

Assessment Details

.NET API Coding Assessment

The goal of this test is to assess skills for development of Restful API's in ASP.NET MVC project and assess your understanding of business requirements, database design and stored procedure.

Requirements

- Develop an **Order Management System** with Rest API's where API clients will be able to add orders, edit orders, delete orders and view orders.
- An order contains buyer information, order status, shipping address. An order can have many order items. Each order item has a quantity and is associated with a product. A product has properties name, weight, height, image, Stock Keeping Unit (SKU), barcode and available quantity.
- Develop Web API's where API clients can add orders update orders, delete orders, view orders.
- Order can have statuses: Placed, Approved, Cancelled, In Delivery, Completed
- There are two user roles: Administrator and Buyer. Buyer should be able to see only their orders, Administrator should be able to see all orders in the system
- Once order has been placed, buyer should receive an email.

- Technical Requirements:

- Use ASP.NET MVC Web API
- Use ASP.NET MVC 4.5
- Use SQL Server Express Edition
- Use Microsoft Visual Studio Community Edition
- Use Postman for testing the developed API's
- Use RestFul Standard
- Use Stored procedure where necessary to perform joins
- Use Nuget package manager for dependencies
- There is no need of any user interface. Everything need to be tested from a Rest Client like Postman

- Submission Details:

- Once all your code is done, please push your code to a github public repository.
- Please share the github repository to us so that we can review the code. A document explaining your approach would be good. The document can also explain if something is not done as per requirement and what are the problems which you faced.

ASSESSMENT DOCUMENTATION

The Solution of the assessment is divided into 2 projects in order to maintain REST standards

1. **OMSDataAccessLayer** – This is a class library that contains the database entities. Here I have Used **Entity Framework 5.0** to make connection with the Database (OrderManagementSystem).
2. **OrderManagementSystemAPI** – This is the Main REST API project.

SOFTWARES USED

1. SQL SERVER 2017 DEVELOPER EDITION
2. SQL SERVER MANAGEMENT STUDIO 2017
3. MICROSOFT VISUAL STUDIO 2017 PROFESSIONAL EDITION
4. POSTMAN (for testing the API)

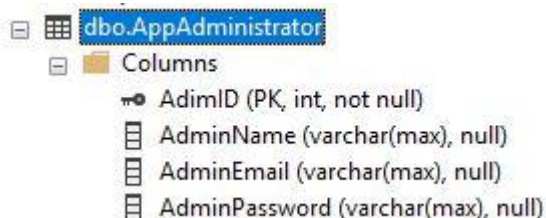
DATABASE STRUCTURE

DATABASE NAME – **OrderManagementSystem**

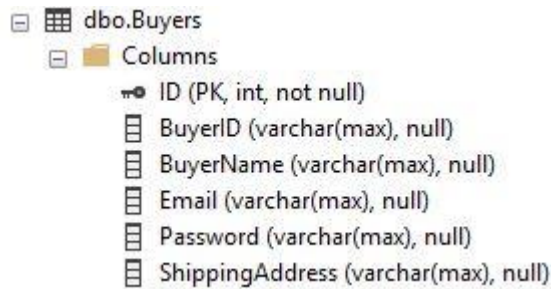
The above mentioned database contains 5 tables and 2 stored procedures.

Here is the list of the tables with columns in image:

1. **dbo.AppAdministrator** (this table contains details of the Administrator which are manually added to the database using sql server management studio)



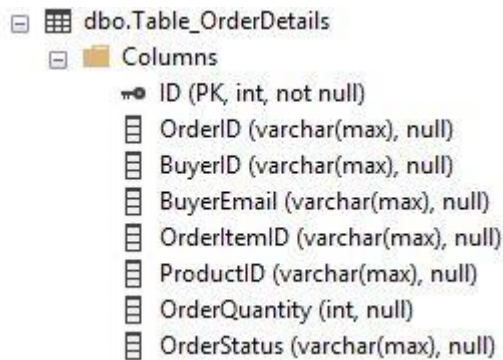
2. **dbo.Buyers** (this table contains client/buyer details)



The screenshot shows the 'Columns' folder expanded for the 'dbo.Buyers' table. The columns listed are:

Column Name	Data Type	Nullability
ID	int	not null
BuyerID	varchar(max)	null
BuyerName	varchar(max)	null
Email	varchar(max)	null
Password	varchar(max)	null
ShippingAddress	varchar(max)	null

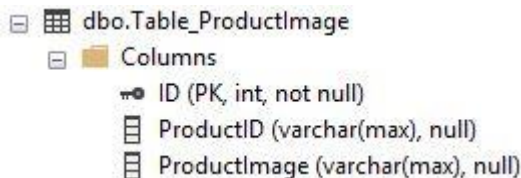
3. **dbo.Table OrderDetails** (this table contains details of the orders placed by the buyer. The BuyerID and BuyerEmail columns contains the buyer's ID and Email who has placed the order)



The screenshot shows the 'Columns' folder expanded for the 'dbo.Table_OrderDetails' table. The columns listed are:

Column Name	Data Type	Nullability
ID	int	not null
OrderID	varchar(max)	null
BuyerID	varchar(max)	null
BuyerEmail	varchar(max)	null
OrderItemID	varchar(max)	null
ProductID	varchar(max)	null
OrderQuantity	int	null
OrderStatus	varchar(max)	null

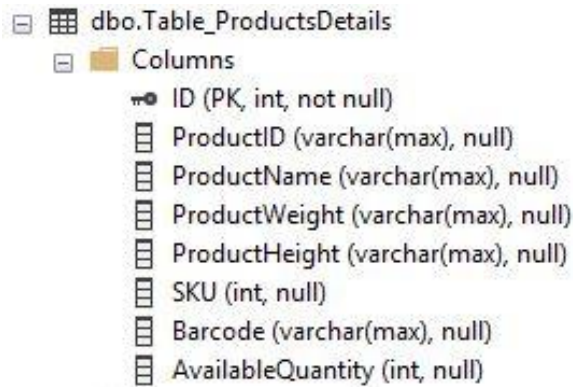
4. **dbo.Table ProductImage** (In this table the ProductID column represents the ID of product which has to be present in the table dbo. Table_ProductsDetails and ProductImage column contains the path of the corresponding uploaded image.)



