**LondonStockAPI – Technical Documentation**

**Contents :**

* Overview
* Architecture Changes
* Detailed Feature Enhancements
* Code Changes Summary
* Testing Updates
* Deployment & Configuration Notes
* Future Recommendations

| **Sl. No.** | **Author** | **Date** | **Version** |
| --- | --- | --- | --- |
| 1. | Manoj B N | 14/08/2025 | V1.0 |

**1. Overview**

This document details the latest enhancements made to the **LondonStockAPI** project to improve **error handling**, **logging**, **performance monitoring**, **validation**, and **caching**.  
The changes align the API with **production-grade best practices** for .NET 6+ applications.

**2. Architecture Changes**

The following architectural improvements were implemented:

| **Area** | **Change** | **Benefit** |
| --- | --- | --- |
| Clean Architecture, CQRS, MVP | Followed the standard architectural patterns and design principles through out the application | Helps in Scalability,Improved Maintainability, Enhanced Testability, Performance Optimization, Simplified Complex Domains, Enhanced Testability of UI Logic,Clean Separation of Concerns |
| Exception Handling | Centralized via ExceptionMiddleware returning custom APIResponse | Custom wrapper for success and error responses |
| Logging | Structured logging with **Serilog enrichment** and **logging scopes** | Better traceability and observability |
| Performance Monitoring | Added Stopwatch timing in service and controller methods | Identifies slow operations |
| Validation | Using DataAnnotations | Automatic validation responses for invalid input |
| Caching | Added distributed caching for GetAllTrades and average prices | Reduces DB load, improves response time |
| Health Checks | Added /health endpoint | Enables monitoring systems to check API health |
| RateLimiting | Configured in Program.cs and enabled Configured in Controller Level | To control the rate of requests a system receives, preventing resource exhaustion and ensuring fair usage and to protect against abuse |
| CORS Enablemenet | Configured in Program.cs and enabled in Controller Level | Enabling it allows servers to specify which origins are permitted to access their resources |
| MOQ Testing | Configured in LondonStockAPI.Tests | Created the Test Library to test all the endpoints |
| Swagger UI | Configured in Program.cs | Supports the Swagger UI for Request handling |

**3. Detailed Feature Enhancements**

**3.1 Exception Handling with APIResponse**

* Implemented ExceptionMiddleware to catch unhandled exceptions globally.
* Returns **APIResponse** JSON in application/ json format.
* Example error response:

**{**

**"success": false,**

**"message": "Invalid request payload",**

**"data": [**

**"The Quantity field must be greater than 0."**

**]**

**}**

**3.2 Structured Logging with Serilog**

* Configured **Serilog** with:
  + Machine name
  + Environment name
  + Log context enrichment
  + Console and file sinks

**3.3 Logging Scopes for Correlation IDs**

* Added ILogger.BeginScope in controllers to generate a **unique Correlation ID** per request.
* All logs within the request share the same ID for easy tracing.

**3.4 Performance Logging**

* Added Stopwatch in service and controller methods to log execution time.

**3.6 Caching**

* Implemented **IDistributedCache** for:
  + GetAllTrades (5 min cache)
  + GetAveragePrice (10 min cache)
* Reduces database queries for frequently requested data.

**3.7 Health Checks**

* Added:

app.MapHealthChecks("/health");

* Allows monitoring tools (e.g., Kubernetes, Azure App Service) to check API health.

**4. Code Changes Summary**

**4.1 Controllers**

* TradesController updated to:
  + Use logging scopes
  + Return APIResponse for errors
  + Measure execution time
  + Call services with caching

**4.2 Services**

* TradeCommandService:
  + Throws InvalidOperationException for business rule violations
  + Logs execution time
* TradeQueryService:
  + Uses AsNoTracking for read-only queries
  + Implements caching for performance

**4.3 Middleware**

* ExceptionMiddleware:
  + Logs unhandled exceptions
  + Returns custom error responses

**4.4 Model.IsValid**

* ValidationFilter:
  + Automatically handles model validation errors

**5. Testing Updates**

Unit tests updated to:

