WEB DEVELOPER PRACTICAL TEST

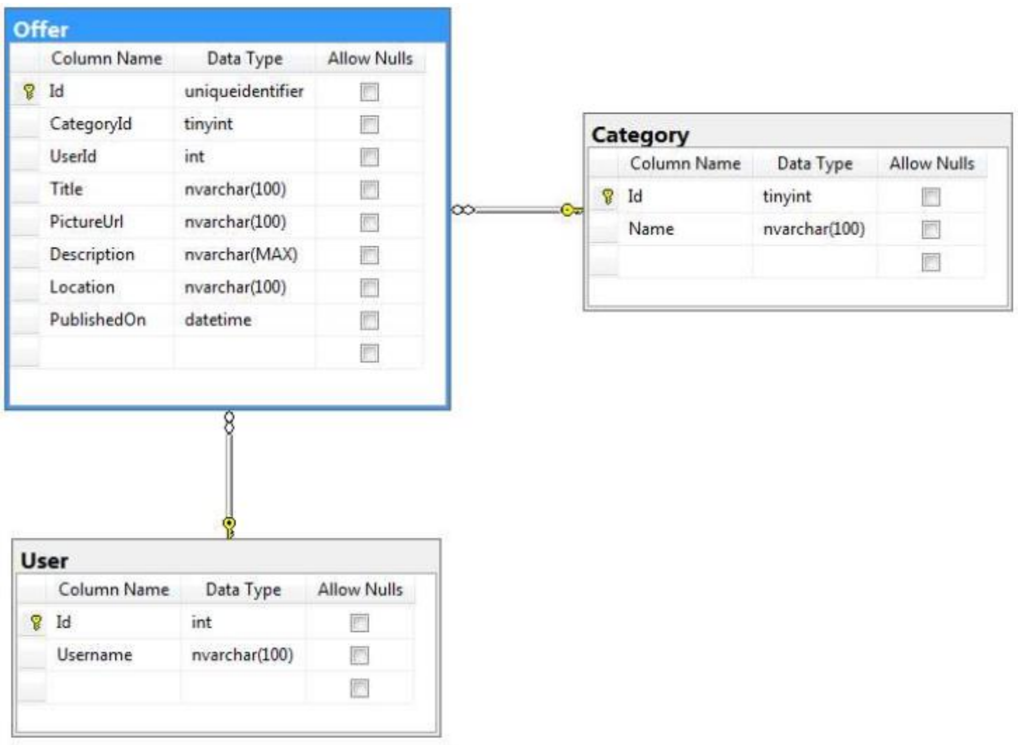
Read first: Pre-requirements

- GitHub account
- Git
- .NET CORE 6 SDK
  https://dotnet.microsoft.com/en-us/download/dotnet/6.0
- VS Code (or any other IDE you can work on)
  https://code.visualstudio.com/download
- Angular v16, Node.js, npm
  https://nodejs.org/es/download
- SQL Server 2016 Express or higher
  https://www.microsoft.com/en-us/sql-server/sql-server-downloads
- SQL Server Management Studio
  https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms.
- MongoDB Community Edition (v6.0)
- Mongo Shell
- Mongo Compass
- Mongo Command Line Database Tools

ROSEN Technology & Research Center (RTRC) requires a simple marketplace for its employees in order to publish items for resale, offers of services or inquiries for specific items. It was decided that a small responsive website should be developed using Angular, .NET 6, and SQL Lite.

You are presented with a solution started by another developer and the following design:

- A SQLite database attached into the project with the following model ready to be queried:



Offer

	Column Name	Data Type	Allow Nulls
🔑	Id	uniqueidentifier	☐
	CategoryId	tinyint	☐
	UserId	int	☐
	Title	nvarchar(100)	☐
	PictureUrl	nvarchar(100)	☐
	Description	nvarchar(MAX)	☐
	Location	nvarchar(100)	☐
	PublishedOn	datetime	☐
			☐

