Ecommerce Django API's

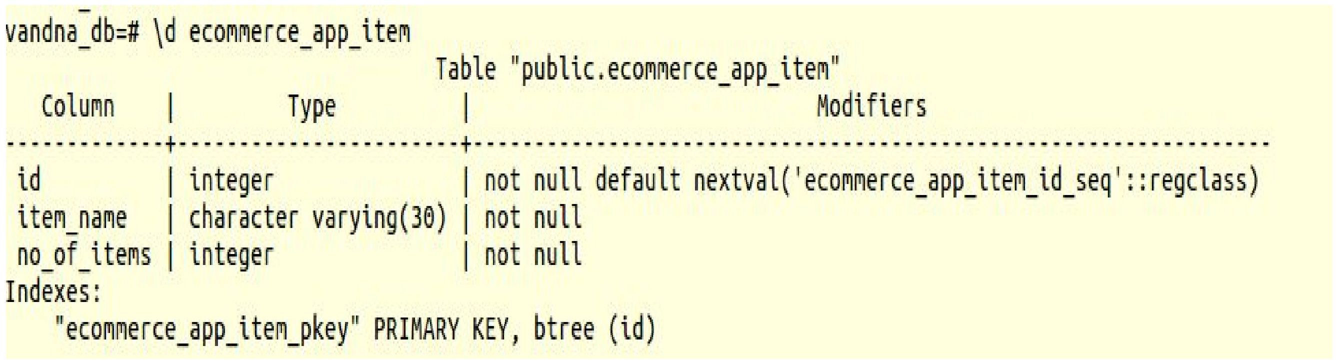
Framework : Django

Database : PostgreSQL

2 tables created in PostgreSQL:

Item

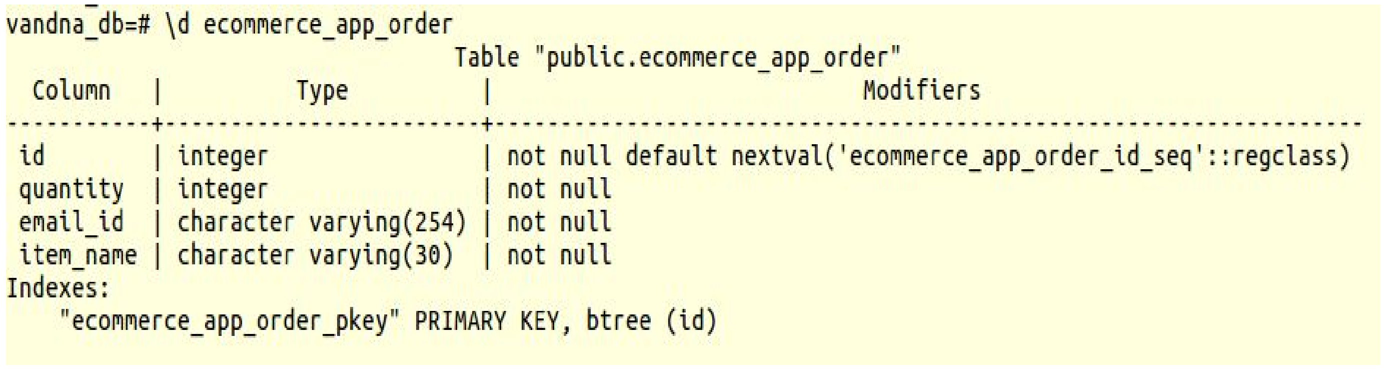
For storing item details.



```
vandna_db=# \d ecommerce_app_item
                              Table "public.ecommerce_app_item"
   Column    |         Type          |                          Modifiers
-------------+-----------------------+-------------------------------------------------------------
 id          | integer               | not null default nextval('ecommerce_app_item_id_seq'::regclass)
 item_name   | character varying(30) | not null
 no_of_items | integer               | not null
Indexes:
    "ecommerce_app_item_pkey" PRIMARY KEY, btree (id)
```

Order

For storing order placed.



```
vandna_db=# \d ecommerce_app_order
                              Table "public.ecommerce_app_order"
  Column   |          Type          |                            Modifiers
-----------+------------------------+----------------------------------------------------------------
 id        | integer                | not null default nextval('ecommerce_app_order_id_seq'::regclass)
 quantity  | integer                | not null
 email_id  | character varying(254) | not null
 item_name | character varying(30)  | not null
Indexes:
    "ecommerce_app_order_pkey" PRIMARY KEY, btree (id)
```

CRUD operations on items

Create/Update Item

API: http://localhost:8000/ecommerce_app/add_item/

Description:

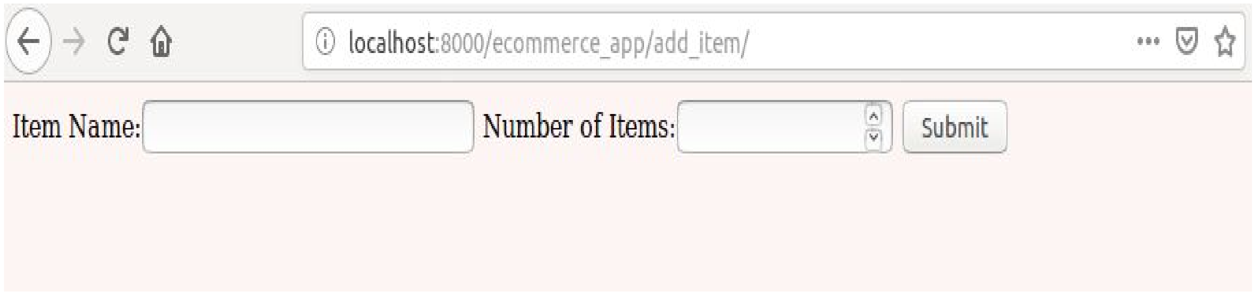
- Add items in item table
- It does the work of upsert (insert + update)
- In case the item is not present in Item table, It is added to Item table.
- In case it's already present, it's attributes are updated in Item table.
- Save item information in Item Table.
- It can be zero just to let the end user know that it is currently out of stock
- instead of unavailability of that item entirely.

Param request: GET/POST

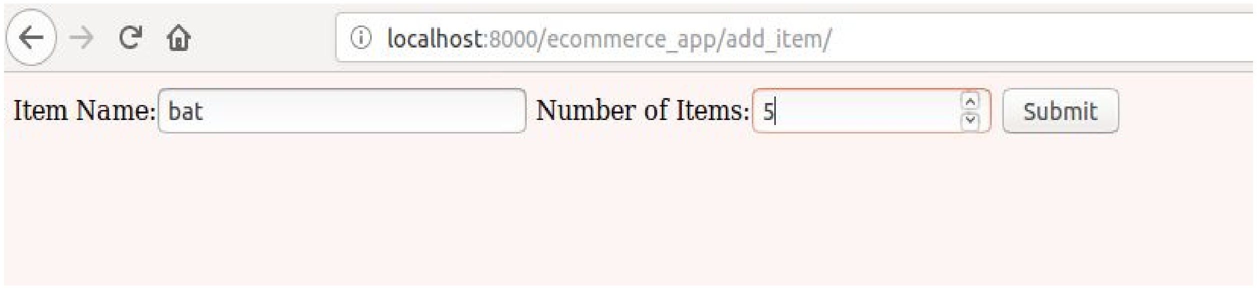
Return: blank form (GET), save form information in db (POST)

Snapshots:

Add/Update Item Form

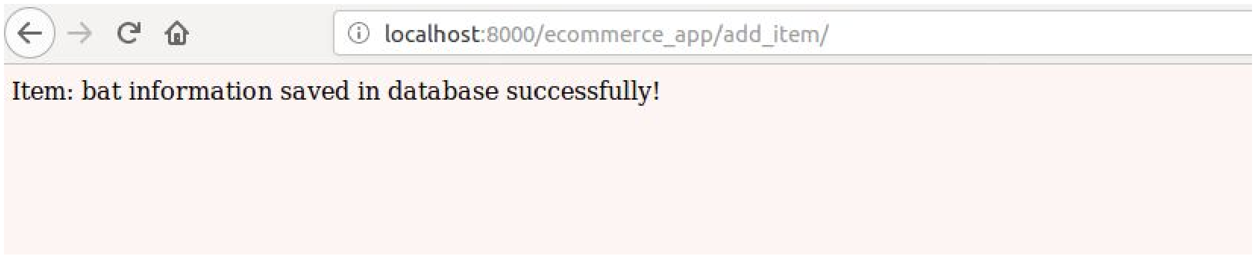


Add Items in form



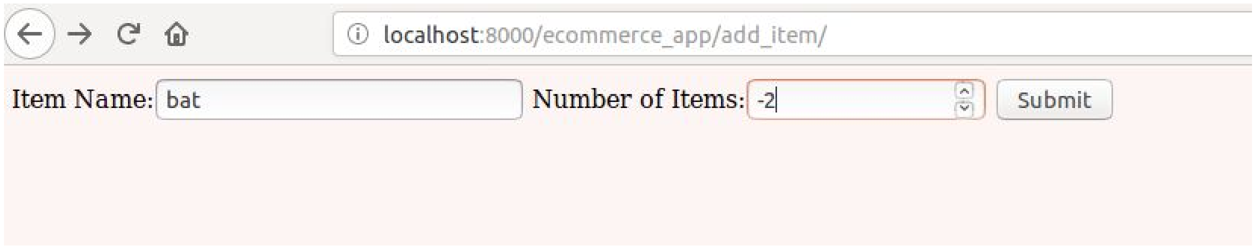
Item Name: bat
Number of Items: 5

Save item information in Item Table.

