ABSTRACT

People interact with systems more and more through voice assistants and chatbots. The days of solely engaging with a service through a keyboard are over. These new modes of user interaction are aided in part by This research will investigate how advancements in Artificial Intelligence and Machine Learning technology .are being used to improve many services. In particular it will look at the development of chatbots as a channel for information distribution. This project aimed to implement a web-based chatbot to assist with online banking, using tools that expose artificial intelligence methods such as natural language understanding. Allowing users to interact with the chatbot using natural language input and to train the chatbot using appropriate methods so it will be able to generate a response. The chatbot will allow users to view all their personal banking information all from within the chatbot. The chatbot was tested across a range of devices such as Google Home and Assistant on android devices to outline the key differences between the two modes of interaction , spoken and text dialog. These test were carried out to identify the value in integrating such technology surrounding the recent interest In chatbots and conversational interfaces. Proving chatbots can be applied to a specific domain to enhance accessibility, reaffirming that they are more than just a passing fad and have a viable use

INTRODUCTION

Chatbot is a computer program that humans will interact with in natural spoken language and including artificial intelligence techniques such as NLP (Natural language processing) that makes the chatbot more interactive and more reliable. Based on the recent epidemiological situation, the increasing demand and reliance on electronic education has become very difficult to access to the university due to the curfew imposed, and this has led to limited access to information for academics at the university. This project aims to build a chatbot for Admission and Registration to answer every person who asks about the university, colleges, majors and admission policy.

The term chatbot is synonymous with text conversation but is growing quickly through voice communication Alexa, what time is it? The chatbot can talk to you through different channels; such as Facebook, Messenger, siri, WeChat, Telegram,SMS,Slack,Skype and many others.

Examples of companies using chatbots

- Uber to book a taxi
- KLM to deliver right information
- Pizza Hut to help you order a pizza
- CNN to keep you up-to-date with news content
- Tech crunch to keep you up-to-date with techie content
- Sephora to provide beauty tips and a shopping experience
- Bank of America to connect customers and their nuance

You'll get the basic chatbot up and running right away in **step one**, but the most interesting part is the learning phase, when you get to train your chatbot. The quality and preparation of your training data will make a big difference in your chatbot's performance.

To simulate a real-world process that you might go through to create an industry-relevant chatbot, you'll learn how to customize the chatbot's responses. You'll do this by preparing **WhatsApp chat data** to train the chatbot. You can apply a similar process to train your bot from different conversational data in any domain-specific topic.

EXISTING SYSTEM OF CHATBOTS

Briefly and as mentioned in the definition, humans interact with chatbots. There are two ways to interact with a chatbot:

a. Text chatbot analyses the inputted text and matches the text with predefined data called intents which are categorized to manage the conversation. The user utterance is tagged with one of these intents, even if what the user says stretches over two or more intents. Most chatbots will take the intent with the highest score and take the conversation down that avenue. 8

b. Voice Some chatbots can interact and understand the voice of the user using a set of application programming interfaces (API's) that converts the recorded voice to the language and then convert the voice to words of that language and then deal with the transformed text as mentioned above.

PROBLEM STATEMENT

Even though we have many chatbots, we are lacking in chatbot system for whatsapp app which we use regularly. Providing chatbot option for whatsapp can improve working nature with it and can be able to understand any problems regarding the app.

PROPOSED SYSTEM OF CHATBOTS

The following documentation is a project based on chatbots for whatsapp chats data to train the chatbots. Simply we can put that we use a chatbots for whatsapp app. It is a detailed summary of all the drawbacks of the old system and how the new proposed system overcomes the these shortcomings.

LITERATURE SURVEY

Changing of requirements As an industrial project to build a product, we must follow the requirement from the user. However, because the project' s goal is to be used by the business team, but it is responsible by the technical team, the requirement changed a lot in the middle after a meeting with the business team. The business team want a simple bot that can give recommendation immediately. We had to archive what we had done before and build a new one.

.

REQUIREMENTS AND SPECIFICATIONS

SOFTWARE REQUIREMENTS

- Conditional statements

- while loops for iteration
- Lists and tuples
- Python functions
- Substring checks and substring replacement
- File input/output
- Python comprehensions and generator expressions
- Regular expressions (regex) using re

HARDWARE REQUIREMENTS:

Processor : Pentium IV(minimum)

Hard Disk : 40GB

RAM : 256MB (minimum)

SYSTEM ANALYSIS

OBJECTIVES

- Build a **command-line chatbot** with ChatterBot
- **Train** the chatbot to customize its responses
- **Export** your WhatsApp chat history
- Perform **data cleaning** on the chat export using **regular expressions**
- **Retrain** the chatbot with industry-specific data

SCOPE OF THE PROJECT

Our project will be able to train the chatbot by using whatsapp data. It helps whatsapp users by answering any queries regarding chat suggestions and other related doubts regarding chats in whatsapp. We can either fetch the conversation history of one of our WhatsApp chats or use the provided chat.txt fle. It can provide an interactive experience with relevant replies.

METHODOLOGY

You may have this question in your mind, how to create a chatbot? We' ll take a step by step approach and break down the process of building a Python chatbot.

To build a chatbot in Python, you have to import all the necessary packages and initialize the variables you want to use in your chatbot project. Also, remember that when working with text data, you need to perform data preprocessing on your dataset before designing an ML model.

This is where tokenizing helps with text data – it helps fragment the large text dataset into smaller, readable chunks (like words). Once that is done, you can also go for lemmatization that transforms a word into its lemma form. Then it creates a pickle file to store the python objects that are used for predicting the responses of the bot.

Another vital part of the chatbot development process is creating the training and testing datasets.

SYSTEM ARCHITECTURE

let' s dive deeper into the actual process! Before you get started, make sure that you have a Python version available that works for this ChatterBot project. What version of Python you need depends on your operating system?... We need to use a Python version below 3.8 to successfully work with the recommended version of ChatterBot. We' ll touch on a handful of Python concepts while working through this project on chatbot

