**.NET Core Trades Web API Documentation**

## Introduction

Welcome to the documentation for the .NET Web API.

This Trades API allows you to perform CRUD operations with retry policy and global exception handling. Additionally Search Api with pagination and sorting capabilities has been implemented.

## Prerequisites

* .NET 6.0 SDK or later
* SQL server
* SQL database schema and table script to be executed from path <https://github.com/KatkarShweta/TradesWebAPI/blob/main/SQLTradesWebAPI.sql>
* Github Code Repo link <https://github.com/KatkarShweta/TradesWebAPI.git>

## Getting Started with TradesWebAPI

To begin working with the TradesWebAPI project, follow these step-by-step instructions:

1. **Clone the Repository:**
   * Open your terminal or command prompt.
   * Execute the following command to clone the repository:

git clone https://github.com/KatkarShweta/TradesWebAPI.git

1. **Navigate to the Project Directory:**
   * Change into the directory where the project has been cloned:

cd TradesWebAPI

1. **Restore Dependencies:**
   * Ensure all required packages and dependencies are restored:

dotnet restore

1. **Build the Project:**
   * Build the solution to compile the code and resolve any build-time issues:

dotnet build

1. **Run the Application:**
   * Once built successfully, start the application:

dotnet run

* + This command will launch the web API application at below application urls
  + https://<your application\_host>:7028
  + http://< your application\_host >:5209

**Key Considerations in Trades Web API Project Development**

1. **Three-Layer Architecture:**
   * Utilized a structured approach with separate layers:
     + **Presentation Layer:** Handles HTTP requests and responses.
     + **Service Layer:** Implements business logic and orchestrates data operations.
     + **Data Access Layer:** Interacts with the database using repository pattern.
2. **Shared Library for Common Functionality:**
   * Centralized common data objects, model entity classes, and global exception handling mechanisms.
   * **Global Exception Handler:** Managed exceptions uniformly across the application:
     + Handles common exceptions such as BadRequest, NotFound, and InternalServerError.
     + Ensures consistent error responses and logging, reducing redundancy across layers.
3. **Retry Backoff Policy:**
   * Implemented retry mechanisms in repository layers to handle transient faults:

Policy

.Handle<SqlException>()

.Or<TimeoutException>()

.WaitAndRetryAsync(

retryCount: 3,

sleepDurationProvider: attempt => TimeSpan.FromSeconds(Math.Pow(2, attempt)),

onRetry: (exception, timespan, retryCount, context) =>

{

\_logger.LogInformation($"Retry {retryCount} encountered {exception.GetType().Name}: {exception.Message}. Waiting {timespan} before next retry.");

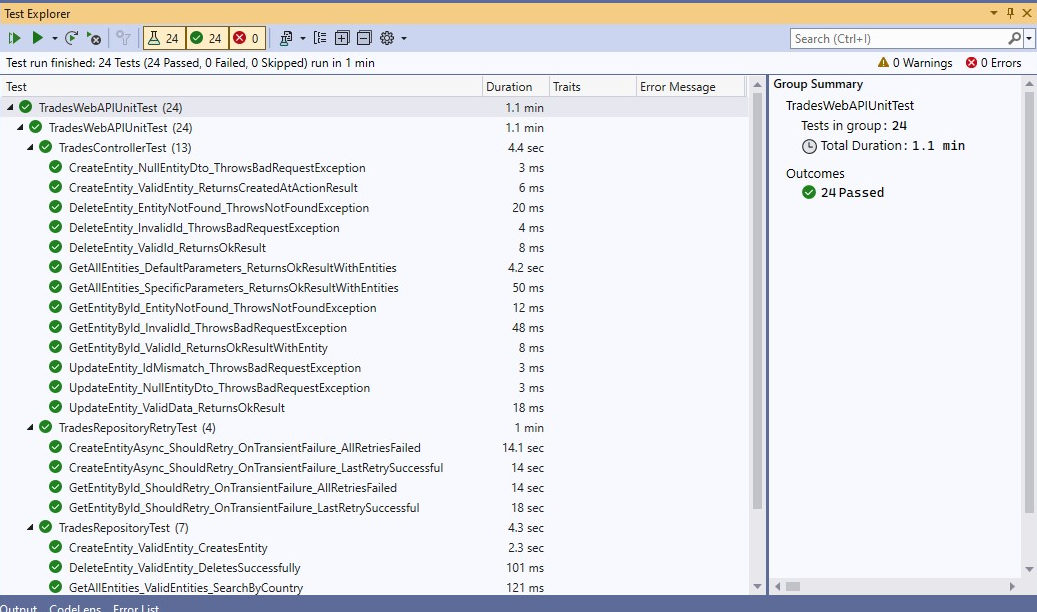
});

* + **Policy Rationale:**
    - Handles SqlException and TimeoutException to ensure robust database operations.
    - Configured for 3 retries with exponentially increasing intervals (2s, 4s, 8s) to mitigate contention.
    - Logs retry attempts to aid troubleshooting and performance monitoring.

1. **Comprehensive Testing Strategy:**
   * **Test Project Inclusion:**
     + Validates Controller and Repository layer API functionalities.
     + Includes specific tests for retry policies, utilizing in-memory databases and mocks.
     + Ensures reliability and correctness of retry mechanisms under various scenarios for read and write database operations by mocking timeout exception.
2. **Swagger documentation :**

<https://github.com/KatkarShweta/TradesWebAPI/blob/main/trades_web_api_swagger.json>

1. **Test Outcome:**



**Retry Test Scenarios:**

Following test cases created to test the retry backoff policy.

