Shopimax API

API Version 1

API Overview

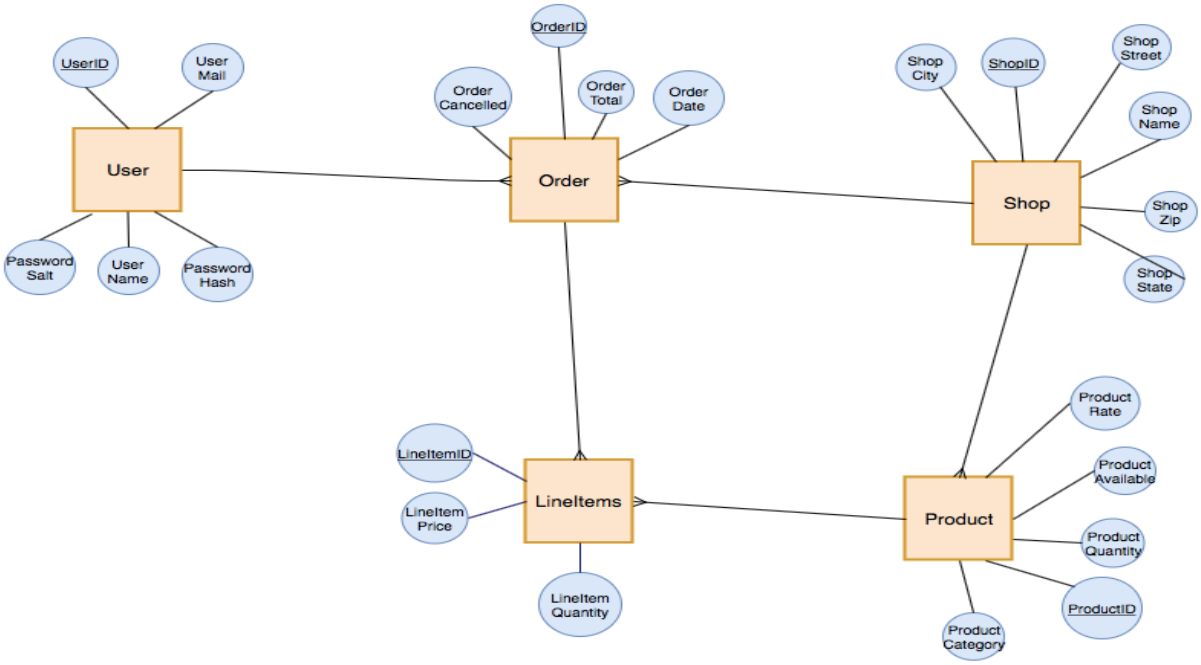
Shopimax API is an ecommerce API that is built on “code-first approach” in the latest ASP.NET Core Framework which provides endpoints to perform CRUD operations on commodities such as products, line items, shops and orders.

http://<host>:<port>/api/

Basic Organization of API

Technical Specification

S No	Component	Technology
1	Backend Technology	ASP.NET Core Framework 2.1
2	Middleware	JWT Token Security
3	Database	SQLite
4	ORM	Entity Framework Core
5	Unit Testing	Moq, NUnit
6	IDE	Visual Studio Code



ER Diagram

- An ASP.NET Core Web API project “Shopimax.API” is created
- Five entities such as Order, Shop, User, Products and LineItem are created as DBsets under “Models” folder.
- Once the entities are created, relationships are established among them. Migration files “AddedModels” are created and database is updated with this Db Migration file. The Migrations are present in “Migrations” Folder
- A new Shopimax.db file is generated from this migration instance.
- Repository pattern is implemented – AuthRespository and ShopiMaxRespository is responsible for business logic operations.
- Data Transfer Objects(DTOs) are created that transfers data from Controller to data access layer.

Prerequisites

Dotnet Core SDK 2.1.4 is required for running this application. The application runs on the latest version of ASP.NET CORE environment. Dotnet CLI is also required for the application. Though Dotnet consists of built in package for SQLite. DB Browser for SQLite can also be installed to view data tables.
“dotnet restore” CLI command need to be run to install all dependencies.

