**ONLINE SHOPPING CART MANAGEMENT SYSTEM**

**By**

**NDIPHIWE NOMBULA**

**OCTOBER 2023**

**BRIEF SUMMARY**

The Online Shopping Cart Management System (SCMSystem) is a RESTful API based application intended for online retailers.

The main objective of this application is to design a RESTful API using .NET Core 6 which enables the user to perform shopping cart management activities. This application should be able to handle tasks such as adding new items to the cart, updating the details of the existing items, fetching the details (metadata) of the items, and removing the items from the cart.

The system should be secured using OAuth 2.0 and data should be stored in a relational

database.

**Table of Contents**

1. Introduction
   1. Goals
   2. Need
   3. Scope
   4. Platform Specifications – Deployment
2. System Analysis
   1. Entity Relational Diagram
   2. Data Flow Diagram
   3. Use case Diagram
3. Design
   1. Design Goals
   2. Architectural Design
   3. Procedural/Modular Approach
4. Implementation
   1. Database Design and Implementation
   2. User Interface Design and Implementation
   3. Technical Discussions
5. Security
   1. JWT and OAuth2 Service
   2. Rate Limit
6. Testing
   1. Unit Testing
7. Limitations
8. Scope for Future Work
9. Approach to address scaling and performance as the application grows.
10. Conclusions
11. **Introduction**
    1. **Goal**

To develop an easy to use web based interface where users can search for products, view a complete description of the products and order the products.

* 1. **Need of the application**

There are large numbers of commercial Online Shopping websites offering large number of products tailored to meet the shopping interests of large number of customers. These online marketplaces have thousands of products listed under various categories.

* 1. **Scope**

This system can be extended to allow the users to create accounts and save products in to wish list.

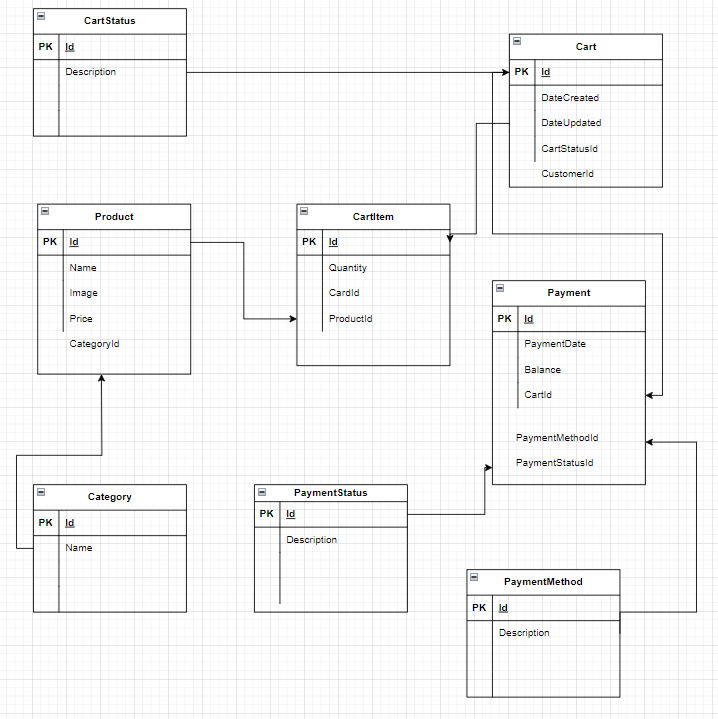
• The users could subscribe for price alerts which would enable them to receive messages when price for products fall below a particular level.

• The current system is confined only to the shopping cart process. It can be extended to have a easy to use check out process.