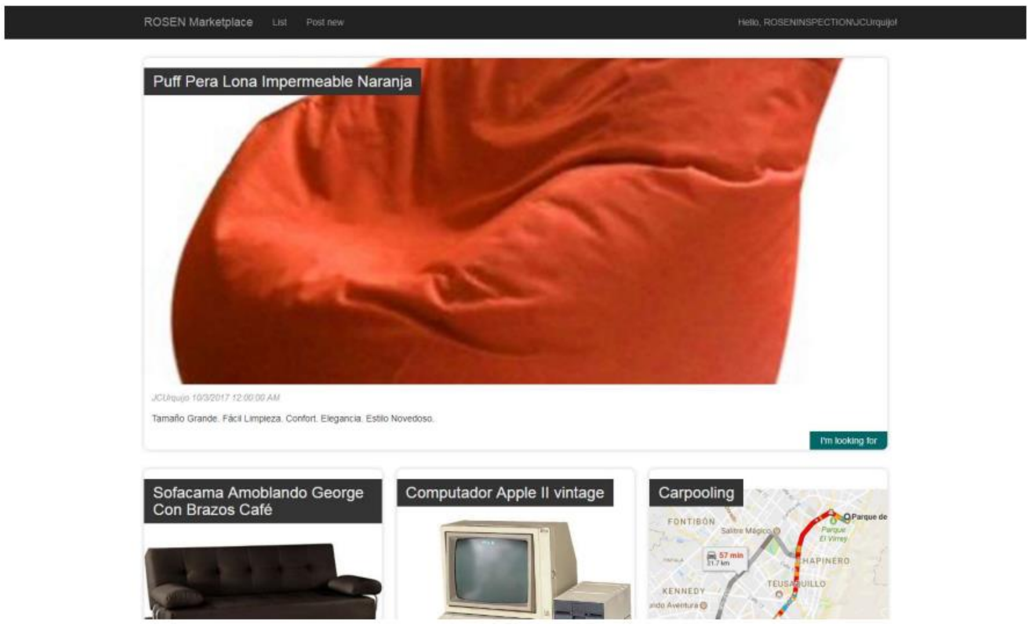
Category

	Column Name	Data Type	Allow Nulls
🔑	Id	tinyint	☐
	Name	nvarchar(100)	☐
			☐

User

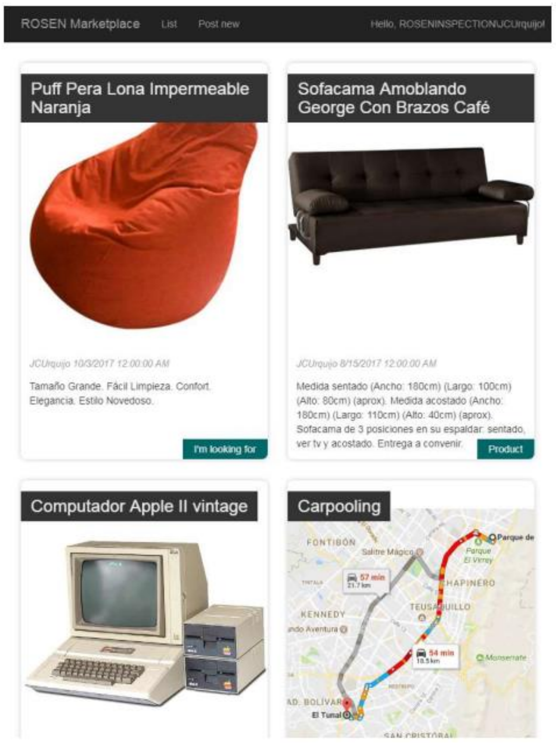
	Column Name	Data Type	Allow Nulls
🔑	Id	int	☐
	Username	nvarchar(100)	☐
			☐

- The UX team designed the following mockup:


ROSEN Marketplace
List
Post new
Hello, ROSENINSPECTION\JCUrquijo!
Puff Pera Lona Impermeable Naranja
JCUrquijo 10/3/2017 12:00:00 AM
I'm looking for
Sofacama Amoblando George Con Brazos Café
Computador Apple II vintage
Carpooling