The screenshot shows the 'Columns' folder expanded for the 'dbo.Table_ProductImage' table. The columns listed are:

Column Name	Data Type	Nullability
ID	int	not null
ProductID	varchar(max)	null
ProductImage	varchar(max)	null

5. **dbo.Table ProductsDetails** (This table Contains Details of the Product which can only be added by the administrator)



CODE FUNCTIONALITY

As written above the solution is divided into 2 parts and the work of **OMSDataAccessLayer** (which is a Class Library Project Type) Is to store the **DATABASE ENTITIES** and create a connection between the Database and the **OrderManagementSystemAPI** (The Main WEB API MVC Project).

In order to connect the 2 projects I have added a reference of the Class Library project (**OMSDataAccessLayer**) to the main WEB API MVC project (**OrderManagementSystemAPI**) and added the Connection string to both the projects in The App.config and Web.config files of the Class Library and WEB API projects respectively.

- **THE PROJECT CONTAINS NO USER INTERFRACE ALL THE TESTING IS DONE THROUGH THE REST CLIENT POSTMAN**

TESTING THE WEB API PROJECT

(OrderManagementSystemAPI)

The API project Contains 2 API-Controllers and 4 class files one of the class file is present in the **Models** directory of the project and the other three are in root directory of the project itself. In the next section I will be explaining functionality of the API-Controllers and class file step by step.

CLASSES IN THE ROOT DIRECTORY OF THE PROJECT

AdministratorSecurity.cs – is a class file is present in the root directory of the project which has **Login** function which helps to check whether the credentials provided for Administrator login (comes from the database table **dbo.AppAdministrator**) are correct.

BuyerSecurity.cs – is a class file is present in the root directory of the project which has **Login** function which helps to check whether the credentials provided for Administrator login (comes from the database table **dbo. Buyers**) are correct.



The LOGIN credentials are provided in base64 encoded format in the HEADER

BasicAuthenticationAttribute.cs – This is a class present in the root directory of the API-Project which decodes the **base64** credentials and helps in BASIC AUTHENTICATION.

BASE64 ENCODED EMAIL AND PASSWORD

ADMIN:

pstark@gmail.com:pstark30071993 –
Chn0YXJrQGdtYWlsLmNvbTpwc3RhcmszMdA3MTk5Mw==

USERS:

speedprince77967@gmail.com:3071993 -
c3BlZWRwcmluY2U3Nzk2N0BnbWFpbC5jb206MzA3MTk5Mw==

aoryan77@gmail.com:3071993 -
YW9yeWFuNzdAZ21haWwuY29tOjMwNzE5OTM=

API-CONTROLLERS

AdministratorActionsController.cs

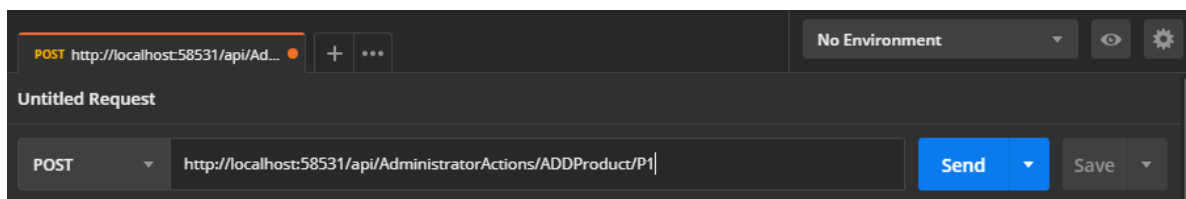
Methods : All the below mentioned methods are designed such that only the ADMINISTRATOR can call them

ADDProduct()

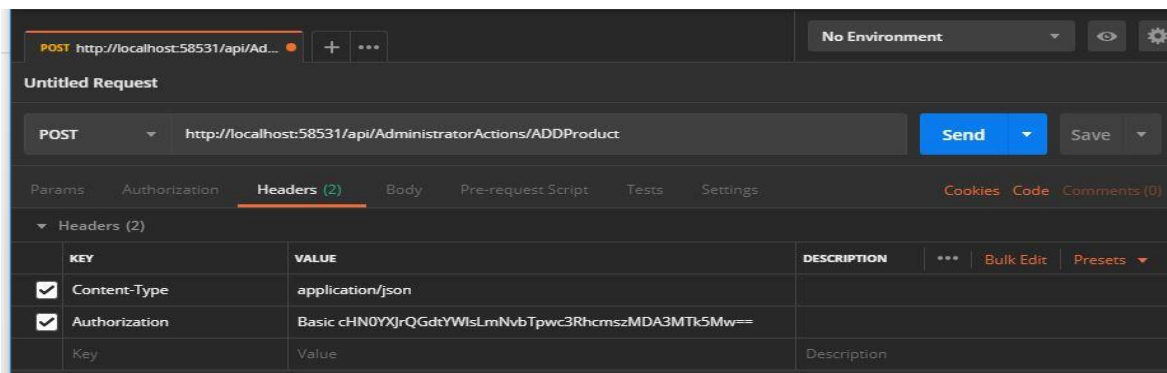
This is **POST Method** which helps the Admin add new products to the database