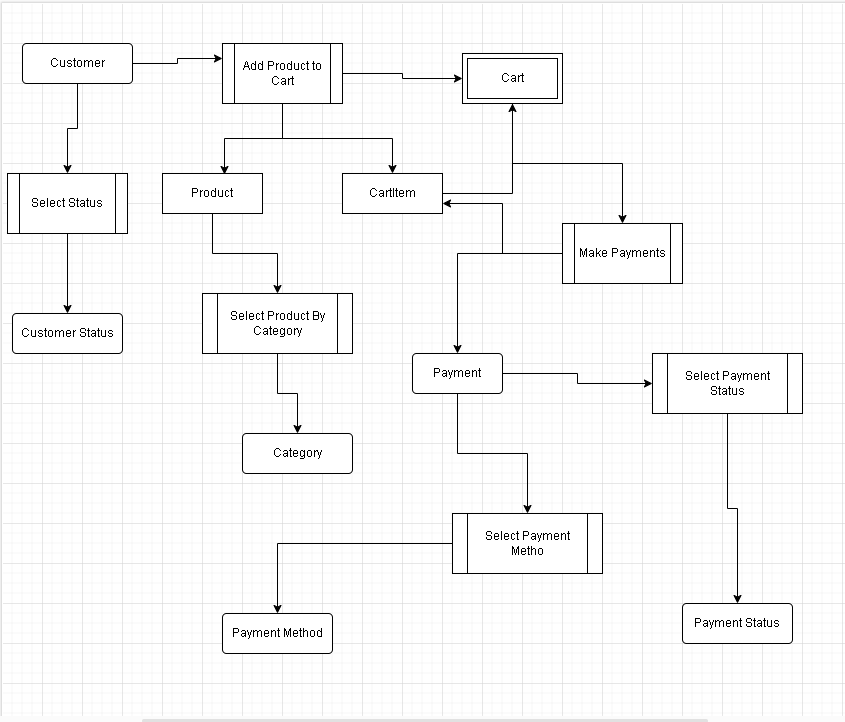
• Users can have multiple shipping and billing information saved. During checkout they can use the drag and drop feature to select shipping and billing information.

* 1. **Platform Specifications – Deployment**
     1. Windows 10 to latest
     2. Visual Studio .Net Core 6
     3. .Net Framework
     4. Database SQL Server 2014 to latest
     5. Swagger
     6. JWT
     7. Rate Limit

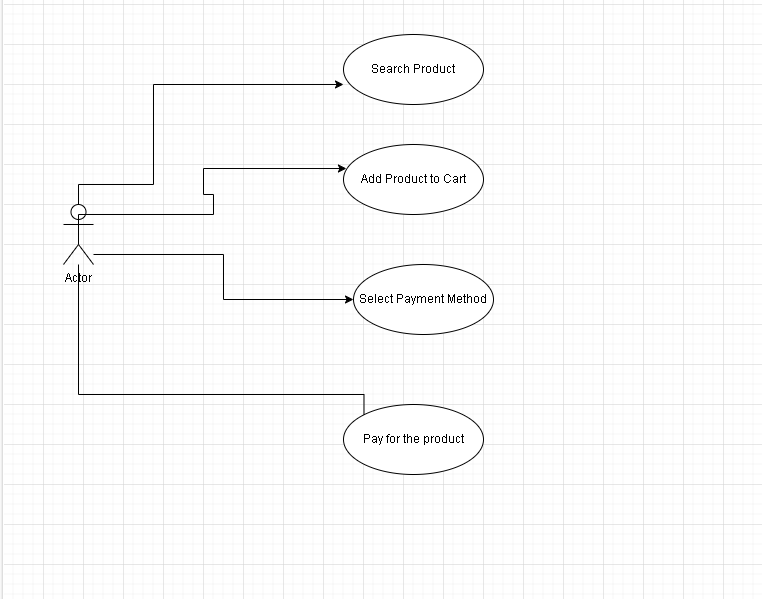
1. **System Analysis**
   1. **ER Diagram**



