DECLARE @OrderTotal DECIMAL(18, 2);

IF EXISTS (

SELECT 1

FROM Orders

WHERE ID = @OrderId AND IsPayed = 0

)

BEGIN

SELECT @OrderTotal = SUM(op.Amount \* op.Price)

FROM OrderPositions op

WHERE op.OrderID = @OrderId;

IF @AmountPaid = @OrderTotal

BEGIN

UPDATE Orders

SET IsPayed = 1

WHERE ID = @OrderId;

END

ELSE

BEGIN

RAISERROR('Paid amount does not match the total amount of the order.', 16, 1);

RETURN;

END

END

ELSE

BEGIN

RAISERROR('Order is already paid or does not exist.', 16, 1);

RETURN;

END

END

CREATE PROCEDURE RemoveProductFromBasket

@UserId INT,

@ProductId INT

AS

BEGIN

DELETE FROM BasketPositions

WHERE ProductId = @ProductId and UserId = @UserId;

END