Conditional statements

while loops for iteration

Lists and tuples

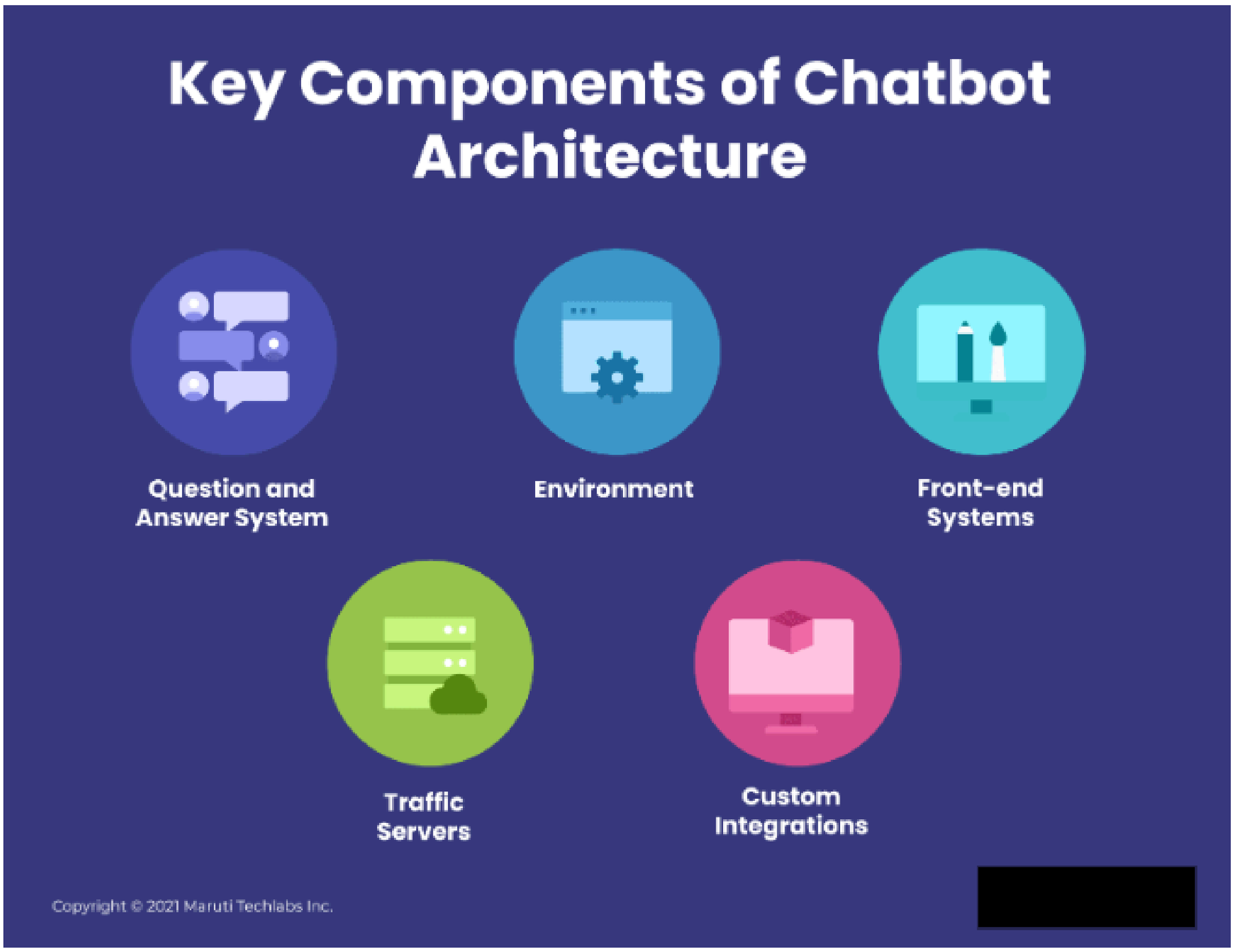
Python functions

Substring checks and substring replacement

File input/output

Python comprehensions and generator expressions

Regular expressions (regex) using re



How to create a chatbot using python ??

1. Prepare the dependencies

The first step in creating a chatbot in Python with the ChatterBot library is to install the library in your system. It is best if you create and use a new Python virtual environment for the installation.

```
pip install chatterbot
pip install chatterbot_corpus
```

2. Import Classes

Importing classes is the second step in the Python chatbot creation process. All you need to do is import two classes - ChatBot from chatterbot and ListTrainer from

chatterbottrainers.

```
pip install --upgrade chatterbot_corpus
pip install --upgrade chatterbot
```

3. Create and Train the Chatbot

This is the third step on creating chatbot in python. The chatbot you are creating will be an instance of the class "ChatBot." After creating a new ChatterBot instance, you can train the bot to improve its performance. Training ensures that the bot has enough knowledge to get started with specific responses to specific inputs.

```
my_bot = ChatBot(name='PyBot', read_only=True,
                 logic_adapters=
['chatterbot.logic.MathematicalEvaluation',
                                'chatterbot.logic.BestMatch'])
```

Here, the argument (that corresponds to the parameter name) represents the name of your Python chatbot. If you wish to disable the bot' s ability to learn after the training, you can include the "read_only=True" command. The command "logic_adapters" denotes the list of adapters used to train the chatbot. While the "chatterbot.logic.MathematicalEvaluation" helps the bot to solve math problems, the "chatterbot.logic.BestMatch" helps it to choose the best match from the list of responses already provided.

4. Communicate with the Python Chatbot

To interact with your Python chatbot, you can use the .get_response() function. This is how it should look while communicating:

```
>>> print(my_bot.get_response("hi"))
how do you do?

>>> print(my_bot.get_response("i feel awesome today"))
excellent, glad to hear that.

 >>> print(my_bot.get_response("what's your name?"))
i'm pybot. ask me a math question, please.

>>> print(my_bot.get_response("show me the pythagorean
theorem"))
a squared plus b squared equals c squared.

>>> print(my_bot.get_response("do you know the law of
cosines?"))
c**2 = a**2 + b**2 - 2 * a * b * cos(gamma)
```

However, it is essential to understand that the chatbot using python might not know how to answer all your questions. Since its knowledge and training is still very limited, you have to give it time and provide more training data to train it further.

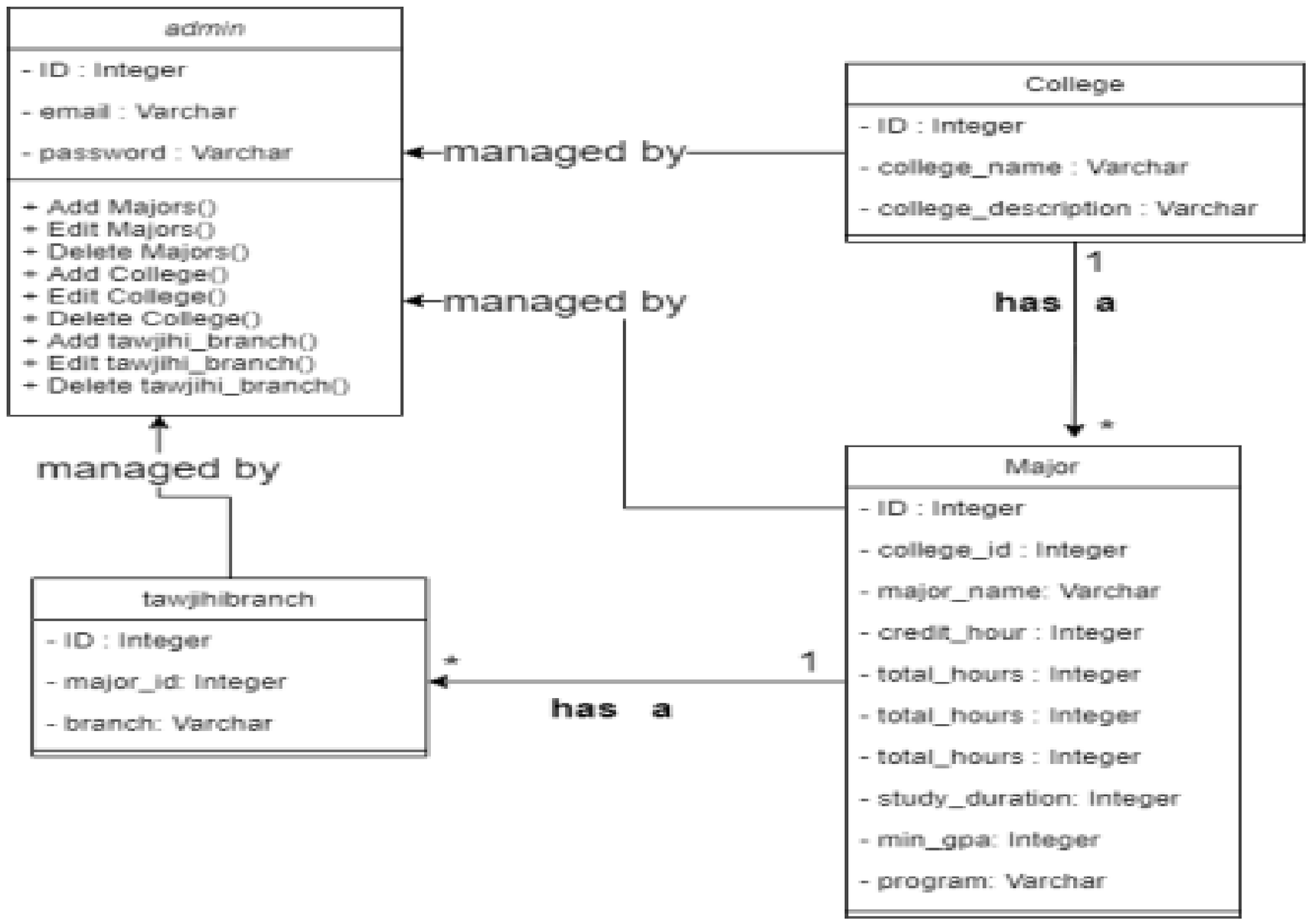
5. Train your Python Chatbot with a Corpus of Data