* Mock ILogger<TradesController>
* Keep existing ApiResponse checks for success cases

Example updated test:

var badRequest = result.Should().BeOfType<BadRequestObjectResult>().Subject;

var apiResponse = badRequest.Value.Should().BeOfType<ApiResponse<string>>().Subject;

apiResponse.Success.Should().BeFalse();

apiResponse.Message.Should().Be("Invalid trade");

**6. Deployment & Configuration Notes**

* **Serilog** configuration in Program.cs:

Log.Logger = new LoggerConfiguration()

.Enrich.FromLogContext()

.Enrich.WithMachineName()

.Enrich.WithEnvironmentName()

.WriteTo.Console()

.WriteTo.File("logs/log.txt", rollingInterval: RollingInterval.Day)

.CreateLogger();

* **Caching** uses IDistributedCache (in-memory by default, can be switched to Redis in production).
* **Health Checks** available at /health.

**7. Future Recommendations**

* Implement **OpenTelemetry** for distributed tracing.
* Use **FluentValidation** for more complex validation rules.
* Add **integration tests** for middleware and caching behavior.

**8. Appendix – API Response Formats**

**Success Response**

**{**

**"success": true,**

**"message": "Stock value retrieved",**

**"data": {**

**"ticker": "VMW",**

**"averagePrice": "£46.00",**

**"currency": "GBP"**

**}**

**}**

**Error Response**

**{**

**"success": false,**

**"message": "Broker with ID 10 does not exist.",**

**"data": null**

**}**

API Versioning , CORS Limiting & Rate Limiting Fetaures are enabled

| **Feature** | **Apply To Controllers?** | **Apply To Services?** | **Notes** |
| --- | --- | --- | --- |
| API Versioning | Yes | No | HTTP contract concern |
| CORS Limiting | Yes | No | HTTP security concern |
| Rate Limiting | Yes | No | Request throttling concern |

