

DEDAN KIMATHI UNIVERSITY OF TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

A PROJECT PROPOSAL FOR THE FINAL YEAR STUDENT IN BSc BUSINESS INFORMATION TECHNOLOGY

PROJECT TITLE: COVIDLY

SUBMITTED BY

NAME: OWINO MIKE OKUMU

REG.NO: C027-01-1110/2017

SUPERVISOR

MR. MICHAEL KAGIRI

A project submitted in partial fulfillment of the requirements for the degree in Business

Information Technology to the Department of Computer Science and

Information Technology.

# DECLARATION

I hereby declare that the project entitled COVIDLY submitted for the B.Sc. Business Information Technology degree is my original work and the project has not formed the basis for an award of any degree, diploma or any other similar titles.

Name: OWINO MIKE OKUMU

Signature ………………………………..

Date:

This project has been submitted for examination with my approval as a University Supervisor. Name MR. MICHAEL KAGIRI.

Signature ………………………………

Date:

# DEDICATION

I dedicate this project to my father, Alfred Okumu for his financial support and constant encouragement throughout my education and paving for me to do this course. I also dedicate it to my family members who has been of constant help during my way Degree program. God bless them abundantly.

# ACKNOLEDGEMENT

I would like to express my very great appreciation to Mr. Kagiri Michael for his valuable and constructive guidance during the entire development of this research work. His willingness to give his time so generously is much appreciated.

I would also like to express my gratitude to Mrs. Kuria Jane for her advice and assistance in moderating my research proposal.

I would like to acknowledge the role played by the department of Information Technology as well as all my lecturers in the B.Sc. Business Information Technology degree program.

I am particularly grateful to Mrs. Ndunge Elizabeth who tutored Research Methods, a subject that did helped a lot in this research.

I also acknowledge the help from all my classmates, specifically Nicholas, Denis, Jennifer among others. Finally, I wish to thank my family and friends for their support and encouragement throughout my study. God bless.

# ABSTRACT

In early December 2019, an outbreak of coronavirus disease 2019 (COVID-19)(Organization & others, 2020), caused by a novel severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2).

It occurred in Wuhan City, Hubei Province, China. On January 30, 2020, the World Health Organization declared the outbreak as a Public Health Emergency of International Concern. As of February 14, 2020, 49,053 laboratory-confirmed and 1,381(Worldometers, 2020) deaths have been reported globally.

Perceived risk of acquiring disease has led to many governments to institute a variety of control measures of reducing the spread of the virus and ways of ensuring that their citizens are well equipped with the required information about the virus.

The project aims to reduce the burden on the head of going to a website in search of the covid cases by developing a convincing Chabot where that information can be easily accessed.

A suitable algorithm is devised to search through the set of data and find a potential number. The program then replies to the user and provides a relevant web link if after the user is sent the reply.

Table of Contents

[DECLARATION ii](#_Toc46852048)

[DEDICATION ii](#_Toc46852049)

[ACKNOLEDGEMENT v](#_Toc46852050)

[ABSTRACT vi](#_Toc46852051)

[List of figures ix](#_Toc46852052)

[List of Tables x](#_Toc46852053)

[ABREVIATIONS xi](#_Toc46852054)

[CHAPTER 1: INTRODUCTION 1](#_Toc46852055)

[1.1 BACKGROUND 1](#_Toc46852056)

[1.2 PROBLEM STATEMENT 2](#_Toc46852057)

[1.3 RESEARCH OBJECTIVES 3](#_Toc46852058)

[1.3.1 General Objectives 3](#_Toc46852059)

[1.3.2 Specific Objectives 3](#_Toc46852060)

[1.3.3 Justification of the Study 3](#_Toc46852061)

[1.3.4 Scope of the Study 4](#_Toc46852062)

[1.3.5 Study Limitation 4](#_Toc46852063)

[CHAPTER 2: LITERATURE REVIEW 5](#_Toc46852064)