In this last step of how to make a chatbot in Python, for training your python chatbot even further, you can use an existing corpus of data. Here' s an example of how to train your Python chatbot with a corpus of data provided by the bot itself:

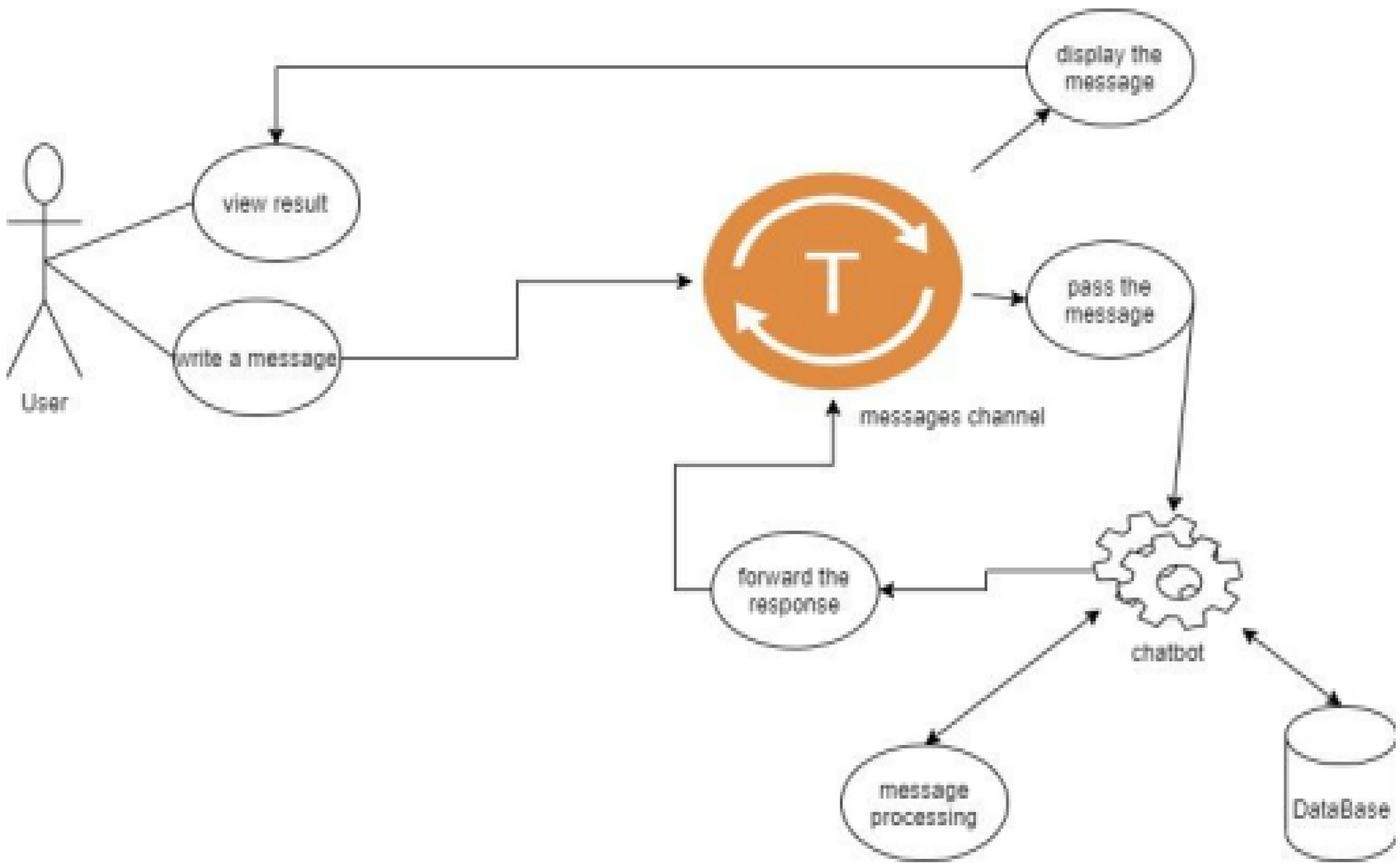
```
from chatterbot.trainers import ChatterBotCorpusTrainer


corpus_trainer = ChatterBotCorpusTrainer(my_bot)
corpus_trainer.train('chatterbot.corpus.english')
```

SYSTEM DESIGN

CLASS DIAGRAM



USE CASE DIAGRAM



BUILDING A CHATBOT USING PYTHON ON WHATSAPP DATA

Inorder to build a chatbot we must follow the following system architecture. There are few steps that we should follow to create the following chatbot on whatsapp data:

STEP 1: CREATE A CHATBOT USING PYTHON CHATTERBOT

In this step, you'll set up a virtual environment and install the necessary dependencies. You'll also create a working command-line chatbot that can reply to you—but it won't have very interesting replies for you yet.

To get started with your chatbot project, create and activate a virtual environment, then install chatterbot and pytz:

```
PS> python -m venv venv
PS> venv\Scripts\activate
(venv) PS> python -m pip install chatterbot==1.0.4 pytz
```

Running these commands in your terminal application installs ChatterBot and its dependencies into a new Python virtual environment.

After the installation is complete, running python -m pip freeze should bring up list of installed dependencies that's similar to what you can find in the provided sample code's requirements.txt file

With the installation out of the way, and ignoring some of the issues that the library currently has, you're ready to get started! Create a new Python file, call it bot.py, and add the code that you need to get a basic chatbot up and running

Python

```python
# bot.py

from chatterbot import ChatBot

chatbot = ChatBot("Chatpot")

exit_conditions = (":q", "quit", "exit")
while True:
    query = input("> ")
    if query in exit_conditions:
        break
    else:
        print(f"🪴 {chatbot.get_response(query)}")
```

After importing ChatBot in line 3, you create an instance of ChatBot in line 5. The only required argument is a name, and you call this one "Chatpot". No, that's not a typo—you'll actually build a chatty flowerpot chatbot in this tutorial! You'll soon notice that pots may not be the best conversation partners after all.

In line 8, you create a while loop that'll keep looping unless you enter one of the exit conditions defined in line 7. Finally, in line 13, you call .get_response() on the ChatBot instance that you created earlier and pass it the user input that you collected in line 9 and assigned to query.

The call to .get_response() in the final line of the short script is the only interaction with your chatbot. And yet—you have a functioning command-line chatbot that you can take for a spin.

When you run bot.py, ChatterBot might download some data and language models associated with the NLTK project. It'll print some information about that to your console. Python won't download this data again during subsequent runs.

If you're ready to communicate with your freshly homegrown Chatpot, then you can go ahead and run the Python file

Shell

```
$ python bot.py
```

After the language models are set up, you'll see the greater than sign (>) that you defined in bot.py as your input prompt. You can now start to interact with your chatty pot

```
Text
```

```
> hello
🤖 hello
> are you a plant?
🤖 hello
> can you chat, pot?
🤖 hello
```

Well ... your chat-pot is *responding*, but it's really struggling to branch out. Tough to expect more from a potted plant— after all, it's never gotten to see the world!

Even if your chat-pot doesn't have much to say yet, it's already learning and growing. To test this out, stop the current session. You can do this by typing one of the exit conditions—

":q", "quit", or "exit". Then start the chatbot another time. Enter a different message, and you'll notice that the chatbot remembers what you typed during the previous run:

```
Text
```

```
> hi
🤖 hello
> what's up?
🤖 are you a plant?
```

During the first run, ChatterBot created a SQLite database file where it stored all your inputs and connected them with possible responses. There should be three new files that have popped up in your working directory

```
./
├── bot.py
├── db.sqlite3
├── db.sqlite3-shm
└── db.sqlite3-wal
```

ChatterBot uses the default SQLStorageAdapter and creates a SQLite file database unless you specify a different storage adapter.