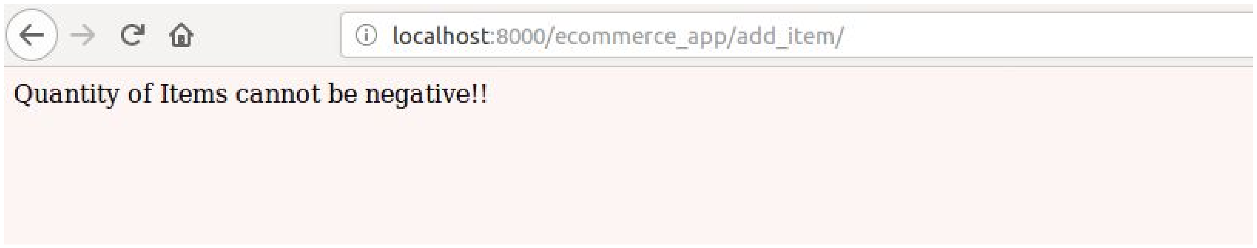

localhost:8000/ecommerce_app/add_item/

Item: bat information saved in database successfully!

Quantity of Items cannot be negative


localhost:8000/ecommerce_app/add_item/

Item Name: bat Number of Items: -2


localhost:8000/ecommerce_app/add_item/

Quantity of Items cannot be negative!!

Delete Item

API: http://localhost:8000/ecommerce_app/delete_item?item_name=ball

Description:

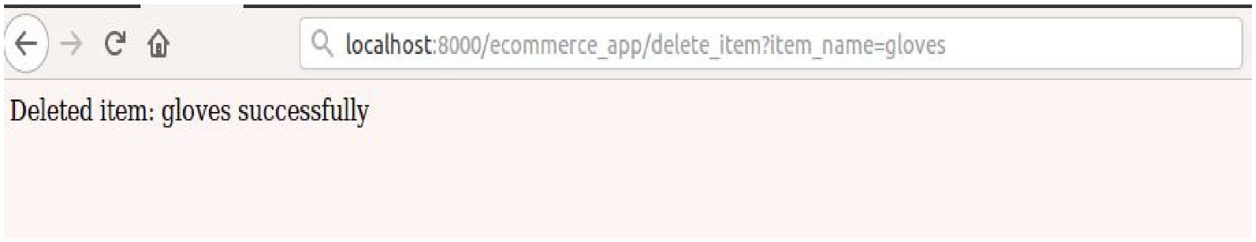
- Delete item from Item table.
- In case item is not present in item table, it gives below message.

Param request: GET

Return: Deletion message

Snapshots

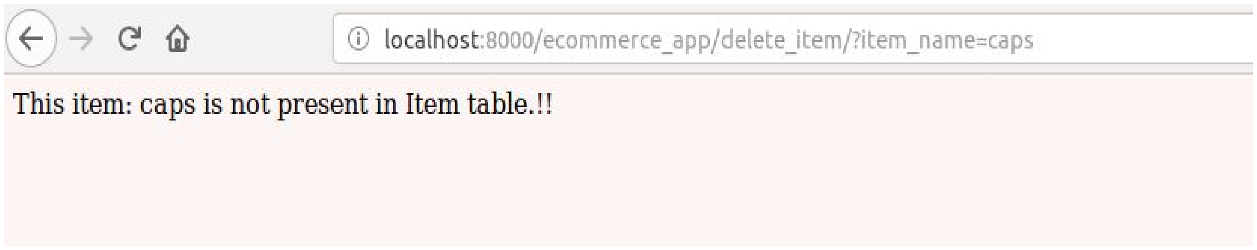
Delete item from item table



localhost:8000/ecommerce_app/delete_item?item_name=gloves

```
Deleted item: gloves successfully
```

In case item is not present in item table, it gives below message.



localhost:8000/ecommerce_app/delete_item/?item_name=caps

```
This item: caps is not present in Item table.!!
```

Get a specific Item

API: http://localhost:8000/ecommerce_app/get_item/wickets/

Description:

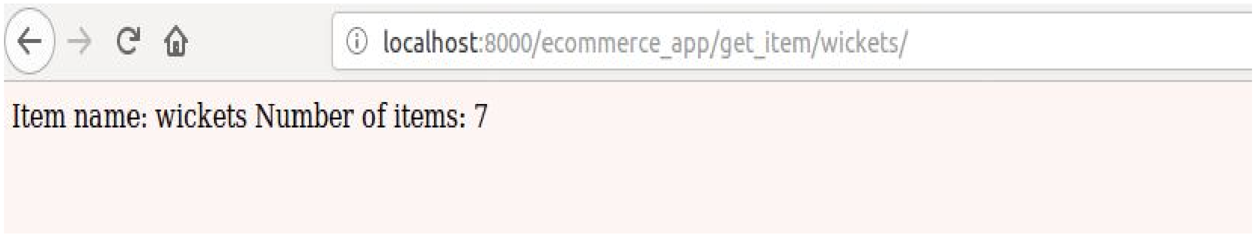
- Get a specific entry from item table.
- In case item is not present in item table, it gives below message.

