



KLE Technological
University
Creating Value
Leveraging Knowledge

PROJECT REPORT ON

A Python-Powered WhatsApp Bot with MongoDB for

Data Handling

Submitted in partial fulfillment of the requirements for the award of the degree

Bachelor of Engineering

In

Mechanical Engineering

Submitted by

Mr. Sudeep V Patil

01FE20BME154



2023-24



School of Mechanical Engineering

CERTIFICATE

Certified that the Project work carried out by Mr. Sudeep V Patil, SRN 01FE20BME154, a bonafide student of **K L E Technological University, Hubballi**, in partial fulfillment for the award of Bachelor of Engineering /Bachelor of Technology in School of Mechanical Engineering of the during the year 2024-25. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Project report deposited in the school library. The Project report has been approved as it satisfies the academic requirements in the said Degree.

Name: Praveen Murugod
Signature:
(University Guide)

Name: Dr. B B Kottursetter
Signature:
(Head, SME)

Name:
Signature:
(Registrar)

External Viva
Name of the examiners Signature with date
1
2.

Springevening Pvt ltd

Springevening private limited

Shri Vijay Concrete Works

Badami road, Bagalkote - 587101

CERTIFICATE

Certified that the Project work carried out by Mr. Sudeep V Patil SRN 01FE20BME154, a bonafide student of **K L E Technological University, Hubballi**, in partial fulfillment for the award of Bachelor of Engineering in School of Mechanical Engineering of the during the year 2024-25. It is certified that, she has completed the Project work satisfactorily.

Name: Shyamsunder Sedemkar

Signature:

(Industry Guide/Mentor)

Name: Dr. B B Kottursetter

Signature:

(Head of organization)

ACKNOWLEDGEMENT

Abstract

This project, titled "A Python-Powered WhatsApp Bot with MongoDB for Data Handling," explores the development of a robust chatbot application leveraging Python, MongoDB, and the WhatsApp Business API through MessageBird. The primary objective is to create a scalable, efficient, and interactive bot capable of handling real-time user interactions and managing data dynamically. The bot utilizes Python's Flask framework to handle incoming messages and process user inputs. MongoDB is employed for data storage, taking advantage of its flexibility and scalability to manage unstructured data effectively. The bot's architecture is modular, with distinct service and utility modules to ensure maintainability and extensibility. Key features include user authentication, dynamic question handling, and efficient CRUD operations for managing user and conversation data. Comprehensive unit and integration tests ensure the reliability and stability of the bot. Performance optimizations, such as database indexing and connection pooling, enhance the bot's responsiveness and scalability. This project serves as a foundational example of building a scalable and efficient chatbot using modern technologies, with potential extensions including advanced natural language processing capabilities, enhanced security measures, and the development of an admin dashboard for better data management. The principles and techniques demonstrated herein can be adapted to a wide range of chatbot applications, making this project a valuable resource for developers and organizations seeking to implement similar solutions.

Table of Content

Chapter 1: Introduction

- 1.1 Project Overview
- 1.2 Objectives
- 1.3 Scope
- 1.4 Technologies used

Chapter 2: Project Setup

- 2.1 Development Environment
- 2.2 Directory Structure

Chapter 3: Design and Architecture

- 3.1 System Architecture
- 3.2 Workflow Diagram
- 3.3 Database Schema

Chapter 4: WhatsApp API Integration

- 4.1 Overview of WhatsApp Business API
- 4.2 Setting Up WhatsApp Sandbox
- 4.3 Authenticating and Connecting to WhatsApp API
- 4.4 Sending and Receiving Messages

Chapter 5: Python Implementation

- 5.1 Project Structure
- 5.2 Initializing the Bot
- 5.3 Handling Incoming Message, Message Parsing and Processing,
Responding to User Queries

Chapter 5: MongoDB Integration

- 6.1 Introduction to MongoDB
- 6.2 Connecting Python to MongoDB
- 6.3 CRUD Operations in MongoDB
- 6.4 Storing and Retrieving Bot Data

Chapter 6: Result

Chapter 7: Conclusion

LIST OF FIGURES

Figure No	Description	Page No
2.1	Setting of webhook	12
2.2	Structure of service module	13
2.3	Structure of utility module	14
3.1	Architecture diagram	16
3.2	Flas app structure	17
3.3	Service module employee.py code	18
3.4	Expenses.py code	18
3.5	All these images are stock.py code	19
3.6	Conversatio.py code	20
3.7	Conversatio.py continuation code	20
3.8	Conversatio.py continuation code	21
3.9	Conversatio.py continuation code	21
3.10	Mongodb.py code	22
3.11	Message.py code	22
3.12	Workflow diagram	24
3.13	Actionschema.py code	28
3.14	Employschema.py code	29
3.15	Mappingschema.py code	30
4.1	Authenticate with MessageBird	31
4.2	Sending Messages	32

Figure No	Description	Page No
4.3	Receiving Messages	32
5.1	App.py code	35
7.1	WhatsApp interface for stock Overflow working	40
7.2	Backend storage of data related to stock	40
7.3	WhatsApp interface for inventory	41
7.4	Backend storage of data related to inventory	41
7.5	WhatsApp interfacing for expenses	42
7.6	Backend storage of data related to expenses	42

Chapter 1: Introduction

1.1 Project Overview

This project aims to develop an intelligent WhatsApp chatbot that leverages the capabilities of Python for its core logic and MongoDB for efficient data storage and management. The bot will facilitate seamless communication with users, providing instant responses and handling data dynamically. This system is designed to be scalable, secure, and user-friendly, catering to various use cases such as customer support, information dissemination, and automated tasks. This is live project based on the requirement of the company management to manage all there daily stock, expenses, and inventory so they can store, handle, all the data remotely in a systematic manner.

1.2 Objectives

The project aims to develop an intelligent and efficient WhatsApp chatbot powered by Python and MongoDB for seamless communication and data management. The key objectives of this project are:

1. **Develop a Functional WhatsApp Chatbot**
 - Create a robust and responsive chatbot using Python that can interact with users on WhatsApp.
 - Ensure the bot can handle various types of user queries and provide appropriate responses.
2. **Integrate MongoDB for Data Management**
 - Utilize MongoDB as the backend database to store and manage user data, conversation logs, and other relevant information.
 - Implement efficient data retrieval and storage mechanisms to support real-time interactions.
3. **Enable Real-Time Communication**
 - Ensure the chatbot can process incoming messages and send responses in real-time, providing a seamless user experience.
 - Maintain consistent and reliable communication between the chatbot and users.
4. **Ensure Data Security and Privacy**
 - Implement robust security measures to protect user data and ensure secure communication channels.
 - Comply with relevant data privacy regulations and ensure the chatbot handles sensitive information appropriately.
5. **Scalability and Performance Optimization**
 - Design the chatbot architecture to be scalable, allowing it to handle a growing number of users and interactions without performance degradation.
 - Optimize the bot's performance to provide quick and accurate responses.

6. User-Friendly Interaction Design

- Develop interactive menus and commands to guide users through the chatbot's functionalities.
- Ensure the user interface is intuitive and easy to use.

7. Testing and Debugging

- Conduct thorough testing, including unit and integration tests, to ensure the chatbot functions correctly under various scenarios.
- Implement effective debugging techniques to identify and resolve issues promptly.

1.3 Scope

1. Setting Up the Development Environment

- **Software Installation:** Install Python (version 3.7 or higher), MongoDB, Flask, and necessary libraries.
- **Configuration:** Configure the development environment to ensure all tools and dependencies are properly set up.
- **Version Control:** Set up a version control system (e.g., Git) to manage code changes and collaborate effectively.

2. Integrating the WhatsApp API

- **WhatsApp Business Account:** Register for a WhatsApp Business Account and obtain API credentials.
- **API Configuration:** Configure the WhatsApp API with the development environment, including setting up the sandbox for testing.
- **Webhook Setup:** Create and configure webhooks to handle incoming and outgoing messages between the chatbot and WhatsApp.

3. Developing Chatbot Logic in Python

- **Core Logic Development:** Implement the core logic of the chatbot using Python, handling user messages and generating appropriate responses.
- **Message Processing:** Develop functions to process and interpret user inputs, including basic Natural Language Processing (NLP) capabilities.
- **Interactive Features:** Implement interactive menus and commands to guide users through various functionalities of the chatbot.
- **User Personalization:** Develop features to personalize interactions based on user profiles and historical data.

4. Using MongoDB for Data Management

- **Database Setup:** Set up a MongoDB database to store user data, conversation logs, and other relevant information.
- **Schema Design:** Design the database schema to support efficient data storage and retrieval.
- **CRUD Operations:** Implement Create, Read, Update, Delete (CRUD) operations to manage data within the database.
- **Data Security:** Ensure data security and integrity by implementing appropriate access controls and encryption mechanisms.

5. Testing and Deploying the Bot

- **Unit Testing:** Write and execute unit tests to verify the functionality of individual components.
- **Integration Testing:** Conduct integration testing to ensure seamless communication between the chatbot, WhatsApp API, and MongoDB.
- **Debugging:** Identify and resolve any issues or bugs in the chatbot's functionality.
- **Deployment Preparation:** Prepare the bot for deployment, ensuring all dependencies and configurations are correctly set up.
- **Cloud Deployment:** Deploy the chatbot on a cloud platform (e.g., Heroku, AWS) for real-world use.
- **Continuous Integration/Continuous Deployment (CI/CD):** Set up CI/CD pipelines to automate testing and deployment processes for ongoing development and updates.

1.4 Technologies Used

Python: Python will serve as the primary programming language for developing the chatbot's logic and functionalities. It offers simplicity, versatility, and a rich ecosystem of libraries for various tasks.

MongoDB: MongoDB will be used as the backend database for storing and managing user data, conversation logs, and other relevant information. Its flexibility and scalability make it well-suited for handling dynamic data in real-time applications.

WhatsApp Business API: The WhatsApp Business API will facilitate communication between the chatbot and WhatsApp users. It allows for sending and receiving messages, managing contacts, and automating interactions with users on the WhatsApp platform.

Flask (web framework): Flask will be employed as the web framework for developing the backend of the chatbot application. Its lightweight nature and simplicity make it ideal for building web applications, including APIs for handling HTTP requests and responses.

MessageBird: MessageBird is a cloud communications platform that provides a suite of APIs for businesses to engage with their customers via various communication channels, including SMS, Voice, Chat, and Email. It offers reliable and scalable solutions for businesses to send and receive messages globally.

Chapter 2: Project Setup

2.1. Development Environment

To set up the development environment for this project, you'll need to install and configure various tools and dependencies. Follow the steps below to prepare your development environment.

Required Libraries and Dependencies

The project requires several libraries and dependencies. Here is a list of the primary ones you'll need:

- Python (version 3.7 or higher)
- MongoDB
- Flask
- pymongo
- requests
- python-dotenv

Installing Python and MongoDB

Installing Python

1. **Download Python:** Go to the official [Python website](#) and download the latest version of Python 3.7 or higher for your operating system.
2. **Install Python:** Follow the installation instructions for your operating system. Ensure that you add Python to your system PATH during the installation process.
3. **Verify Installation:** Open a terminal or command prompt and type the following command to verify that Python is installed correctly:

Installing MongoDB

1. **Download MongoDB:** Visit the [MongoDB download center](#) and download the appropriate version of MongoDB for your operating system.
2. **Install MongoDB:** Follow the installation instructions for your operating system.
3. **Start MongoDB:** After installation, start the MongoDB server. For most systems, you can start MongoDB by running:

Setting Up WhatsApp API

To integrate the WhatsApp API, you will need to set up a WhatsApp Business Account and configure the API credentials.

Register for WhatsApp Business Account

1. **Create an Account:** Visit the WhatsApp Business API page and sign up for an account.
2. **Obtain API Credentials:** After registration, you will receive API credentials, including a WhatsApp Business API URL and access token.