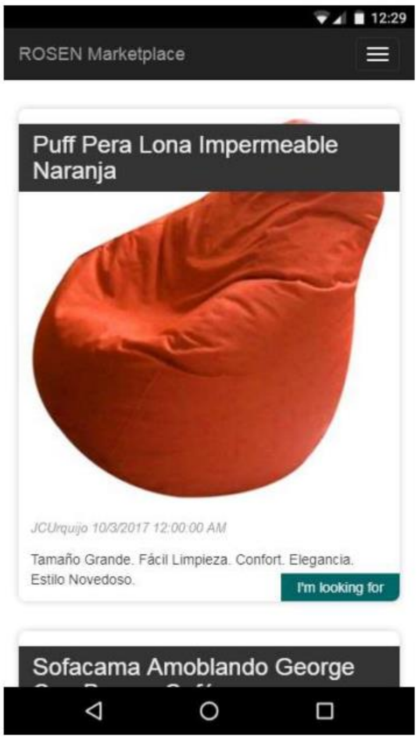
Tamaño Grande. Fácil Limpieza. Confort. Elegancia. Estilo Novedoso.

Desktop and larger Tablets in landscape mode. (width >= 992px)


ROSEN Marketplace
List
Post new
Hello, ROSENINSPECTION\JCUrquijo!
Puff Pera Lona Impermeable Naranja
JCUrquijo 10/3/2017 12:00:00 AM
I'm looking for
Sofacama Amoblando George Con Brazos Café
JCUrquijo 8/15/2017 12:00:00 AM
Product
Computador Apple II vintage
Carpooling

Tamaño Grande. Fácil Limpieza. Confort. Elegancia. Estilo Novedoso.

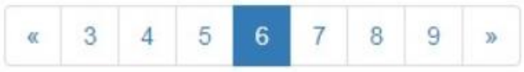
Medida sentado (Ancho: 180cm) (Largo: 100cm) (Alto: 80cm) (aprox). Medida acostado (Ancho: 180cm) (Largo: 110cm) (Alto: 40cm) (aprox). Sofacama de 3 posiciones en su espaldar: sentado, ver tv y acostado. Entrega a convenir.

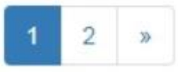

12:29
ROSEN Marketplace
Puff Pera Lona Impermeable Naranja
JCUrquijo 10/3/2017 12:00:00 AM
I'm looking for
Sofacama Amoblando George

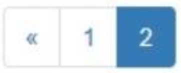
Tamaño Grande. Fácil Limpieza. Confort. Elegancia. Estilo Novedoso.

Tablets in Portrait mode (screen width >= 768px) Smartphones (screen width <768px)

- Pagination should look like so and non-existing links (i.e.: when there are too few pages) should be hidden:



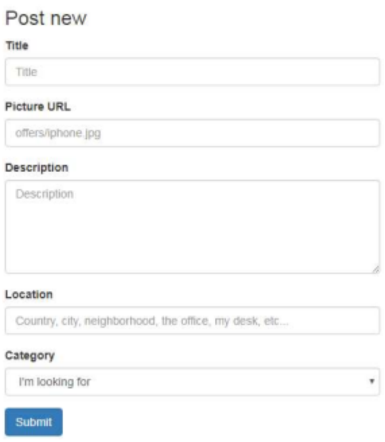




- The architect has decided that the pagination should be implemented in the frontend as the following class:

Page<T> <<class>>		
Items	Array<T>	Gets the items.
NextPageIndex	number	Gets the index of the next page or null if at the last page.
PageCount	number	Gets the page count.
PageIndex	number	Gets the index of the current page.
PreviousPageIndex	number	Gets the index of the previous page or null if at the first page.
Ctor(items, pageIndex, pageCount)		For parameter types, see the types of the homonymous properties.
GetNextPageIndexes()	Array<number>	Gets at most three next page indexes (only the ones that exists).
GetPreviousPageIndexes()	Array<number>	Gets at most three previous page indexes (only the ones that exists)

- Any user should be able to post a new offer using the form at the bottom of the page.