Because you said both *hello* and *hi* at the beginning of the chat, your chat-pot learned that it can use these messages interchangeably. That means if you chat a lot with your new chatbot, it'll gradually have better replies for you. But improving its responses manually sounds like a long process!

Now that you've created a working command-line chatbot, you'll learn how to train it so you can have slightly more interesting conversations.

STEP 2: BEGIN TRAINING YOUR CHATBOT

In the previous step, you built a chatbot that you could interact with from your command line. The chatbot started from a clean slate and wasn't very interesting to talk to.

In this step, you'll train your chatbot using ListTrainer to make it a little smarter from the start. You'll also learn about built-in trainers that come with ChatterBot, including their limitations.

Your chatbot doesn't have to start from scratch, and ChatterBot provides you with a quick way to train your bot. You'll use ChatterBot's ListTrainer to provide some conversation samples that'll give your chatbot more room to grow:

Python

```
1  # bot.py
2
3  from chatterbot import ChatBot
4  from chatterbot.trainers import ListTrainer
5
6  chatbot = ChatBot("Chatpot")
7
8  trainer = ListTrainer(chatbot)
9  trainer.train([
10     "Hi",
11     "Welcome, friend 🤗",
12 ])
13 trainer.train([
14     "Are you a plant?",
15     "No, I'm the pot below the plant!",
16 ])
17
18 exit_conditions = (":q", "quit", "exit")
19 while True:
20     query = input("> ")
21     if query in exit_conditions:
22         break
23     else:
24         print(f"🪴 {chatbot.get_response(query)}")
```

In line 4, you import ListTrainer, to which you pass your chatbot on line 8 to create trainer.

In lines 9 to 12, you set up the first training round, where you pass a list of two strings to trainer.train(). Using .train() injects entries into your database to build upon the graph structure that ChatterBot uses to choose possible replies.

You can run more than one training session, so in lines 13 to 16, you add another statement and another reply to your chatbot's database.

If you now run the interactive chatbot once again using python bot.py, you can elicit somewhat different responses from it than before:

```
Text
> hi
🤖 Welcome, friend 🤗
> hello
🤖 are you a plant?
> me?
🤖 are you a plant?
> yes
🤖 hi
> are you a plant?
🤖 No, I'm the pot below the plant!
> cool
🤖 Welcome, friend 🤗
```

The conversation isn't yet fluent enough that you'd like to go on a second date, but there's additional context that you didn't have before! When you train your chatbot with more data, it'll get better at responding to user inputs.

The ChatterBot library comes with some corpora that you can use to train your chatbot. However, at the time of writing, there are some issues if you try to use these resources straight out of the box.

While the provided corpora might be enough for you, in this tutorial you'll skip them entirely and instead learn how to adapt your own conversational input data for training with ChatterBot's ListTrainer.

To train your chatbot to respond to industry-relevant questions, you'll probably need to work with custom data, for example from existing support requests or chat logs from your company.

Moving forward, you'll work through the steps of converting chat data from a WhatsApp conversation into a format that you can use to train your chatbot. If your own resource is WhatsApp conversation data, then you can use these steps directly. If

your data comes from elsewhere, then you can adapt the steps to fit your specific text format.

To start off, you'll learn how to export data from a WhatsApp chat conversation.

STEP 3: EXPORT A WHATSAPP CHAT

At the end of this step, you'll have downloaded a TXT file that contains the chat history of a WhatsApp conversation. If you don't have a WhatsApp account or don't want to work with your own conversational data, then you can download a sample chat export below

If you're going to work with the provided chat history sample, you can skip to the next section, where you'll clean your chat export.

To export the history of a conversation that you've had on WhatsApp, you need to open the conversation on your phone. Once you're on the conversation screen, you can access the export menu:

1. Click on the three dots (⋮) in the top right corner to open the main menu.
2. Choose *More* to bring up additional menu options.
3. Select *Export chat* to create a TXT export of your conversation

Once you've clicked on *Export chat*, you need to decide whether or not to include media, such as photos or audio messages. Because your chatbot is only dealing with text, select *WITHOUT MEDIA*. Then, you can declare where you'd like to send the file.

In this example, you saved the chat export file to a Google Drive folder named *Chat exports*. You'll have to set up that folder in your Google Drive before you can select it as an option. Of course, you don't need to use Google Drive. As long as you save or send your chat export file so that you can access to it on your computer, you're good to go.

Once that's done, switch back to your computer. Find the file that you saved, and download it to your machine.

Specifically, you should save the file to the folder that also contains bot.py and rename it chat.txt. Then, open it with your favorite text editor to inspect the data that you received:

Text

```
9/15/22, 14:50 - Messages and calls are end-to-end encrypted.
↳ No one outside of this chat, not even WhatsApp, can read
↳ or listen to them. Tap to learn more.
9/15/22, 14:49 - Philipp: Hi Martin, Philipp here!
9/15/22, 14:50 - Philipp: I'm ready to talk about plants!
9/15/22, 14:51 - Martin: Oh that's great!
9/15/22, 14:52 - Martin: I've been waiting for a good convo about
↳ plants for a long time
9/15/22, 14:52 - Philipp: We all have.
9/15/22, 14:52 - Martin: Did you know they need water to grow?
...
```

Step 4: Clean Your Chat Export

In this step, you'll clean the WhatsApp chat export data so that you can use it as input to train your chatbot on an industry-specific topic. In this example, the topic will be ... houseplants!

Most data that you'll use to train your chatbot will require some kind of cleaning before it can produce useful results. It's just like the old saying goes:

Garbage in, garbage out (Source)

Take some time to explore the data that you're working with and to identify potential issues:

Text

```
9/15/22, 14:50 - Messages and calls are end-to-end encrypted.
↳ No one outside of this chat, not even WhatsApp, can read
↳ or listen to them. Tap to learn more.

...

9/15/22, 14:50 - Philipp: I'm ready to talk about plants!

...

9/16/22, 06:34 - Martin: <Media omitted>

...
```

For example, you may notice that the first line of the provided chat export isn't part of the conversation. Also, each actual message starts with metadata that includes a date, a time, and the username of the message sender.

If you scroll further down the conversation file, you'll find lines that aren't real messages. Because you didn't include media files in the chat export, WhatsApp replaced these files with the text <Media omitted>.

All of this data would interfere with the output of your chatbot and would certainly make it sound much less conversational. Therefore, it's a good idea to remove this data.

Open up a new Python file to preprocess your data before handing it to ChatterBot for training. Start by reading in the file content and removing the chat metadata:

Python

```python
# cleaner.py

import re

def remove_chat_metadata(chat_export_file):
    date_time = r"(\d+\/\d+\/\d+,\s\d+:\d+)"  # e.g. "9/16/22, 06:34"
    dash_whitespace = r"\s-\s"  # " - "
    username = r"([\w\s]+)"  # e.g. "Martin"
    metadata_end = r":\s"  # ": "
    pattern = date_time + dash_whitespace + username + metadata_end

    with open(chat_export_file, "r") as corpus_file:
        content = corpus_file.read()
    cleaned_corpus = re.sub(pattern, "", content)
    return tuple(cleaned_corpus.split("\n"))

if __name__ == "__main__":
    print(remove_chat_metadata("chat.txt"))
```

This function removes conversation-irrelevant message metadata from the chat export file using the built-in re module, which allows you to work with regular expressions:

- **Line 3** imports re.
- **Lines 6 to 9** define multiple regex patterns. Constructing multiple patterns helps you keep track of what you're matching and gives you the flexibility to use the separate capturing groups to apply further preprocessing later on. For example, with access to username, you could chunk conversations by merging messages sent consecutively by the same user
- **Line 10** concatenates the regex patterns that you defined in lines 6 to 9 into a single pattern. The complete pattern matches all the metadata that you want to remove.
- **Lines 12 and 13** open the chat export file and read the data into memory.
- **Line 14** uses re.sub() to replace each occurrence of the pattern that you defined in pattern with an empty string (""), effectively deleting it from the string.
- **Line 15** first splits the file content string into list items using .split("\n"). This breaks up cleaned_corpus into a list where each line represents a separate item. Then, you convert this list into a tuple and return it from remove_chat_metadata().