Setting Up 'mbwebhook.py'

To set up a webhook for your WhatsApp bot, you need to create a Python script, `webhook.py`, that handles incoming HTTP POST requests from the WhatsApp API and processes the messages. This script will use the `requests` library to send responses back to WhatsApp and `json` to parse and handle JSON data.

```
import requests
import json

reqUrl = "https://conversations.messagebird.com/v1/webhooks/55dec2f1b5b4410ca05e7d4df05678bb"
headersList = {
    "Authorization": "AccessKey HMKzx01lI8tmpmahvny6scCaW",
    "Content-Type": "application/json"
}
payload = json.dumps({
    "url": "https://cb1b-117-247-26-91.ngrok-free.app/webhook"
})

response = requests.request("PATCH", reqUrl, data=payload, headers=headersList)
#this fine is ignored dd
print(response.text)
```

Figure 2.1 Setting of webhook

Setting Up 'ngroksetup.py'

Ngrok is a tool that allows you to expose a local server to the internet securely. It creates a secure tunnel to your localhost, making your local development environment accessible from the web. This is particularly useful for testing webhooks, APIs, and other services that require a public URL to interact with external systems.

2.2 Directory Structure

To structure our project efficiently, we created a directory named `service` that contains various modules (`__init__.py`, `employee.py`, `expenses.py`, `inventory.py`, and `stock.py`). Each module will handle different aspects of our WhatsApp bot's functionalities. This modular approach helps in organizing our code and making it more maintainable.

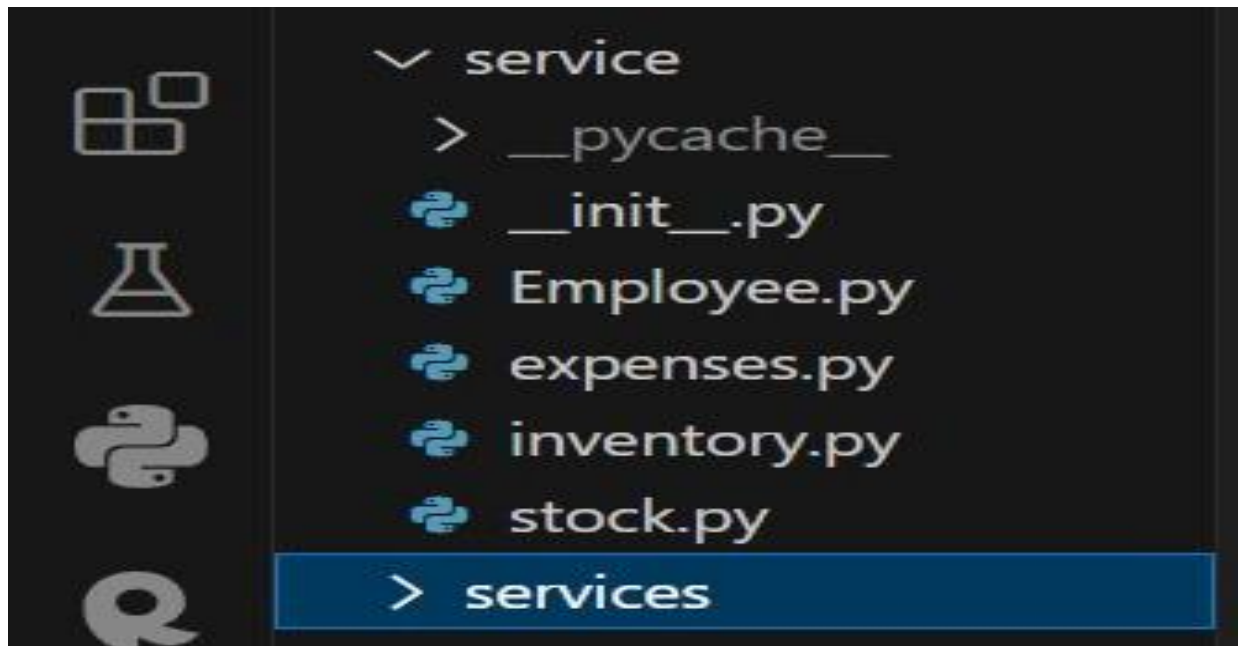


Figure 2.2 Structure of service module

Explanation of Each Module

1. **__init__.py:** This file is used to mark the directory as a Python package. It can be empty or used to initialize package-level variables.
2. **employee.py:** This module will manage employee-related functionalities, such as storing employee details, retrieving information, and handling queries related to employees.
3. **expenses.py:** This module will manage expense-related functionalities, such as tracking expenses, storing expense records, and handling queries related to expenses.
4. **inventory.py:** This module will manage inventory-related functionalities, such as managing inventory records, retrieving inventory information, and handling queries related to inventory.
5. **stock.py:** This module will manage stock-related functionalities, such as managing stock levels, retrieving stock information, and handling queries related to stock.

Creating a utility module with sub-files (conversation.py, message.py, and mongoDB.py) is a great way to organize common functionalities that can be reused across different parts of our project. Here's an explanation of what each file will do and how they fit into the context of "A Python-Powered WhatsApp Bot with MongoDB for Data Handling."

Structure:

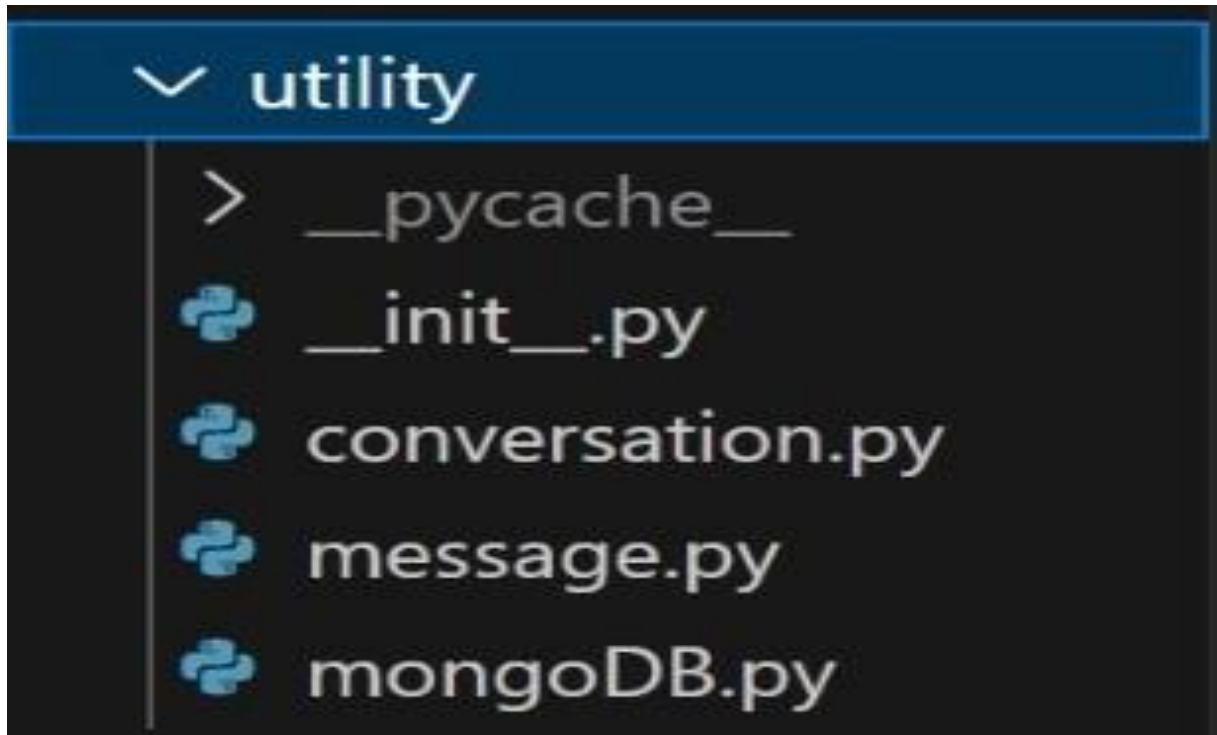


Figure 2.3 Structure of utility module

Chapter 3: Design and Architecture

3.1 System Architecture

The architecture of this project involves multiple components working together to provide a seamless user experience through WhatsApp, while handling data processing and storage efficiently in the backend. Here's a detailed breakdown of the system architecture.

1. Frontend: WhatsApp as the User Interface

- **User Interaction:** Users interact with the bot through WhatsApp, which serves as the user interface.
- **WhatsApp Business API:** This API allows the bot to send and receive messages from WhatsApp users.

2. Backend: Python with Flask for Handling Requests

- **Flask Application:** A lightweight WSGI web application framework for Python that handles incoming HTTP requests from the WhatsApp API.
- **Endpoints:** The Flask app exposes endpoints to handle different functionalities such as receiving messages, sending responses, and managing data.

3. Database: MongoDB for Storing User Data and Chat Logs

- **Data Storage:** MongoDB is used to store user information, chat logs, inventory details, employee records, expense data, and stock levels.
- **Data Management:** MongoDB collections are used to organize different types of data, making it easy to query and manage.

4. Integration: WhatsApp API for Message Exchange

- **Message Handling:** The WhatsApp API is used to handle message exchange between the users and the bot.
- **Webhook:** A webhook endpoint in the Flask app processes incoming messages and triggers appropriate responses.

Architecture Diagram

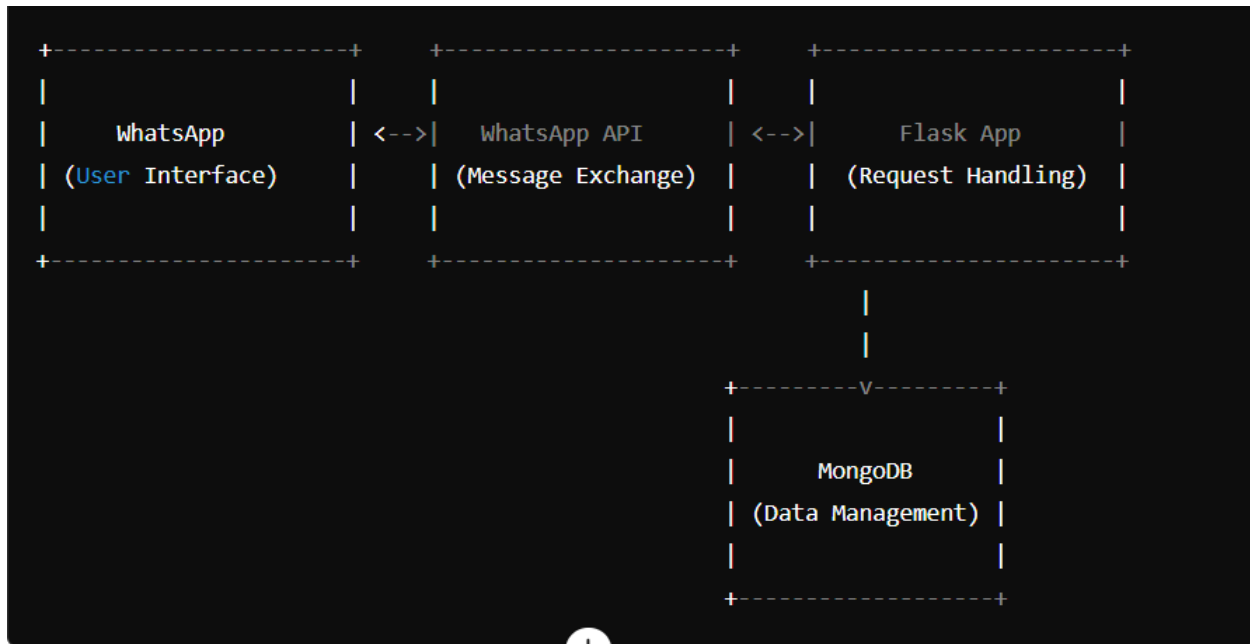


Figure 3.1 Architecture diagram

Components Breakdown

1. **WhatsApp (User Interface)**
 - Users send messages to the bot through their WhatsApp application.
 - Users receive responses from the bot in their WhatsApp application.
2. **WhatsApp API (Message Exchange)**
 - **Inbound Messages:** Receives messages from users and forwards them to the Flask app via HTTP POST requests.
 - **Outbound Messages:** Receives HTTP POST requests from the Flask app and forwards the messages to users.
3. **Flask App (Request Handling)**
 - **Webhook Endpoint:** Receives incoming messages from the WhatsApp API and processes them.
 - **Message Processing:** Determines the appropriate response based on the user's message and the business logic.
 - **Data Operations:** Interacts with MongoDB to store and retrieve data as needed.
 - **Service Modules:**
 - **employee.py:** Manages employee-related data.
 - **expenses.py:** Manages expense-related data.
 - **inventory.py:** Manages inventory data.
 - **stock.py:** Manages stock data.