Param request:GET

Return: Item details

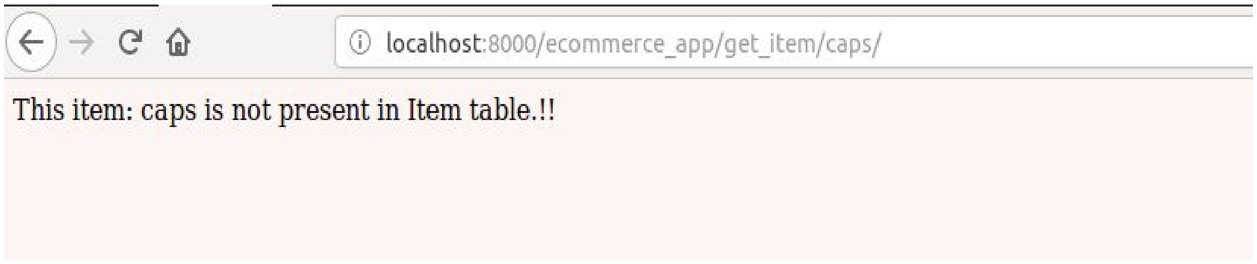
Snapshots

Get a specific entry from item table.


localhost:8000/ecommerce_app/get_item/wickets/

```
Item name: wickets Number of items: 7
```

In case item is not present in item table, it gives below message.


localhost:8000/ecommerce_app/get_item/caps/

```
This item: caps is not present in Item table.!!
```

All items listing

Get all items

API: http://localhost:8000/ecommerce_app/get_all_items/

Description:

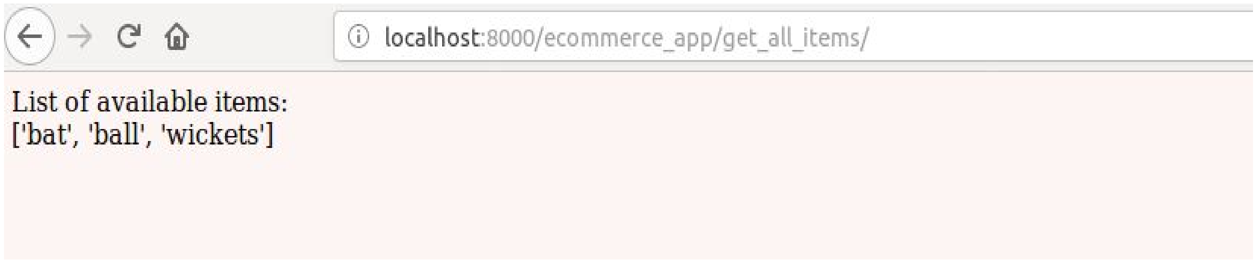
- Get all items from Item table

Param request: GET

Return: List of available items in Item table

Snapshot

Get all items from Item table


localhost:8000/ecommerce_app/get_all_items/

```
List of available items:
['bat', 'ball', 'wickets']
```

Single & bulk ordering (Just consider the item, no. of items & email ids as params for ordering)

Single Order

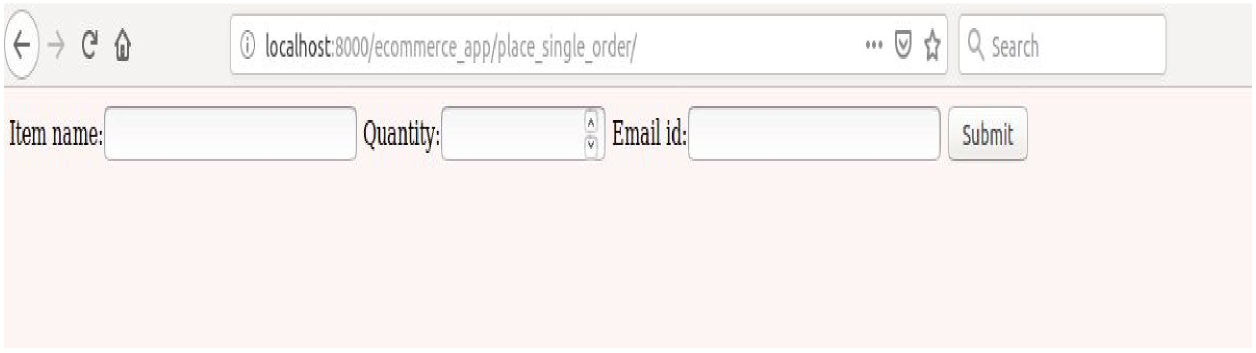
API: http://localhost:8000/ecommerce_app/place_single_order/
Description :

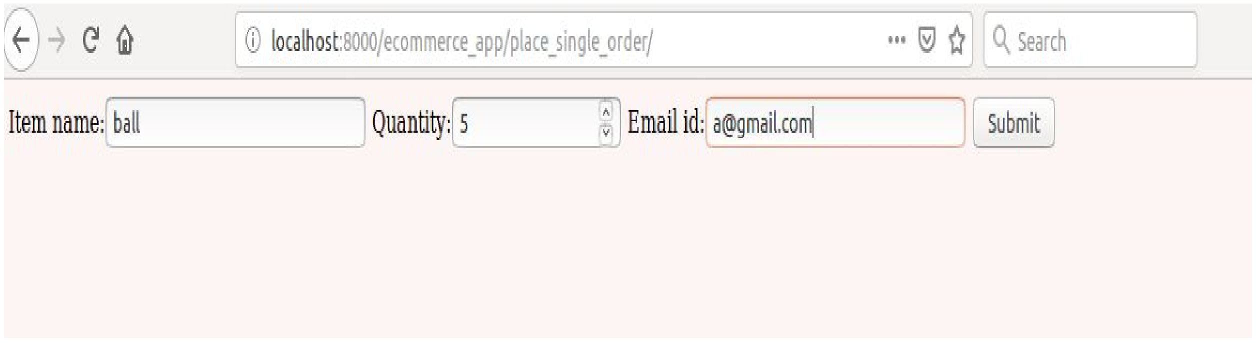
- Place a single order
- Save order information in Order Table
- Get order information from form and save it in Order table.
- Various cases are handled while placing order like:
- If the item ordered doesn't exist in Item table, will give a message.
- The quantity ordered of the item should be atleast one.
- The quantity ordered cannot be more than the number of items present in Items table.
- If the item ordered is out of stock i.e. in Item table.
- If none of the above conditions are true, order is successfully placed and saved in Order table.