[2.1 Case Studies 5](#_Toc46852065)

[2.1.1 CASE 1 5](#_Toc46852066)

[2.1.2 CASE 2 6](#_Toc46852067)

[2.1.3 CASE 3 7](#_Toc46852068)

[2.1.4 CASE 4 8](#_Toc46852069)

[CHAPTER 3: METHODOLOGY 9](#_Toc46852070)

[3.1 INTRODUCTION 9](#_Toc46852071)

[3.2 SYSTEM DEVELOPMENT LIFE CYCLE 9](#_Toc46852072)

[The purpose of waterfall method is that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one. 9](#_Toc46852073)

[3.2.1 REQUIREMENTS 10](#_Toc46852074)

[3.2.2 RESEARCH DESIGN 11](#_Toc46852075)

[3.2.3 IMPLEMENTATION 11](#_Toc46852076)

[3.2.4 TESTING 11](#_Toc46852077)

[3.2.5 DEPLOYMENT 12](#_Toc46852078)

[3.2.6 MAINTENANCE 12](#_Toc46852079)

[3.3 TOTAL POPULATION 12](#_Toc46852080)

[3.4 SAMPLING PROCEDURE 12](#_Toc46852081)

[3.4.1 Sample size 12](#_Toc46852082)

[3.4.2 Sampling Design 13](#_Toc46852083)

[3.5 DATA COLLECTION METHOD 13](#_Toc46852084)

[3.5.1 Documents and Records 13](#_Toc46852085)

[3.5.2 Internet Research 13](#_Toc46852086)

[3.6 DATA ANALYSIS 13](#_Toc46852087)

[3.6.1 Test for Covidly functionality 13](#_Toc46852088)

[CHAPTER FOUR: DESIGN AND ANALYSIS 14](#_Toc46852089)

[4.1 INTRODUCTION 14](#_Toc46852090)

[4.2 REQUIREMENT ANALYSIS 14](#_Toc46852091)

[4.2.1 Functional Requirements 14](#_Toc46852092)

[4.2.2 Non-Functional Requirements 14](#_Toc46852093)

[4.3 DATA PROCESSING AND ANALYSIS 15](#_Toc46852094)

[4.3.1 Questionnaire 15](#_Toc46852095)

[4.3.2 Charts and Graphs 16](#_Toc46852096)

[4.4 SYSTEM ANALYSIS 19](#_Toc46852097)

[4.4.1 INTRODUCTION 19](#_Toc46852098)

[4.4.2 Sequence Diagram 19](#_Toc46852099)

[4.4.3 Use Case Diagram 20](#_Toc46852100)

[CHAPTER FIVE: DISCUSSION, CONCLUSION AND RECOMMENDATIONS 21](#_Toc46852101)

[5.1 INTRODUCTION 21](#_Toc46852102)

[5.2 DISCUSSIONS 21](#_Toc46852103)

[5.3 CONCLUSIONS 23](#_Toc46852104)

[5.4 RECOMMENDATION 23](#_Toc46852105)

[5.5 LIMITATIONS OF THE STUDY 24](#_Toc46852106)

[References 25](#_Toc46852107)

[Appendix A 27](#_Toc46852108)

[A1 PYTHON LANGUAGE GRAMMER SPECIFICATIONS 27](#_Toc46852109)

[A2 JSON LANGUAGE GRAMMER SPECIFICATIONS 33](#_Toc46852110)

[Appendix B 35](#_Toc46852111)

[Sample source code and output listing 35](#_Toc46852112)

[B1 COMMANDS CODE FOR THE BOT AND REPLIES 35](#_Toc46852113)

[B2 CODE FOR ACCESSING THE COVID-19 API SERVERS 40](#_Toc46852114)

[B3 MAIN SOURCE CODE OF THE BOT INCLUDING ACCESS TO THE BOT API TOKEN AND NOTIFY TIME FOR SENDING THE DAILY NOTIFICATIONS 43](#_Toc46852115)