4. MongoDB (Data Management)

- **Collections:**
 - **employees:** Stores employee data.
 - **expenses:** Stores expense records.
 - **inventory:** Stores inventory items.
 - **stock:** Stores stock records.
- **CRUD Operations:** Supports Create, Read, Update, and Delete operations for various data types.

Implementation Details

Flask App Structure (app.py)

```
app.py > ...
1  import json
2  from flask import Flask, jsonify, request
3  import sys
4
5  sys.path.append('utility')
6  sys.path.append('service')
7  from utility import conversation
8  import pymongo
9
10 app = Flask(__name__)
11 @app.route("/")
12 def hello():
13     return "Bot is alive from flask!"
14
15 @app.route('/webhook', methods=['POST'])
16 def bot():
17
18
19     data = request.json
20     conversation.intiate(data)
21     return {}
22 if __name__ == '__main__':
23     app.run()
```

Figure 3.2 Flask app structure

The service/employee.py module demonstrates a structured approach to managing employee data and operations within the context of the WhatsApp bot project. By encapsulating related functionalities into cohesive modules, the codebase becomes more maintainable, extensible, and conducive to collaborative development.

Service Modules Example

service/employee.py

```
service > Employee.py > ...
1  from flask import Flask, request
2
3  app = Flask(__name__)
4
5  def authenticate_user(mongo_handler, mobile_number, actions):
6
7      employee_doc = mongo_handler.find_one("employees", {"phoneNumber": mobile_number})
8      print(str(employee_doc))
9      if employee_doc:
10         employee_actions = employee_doc.get("actions", [])
11         #print(str(employee_actions))
12         for action in employee_actions:
13             if actions not in employee_actions:
14                 return False
15         return True
16     else:
17         return False
18
19 if __name__ == "__main__":
20     app.run(debug=True)
21
```

Figure 3.3 Service module employee.py code

service/expenses.py

```
service > expenses.py > addexpensesdb
1  from operator import index
2  from utility import message
3  from utility import conversation
4  from utility import mongoDB
5
6  def initiate(data):
7
8      return message.conversation_reply(data["conversation"]["id"], "Expense:What is today's expenses?")
9
10 ...
11
12 Description:
13 This function adds expense data to the database using information from the provided data.
14 It extracts the local date and time from the "updatedatetime" field in the message.
15 It prepares action data, including date, time, phone number, and initial expense information.
16 Params:
17 mongo_handler: MongoDB handler for database operations.
18 data:
19 Data containing information about the expense addition, including message and contact details.
20 ...
21
22
23 def addexpensesdb(mongo_handler, data):
24     localtime, localtime = data["message"]["updatedatetime"].split('T')
25     localtime, localtime = localtime.split('.')[:2]
26     actionData = {
27         "date": localtime,
28         "time": localtime,
29         "phoneNumber": data["contact"]["misdn"],
30         "conversations": [],
31         "action": "expense",
32         "product": "",
33         "id": data["message"]["id"],
34         "value": "0",
35     }
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
256
```

service/stock.py

```
def updatetockdb(mongo_handler, data):
    localtime, localtime = data["message"]["updatedDatetime"].split('T')
    phone_number = data["contact"]["msisdn"]
    # Define the query to find the document in the "stock" collection
    product_result, batch_result = extract_word(data["message"]["content"]["text"])
    query = {
        "date": localtime,
        "phoneNumber": phone_number,
        "conversations.value": batch_result
    }
    # Find the stock document by action_id
    stock_document = mongo_handler.find_one("stock", query)
    print(stock_document)
    new_conversation_item = {
        "action": "Stock",
        "product": product_result,
        "id": data["message"]["id"],
        "value": ""
    }
    if stock_document:
        # Get conversations from the expense document
        conversations = stock_document.get("conversations", [])
        update_data = {
            f"conversations.{len(conversations)}": new_conversation_item
        }
        conversations.append(new_conversation_item)
        mongo_handler.update_one("stock",
            stock_document["_id"],
            update_data
        )
```

```
'''
Description:
    Extracts words between single quotes in the input string.
Params:
    Input_string: The input string containing words enclosed in single quotes.
Response:
    Returns a tuple containing two elements:
    The word between single quotes representing the product.
    The word following the pattern "Batch [word] Stock" representing the batch.
'''

def extract_word(input_string):
    pattern_product = r"'(.*)'"
    pattern_batch = r"Batch (.*) Stock"
    matches_product = re.findall(pattern_product, input_string)
    matches_batch = re.findall(pattern_batch, input_string)
    return matches_product[0], matches_batch[0]

def ask4questions(batchno, convid):
    # Ask the first question
    message.conversation_reply(convid, "Stock Batch " + batchno + " Stock Product: 'Idly packet' Batch:1 How many?")
    # Ask the second question
    message.conversation_reply(convid, "Stock Batch " + batchno + " Stock Product: 'Dosa packet' Batch:1 How many?")
    # Ask the third question
    message.conversation_reply(convid, "Stock Batch " + batchno + " Stock Product: 'Idly Loose' Batch:1 How many?")
    # Ask the fourth question
    message.conversation_reply(convid, "Stock Batch " + batchno + " Stock Product: 'Dosa Loose' Batch:1 How many?")
```

Figures 3.5 All these images are stock.py code

```
'''
Description:
    Adds a new batch document to the stock collection in MongoDB.
Params:
    mongo_handler: MongoDB handler for database operations.
    data: Data containing the message information.
'''

def addbatchdb(mongo_handler, data):
    localtime, localtime = data["message"]["updatedDatetime"].split('T')
    localtime = localtime.split('.')[0]
    actionData = {
        "date": localtime,
        "time": localtime,
        "phoneNumber": data["contact"]["msisdn"],
        "conversations": [
            {
                "action": "Batch",
                "product": "",
                "id": data["message"]["id"],
                "value": "0",
            }
        ]
    }
    print(str(actionData))
    insert_result = mongo_handler.insert_one("stock", actionData, "action")
    action_id = str(insert_result.inserted_id)

    # Additional Mapping Information
    mapping_data = {
```

```
'''
Description:
    This function finds and updates a document in the "stock" collection in MongoDB based on certain conditions.
Params:
    mongo_handler: MongoDB handler for database operations.
    action_id: Identifier for the action or document to be updated.
    mapping_document: Mapping information used to identify the conversation.
    data: Data containing the message information.
'''

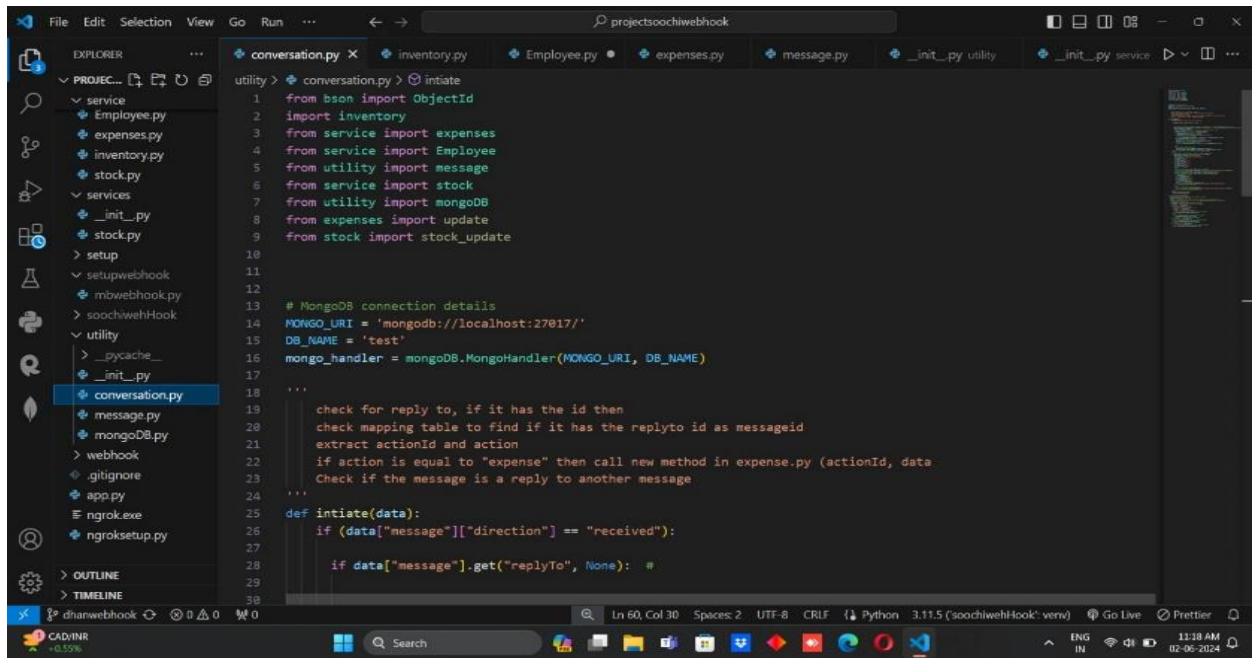
def stock_update(mongo_handler, action_id, mapping_document, data):
    # Find the expense document by action_id
    stock_document = mongo_handler.find_one("stock", {"_id": action_id})
    print(str(stock_document))

    if stock_document:
        # Get conversations from the expense document
        conversations = stock_document.get("conversations", [])
        # Iterate over conversations
        for index, conv in enumerate(conversations):
            if conv.get("id") == mapping_document["messageId"]:
                print("Inside if condition stock update")
                update_data = {
                    f"conversations.{index}.value": data["message"]["content"]["text"]
                }
                update_result = mongo_handler.update_one("stock", action_id, update_data)
                ask4questions(data["message"]["content"]["text"], data["conversation"]["id"])
```

```
1 from operator import index
2 from utility import message
3 from utility import conversation
4 from utility import mongoDB
5 import re
6
7
8 def initiate(data):
9     return message.conversation_reply(data["conversation"]["id"], "Batch: Which Batch?")
10
11
12
13
14 Descriptions:
15     This function is responsible for updating the stock document in the MongoDB collection.
16     It retrieves the existing stock document, finds the conversation that matches the provided
17     message ID, and updates the value of that conversation.
18
19 Params:
20     mongo_handler: MongoDB handler for database operations.
21     action_id: ID of the stock document to be updated.
22     mapping_document: Document containing mapping information.
23     data: Data containing the message information.
24
25 Response:
26     None
27
28
29 def updatetockdb(mongo_handler, data):
30     localtime, localtime = data["message"]["updatedDatetime"].split('T')
31     phone_number = data["contact"]["msisdn"]
```

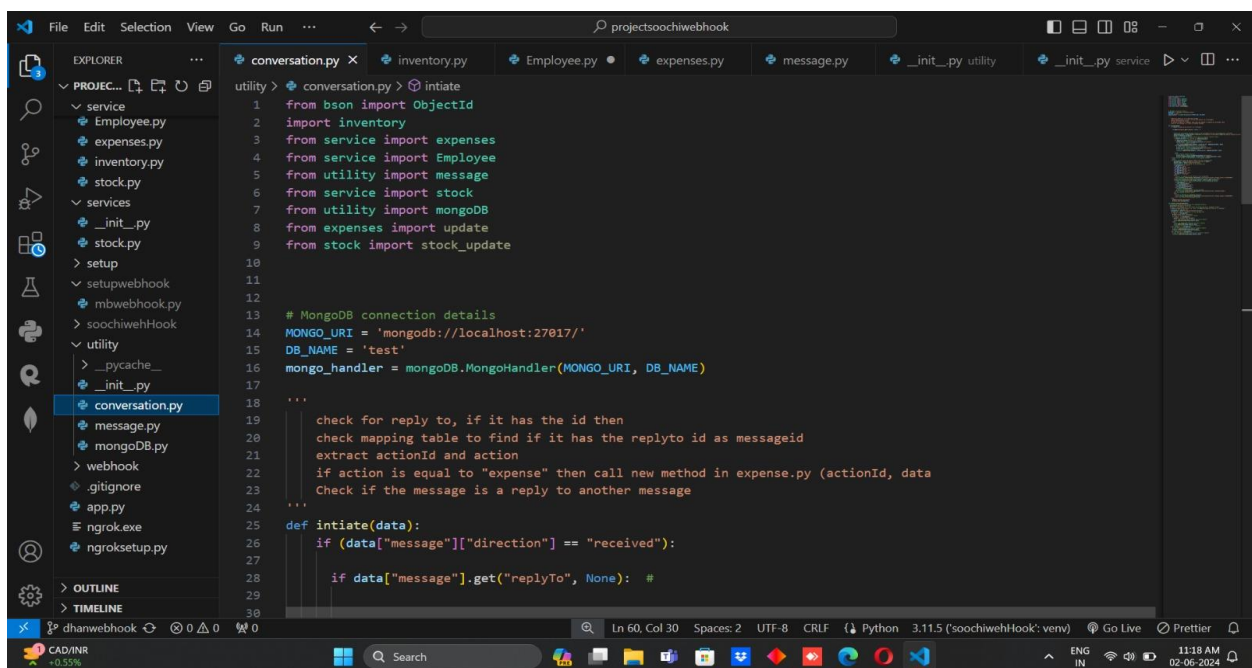
1. utility/conversation.py

This module handles conversation flows and logic for the chatbot. It could include functions for managing user interactions, handling different conversation states, and generating responses.



```
1 from bson import ObjectId
2 import inventory
3 from service import expenses
4 from service import Employee
5 from utility import message
6 from service import stock
7 from utility import mongoDB
8 from expenses import update
9 from stock import stock_update
10
11
12
13 # MongoDB connection details
14 MONGO_URI = 'mongodb://localhost:27017/'
15 DB_NAME = 'test'
16 mongo_handler = mongoDB.MongoHandler(MONGO_URI, DB_NAME)
17
18 ...
19 check for reply to, if it has the id then
20 check mapping table to find if it has the replyto id as messageid
21 extract actionId and action
22 if action is equal to "expense" then call new method in expense.py (actionId, data
23 Check if the message is a reply to another message
24
25 def initiate(data):
26     if (data["message"]["direction"] == "received"):
27
28         if data["message"].get("replyTo", None): #
```