For example:

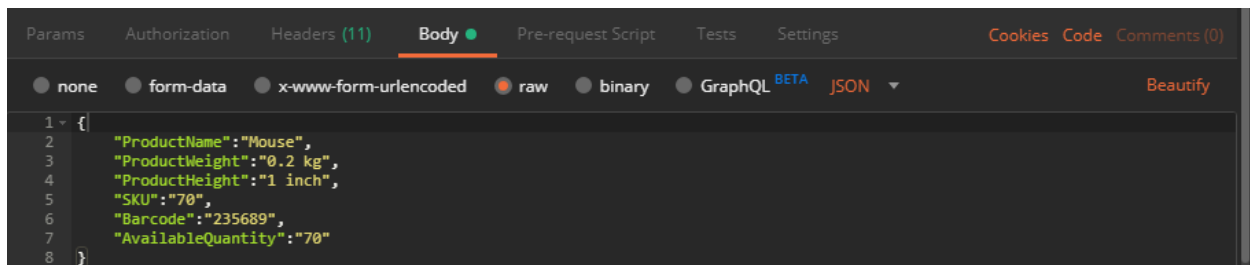
STEP 1: Open Postman Add the URI(Pass ProductID in URI) and set the Request Type to Post



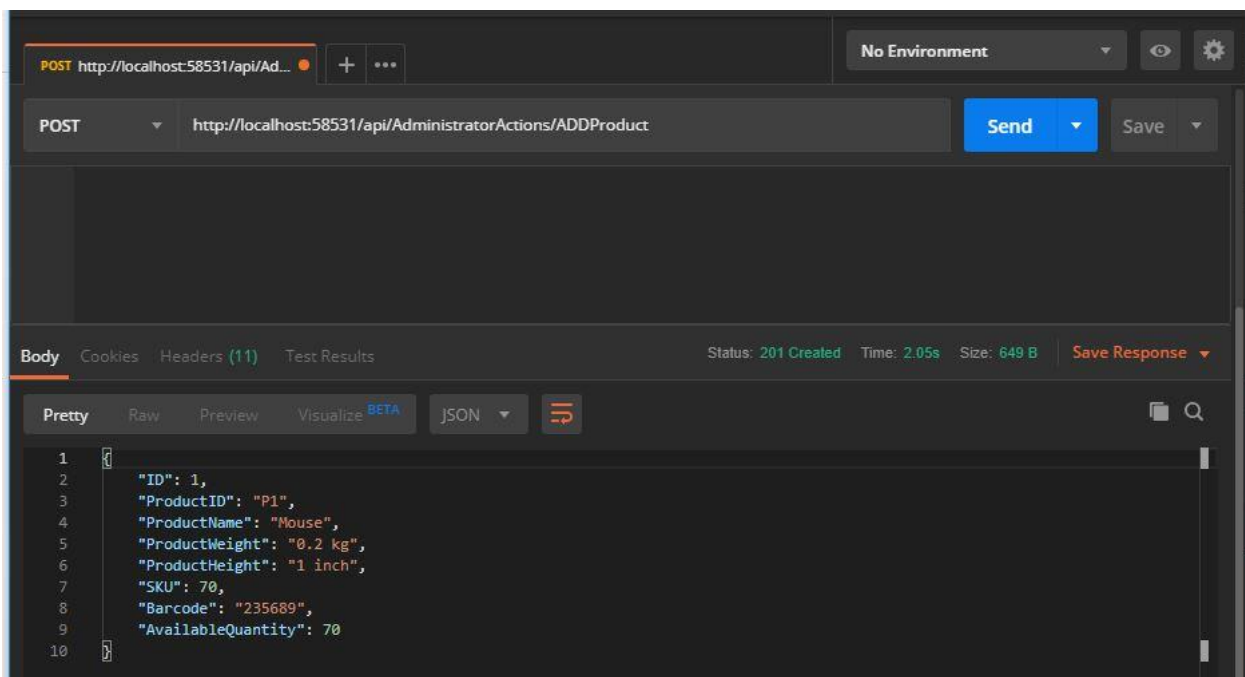
STEP 2: add **Content-Type** and **Administrator Authorization(BASE64 ENCODED)** to HEADER



STEP 3: Add **JSON** formatted data to Body



STEP 4: Click on **Send**, then you get the following result

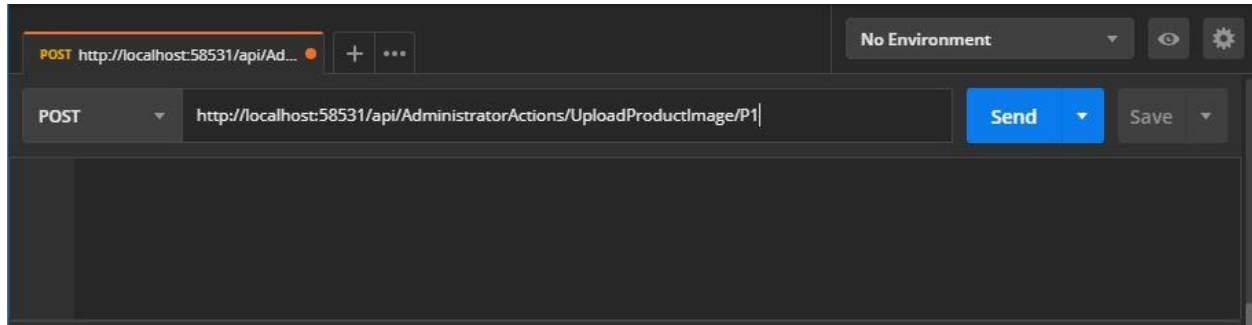


UploadProductImage()

