Lost and Found RESTful API

These are the endpoints available in the lost and found application.

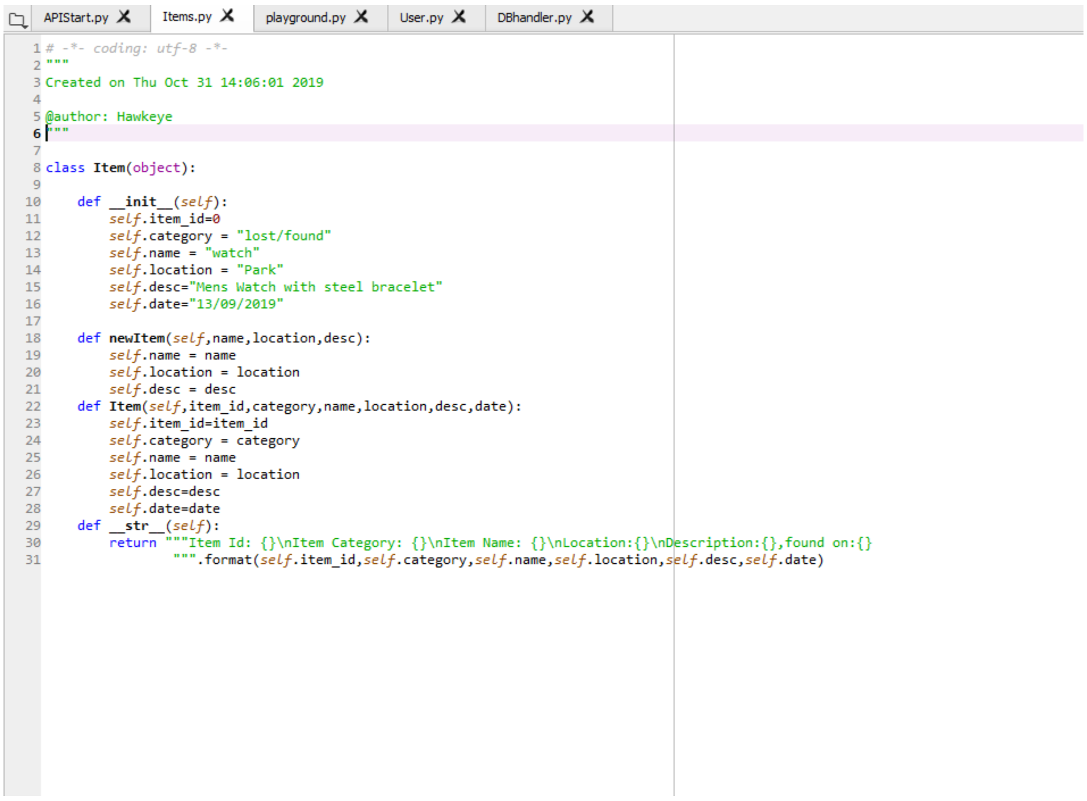
Table 1 End points Detail

Method Type	Method Name	Endpoint	Parameters	Description
POST	Create/Add Item	'items/create'	In json format name, location and description	Adds a new item in lost and found Application
POST	Update Item	'items/update'	In json format Item_Id, name, location and description	Updates an Already Existing Item provided that Item_id is given.
GET	View Items	'Items/view'	------	Returns a list of all items present in the lost and found Application
DELETE	Delete Item	'/items/delete/<int:item_id>'	Give item_id in URL after the delete/'Here'	Deletes an item from the Application on basis of given item_id
GET	Search Item by Location	'/item/search/<string:loc>'	Give location of the item in the url	Returns a list of Items by flirting on the basis of location
PUT	Search Item by Name	'/item/search/<string:name>'	Give name of the item in the url	Returns a list of Items by flirting on the basis of name
POST	Register User	'/user/register'	In Json format Username, email, password	Registers a user in the Lost and Found App
POST	Login User	'/user/login'	In Json format Email, password	Login's a user in the Lost and Found App

Implementation details

1. First of all, I implemented the models required for this application such as Items class and User class shown in the figures 1 and figure 2.
2. Then I implemented a Database Handler class which will be responsible for handling the connection and all the other CRUD operations in the database shown in the figure 3.
3. Then I implemented the Flask entry point and defined all the endpoints shown in the Table 1 above
4. Followed the Modular approach to implement this application.

