Ampalibe

Release 1.0.4

iTeam-$

Apr 16, 2022

CONTENTS

Ampalibe is a Python framework to quickly create bots or applications on facebook messenger. It uses api from the Messenger platforms and offers a *simple* and *quick* functions to use it.

Check out the *Get Started* section for further information, including how to *Installation* the project.

you can also see video resource links here

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 Why use Ampalibe

1.1.1 Webhooks & process Management

No need to manage webhooks and data

messages are received directly in a main function

```
import ampalibe
from conf import Configuration

bot = ampalibe.init(Configuration())
chat = bot.chat

@ampalibe.command('/')
def main(sender_id, cmd, **extends):
   chat.send_message(sender_id, 'Hello world')
   chat.send_message(sender_id, f'This is your message: {cmd}')
   chat.send_message(sender_id, f'and this is your facebook id: {sender_id}')
```

1.1.2 Action Management

Manages the actions expected by the users

define the function of the next treatment

```
import ampalibe
from conf import Configuration

bot = ampalibe.init(Configuration())
chat = bot.chat
query = bot.query

@ampalibe.command('/')
def main(sender_id, cmd, **extends):
    chat.send_message(sender_id, 'Enter your name')
    query.set_action(sender_id, '/get_name')
```

(continues on next page)

(continued from previous page)

```
@ampalibe.action('/get_name')
def get_name(sender_id, cmd, **extends):
    query.set_action(sender_id, None)  #  clear current action
    chat.send_message(sender_id, f'Hello {cmd}')
```

1.1.3 Temporary data Management

Manage temporary data easily with set, get, and delete methods

```
import ampalibe
from conf import Configuration

bot = ampalibe.init(Configuration())
chat = bot.chat
query = bot.query

@ampalibe.command('/')
def main(sender_id, cmd, **extends):
    chat.send_message(sender_id, 'Enter your mail')
    query.set_action(sender_id, '/get_mail')

@ampalibe.action('/get_mail')
def get_mail(sender_id, cmd, **extends):
    # save the mail in temporary data
    query.set_temp(sender_id, 'mail', cmd)

    chat.send_message(sender_id, f'Enter your password')
    query.set_action(sender_id, '/get_password')


@ampalibe.action('/get_password')
def get_password(sender_id, cmd, **extends):
    query.set_action(sender_id, None)  # clear current action
    mail = query.get_temp(sender_id, 'mail')  # get mail in temporary data

    chat.send_message(sender_id, f'your mail and your password are {mail} {cmd}')
    query.del_temp(sender_id, 'mail')  # delete temporary data
```

1.1.4 Payload Management

Manage Payload easily

send data with Payload object and get it in destination function's parameter

```
import ampalibe
from ampalibe import Payload
from conf import Configuration

bot = ampalibe.init(Configuration())
chat = bot.chat


@ampalibe.command('/')
def main(sender_id, cmd, **extends):
    quick_rep = [
        {
            "content_type": "text",
            "title": 'Angela',
            "payload": Payload('/membre', name='Angela', ref='2016-sac')
        },
        {
            "content_type": "text",
            "title": 'Rivo',
            "payload": Payload('/membre', name='Rivo')
        }
    ]
    chat.send_quick_reply(sender_id, quick_rep, 'Who?')


@ampalibe.command('/membre')
def get_membre(sender_id, cmd, name, **extends):
    chat.send_message(sender_id, "Hello " + name)

    # if the arg is not defined in the list of parameters,
    # it is put in the extends variable
    if extends.get('ref'):
        chat.send_message(sender_id, 'your ref is ' + extends.get('ref'))
```

1.1.5 Advanced Messenger API

No need to manage the length of the items to send: A next page button will be displayed directly

```
import ampalibe
from ampalibe import Payload
from conf import Configuration

bot = ampalibe.init(Configuration())
chat = bot.chat
```

(continues on next page)