Param request: POST/GET
Return: blank form (GET), save form information in db (POST)

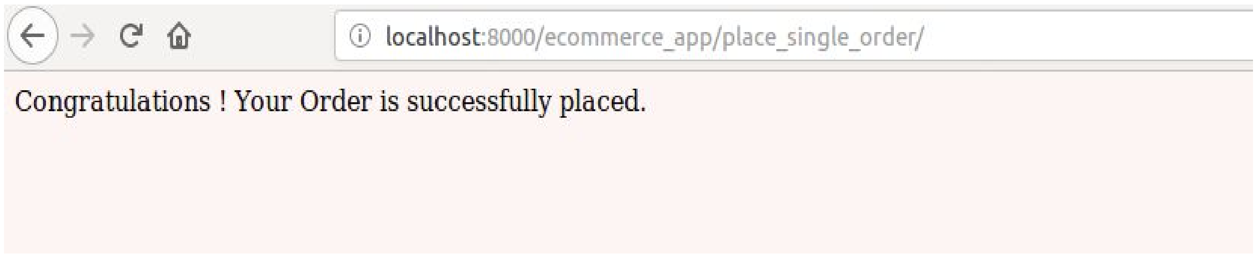
Snapshots

Fill order form (Single order can be placed).Order is successfully placed and saved in Order table.Also, the number of items ordered are decremented from Item table for that item.



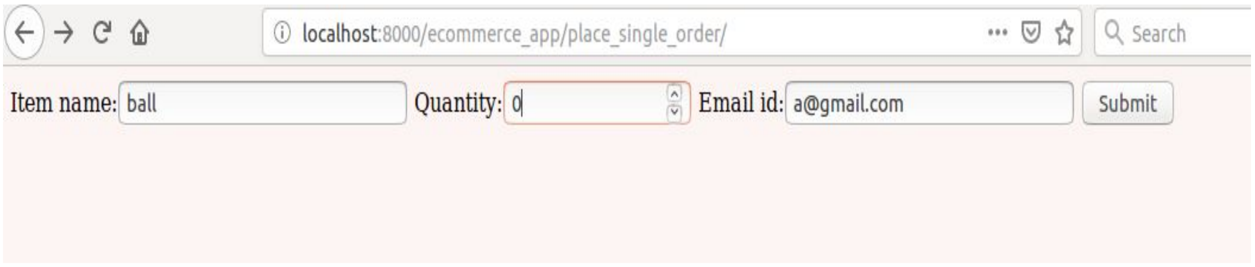


Item name: ball Quantity: 5 Email id: a@gmail.com

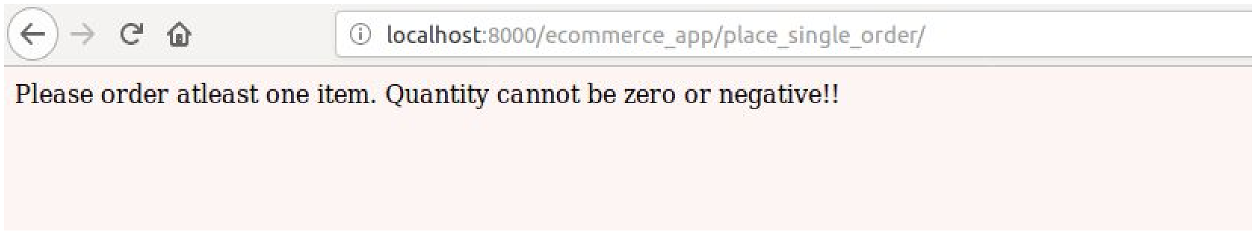

localhost:8000/ecommerce_app/place_single_order/

Congratulations ! Your Order is successfully placed.

The quantity ordered of the item should be atleast one.