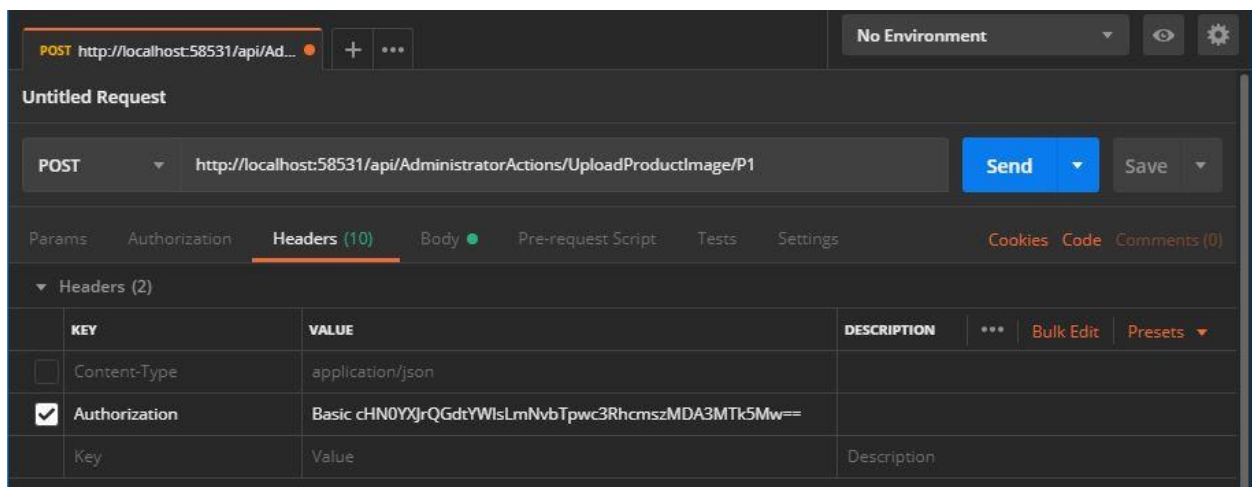
This is **POST Method** which helps the Admin upload image of currently present products to the sever and upload the file path to the database

For example:

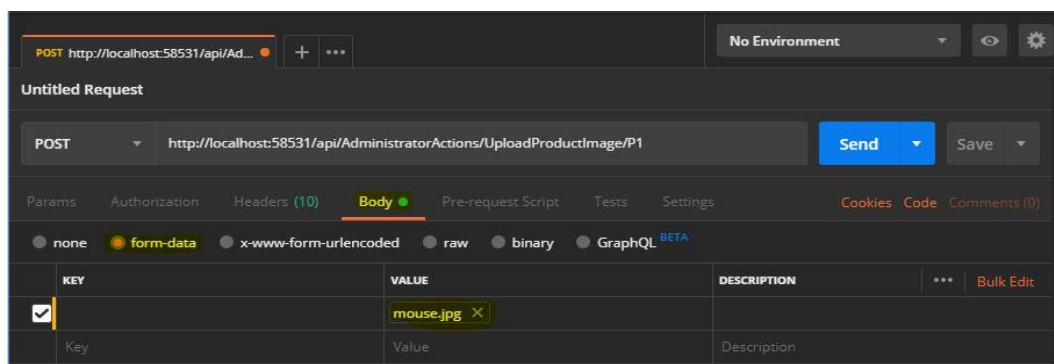
STEP 1: Open Postman Add the Uri and set the Request Type to Post and Pass the ProductId to the Uri.



STEP 2: add Administrator **Authorization(BASE64 ENCODED)** to HEADER



STEP 3: Click on **Body** and Then Click on **form-data** , then change Key from text to file using the dropdown and add brow the image to be uploaded



STEP 4: Click **SEND** Image will be uploaded to ProductImages folder present in the root directory of the project and corresponding data will be saved to database

DESKTOP-Q4RM28V...ble_ProductImage			
	ID	ProductID	ProductImage
	1	P1	~/ProductImages/mouseP1.jpg

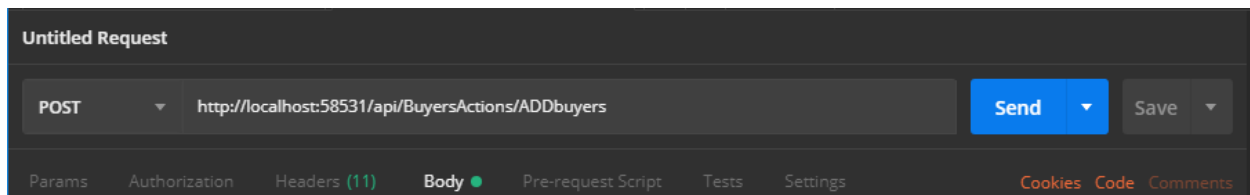
BEFORE WE CAN PROCCED WITH THE OTHER METHODS IN THE CONTROLLER WE NEED TO ADD A BUYER AND PLACE A ORDER

Let's FLIP TO -> **BuyersActionsController.cs**

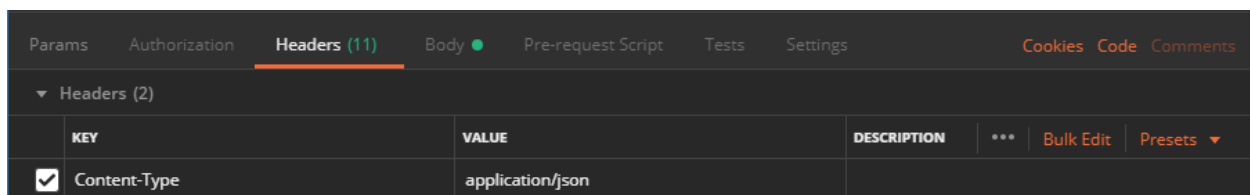
Methods: ALL Methods except Addbuyer() uses Basic Authentication of Buyer email and password in order to run.