Post new

Title: Title

Picture URL: offers/iphone.jpg

Description: Description

Location: Country, city, neighborhood, the office, my desk, etc...

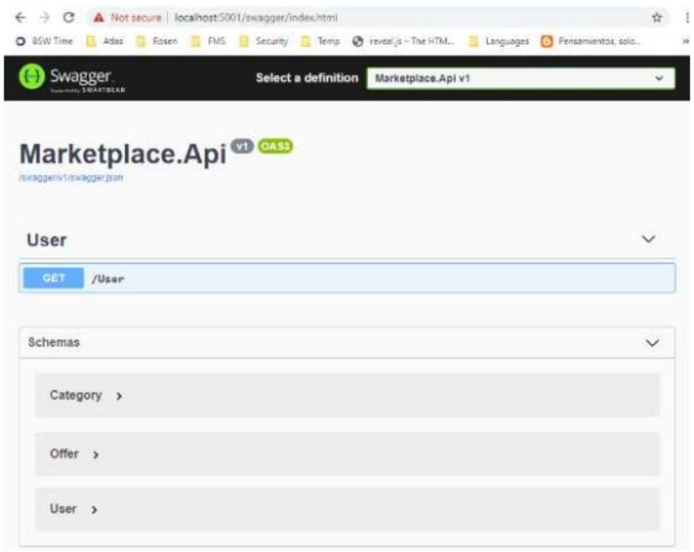
Category: I'm looking for

Submit

- Another developer on your team already started the project and did some work on the initial solution inside the given repository, which includes the API project and an Angular Application:

  The API Solution can be found in `./Marketplace/Api` and contains the following projects:

  1. Marketplace.Api (REST API Endpoints Controllers)
  2. Marketplace.Core (Solution Contracts)
  3. Marketplace.Bl (Bussiness Logic Contract Implementation)
  4. Marketplace.Dal (Data Access Layer)



