Mini Shopping Cart API

Introduction

The "Mini Shopping Cart API " project aims to develop backend interfaces to facilitate essential shopping cart operations, including adding products, viewing product listings, and checking out items. These interfaces are designed to serve users registered with the organization, allowing them to interact seamlessly with the shopping cart system. To evaluate the functionality of the developed interfaces, the project utilizes Postman, a tool for testing RESTful APIs. The project's database architecture consists of three main entities: users, products, and carts. Each user and product is assigned a unique identification number within the relational database. The carts table serves to link users with the products they have added, maintaining information about user-selected items and quantities. This relational database structure enables efficient extraction and viewing of products associated with specific users. Users have the authority to check out their selected items, allowing updates to the database to reflect changes in product quantities. In summary, the "Mini Shopping Cart API " project integrates backend interfaces with a relational database system to offer users a seamless shopping experience while ensuring data integrity and efficient management of product information.

Problem Statement

The objective of the "Mini Shopping Cart API " project is to develop a RESTful API which offers guidelines for enhancing the performance and making codebase lightweight in nature allowing for the stability of api services. This project aims to deliver a robust API solution for users to interact with the shopping cart system efficiently and securely.

Project Setup Guide

Github Repo: https://github.com/manish-gowdans/ShoppingCartAPI

Prerequisites

- **Visual Studio 2022:** Make sure that Visual Studio 2022 is installed. It can be downloaded from here.
- **SQL Server and SQL Server Management Studio (SSMS):** Download SQL Server from here and SQL Server Management Studio (SSMS) from here.
- **Postman:** Download Postman from here and install it on a machine to test the API endpoints.
- **NuGet Packages:** Ensure the following NuGet packages are installed in project:
  - Microsoft.EntityFrameworkCore
  - Microsoft.EntityFrameworkCore.Sqlite

        - Microsoft.EntityFrameworkCore.SqlServer
        - Microsoft.EntityFrameworkCore.Tools

Installation Steps

- **Clone the Repository:** Clone the Mini Shopping Cart API repository using git bash.
- **Open Project in Visual Studio:** Open the solution file (MiniShoppingCartAPI.sln) in Visual Studio 2022.
- **Install NuGet Packages:** Open (Manage Nuget Packages) through right click on (Dependencies) in (Solution explorer) and then install the mentioned packages
- **Database Migration:** Execute the following commands in the Package Manager Console to perform database migration:
    - add-migration <Migration-Name>
    - update-database

Testing the API

- **Run the Application:** Build and run the Mini Shopping Cart API project in Visual Studio.
- **Test Endpoints with Postman:** Import the provided Postman collections to test the API endpoints.

Methodology

This project focuses on creating a backend API to power various functionalities. Initial steps involve setting up a database using entities defined within DbContexts. These entities represent the structure of our data. To transfer data between different parts of the application, we use Data Transfer Objects (DTOs). Data transfer objects are lightweight objects responsible for exchanging data between the API and other applications. Mapping code ensures smooth conversion between entities and DTOs. Repository patterns are employed to manage interactions with the database. Repositories handle database logic, abstracting complexities and providing a clean interface for controllers. Controllers are the gatekeepers of our API. They handle incoming requests and formulate appropriate responses. Through this setup, our API facilitates seamless communication, ensuring efficient data flow across applications.