* 1. **Data Flow Diagram**



* 1. **Use Case Diagram**

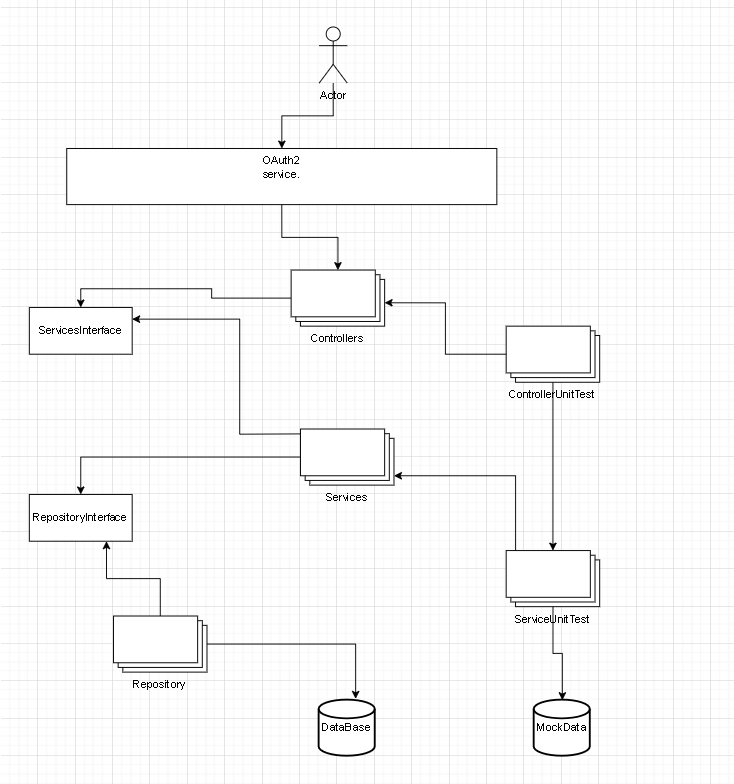


1. **Design**
   1. **Design Goals**

The design of the API involves the design of the endpoints for listing the products, search for products, display the complete specification for the product, and design a shopping cart that is easy to use.

• Design of an interactive application that enables the user to filter the products based on different parameters.

* 1. **Architectural Design**



* 1. **Description of Architectural Design**

In the system, we can observe that the user interacts with the application through OAuth2 Service Authentication to be able to access the Controller endpoints.

The Controllers access Services via Services Interface.

Services access the Repository via Repository Interface.

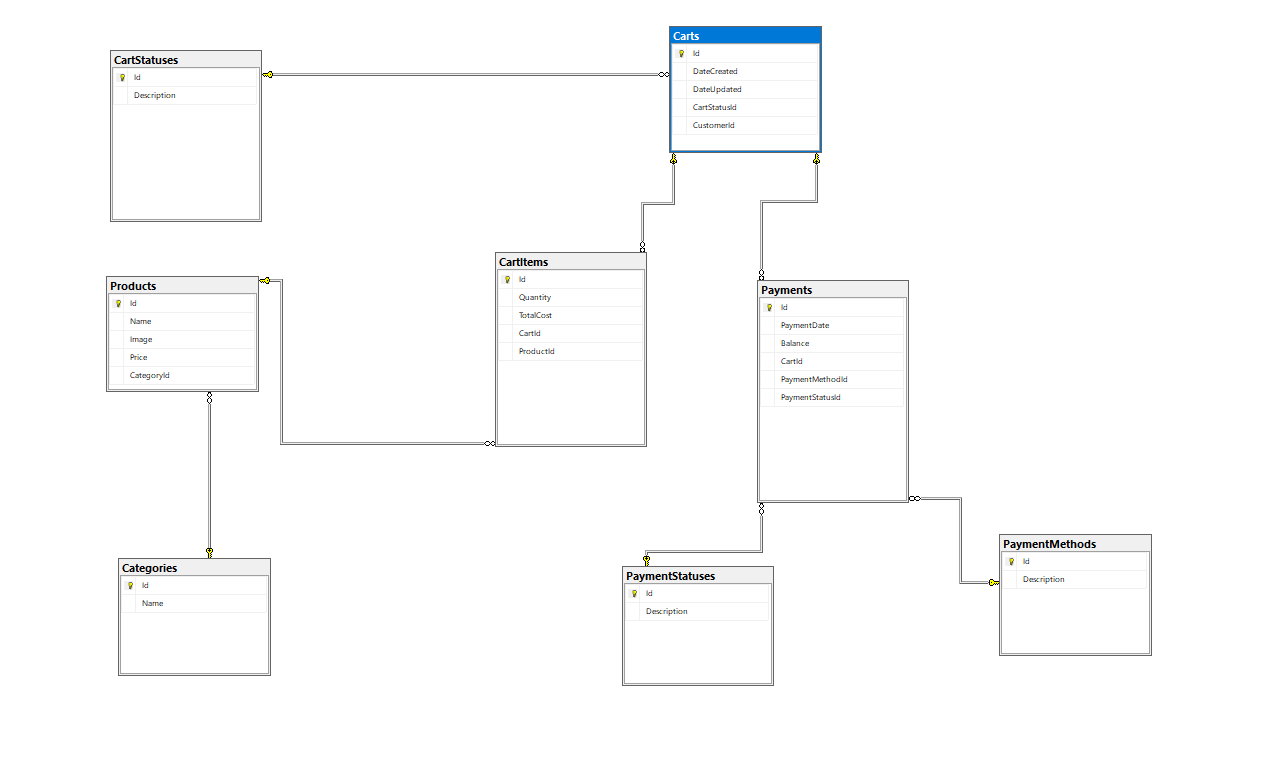
Repository calls the Database to Retrieve data.

We also have Unit test that test Controller endpoints and also a Unit test that test the actual Services and Repositories.

