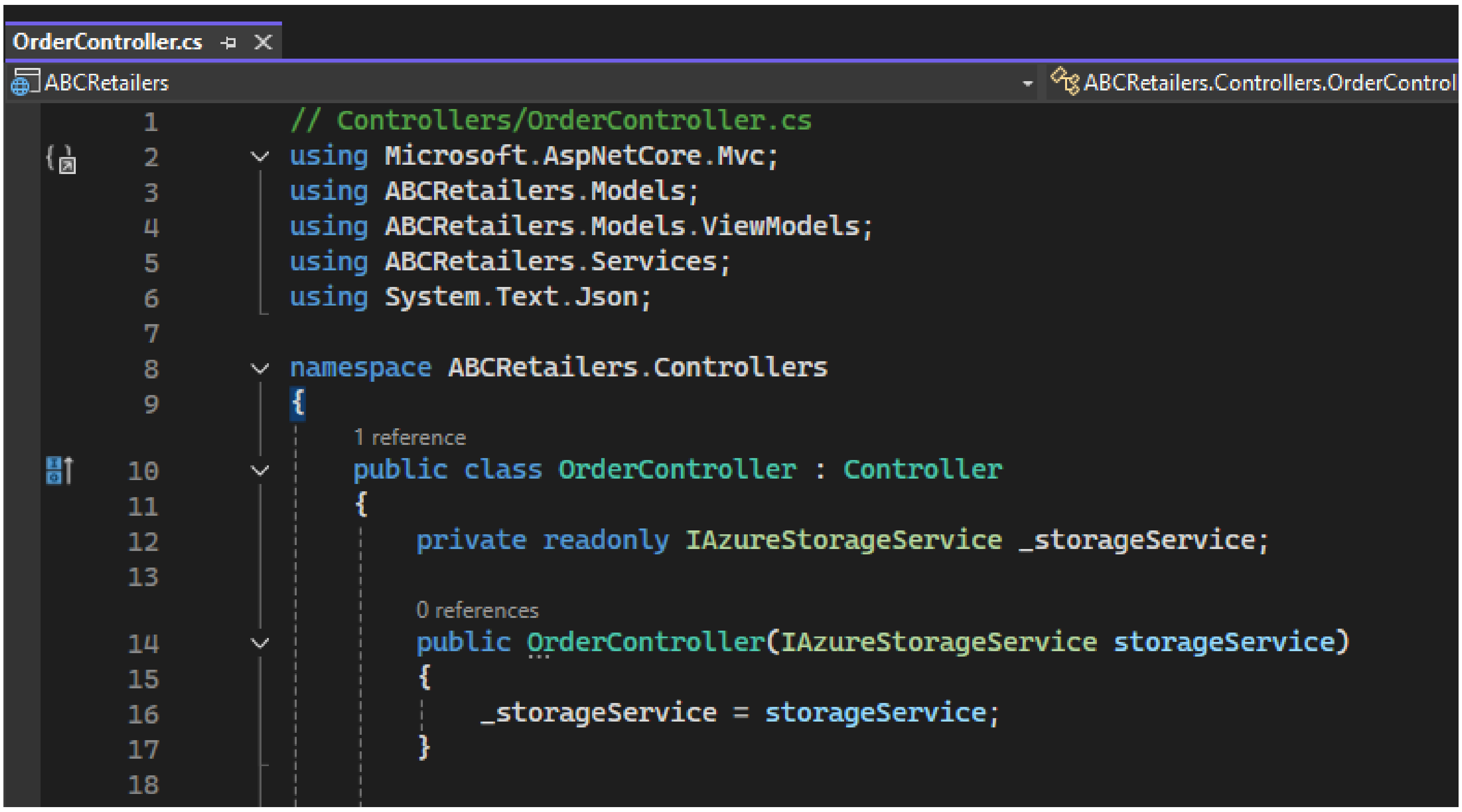
ABCRetailers Web Application - OrderController Code Snapshots

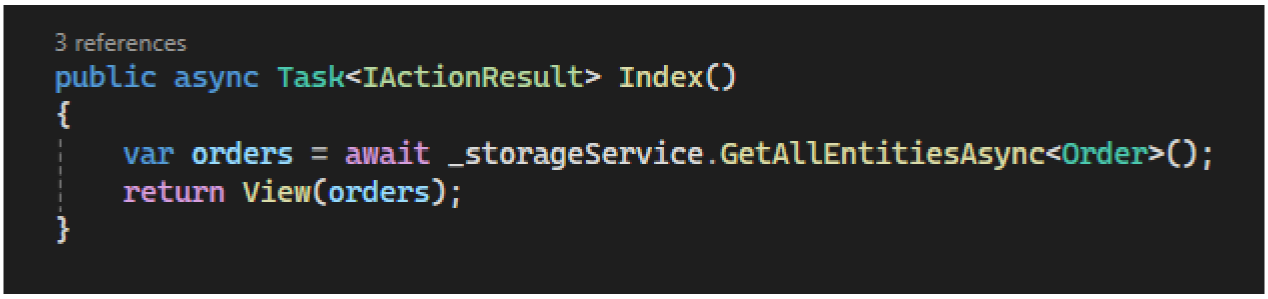
Difficult to take Full Code Snapshot – Watch Video Recording for Start and End of Code – The Start and End of Code is also captured in the First and Last Snapshot Respectively)

OrderController.cs (IAzureStorageService Action Code Snapshot – NOTE THAT THE PREAMBLE PART OF THE CODE IS CAPTURED HERE)