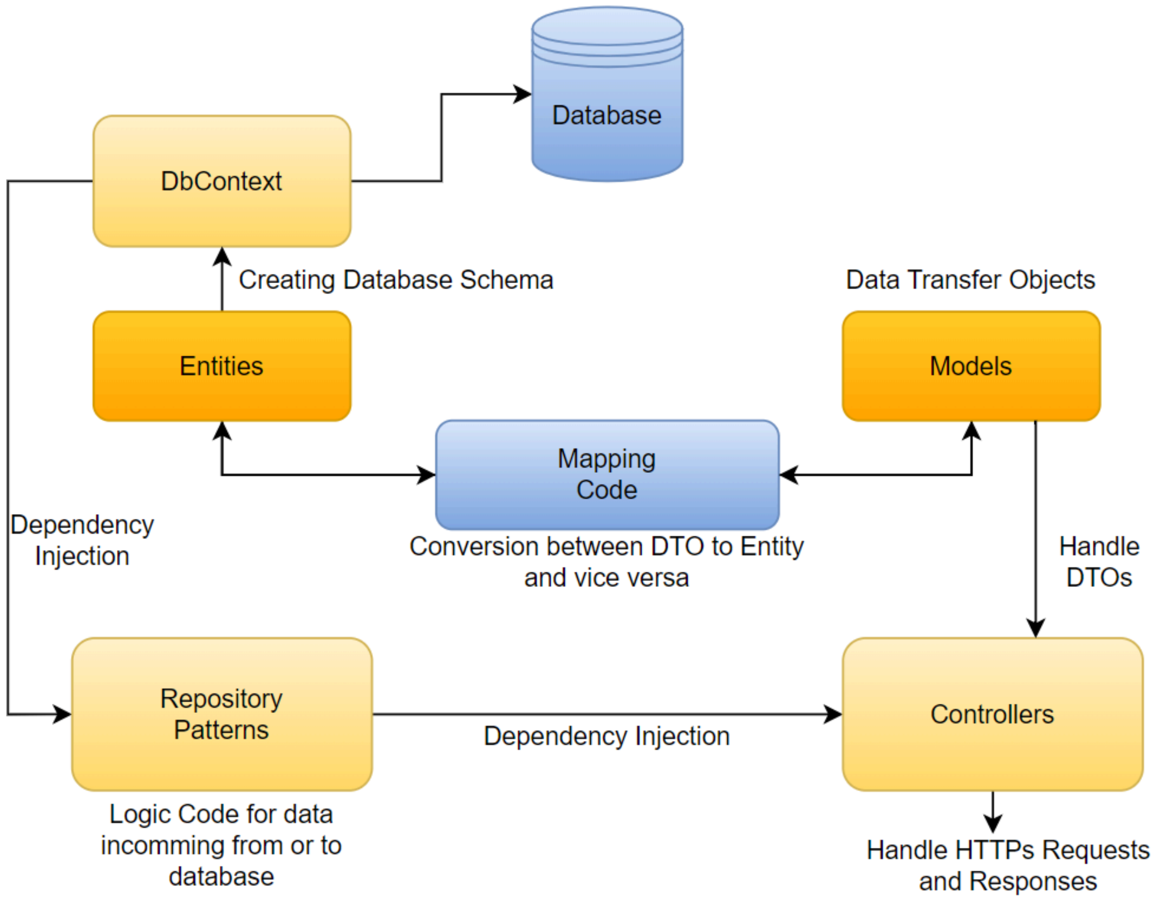


Figure a. Concept for creating the Shopping Cart API

Tech Stacks Being Used

- **Backend:** The API's are implemented using the:
  - **Programming Language:** C#
  - **Framework:** ASP.NET Core
- **Database:** SQL Express Server and SQLServer Connection using Entity Framework Core.
- **Database Management Tool:** SQL Server Management Studio 19

Database Implementation

The database connected with the "Mini Shopping Cart Api" project consists of three tables named as Users, Cart and Products, which are created through the entities that are migrated using EntityFrameCore.Tools. The Migration is a process of managing changes to the database schema by the code generated which involve creating, updating or deleting the tables. One of the major advantages of migration is that it allows the evolution of the database schema along with the application's data model and also helps in maintaining the consistency.

The entity consists of three classes: cart, product and user. All these entities are imported into the DbContext through the **generic DbSet** taking the entity classes.