localhost:5001/swagger/index.html

Swagger — Select a definition: Marketplace.Api v1

Marketplace.Api v1 OAS3

/swagger/v1/swagger.json

User

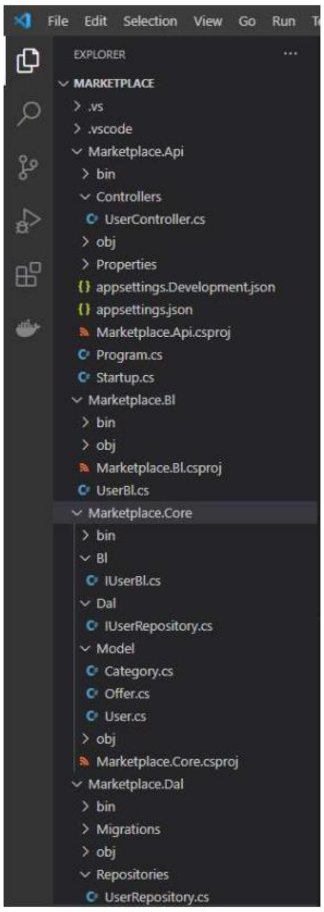
GET /User

Schemas

Category >

Offer >

User >

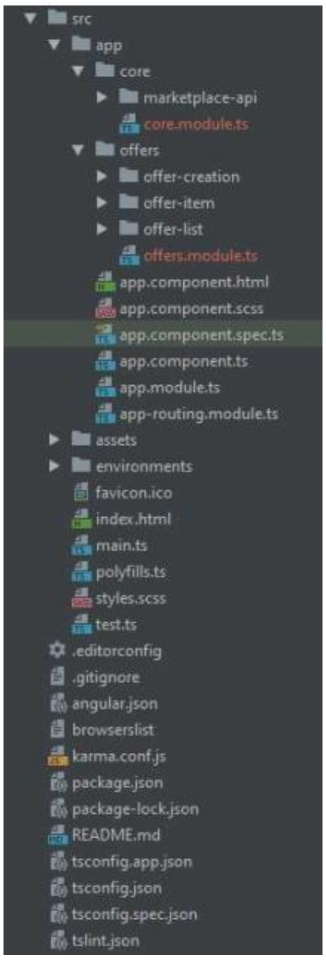


```
EXPLORER
MARKETPLACE
 > .vs
 > .vscode
 v Marketplace.Api
   > bin
   v Controllers
     UserController.cs
   > obj
   > Properties
   appsettings.Development.json
   appsettings.json
   Marketplace.Api.csproj
   Program.cs
   Startup.cs
 v Marketplace.Bl
   > bin
   > obj
   Marketplace.Bl.csproj
   UserBl.cs
 v Marketplace.Core
   > bin
   v Bl
     IUserBl.cs
   v Dal
     IUserRepository.cs
   v Model
     Category.cs
     Offer.cs
     User.cs
   > obj
   Marketplace.Core.csproj
 v Marketplace.Dal
   > bin
   > Migrations
   > obj
   v Repositories
     UserRepository.cs
```

- The Angular application can be found in `./Marketplace/WebApp` and has the following structure:

  1. core module (contains services used in all the application)
  2. offers module (contains all the components fort he offers)
  3. assets (contains the assets that need to be served as individual elements like images).

  The other developer already did the HTML for the list, the pagination and form which you'll find in offers module and an initial implementation of the `Page<T>` class, found in the core module.



```
▼ src
  ▼ app
    ▼ core
      ▶ marketplace-api
        core.module.ts
    ▼ offers
      ▶ offer-creation
      ▶ offer-item
      ▶ offer-list
        offers.module.ts
      app.component.html
      app.component.scss
      app.component.spec.ts
      app.component.ts
      app.module.ts
      app-routing.module.ts
  ▶ assets
  ▶ environments
    favicon.ico
    index.html
    main.ts
    polyfills.ts
    styles.scss
    test.ts
  .editorconfig
  .gitignore
  angular.json
  browserslist
  karma.conf.js
  package.json
  package-lock.json
  README.md
  tsconfig.app.json
  tsconfig.json
  tsconfig.spec.json
  tslint.json
```

Your task:

1. Implement the design proposed by the UX team.

2. Implement the `Page<T>` class by first creating a set of unit tests. Make sure you test the behavior of the `Page<T>` class when in the following states:

   - Only one page is available
   - There are many pages (10 at least) and you are at the first page.
   - There are many pages (10 at least) and you are at the middle page.
   - There are many pages (10 at least) and you are at the last page.
   - Any other state you find interesting

3. Implement the `offer` components so that all the expected behaviors work correctly. Keep in mind that you have 200 thousand Offers' rows and could be many more and it is ideal to display the information in the most optimal way possible, please consider the best practices for both front-end and back-end side.

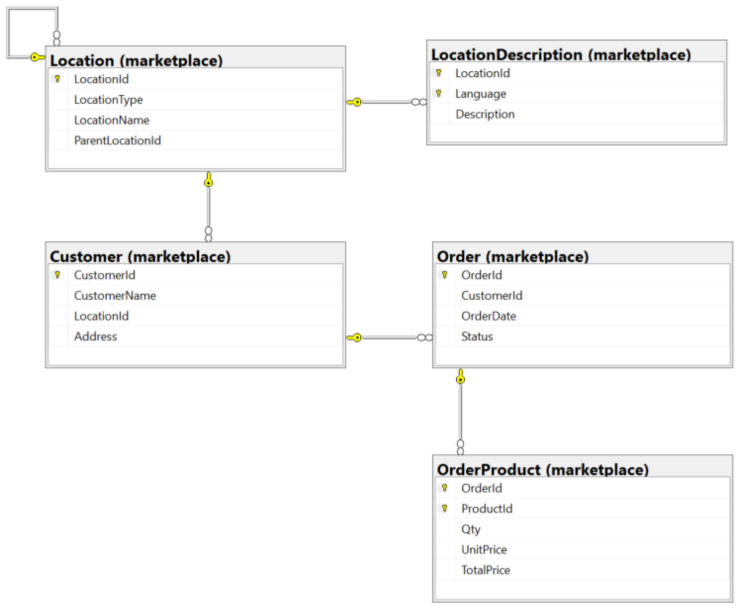
4. Take into account that the `Offer` table has a reference to the `User`. If the current user doesn't exists, the user shall be created before saving the offer.