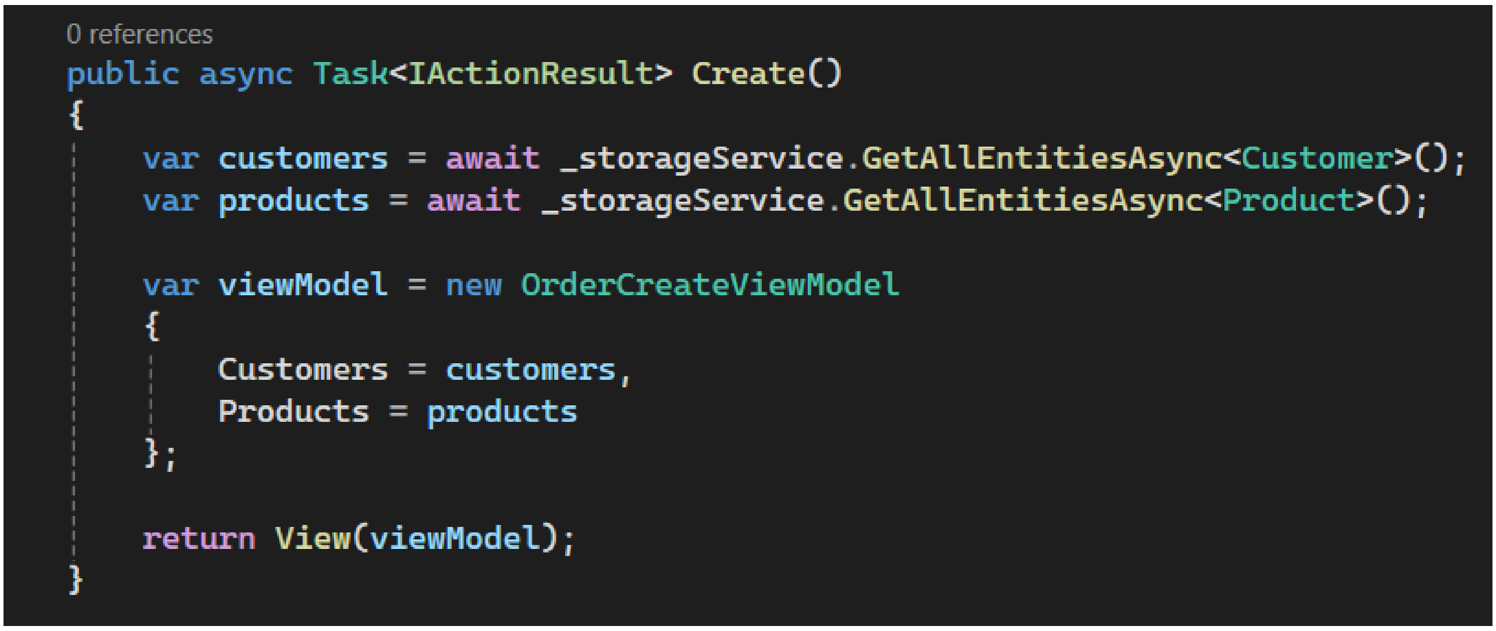
```
// Controllers/OrderController.cs
using Microsoft.AspNetCore.Mvc;
using ABCRetailers.Models;
using ABCRetailers.Models.ViewModels;
using ABCRetailers.Services;
using System.Text.Json;

namespace ABCRetailers.Controllers
{
    public class OrderController : Controller
    {
        private readonly IAzureStorageService _storageService;

        public OrderController(IAzureStorageService storageService)
        {
            _storageService = storageService;
        }

```

OrderController.cs (Index Action Code Snapshot)



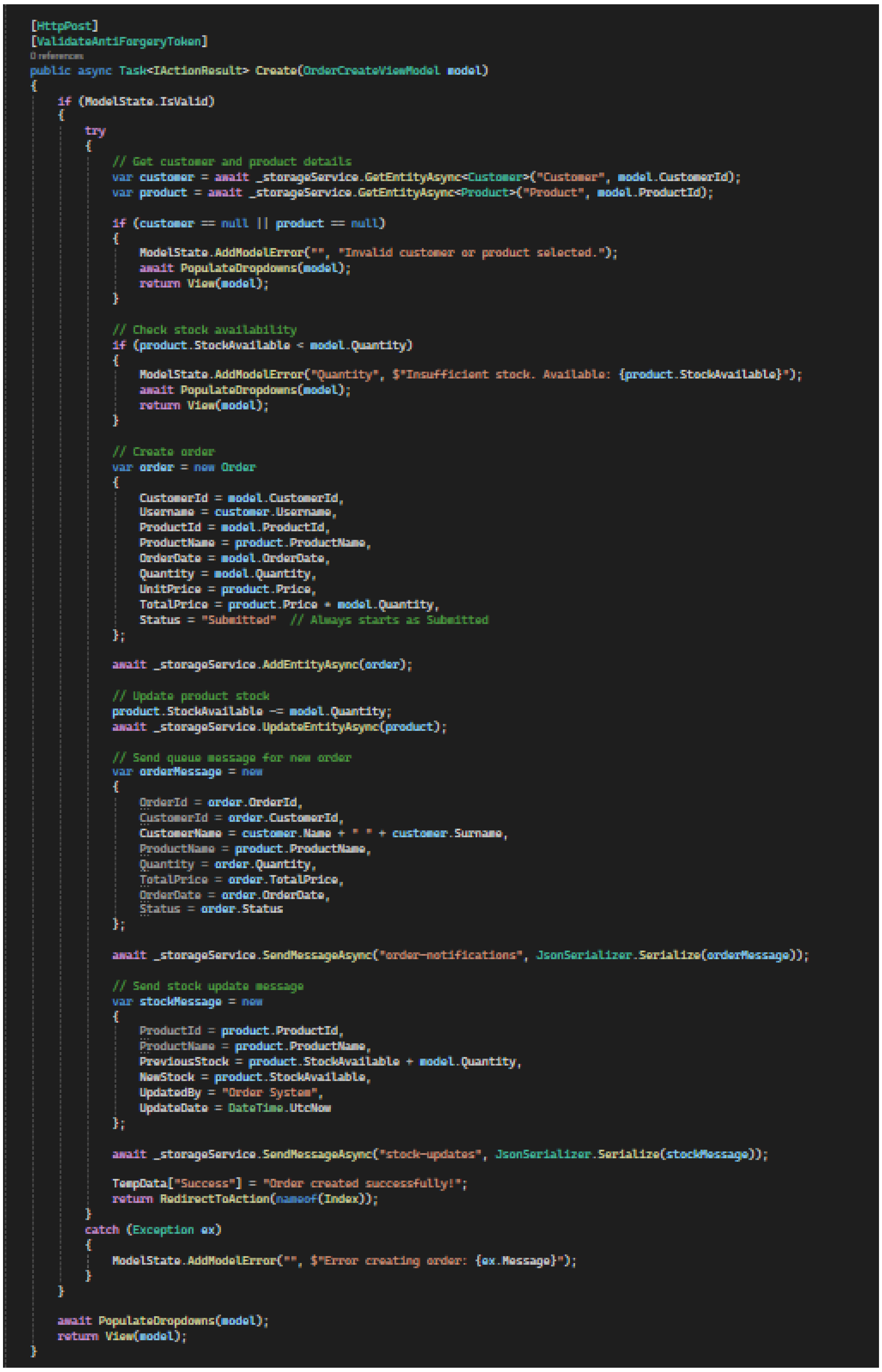
```
public async Task<IActionResult> Index()
{
    var orders = await _storageService.GetAllEntitiesAsync<Order>();
    return View(orders);
}
```