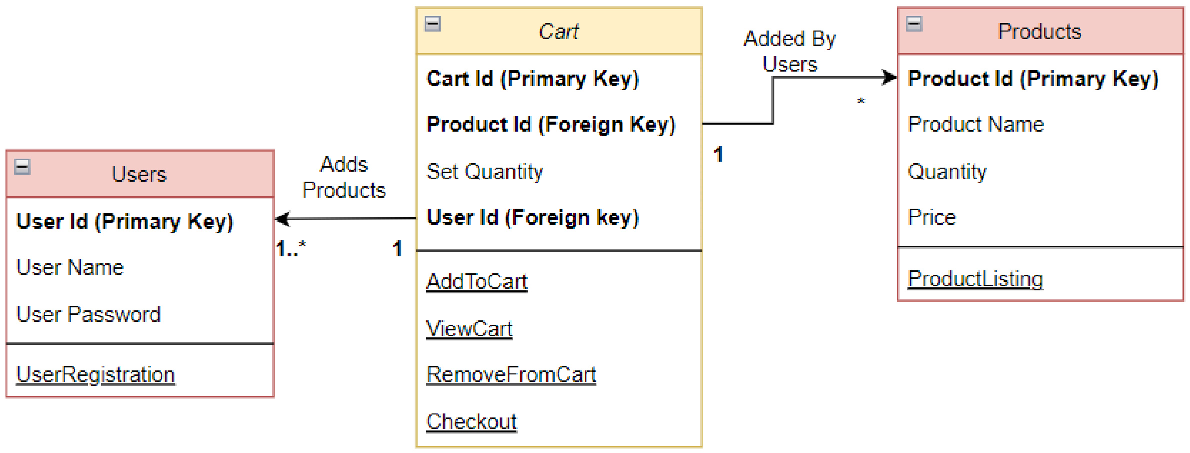


Figure b. Database Schema created through entities

Entities

The **"Cart"** entity within the database schema is designed with two foreign key relationships: one with the **"Users"** table, where the user ID serves as both the primary key in the "Users" table and a foreign key in the "Cart" table, and another with the **"Products"** table, where the product ID acts as a foreign key in the "Cart" table.

When a user interacts with the system, the "Cart" entity handles the process of storing items selected by the user for purchase. This includes capturing the user's ID, which is typically obtained from the HTTP request made by the client application. Additionally, the user specifies the product ID and the total quantity needed for each product, while adding into cart. By querying the "Cart" table with the user's ID, the system can return a list of products added to the cart by that particular user.

In essence, the "Cart" entity serves as a bridge between users and products, allowing users to store their selected items for purchase and enabling the system to retrieve and manage these selections efficiently.

Models

The Models facilitate the transfer of the data between the processes and components. Main factor of models is that it encapsulates the data structures for seamless communication.

The Models consist of Data transferable objects which serve the purpose of exchanging data between controllers, services and other data accessing components.

- **UserDto**

  Consists of the public properties which can be accessed across the application which involves:

  - **UserId:** Unique Identification for the user
  - **Username:** Username entered during registration
  - **Password:** Password to check whether a correct user is using the application

  Constructor classes are created and the username and password are initialized to make sure that null values are not provided.

- **ProductDto**

  Consists of the public properties:
  - **ProductId:** Unique identification for various products
  - **ProductName:** Unique product names given to different products
  - **Quantity:** Indicates the number of products available at that present period of time
  - **Price:** The price for each product

- **AddToCartDto**

  It consists of the public properties for objects:
  - **UserId:** It serves as a reference to associate carts with specific users through a one-to-many relationship between the Cart and User entities.
  - **ProductId:** It allows for the identification of the specific product being added to a user's cart. It becomes easier to track the products added by individual users when this product Id is combined with the User Id.
  - **SetQuantity:** This field represents the quantity of a particular product that a user intends to add to their cart.This information helps in managing inventory and calculating the total amount of items in the cart that needs to be added based on the availability of the products in inventory.

- **ViewCartInfoDto**

  This section handles the presentation of product names and prices for items stored in a user's cart. It uses a user's unique identifier to retrieve a list of product IDs linked to their cart in the database. This list can be used to either remove the products from cart or update their quantities.

- **ViewCartDto**

  This section encapsulates the user's username and the total count of products they have added to their cart, then these products are extracted based on the specific user ID provided through the HTTP request. Additionally, the ViewCartInfoDto is included in this section in list format to thoroughly display cart information when users wish to review their cart contents.

