**LondonStockAPI – Technical Documentation**

**Contents :**

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

**Document Revision History :**

| **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 standard architectural patterns and design principles throughout the application | Scalability, maintainability, testability, performance optimization, clean separation of concerns |
| **Authentication and Authorization** | Implemented **JWT OAuth 2.0** authentication and **role-based authorization** using Microsoft.AspNetCore.Identity | Secures API endpoints, supports user roles, enables token-based stateless authentication |
| Exception Handling | Centralized via ExceptionMiddleware returning custom APIResponse | Consistent error handling |
| 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 |
| Rate Limiting | Configured in Program.cs and enabled at controller level | Prevents abuse and resource exhaustion |
| CORS Enablement | Configured in Program.cs and enabled at controller level | Controls cross-origin requests |
| Unit Testing | Configured with Moq, xUnit, FluentAssertions | Ensures endpoint correctness |
| Swagger UI | Configured in Program.cs | Interactive API documentation |

### 3. Detailed Feature Enhancements 3.1 Authentication & Authorization

* Implemented **JWT-based authentication** using:
  + Microsoft.AspNetCore.Identity for user management
  + System.IdentityModel.Tokens.Jwt for token generation
  + Microsoft.IdentityModel.Tokens for signing credentials
* **Login Flow**:
  + User sends credentials to /api/v1/Auth/login
  + Credentials validated via UserManager
  + JWT token generated with:
    - ClaimTypes.Name
    - ClaimTypes.Role
    - Expiry from configuration (Jwt:ExpireMinutes)
  + Token returned to client
* **Authorization**:
  + [Authorize] attribute applied to protected controllers/actions
  + Role-based restrictions via [Authorize(Roles = "Admin")]
* **Security Benefits**:
  + Prevents unauthorized access
  + Supports fine-grained role-based access control
  + Tokens are stateless and scalable

"Jwt": {

"Key": "SecretKeyHere",

"Issuer": "LondonStockAPI",

"Audience": "LondonStockAPIUsers",

"ExpireMinutes": 60

}

**3.2 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."**

**]**

**}**

**3.3 Structured Logging with Serilog**

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

**3.4 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.

**3.5 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.

**3.8 API Versioning , CORS Limiting & Rate Limiting Fetaures :**

| **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 |

**3.9) 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.

**3.10 Swagger Security Documentation**

Added **AuthResponsesOperationFilter** to automatically document security-related responses in Swagger UI.  
  
**Purpose:**

* Automatically adds:
  + 401 Unauthorized response for missing/invalid JWT
  + 403 Forbidden response for insufficient permissions
* Ensures **Swagger UI** clearly communicates security requirements

**4. Code Changes Summary**

**4.1 Controllers**

* TradesController updated to:
  + Use logging scopes
  + Return APIResponse for errors
  + Measure execution time
  + Call services with caching
* AuthController updated to:
  + Add Authentication and Authorization based on the Token
  + Return APIResponse for errors

**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 Added for each of the Controllers (Trades & Auth)

**TradeServiceTests**:

* Mock ILogger<TradesController>
* Keep existing ApiResponse checks for success cases  
    
  **AuthServiceTests**:
  + Valid login returns JWT token.
  + Invalid password returns Unauthorized.
  + Unknown user returns Unauthorized.
* Used **Moq** to mock UserManager<ApplicationUser> and IConfiguration.
* Used **FluentAssertions** to validate token