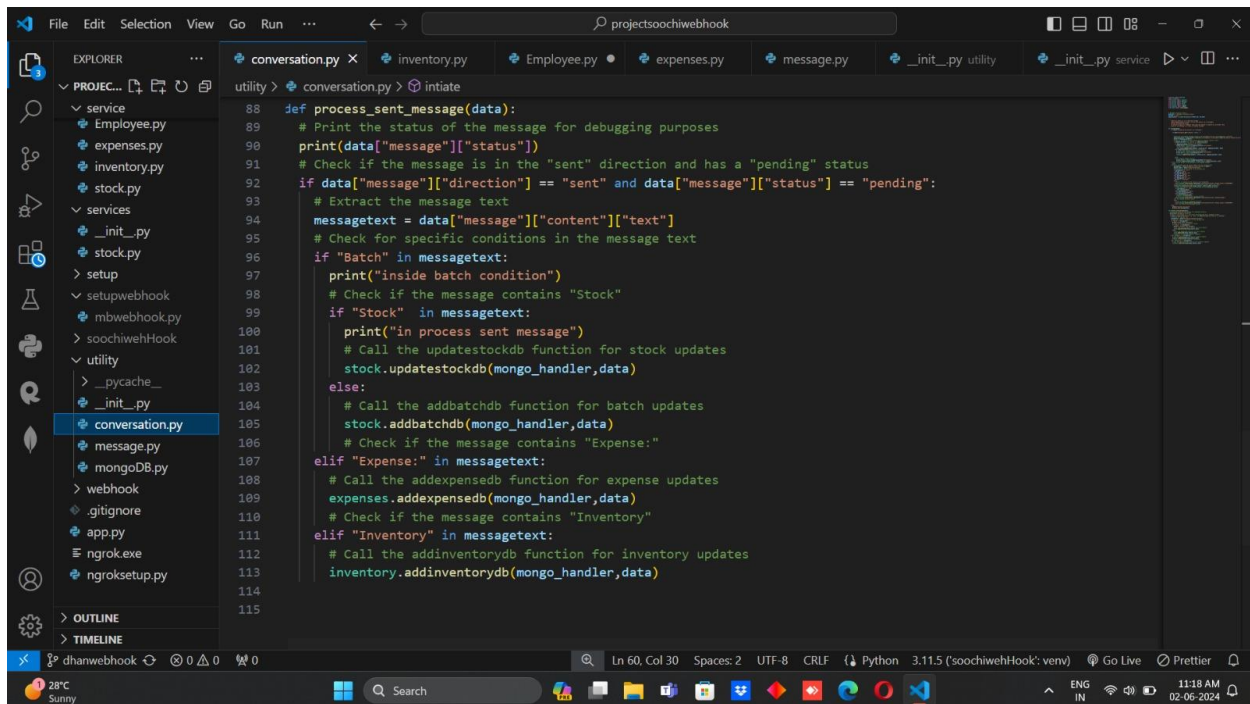
Figure 3.6 conversatio.py code



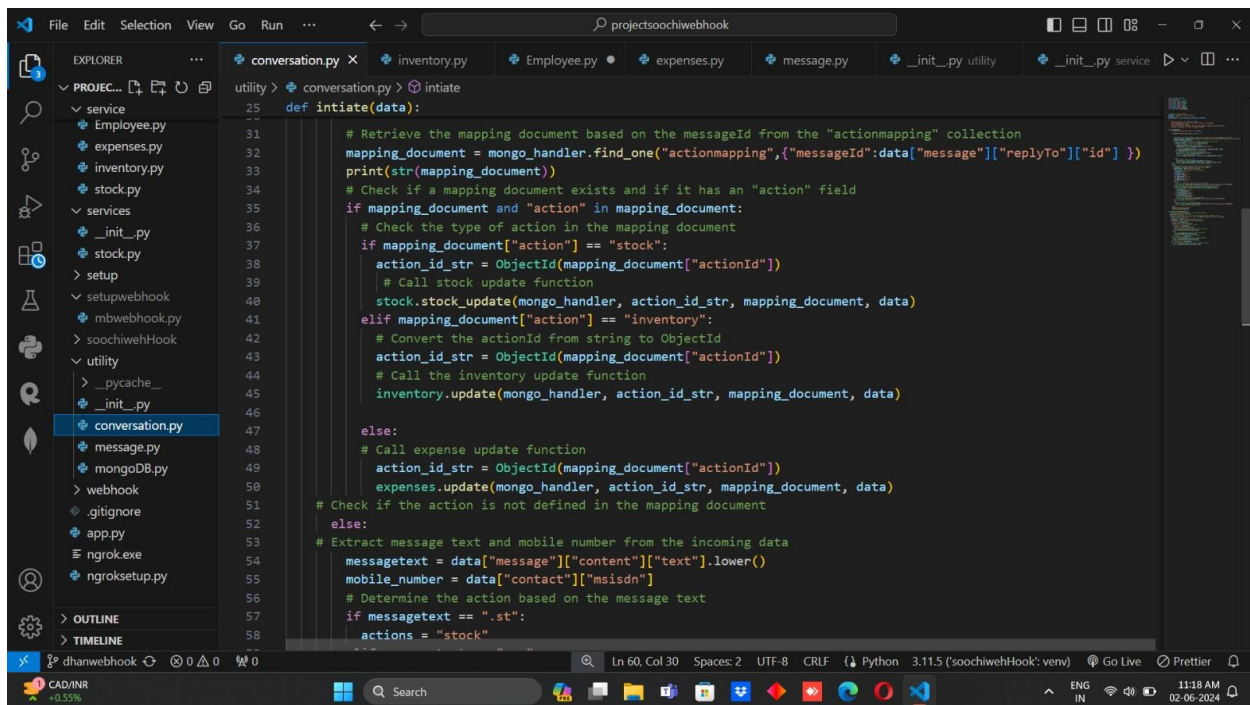
```
1 from bson import ObjectId
2 import inventory
3 from service import expenses
4 from service import Employee
5 from utility import message
6 from service import stock
7 from utility import mongoDB
8 from expenses import update
9 from stock import stock_update
10
11
12
13 # MongoDB connection details
14 MONGO_URI = 'mongodb://localhost:27017/'
15 DB_NAME = 'test'
16 mongo_handler = mongoDB.MongoHandler(MONGO_URI, DB_NAME)
17
18 ...
19 check for reply to, if it has the id then
20 check mapping table to find if it has the replyto id as messageid
21 extract actionId and action
22 if action is equal to "expense" then call new method in expense.py (actionId, data
23 Check if the message is a reply to another message
24
25 def initiate(data):
26     if (data["message"]["direction"] == "received"):
27
28         if data["message"].get("replyTo", None): #
```

Figure 3.7 conversatio.py continuation code

Figure 3.8 conversatio.py continuation code



```
88 def process_sent_message(data):
89     # Print the status of the message for debugging purposes
90     print(data["message"]["status"])
91     # Check if the message is in the "sent" direction and has a "pending" status
92     if data["message"]["direction"] == "sent" and data["message"]["status"] == "pending":
93         # Extract the message text
94         messagetext = data["message"]["content"]["text"]
95         # Check for specific conditions in the message text
96         if "Batch" in messagetext:
97             print("inside batch condition")
98             # Check if the message contains "Stock"
99             if "Stock" in messagetext:
100                 print("in process sent message")
101                 # Call the updatestockdb function for stock updates
102                 stock.updatestockdb(mongo_handler,data)
103             else:
104                 # Call the addbatchdb function for batch updates
105                 stock.addbatchdb(mongo_handler,data)
106             # Check if the message contains "Expense:"
107             elif "Expense:" in messagetext:
108                 # Call the addexpensedb function for expense updates
109                 expenses.addexpensedb(mongo_handler,data)
110             # Check if the message contains "Inventory"
111             elif "Inventory" in messagetext:
112                 # Call the addinventorydb function for inventory updates
113                 inventory.addinventorydb(mongo_handler,data)
114
115
```



```
25 def initiate(data):
31     # Retrieve the mapping document based on the messageId from the "actionmapping" collection
32     mapping_document = mongo_handler.find_one("actionmapping",{"messageId":data["message"]["replyTo"]["id"] })
33     print(str(mapping_document))
34     # Check if a mapping document exists and if it has an "action" field
35     if mapping_document and "action" in mapping_document:
36         # Check the type of action in the mapping document
37         if mapping_document["action"] == "stock":
38             action_id_str = ObjectId(mapping_document["actionId"])
39             # Call stock update function
40             stock.stock_update(mongo_handler, action_id_str, mapping_document, data)
41         elif mapping_document["action"] == "inventory":
42             # Convert the actionId from string to ObjectId
43             action_id_str = ObjectId(mapping_document["actionId"])
44             # Call the inventory update function
45             inventory.update(mongo_handler, action_id_str, mapping_document, data)
46         else:
47             # Call expense update function
48             action_id_str = ObjectId(mapping_document["actionId"])
49             expenses.update(mongo_handler, action_id_str, mapping_document, data)
50     # Check if the action is not defined in the mapping document
51     else:
52         # Extract message text and mobile number from the incoming data
53         messagetext = data["message"]["content"]["text"].lower()
54         mobile_number = data["contact"]["msisdn"]
55         # Determine the action based on the message text
56         if messagetext == ".st":
57             actions = "stock"
58
```

Figure 3.9 conversatio.py continuation code

3. utility/mongoDB.py

This module handles MongoDB interactions, such as connecting to the database and performing CRUD operations. It abstracts the database logic, making it easier to interact with MongoDB across different parts of the application.