5. Implement the API Controllers and the contracts you consider for it.

6. Add Unit tests when you consider them necessary, this will be taken into account for your score as additional points!

A second data model for our Marketplace would like to be evaluated.

For the data layer of our Marketplace we want to implement the following model:



For the following exercises please consider the scripts located on DB Test folder.

1. Create the `Sales` database in SQL Server with basic `Order` information, using the script `1. SQL_marketplace.sql`. Verify that after the execution of the script this is the model of the database.

2. We need to perform operations of creation, update and deletion of records in the `OrderProduct` table. To achieve this, you need to complete the implementation of the stored procedure `marketplace.WriteOrderProduct`. Please note the following conditions:

   - Do not modify the parameters already defined.
   - It is possible to add, modify or delete records from the `OrderProduct` table only if the status of its parent in the `Order` table is `Open`.
   - The values for `ProductId`, `Qty`, `UnitPrice` and `TotalPrice` must be greater than zero.
   - Implement the validations that you consider necessary.
   - Use the `MERGE` statement to perform create, update and delete.

3. We also need to perform create, update and delete operations on the `Order` table. Help us to complete the implementation of the stored procedure `marketplace.WriteOrder` taking into account the following conditions:

   - Do not modify the parameters already defined.
   - Use the `MERGE` statement to perform the creation, update and deletion.
   - When a new `Order` is created, the identifier assigned to the newly created order must be returned.
   - Use `marketplace.WriteOrderProduct` if you consider it necessary.
   - Implement the validations that you consider necessary.

4. Now, it is necessary to be able to query the information of the tables `Order` and `OrderProductComplete`. To achieve this, you need to complete the implementation of the stored procedure `marketplace.ReadOrder`, taking into account the following conditions:

   - Do not modify the parameters already defined.
   - Implement the validations that you consider necessary.
   - This stored procedure must support at least the following 3 cases:

     A. Obtain through a single query the `OrderId`, `CustomerId`, `CustomerName`, `OrderDate` and `Status` of one or several `Orders`, given the `CustomerId` and/or `OrderId` and/or `Status`.

        - Name of the result: `Order-Header`.

- It implements the indexes it considers appropriate.
- Through this query we should be able to obtain:
    - All Orders
    - All the Orders for a specific customer
    - All Orders in a specific status for a specific client
    - All Orders in a specific status for all customers

B. Obtain through a single query the `OrderId`, `ProductId`, `Qty`, `UnitPrice` and `TotalPrice` of one or several `Orders`, given the `CustomerId` and/or `OrderId` and/or `Status`.

- Result Name: `Order-Detail`.
- It implements the indexes it considers appropriate.
- Through this query we should be able to obtain:
    - Details of all Orders
    - Details of all the Orders for a specific customer.
    - Details of all Orders in a specific status for a specific customer.
    - Details of all Orders in a specific status for all customers.

C. Get in `JSON` format, through a single query, the historical information of `Orders` along with their `Details`, given the `CustomerId` and/or `OrderId` and/or `Status`.

- Name of the Result: `Order-History-Json`.
- It implements the indexes it considers appropriate.
- Through this query we should be able to obtain:
    - All Orders and their Details
    - All Orders and their Details for a specific customer
    - All Orders and their Details in a specific state for a specific client
    - All Orders and their Details in a specific status for all customers.

The result in `JSON` format must have the structure shown in `Annex 1`. Please note that:

- `_id` corresponds to the Order #:
- The value for `CustomerLocationHierarchy` must correspond according to the `CustomerLocation` hierarchy of the `Order` and depending on the `Language` indicated in the `@Language` parameter.
- `ProductsCount` corresponds to the quantity of products included in the `Order`.
- `TotalOrder` corresponds to the sum of the selling price of all the products in the `Order`.