Addbuyer()

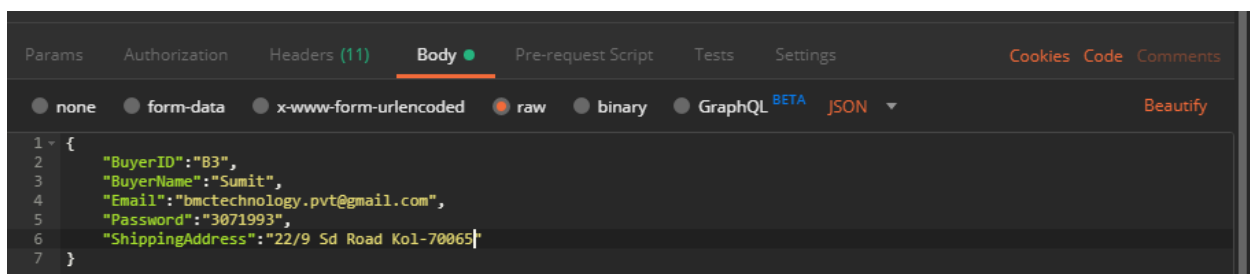
STEP 1: Open Postman Add the Uri and set the Request Type to Post



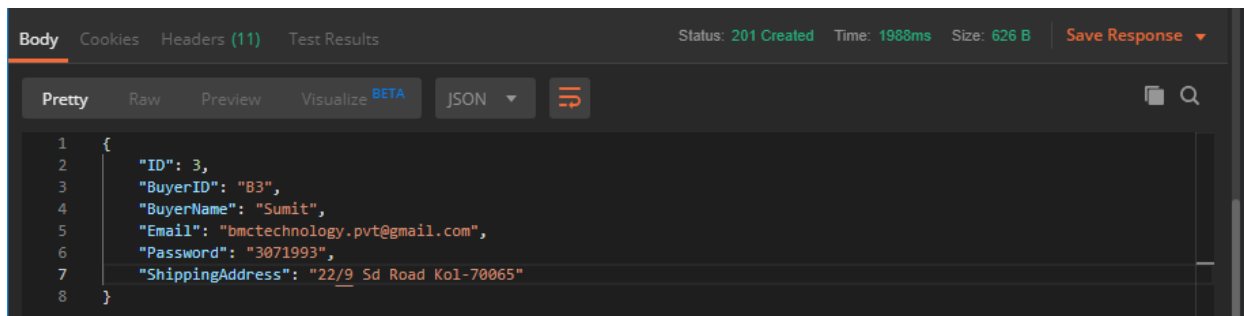
STEP 2: add **Content-Type** to HEADER



STEP 3: Add **JSON** formatted data to Body



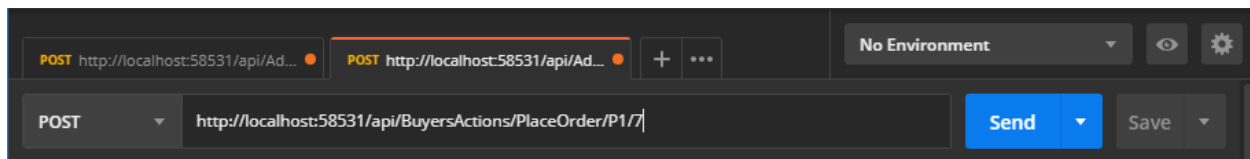
STEP 4: Click on **Send**, then you get the following result



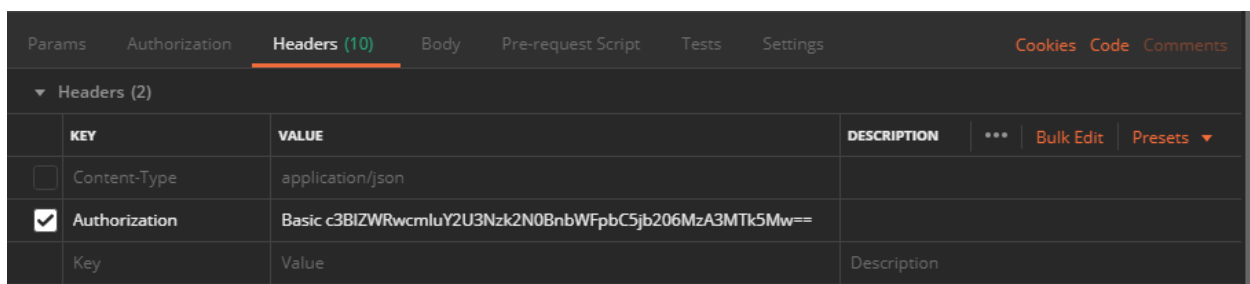
PlaceOrder()

This is a POST Method used by buyers to place new orders

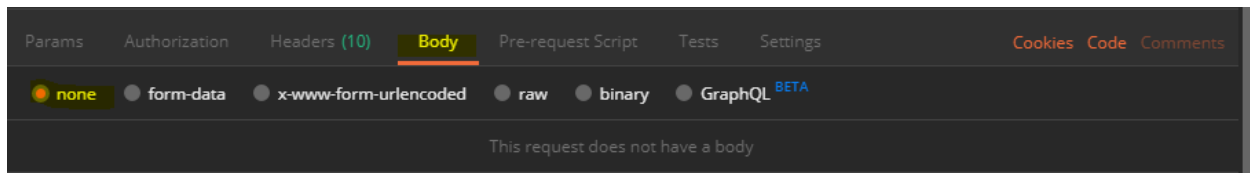
STEP 1: Open Postman Add the URI (Pass ProductID and Order Quantity to the URI respectively) and set the Request Type to Post



