In this report, we present the Data Dictionary for three key entities in our system: Sale, Product and Customer. These documents provide a complete overview of how sales, product and customer related data is stored and managed in our database.

The Data Dictionaries for the entities Sale, Product and Customer are detailed below, providing a complete description of the data structure of these entities.

**Note 1 :** We do not present a data dictionary for the User table since we are using Identity and the user has the same properties as the user identity.

**Note 2** : the application has swagger and it explains the model that expects each endpoint in case it is expected.

**Note3 :** El usuario Admin puede realizar todo tipo de operaciones, pero el usuario Standard solo puede ver los datos.

**Data Dictionary for the Sale Class**

| **Field Name** | **Description** | **Data Type** | **Length/Constraint** | **Default Value** |
| --- | --- | --- | --- | --- |
| IdSale | Identifier of the sale | GUID | - | Auto-incremental |
| IdCustomer | Identifier of the customer | GUID | Foreign Key to **Customers** table | - |
| IdProduct | Identifier of the product | GUID | Foreign Key to **Product** table. | - |
|  |  |  |  |  |
| Quantity | Quantity of products in the sale | Integer | Minimum: 1 | - |
| Date | Date of the sale | Date | - | Current Date |

**Data Dictionary for the Product Class**

| **Field Name** | **Description** | **Data Type** | **Length/Constraint** | **Default Value** |
| --- | --- | --- | --- | --- |
| IdProduct | Identifier of the product | GUID | - | Auto-incremental |
| Description | Description of the product | String | Maximum: 100 characters | - |
| Price | Price of the product | Decimal | Minimum: 0.01, Maximum: 999999.99 | - |

**Data Dictionary for the Customer Class**

| **Field Name** | **Description** | **Data Type** | **Length/Constraint** | **Default Value** |
| --- | --- | --- | --- | --- |
| IdCustomer | Identifier of the customer | GUID | - | Auto-incremental |
| Name | First name of the customer | String | Maximum: 50 characters | - |
| LastName | Last name of the customer | String | Maximum: 50 characters | - |
| Email | Email address of the customer | String | Maximum: 50 characters | - |
| Address | Address of the customer | String | Maximum: 100 characters | - |
| City | City where the customer resides | String | Maximum: 50 characters | - |

**WEB API Project Structure**

The API was built in the .Net framework and has 4 projects:

**ECOMMERCE:** This is a Web API that has been configured to use JWT in the startup.cs file is called ConfigureJwtAuthentication to configure JWT authentication (JSON Web Tokens). This includes obtaining the token's secret key from the application configuration, creating a SymmetricSecurityKey with the secret key and configuring the token's validation parameters, then UseJwtBearerAuthentication enables JWT authentication in the application, using the secret key and validation parameters configured in the previous step. This allows the application to verify and authenticate the JWT tokens that users present when accessing protected resources (cookie validation settings are left enabled).

Swagger has been configured to present the endpoints and documentation for each.

**Note:** This is a Web Api that has been created with visual studio scafolding and this is why some files such as views, scripts and others I will not explain.

In the Controller folder you will find the 5 controllers of the application, all except HomeController are the ones we are interested in, inside each one of them you will find the endpoints we are working with.

AccountController uses usermanager and from there we are going to manage everything related to the users of the APP and the Login, the others receive as dependency injection the repositories (later I will talk about them) and what we do is to call those repositories using async and await to avoid crashes in our server.

In the Helper folder there is a UnityResolver class that uses the IoC container called Unity to manage and resolve these dependencies, so we can inject them into the controllers.

**Common**: Es ua libreria de clases que he creado con la idea de tener los archivos comunes en una misma libreria, solo tenemos un enumerador UserRole que contiene los roles que acepta un Usuario. 0 Admin, 1 Standard.

**Models:** is another class library that contains classes and components that define how data is stored, retrieved and manipulated, as well as the logic that governs the behaviour of the application. That is, Models, Dto and the repository interface.

**Infrastructure:** It is another library of classes where we have the repositories that serve us for the operations with the database.

**Aplicacion Web**

La aplicacion Web basicamente son archivos html, js y css no estoy utilizando Angular o React debido a un problem que tengo en mi maquina con npm. El archivo principal es Index.hmtl, es el archivo que contiene el login y que se encarga de iniciar el flujo del frontend.

**The console client follows an MVC architecture.**

We have some forms that we use to log in to the APP, after logging in we will see a menu and from there we can go to the CRUD of the entities, Users, Customers and Products.

We have a helper folder where we have a TokenManager class that helps us to manage the token and we have a Util class where there are some methods that help us to save code and reuse.

Model: It is a library of classes where we have the interfaces of the repositories and the Dto of the application.

Services: Another class library where we implement the interfaces of the model and communicate from there with our API.

this is the diagram of our database