OrderController.cs (Create GET Action Code Snapshot)



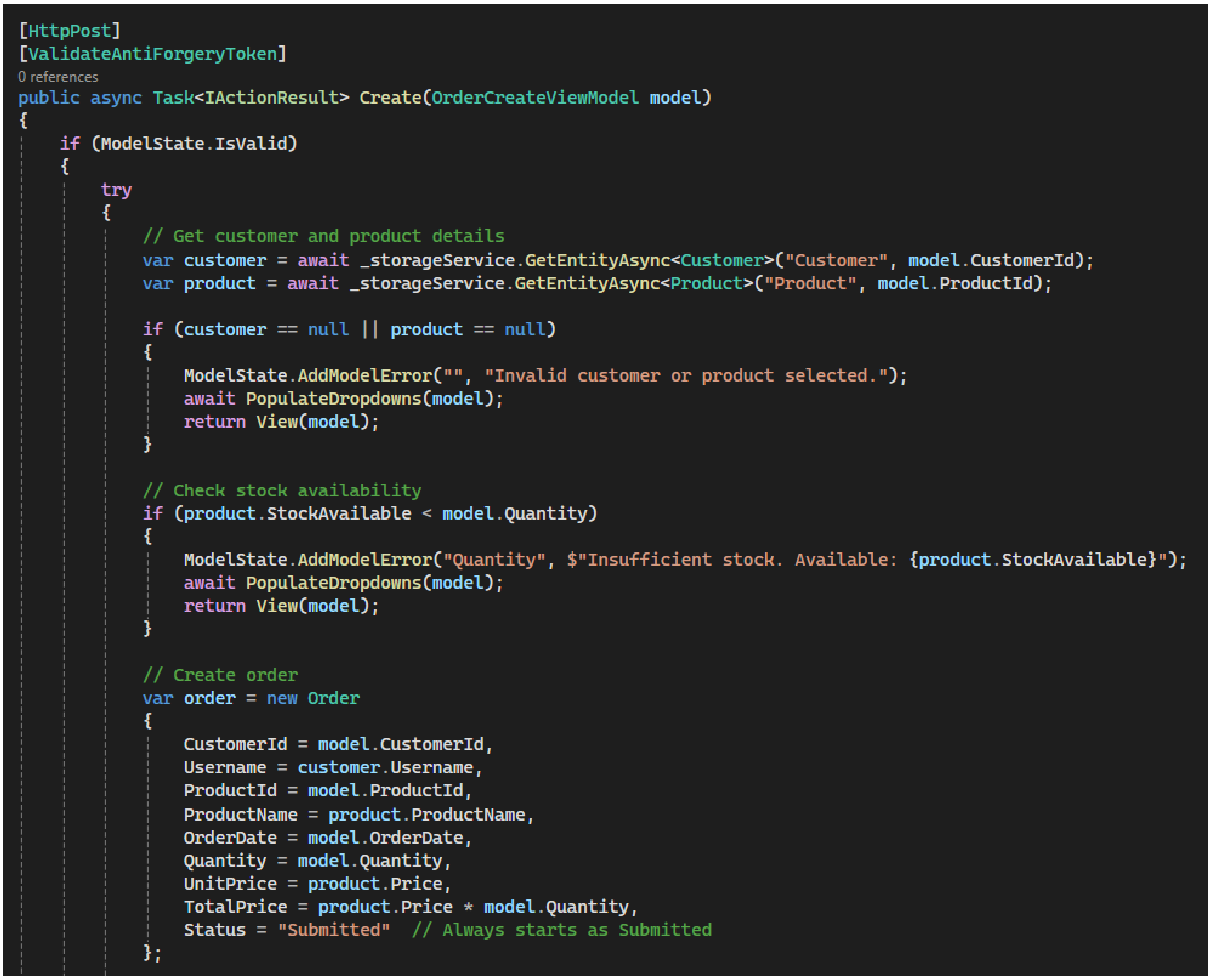
```
0 references
public async Task<IActionResult> Create()
{
    var customers = await _storageService.GetAllEntitiesAsync<Customer>();
    var products = await _storageService.GetAllEntitiesAsync<Product>();

    var viewModel = new OrderCreateViewModel
    {
        Customers = customers,
        Products = products
    };

    return View(viewModel);
}
```

OrderController.cs (Create POST Action Full Code Snapshot)



