# Financial Document Retrieval API

## Introduction

The goal of this project was to develop a new .NET C# Web API endpoint that allows clients to retrieve financial documents within a suite of applications used for submitting tax returns. The solution demonstrates the application of clean code and best practices in software development.

This documentation reflects my interpretation of the project requirements and objectives. Please note that it may contain inadvertent inaccuracies or misinterpretations.

## Objective

The primary objective was to create a Web API endpoint for financial document retrieval. This includes processing requests with various criteria and returning the requested data in a anonymized format.

## Project Structure

The project is structured into three main components: Core, Infrastructure, and API. The Core component defines the fundamental entities used across the application. The Infrastructure component includes repositories for data handling. The API component contains services and controllers for processing requests and delivering responses.

## Core Entities

Core entities include Client, Company, FinancialDocument, Product, Tenant, and Transaction. Each entity represents a specific aspect of the financial document retrieval process and is used to store and manage relevant data.

## Infrastructure Repositories

Repositories like ClientRepository, CompanyRepository, FinancialDocumentRepository, ProductRepository, and TenantRepository provide data access and manipulation capabilities, abstracting the underlying database interactions.

## API Services

Services like ClientService, FinancialDocumentService, CompanyService, ProductValidationService, and TenantService are responsible for executing business logic. They interact with repositories to retrieve data and process it according to the business rules.

## Controller

The FinancialDocumentRetrievalController is the entry point for API requests. It coordinates the workflow of processing a request, invoking the necessary services, and returning the appropriate response.

## Database and ORM

SQLite database was used for data storage, and Entity Framework Core served as the Object-Relational Mapping (ORM) tool.

## Data Seeding

Initial data was seeded into the SQLite database to facilitate testing and demonstration. This included predefined sets of clients, tenants, financial documents, and transactions.

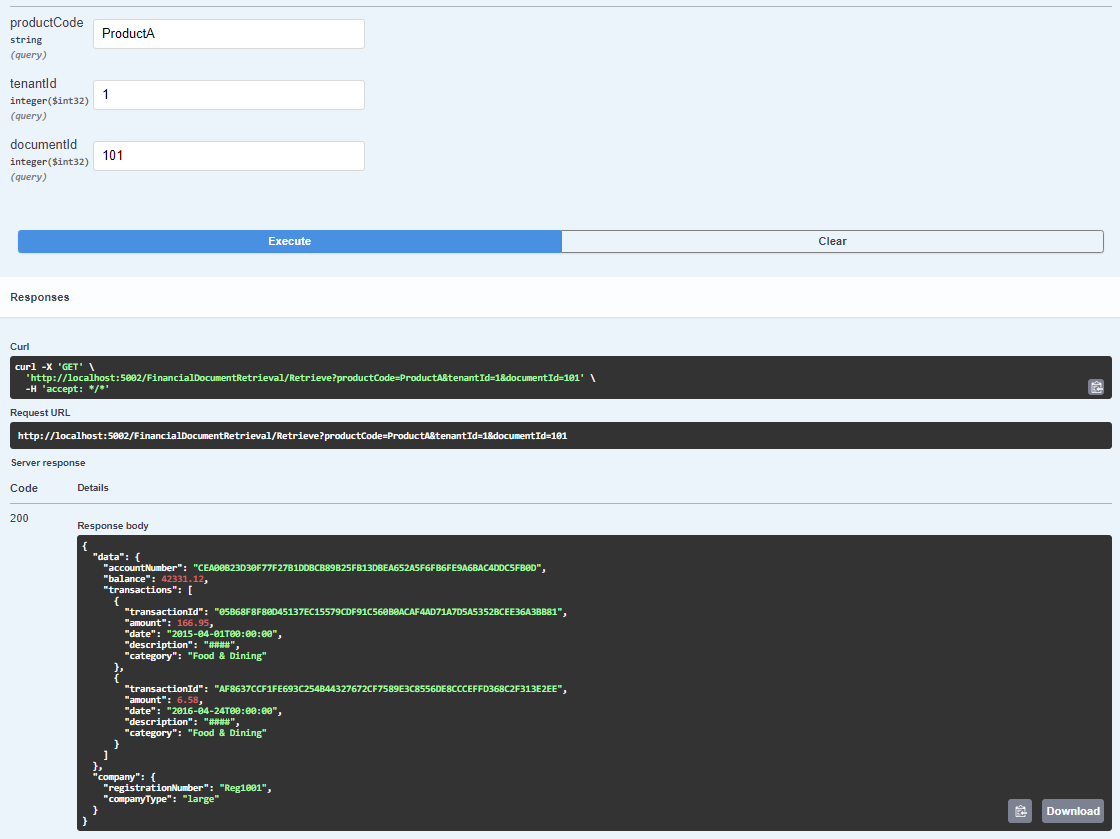
## Data Transfer Objects (DTOs)

DTOs like FinancialDocumentDTO and TransactionDTO were utilized to transfer data between different layers of the application. They ensure that only necessary data is exposed to the clients, enhancing security and efficiency.

