Python Test V1 - Answer

ADRIÁN ALBERTO GUTIÉRREZ LEAL

PLAYVOX
Junior Software Engineer
MEDELLÍN
2022

1. **Please, explain in your own words what the fragment of code does. It is ok to describe the functionality in broad terms.**

From my perspective, the fragment of code serves a single purpose: to send a message to a user regarding their account, taking into special consideration what I understand as a ‘subscription class’ or ‘membership’ (i.e. Platinum, Gold…), with a service link for more information.

It seems to represent a small fraction of a larger system, that manages different membership levels, probably with different treatment within them and provides some service depending on their class.

More accurately, the code seems to read a text file with the user subscription class and profile, then makes queries to a local database by using said profile as a parameter. Finally, it assembles a message depending on a subscription class and sends it through an AWS client, to tell users about some news in their account.

2. **What information do you think is missing to understand the purpose of this code fully?**

The first piece of information I believe is missing from this code is the context. There is no information whatsoever as to what it's supposed to do, nor a Readme file or even a file name, only plain code, from which the context can only (maybe accurately) be assumed.

In a more in-depth look of the code itself, it lacks comments to detail in natural words what the code is supposed to do. Moreover, variables and methods are not named after what they do, they go by names such as func01, which makes it a lot harder to read and understand the code.

3. **Now, take the time to reread the code, and please suggest all the refactoring you consider necessary. You can aim to improve the code in several ways. For example, is there a way to make this code more readable or improve performance or security? Rewrite the code to reflect those changes you suggest and add comments to explain each change's rationale.**

The main correction done to the code was to its readability, mainly function and variable names. Some redundancy was also corrected, as well as minor issues and improvements in performance.

As for security, it's best not to add credentials inside the code. These were, instead, added as environment variables. For the purpose of this test the variables can be set within an attached script.

The rewritten code goes as follows:

```
import os #security
import mysql.connector
import boto3

'''
Environment variables created for security, so keys, passwords and other info
are not visible from repo. As this is an academic exercise, all variables will
be set in a separate file so they can be seen.
'''
user=os.environ('PLAYVOX_USER')
password=os.environ('PLAYVOX_PASSWORD')
aws_key_id=os.environ('PLAYVOX_AWS_ACCESS_KEY_ID')
aws_secret=os.environ('PLAYVOX_AWS_SECRET_ACCESS_KEY')

client = boto3.client(
    "sns",
    aws_access_key_id=aws_key_id,
    aws_secret_access_key=aws_secret,
    region_name="us-east-1"
)

cnx = mysql.connector.connect(user=user, password=password,
                              host='127.0.0.1',
                              database='users')
```

```
def send_message(message, number):
    client.publish(
        phone_number=number, #variable name changed to snake for consistency
        message=message  #variable name changed to snake for consistency
    )

#functions func03 to func06 were replaced by 'build_message' to make it easier to
understand what it does and save lines of code
'''
The previous implementation was highly redundant. It used 4 functions with the
exact same code, differing only
in the user subscription level (Gold, Silver...), which was hardcoded as a
String. To solve this, a new parameter
'user_subscription' was added in order to be sent within the message.
'''
def build_message(user_subscription,user_profile):
    query = "select * from users where profile ="+user_profile
    cursor = cnx.cursor()
    cursor.execute(query)
    db_records = cursor.fetchall() #To make iterations easier than they were

    for row in db_records: #only iterating through rows in the database, instead
of columns
        #iterate through phone numbers in record
        for number in row['phone_number']:
            '''
            Since every individual phone number is already accounted for in the
loop, it was
            considered redundant to make a new query and store it in a dictionary
with only
            one key, since the only thing sent to the send_message function is a
single value
            '''
            send_message("Hey "+row['first_name']+" "+row['last_name'] +" we have
some information about your "+user_subscription+" account, please go to "+
row['service_link']
                            + " to get more details", number)



#function func02 was renamed 'check_subscription' to make it easier to understand
what it does
'''
The if statement was replaced by adding the user subscription as a parameter to
be used
```

```
by a single function, in contrast with the previous 4 that did the exact same
thing with
only the subscription (formerly hard-coded) as a difference.
'''
def check_subscription(user_info): #parameter 'x' was renamed 'user_info' to make
it easier to understand what it does
    #variable 'y' was renamed 'user_subscription' to make it easier to understand
what it does
    user_subscription = user_info[0].split(' ')[1]
    #variable 'user_profile' was created to favour readability
    user_profile = user_info[1]
    #func03 - 06 were replaced by 'build_message()'
    build_message(user_subscription,user_profile)

#function func01 was renamed 'read_file' to make it easier to understand what it
does
def read_file():
    #variable 'f' was renamed 'file' to make it easier to understand what it does
    file = open("src/profileDB_id.txt", "r")#';' was removed due to it being
inconsistent with the rest of the code's writting style
    lines = file.readlines() #initial was changed to lowercase for consistency

    #There were both iteration and tail recursion being implemented. Only the
iteration was kept.
    '''
    Iteration was kept due to it being much faster and space efficient than
recursion
    For this particular case, it's simple enough to be easily understood through
iteratitions
    '''
    for line in lines:
        #variable 'x' was renamed to 'user_info' to make it easier to understand
what it does
        user_info = line.split(";") # ';' was removed due to it being
inconsistent with the rest of the code's writting style
        check_subscription(user_info)
```

NOTE 1: This particular file (messages.py) can also be found in the attachment files (src/messages.py) or in the GitHub repository created for this test.