- **Lines 17 and 18** use Python's name-main idiom to call remove_chat_metadata() with "chat.txt" as its argument, so that you can inspect the output when you run the script.

Eventually, you'll use cleaner as a module and import the functionality directly into bot.py. But while you're developing the script,it's helpful to inspect intermediate outputs, for example with a print() call, as shown in line 18.

After removing the message metadata from each line, you also want to remove a few complete lines that aren't relevant for the conversation. To do this, create a second function in your data cleaning script:

Python

```
1  # cleaner.py
2
3  # ...
4
5  def remove_non_message_text(export_text_lines):
6      messages = export_text_lines[1:-1]
7
8      filter_out_msgs = ("<Media omitted>",)
9      return tuple((msg for msg in messages if msg not in filter_out_msgs))
10
11 if __name__ == "__main__":
12     message_corpus = remove_chat_metadata("chat.txt")
13     cleaned_corpus = remove_non_message_text(message_corpus)
14     print(cleaned_corpus)
```

- In remove_non_message_text(), you've written code that allows you to remove **Line 6** removes the first introduction line, which every WhatsApp chat export comes with, as well as the empty line at the end of the file.
- **Line 8** creates a tuple where you can define what strings you want to exclude from the data that'll make it to training. For now, it only contains one string, but if you wanted to remove other content as well, you could quickly add more strings to this tuple as items.
- **Line 9** filters messages for the strings defined in filter_out_msgs using a generator expression that you convert to a tuple before returning it.

Finally, you've also changed lines 12 to 14. You now collect the return value of the first function call in the variable message_corpus, then use it as an argument to remove_non_message_text(). You save the result of that function call to cleaned_corpus and print that value to your console on line 14.

Because you want to treat cleaner as a module and run the cleaning code in bot.py, it's best to now refactor the code in the name-main idiom into a main function that you can then import and call in bot.py

Python

```python
# cleaner.py

import re

def clean_corpus(chat_export_file):
    message_corpus = remove_chat_metadata(chat_export_file)
    cleaned_corpus = remove_non_message_text(message_corpus)
    return cleaned_corpus

# ...

# Deleted: if __name__ == "__main__":
```

You refactor your code by moving the function calls from the name-main idiom into a dedicated function, clean_corpus(), that you define toward the top of the file. In line 6, you replace "chat.txt" with the parameter chat_export_file to make it more general. You'll provide the filename when calling the function. The clean_corpus() function returns the cleaned corpus, which you can use to train your chatbot.

After creating your cleaning module, you can now head back over to bot.py and integrate the code into your pipeline.

Python

```python
# bot.py

from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer
from cleaner import clean_corpus

CORPUS_FILE = "chat.txt"

chatbot = ChatBot("Chatpot")

trainer = ListTrainer(chatbot)
cleaned_corpus = clean_corpus(CORPUS_FILE)
trainer.train(cleaned_corpus)

exit_conditions = (":q", "quit", "exit")
while True:
    query = input("> ")
    if query in exit_conditions:
        break
    else:
        print(f"🪴 {chatbot.get_response(query)}")
```

Step 5: Train Your Chatbot on Custom Data and Start Chatting

In this step, you'll train your chatbot with the WhatsApp conversation data that you cleaned in the previous step. You'll end up with a chatbot that you've trained on industry-specific conversational data, and you'll be able to chat with the bot—about houseplants!

Open up bot.py and include calls to your cleaning functions in the code:

Python

```python
# bot.py

from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer
from cleaner import clean_corpus

CORPUS_FILE = "chat.txt"

chatbot = ChatBot("Chatpot")

trainer = ListTrainer(chatbot)
cleaned_corpus = clean_corpus(CORPUS_FILE)
trainer.train(cleaned_corpus)

exit_conditions = (":q", "quit", "exit")
while True:
    query = input("> ")
    if query in exit_conditions:
        break
    else:
        print(f"🤖 {chatbot.get_response(query)}")
```

You don't need to do a lot of extra work in this file, because you've prepared your cleaning functions well:

- **Line 5** imports clean_corpus() from cleaner.
- **Line 7** defines the filename of your chat export file. In this example, you assume that it's called "chat.txt", and it's located in the same directory as bot.py. If you need more advanced path handling, then take a look at Python's pathlib module.
- **Line 12** applies your cleaning code to the chat history file and returns a tuple of cleaned messages, which you call cleaned_corpus.
- **Line 13** finally uses that data as input to .train(), effectively training your chatbot with the WhatsApp conversation data.

It's important to know that ChatterBot will treat an iterable with more than two items slightly differently than when there are only two items, like you had in your previous training runs:

If you provide longer lists of training conversations, then this will establish each item in the list as a possible response to its predecessor in the list. (Source)

Depending on your input data, this may or may not be exactly what you want. For the provided WhatsApp chat export data, this isn't ideal because not every line represents a question followed by an answer.

To deal with this, you could apply additional preprocessing on your data, where you might want to group all messages sent by the same person into one line, or chunk the chat export by time and date. That way, messages sent within a certain time period could be considered a single conversation.

There's always more to clean to improve your input data! For this tutorial, however, it's time to give the conversational houseplant chatbot another go. Run bot.py and chat with your chatty pot to test how its responses have changed due to training:

Text

```
> hi
🪴 Welcome, friend 😊
> thanks for the green welcome
🪴 I let you
> you let me be here?
🪴 It's a monsters!
> did you mean monstera?
🪴 The leafs are getting dryer and dryer. But she's growing new ones
> who?
🪴 Do raindrops touch their leaves?
> very philosophical!
🪴 Lol
> ;)
🪴 I don't grow any crop at home
> no crops in pots
🪴 Ah, gotcha!
> ah, I just thought you meant that
🪴 But water makes sense
> yes, water in pots
🪴 Do you have any plant care pro tips?
> water.
🪴 I always thought that love and music was more than enough
> haha
🪴 Well, no. Just the balcony. This worked best
> ok, balcony is notes
🪴 I let you
```

Your chatbot has increased its range of responses based on the training data that you fed to it. As you might notice when you interact with your chatbot, the responses don't always make a lot of sense.

ChatterBot uses complete sentences as replies, and you trained it on just a small sample of conversational data. The small size of the training data means that it doesn't have a big pool of replies to pick from. And because the data is conversational, not all of the sentences are very useful as replies. But Chatpot is doing all it can to find the best matching reply to any new message that you type!

To select a response to your input, ChatterBot uses the BestMatch logic adapter by default. This logic adapter uses the Levenshtein distance to compare the input string to all statements in the database. It then picks a reply to the statement that's closest to the input string.

If you use well-structured input data, then the default settings of ChatterBot give you decent results out of the box. And if you're ready to do some extra work to get just what you want, then you're in luck! ChatterBot allows for a lot of customization and provides some instructions to guide you in the right direction:

ChatterBot provides you with reasonable defaults. But if you want to customize any part of the process, then it gives you all the freedom to do so.

In this section, you put everything back together and trained your chatbot with the cleaned corpus from your WhatsApp conversation chat export. At this point, you can already have fun conversations with your chatbot, even though they may be somewhat nonsensical. Depending on the amount and quality of your training data, your chatbot might already be more or less useful.

SOURCE OVERVIEW

Chatbots— automated tools that engage customers in conversation— allow companies to answer customers' queries while minimising the resources required to do so. For example, companies can use chatbots to help prospects navigate their website and make purchases. However, chatbots have a downside—they're usually only available on companies' websites. This is where WhatsApp chatbots come in.

WhatsApp chatbots are exactly as they sound: **automated chatbots that run on the WhatsApp platform**. They work just like other chatbots, sending customers messages, responding to their queries, and helping them navigate their buying journey.