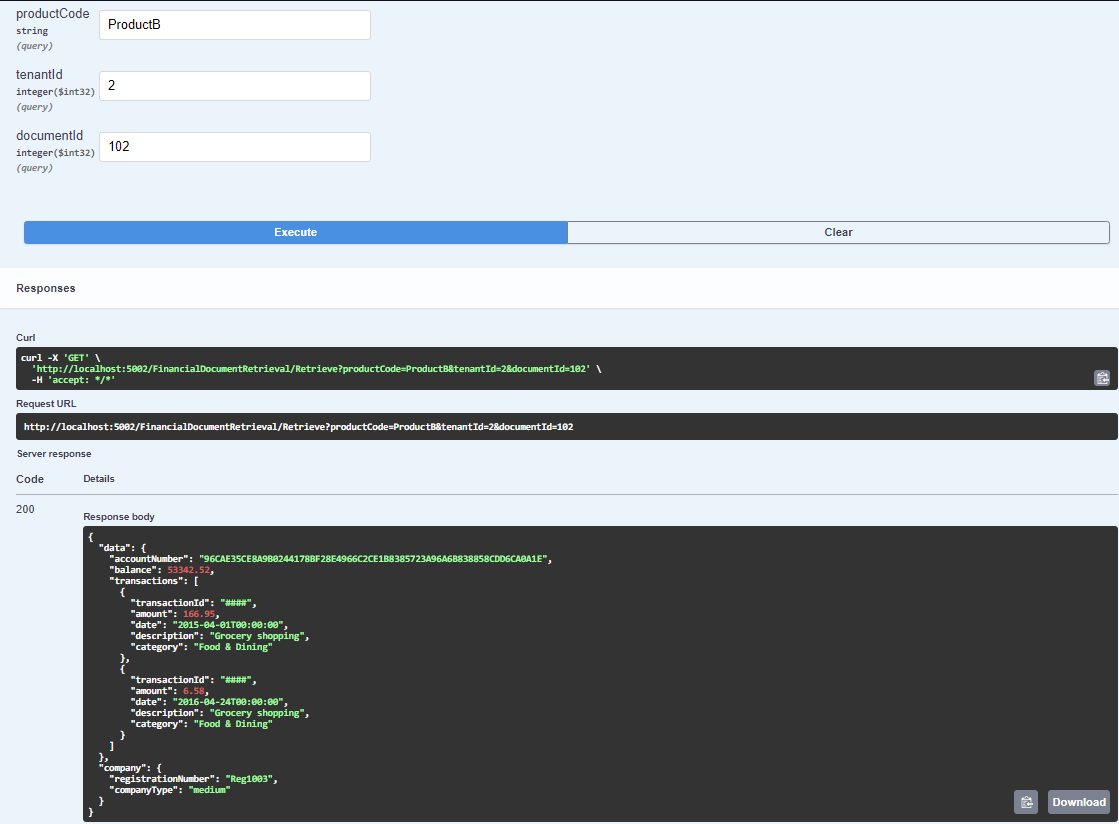
## Testing

Testing was primarily conducted using Swagger UI, which provided an interactive interface for sending requests to the API and observing responses. This helped in verifying the correct functionality of each endpoint and ensuring that the data was handled as expected.

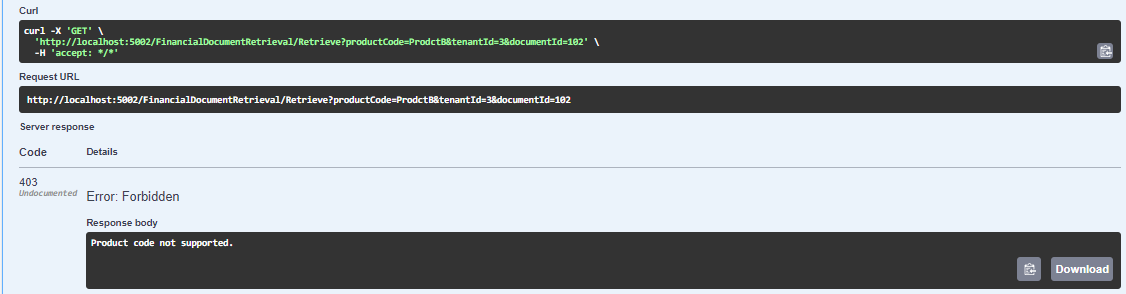
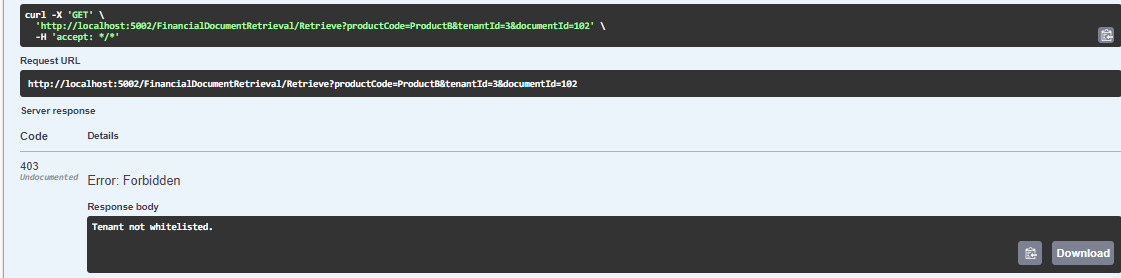
**Test scenario #1**: { "productCode": "ProductA", "tenantId": "1", "documentId": "101" }, some fields are set as hashed and some as masked for testing purpose.



**Test scenario #2**: { "productCode": "ProductB", "tenantId": "2", "documentId": "102" }, some fields are set as hashed and some as masked for testing purpose.



**Error handling showcased:**

* Incorrect Product Code (ProdctB):  
  
* Tenant not whitelisted (id 3):  
  
* Client not whitelisted(provided a tenanted of 1 which points to a not whitelisted client):



* Company Type = small:  
  