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 :**

**{**

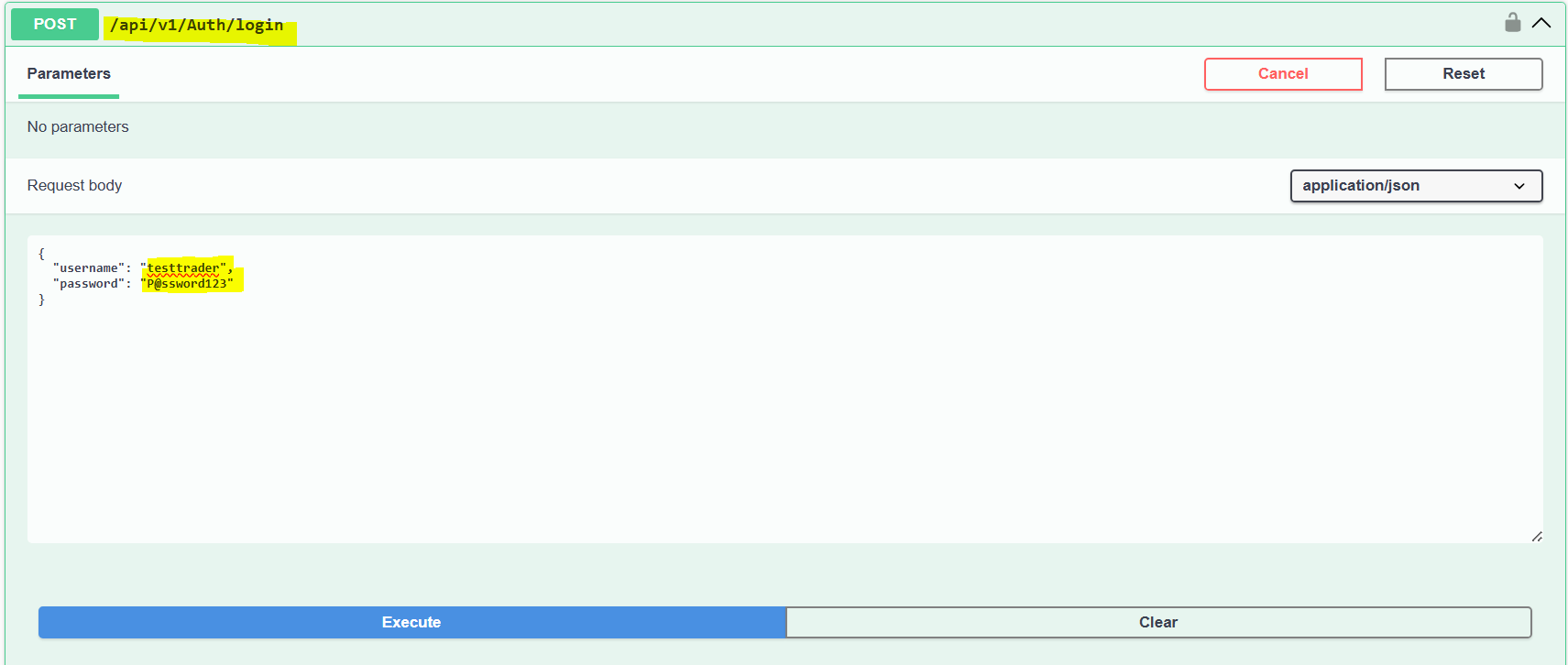
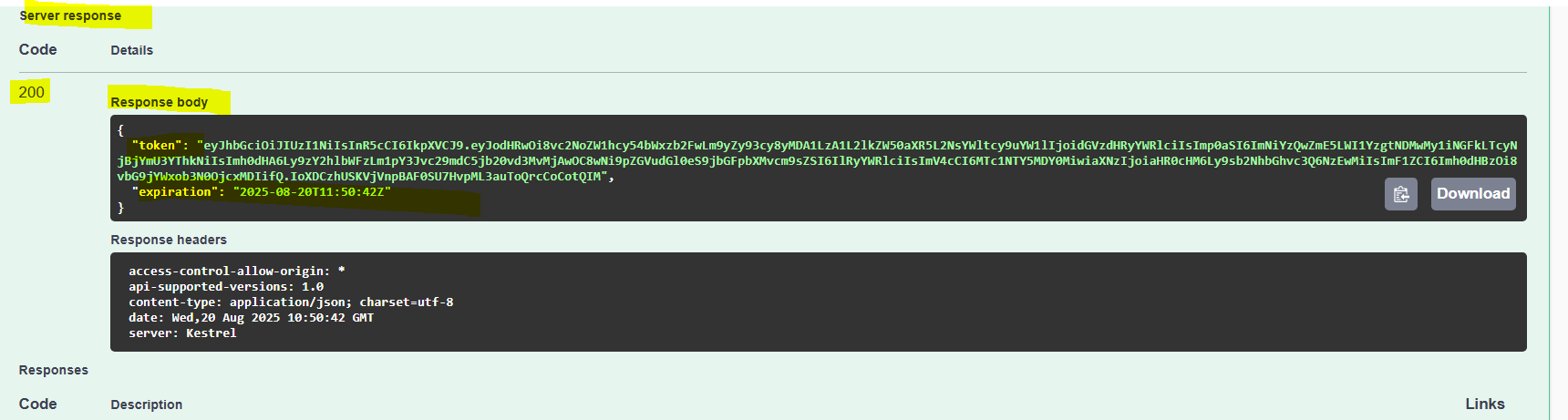
**"success": false,**