Think of WhatsApp chatbots as the latest iteration of chatbot technology. Companies have long recognised how powerful chatbots can be, improving customers' experience while minimising unnecessary customer service expenditure.

WhatsApp chatbots provide the same level of customer service and support as regular chatbots. But consumers don't need to visit companies' websites to gain answers to their questions from these bots—they can just ping them by sending a message through the world's most popular messaging app.

WhatsApp boasts over 2 billion monthly active users, and its user base is increasing each year. It's a channel consumers already know and love— which makes it ideal for customer service. Companies can meet consumers where they are, offering proactive assistance that's more likely to boost engagement. For example, rather than visiting your website and logging in to track an order status, customers can just ping off a quick WhatsApp message and receive a near-instant response.

TYPES OF WHATSAPP CHATBOT

Once you have the business goal in mind, you can start thinking about the type of chatbot that would be best suited to help. Chatbots are all about improving the customer experience by offering immediate service, 24/7. This will in turn help to offload repetitive work from your contact center, who can concentrate on queries that require

a human. Remember that you can get a lot of value from a simple chatbot that is designed for a specific purpose. It is a good idea to start with a simple use case and then extend to more advanced functionality once you have mastered the basics.

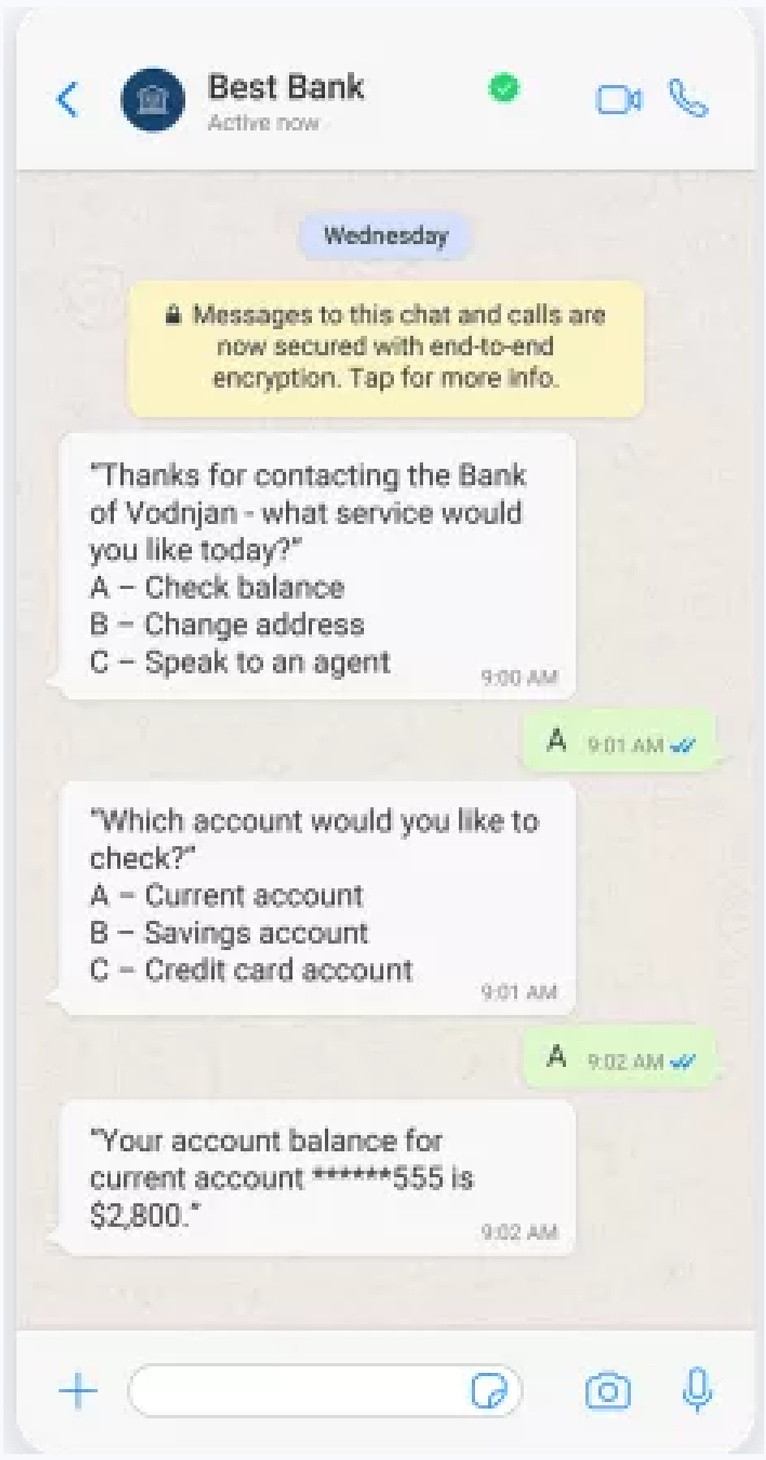
Here are three types of chatbot that you might want to consider.

1. RULE-BASED CHATBOTS

These are the simplest type of WhatsApp chatbots that can literally be created in minutes. They offer a list of options for the person to choose, using interactive buttons in the UI or by replying with option 'A' etc.

Simplicity does not mean they aren't incredibly useful – consider the following example:

A quick and easy interaction, but very powerful considering that you just saved your customer several minutes on hold and freed up an agent to answer a more complex query.



2. INTENT BASED CHATBOTS

These are more advanced bots that use natural language processing (NLP) to work out what the person is trying to achieve – i.e. what their intent is.

For example, the following phrases would all be recognized and provided with the same answer:

"account balance"

"what is my account total?"

"how much is in my current account?"

These bots aim to replicate the normal conversational experience that you would have with another human – think of them as WhatsApp equivalents of Alexa or Siri, which are trained to understand human language with all its variances and nuances. The more data they have access to, the more useful they will be.

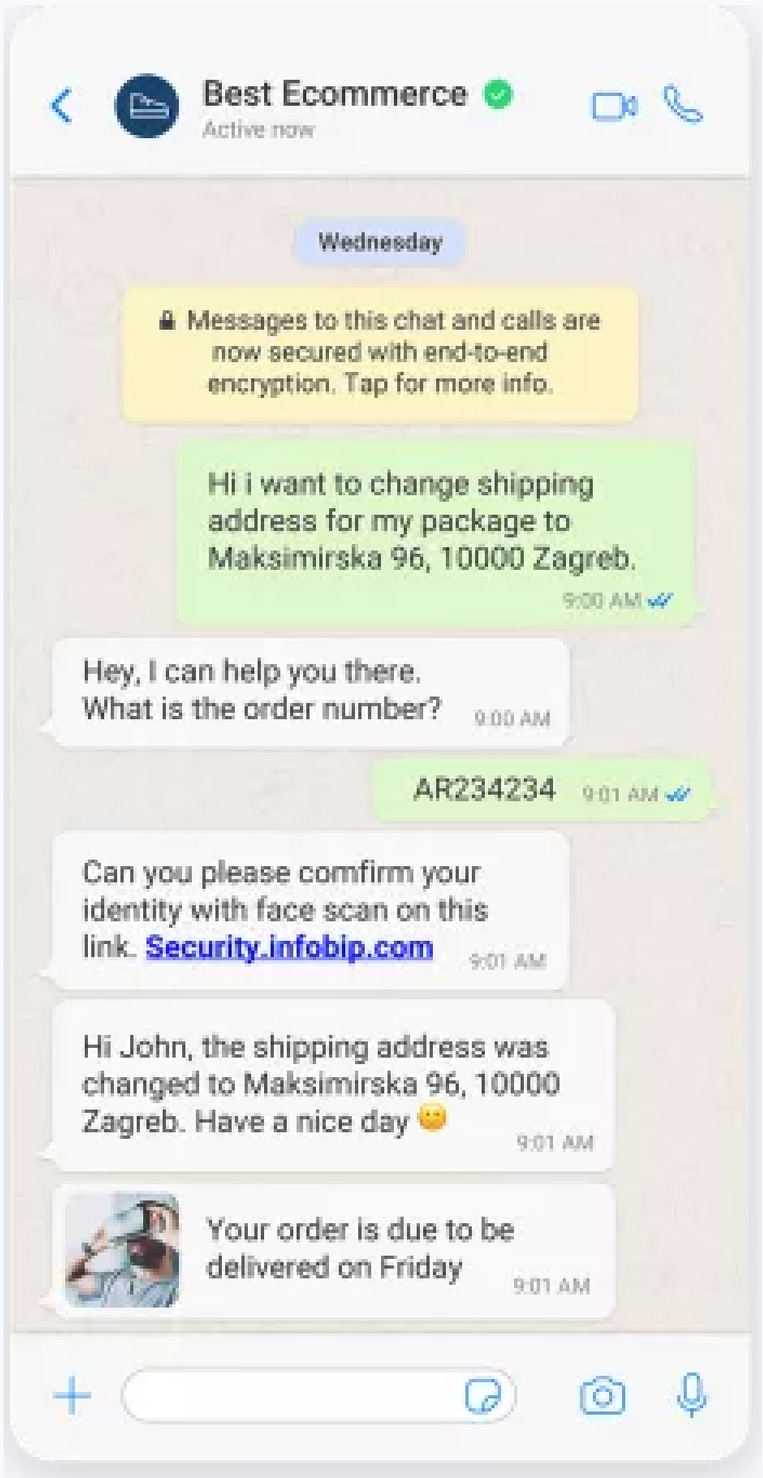
Training your bot will take time and resources, but this can be offset by crowdsourcing intents from your chatbot supplier or making their creation a group effort for your company. The more people writing intents for your chatbot, the more it will be able to identify and accurately respond to different users' questions.