(continued from previous page)

```
@ampalibe.command('/')
def main(sender_id, cmd, **extends):
    list_items = [
        {
            "title": f"item n°{i+1}",
            "image_url": "https://i.imgflip.com/6b45bi.jpg",
            "buttons": [
                {
                    "type": "postback",
                    "title": "Get item",
                    "payload": Payload("/item", id_item=i+1)
                }
            ]
        }
        for i in range(30)
    ]
    # next=True for displaying directly next page button.
    chat.send_template(sender_id, list_items, next=True)

@ampalibe.command('/item')
def get_item(sender_id, id_item, **extends):
    chat.send_message(sender_id, f"item n°{id_item} selected")
```

1.2 Get Started

1.2.1 Installation

To use Ampalibe, first install it using pip:

```
$ pip install ampalibe
```

1.2.2 Creation of a new project

After installation, an `ampalibe` executable is available in your system path, and we will create our project with this command.

Important: command-line `ampalibe` is `ampalibe.bat` for **Windows**

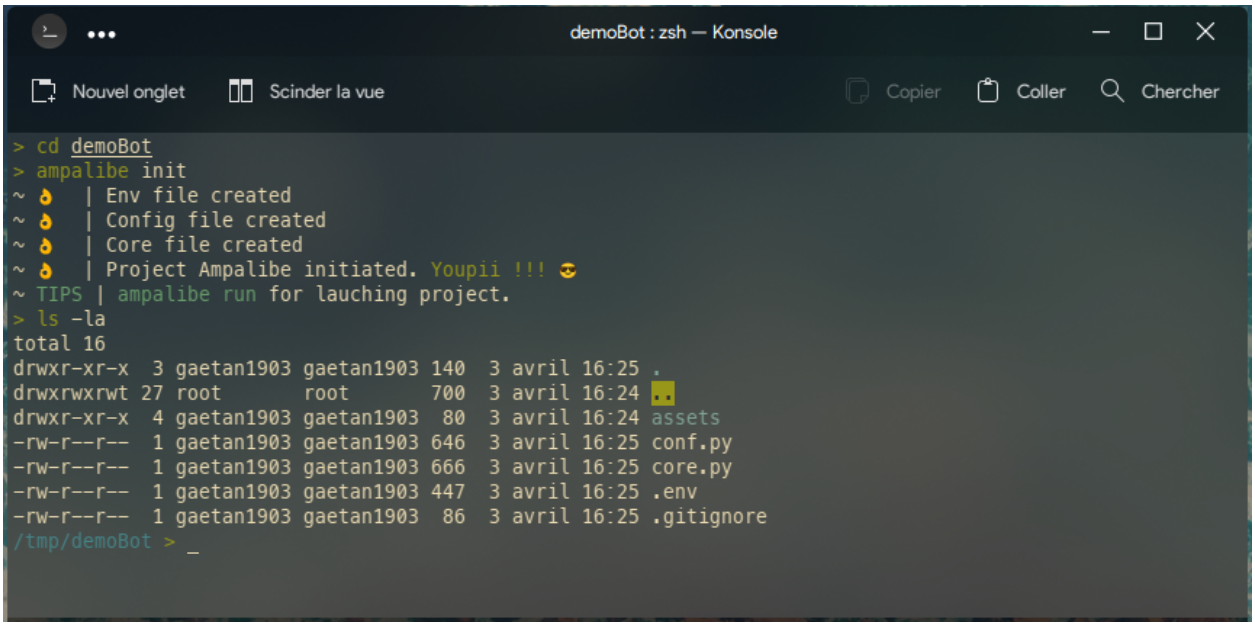
```
$ ampalibe create myfirstbot
```

There will be a created directory named **myfirstbot/** and all the files contained in it.