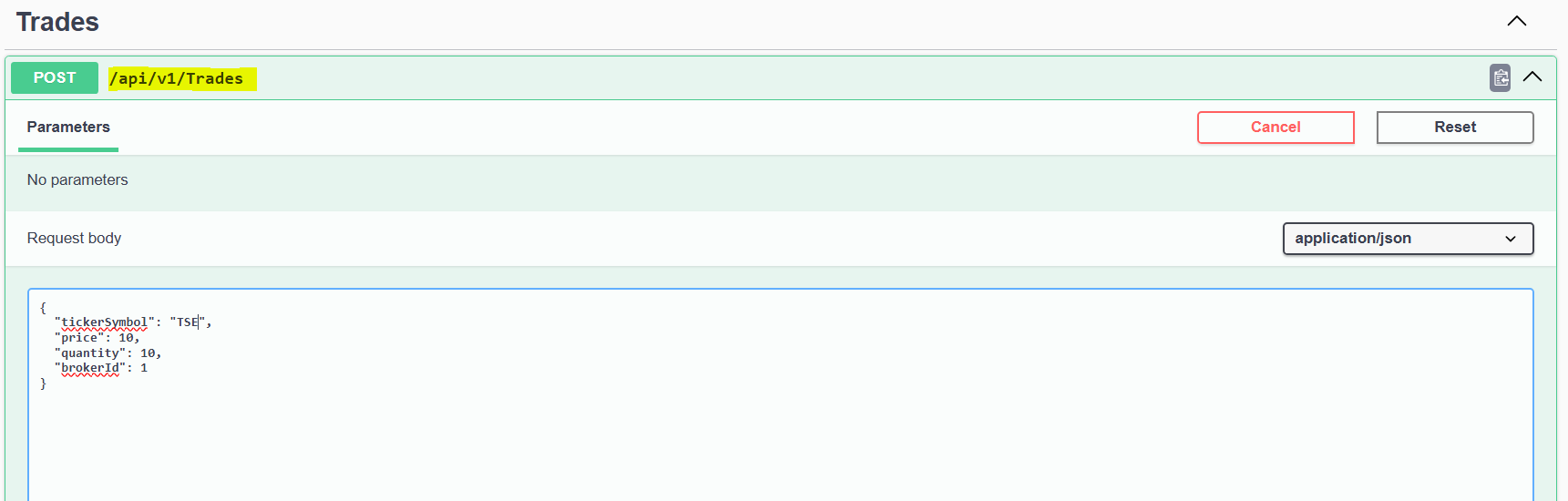
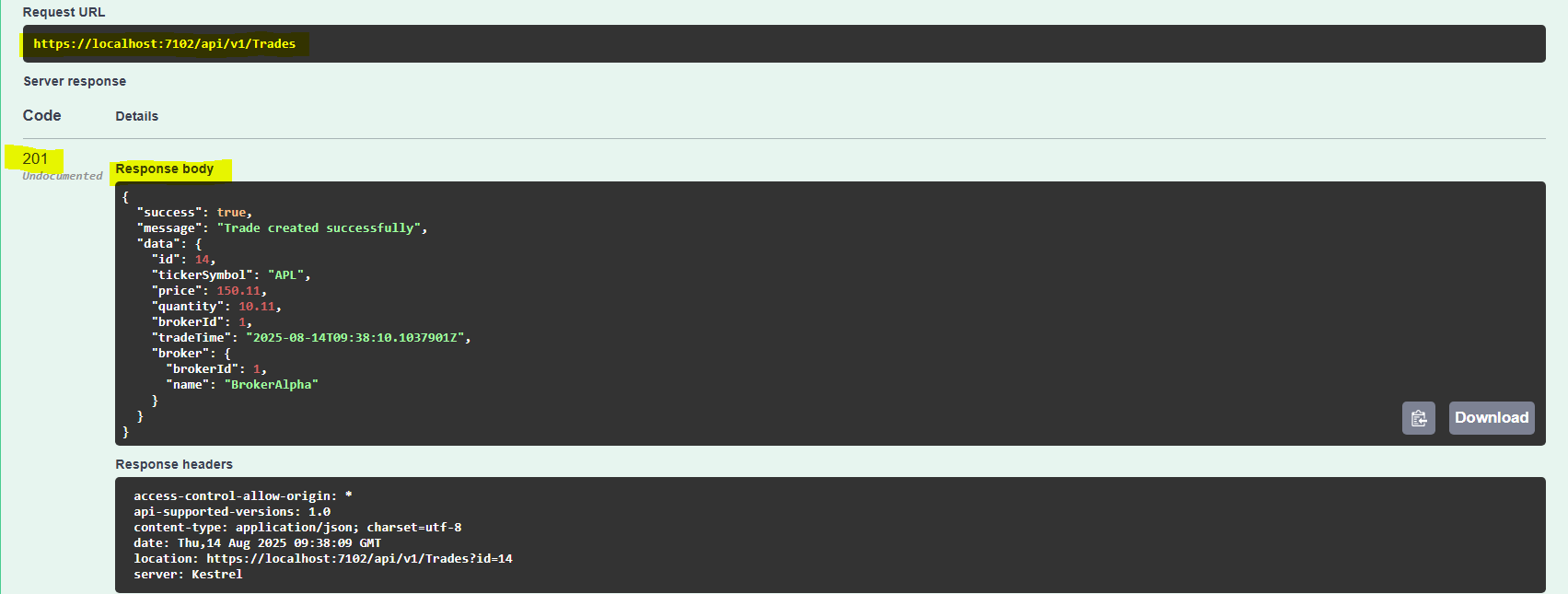