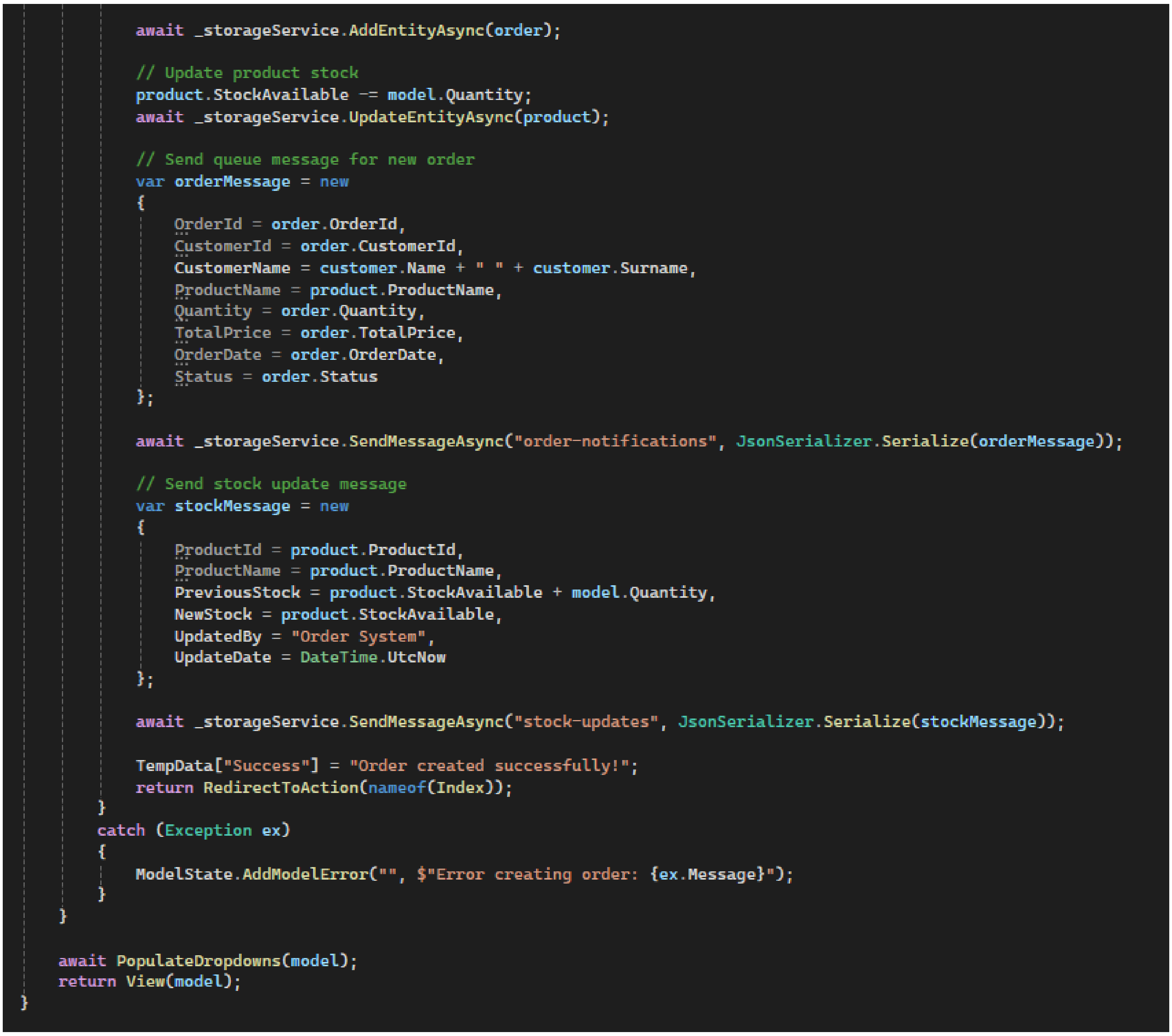
```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create(OrderCreateViewModel model)
{
    if (ModelState.IsValid)
    {
        try
        {
            // Get customer and product details
            var customer = await _storageService.GetEntityAsync<Customer>("Customer", model.CustomerId);
            var product = await _storageService.GetEntityAsync<Product>("Product", model.ProductId);

            if (customer == null || product == null)
            {
                ModelState.AddModelError("", "Invalid customer or product selected.");
                await PopulateDropdowns(model);
                return View(model);
            }

            // Check stock availability
            if (product.StockAvailable < model.Quantity)
            {
                ModelState.AddModelError("Quantity", $"Insufficient stock. Available: {product.StockAvailable}");
                await PopulateDropdowns(model);
                return View(model);
            }

            // Create order
            var order = new Order
            {
                CustomerId = model.CustomerId,
                Username = customer.Username,
                ProductId = model.ProductId,
                ProductName = product.ProductName,
                OrderDate = model.OrderDate,
                Quantity = model.Quantity,
                UnitPrice = product.Price,
                TotalPrice = product.Price * model.Quantity,
                Status = "Submitted"  // Always starts as Submitted
            };

            await _storageService.AddEntityAsync(order);

            // Update product stock
            product.StockAvailable -= model.Quantity;
            await _storageService.UpdateEntityAsync(product);

            // Send queue message for new order
            var orderMessage = new
            {
                OrderId = order.OrderId,
                CustomerId = order.CustomerId,
                CustomerName = customer.Name + " " + customer.Surname,
                ProductName = product.ProductName,
                Quantity = order.Quantity,
                TotalPrice = order.TotalPrice,
                OrderDate = order.OrderDate,
                Status = order.Status
            };

            await _storageService.SendMessageAsync("order-notifications", JsonSerializer.Serialize(orderMessage));

            // Send stock update message
            var stockMessage = new
            {
                ProductId = product.ProductId,
                ProductName = product.ProductName,
                PreviousStock = product.StockAvailable + model.Quantity,
                NewStock = product.StockAvailable,
                UpdatedBy = "Order System",
                UpdateDate = DateTime.UtcNow
            };

            await _storageService.SendMessageAsync("stock-updates", JsonSerializer.Serialize(stockMessage));

            TempData["Success"] = "Order created successfully!";
            return RedirectToAction(nameof(Index));
        }
        catch (Exception ex)
        {
            ModelState.AddModelError("", $"Error creating order: {ex.Message}");
        }
    }

    await PopulateDropdowns(model);
    return View(model);
}
```

OrderController.cs (Create POST Action Splitted Code Snapshot – v1)