💪 If you have experience using **tSQLt** for unit test creation, you can use it to validate the creation, update and deletion in the above points. Otherwise, please share with us evidence of the execution of each stored procedure along with the result obtained.

Finally, our analytics area is interested in taking advantage of the data from our Marketplace but in order to use it in analytics tools they require us to deliver the information through MongoDB.

1. Environment preparation

   - Create a MongoDb database named `Marketplace`.
   - Create the product collection and load the information provided in the `product.json` file.
   - Create the `orderHistory` collection and load the information provided in the `orderHistory.json` file.

2. Perform a query through which to obtain the `ID` and `Name` of the 3 best selling products during January 2023, taking into account that:

   - For each `product` show the quantity of products sold, and the sum of sales value.
   - The result should be presented in order x total sales price from highest to lowest.
   - Create the indexes you consider appropriate.

   The expected result is the following, according to the information loaded in step 1:

```
[
    {
        "TotalSoldItems": 9,
        "TotalPrice": 90,
        "ProductId": 66,
        "ProductName": "Divi Engine String Bag (Big Logo)"
    },
    {
        "TotalSoldItems": 4,
        "TotalPrice": 80,
        "ProductId": 114,
        "ProductName": "WordPress Tee - Small"
    },
    {
        "TotalSoldItems": 7,
        "TotalPrice": 35,
        "ProductId": 117,
        "ProductName": "Mens Divi Hoodie"
    }
]
```

Share with us evidence of the process, instruction, method, command used, etc., and where applicable, the result obtained.

ANNEX 1. EXAMPLE OF RESULT FOR ORDERS AND THEIR DETAILS HISTORICAL INFORMATION

```
[
    {
        "_id": 1,
        "OrderDate": "2023-01-15T00:00:00",
        "CustomerId": 1,
        "CustomerName": "Tom Cruise",
        "CustomerLocationCity": "New York",
        "CustomerLocationHierarchy": "America | United States | New York",
        "CustomerAddress": "1100 Main Street 248 PH98, Manhattan, NYC",
        "Products": [
            {
                "ProductId": 66,
                "Qty": 8,
                "UnitPrice": 10.0,
                "TotalPrice": 80.0
            },
            {
                "ProductId": 100,
                "Qty": 3,
                "UnitPrice": 10.0,
                "TotalPrice": 30.0
            },
            {
                "ProductId": 117,
                "Qty": 7,
                "UnitPrice": 5.0,
                "TotalPrice": 35.0
            }
        ],
        "ProductsCount": 3,
        "TotalOrder": 145.0
    },
    {
        "_id": 2,
        "OrderDate": "2023-01-20T00:00:00",
        "CustomerId": 3,
        "CustomerName": "Shakira Mebarak",
        "CustomerLocationCity": "Barcelona",
        "CustomerLocationHierarchy": "Europe | Spain | Barcelona",
        "CustomerAddress": "Ciudad Diagonal, Esplugas de Llobregat",
        "Products": [
            {
                "ProductId": 132,
                "Qty": 1,
                "UnitPrice": 14.0,
                "TotalPrice": 14.0
            }
        ],
        "ProductsCount": 1,
        "TotalOrder": 14.0
    },
    {
        "_id": 3,
        "OrderDate": "2023-01-30T00:00:00",
        "CustomerId": 1,
        "CustomerName": "Tom Cruise",
        "CustomerLocationCity": "New York",
        "CustomerLocationHierarchy": "America | United States | New York",
        "CustomerAddress": "1100 Main Street 248 PH98, Manhattan, NYC",
        "Products": [
            {
                "ProductId": 66,
                "Qty": 1,
                "UnitPrice": 10.0,
                "TotalPrice": 10.0
            },
            {
                "ProductId": 114,
                "Qty": 4,
                "UnitPrice": 20.0,
                "TotalPrice": 80.0
            }
        ],
        "ProductsCount": 2,
        "TotalOrder": 90.0
    },
    ...
]
```