How I wrote my first telegram bot server using simple getUpdates?

Webhook:

This method requires a registered public domain name to listen for messages from Telegram bot so I kept it for later once I have my personal basic bot.

Polling(getUpdates):

This method doesn’t have any pre-requirements. So I started right away

Create a bot:

If new to telegram open an account using mobile number and create a bot by following <https://core.telegram.org/bots#3-how-do-i-create-a-bot>

(Optional) Install Desktop app and do a web login to your account to avoid switching between dev setup and phone

Run backend server:

After immense google search came up with tbot (github.com/yanzay/tbot) which is a good golang library for both Webhook and Polling with wrapper functions over telegram APIs and structs.

But it was a bit high level(that’s what libraries are for!) where the basic http GET, POST queries are hidden within good wrappers which makes our work much easier. To learn on what really is happening behind the scenes I started with mkdir and a main.go file with the telegram API page in side <https://core.telegram.org/bots/api>

Steps to copy any version code and run:

Set your bot token as environment variable TELEGRAM\_TOKEN and run the code

$ export TELEGRAM\_TOKEN=<token>

$ go run main.go

Basic Version:

The following is the first version of my main.go which does a http GET query on the telegram API “getUpdates”.

package main

import (

"io/ioutil"

"log"

"net/http"

"os"

)

func main() {

token := os.Getenv("TELEGRAM\_TOKEN")

if token == "" {

log.Fatal("Not a valid token")

}

query := "https://api.telegram.org/bot" + token + "/getUpdates"

resp, err := http.Get(query)

if err != nil {

log.Fatal("Error during Get: " + err.Error())

return

}

body, err := ioutil.ReadAll(resp.Body)

err = resp.Body.Close()

if err != nil {

log.Fatal("Error during response body close: " + err.Error())

}

log.Print("Response: ", string(body))

}

OUTPUT:

RATHEGS-M-C3XA:src rathegs$ go run main.go

2020/05/02 17:01:10 Response: {"ok":true,"result":[{"update\_id":332016300,

"message":{"message\_id":54,"from":{"id":1060392138,"is\_bot":false,"first\_name":"Rathega","language\_code":"en"},"chat":{"id":1060392138,"first\_name":"Rathega","type":"private"},"date":1588406550,"text":"yo"}},{"update\_id":332016301,

"message":{"message\_id":55,"from":{"id":1060392138,"is\_bot":false,"first\_name":"Rathega","language\_code":"en"},"chat":{"id":1060392138,"first\_name":"Rathega","type":"private"},"date":1588406550,"text":"hi"}}]}

Dumb Reply Version:

Once I saw the text I entered I am now curious to reply to this message and see it in my bot. This can be achived using “sendMessage” that takes two parameters as mandatory chat\_id, text.

chat\_id - should be taken from getUpdates response

text - any string we want to send as reply

To parse the response I got from previous GET I created struct for the visible fields by keeping  [getUpdates](https://core.telegram.org/bots/api#getupdates) and [Update](https://core.telegram.org/bots/api#update) object as reference and came up with this dumb reply version.

package main

import (

"encoding/json"

"io/ioutil"

"log"

"net/http"

"net/url"

"os"

"strconv"

)

type Response struct {

OK bool `json:"ok"`

Result []Update `json:"result"`

}

type Update struct {

UpdateID int `json:"update\_id"`

Message Message `json:"message"`

}

type Message struct {

MessageID int `json:"message\_id"`

From User `json:"from"`

Chat Chat `json:"Chat"`

Date int `json:"date"`

Text string `json:"text"`

}

type User struct {

ID int `json:"id"`

IsBot bool `json:"is\_bot"`

FirstName string `json:"first\_name"`

LanguageCode string `json:"language\_code"`

}

type Chat struct {

ID int `json:"id"`

FirstName string `json:"first\_name"`

Type string `json:"type"`

}

func main() {

token := os.Getenv("TELEGRAM\_TOKEN")

if token == "" {

log.Fatal("Not a valid token")

}

// getUpdates

query := "https://api.telegram.org/bot" + token + "/getUpdates"

resp, err := http.Get(query)

if err != nil {

log.Fatal("Error during getUpdates: " + err.Error())

return

}

body, err := ioutil.ReadAll(resp.Body)

err = resp.Body.Close()

if err != nil {

log.Fatal("Error during response body close: " + err.Error())

}

log.Print("Response: ", string(body))

data := Response{}

err = json.Unmarshal(body, &data)

if err != nil {

log.Fatal("Error during response unmarshal: " + err.Error())

}

log.Print("Response struct: ", data)

// sendMessage

reply := "hello"

query = "https://api.telegram.org/bot" + token + "/sendMessage"

for \_, update := range data.Result {

chatID := update.Message.Chat.ID

vals := url.Values{}

vals.Set("chat\_id", strconv.Itoa(chatID))

vals.Set("text", reply)

resp, err = http.PostForm(query, vals)

log.Print("Sending reply as ", reply, " to chatID ", chatID)

if err != nil {

log.Fatal("Error during sendMessage: " + err.Error())

}

}

}

OUTPUT:

RATHEGS-M-C3XA:src rathegs$ go run main.go

2020/05/02 17:36:42 Response: {"ok":true,"result":[{"update\_id":332016300,

"message":{"message\_id":54,"from":{"id":1060392138,"is\_bot":false,"first\_name":"Rathega","language\_code":"en"},"chat":{"id":1060392138,"first\_name":"Rathega","type":"private"},"date":1588406550,"text":"yo"}},{"update\_id":332016301,