NOTE 2: All other files (test, profileDB_id.txt, environment script…) can also be found in the attachment files or in the GitHub repository created for this test:
https://github.com/GrayDiamond493/PlayVoxPythonTest_AdrianGutierrez.git

4. **As you may suspect by now, this code is part of a more extensive system. With the information that you have, can you explain how the architecture of this system is? You can be as detailed as you want. You can use draws or diagrams if you find it necessary**

The architecture of this system would probably be a mix of a Layered and a Client-Server pattern, with some distributed system characteristics provided by AWS and maybe the HTTP protocol for simple client-server communication. There is not enough information to affirm whether this application implements or is oriented towards services.

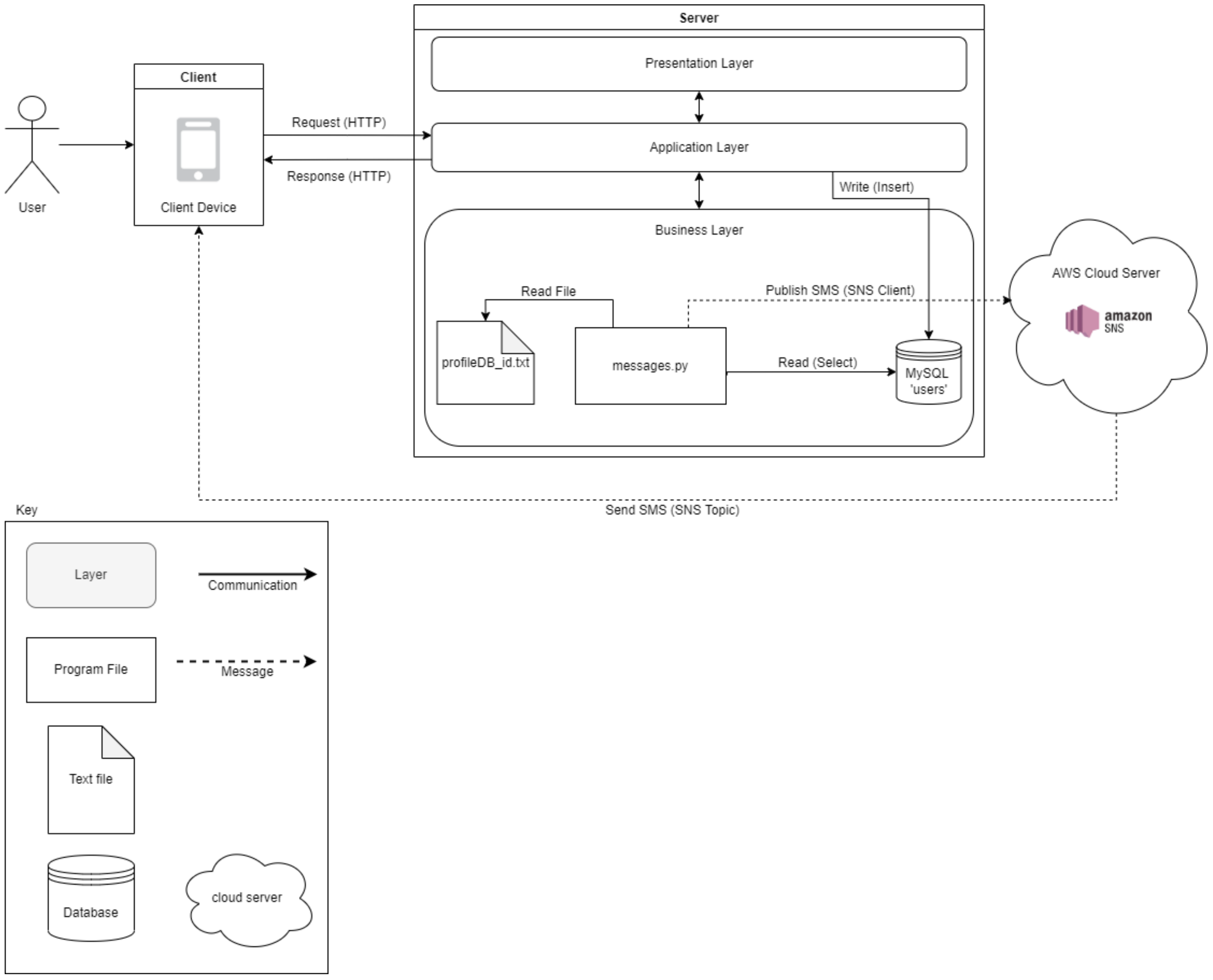
From what can be seen inside the code, the global system would have customers who register and create profiles with their personal information. This would be running in a client machine, which then communicates with a server that stores user profile information in a local database and associates each user with a membership class via a text file.

The code itself represents what seems to be a Business Layer, which queries the database and builds a message, based on user-provided profile information, to send it to the cloud via an AWS SNS client, which then provides many-to-many messaging to distribute SMSs to many subscribers.

Not much is known about the presentation and application layers, but their existence can be assumed due to the necessity of storing customer information (provided by the user), which should mean there is some sort of UI (Presentation Layer) for the user to interact with the application. From its nature (managing memberships and sending messages) it’s safe to assume there is direct interaction with the end user.

There doesn’t seem to be any implementations of Models since the database is accessed directly from the code. For the same reason, no particular framework, directory order or additional files were assumed.

My view of the architecture can be explained through the following diagram:



NOTE: A more detailed image (PDF format) can also be found in the attachment files or in the GitHub repository created for this test.

https://github.com/GrayDiamond493/PlayVoxPythonTest_AdrianGutierrez.git