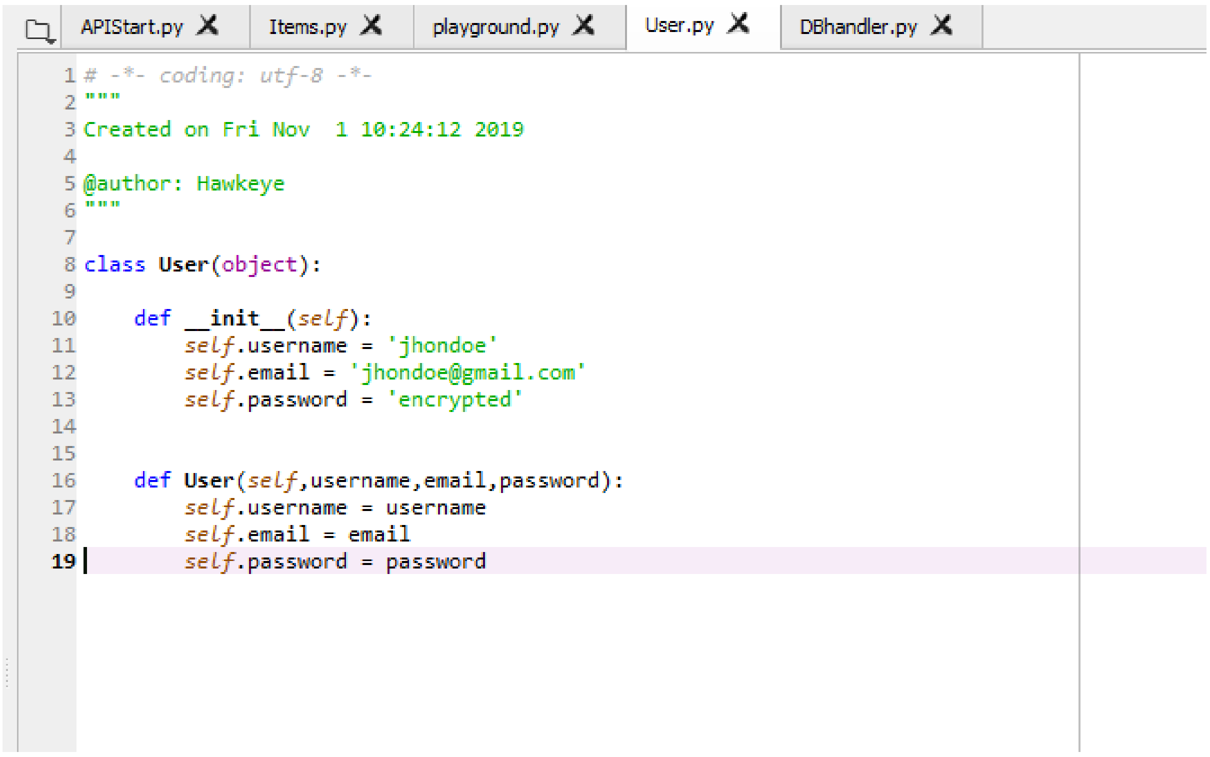
Figure 1



```python
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 31 14:06:01 2019

@author: Hawkeye
"""

class Item(object):

    def __init__(self):
        self.item_id=0
        self.category = "lost/found"
        self.name = "watch"
        self.location = "Park"
        self.desc="Mens Watch with steel bracelet"
        self.date="13/09/2019"

    def newItem(self,name,location,desc):
        self.name = name
        self.location = location
        self.desc = desc
    def Item(self,item_id,category,name,location,desc,date):
        self.item_id=item_id
        self.category = category
        self.name = name
        self.location = location
        self.desc=desc
        self.date=date
    def __str__(self):
        return """Item Id: {}\nItem Category: {}\nItem Name: {}\nLocation:{}\nDescription:{},found on:{}
                """.format(self.item_id,self.category,self.name,self.location,self.desc,self.date)
```