Mapping Code

Data transferable objects are lightweight objects used to transfer data between different layers of applications while entities represent the domain objects that are persistent in the database. This section involves the conversion logic to be implemented between DTOs and entities allowing the code for the logic implementation to be encapsulated. During the conversion process, the logic ensures that the properties of the DTOs, such as product name or ID, are matched with corresponding properties in the entities. This matching ensures that the data is correctly mapped from the DTOs to the entities, maintaining consistency and integrity in the application's data model. Once the DTOs are converted into entities they can be used to store or obtain information for any specific users based on their id or products that are stored inside the database, this ensures scalability for storing the information in the database and maintains effective data flow between the application.

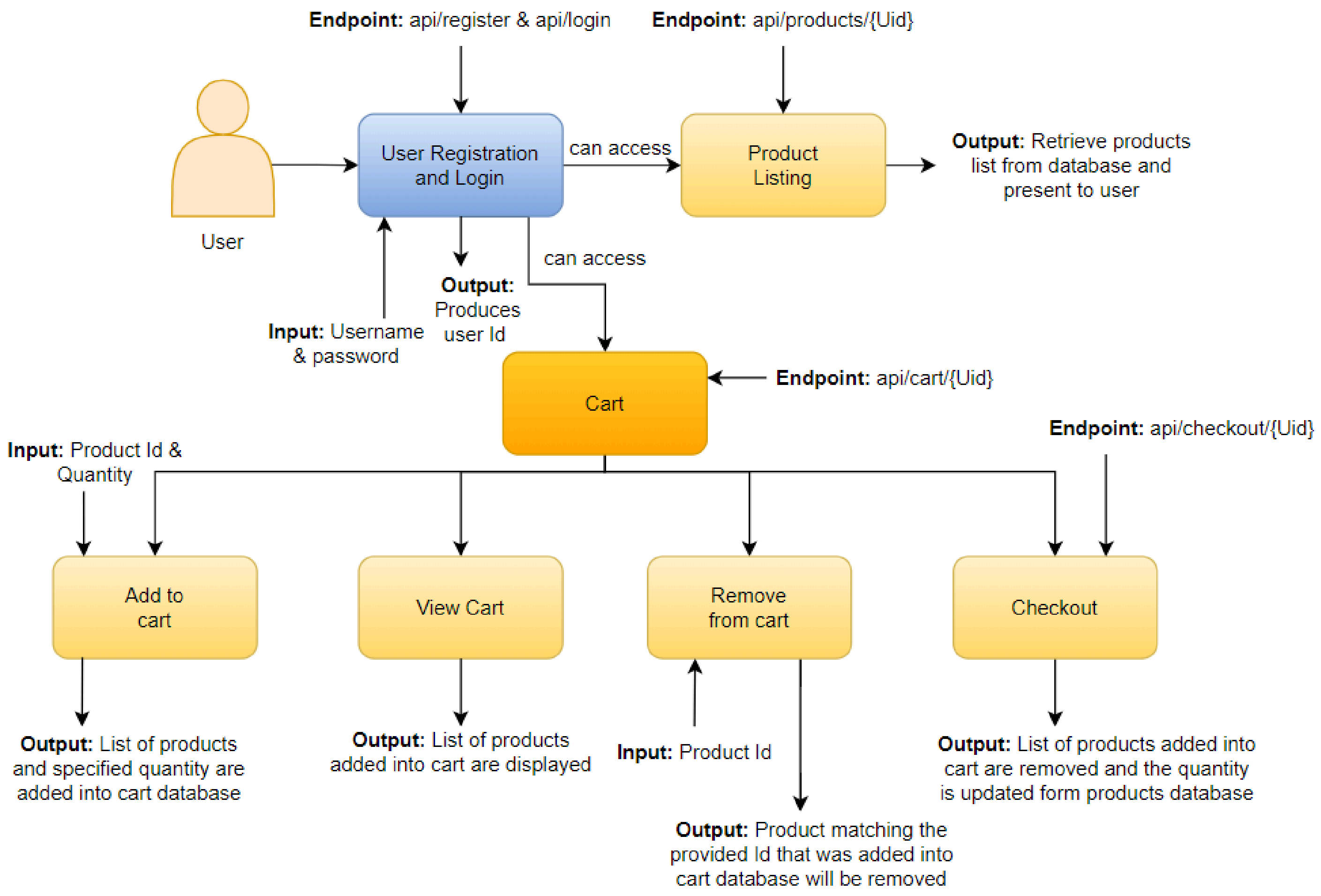
Repository Patterns

These are the design patterns which are commonly used in development to separate the data access logic for the database from the rest of the application allowing for a cleaner and modular codebase.
Initially it involves the creation of an interface which is used for data access operations allowing the methods declared inside this interface to be accessed by the application handling the https requests and responses.
The DbContext instances, representing connections to the database, are injected into repository classes via constructor injection. Within repository classes, the injected DbContext will allow the smooth operations with the database which includes executing queries, updating entity states, and persisting changes back to the database.