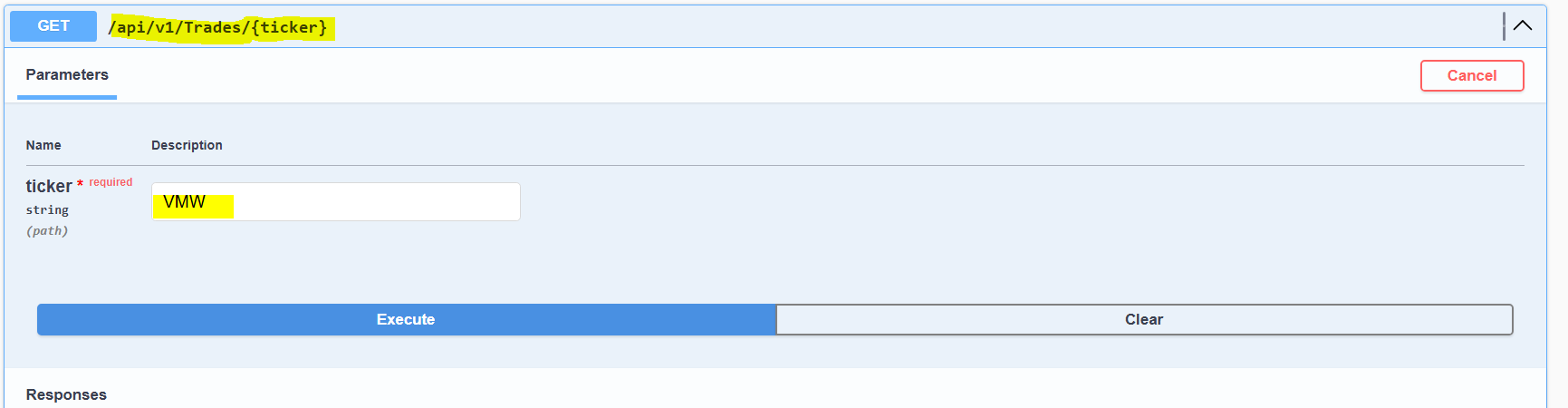
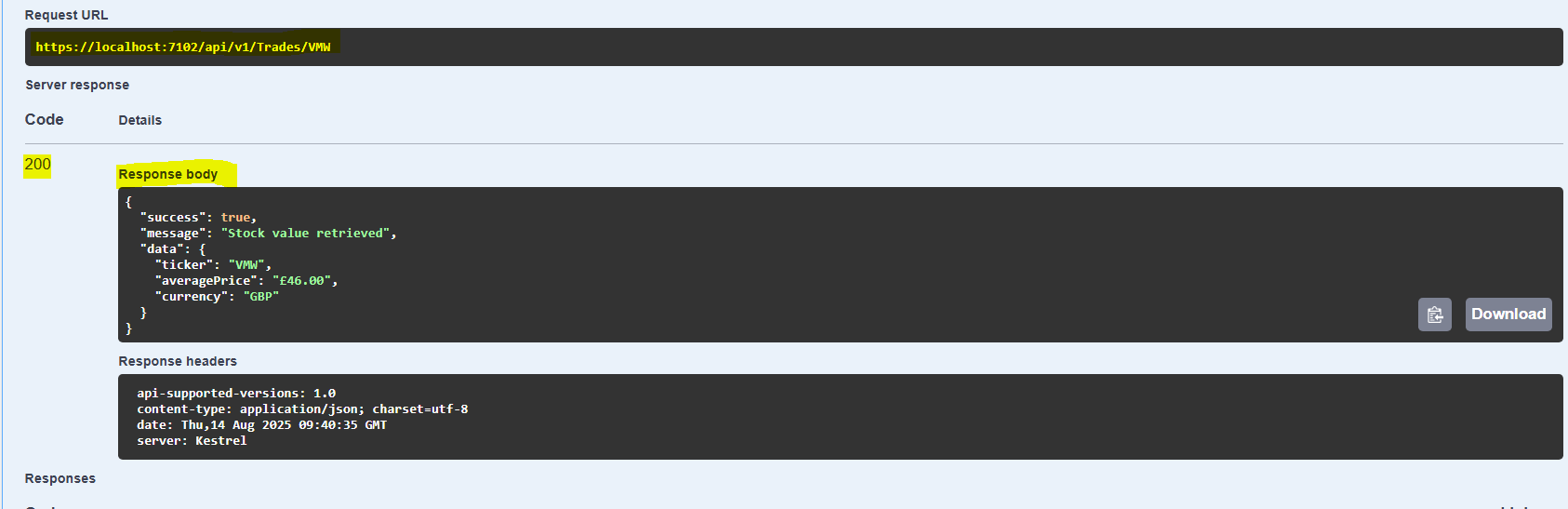
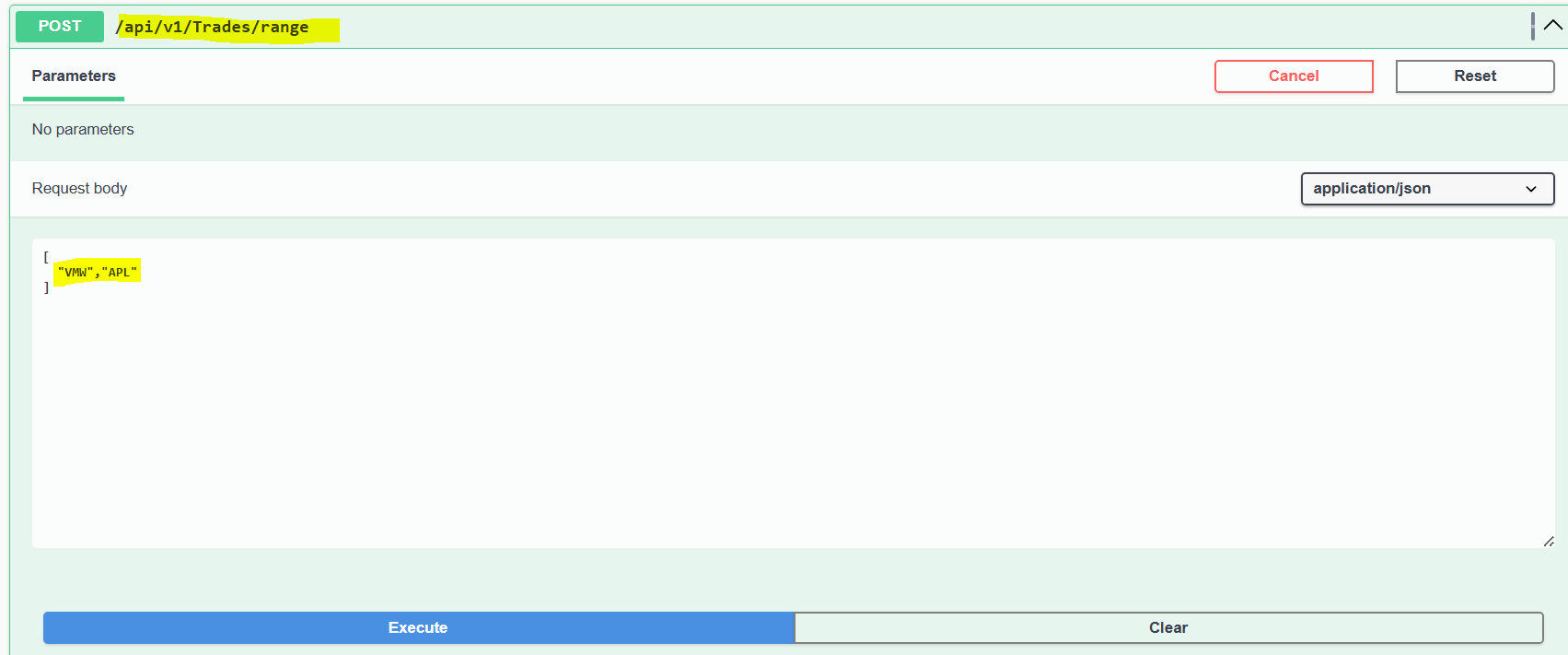