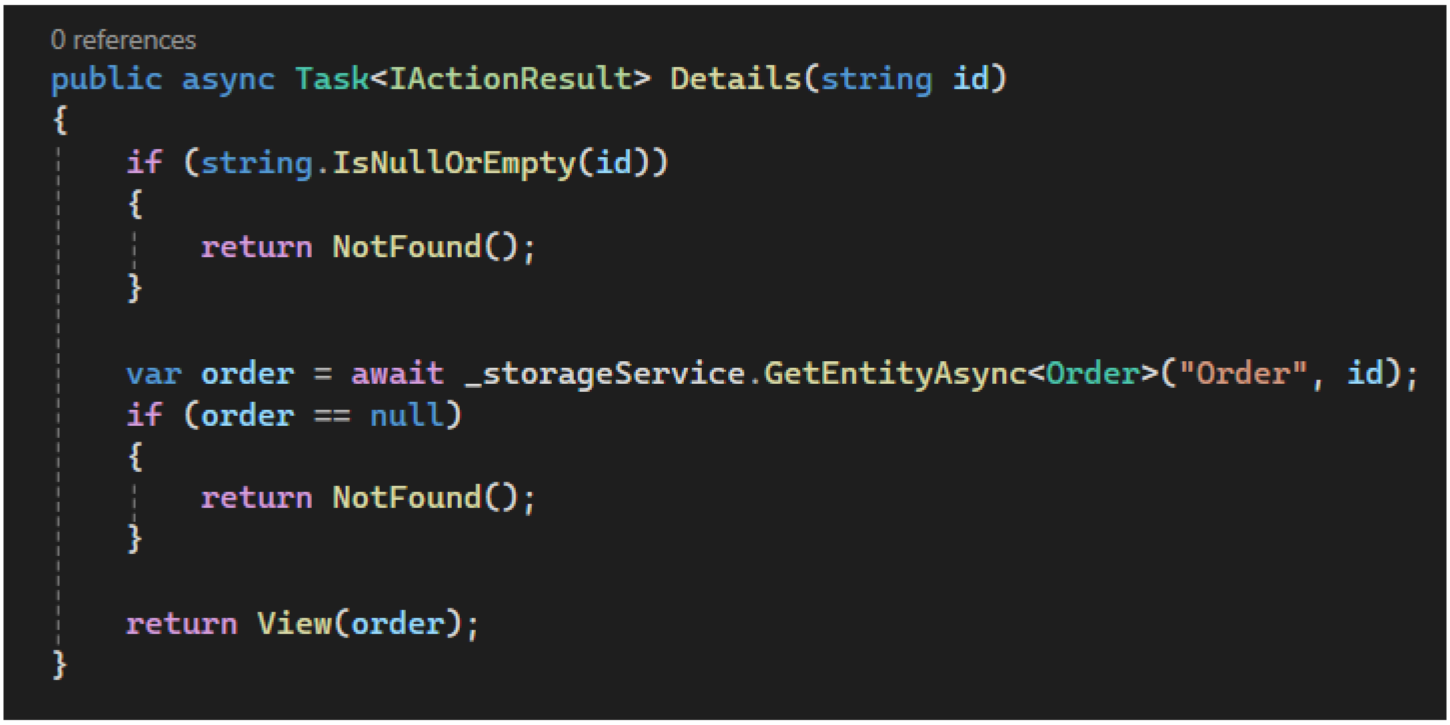
```
[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Create(OrderCreateViewModel model)
{
    if (ModelState.IsValid)
    {
        try
        {
            // Get customer and product details
            var customer = await _storageService.GetEntityAsync<Customer>("Customer", model.CustomerId);
            var product = await _storageService.GetEntityAsync<Product>("Product", model.ProductId);

            if (customer == null || product == null)
            {
                ModelState.AddModelError("", "Invalid customer or product selected.");
                await PopulateDropdowns(model);
                return View(model);
            }

            // Check stock availability
            if (product.StockAvailable < model.Quantity)
            {
                ModelState.AddModelError("Quantity", $"Insufficient stock. Available: {product.StockAvailable}");
                await PopulateDropdowns(model);
                return View(model);
            }

            // Create order
            var order = new Order
            {
                CustomerId = model.CustomerId,
                Username = customer.Username,
                ProductId = model.ProductId,
                ProductName = product.ProductName,
                OrderDate = model.OrderDate,
                Quantity = model.Quantity,
                UnitPrice = product.Price,
                TotalPrice = product.Price * model.Quantity,
                Status = "Submitted"  // Always starts as Submitted
            };
```

OrderController.cs (Create POST Action Splitted Code Snapshot – v2)



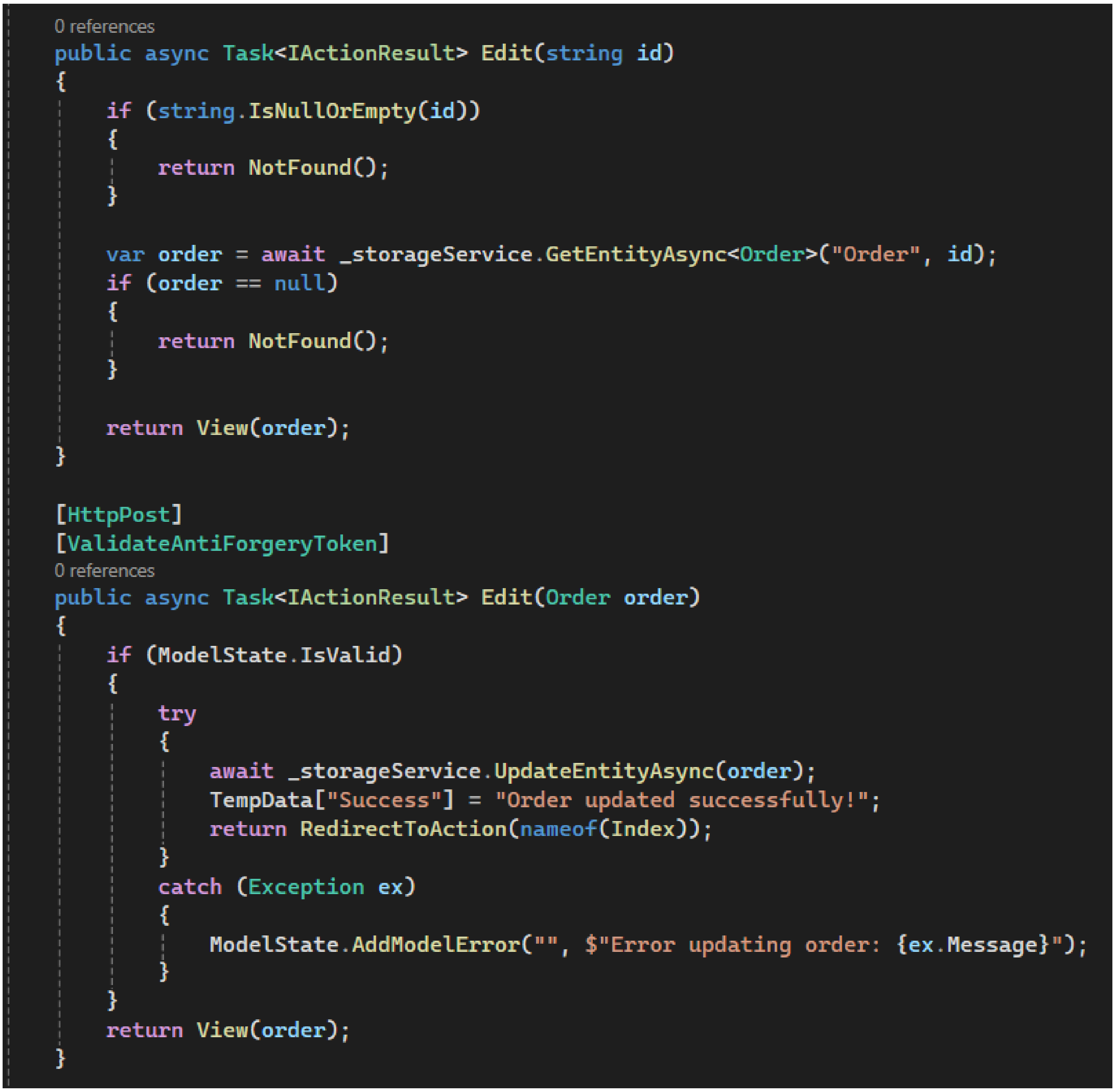
```
                await _storageService.AddEntityAsync(order);

                // Update product stock
                product.StockAvailable -= model.Quantity;
                await _storageService.UpdateEntityAsync(product);

                // Send queue message for new order
                var orderMessage = new
                {
                    OrderId = order.OrderId,
                    CustomerId = order.CustomerId,
                    CustomerName = customer.Name + " " + customer.Surname,
                    ProductName = product.ProductName,
                    Quantity = order.Quantity,
                    TotalPrice = order.TotalPrice,
                    OrderDate = order.OrderDate,
                    Status = order.Status
                };

                await _storageService.SendMessageAsync("order-notifications", JsonSerializer.Serialize(orderMessage));

                // Send stock update message
                var stockMessage = new
                {
                    ProductId = product.ProductId,
                    ProductName = product.ProductName,
                    PreviousStock = product.StockAvailable + model.Quantity,
                    NewStock = product.StockAvailable,
                    UpdatedBy = "Order System",
                    UpdateDate = DateTime.UtcNow
                };

                await _storageService.SendMessageAsync("stock-updates", JsonSerializer.Serialize(stockMessage));

                TempData["Success"] = "Order created successfully!";
                return RedirectToAction(nameof(Index));
            }
            catch (Exception ex)
            {
                ModelState.AddModelError("", $"Error creating order: {ex.Message}");
            }
        }

        await PopulateDropdowns(model);
        return View(model);
    }
```

