

Lost and Found RESTful API

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

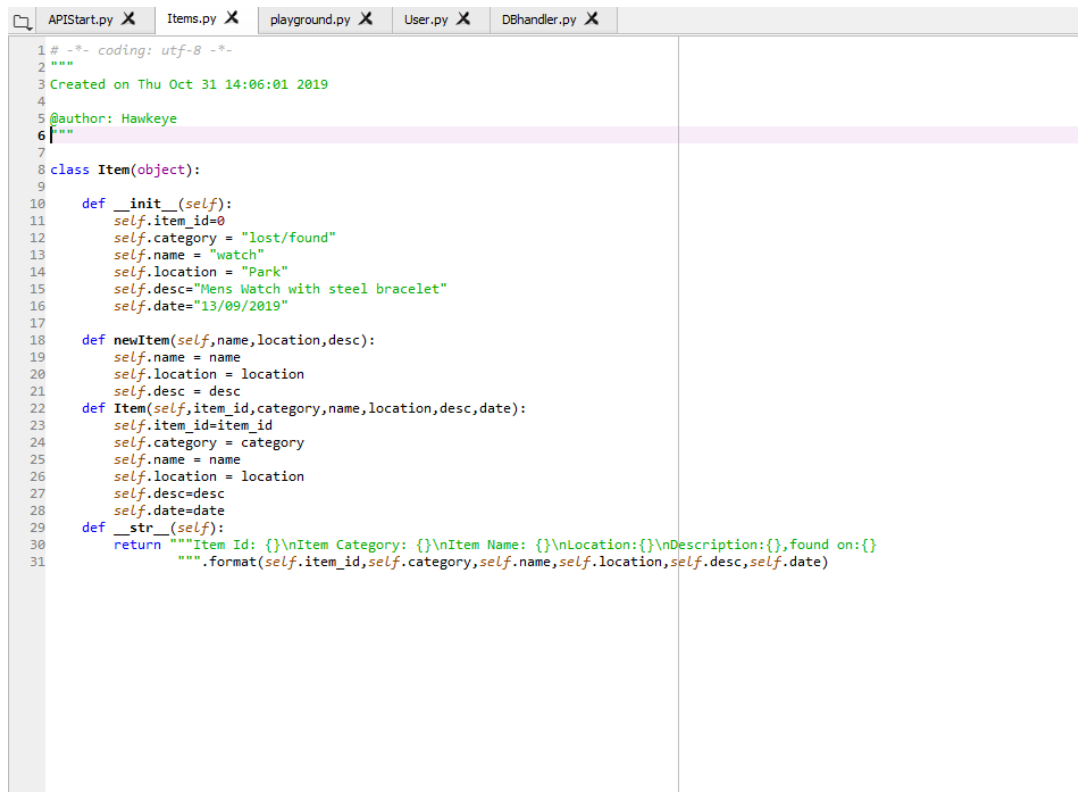
Table 1 End points Detail

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

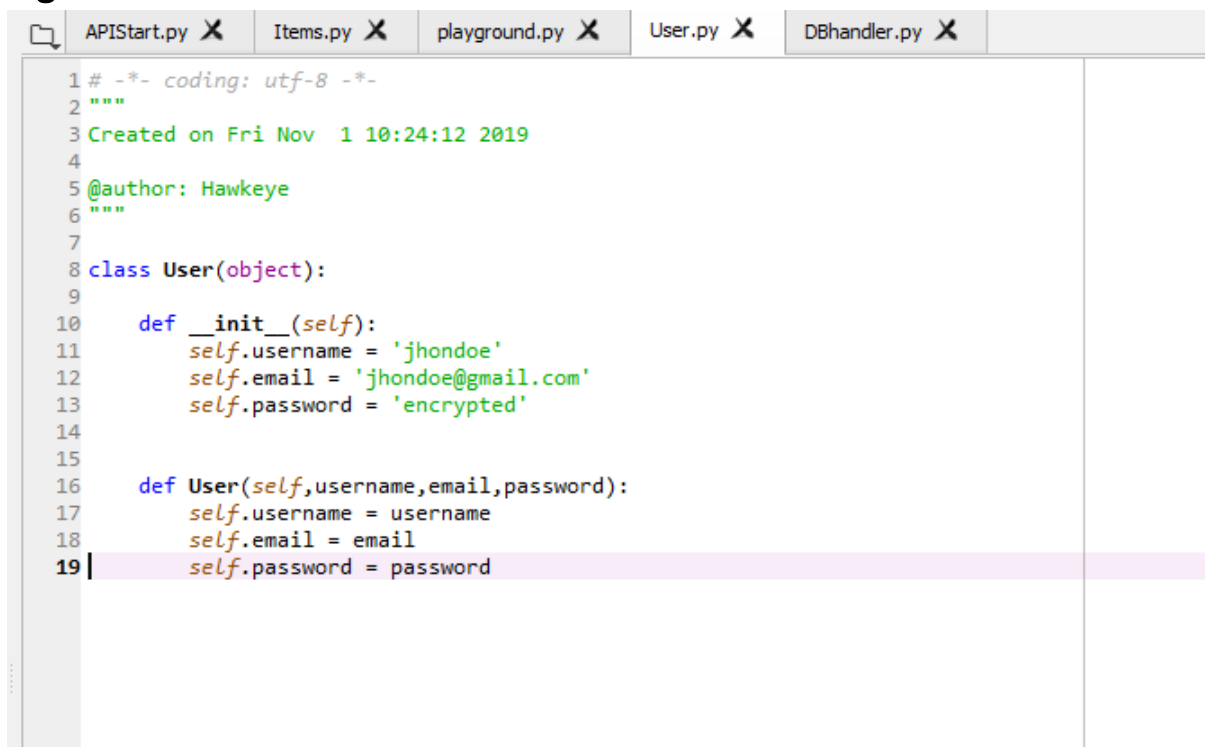


The screenshot shows a code editor with a tab labeled 'Items.py'. The code defines a class 'Item' that inherits from 'object'. It includes an 'init' method for initializing attributes like item_id, category, name, location, description, and date. It also has a 'newItem' method for creating new items and a '__str__' method for string representation. The code is as follows:

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Oct 31 14:06:01 2019
4
5 @author: Hawkeye
6 """
7
8 class Item(object):
9
10     def __init__(self):
11         self.item_id=0
12         self.category = "lost/found"
13         self.name = "watch"
14         self.location = "Park"
15         self.desc="Mens Watch with steel bracelet"
16         self.date="13/09/2019"
17
18     def newItem(self,name,location,desc):
19         self.name = name
20         self.location = location
21         self.desc = desc
22
23     def Item(self,item_id,category,name,location,desc,date):
24         self.item_id=item_id
25         self.category = category
26         self.name = name
27         self.location = location
28         self.desc=desc
29         self.date=date
30
31     def __str__(self):
32         return """Item Id: {} \nItem Category: {} \nItem Name: {} \nLocation: {} \nDescription: {},found on: {}
33         """.format(self.item_id,self.category,self.name,self.location,self.desc,self.date)
```

Figure 1 Items Class

Figure 2



The screenshot shows a code editor with a tab labeled 'User.py'. The code defines a class 'User' that inherits from 'object'. It includes an 'init' method for initializing attributes like username, email, and password. It also has a 'User' method for creating new users. The code is as follows:

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Fri Nov 1 10:24:12 2019
4
5 @author: Hawkeye
6 """
7
8 class User(object):
9
10     def __init__(self):
11         self.username = 'jhondoe'
12         self.email = 'jhondoe@gmail.com'
13         self.password = 'encrypted'
14
15
16     def User(self,username,email,password):
17         self.username = username
18         self.email = email
19         self.password = password
```

Figure 2 User Class

Figure 3

```
APISStart.py X Items.py X playground.py X User.py X DBHandler.py X
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Oct 31 14:09:17 2019
4
5 @author: Hawkeye
6 """
7
8
9
10 import pymysql as connector
11 from Items import Item as item
12 from datetime import datetime as datetime
13
14 class DatabaseHandler(object):
15
16     def __init__(self):
17         self.mydb = connector.connect(host="localhost",user="root",password="xzcv",db="lostandfound")
18
19     def insert_item_db(self,item):
20         date = datetime.now()
21         items.date = date.strftime('%m/%d/%Y')
22         mycursor=self.mydb.cursor()
23         sql = "INSERT INTO `items` (`item_category`,`item_name`,`location`,`description`,`date`) VALUES (%s,%s,%s,%s,%s);"
24         val= (items.category,item.name,item.location,item.desc,item.date)
25         mycursor.execute(sql,val)
26         self.mydb.commit()
27         print(mycursor.rowcount, "record inserted.")
28         mycursor.close()
29         self.mydb.close()
30
31     def update_item(self,item,item_id):
32         mycursor=self.mydb.cursor()
33         sql = "UPDATE `items` SET `item_category` =%s,`item_name` =%s,`location` =%s,`description` =%s,`date` =%s WHERE `itemid` =%s;"
34         val= (items.category,item.name,item.location,item.desc,item.date,item_id)
35         res = mycursor.execute(sql,val)
36         self.mydb.commit()
37         mycursor.close()
38         self.mydb.close()
39         if res==0:
40             return False
41         else:
42             return True
43
```

Figure 3 Database Handler Class

Figure 4

```
APISStart.py X Items.py X playground.py X User.py X DBHandler.py X
62     resp.status_code = 200
63     return resp
64
65     except Exception as e:
66         print(e)
67 @app.route('/items/view',methods=['GET'])
68 def view():
69     testdb = db()
70     resultlist=testdb.view_items()
71     json_string = json.dumps([ob.__dict__ for ob in resultlist])
72     return json_string
73 @app.route('/items/delete/<int:item_id>',methods=['DELETE'])
74 def delete_item(item_id):
75     try:
76         testdb = db()
77         result=testdb.delete_item(item_id)
78         if result:
79             resp = jsonify({"Action":'Item Deleted Successfully {}'.format(item_id)})
80             resp.status_code = 200
81             return resp
82         else:
83             resp = jsonify({"Action":'Item Not Found with id {}'.format(item_id)})
84             resp.status_code = 200
85     except Exception as e:
86         print(e)
87 @app.route('/item/search/<string:loc>',methods=['GET'])
88 def search_item(loc):
89     try:
90         testdb = db()
91         resultlist = testdb.search_item_by_loc(loc)
92         json_string = json.dumps([ob.__dict__ for ob in resultlist])
93         return json_string
94     except Exception as e:
95         print(e)
96
97
98 @app.route('/item/search/<string:name>',methods=['PUT'])
99 def search_item_name(name):
100     try:
101         testdb = db()
102         resultlist = testdb.search_item_by_name(name)
103         json_string = json.dumps([ob.__dict__ for ob in resultlist])
104         return json_string
105     except Exception as e:
106         print(e)
107
108 @app.route('/user/register',methods=['POST'])
109 def register_user():
110     try:

```

Figure 4 Flask Entry point