In the end you will end up with something exceptional that can provide unique and memorable experiences for your customers.

3. SECURED CHATBOTS

This is more a bot feature than a type on its own. As we know, WhatsApp is already highly secure with all messages and media encrypted. However, when dealing with sensitive information you can add an additional layer of security that requires users to authenticate themselves, just like they would if they were logging into an account online or calling an agent and providing their security credentials.

You can even incorporate biometric authentication like face scan technology that makes the security check process as quick and simple as taking a selfie.



4. TRUSTED PERSONAL ASSISTANTS

With the development of secure chatbots there has been a shift in the types of use cases that organizations are able to fulfill. A key grouth area is the introduction of WhatsApp chatbots that help people in their private lives. This could be for medical purposes, financial planning, or addiction recovery. The key is that people now have a **high level of trust** in these chatbots and are willing to share personal information in return for the support and advice that the chatbot can offer.

An excellent example of this is the WhatsApp chatbot that has been created by Magdalena Clinic, an Infobip partner who provide their patients with comprehensive cardiovascular care. Recognizing that a lot of their chronic outpatients needed additional support, the clinic built a chatbot using Infobip' s self-service platform.

Called Megi, the chatbot' s key advantage is that it can facilitate the **two-way exchange of valuable data and information**. Patients can provide details of symptoms and submit measurements of their blood pressure and Megi can then provide relevant support and

information. Where the submitted readings indicate that intervention is required, the chatbot can connect patients directly with a medical professional.

Patients have consistently commented that dealing with Megi is like speaking to a real person and mention the sense of comfort they get knowing that the service is always available. Nothing can replace the care provided by humans, but with budgets being squeezed, there is a huge opportunity for organizations to use chatbots to supplement the work of trained professionals.

SPECIFIC WHATSAPP CHATBOT USE CASES

Okay, enough theory— let' s explore precisely how companies can use WhatsApp chatbots to boost their customer experience.

1. Generating sales: WhatsApp chatbots can attract sales via ads. For example, a company might run a series of social ads on Instagram and Facebook that contain a call-to-action (CTA), something simple and direct, like 'Contact Us' . When customers click the CTA, they are redirected to the WhatsApp chatbot, which they can use to place orders.

2. Support throughout the sales funnel: Companies can engage WhatsApp chatbots throughout all stages of the sales funnel. For example, prospects can use WhatsApp chatbots to ask about an item' s price, the various sizing options that are available, when it' ll next be in stock, and its scheduled delivery date. Post-purchase, they can use the same WhatsApp chatbot to schedule returns, sort out refunds, or leave a review.

3. To gather feedback: As things stand, 89% of companies compete primarily based on customer experience (CX). Having a fantastic CX is a huge competitive advantage— it sets you apart from other companies and boosts your brand' s wider reputation. Companies can use WhatsApp chatbots to gather product reviews, solicit feedback on the sales process; and help determine customer satisfaction (CSAT) scores. Best of all, these reviews are incredibly easy for customers to leave. They just have to open WhatsApp and ping off a quick message (or select a suitable emoji)— the process takes a matter of seconds.

4. Abandoned cart recovery: Unfortunately, 68.8% of all carts are abandoned during the sales process. But there' s good news— by sending a personalised message

reminding customers that they have products waiting in their cart, companies can encourage prospects to re-engage with the buying process. One company even used WhatsApp reminders to recover a massive 70% of its abandoned carts.

However, remember a few clear rules. First, don't be overly pushy. Gently remind customers that they've left some products in their cart— don't demand that they check out. Second, remind them of any upcoming deadlines if relevant (e.g. 'Complete your purchase before the 20% sale ends this Friday'). Third, make it as easy as possible for them to complete the purchase, including a link that takes them straight to the checkout page.

THE FUTURE OF CHATBOTS

WhatsApp chatbots are the future of customer service, allowing businesses to leverage two increasingly popular communication channels: chatbots and WhatsApp. Companies can provide unbeatable customer support at all hours while meeting consumers where they are. In turn, this will boost the customer experience, transform retention rates, and increase the business's bottom line.