OrderController.cs (Details Action Code Snapshot)



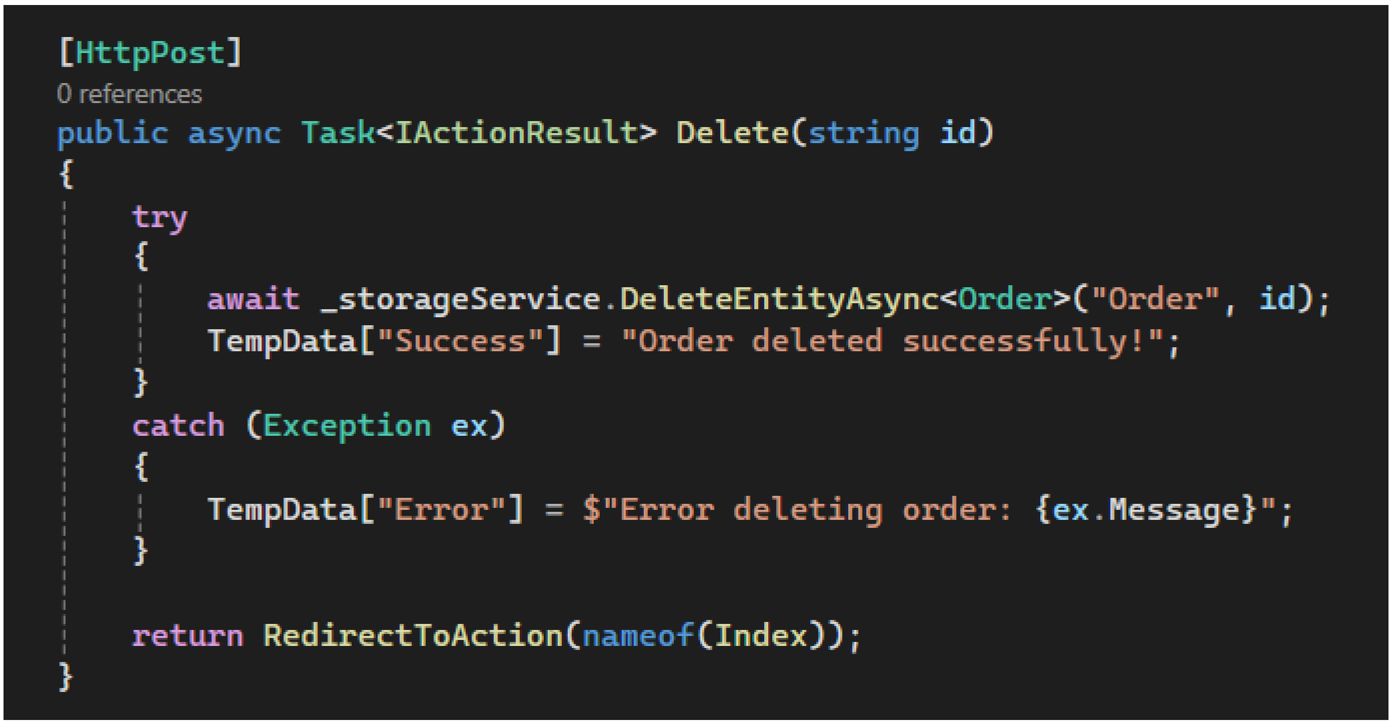
```
0 references
public async Task<IActionResult> Details(string id)
{
    if (string.IsNullOrEmpty(id))
    {
        return NotFound();
    }

    var order = await _storageService.GetEntityAsync<Order>("Order", id);
    if (order == null)
    {
        return NotFound();
    }

    return View(order);
}
```

OrderController.cs (Edit Action Code Snapshot)