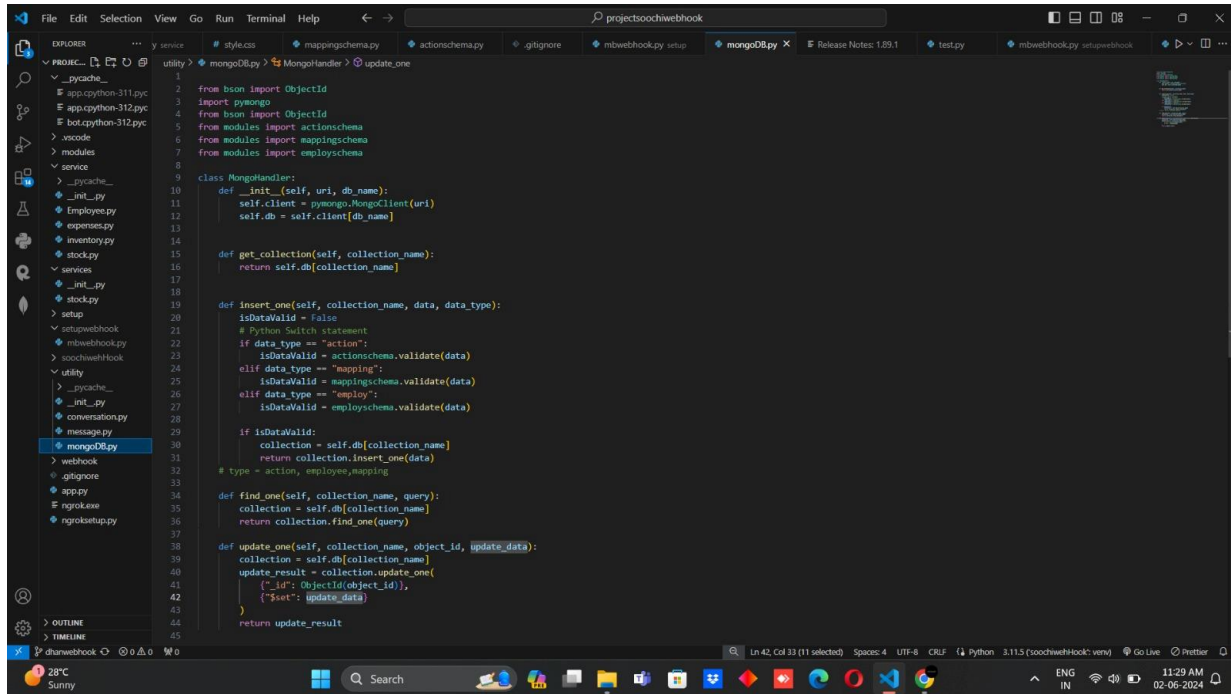


Figure 3.10 mongodb.py code

3.utility/message.py

This module manages message formatting and sending messages via the WhatsApp API. It includes functions to construct message payloads and send them using the appropriate API endpoint.

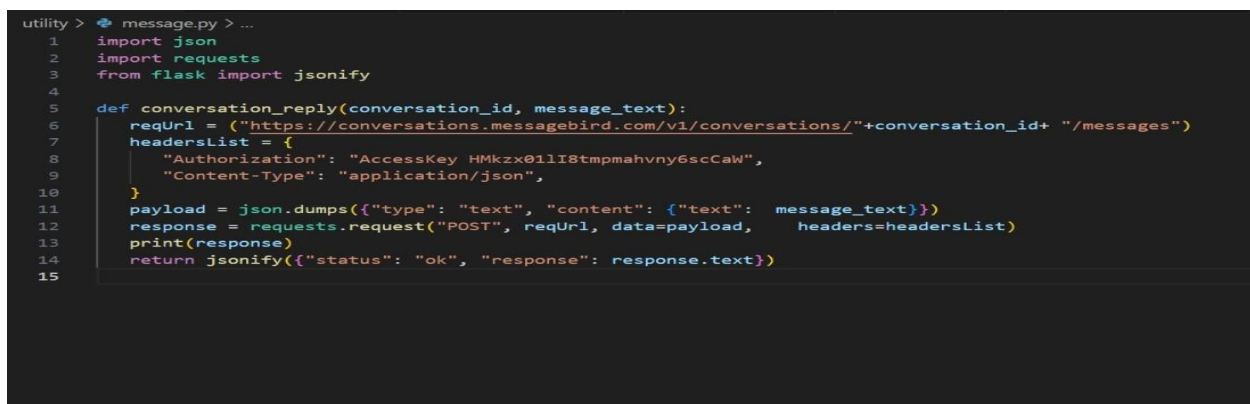


Figure 3.11 message.py code

Running the Application

1.Start Flask Application:

```
python app.py
```

2.Start Ngrok:

```
ngrok http 5000
```

3.Configure WhatsApp Webhook: Set the webhook URL in the WhatsApp Business API settings to the public URL provided by Ngrok (e.g., <https://abcd1234.ngrok.io/webhook>).

This architecture ensures a clean separation of concerns, making the system scalable and easier to manage. Each component has a specific role, contributing to the overall functionality of the WhatsApp bot.

3.2 Workflow Diagram

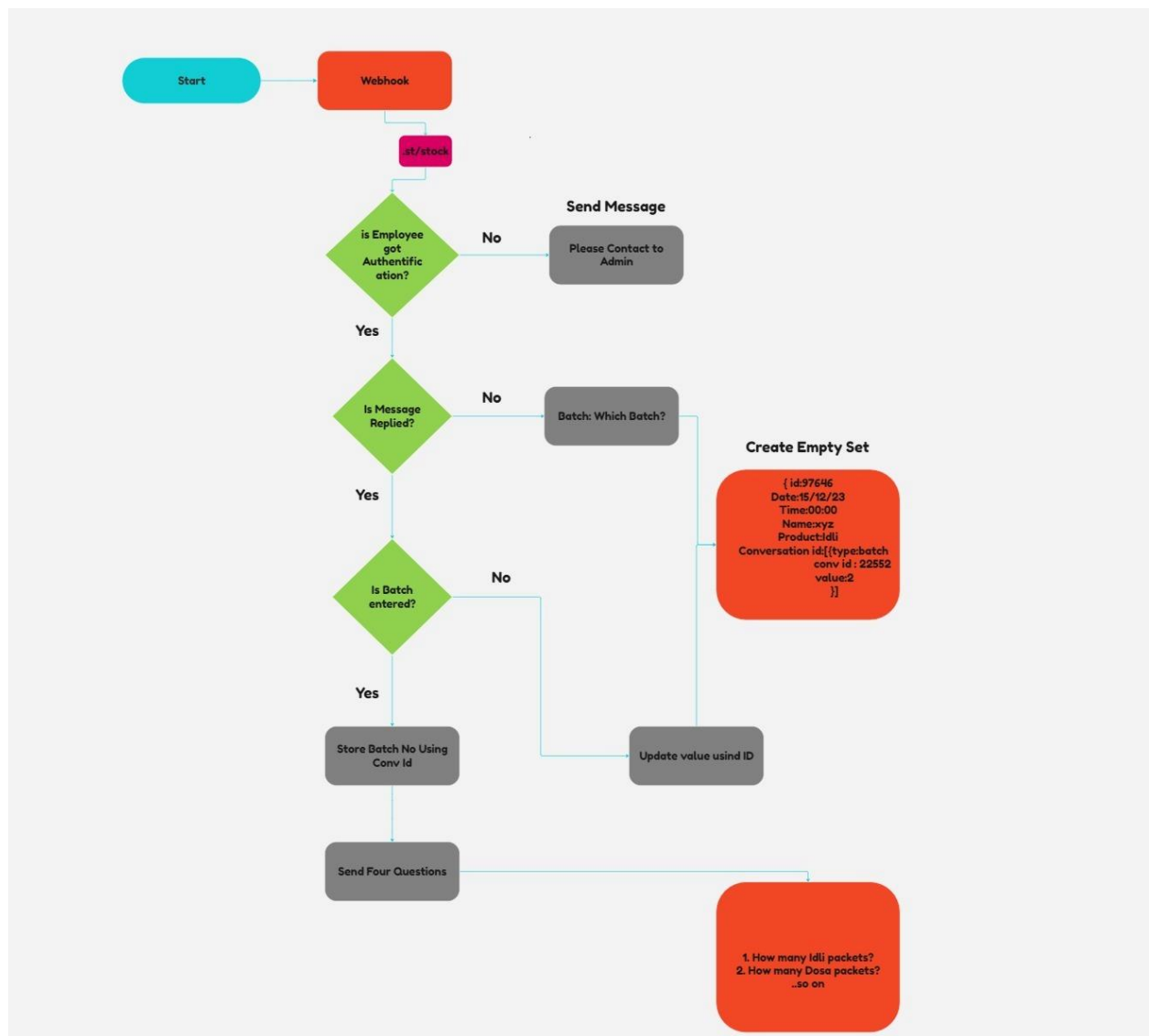


Figure 3.12 workflow diagram

Technical Explanation

1. User Interaction (WhatsApp User)

- The user sends a message to the WhatsApp bot through the WhatsApp application.
- The WhatsApp API receives this message and forwards it to the Flask application's webhook endpoint.

2. Receiving the Message (Flask App)

- The Flask app has a webhook endpoint configured to receive incoming messages from the WhatsApp API.
- The received message data is processed to determine the appropriate response.

3. Processing the Message (Flask App)

- The process message function in the Flask app handles the incoming message.
- Based on the message content, the function decides which service or utility module to use for generating the response.
 - For example, if the message contains ".st" it routes to the stock service module to fetch stock data.
 - If the message contain ".in" it routes to the inventory service module to fetch inventory data.
 - If the message contain ".ex" it routes to the expenses service module to fetch expenses data.

4. Service/Utility Modules

- **Service Modules:** These modules (`employee.py`, `expenses.py`, `inventory.py`, `stock.py`) interact with the MongoDB database to perform CRUD operations and retrieve or update data as needed.
- **Utility Modules:** These modules (`conversation.py`, `message.py`, `mongoDB.py`) provide helper functions for conversation management, message formatting, and database interactions.

5. Generating the Response

- The appropriate service or utility module processes the request and generates a response.
- For example, if the user asked for employee details, the `employee.get_employe ()` function retrieves the data from MongoDB, and the response is formatted accordingly.
- If the user asked about stock detail's, the `stock_document.get ()` function retrieves the data from MongoDB, and the response is formatted accordingly.
- If the user asked about stock detail's, the `inventory_document.get ()` function retrieves the data from MongoDB, and the response is formatted accordingly.

6. Sending the Response (WhatsApp API)

- The generated response is sent back to the user via the WhatsApp API.
- The `message.send_message ()` function handles the actual sending of the message, ensuring the user receives a formatted response.
- For example, it checks what is response if the send message from user is ".st" it check the authentication and if the user if authenticated then four questions is being asked from the WhatsApp to the user.

Working Explanation

1. **Start:**
 - The process begins when the user interacts with the WhatsApp bot.
2. **Webhook:**
 - The WhatsApp bot receives messages via a webhook.
3. **/st/stock Endpoint:**
 - The webhook forwards messages related to stock inquiries to the `/st/stock` endpoint.
4. **Is Employee Authenticated?**
 - The system checks if the user (employee) is authenticated.
 - **Yes:** If authenticated, proceed to the next step.
 - **No:** If not authenticated, send a message to contact the admin.
5. **Is Message Replied?**
 - The system checks if the user has replied to the previous message.
 - **Yes:** If replied, proceed to the next step.
 - **No:** Send a message asking for the batch number with "Batch: Which Batch?".
6. **Is Batch Entered?**
 - The system checks if the batch number has been entered by the user.
 - **Yes:** If entered, store the batch number using the conversation ID and proceed to the next step.
 - **No:** Create an empty set with the initial details and update the values using the ID provided in the user's message.
7. **Create Empty Set:**
 - An empty data structure is created with initial information such as:
 - ID
 - Date
 - Time
 - Name
 - Product details
 - Conversation ID
 - Initial values (which can be updated later)
8. **Store Batch Number Using Conversation ID:**
 - Once the batch number is provided, it is stored in the system using the conversation ID for future reference.
9. **Send Four Questions:**
 - After storing the batch number, the system sends a set of predefined questions to the user. These questions are related to stock details, such as:
 1. How many idli packets?
 2. How many dosa packets?
 3. Additional questions related to inventory or stock management.

Detailed Steps in the Flowchart:

- **Webhook and Endpoint Handling:**
 - The process starts with the webhook receiving a message from the WhatsApp API. It identifies the message type and forwards stock-related inquiries to the appropriate endpoint (`/st/stock`).
- **Authentication Check:**
 - The bot checks if the user is authenticated. If the user is not authenticated, the bot sends a message prompting the user to contact the admin for authentication.
- **Message Reply and Batch Number Check:**
 - If the user is authenticated, the bot checks if the user has replied to the initial message.
 - If no reply is detected, the bot sends a prompt asking the user to specify the batch number.
 - If the batch number is not provided, the bot creates an empty set with initial placeholders and waits for the user to update the values.
- **Creating and Updating Data:**
 - When the user provides the batch number, it is stored using the conversation ID for reference.
 - An empty set is created initially if no batch number is provided, and it is updated later with the provided details.
- **Sending Follow-up Questions:**
 - After the batch number is stored, the bot sends a series of follow-up questions to gather specific details about the stock (e.g., number of idli and dosa packets).

This workflow ensures that the bot effectively handles stock inquiries by verifying user authentication, prompting for necessary information, and storing and updating data systematically.