"message":{"message\_id":55,"from":{"id":1060392138,"is\_bot":false,"first\_name":"Rathega","language\_code":"en"},"chat":{"id":1060392138,"first\_name":"Rathega","type":"private"},"date":1588406550,"text":"hi"}}]}

2020/05/02 17:36:42 Response struct: {true [{332016300 {54 {1060392138 false Rathega en} {1060392138 Rathega private} 1588406550 yo}} {332016301 {55 {1060392138 false Rathega en} {1060392138 Rathega private} 1588406550 hi}}]}

2020/05/02 17:36:42 Sending reply as hello to chatID 1060392138

2020/05/02 17:36:43 Sending reply as hello to chatID 1060392138

The above program will send one *hello* to each of the text that was sent to our bot before.

A screenshot of a cell phone

Description automatically generated

Smart Reply Version:

Instead of sending *hello* to each of the message I want to greet persons who say only *hi* to bot. For all others I wanted to reply as *Say hi* message.

The same APIs are used as before with two changes

* Before sending reply I added a little check on the text content
* I am setting reply\_to\_message\_id with the message\_id field from getUpdates response so that on bot each of our response will appear as reply to user’s message.

package main

import (

"encoding/json"

"io/ioutil"

"log"

"net/http"

"net/url"

"os"

"strconv"

"strings"

)

type Response struct {

OK bool `json:"ok"`

Result []Update `json:"result"`

}

type Update struct {

UpdateID int `json:"update\_id"`

Message Message `json:"message"`

}

type Message struct {

MessageID int `json:"message\_id"`

From User `json:"from"`

Chat Chat `json:"Chat"`

Date int `json:"date"`

Text string `json:"text"`

}

type User struct {

ID int `json:"id"`

IsBot bool `json:"is\_bot"`

FirstName string `json:"first\_name"`

LanguageCode string `json:"language\_code"`

}

type Chat struct {

ID int `json:"id"`

FirstName string `json:"first\_name"`

Type string `json:"type"`

}

func main() {

token := os.Getenv("TELEGRAM\_TOKEN")

if token == "" {

log.Fatal("Not a valid token")

}

// getUpdates

query := "https://api.telegram.org/bot" + token + "/getUpdates"

resp, err := http.Get(query)

if err != nil {

log.Fatal("Error during getUpdates: " + err.Error())

return

}

body, err := ioutil.ReadAll(resp.Body)

err = resp.Body.Close()

if err != nil {

log.Fatal("Error during response body close: " + err.Error())

}

log.Print("Response: ", string(body))

data := Response{}

err = json.Unmarshal(body, &data)

if err != nil {

log.Fatal("Error during response unmarshal: " + err.Error())

}

log.Print("Response struct: ", data)

// sendMessage

reply := ""

query = "https://api.telegram.org/bot" + token + "/sendMessage"

for \_, update := range data.Result {

chatID := update.Message.Chat.ID

msgID := update.Message.MessageID // enhancement2

inpMsg := strings.ToLower(update.Message.Text)

if strings.Contains(inpMsg, "hi") { // enhancement1

reply = "hello"

} else {

reply = "say hi"

}

vals := url.Values{}

vals.Set("chat\_id", strconv.Itoa(chatID))

vals.Set("text", reply)

vals.Set("reply\_to\_message\_id", strconv.Itoa(msgID)) // enhancement2

resp, err = http.PostForm(query, vals)

log.Print("Sending reply as ", reply, " to chatID ", chatID, "msgID ", msgID)

if err != nil {

log.Fatal("Error during sendMessage: " + err.Error())

}

}

}

OUTPUT:

RATHEGS-M-C3XA:src rathegs$ go run main.go

2020/05/02 18:00:36 Response: {"ok":true,"result":[{"update\_id":332016300,

"message":{"message\_id":54,"from":{"id":1060392138,"is\_bot":false,"first\_name":"Rathega","language\_code":"en"},"chat":{"id":1060392138,"first\_name":"Rathega","type":"private"},"date":1588406550,"text":"yo"}},{"update\_id":332016301,

"message":{"message\_id":55,"from":{"id":1060392138,"is\_bot":false,"first\_name":"Rathega","language\_code":"en"},"chat":{"id":1060392138,"first\_name":"Rathega","type":"private"},"date":1588406550,"text":"hi"}}]}

2020/05/02 18:00:36 Response struct: {true [{332016300 {54 {1060392138 false Rathega en} {1060392138 Rathega private} 1588406550 yo}} {332016301 {55 {1060392138 false Rathega en} {1060392138 Rathega private} 1588406550 hi}}]}

2020/05/02 18:00:37 Sending reply as say hi to chatID 1060392138msgID 54

2020/05/02 18:00:38 Sending reply as hello to chatID 1060392138msgID 55

The above program sends *hello* only when *hi* is sent or will reply as *say hi*

A screenshot of a cell phone screen with text

Description automatically generated

Notes:

* This program needs to be structured and updates from getUpdates should be processed via channels to maintain ordering and should be put in an infinite for loop for continuously processing user requests.
* Also while doing GET on getUpdates please take care of setting update\_id field by taking reference from manual otherwise in infinite for loop we will end up processing same messages infinite times.

This basic experiment helped me on how to read the telegram bot API manual and implement the required functionalities. Hope it helps someone!