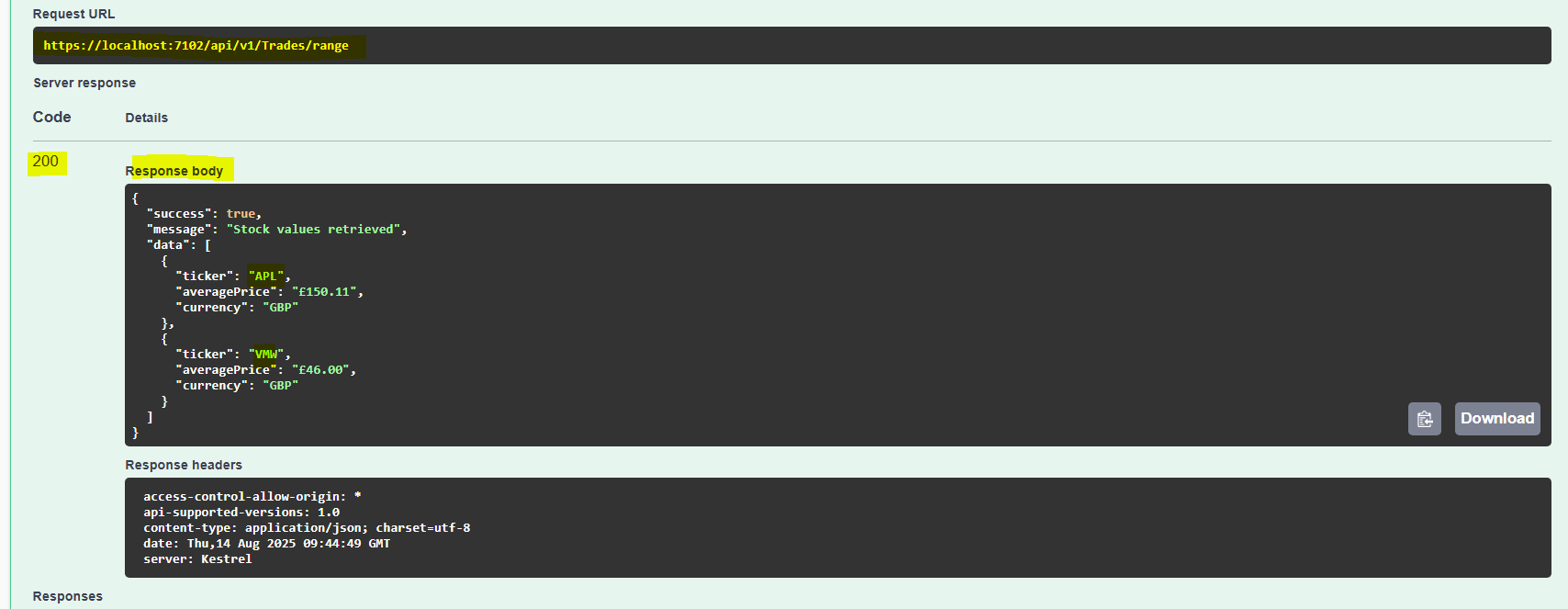
**Handling Concurrency Requests :**