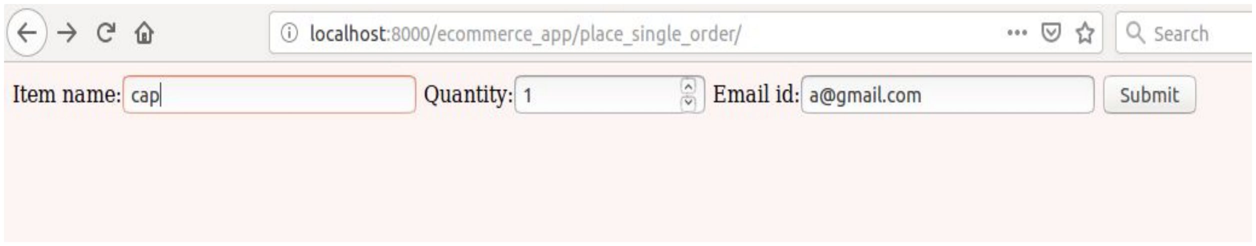

localhost:8000/ecommerce_app/place_single_order/

Item name: ball Quantity: 0 Email id: a@gmail.com

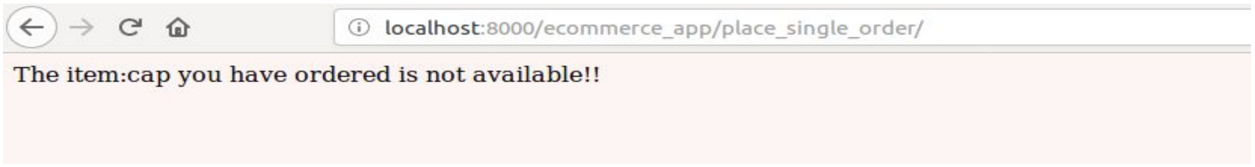

localhost:8000/ecommerce_app/place_single_order/

Please order atleast one item. Quantity cannot be zero or negative!!

If the item ordered doesn't exist in Item table, will give a message.

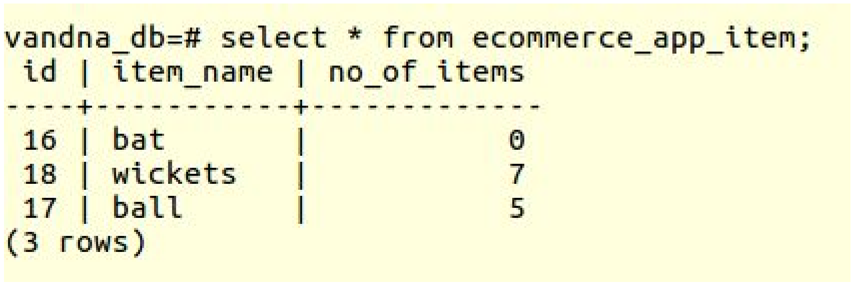

localhost:8000/ecommerce_app/place_single_order/

Item name: cap Quantity: 1 Email id: a@gmail.com

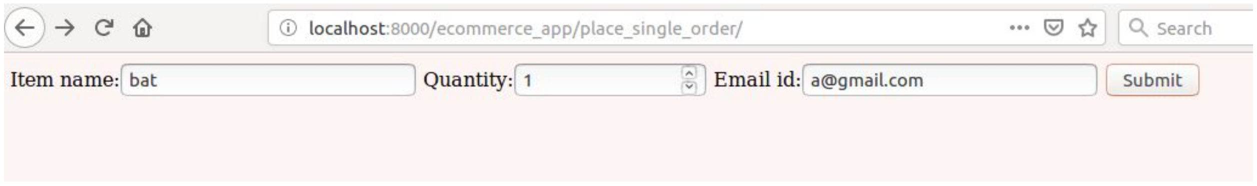

localhost:8000/ecommerce_app/place_single_order/

The item:cap you have ordered is not available!!

If the item ordered is out of stock i.e. in Item table.

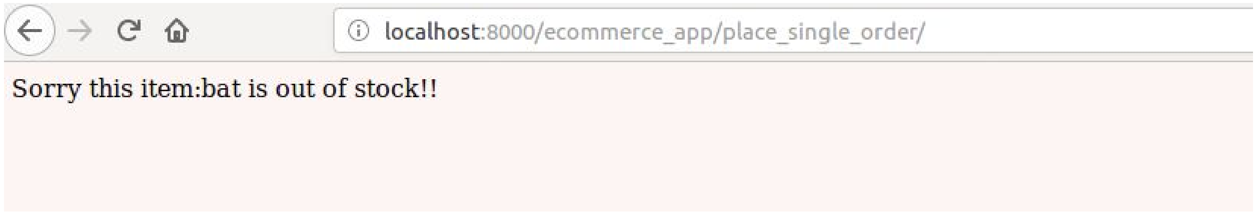


```
vandna_db=# select * from ecommerce_app_item;
 id | item_name | no_of_items
----+-----------+------------
 16 | bat       |           0
 18 | wickets   |           7
 17 | ball      |           5
(3 rows)
```



localhost:8000/ecommerce_app/place_single_order/

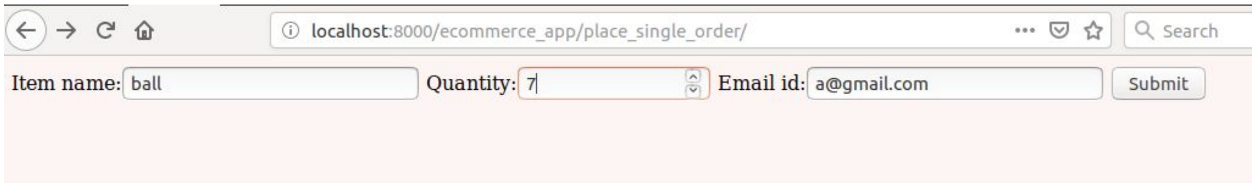
Item name: bat Quantity: 1 Email id: a@gmail.com