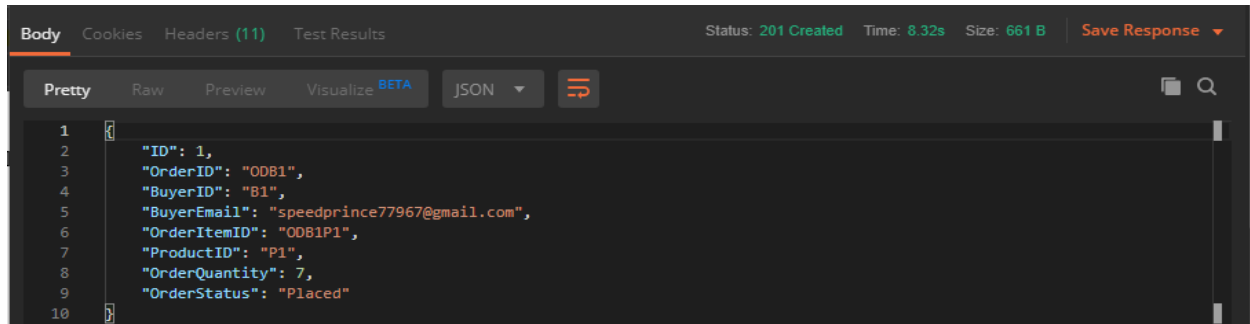
STEP 2: add **Buyer Authorization(BASE64 ENCODED)** to HEADER



STEP 3: Click on **Body** and Then Click on **none** ,



STEP 4: Click on **Send**, then you get the following result



Postman interface showing the response body for a successful order placement. The status is 201 Created, time is 8.32s, and size is 661 B. The response is in JSON format and contains the following data:

```
{
  "ID": 1,
  "OrderID": "00B1",
  "BuyerID": "B1",
  "BuyerEmail": "speedprince77967@gmail.com",
  "OrderItemID": "00B1P1",
  "ProductID": "P1",
  "OrderQuantity": 7,
  "OrderStatus": "Placed"
}
```

NOTE – When a Order is placed a auto generated email is send to the Buyers email address(only gmail accounts) and the Available quantity of the Product in the Table_ProductDetails table in the database is reduced according to order quantity . A buyer cannot place an order of the same product again until the first orderStatus is Completed instead the buyer will be asked to update the previous order

DESKTOP-Q4RM28V...e_ProductsDetails								
	ID	ProductID	ProductName	ProductWeight	ProductHeight	SKU	Barcode	AvailableQuantity
▶	1	P1	Mouse	0.2 kg	1 inch	70	235689	63
	2	P2	Joystick	0.35 kg	2.5 inch	77	235689	77



sam077royal@gmail.com

to me ▾



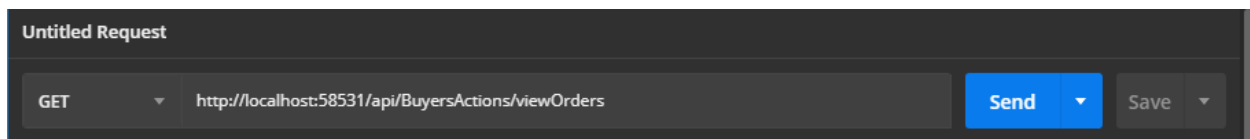
05:15 (15 minutes ago)



Your Order has been successfully placed

viewOrders()

STEP 1: Open Postman Add the Uri and set the Request Type to GET



STEP 2: add **Buyer Authorization(BASE64 ENCODED)** to HEADER

Params

Authorization

Headers (10)

Body

Pre-request Script

Tests

Settings

Cookies

Code

Comments

▼ Headers (2)

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets ▼
<input type="checkbox"/>	Content-Type	application/json				
<input checked="" type="checkbox"/>	Authorization	Basic c3BIZWRwcmVudWU3Nzk2N0BnbWFpbC5jb206MzA3MTk5Mw==				
	Key	Value	Description			

STEP 4: Click on **Send**, then you get the following result(A buyer can only view order details of orders ordered by them)

```
Body Cookies Headers (10) Test Results Status: 200 OK Time: 10.27s Size: 718 B Save Response ▼
Pretty Raw Preview Visualize BETA JSON ▼
1 {
2   {
3     "BuyerName": "Speed",
4     "ShippingAddress": "Z/3 299 RSR Kol-70055",
5     "OrderID": "ODB1",
6     "BuyerEmail": "speedprince77967@gmail.com",
7     "OrderItemID": "ODB1P1",
8     "OrderQuantity": 7,
9     "OrderStatus": "Placed",
10    "ProductImage": "~/ProductImages/mouseP1.jpg",
11    "ProductName": "Mouse",
12    "ProductWeight": "0.2 kg",
13    "ProductHeight": "1 inch"
14  }
15 }
```

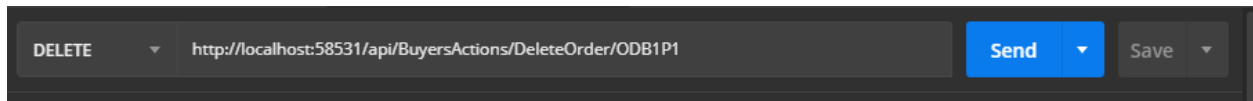
NOTE : The class `JoinOrderDetails.cs`(This class is present inside `Models` Directory of the `API-Project`) is used as a model class and stored procedure `JoinOrderDetails(BuyerID)` is called in the above method.