3.3 Database Schema

For the project "A Python-Powered WhatsApp Bot with MongoDB for Data Handling," we will define the schema for different modules using Python classes and MongoDB schema definitions. These modules (`actionschema.py`, `employschema.py`, `mappingschema.py`) will help organize and manage the data structure for different entities involved in the project.

1. actionschema.py

This module will define the schema for actions taken by the bot or users. Actions might include sending messages, updating records, or other operations.

```
modules > actionschema.py > validate
1  from jsonschema import Draft7Validator
2
3  # Define the JSON schema
4  schema = {
5      "type": "object",
6      "properties": {
7
8          # write the date and according to given whatsapp data
9          "date": {"type": "string", "pattern": "^\\d{4}-\\d{2}-\\d{2}$"}, # Date format: YYYY-MM-DD
10         "time": {"type": "string", "pattern": "^\\d{2}:\\d{2}:\\d{2}$"}, # Time format: HH:MM:SS
11         "phoneNumber": {"type": "integer"},
12         "conversations": {
13             "type": "array",
14             "items": {
15                 "type": "object",
16                 "properties": {
17                     "action": {"type": "string"},
18                     "product": {"type": "string"},
19                     "id": {"type": "string"},
20                     "value": {"type": "string"}
21                 },
22                 "required": ["action", "product", "id", "value"]
23             },
24             "minItems": 1,
25         },
26     },
27     "required": ["date", "time", "phoneNumber", "conversations"]
28 }
29 # Validate the data against the schema
30 validator = Draft7Validator(schema)
31
32 def validate(data):
33     validator = Draft7Validator(schema)
34     for error in validator.iter_errors(data):
35         print("Validation error:", error.message)
36     return False
37
38 return True
```

Figure 3.13 actionschema.py code

2. employschema.py

This module will define the schema for employee data. It includes fields for employee identification, authentication status, and other relevant details.

```
1  from jsonschema import Draft7Validator
2
3  schema = {
4      "type": "object",
5      "properties": {
6          "phoneNumber": {"type": "integer"},
7          "name": {"type": "string"},
8          "role": {"type": "string"},
9          "actions": {
10             "type": "array",
11             "properties": {
12                 "stock": {"type": "string"},
13                 "inventory": {"type": "string"},
14                 "expenses": {"type": "string"},
15             },
16             "required": ["stock", "inventory", "expenses"]
17         },
18     },
19     "required": ["phoneNumber", "name", "role", "actions"]
20 }
21 # Validate the data against the schema
22 validator = Draft7Validator(schema)
23
24 def validate(data):
25     validator = Draft7Validator(schema)
26     for error in validator.iter_errors(data):
27         print("Validation error:", error.message)
28         return False
29     return True
```

Figure 3.14 employschema.py code

3. mappingschema.py

This module will define the schema for mapping various entities such as users to their conversations or actions. It helps in organizing the relationships between different data entities.

```
modules > mappingschema.py > validate
1  from jsonschema import Draft7Validator
2
3  schema = {
4      "type": "object",
5      "properties": {
6          "actionId": {"type": "string"},
7          "messageId": {"type": "string"},
8          "action": {"type": "string"},
9      },
10     "required": ["actionId", "messageId", "action",]
11 }
12 # Validate the data against the schema
13 validator = Draft7Validator(schema)
14
15 def validate(data):
16     validator = Draft7Validator(schema)
17     for error in validator.iter_errors(data):
18         print("Validation error:", error.message)
19     return False
20     return True
21
22
23
```

Figure 3.15 mappingschema.py code

The schema definitions provided help organize the data structures for actions, employees, and mappings. Integrating these schemas into MongoDB and Flask allows for systematic data management and seamless communication between the bot and the users. Each module serves a specific purpose, ensuring clarity and maintainability in the codebase.

Chapter 4: WhatsApp API Integration

4.1 Overview of WhatsApp Business API

The WhatsApp Business API allows businesses to interact with customers in a reliable and secure manner. It provides endpoints to send and receive messages, manage contacts, and track message statuses. Using Message Bird's API simplifies the integration process, providing robust features for communication through WhatsApp.

4.2 Setting Up WhatsApp Sandbox with MessageBird

- **Sign Up and Get API Keys:**

- Sign up for a MessageBird account.
- Navigate to the WhatsApp channel setup and follow the instructions to link your WhatsApp number with the sandbox.

- **Configure Webhook URL:**

- Set the webhook URL in the MessageBird dashboard to point to your server where the bot is hosted.
- For local development, use a tunneling service like Ngrok to expose your local server to the internet.

- **Test Environment Setup:**

- Use the sandbox number provided by MessageBird to send and receive messages for testing purposes.

4.3 Authenticating and Connecting to WhatsApp API

Install MessageBird Python Library

First, install the MessageBird Python SDK.

```
pip install messagebird
```

Authenticate with MessageBird

Here's how you can set up authentication with MessageBird in your Python application.

```
# app.py

import messagebird
import os
from dotenv import load_dotenv

# Load environment variables
load_dotenv()

# MessageBird credentials
access_key = os.getenv('MESSAGEBIRD_ACCESS_KEY')

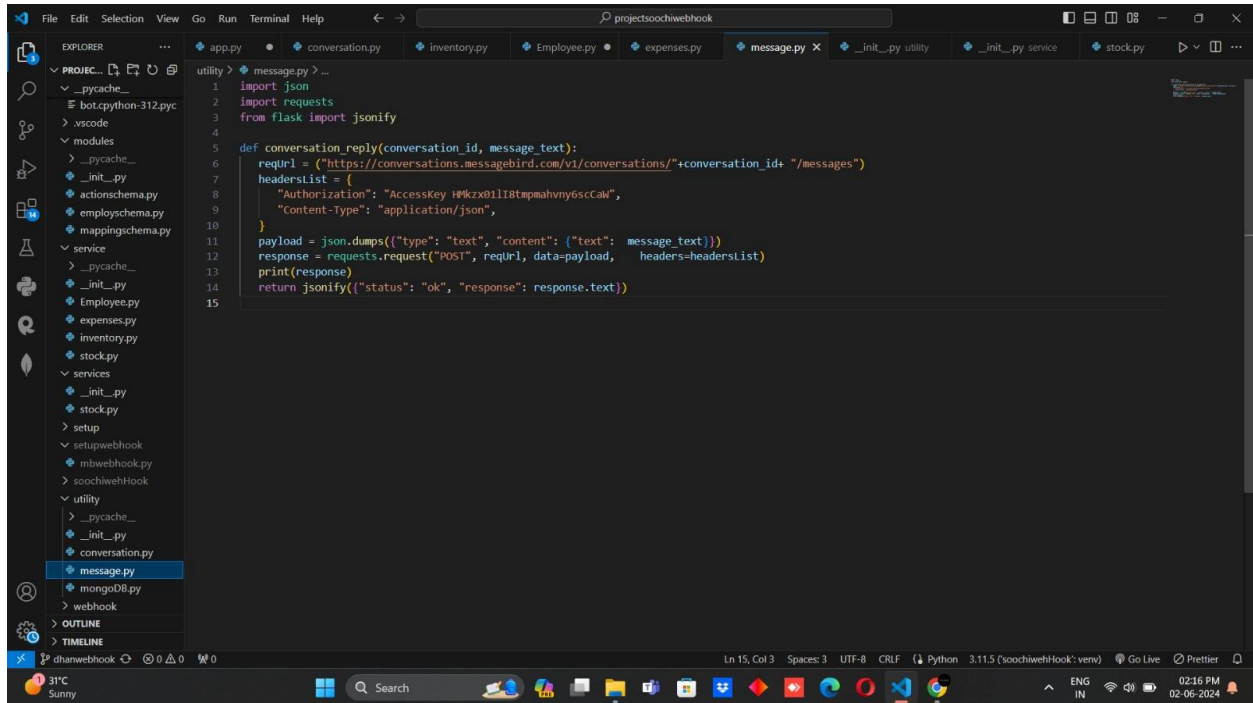
# Initialize MessageBird client
client = messagebird.Client(access_key)
```

Figure 4.1 Authenticate with MessageBird

4.4 Sending and Receiving Messages

Sending Messages

To send messages using MessageBird, define a function to interact with the MessageBird API.



```
File Edit Selection View Go Run Terminal Help
projectsoochiwebhook

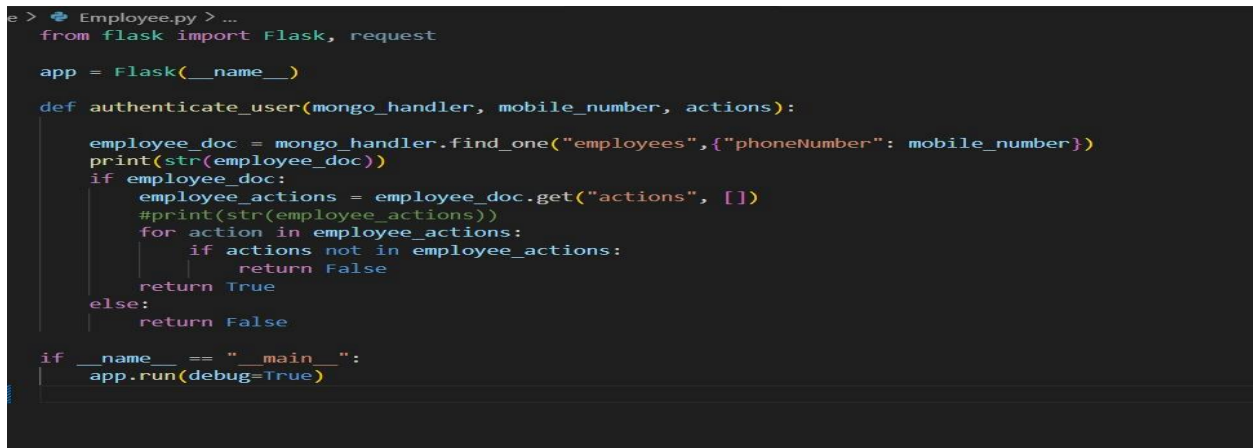
EXPLORER
PROJECT...
  _pycache_
  botcpython-312.pyc
  vscode
  modules
    _pycache_
    _init_.py
  actionschema.py
  employschema.py
  mappingschema.py
  service
    _pycache_
    _init_.py
  Employee.py
  expenses.py
  inventory.py
  stock.py
  services
    _init_.py
    stock.py
  setup
  setupwebhook
  mbwebhook.py
  soochiwebhook
  utility
    _pycache_
    _init_.py
    conversation.py
    message.py
  mongoDB.py
  webhook
  OUTLINE
  TIMELINE

utility > message.py > ...
1 import json
2 import requests
3 from flask import jsonify
4
5 def conversation_reply(conversation_id, message_text):
6     requrl = ("https://conversations.messagebird.com/v1/conversations/"+conversation_id+"/messages")
7     headersList = {
8         "Authorization": "AccessKey H4Kczx0118tampmahvny6scCak",
9         "Content-Type": "application/json",
10     }
11     payload = json.dumps({"type": "text", "content": {"text": message_text}})
12     response = requests.request("POST", requrl, data=payload, headers=headersList)
13     print(response)
14     return jsonify({"status": "ok", "response": response.text})
15
```

Figure 4.2 Sending Messages

Receiving Messages

To handle incoming messages, set up a Flask route that MessageBird will call when a new message is received.



```
e > Employee.py > ...
from flask import Flask, request

app = Flask(__name__)

def authenticate_user(mongo_handler, mobile_number, actions):
    employee_doc = mongo_handler.find_one("employees", {"phoneNumber": mobile_number})
    print(str(employee_doc))
    if employee_doc:
        employee_actions = employee_doc.get("actions", [])
        #print(str(employee_actions))
        for action in employee_actions:
            if actions not in employee_actions:
                return False
            return True
    else:
        return False

if __name__ == "__main__":
    app.run(debug=True)
```

Figure 4.3 Receiving Messages

Chapter 5: WhatsApp API Integration

5.1 Project Structure

Here's an outline of the project structure for the WhatsApp bot:

WhatsApp Bot/

- |— **app.py**
- |— **utility/**
 - | |— **__init__.py**
 - | |— **conversation.py**
 - | |— **message.py**
 - | |— **mongoDB.py**
- |— **service/**
 - | |— **__init__.py**
 - | |— **employee.py**
 - | |— **expenses.py**
 - | |— **inventory.py**
 - | |— **stock.py**
- |— **modules/**
 - | |— **__init__.py**
 - | |— **actionschema.py**
 - | |— **employschema.py**
 - | |— **mappingschema.py**

5.2 Initializing the Bot

app.py

This is the main application file that sets up the Flask server and handles incoming requests.

```
app.py > ...
1  import json
2  from flask import Flask, jsonify, request
3  import sys
4
5  sys.path.append('utility')
6  sys.path.append('service')
7  from utility import conversation
8  import pymongo
9
10 app = Flask(__name__)
11 @app.route("/")
12 def hello():
13     return "Bot is alive from flask!"
14
15 @app.route('/webhook', methods=['POST'])
16 def bot():
17
18     data = request.json
19     conversation.intiate(data)
20     return {}
21
22 if __name__ == '__main__':
23     app.run()
```

Figure 5.1 app.py code

5.3 Handling Incoming Messages, Message Parsing and Processing, Responding to User Queries

Handling Incoming Messages

The `webhook` route in `app.py` is responsible for receiving incoming messages. It calls `process_message` to handle the logic.

