**SUPERMARKET BILLING API**

* This API has been developed using **Django 3.2** on backend. The use of **Django-rest-framework** and **CORS** has allows the API to be accessed by any website or application.
* The API has been tested using **Postman** for its various goals.
* The core API offered by rest-framework is not used, instead custom API is made which enables more control and customization.
* Single URL API has been made for multiple purposes.
* API Link :- **http://127.0.0.1:8000/**

**Functions**

1. **Get all items**

* API Link:- [**http://127.0.0.1:8000/items/**](http://127.0.0.1:8000/items/)
* Request type:- GET
* This API call will return all the items currently present in Database.
* The data will be in the form of JSON contained inside an array.

1. **Get all items of a category with all details**

* API Link:- [**http://127.0.0.1:8000/items/?category=input**](http://127.0.0.1:8000/items/?category=input)
* Request type:- GET
* This API call will return all the items which have “category = input”. Here input will be replaced by your desired category.
* For example :- [**http://127.0.0.1:8000/items/?category=food**](http://127.0.0.1:8000/items/?category=food)
* Here category is set to be food, so all the items having category as “food” will be returned.
* The data will be in the form of JSON contained inside an array.

1. **Get all items of a sub-category with all details**

* API Link:- [**http://127.0.0.1:8000/items/?sub\_category=input**](http://127.0.0.1:8000/items/?sub_category=input)
* Request type:- GET
* This API call will return all the items which have “sub category = input”. Here input will be replaced by your desired sub category.
* For example :- **http://127.0.0.1:8000/items/?sub\_category=vegetable**
* Here sub category is set to be vegetable, so all the items having sub category as “vegetable” will be returned.
* The data will be in the form of JSON contained inside an array.

1. **Put items into the database**

* API Link:- **http://127.0.0.1:8000/items/**
* Request type:- POST
* Body :- { **name**: “Name of item”,

**category**: ”category of item”,

**sub\_category**: “sub category of item”,

**amount**: “amount of item”}

The body of the post request should follow the above format taking into account the keywords and the data types. There should not be any typos which will result in failure of data upload. For example:-

{ **name**: “Pepsi”,

**category**: ”beverage”,

**sub\_category**: “Soft drink”,

**amount**: 55}

**Note**:- name of the item should be unique.

There are only three categories to choose from. Namely food, beverage and household. Filling anything other than these will result in leaving the field empty.

* On successful upload of item in database JSON response stating success will be returned. For example:-

{“**Status**”: “Success”,

“Body”: {**name**: “Pepsi”,

**category**: ”Beverage”,

**sub\_category**: “Soft drink”,

**amount**: 55}}

* If there is some error while uploading the item to database JSON response stating the error will be returned. For example:-

{“**Status**”: “Success”,

“**Body**”: "UNIQUE constraint failed: supermart\_item.name"}

1. **Update item details**

* API Link:- **http://127.0.0.1:8000/items/**
* Request type:- PUT
* Body :- { **name**: “Name of item on which changes are to be made”,

**new\_name**: ”New name of the item”,

**new\_category**: “ new category of item”,

**new\_sub\_category**: “new sub category of item”,

**new\_amount**: “new amount of item”}

The body of the put request should follow the above format taking into account the keywords and the data types. There should not be any typos which will result in failure of data upload.

The name of the item will be used to get the item.

Only those fields are to be mentioned which are to be updated. For example:-

{ **name**: “Pepsi”,

**new\_name**: ”Coca Cola”,

**amount:** 60 **}**

Here we are updating name and amount of Pepsi to Coca Cola and 60 respectively.