Note: You can directly create project without a new directory with **init** command

```
$ cd myExistingDirectory
$ ampalibe init
```

1.2.3 Understanding of files



- **`assets/` statics file folder**
  `public/` reachable via url
  `private/` not accessible via url
- `.env` environment variable file
- `conf.py` configuration file that retrieves environment variables
- `core.py` file containing the starting point of the code

Important: .env file is env.bat in Windows

1.2.4 Before starting

How to complete the environment variable file

- **AMP_ACCESS_TOKEN** Facebook Page access token
- **AMP_VERIF_TOKEN** Token that Facebook use as part of the recall URL check.
- **ADAPTER type of database used by ampalibe (SQLITE OR MYSQL)**

  FOR MYSQL ADAPTER
  *DB_HOST**
  DB_USER
  DB_PASSWORD
  DB_NAME
  DB_PORT

  FOR SQLITE ADAPTER
  DB_FILE
- **AMP_HOST** server listening address
- **AMP_PORT** server listening port
- **AMP_URL** URL of the server given to Facebook

1.2.5 Run the app

In the project folder, type

```
$ ampalibe run
```

for dev mode with **Hot Reload**

```
$ ampalibe run --dev
```

```
INFO:     Started server process [26753]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:4555 (Press CTRL+C to quit)
```

Note: Ampalibe use uvicorn to run server, so it is an output of uvicorn

1.3 Usage

1.3.1 A Minimal Application

A minimal Ampalibe application looks something like this:

```
import ampalibe
from conf import Configuration

bot = ampalibe.init(Configuration())

@ampalibe.command('/')
def main(sender_id, cmd, **extends):
    bot.chat.send_message(sender_id, "Hello, Ampalibe")
```

So what did that code do?

- `line 1` we import ampalibe package.
- `line 2` we import Configuration class which contains our env variable.
- `line 4` we inite ampalibe with configuration and store the result instance.
- we create a function decorated by ampalibe.command('/') to say Ampalibe that it's the main function.
- `line 8` We respond to the sender by greeting Ampalibe

Note: messages are received directly in a main function

1.3.2 Command

Ampalibe's philosophy says that all received messages, whether it is a simple or payload message, or an image, is considered to be commands

The command decorator of ampalibe is used to specify the function where a specific command must launch. If no corresponding command is found, it launches the main function

```
import ampalibe
from conf import Configuration

bot = ampalibe.init(Configuration())

@ampalibe.command('/')
def main(sender_id, cmd, **extends):
    bot.chat.send_message(sender_id, "Hello, Ampalibe")

'''
    if the message received start with '/eat'
    the code enter here, not in main
    ex: /eat 1 jackfruit

    in cmd value, /eat is removed
'''
@ampalibe.command('/eat')
def spec(sender_id, cmd, **extends):
    print(sender_id)  # 1555554455
    print(cmd)  # '1 jackfruit'


'''
   ex: /drink_a_jackfruit_juice
'''
@ampalibe.command('/drink_a_jackfruit_juice')
def spec(sender_id, cmd, **extends):
    print(sender_id)  # 1555554455
    print(cmd)  #  a empty value cause the key is removed
```

Note: When we create a function decorated by ampalibe.command, `**extends` parameter must be present

1.3.3 Action

At some point, we will need to point the user to a specific function, to retrieve user-specific data, for example, ask for his email, ask for the word the person wants to search for.

To do this, you have to define the action expected by the user, to define what should be expected from the user.

in this example, we will use two things, the **action decorator** and the **query.set_action** method

Example 1: Ask the name of user, and greet him

```
import ampalibe
from conf import Configuration
```

(continues on next page)

(continued from previous page)

```
bot = ampalibe.init(Configuration())
chat = bot.chat
query = bot.query

@ampalibe.command('/')
def main(sender_id, cmd, **extends):
    chat.send_message(sender_id, 'Enter your name')
    query.set_action(sender_id, '/get_name')

@ampalibe.action('/get_name')
def get_name(sender_id,  cmd, **extends):
    query.set_action(sender_id, None)  #  clear current action
    chat.send_message(sender_id, f'Hello {cmd}')
```