Message Parsing and Processing

The `process_message` function in `app.py` processes the incoming message to determine the appropriate response. It checks the message content and updates the conversation context accordingly.

Responding to User Queries

The `send_message` function in `utility/message.py` is used to send messages back to the user.

Combining all these pieces, we create a complete WhatsApp bot that handles authentication, processes messages, updates conversation contexts, and responds to user queries.

Chapter 6. MongoDB Integration

6.1 Introduction to MongoDB

MongoDB is a NoSQL database known for its flexibility and scalability. It stores data in a JSON-like format, making it easy to work with within a dynamic and fast-changing environment. MongoDB is ideal for handling large amounts of unstructured data, which is typical in messaging and chatbot applications.

6.2 Connecting Python to MongoDB

To connect Python to MongoDB, we use the `pymongo` library. This library allows Python applications to interact with MongoDB in a seamless manner.

Create a utility module to handle MongoDB connections and operations. This will typically go in a file like `utility/mongoDB.py`.

```
python
Copy code
# utility/mongoDB.py

from pymongo import MongoClient
import os
from dotenv import load_dotenv

# Load environment variables
load_dotenv()

# MongoDB setup
client = MongoClient(os.getenv('MONGODB_URI'))
db = client.get_database(os.getenv('MONGODB_DB_NAME'))

def get_user(phone_number):
    return db.users.find_one({"phone_number": phone_number})

def add_user(user_data):
    return db.users.insert_one(user_data)

def update_user(user_id, update_data):
    return db.users.update_one({"_id": user_id}, {"$set": update_data})
```

```
def get_conversation(conversation_id):
    return db.conversations.find_one({"conversation_id": conversation_id})

def add_conversation(conversation_data):
    return db.conversations.insert_one(conversation_data)

def update_conversation(conversation_id, update_data):
    return db.conversations.update_one({"conversation_id": conversation_id},
    {"$set": update_data})
```

This module sets up the connection to MongoDB and provides basic CRUD operations for users and conversations.

6.3 CRUD Operations in MongoDB

CRUD stands for Create, Read, Update, and Delete. These are the basic operations we perform on a database.

Create

Inserting a new document into a MongoDB collection:

```
python
Copy code
def add_user(user_data):
    return db.users.insert_one(user_data)
```

Read

Retrieving documents from a MongoDB collection:

```
python
Copy code
def get_user(phone_number):
    return db.users.find_one({"phone_number": phone_number})
```

Update

Updating an existing document in a MongoDB collection:

```
python
Copy code
def update_user(user_id, update_data):
    return db.users.update_one({"_id": user_id}, {"$set": update_data})
```

Delete

Removing a document from a MongoDB collection.

6.4 Storing and Retrieving Bot Data

To store and retrieve bot data, we define functions that interact with the MongoDB collections. Here are the functions we have:

Users Collection

- `get_user(phone_number)`: Retrieves user information based on the phone number.
- `add_user(user_data)`: Adds a new user to the collection.
- `update_user(user_id, update_data)`: Updates user information.

Conversations Collection

- `get_conversation(conversation_id)`: Retrieves conversation data based on the conversation ID.
- `add_conversation(conversation_data)`: Adds a new conversation to the collection.
- `update_conversation(conversation_id, update_data)`: Updates conversation data.

This guide shows how to integrate MongoDB into a Python-based WhatsApp bot using the MessageBird API. By following these steps, you can create a scalable and maintainable system for handling user interactions and storing conversation data. The modular structure ensures that each component is easy to manage and extend as the project grows.

Chapter 7: Results

Project Outcome

The primary objective of this project was to develop a Python-powered WhatsApp chatbot integrated with MongoDB for data handling. The following outcomes were achieved:

1. **Successful Integration of WhatsApp API:** Using MessageBird's API, the chatbot successfully communicated with users via WhatsApp.
2. **Robust Backend with Flask:** A Flask-based backend was created to handle incoming messages, process them, and respond appropriately.
3. **Efficient Data Management with MongoDB:** MongoDB was used to store and retrieve user data, conversation logs, and bot interactions efficiently.
4. **Modular Code Structure:** The project was structured in a modular way, making it easy to maintain and extend. This included separate modules for services, utilities, and database schemas.
5. **Testing and Debugging:** Comprehensive unit and integration tests were written to ensure the reliability of the bot. Logging and debugging techniques were employed to identify and fix issues.

Key Features

- **Authentication and Authorization:** Users needed to authenticate via their phone numbers. Unauthorized users were prompted to contact the admin.
- **Dynamic Question Handling:** The bot could ask a series of questions and store user responses dynamically.
- **CRUD Operations:** Efficient handling of Create, Read, Update, and Delete operations on user and conversation data in MongoDB.
- **Scalability and Performance:** The system was designed to be scalable, with considerations for database indexing, connection pooling, and caching to optimize performance.

Testing and Debugging Results

- **Unit Tests:** The unit tests covered individual functions in the mongoDB.py utility module and main application logic in app.py. All unit tests passed, indicating that the individual components functioned correctly.
- **Integration Tests:** The integration tests verified the interaction between different components. These tests ensured that the end-to-end flow, from receiving a message to storing user data and responding, worked seamlessly.
- **Debugging:** Debugging techniques such as logging, breakpoints, and exception handling were used to identify and fix issues. These methods helped trace and resolve bugs, ensuring the bot's smooth operation.