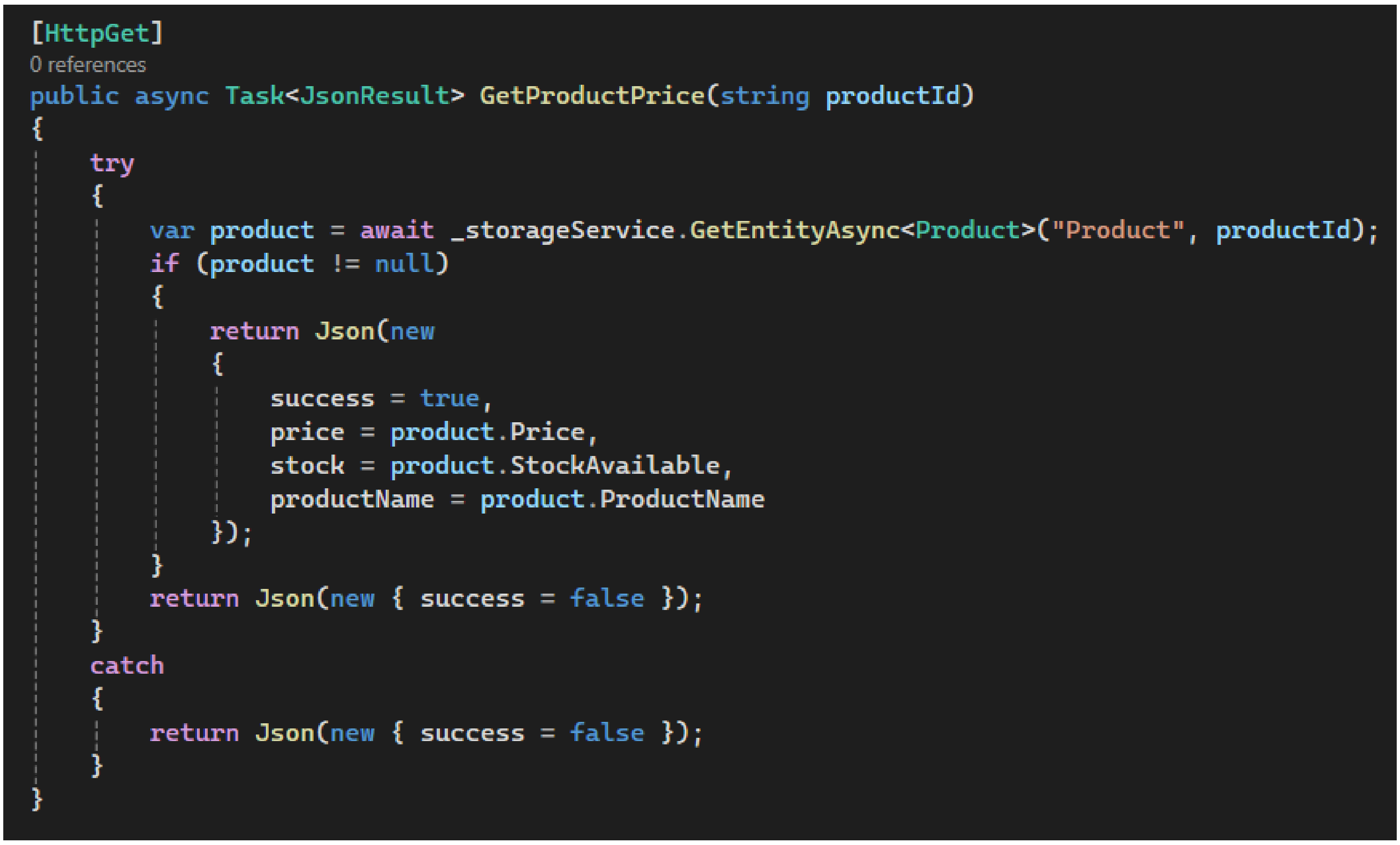
```
0 references
public async Task<IActionResult> Edit(string id)
{
    if (string.IsNullOrEmpty(id))
    {
        return NotFound();
    }

    var order = await _storageService.GetEntityAsync<Order>("Order", id);
    if (order == null)
    {
        return NotFound();
    }

    return View(order);
}

[HttpPost]
[ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Edit(Order order)
{
    if (ModelState.IsValid)
    {
        try
        {
            await _storageService.UpdateEntityAsync(order);
            TempData["Success"] = "Order updated successfully!";
            return RedirectToAction(nameof(Index));
        }
        catch (Exception ex)
        {
            ModelState.AddModelError("", $"Error updating order: {ex.Message}");
        }
    }
    return View(order);
}
```

OrderController.cs (Delete Action Code Snapshot)