Seeding data

When the application is started, Startup.cs file is run. Services such as Mappers, JWT Bearers, CORS Support, Repository are added in the file. Additionally, sample data need to be seeded the first time when the application is run. “dotnet run” CLI command is executed.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env, Seed seeder)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        //app.UseHsts();
    }
    seeder.SeedUserData();
    seeder.SeedShopData();
    seeder.SeedProductData();
    seeder.SeedOrderData();
    seeder.SeedLineItemData();

    app.UseCors(x => x.AllowAnyOrigin().AllowAnyHeader().AllowAnyMethod());
    app.UseAuthentication();
```

Once the application is run the first time, seeder methods need to be commented so that it would not be executed again.

Resources

The entities are exposed as resources according to their types.
e.g. the 'datasets' resources are accessible from http://localhost:5000/api/<Resource>.

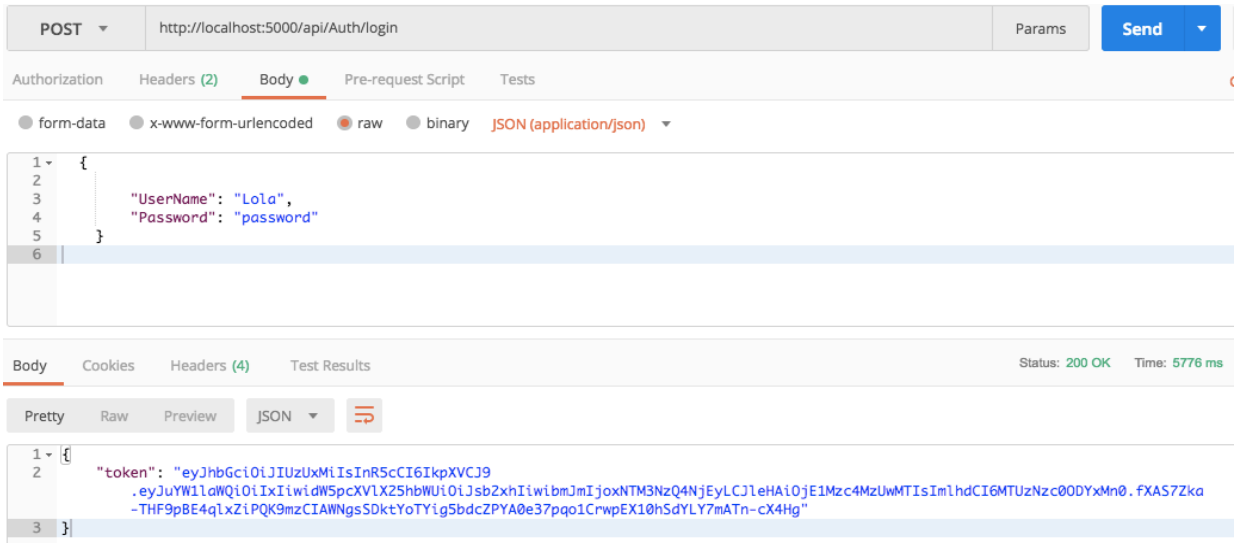
Resource	Description	Route
Auth	User Login and Register	/Auth
User	Get Users by orders and id, delete user account.	/User
Order	CRUD operations on Order Entity	/Order
LineItem	CRUD operations on LineItem Entity	/LineItem
Product	CRUD operations on Product Entity	/Product
Shops	CRUD operations on Shops Entity	/Shops

Endpoint Description

Authorization

Auth Controller is responsible for establishing security to the Shopimax API. JWT Token authorization technique is used as a middleware.

- In the event of successful login, a token (scripted with HMACSHA 512 algorithm) is sent back which is employed for accessing Shopimax API endpoints.



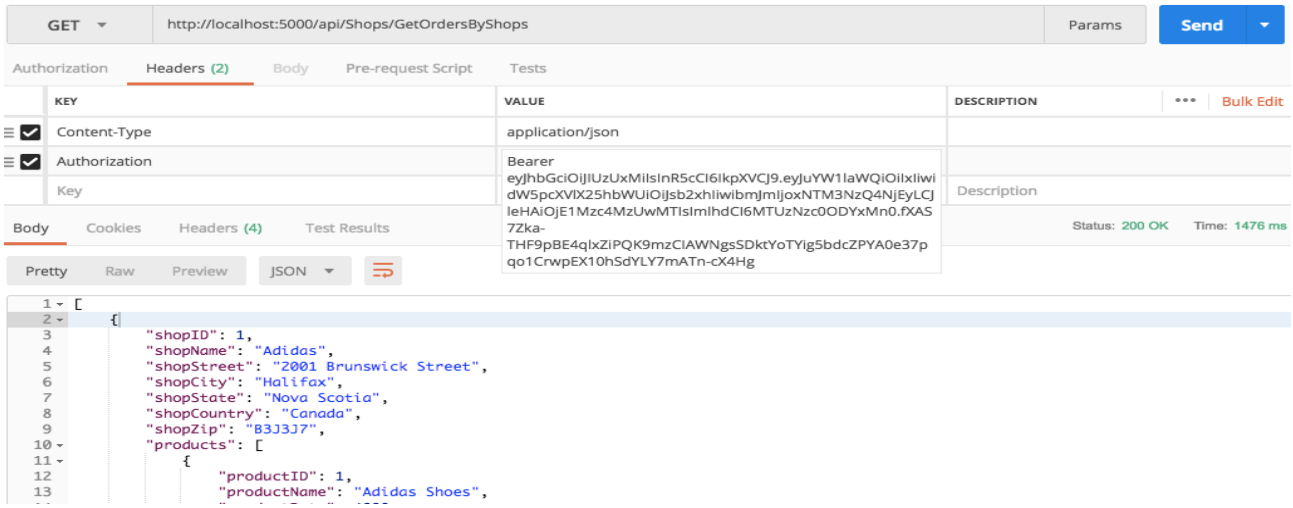
- An expiry period of 1 day is set by default. Username, Password, Email are set as required fields. For development purpose, distinct user name check is implemented.

HTTP verb	Route	Description
Post	/Auth/register	User Register after validation
Post	/Auth/login	User Login

Shops

Shop Controller is responsible for performing CRUD operations on Shop entity. Upon successful authorization, the user would be able to add/delete/select/update shop details.

- In the event of successful login, a token (scripted with HMACSHA 512 algorithm) is sent back. The token is added in the header of the request as Authorization key. Since JWT Bearer is used, [bearer <space> token] format is followed.



HTTP verb	Route	Description
Post	/Shops/add	Add a shop
Post	/Shops/edit	Update shop details
Get	/Shops/	Get all shops
Get	/Shops/{id}	Get a particular shop
Delete	/Shops/{id}	Delete a particular shop
Get	/Shops/GetProductsByShops	Get Shops with corresponding products
Get	/Shops/GetOrdersByShops	Get Shops with corresponding products and orders.

Products