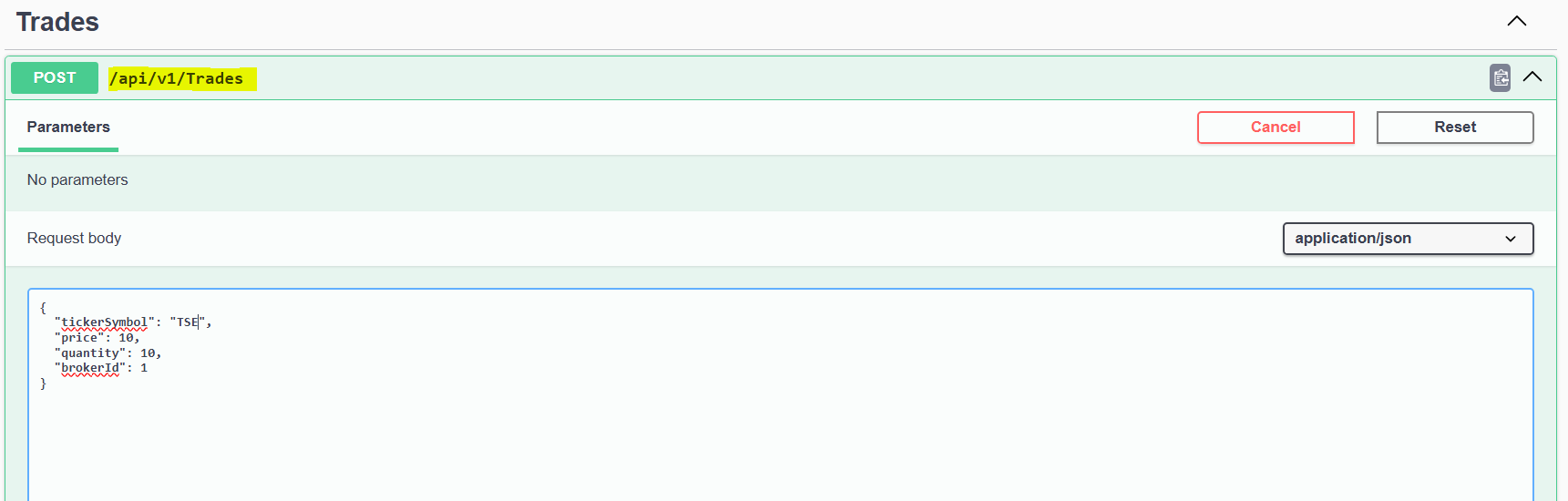
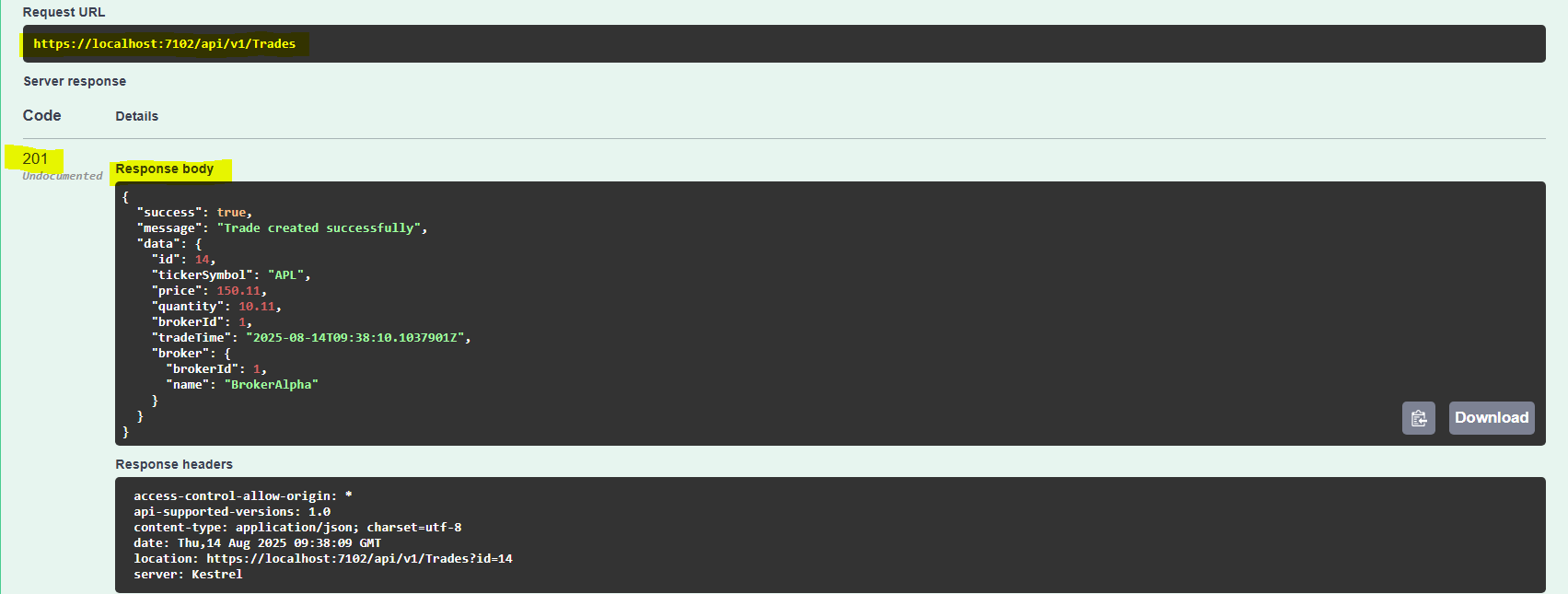
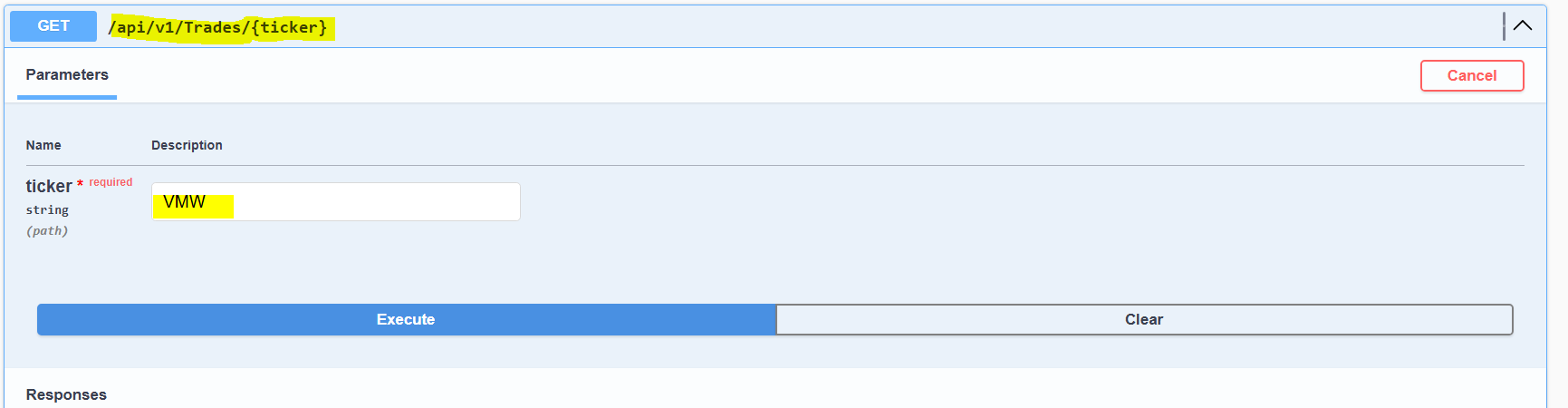
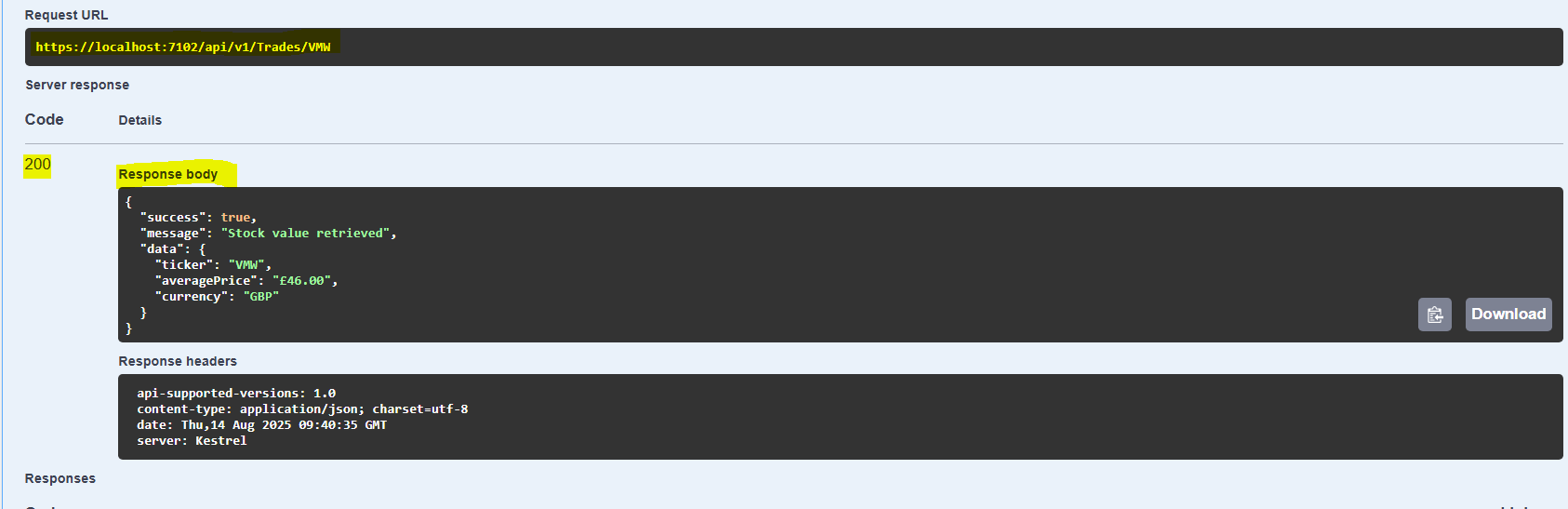