Controllers

They are responsible for routing incoming HTTP requests to the action methods (GET, POST, PUT, DELETE, etc), to which appropriate status code responses are represented. Controllers also handle content negotiation to determine the format of the responses. DTOs allow controllers to encapsulate the data returned by different action methods which allows them to provide a consistent API interface for the clients.

The functionality of the shopping cart involves:



Endpoint: api/register & api/login
Endpoint: api/products/{Uid}
User
User Registration and Login
can access
Product Listing
Output: Retrieve products list from database and present to user
can access
Output: Produces user Id
Input: Username & password
Cart
Endpoint: api/cart/{Uid}
Endpoint: api/checkout/{Uid}
Input: Product Id & Quantity
Add to cart
View Cart
Remove from cart
Checkout
Output: List of products and specified quantity are added into cart database
Output: List of products added into cart are displayed
Input: Product Id
Output: List of products added into cart are removed and the quantity is updated form products database
Output: Product matching the provided Id that was added into cart database will be removed

Figure c. Implementation of the shopping cart API and their functionalities

API Collection

The api collection involved in testing the shopping cart backend application are:

- **User Registration**
  - **Method:** POST
  - **URI:** https://localhost:{portNumeber}/api/register
  - **Input:** { "username":<username>, "password": <password>}
  - **Output:** Username and password are stored in database
- **User Login**
  - **Method:** POST
  - **URI:** https://localhost:{portNumeber}/api/login
  - **Input:** { "username":<username>, "password": <password>}
  - **Output:** Verify user using login using username and password, then provide the user Id for shopping

- **Product Listing**
    - **Method:** GET
    - **URI:** https://localhost:{portNumeber}/api/products
    - **Input:** { NULL }
    - **Output:** List of all products available for the user from database
- **Add To Cart**
    - **Method:** POST
    - **URI:** https://localhost:{portNumeber}/api/cart/{Uid}
    - **Input:** { "Product Id": <product id to be added into car>, "Set Quantity":<quantity of that product>}
    - **Output:** The products listed by user are added into cart providing relationship between user Id and product Id in the same row.
- **View Cart**
    - **Method:** GET
    - **URI:** https://localhost:{portNumeber}/api/cart/{Uid}
    - **Input:** { NULL }
    - **Output:** Based on user Id from HTTP request retrieve all the products from the database and present to user
- **Remove From Cart**
    - **Method:** DELETE
    - **URI:** https://localhost:{portNumeber}/api/cart/{Uid}
    - **Input:** {"Product Id": <product Id> }
    - **Output:** Match the user id from http request and product id from the users input, then delete from cart database matching the specific cart id that stores the product to be removed for that specific user.
- **Checkout**
    - **Method:** POST
    - **URI:** https://localhost:{portNumeber}/api/cart/{Uid}
    - **Input:** { NULL }
    - **Output:** Remove all the products from the cart matching the user id present with http request, then update the quantity of those specific products that were removed from the users cart in the product database.

Conclusion

In conclusion, the development of the shopping cart API involved the utilization of various concepts and methodologies to create a robust and functional solution. By leveraging technologies such as ASP.NET Core, Entity Framework, and RESTful principles, it was successful to implement an API that facilitates seamless online shopping cart experiences. The

API documentation provided detailed information about the available endpoints, request parameters, and response formats, enabling easy integration with client applications. The implementation logic followed industry best practices, ensuring code maintainability, scalability, and performance. Overall, this shopping cart API project demonstrates the ability to conceptualize, design, and implement a solution that addresses real-world business requirements.