Example 2: Ask a number and say if it a even number or odd number

```
import ampalibe
from conf import Configuration

bot = ampalibe.init(Configuration())
chat = bot.chat
query = bot.query

@ampalibe.command('/')
def main(sender_id, cmd, **extends):
    chat.send_message(sender_id, 'Enter a number')
    query.set_action(sender_id, '/get_number')

@ampalibe.action('/get_number')
def get_number(sender_id, cmd, **extends):
    query.set_action(sender_id, None)  #  clear current action
    if cmd.isdigit():
        if int(cmd) % 2 == 0:
            chat.send_message(sender_id, 'even number')
        else:
            chat.send_message(sender_id, 'odd number')
    else:
        chat.send_message(sender_id, f'{cmd} is not a number')
```

We define the next function in which the user message entered and can obtain all the texts of the message in “cmd”

Important: Remember to erase the current action to prevent the message from entering the same function each time

Note: When we create a function decorated by ampalibe.action, `**extends` parameter must be present

1.3.4 Temporary data

For each processing of each message, we will need to store information temporarily, like saving the login while waiting to ask for the password

the methods used are **set_temp**, **get_temp**, **del_temp**

```
import ampalibe
from conf import Configuration

bot = ampalibe.init(Configuration())
chat = bot.chat
query = bot.query

@ampalibe.command('/')
def main(sender_id, cmd, **extends):
    chat.send_message(sender_id, 'Enter your mail')
    query.set_action(sender_id, '/get_mail')

@ampalibe.action('/get_mail')
def get_mail(sender_id, cmd, **extends):
    # save the mail in temporary data
    query.set_temp(sender_id, 'mail', cmd)

    chat.send_message(sender_id, f'Enter your password')
    query.set_action(sender_id, '/get_password')


@ampalibe.action('/get_password')
def get_password(sender_id, cmd, **extends):
    query.set_action(sender_id, None)  # clear current action
    # get mail in temporary data
    mail = query.get_temp(sender_id, 'mail')
    chat.send_message(sender_id, f'your mail and your password are {mail} {cmd}')
    # delete mail in temporary data
    query.del_temp(sender_id, 'mail')
```

1.3.5 Payload Management

Ampalibe facilitates the management of payloads with the possibility of sending arguments.

You can send data with `Payload` object and get it in destination function's parameter

```
import ampalibe
# import the Payload class
from ampalibe import Payload
from conf import Configuration

bot = ampalibe.init(Configuration())
chat = bot.chat


@ampalibe.command('/')
```

(continues on next page)

(continued from previous page)

```
def main(sender_id, cmd, **extends):
    quick_rep = [
        {
            "content_type": "text",
            "title": 'Angela',
            #  Customise your payload
            "payload": Payload('/member', name='Angela', ref='2016-sac')
        },
        {
            "content_type": "text",
            "title": 'Rivo',
            #  Customise your payload
            "payload": Payload('/member', name='Rivo')
        }
    ]
    chat.send_quick_reply(sender_id, quick_rep, 'Who?')


@ampalibe.command('/member')
def get_membre(sender_id, cmd, name, **extends):
    '''
        You can receive the arguments payload in extends or
        specifying the name of the argument in the parameters
    '''
    chat.send_message(sender_id, "Hello " + name)

    # if the arg is not defined in the list of parameters,
    # it is put in the extends variable
    if extends.get('ref'):
        chat.send_message(sender_id, 'your ref is ' + extends.get('ref'))
```

1.3.6 File management

We recommand to make static file in assets folder,

for files you use as a URL file, you must put assets/public, in assets/private otherwise

```
'''
Suppose that a logo file is in "assets/public/iTeamS.png" and that we must send it via␣
↪url
'''

import ampalibe
from conf import Configuration

bot = ampalibe.init(Configuration())
chat = bot.chat


@ampalibe.command('/')
def main(sender_id, cmd, **extends):
```

(continues on next page)

(continued from previous page)

```
    '''
        to get a file in assets/public folder,
        the route is <adresse>/asset/<file>
    '''
    chat.send_file_url(
        sender_id,
        Configuration.APP_URL + '/asset/iTeamS.png',
        filetype='image'
    )
```

1.4 Messenger API

list of methods for sending messages to messenger

Note: All returns of these functions are a POST Requests <Response>

1.4.1 send_message

This method allows you to send a text message to the given recipient, Note that the number of characters to send is limited to 2000 characters

Ref: https://developers.facebook.com/docs/messenger-platform/send-messages#sending_text

Args:

dest_id (str): user id facebook for the destination

message (str): message want to send

Example:

```
chat.send_message(sender_id, "Hello world")
```

1.4.2 send_action

This method is used to simulate an action on messages. example: view, writing.