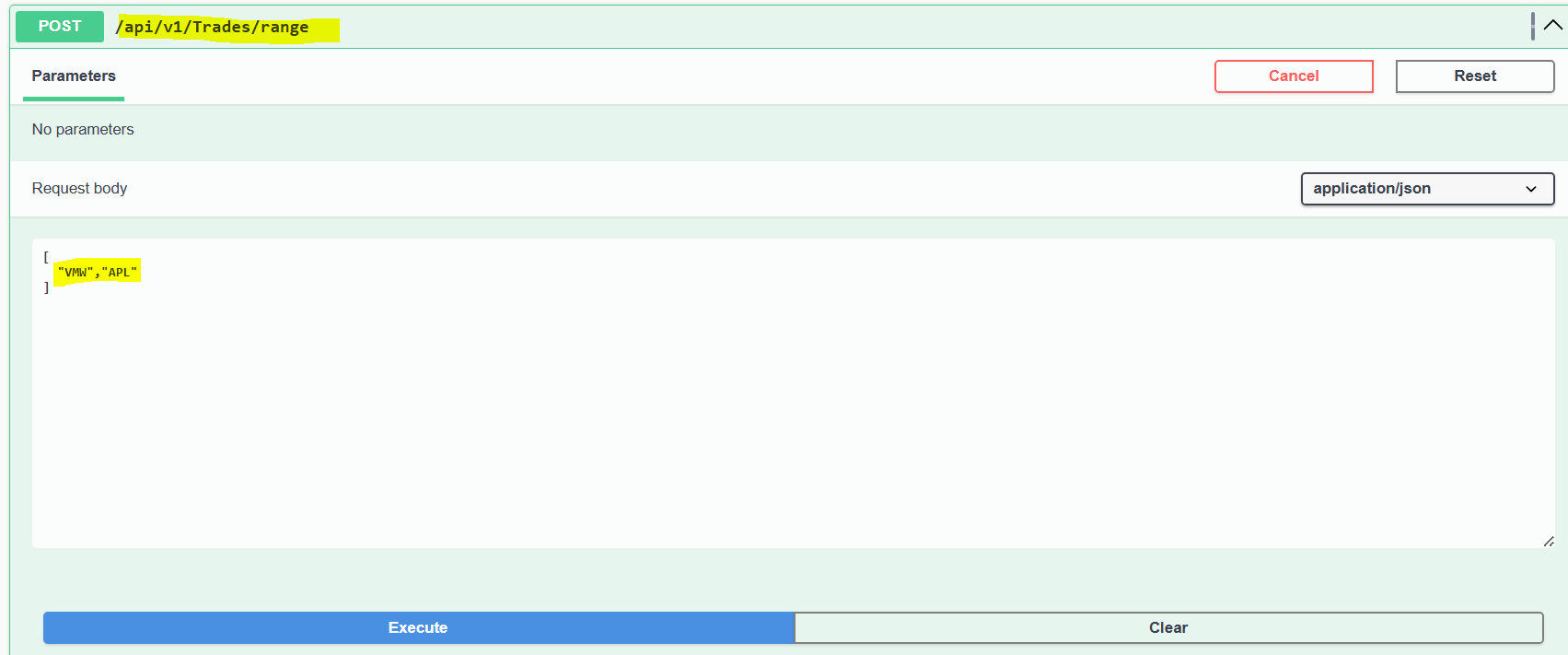
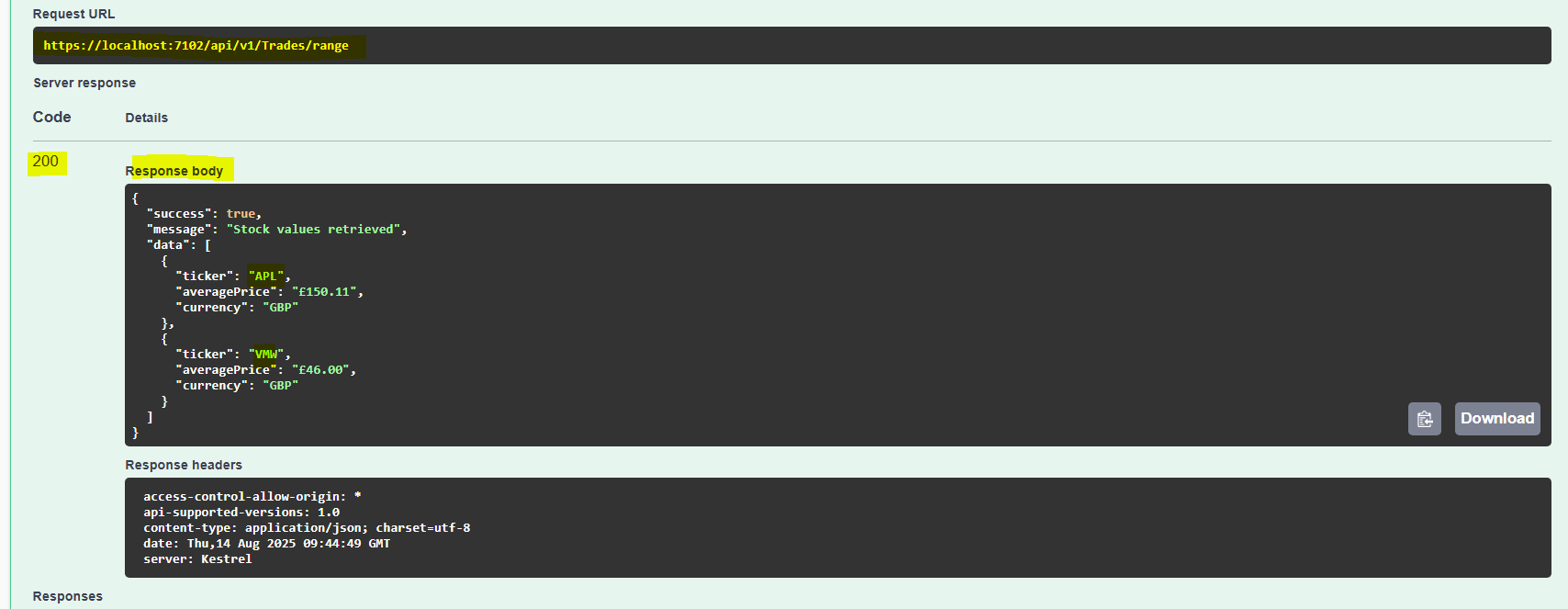
**"message": "Broker with ID 10 does not exist.",**

**"data": null**

**}**

**LSEStockAPI Test Results :**

Auth Controller:1) Post Request :   
Response :  
  
 **Note :** Use the above token generted in the Concurrent Requests in the beloweach requests by configuring the token in the **Authorize** in the Swagger UI

i.e Click on Authorize -> add **Bearer “token generated value”**  
  
  
  
  
  
  
  
  
  
  
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)  