localhost:8000/ecommerce_app/place_single_order/

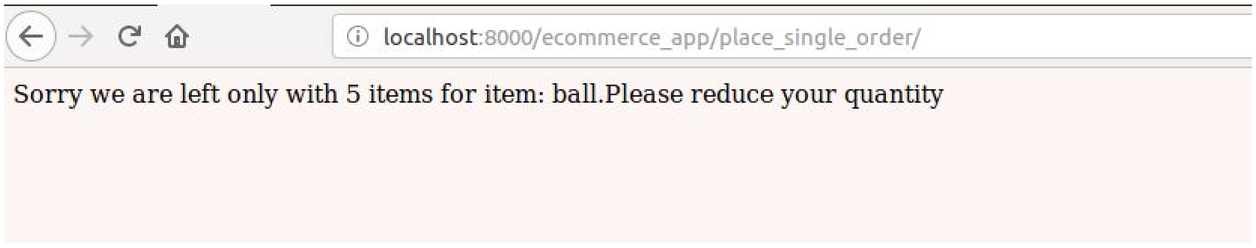
Sorry this item:bat is out of stock!!

The quantity ordered cannot be more than the number of items present in Items table.



localhost:8000/ecommerce_app/place_single_order/

Item name: ball Quantity: 7 Email id: a@gmail.com



localhost:8000/ecommerce_app/place_single_order/

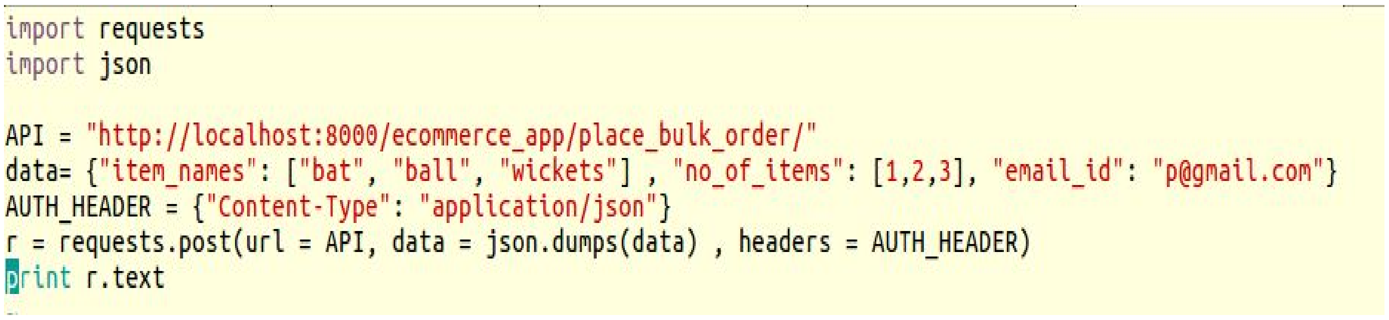
Sorry we are left only with 5 items for item: ball.Please reduce your quantity

Bulk Order
API: http://localhost:8000/ecommerce_app/place_bulk_order
Description:

- Place bulk order
- The client in this case is another python script, which on execution places bulk order in Order table.
- It iterates over item and their quantity, and saves information in Order table.

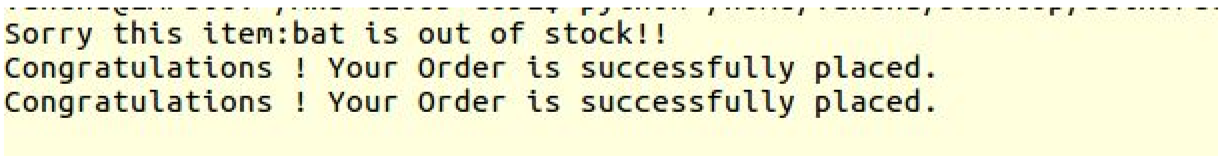
param request: POST

Snapshots
The client in this case is another python script, which on execution places bulk order in Order table.
It iterates over item and their quantity, and saves information in Order table.