| **Layer** | **Concurrency Concern** | **Why** |
| --- | --- | --- |
| **TradesController** | Yes | Needs to be async, avoid blocking threads. |
| **TradeCommandService** | Yes | Writes to DB — must handle concurrent writes safely. |
| **TradeQueryService** | Yes | Reads from DB — must be efficient under load. |
| **Database** | Yes | Needs proper indexing, connection pooling. |

### Database Layer

* **Added indexes** on frequently queried columns (e.g., TickerSymbol).
* **Used read replicas** for heavy read load.

**In TradeCommandService / TradeQueryService**

* **Made all DB calls async** (await dbContext.Trades.ToListAsync()).
* **Used dependency injection** — services should be stateless.
* **Avoided static mutable state** — it’s not thread-safe.
* **Used EF Core’s DbContext per request** — it’s not thread-safe across requests.  
    
    
    
    
    
    
    
    
    
    
    
    
    
    
    
   **LSEStockAPI Request -Reponse :**  
  1) Post : To post the required payload  
    
  Request :  
    
    
  Response :  
    
    
  2) GET Request – With Single TickerSymbol  
    
  Request :  
  Response :   
    
    
  3) POST– With multiple TickerSymbol range  
    
  Request :  
    
    
  Response :  
    
    
    
    
  4)GET – Get All the Stocks from the market (Paginations can be implemented if required)  
  