* 1. **Procedural/Modular Approach**

This module starts when the user visits the Swagger page to search for a product by entering a product Id. This part of the application includes displaying all the products that are available or the products that match the search term entered by the user. The user can then filter these products based on various parameters like product category. The user browse through the products and each product would be displayed with an image. A user can add a product to the cart. The user would be able to see the shopping cart summary.

1. **Implementation**
   1. **Database Design**



* 1. **User Interface Design**

Swagger is used as API interface for this project.

A screenshot of a computer

Description automatically generated

1. **Security**
   1. **JWT and OAuth2 Service**

I’ve implemented OAuth2 Service for authorization to access the end API endpoints and JWT for authentication and exchanging information.

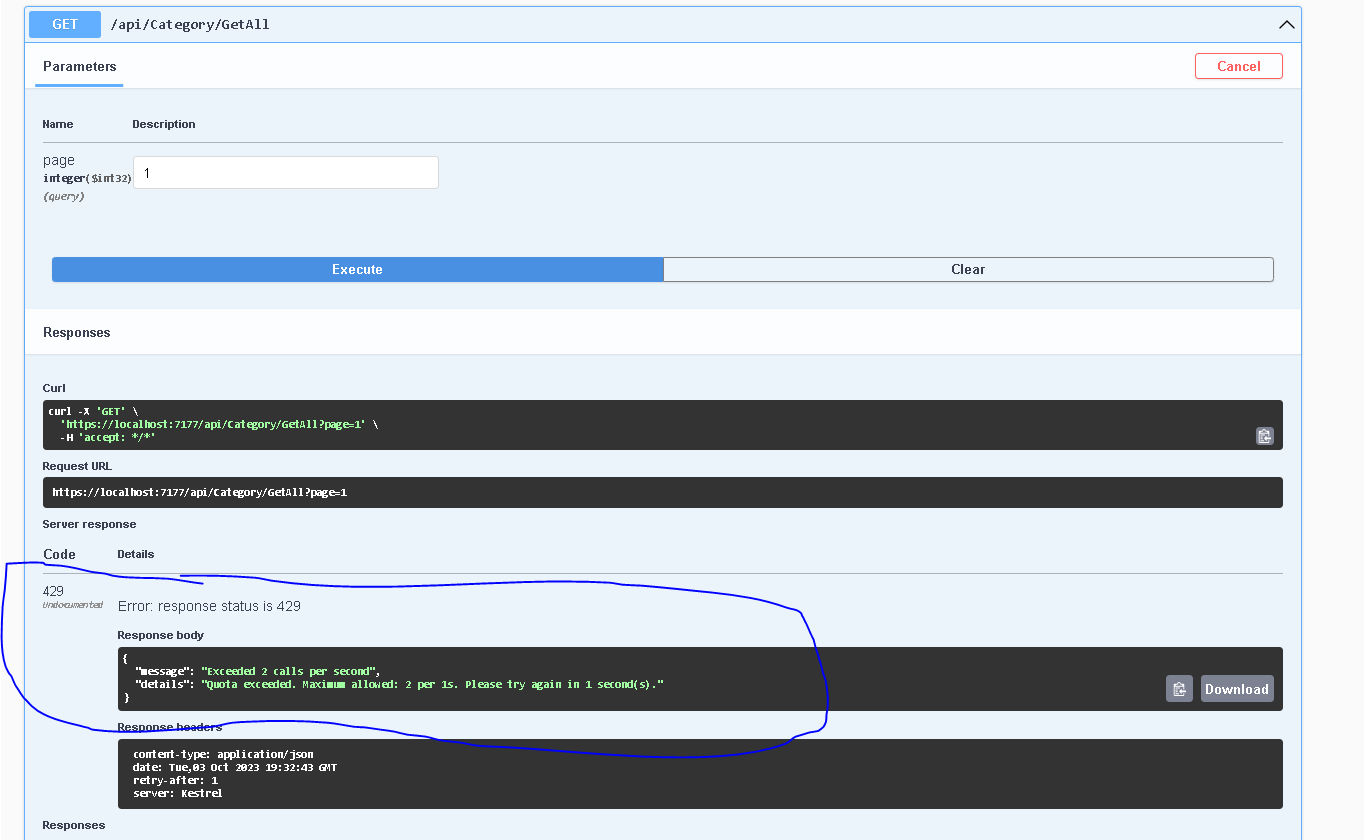
JWT delivers the best value when it’s used in an environment that requires partial information transformation to any unverified client. In addition, the situation also demands client-side information verification at the payload. It’s a great choice to make when your goal is API and server-to-server authorization.