Action available: ['mark_seen', 'typing_on', 'typing_off']

Ref: https://developers.facebook.com/docs/messenger-platform/send-messages/sender-actions

Args:

dest_id (str): user id facebook for the destination

action (str): action ['mark_seen', 'typing_on', 'typing_off']

Example:

```
chat.send_action(sender_id, "mark_seen")
```

1.4.3 send_quick_reply

Quick replies provide a way to present a set of up to 13 buttons in-conversation that contain a title and optional image, and appear prominently above the composer.

You can also use quick replies to request a person's location, email address, and phone number.

Ref: https://developers.facebook.com/docs/messenger-platform/send-messages/quick-replies

Args:

dest_id (str): user id facebook for the destoination

quick_rep (list of dict): list of the different quick_reply to send a user

text (str): A text of a little description for each <quick_reply>

Example:

```
quick_rep = [
    {
        "content_type": "text",
        "title": 'Angela',
        "payload": '/membre',
        # image is optionnal
        "image_url": "https://i.imgflip.com/6b45bi.jpg"
    },
    {
        "content_type": "text",
        "title": 'Rivo',
        "payload": '/membre',
        # image is optionnal
        "image_url": "https://i.imgflip.com/6b45bi.jpg"
    }
]

chat.send_quick_reply(sender_id, quick_rep, 'who do you choose ?')
```

1.4.4 send_template

The method send_result represent a Message templates who offer a way for you to offer a richer in-conversation experience than standard text messages by integrating buttons, images, lists, and more alongside text a single message. Templates can be use for many purposes, such as displaying product information, asking the messagerecipient to choose from a pre-determined set of options, and showing search results.

For this, messenger only validates 10 templates for the first display, so we put the parameter <next> to manage these numbers if it is a number of elements more than 10. So, there is a quick_reply which acts as a "next page" displaying all requested templates

Ref: https://developers.facebook.com/docs/messenger-platform/send-messages/template/generic

Args:

dest_id (str): user id facebook for the destination

elements (list of dict): the list of the specific elements to define the structure for the template

quick_rep(list of dict): addition quick reply at the bottom of the template

next(bool): this params activate the next page when elements have a length more than 10

Example:

```
list_items = [
    {
        "title": "item n°1",
        "image_url": "https://i.imgflip.com/6b45bi.jpg",
        "buttons": [
            {
                "type": "postback",
                "title": "Get item",
                "payload": "/item 1"
            }
        ]
    },
    {
        "title": "item n°2",
        "image_url": "https://i.imgflip.com/6b45bi.jpg",
        "buttons": [
            {
                "type": "postback",
                "title": "Get item",
                "payload": "/item 2"
            }
        ]
    },

]

chat.send_template(sender_id, list_items)
```

```
list_items = [
    {
        "title": f"item n°{i+1}",
        "image_url": "https://i.imgflip.com/6b45bi.jpg",
        "buttons": [
            {
                "type": "postback",
                "title": "Get item",
                "payload": Payload("/item", id_item=i+1)
            }
        ]
    }
    for i in range(30)
]
# next=True for displaying directly next page button.
chat.send_template(sender_id, list_items, next=True)
```

1.4.5 send_file_url

The Messenger Platform allows you to attach assets to messages, including audio, video, images, and files.All this is the role of this Method. The maximum attachment size is 25 MB.

Args:

dest_id (str): user id facebook for destination

url (str): the origin url for the file

filetype (str, optional): type of showing file[“video”,”image”,”audio”,”file”]. Defaults to ‘file’.