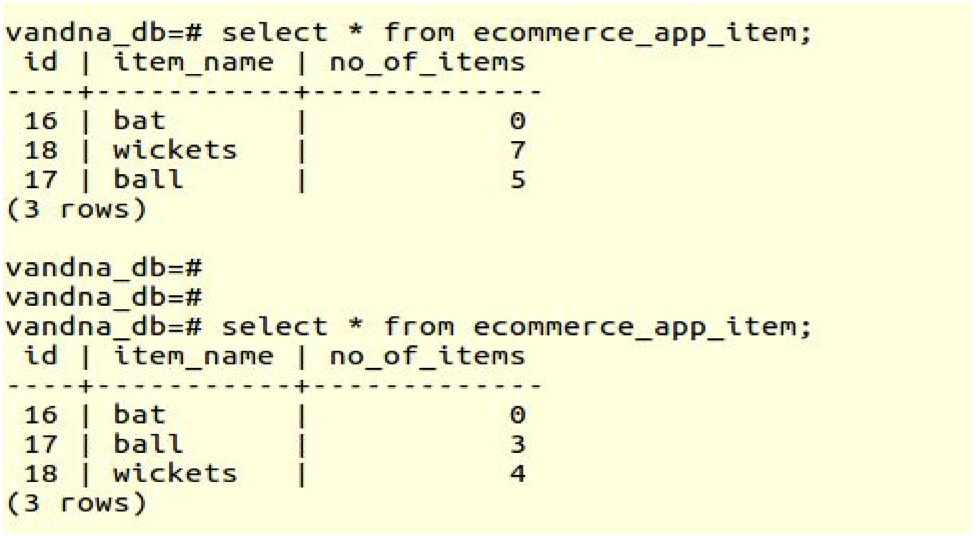
```
import requests
import json

API = "http://localhost:8000/ecommerce_app/place_bulk_order/"
data= {"item_names": ["bat", "ball", "wickets"] , "no_of_items": [1,2,3], "email_id": "p@gmail.com"}
AUTH_HEADER = {"Content-Type": "application/json"}
r = requests.post(url = API, data = json.dumps(data) , headers = AUTH_HEADER)
print r.text
```

In bulk order, we have placed order for 1 bat, 2 balls, 3 wickets.
We can also see the below response we get on executing our script for placing bulk order. Since, the bats are out of stock, this order couldn't be placed.balls and wickets order is successfully placed and also their quantity is decremented from Item table after the order is placed.



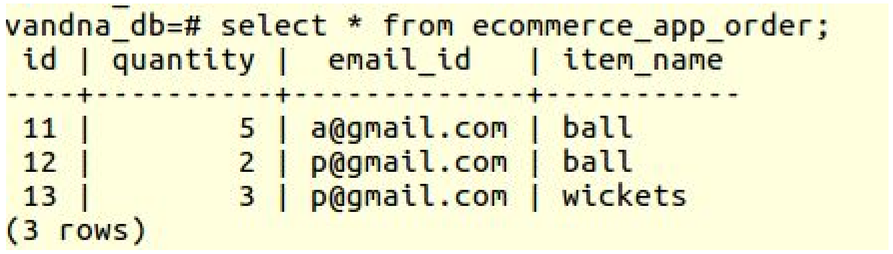
```
Sorry this item:bat is out of stock!!
Congratulations ! Your Order is successfully placed.
Congratulations ! Your Order is successfully placed.
```

Before (first table) and After (second table) placing order, postgreSQL Item tables look like this:



```
vandna_db=# select * from ecommerce_app_item;
 id | item_name | no_of_items
----+-----------+-------------
 16 | bat       |           0
 18 | wickets   |           7
 17 | ball      |           5
(3 rows)

vandna_db=#
vandna_db=#
vandna_db=# select * from ecommerce_app_item;
 id | item_name | no_of_items
----+-----------+-------------
 16 | bat       |           0
 17 | ball      |           3
 18 | wickets   |           4
(3 rows)
```

After placing order, postgreSQL Order tables look like this:



```
vandna_db=# select * from ecommerce_app_order;
 id | quantity |   email_id    | item_name
----+----------+---------------+-----------
 11 |        5 | a@gmail.com | ball
 12 |        2 | p@gmail.com | ball
 13 |        3 | p@gmail.com | wickets
(3 rows)
```

All orders

Get all orders

API: http://localhost:8000/ecommerce_app/get_all_orders/

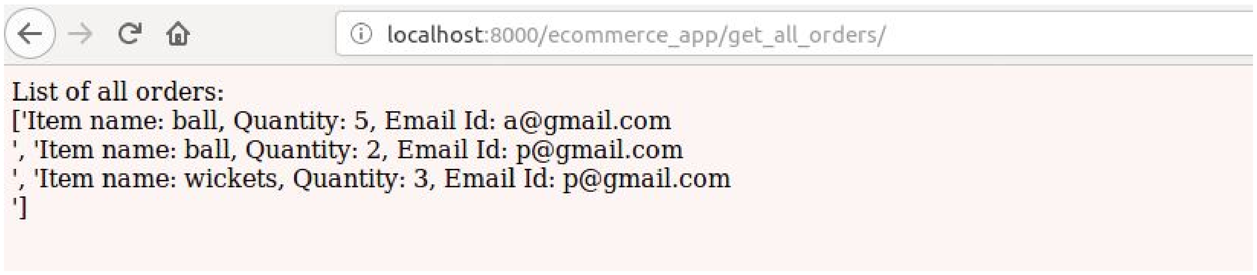
Description:

- Get all entries from Order table.

Param request: GET

Return: List of all orders in Order table.

Snapshot



List of all orders:
['Item name: ball, Quantity: 5, Email Id: a@gmail.com
', 'Item name: ball, Quantity: 2, Email Id: p@gmail.com
', 'Item name: wickets, Quantity: 3, Email Id: p@gmail.com
']