* 1. **Rate Limiting**

I also implemented a rate limiting to protect the API against abuse. These limit settings exist to prevent your API—and your account—from being overwhelmed by too many requests. These limits are set by AWS and can't be changed by a customer. Per-account limits are applied to all APIs in an account in a specified Region.



Results after several request within a second



1. **Testing**

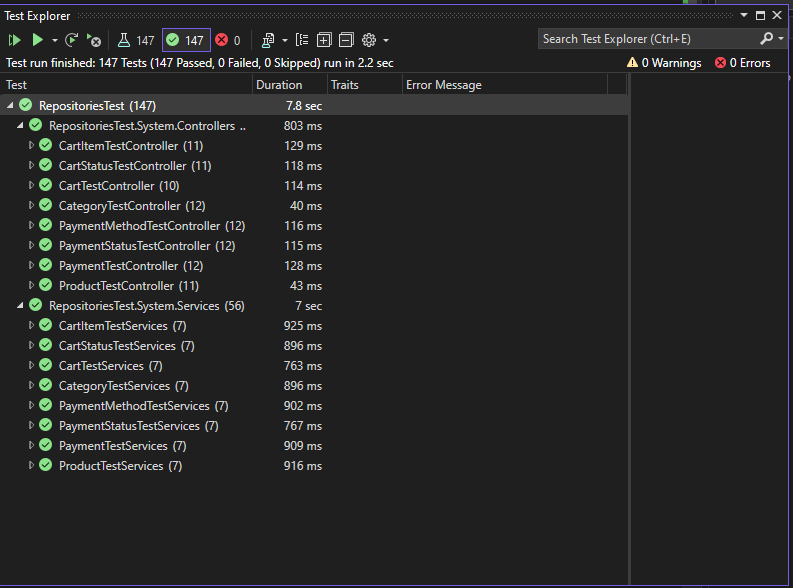
Software testing is a process of running with intent of finding errors in software. Software testing assures the quality of software and represents final review of other phases of software like specification, design, code generation etc.

* 1. **Unit Testing**

Unit testing emphasizes the verification effort on the smallest unit of software design i.e.; a software component or module. Unit testing is a dynamic method for verification, where program is actually compiled and executed. Unit testing is performed in parallel with the coding phase. Unit testing tests units or modules not the whole software.

I have tested each view/module of the application individually. As the modules were built up testing was carried out simultaneously, tracking out each and every kind of input and checking the corresponding output until module is working correctly.

The functionality of the modules was also tested as separate units. Each of the three modules was tested as separate units. In each module all the functionalities were tested in isolation.



1. **Limitations**

This application does not have a built in check out process. An external checkout package has to be integrated in to this application. Also users cannot save the shopping carts so that they can access later i.e. they cannot create wish lists which they can access later. This application does not have features by which user can set price ranges for products and receive alerts once the price reaches the particular range.

1. **Scope for Future Work**

The following things can be done in future.

• The current system can be extended to allow the users to create accounts and save products in to wish list.

• The users could subscribe for price alerts which would enable them to receive messages when price for products fall below a particular level.

• The current system is confined only to the shopping cart process. It can be extended to have an easy to use check out process.

• Users can have multiple shipping and billing information saved. During checkout they can use the drag and drop feature to select shipping and billing information.

1. **Approach to address scaling and performance as the application grows.**

There are different approaches to address scaling depending on which areas of the application experiences the most demand.

1. Middleware demand

The approach to handle scaling would be to deconstruct parts of the system into separate microservices that can be deployed into a cloud infrastructure with the capability to scale vertically or horizontally.

The first step is to identify those components that are experiencing heavy load and to then move them into separate containerized services. Another effective approach is to host the services in a platform such as AWS Fargate or on Kubernetes and allow for auto scaling of multiple pods as required.

1. Database demand

Even with multiple versions of a service deployed, often the performance will be limited by the single database in use. Effective methods for addressing database performance are to cluster the database and allow for multiple replicas.

1. Front end demand

Lastly, if the application is hosted on a cloud provider such as Azure or AWS, make use of features such as AWS CloudFront that will create replicas of the front-end site at the nearest endpoint to a user. This allows for the front-end to host faster.

1. **Conclusion**

The ‘Online Shopping’ is designed to provide a web based application that would make searching, viewing and selection of a product easier. The search engine provides an easy and convenient way to search for products where a user can Search for a product interactively and the search engine would refine the products available based on the user’s input. The user can then view the complete specification of each product.