Ref: https://developers.facebook.com/docs/messenger-platform/send-messages#url

Example:

```
chat.send_file_url(sender_id, 'https://i.imgflip.com/6b45bi.jpg', filetype='image')
```

1.4.6 send_file

This method send an attachment from file

Ref: https://developers.facebook.com/docs/messenger-platform/send-messages#file

Args:

dest_id (str): user id facebook for the destination

file (str): name of the file in local folder

filetype (str, optional): type of the file[“video”,”image”,...]. Defaults to “file”.

filename (str, optional): A filename received for de destination . Defaults to name of file in local.

Example:

```
chat.send_file(sender_id, "mydocument.pdf")

chat.send_file(sender_id, "intro.mp4", filetype='video')

chat.send_file(sender_id, "myvoice.m4a", filetype='audio')
```

1.4.7 send_media

Method that sends files media as image and video via facebook link. This model does not allow any external URLs, only those on Facebook.

Ref: https://developers.facebook.com/docs/messenger-platform/send-messages/template/media

Args:

dest_id (str): user id facebook for the destination

fb_url (str): url of the media to send on facebook

media_type (str): the type of the media who to want send, available[“image”,”video”]

Example:

```
chat.send_media(sender_id, "https://www.facebook.com/iTeam.Community/videos/
↪476926027465187", 'video')
```

1.4.8 send_button

The button template sends a text message with up to three buttons attached. This template gives the message recipient different options to choose from, such as predefined answers to questions or actions to take.

Ref: https://developers.facebook.com/docs/messenger-platform/send-messages/template/button

Args:

dest_id (str): user id facebook for the destination

buttons (list of dict): The list of buttons who want send

text (str): A text to describe the fonctionnality of the buttons

Example:

```
buttons = [
    {
        "type": "postback",
        "title": "Informations",
        "payload": '/contact'
    }
]
chat.send_button(sender_id, buttons, "What do you want to do?")
```

1.4.9 get_started

Method that GET STARTED button when the user talk first to the bot.

Ref: https://developers.facebook.com/docs/messenger-platform/reference/messenger-profile-api/get-started-button

Args:

dest_id (str): user id facebook for the destination

payload (str): payload of get started, default: '/'

Example:

```
chat.get_started()
```

1.4.10 persistent_menu

The Persistent Menu disabling the composer best practices allows you to have an always-on user interface element inside Messenger conversations. This is an easy way to help people discover and access the core functionality of your Messenger bot at any point in the conversation

Ref: https://developers.facebook.com/docs/messenger-platform/send-messages/persistent-menu

Args:

dest_id (str): user id for destination

persistent_menu (list of dict): the elements of the persistent menu to enable

action (str, optional): the action for benefit[“PUT”,”DELETE”]. Defaults to ‘PUT’.

locale [optionnel]

composer_input_disabled [optionnel]

Example:

```
persistent_menu = [
    {
        "type": "postback",
        "title": "Menu",
        "payload": "/menu"
    },
    {
        "type": "postback",
        "title": "Logout",
        "payload": "/logout"
    }
]

chat.persistent_menu(sender_id, persistent_menu)
```

1.5 Supported Database

The advantage of using Ampalibe’s supported databases is that we can inherit the Model object of ampalibe to make a database request.

We no longer need to make the connection to the database.

we use the instances received from the *Model* object as the variable **db**, **cursor**,

Example

`model.py` file

```
from ampalibe import Model

class Database(Model):
    def __init__(self, conf):
        super().__init__(conf)

    @Model.verif_db
    def get_list_users(self):
        '''
            CREATE CUSTOM method to communicate with database
        '''
        req = "SELECT * from amp_user"
        self.cursor.execute(req)
        data = self.cursor.fetchall()
        self.db.commit()
        return data
```

core.py file

```
import ampalibe
from conf import Configuration
from model import Database

req = Database(Configuration())


bot = ampalibe.init(Configuration())
chat = bot.chat


@ampalibe.command('/')
def main(sender_id, cmd, **extends):
   print(req.get_list_users())
```

Important: *Model.verif_db* is a decorator who checks that the application is well connected to the database before launching the request.