EXECUTION OF PROGRAM SCREEN SHOT

SCREEN SHOT 1

Text

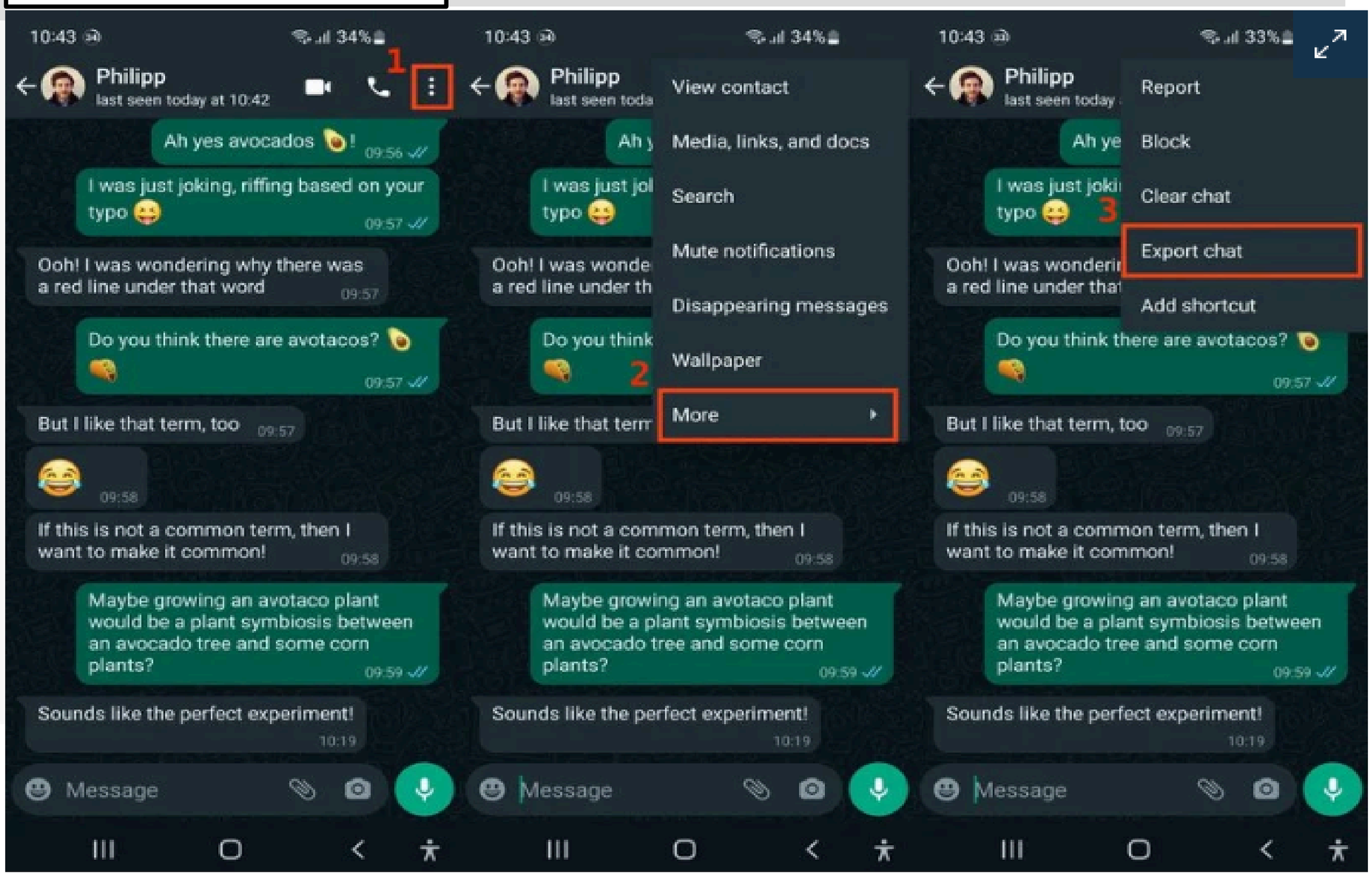
```
> hello
 hello
> are you a plant?
 hello
> can you chat, pot?
 hello
```

SCREEN SHOT 2

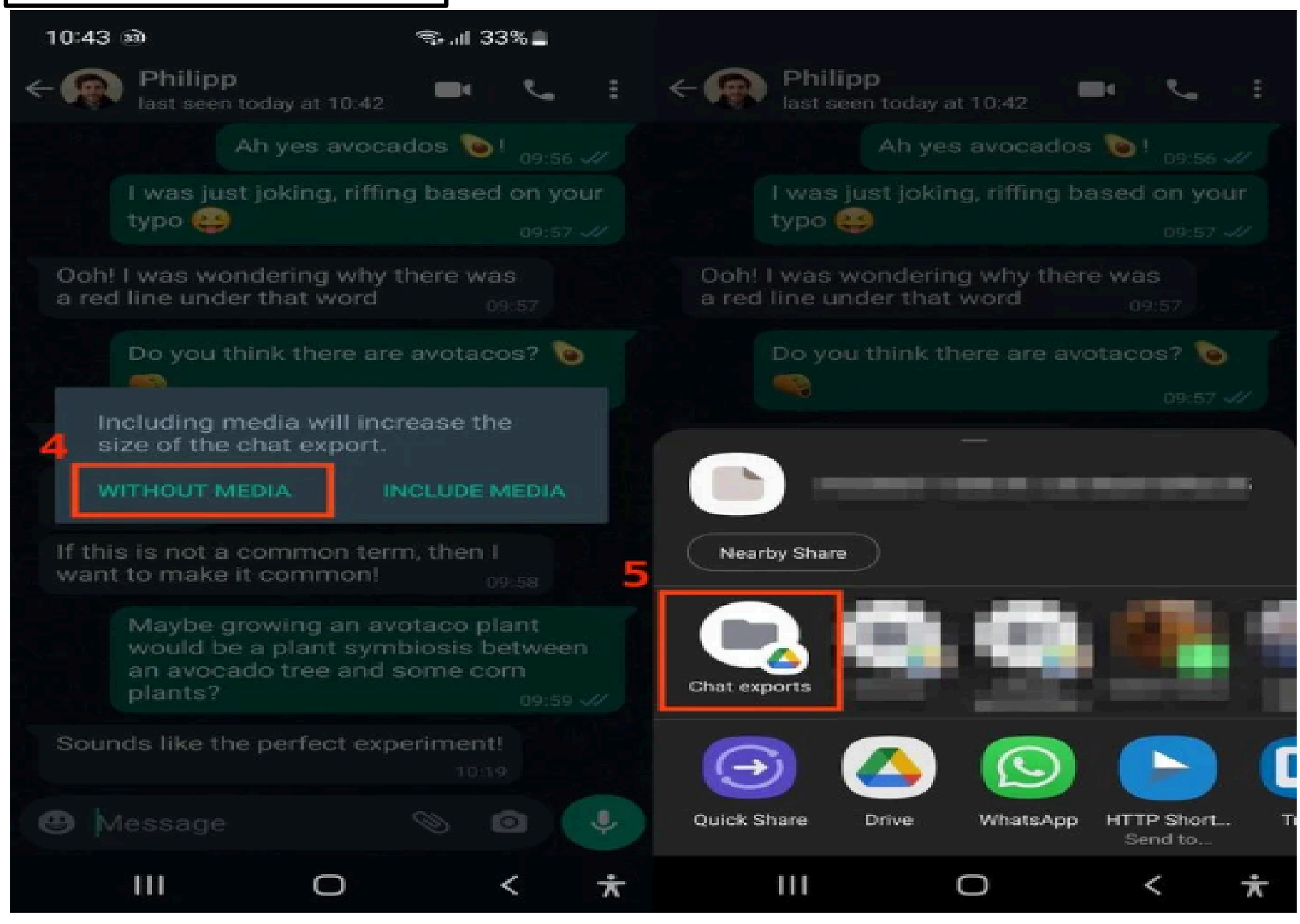
Text

```
> hi
 Welcome, friend 
> hello
 are you a plant?
> me?
 are you a plant?
> yes
 hi
> are you a plant?
 No, I'm the pot below the plant!
> cool
 Welcome, friend 
```

SCREEN SHOT 3



SCREEN SHOT4



SCREEN SHOT 5

Text

```
9/15/22, 14:50 - Messages and calls are end-to-end encrypted.
↳ No one outside of this chat, not even WhatsApp, can read
↳ or listen to them. Tap to learn more.
9/15/22, 14:49 - Philipp: Hi Martin, Philipp here!
9/15/22, 14:50 - Philipp: I'm ready to talk about plants!
9/15/22, 14:51 - Martin: Oh that's great!
9/15/22, 14:52 - Martin: I've been waiting for a good convo about
↳ plants for a long time
9/15/22, 14:52 - Philipp: We all have.
9/15/22, 14:52 - Martin: Did you know they need water to grow?
...
```

SCREEN SHOT 6

Text

```
9/15/22, 14:50 - Messages and calls are end-to-end encrypted.
↳ No one outside of this chat, not even WhatsApp, can read
↳ or listen to them. Tap to learn more.

...

9/15/22, 14:50 - Philipp: I'm ready to talk about plants!

...

9/16/22, 06:34 - Martin: <Media omitted>

...
```

Text

```
> hi
 Welcome, friend 🤗
> thanks for the green welcome
 I let you
> you let me be here?
 It's a monsters!
> did you mean monstera?
 The leafs are getting dryer and dryer. But she's growing new ones
> who?
 Do raindrops touch their leaves?
> very philosophical!
 Lol
> ;)
 I don't grow any crop at home
> no crops in pots
 Ah, gotcha!
> ah, I just thought you meant that
 But water makes sense
> yes, water in pots
 Do you have any plant care pro tips?
> water.
 I always thought that love and music was more than enough
> haha
 Well, no. Just the balcony. This worked best
> ok, balcony is notes
 I let you
```