DeleteOrder()

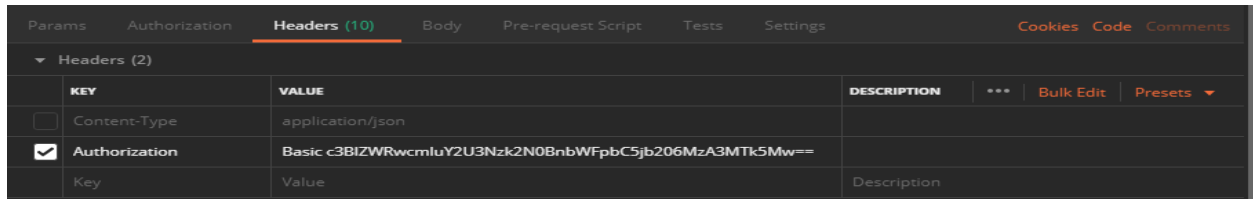
Orders in Database for Buyer-Speed, BuyerID-B1 Before Delete

```
Pretty Raw Preview Visualize BETA JSON ▼
1 {
2   {
3     "BuyerName": "Speed",
4     "ShippingAddress": "Z/3 299 RSR Kol-70055",
5     "OrderID": "ODB1",
6     "BuyerEmail": "speedprince77967@gmail.com",
7     "OrderItemID": "ODB1P1",
8     "OrderQuantity": 7,
9     "OrderStatus": "Placed",
10    "ProductImage": "~/ProductImages/mouseP1.jpg",
11    "ProductName": "Mouse",
12    "ProductWeight": "0.2 kg",
13    "ProductHeight": "1 inch"
14  },
15  {
16    "BuyerName": "Speed",
17    "ShippingAddress": "Z/3 299 RSR Kol-70055",
18    "OrderID": "ODB1",
19    "BuyerEmail": "speedprince77967@gmail.com",
20    "OrderItemID": "ODB1P2",
21    "OrderQuantity": 8,
22    "OrderStatus": "Placed",
23    "ProductImage": "~/ProductImages/joystickP2.jpg",
24    "ProductName": "Joystick",
25    "ProductWeight": "0.35 kg",
26    "ProductHeight": "2.5 inch"
27  }
28 }
```

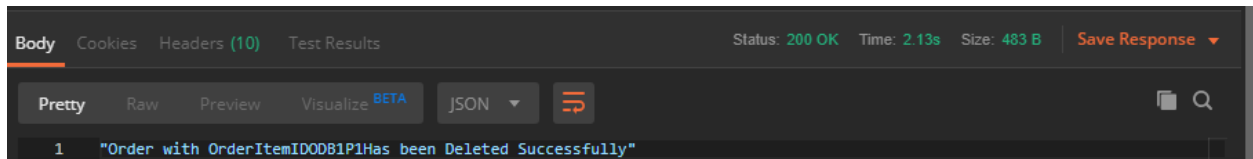
STEP 1: Open Postman Add the URI (Pass OrderItemID to the URI) and set the Request Type to DELETE



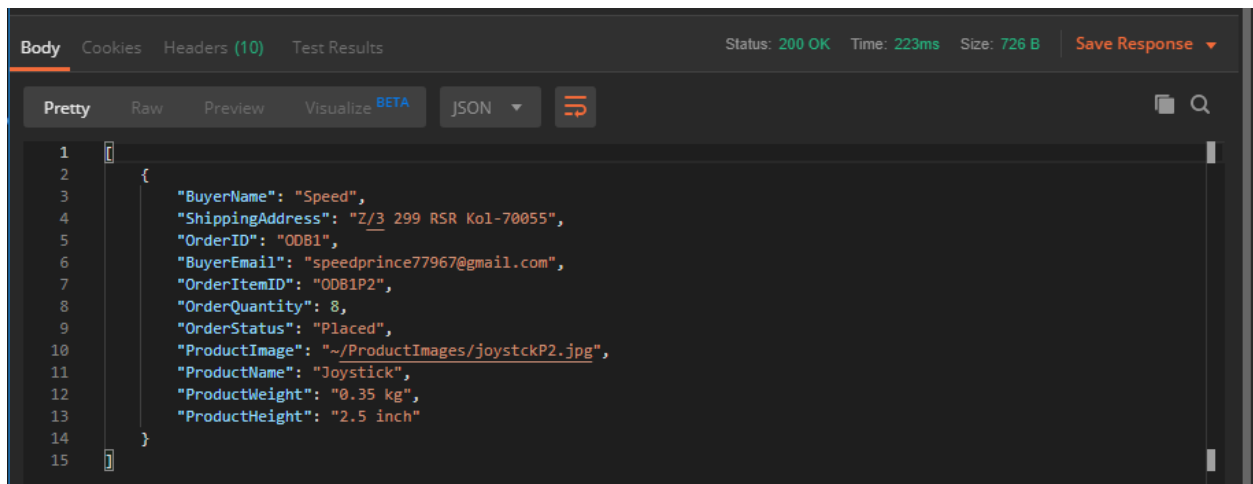
STEP 2: add **Buyer Authorization(BASE64 ENCODED)** to HEADER



STEP 4: Click on **Send**, then you get the following result(A buyer can only delete a order ordered by them only)



Orders in Database for Buyer-Speed, BuyerID-B1 After Delete



UpdateOrderQuantity()

This is a PUT method which used by buyers to update order quantity

STEP 1: Open Postman Add the URI (Pass OrderItemID and Order Quantity (if buyer wants to increase quantity buyer should pass the number of items he wants to increase the product- Quantity by[for example /3] , if buyer wants to reduce then pass the negative value of number of items he wants to decrease the product-Quantity by[for example /-3]) to the URI respectively) and set the Request Type to **PUT**