Figure 1 Items Class

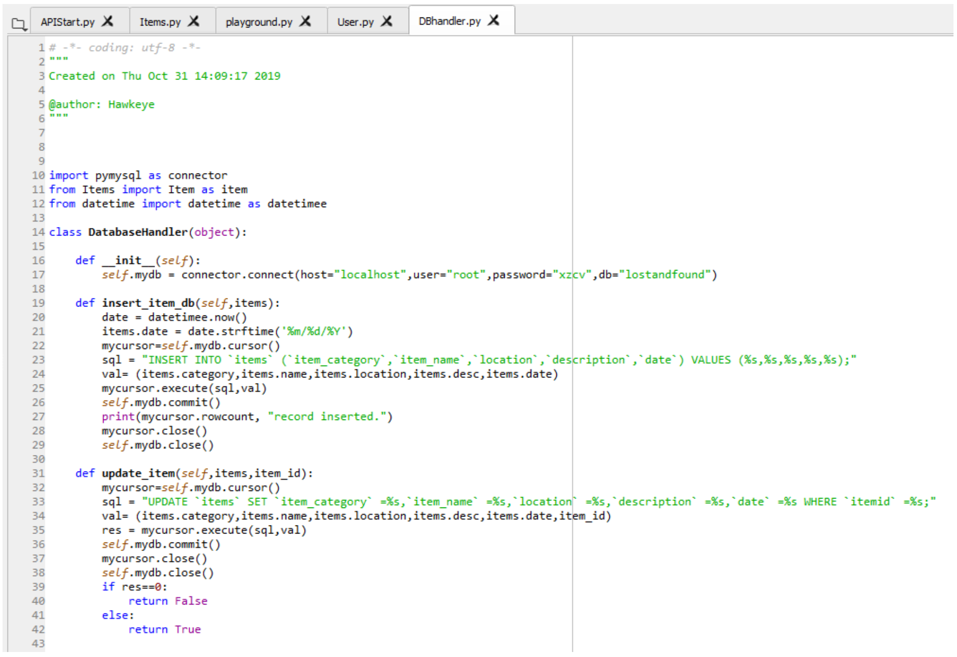
Figure 2



```python
# -*- coding: utf-8 -*-
"""
Created on Fri Nov  1 10:24:12 2019

@author: Hawkeye
"""

class User(object):

    def __init__(self):
        self.username = 'jhondoe'
        self.email = 'jhondoe@gmail.com'
        self.password = 'encrypted'


    def User(self,username,email,password):
        self.username = username
        self.email = email
        self.password = password
```

