Marcus Logan, Prentiss Bryant

KMarket Repository

**Introduction:**

The KMarket Repository is a project designed to loosely simulate a local grocery store (in this case, Kroger), by incorporating two distinct tables of hot meals sold at the Café and regular items you can purchase on the shelves, and two respective tables for tracking orders placed on quantities of those items and meals. This way, we meet the minimum table requirement as well as the foreign-key relationship also needed to satisfy the MVP.

We also chose this project as a way to do something very modular and scalable in scope. While some pieces of the database may need to be refactored depending on future design specifications, the current design lends itself to more sophisticated implementations, such as actual item tracking (details on an item from its creation to the time it leaves the store) and a unified ordering system that can handle multiple types of objects being bought in any given transaction.

**Instructions for Local Use:**

Given the nature of the assignment, we assume a few steps must be carried out in a specific order so that this project can be successfully ran on any given local machine. We assume you have a compatible version of Visual Studio installed on your machine (2019), and that your computer is a Windows machine with nothing too special in terms of general use that would be hard to account for.

First, clone the repository or download the ZIP file from GitHub.com. If you chose to download the ZIP, extract the zipped folder before attempting to interact with the files inside of it.

Once you have cloned/extracted the KMarket repository (possibly submitted as KMarketFinal), you need to open the Solution file provided.

This should open the same version of the solution that we see in your Visual Studio program. I would recommend going to Build -> Rebuild Solution to make sure the compilation files match your machine. You should then be able to press the green Run button (possibly labeled IIS Express [yourbrowser]) to run the program.

This will open the program inside of a browser on your machine. Once it is running there, you can now interact with the API.

We used PostMan to handle all of our API requests and recommend doing the same. Below we have listed the basic CRUD functionality for each table.

The general use pattern should be:

1. Make an account. To do so, open PostMan and create a new request. Make it a POST request, and have the address be <https://localhost:[XXXXX]/api/Account>. Then, make your Body Parameters the x-www-form-urlencoded, and have them be Email, Password, and ConfirmPassword, putting in relevant information for each field.
   1. After making your account, generate your token. Make a new request in PostMan and have it be <https://localhost:[XXXXX]/token>. Change the Body Parameters to x-www-form-urlencoded and then have grant-type: password, password, and username, using the same information for when you generated your account. This will produce a token for you, which you will include in every PostMan request in the Headers section in the field Authorization from now on.
2. After generating your account, and your token, you are now ready to interact with the tables. CRUD functionality exists for each, but since OrderMeals and OrderItems (tables used to track orders of meals and items independently) rely on pre-existing KCafeMeals and KGrocerItems, we strongly recommend having some Meals and Items already generated before attempting to do anything with the Orders.
3. CRUD functionality exists as expected and is listed in detail below. An example for creating a KCafeMeal, however, would look like this: Create a new PostMan request. Make it POST. Change the address to <https://localhost:[XXXXX]/api/KCafeMeal>. Make sure to include your token in the Authorization header, then in Body select x-www-form-urlencoded. Now, to post a KCafeMeal (basically creating the class type of a hot meal served at the Café), you would include the body parameters of Name, Price, Description, and Ingredients. The rest of the relevant information to a KCafeMeal, such as when it was created in the Repository and who created it are stored automatically by design. Once you have entered information for the Parameters, hit the Send button. You should receive a 200 OK message indicating the Meal was successfully created.
4. Do the same for a KGrocerItem (details for this are listed below. Similar to but different than a KCafeMeal). After you’ve done both, you can try creating Orders on the relevant Items. To do so, go to either <https://localhost:[XXXXX]/api/OrderMeal> for meals, or api/OrderItem for items. For placing an Order, it will be the same as posting elsewhere but instead you will provide either an ItemID or a MealID, and the Quantity that was purchased. The rest of the information is handled by the system.
5. Later, once you have data, you can view it by calling the relevant GET method, as listed below. If you want to update any data, this will be handled through PUT requests. Deletion is also supported.

**Account:**

Account register

POST api/Account/Register

* Email
* Password
* ConfirmPassword

Account set password

POST api/Account/SetPassword

* NewPassword
* ConfirmPassword

**KcafeMeal:**

Get all KCafe Meals

GET api/KCafeMeal

* No necessary body parameters

Get a KCafe meals by ID

GET api/KCafeMeal/{id}

* MealID

Create a KCafe meal

POST api/KCafeMeal

* Name
* Price
* (string) Description
* (string) Ingredients

Edit a KCafe meal

PUT api/KCafeMeal

* MealID
* Name
* Price
* (string) Description
* I(string) ngredients

Delete a KCafe meal by ID

DELETE api/KCafeMeal/{id}

* id

**KGrocerItem:**

Get all KGrocer items

GET api/KGrocerItem

* No necessary body parameters

Get a KGrocer items by ID

GET api/KGrocerItem/{id}

* ItemID

Create a KGrocer item

POST api/KGrocerItem

* Name
* Price
* (string) Description
* (string) Category
* DaysToExpire

Edit a KGrocer item

PUT api/KGrocerItem

* ItemID
* Name
* Price
* (string) Description
* (string) Category
* DaysToExpire

Delete a KGrocer item by ID

DELETE api/KGrocerItem/{id}

* ItemID

**OrderMeal:**

Get all OrderMeals

GET api/OrderMeal

* No necessary body parameters

Get an Order OrderMeal by ID

GET api/Order OrderMeal /{id}

* OrderID

Create an OrderMeal

POST api/OrderMeal

* MealID
* Quantity

Edit an OrderMeal

PUT api/OrderMeal

* OrderID
* MealID
* Quantity

Delete an order by ID

DELETE api/OrderMeal/{id}

* OrderID

**OrderItem:**

Get all order items

GET api/OrderItem

* No necessary body parameters

Get an order item by ID

GET api/OrderItem/{id}

* OrderID

Create an order item

POST api/OrderItem

* ItemID
* Quantity

Edit an order item

PUT api/OrderItem

* OrderID
* ItemID
* Quantity

Delete an order item by ID

DELETE api/OrderItem/{id}

* OrderID