For Stock

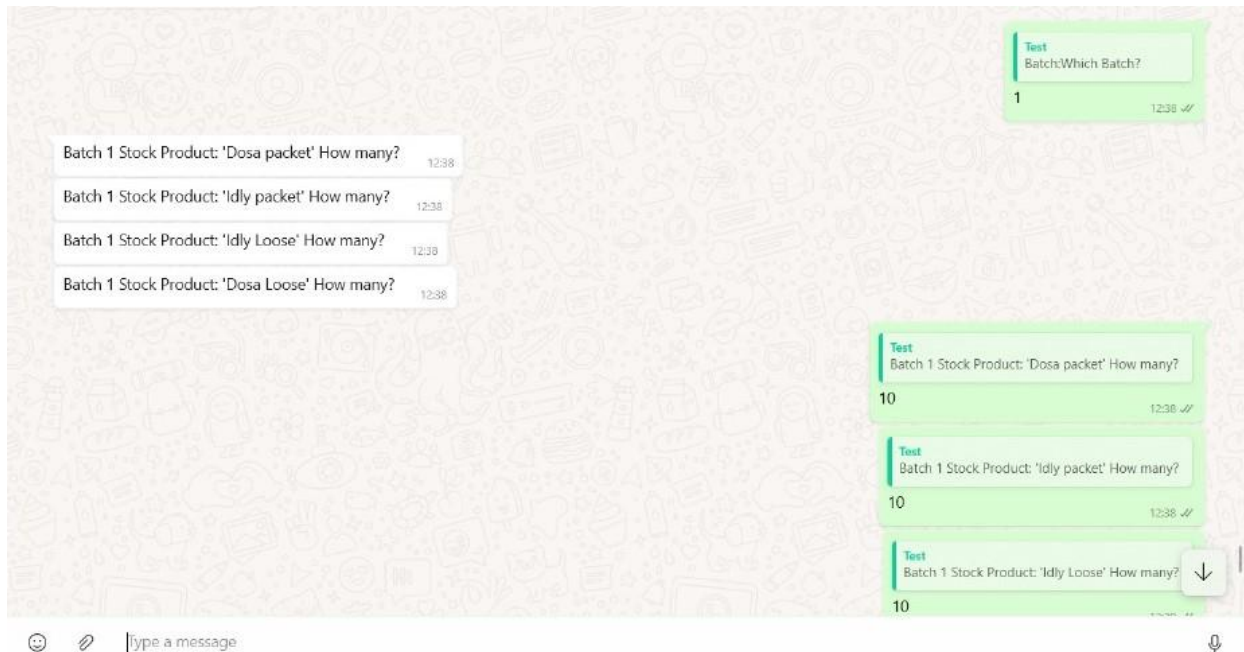


Figure 7.1 WhatsApp interface for stock overflow working

Backend Storage of data in MongoDB

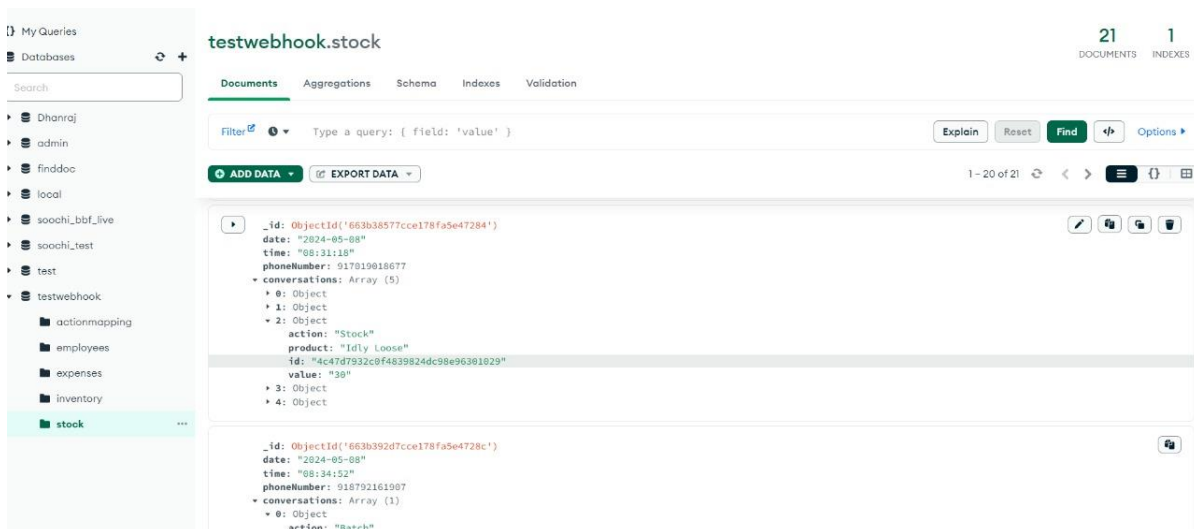


Figure 7.2 Backend storage of data related to stock

For Inventory

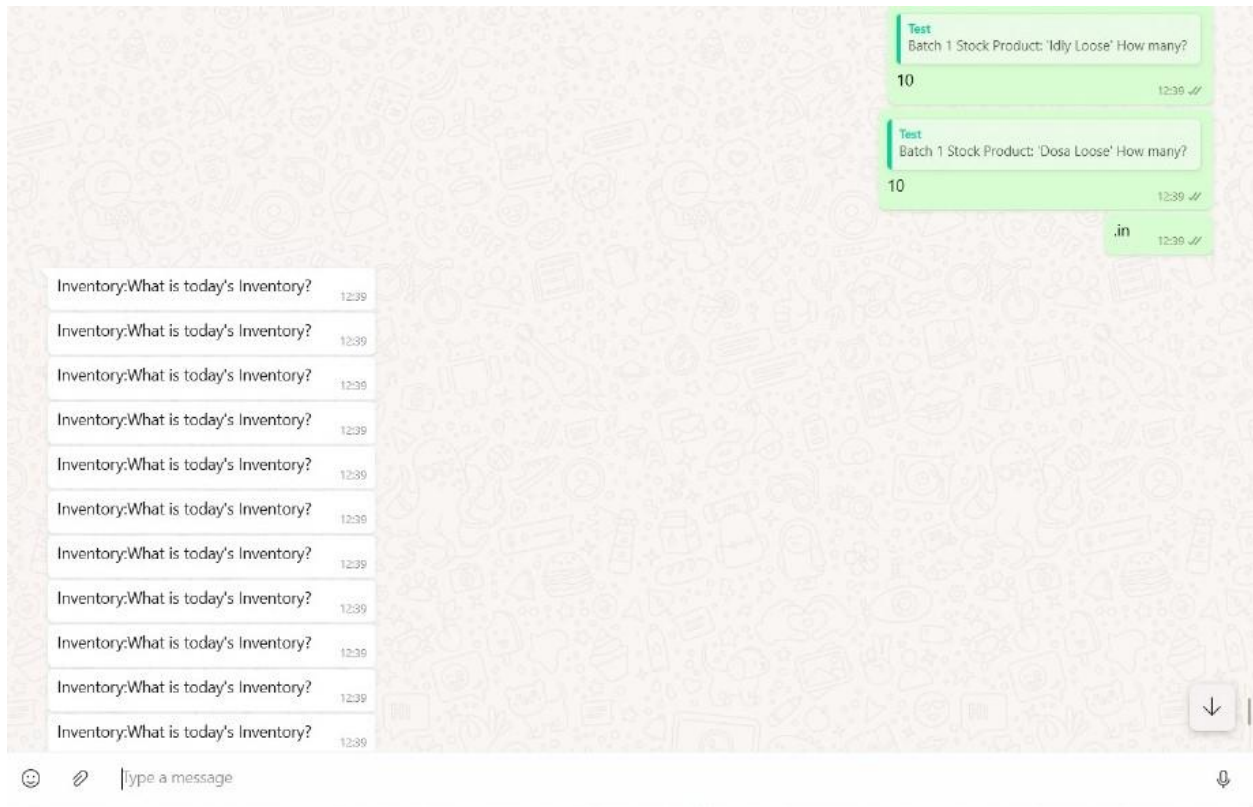


Figure 7.3 WhatsApp interface for inventory

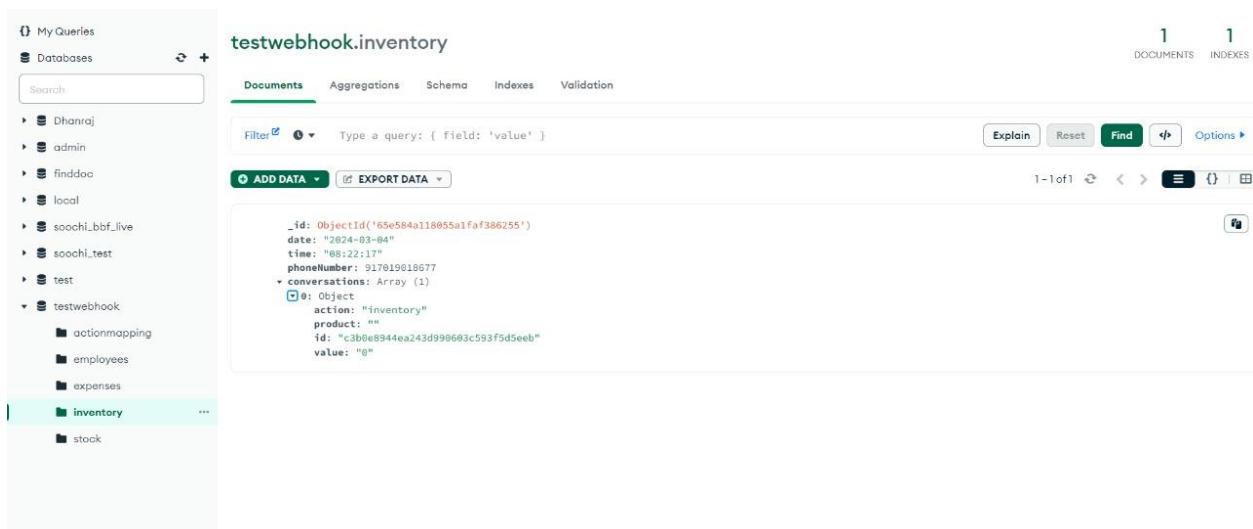


Figure 7.4 Backend storage of data related to inventory

For Expenses

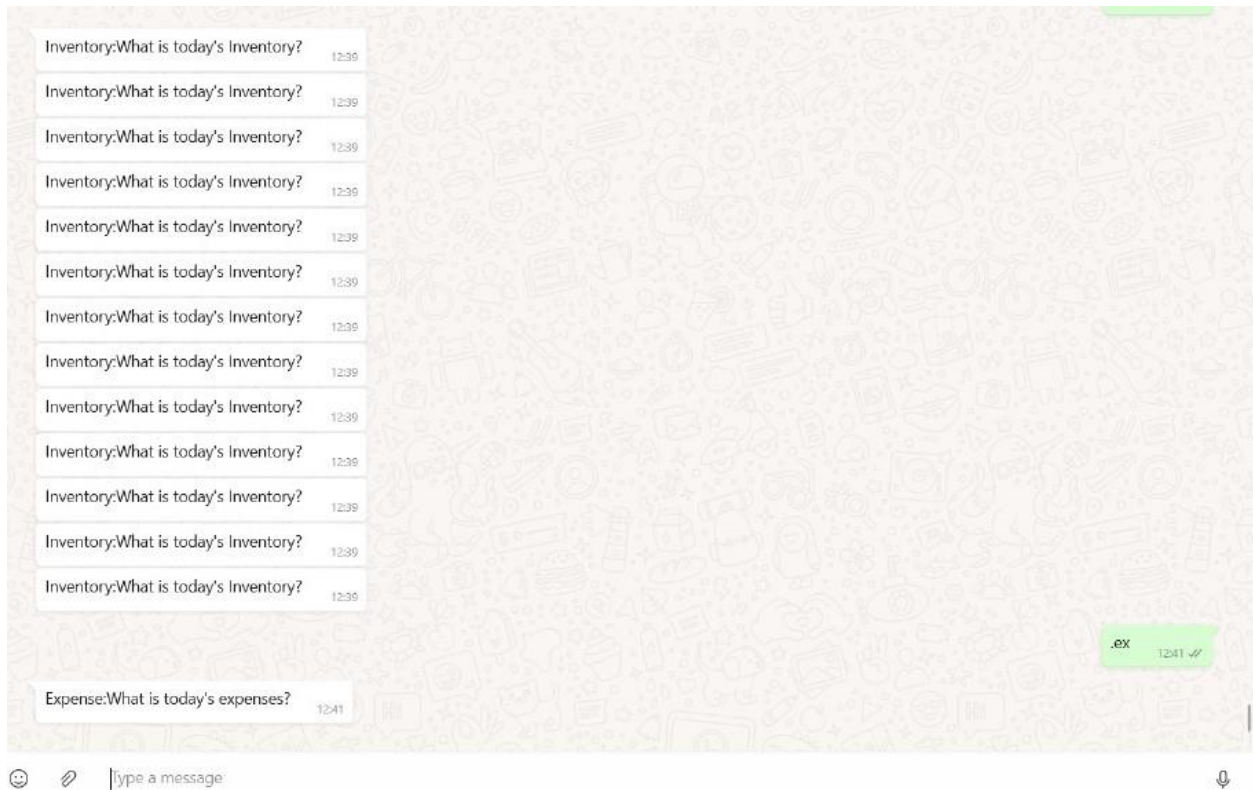


Figure 7.5 WhatsApp interfacing for expenses

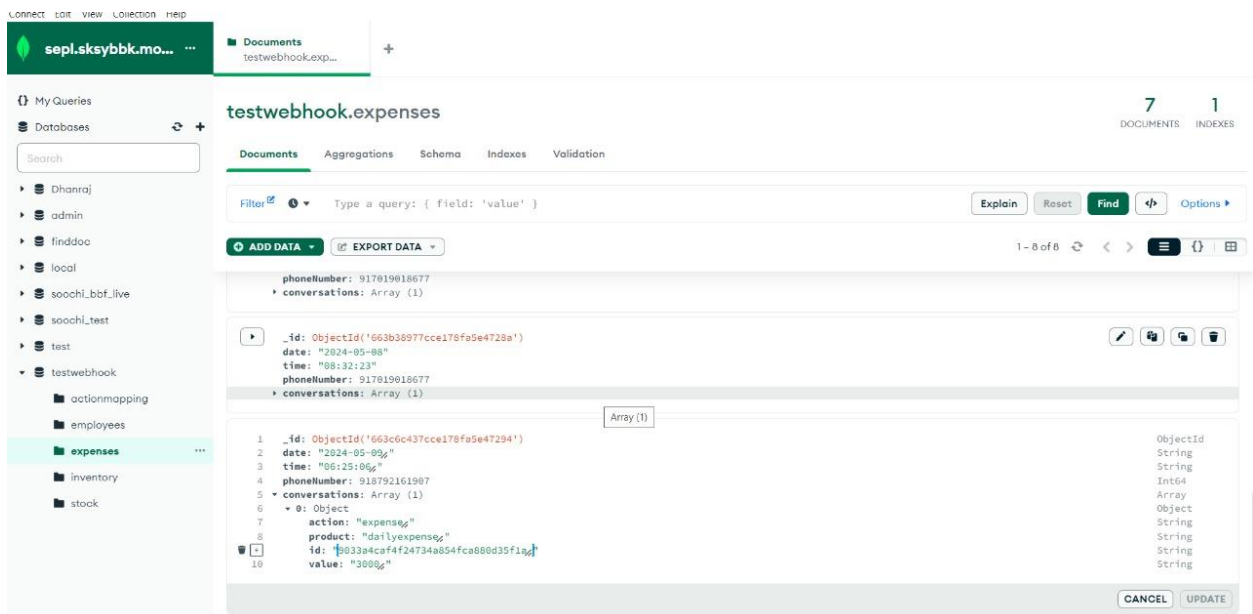


Figure 7.6 Backend storage of data related to expenses

Chapter 8: Conclusion

The project "A Python-Powered WhatsApp Bot with MongoDB for Data Handling" successfully demonstrates the creation of a robust and scalable chatbot application. By leveraging Python, MongoDB, Flask, and MessageBird's WhatsApp API, we were able to build a system that effectively handles user interactions and manages data efficiently.

Achievements

1. **Integration of Technologies:** The seamless integration of WhatsApp API with Python and MongoDB forms the backbone of the chatbot, enabling real-time communication and efficient data management.
2. **Modular Design:** The project was structured into distinct modules, including services, utilities, and schemas, ensuring maintainability and ease of extension.
3. **Authentication and Authorization:** Implementing user authentication ensures secure interactions, prompting unauthorized users to seek admin assistance.
4. **Dynamic Question Handling:** The bot dynamically handles user queries, storing responses contextually, which showcases the flexibility of the system.
5. **Testing and Debugging:** Comprehensive unit and integration testing, along with effective debugging techniques, contributed to the stability and reliability of the bot.
6. **Performance Optimization:** Database indexing, connection pooling, and caching considerations were made to optimize the performance, ensuring the bot's responsiveness under various conditions.

Key Learnings

- **Effective Use of MongoDB:** Using MongoDB for data storage allowed for efficient management of unstructured data, which is typical in chatbot applications.
- **Scalable Architecture:** Designing a system that can scale to handle increased load is crucial for real-world applications, and this project lays a strong foundation for that.
- **Testing Importance:** Writing thorough unit and integration tests ensures that each component of the system works as expected and helps catch issues early in the development cycle.
- **Real-time Interaction Handling:** Managing real-time interactions through the WhatsApp API provides a practical understanding of handling asynchronous communication.

Future Directions

- **Advanced Features:** Incorporating natural language processing (NLP) to better understand user intent and provide more sophisticated responses.
- **Enhanced Security:** Implementing stronger authentication and authorization mechanisms to ensure secure interactions.
- **Admin Dashboard:** Developing a user-friendly admin interface to monitor and manage bot interactions and user data more effectively.

- **Enhanced Caching:** Implementing robust caching mechanisms to further improve the performance of frequently accessed data.
- **Horizontal Scaling:** Exploring horizontal scaling options to accommodate a larger number of users and higher message volumes.

This project serves as a comprehensive guide for building a WhatsApp chatbot using modern technologies. The principles and techniques demonstrated here can be adapted and expanded to fit a variety of use cases and requirements. By focusing on modular design, efficient data handling, and rigorous testing, we have created a robust foundation for further development and enhancement of chatbot applications.