Figure 2 User Class

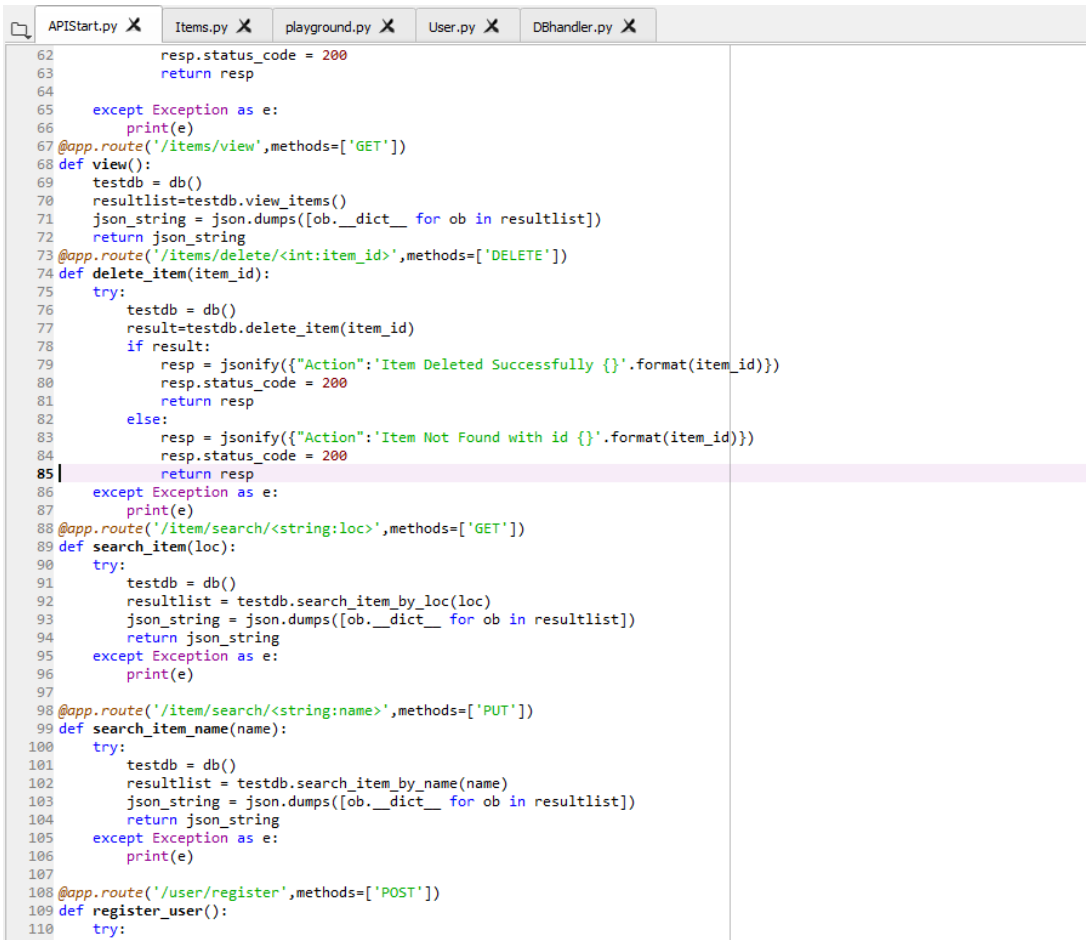
Figure 3



```python
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 31 14:09:17 2019

@author: Hawkeye
"""



import pymysql as connector
from Items import Item as item
from datetime import datetime as datetimee

class DatabaseHandler(object):

    def __init__(self):
        self.mydb = connector.connect(host="localhost",user="root",password="xzcv",db="lostandfound")

    def insert_item_db(self,items):
        date = datetimee.now()
        items.date = date.strftime('%m/%d/%Y')
        mycursor=self.mydb.cursor()
        sql = "INSERT INTO `items` (`item_category`,`item_name`,`location`,`description`,`date`) VALUES (%s,%s,%s,%s,%s);"
        val= (items.category,items.name,items.location,items.desc,items.date)
        mycursor.execute(sql,val)
        self.mydb.commit()
        print(mycursor.rowcount, "record inserted.")
        mycursor.close()
        self.mydb.close()

    def update_item(self,items,item_id):
        mycursor=self.mydb.cursor()
        sql = "UPDATE `items` SET `item_category` =%s,`item_name` =%s,`location` =%s,`description` =%s,`date` =%s WHERE `itemid` =%s;"
        val= (items.category,items.name,items.location,items.desc,items.date,item_id)
        res = mycursor.execute(sql,val)
        self.mydb.commit()
        mycursor.close()
        self.mydb.close()
        if res==0:
            return False
        else:
            return True

```

Figure 3 Database Handler Class

Figure 4



```python
                resp.status_code = 200
                return resp

        except Exception as e:
            print(e)
@app.route('/items/view',methods=['GET'])
def view():
        testdb = db()
        resultlist=testdb.view_items()
        json_string = json.dumps([ob.__dict__ for ob in resultlist])
        return json_string
@app.route('/items/delete/<int:item_id>',methods=['DELETE'])
def delete_item(item_id):
        try:
            testdb = db()
            result=testdb.delete_item(item_id)
            if result:
                resp = jsonify({"Action":'Item Deleted Successfully {}'.format(item_id)})
                resp.status_code = 200
                return resp
            else:
                resp = jsonify({"Action":'Item Not Found with id {}'.format(item_id)})
                resp.status_code = 200
                return resp
        except Exception as e:
            print(e)
@app.route('/item/search/<string:loc>',methods=['GET'])
def search_item(loc):
        try:
            testdb = db()
            resultlist = testdb.search_item_by_loc(loc)
            json_string = json.dumps([ob.__dict__ for ob in resultlist])
            return json_string
        except Exception as e:
            print(e)

@app.route('/item/search/<string:name>',methods=['PUT'])
def search_item_name(name):
        try:
            testdb = db()
            resultlist = testdb.search_item_by_name(name)
            json_string = json.dumps([ob.__dict__ for ob in resultlist])
            return json_string
        except Exception as e:
            print(e)

@app.route('/user/register',methods=['POST'])
def register_user():
        try:
```

Figure 4 Flask Entry point