Product Controller is responsible for performing CRUD operations on Product entity. Upon successful authorization, the user would be able to add/delete/select/update product details. Shop – Product has a cascading relationship. If shop is deleted, then corresponding products are also removed.

HTTP verb	Route	Description
Post	/Product/add	Add a Product
Post	/Product/edit	Update Product details
Get	/Product/	Get all Product
Get	/Product/{id}	Get a Product
Delete	/Product/{id}	Delete a particular Product

Line Items

Line Item Controller is responsible for performing CRUD operations on Line Item entity. Upon successful authorization, the user would be able to add/delete/select/update Line Item details. Line Item – Order has a cascading relationship while Line Item – Product has a restrictive relationship. If product is deleted, line item record would not be deleted. While an order is deleted, the corresponding line items are removed.

HTTP verb	Route	Description
Post	/LineItem/add	Add a LineItem
Post	/LineItem/edit	Update LineItem details
Get	/LineItem/	Get all LineItems
Get	/LineItem/{id}	Get a LineItem
Delete	/LineItem/{id}	Delete a particular LineItem

Orders

Order Controller is responsible for performing CRUD operations on Order entity. Upon successful authorization, the user would be able to add/delete/select/update Order details. User– Order has a cascading relationship while Order – Shop has a restrictive relationship. If a shop is deleted, the corresponding orders would not be deleted. While a user is deleted, the corresponding orders are also removed. Order Dto with order total (sum of all corresponding line items) is sent as output.

HTTP verb	Route	Description
Post	/Order/add	Add an Order
Post	/Order/edit	Update Order details
Get	/Order/	Get all Orders
Get	/Order/{id}	Get an Order
Delete	/Order/{id}	Delete a particular Order
Get	/Order/GetLineItemsByOrders	Get Orders with corresponding lineitems

Users

User Controller is responsible for performing Select and delete operations on User entity.

HTTP verb	Route	Description
Get	/User/	Get all Orders
Get	/User/{id}	Get an Order
Delete	/User/{id}	Delete a particular Order
Get	/Order/GetOrdersByUsers	Get Users with corresponding orders

Unit Testing

- Unit test cases are written using Moq and NUnit frameworks. Since the application is developed in a modular architecture, a new project ShopimaxTest is created.
- “dotnet test” CLI command is used to run the tests present in the test project.

Highlights of the project

- The web API is developed in the latest ASP.NET Core technologies.
- The application is developed in a code – first approach. Entities are created using Entity Framework Core technologies. Migration files are created for data preservation.
- The API is secured using JWT authorization techniques.
- Data Transfer Objects, MVC, and Repository patterns are used. Mappers are used to map Entity and Dto objects.
- Unit test cases are written for several API functionalities.
- Docker images are created.