Increase example:

http://localhost:58531/api/BuyersActions/UpdateOrderQuantity/ODB1P2/3

Decrease example:

http://localhost:58531/api/BuyersActions/UpdateOrderQuantity/ODB1P2/-3

STEP 2: add **Buyer Authorization(BASE64 ENCODED)** to HEADER

Params	Authorization	Headers (10)	Body	Pre-request Script	Tests	Settings	Cookies	Code	Comments
▼ Headers (2)									
	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets	▼		
<input type="checkbox"/>	Content-Type	application/json							
<input checked="" type="checkbox"/>	Authorization	Basic c3BiZWRCwcmIuY2U3NzY2N0BnbWFpbC5jb206MzA3MTk5Mw==							
	Key	Value	Description						

STEP 4: Click on **Send**, then you get the following result(A buyer can only Update a order ordered by them only)

A screenshot of the Chrome DevTools Network tab. The top bar shows 'Body' selected, with 'Cookies', 'Headers (10)', and 'Test Results' as options. To the right, it displays 'Status: 202 Accepted', 'Time: 278ms', 'Size: 477 B', and a 'Save Response' button. Below this, there are tabs for 'Pretty', 'Raw', 'Preview', and 'Visualize BETA', with 'JSON' selected. A 'Copy' icon and a search icon are also present. The main content area shows a single JSON object: { "quantity Updated Successfully": true }. The first key-value pair is highlighted.

NOTE : Available quantity of the Product in the Table_ProductDetails table in the database is reduced/increased according to new order quantity

P.T.O

NOW WE CAN GET BACK

TO

AdministratorActionsController.cs

METHOD: All the below mentioned methods are designed such that only the ADMINISTRATOR can call them

UpdateOrderStatus()

This Is a **PUT** method used by the Administrator to Update order Status

STEP 1: Open Postman Add the URI (Pass OrderItemID to the URI respectively) and set the Request Type to PUT

Untitled Request

PUT http://localhost:58531/api/AdministratorActions/UpdateOrderStatus/ODB1P2

Send Save

STEP 2: add **Content-Type** and Administrator **Authorization(BASE64 ENCODED)** to HEADER

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies Code Comments

▼ Headers (2)

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets ▼
<input checked="" type="checkbox"/>	Content-Type	application/json				
<input checked="" type="checkbox"/>	Authorization	Basic c3BIZWRwcmluY2U3Nzk2N0BnbWFpbC5jb206MzA3MTk5Mw==				
	Key	Value	Description			

STEP 3: Add **JSON** formatted data to Body

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies Code Comments

none form-data x-www-form-urlencoded raw binary GraphQL BETA JSON Beautify

```

1 {
2   "OrderStatus": "Approved"
3 }

```

STEP 4: Click on **Send**, then you get the following result

	ID	OrderID	BuyerID	BuyerEmail	OrderItemID	ProductID	OrderQuantity	OrderStatus
▶	2	ODB1	B1	speedprince779...	ODB1P2	P2	5	Approved

ViewOrders()

This is a **Get** method which is used by the Administrator to view all orders. Here I have used a Stored Procedure named JoinOrderDetailsForAdmin to inner join Tables Table_OrderDetails, Table_ProductsDetails, Table_ProductImage from The database

STEP 1: Open Postman Add the Uri and set the Request Type to GET

GET http://localhost:58531/api/AdministratorActions/ViewOrders

Send Save

STEP 2: add Administrator **Authorization(BASE64 ENCODED)** to HEADER

Params

Authorization

Headers (10)

Body

Pre-request Script

Tests

Settings

Cookies

Code

Comments

▼ Headers (2)

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets ▼
<input type="checkbox"/>	Content-Type	application/json				
<input checked="" type="checkbox"/>	Authorization	Basic cHN0YXJrQGdtYWlsLmNvbTpwczRhcmszMDA3MTk5Mw==				
	Key	Value	Description			

STEP 3: Click on **Send**, then you get the following result

```

1  [
2  {
3    "BuyerName": "Speed",
4    "ShippingAddress": "Z/3 299 RSR Kol-70055",
5    "OrderID": "00B1",
6    "BuyerEmail": "speedprince77967@gmail.com",
7    "OrderItemID": "00B1P2",
8    "OrderQuantity": 5,
9    "OrderStatus": "Approved",
10   "ProductImage": "~/ProductImages/joystickP2.jpg",
11   "ProductName": "Joystick",
12   "ProductWeight": "0.35 kg",
13   "ProductHeight": "2.5 inch"
14 },
15 {
16   "BuyerName": "Aoryan",
17   "ShippingAddress": "Z/3 299 RSR Kol-70055",
18   "OrderID": "00B2",
19   "BuyerEmail": "aoryan77@gmail.com",
20   "OrderItemID": "00B2P2",
21   "OrderQuantity": 5,
22   "OrderStatus": "Placed",
23   "ProductImage": "~/ProductImages/joystickP2.jpg",
24   "ProductName": "Joystick",
25   "ProductWeight": "0.35 kg",
26   "ProductHeight": "2.5 inch"
27 }

```

Base64 encoding done on

<https://www.base64decode.org>

I could have implemented much more functionality into the project like generating ID (OrderID, ProductID, OrderItemID, BuyerID) using SCALAR VALUE FUNCTIONS but I didn't do it as it was not mentioned in the ASSESSMENT.

CONTACT DETAILS

PROJECT BY – SUMIT PAL

EMAIL – aoryan77@gmail.com

CONTACT NUMBER - +919748592062