Note: However we can still use other databases than those currently supported

1.6 Logging and structure

1.6.1 Logging

By default, all error in Messenger API methods is showing in output

Messenger API methods response is a request <Response> so we can use it to view the response

```
res = chat.send_message(sender_id, "Hello Ampalibe")

if res.status_code == 200:
    print("OK! NO problem")
else:
    print('KO')
    print(res.text, res.status_code)
```

1.6.2 Structure

Each developer is **free to choose the structure he wants**, by just importing the files into the core.py.

We can make our functions everywhere, even as methods

core.py file

```
# importing another file contains ampalibe decorator
import user
```

(continues on next page)

(continued from previous page)

```
import ampalibe
from conf import Configuration
from ampalibe import Payload

bot = ampalibe.init(Configuration())
chat = bot.chat


@ampalibe.command('/')
def main(sender_id, cmd, **extends):
    buttons = [
        {
            "type": "postback",
            "title": "Dashboard",
            "payload": '/login/admin'
        }
    ]
    chat.send_button(sender_id, buttons, 'What do you want to do?')


class Admin:

    @ampalibe.command('/login/admin')
    def login(sender_id, **extends):
        '''
            function is always calling when payload or message start by /login/admin
        '''
        bot.query.set_action(sender_id, '/get_username')
        bot.chat.send_message(sender_id, 'Enter your username')
```

user.py file

```
import ampalibe
from conf import Configuration
bot = ampalibe.init(Configuration())
chat = bot.chat


class User:

    @ampalibe.action('/get_username')
    def username(sender_id, cmd, **extends):
        bot.chat.send_message(sender_id, 'OK ' + cmd)
        bot.query.set_action(sender_id, None)
```

Note: if you want use a MVC Pattern

here is an example of an MVC template that can be used: Ampalibe MVC Template

1.7 Custom endpoint

The web server part and the endpoints are managed directly by Ampalibe

However, a custom end point can be created using the FastAPI object instance

```
import ampalibe
from ampalibe import webserver
from conf import Configuration

bot = ampalibe.init(Configuration())


@webserver.get('/test')
def test():
    return 'Hello, test'

@ampalibe.command('/')
def main(sender_id, cmd, **extends):
    bot.chat.send_message(sender_id, "Hello, Ampalibe")
```

1.8 API

ampalibe.command(*args, **kwargs)	A decorator that registers the function as the route
ampalibe.action(*args, **kwargs)	A decorator that registers the function as the route
ampalibe.Payload(payload, **kwargs)	Object for Payload Management
ampalibe.Model(conf)	Object for interact with database with pre-defined function

1.8.1 ampalibe.command

ampalibe.**command**(**args*, ***kwargs*)

A decorator that registers the function as the route of a processing per command sent.

1.8.2 ampalibe.action

ampalibe.**action**(**args*, ***kwargs*)

A decorator that registers the function as the route of a defined action handler.

1.8.3 ampalibe.Payload

class ampalibe.**Payload**(*payload*, ***kwargs*)

Object for Payload Management

__init__(*payload*, ***kwargs*) → None

Object for Payload Management

Methods

__init__(payload, **kwargs)	Object for Payload Management
trt_payload_in(payload)	processing of payloads received in a sequence of structured parameters
trt_payload_out(payload)	Processing of a Payload type as a character string

1.8.4 ampalibe.Model

class ampalibe.**Model**(*conf*)

Object for interact with database with pre-defined function

__init__(*conf*)

object to interact with database

@params: conf [Configuration object] @return: Request object

Methods

__init__(conf)	object to interact with database
del_temp(**kwarg)	
get_action(**kwarg)	
get_temp(**kwarg)	
set_action(**kwarg)	
set_temp(**kwarg)	
verif_db()	decorator that checks if the database is connected or not before doing an operation.

INDEX

Symbols

A

C

M

P