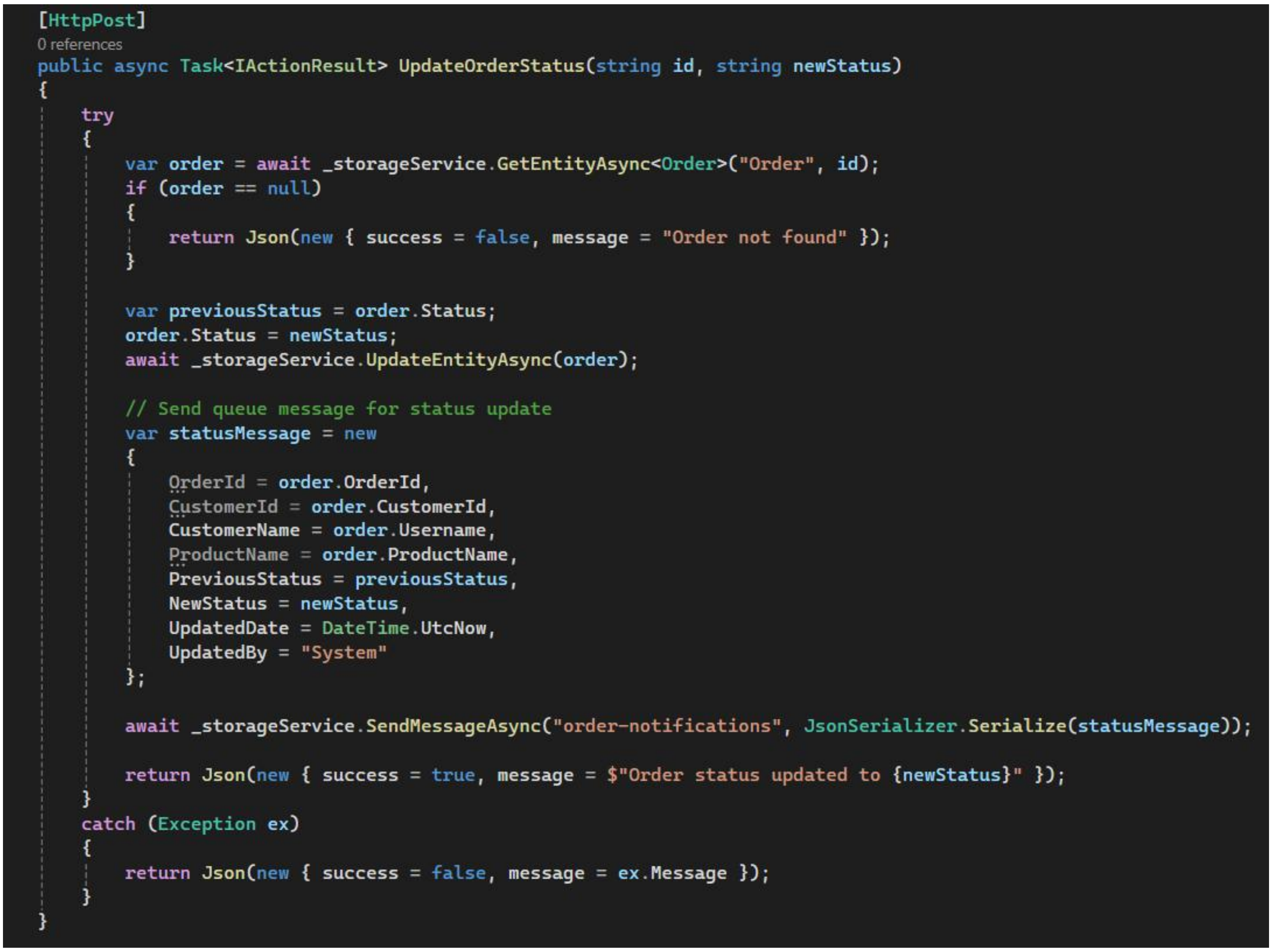
```
[HttpPost]
0 references
public async Task<IActionResult> Delete(string id)
{
    try
    {
        await _storageService.DeleteEntityAsync<Order>("Order", id);
        TempData["Success"] = "Order deleted successfully!";
    }
    catch (Exception ex)
    {
        TempData["Error"] = $"Error deleting order: {ex.Message}";
    }

    return RedirectToAction(nameof(Index));
}
```

OrderController.cs (GetProductPrice Action Code Snapshot)



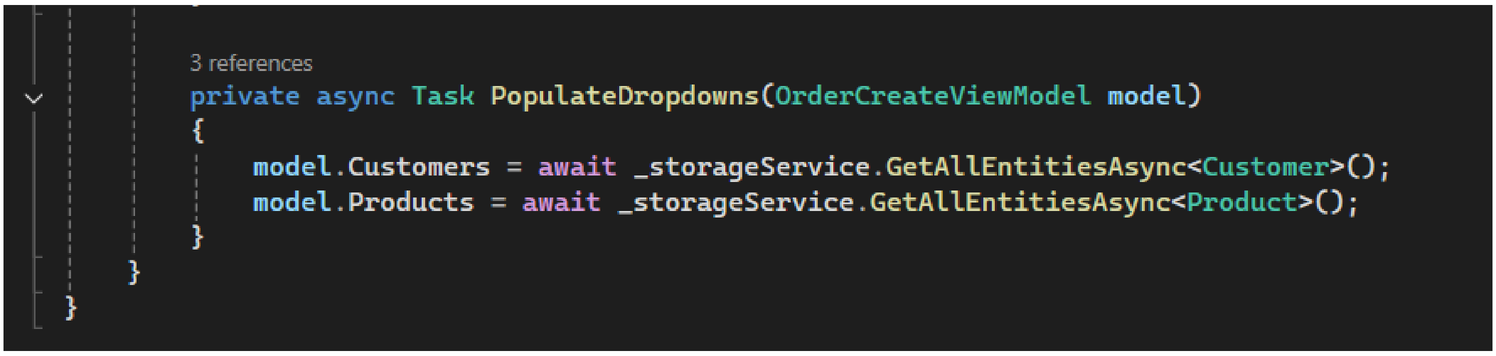
```
[HttpGet]
0 references
public async Task<JsonResult> GetProductPrice(string productId)
{
    try
    {
        var product = await _storageService.GetEntityAsync<Product>("Product", productId);
        if (product != null)
        {
            return Json(new
            {
                success = true,
                price = product.Price,
                stock = product.StockAvailable,
                productName = product.ProductName
            });
        }
        return Json(new { success = false });
    }
    catch
    {
        return Json(new { success = false });
    }
}
```

OrderController.cs (UploadOrderStatus Action Code Snapshot)



```
[HttpPost]
0 references
public async Task<IActionResult> UpdateOrderStatus(string id, string newStatus)
{
    try
    {
        var order = await _storageService.GetEntityAsync<Order>("Order", id);
        if (order == null)
        {
            return Json(new { success = false, message = "Order not found" });
        }

        var previousStatus = order.Status;
        order.Status = newStatus;
        await _storageService.UpdateEntityAsync(order);

        // Send queue message for status update
        var statusMessage = new
        {
            OrderId = order.OrderId,
            CustomerId = order.CustomerId,
            CustomerName = order.Username,
            ProductName = order.ProductName,
            PreviousStatus = previousStatus,
            NewStatus = newStatus,
            UpdatedDate = DateTime.UtcNow,
            UpdatedBy = "System"
        };

        await _storageService.SendMessageAsync("order-notifications", JsonSerializer.Serialize(statusMessage));

        return Json(new { success = true, message = $"Order status updated to {newStatus}" });
    }
    catch (Exception ex)
    {
        return Json(new { success = false, message = ex.Message });
    }
}
```

OrderController.cs (PopulateDropdowns Action Code Snapshot – NOTE THESE PART OF THE CODE CAPTURES THE TWO CLOSING BRACES)



```
        3 references
        private async Task PopulateDropdowns(OrderCreateViewModel model)
        {
            model.Customers = await _storageService.GetAllEntitiesAsync<Customer>();
            model.Products = await _storageService.GetAllEntitiesAsync<Product>();
        }
    }
}
```