[B4 JSON CODE CONTAINING BOT TOKEN AND TIMER 56](#_Toc46852116)

## List of figures

[Figure 1 infected countries on a logarithmic scale.(AaronWard, 2020) 6](#_Toc44251349)

[Figure 2 Covid-19 visualization(localeai, 2020) 7](#_Toc44251350)

[Figure 3 Corona Virus Tracker Swift UI(alfianlosari, 2020) 8](#_Toc44251351)

[Figure 4 Deaths Per Million of Inhabitants(github, 2020) 9](#_Toc44251352)

[Figure 5 Waterfall model(creately.com, 2020) 11](#_Toc44251353)

## List of Tables

[Table 1 Resources Table 17](#_Toc44437106)

[Table 2 Budget Table 17](#_Toc44437107)

[Table 3 GANTT CHART 18](#_Toc44437108)

## ABREVIATIONS

# CHAPTER 1: INTRODUCTION

## 1.1 BACKGROUND

Since the outbreak of the corona virus there has been a need to have systems that will enable citizens of a country to have readily available information to make them get required information of what is going on.

This is after the famous Spanish flu(B. J. Jester et al., 2018) that hit the world 100 years ago(B. Jester et al., 2019) and people were not prepared enough for it despite technological inventions but as per the current technological inventions that we are now there is need to have some applications that allow people to get the required information to minimize the spread of the virus.

The purpose of the project is to build a telegram-based bot that feeds the subscriber information based on the latest count of corona virus that is current statistics of local total confirmed, deaths, and recovered cases.

The bot also provides advice from the government and the required rules to be followed provided by the government. The research findings help the Kenyan citizens to be aware and equipped with the knowledge of how to handle the virus situation and follow the regulations laid out by the government and also get track of the statistics in the country.

## PROBLEM STATEMENT

According to the way the information about corona virus is being passed out via the television, radio, twitter(MOH\_Kenya, 2020) and websites(Worldometers, 2020) there has been an ignorance of the information being conveyed thus seen the increase in the number of the corona virus related cases due to ignorance despite the information given.

The increase in the number of cases has been attributed to the ignorance of the information being conveyed by the government which has led to many people succumbing to the virus.

The aim of this project is to come up with an alternative method of information circulation to the people of Kenya to get timely the information provided by the government and follow up direct from their mobile phone (Telegram(telegram.org, 2020)).

## RESEARCH OBJECTIVES

### General Objectives

The main objective of this study is to contribute to the solution of updating people of the corona virus cases by developing a telegram bot that shows the daily count of corona virus cases in Kenya and the global count of the virus.

### Specific Objectives

1. To develop an information resource were all the relevant information about the questions and the answers will be stored.
2. To provide a summarized format of the covid-19 cases such as the deaths, infected and recovered cases.
3. To develop a bot interface which aims to give the ability to potential users and their friends to request data of covid-19 related cases in the Chatbot and get convincing replies.

### Justification of the Study

The significance of the study should is to provide a channel for transmission of information about the corona virus cases directly from the government. It is important as the bot users will be able to get first-hand information from the government directly from their chat application saving their time of going to the website or other sources.

### Scope of the Study

The study focuses on the telegram users and all other people in Kenya who are capable of joining telegram to make use of its open source API the encamp uses of running a bot

### 1.3.5 Study Limitation

The bot does not provide previous data of the covid-19 related cases such data of the previous months or days.

# CHAPTER 2: LITERATURE REVIEW

## 2.1 Case Studies

### 2.1.1 CASE 1

**Covidify**

Corona virus report and dataset generator for python(AaronWard, 2020) that creates a time series dataset, Creates a daily stats dataset, forecast global and country confirmed cases, generate visualizations ,filter by country, filter by country, shows number of people currently infected and generates an excel report including all of the above.

An ARIMA model is trained and used to forecast the cumulative cases for N number of days into the future.

The following figure gives an example of Covidify displaying covid-19 data.

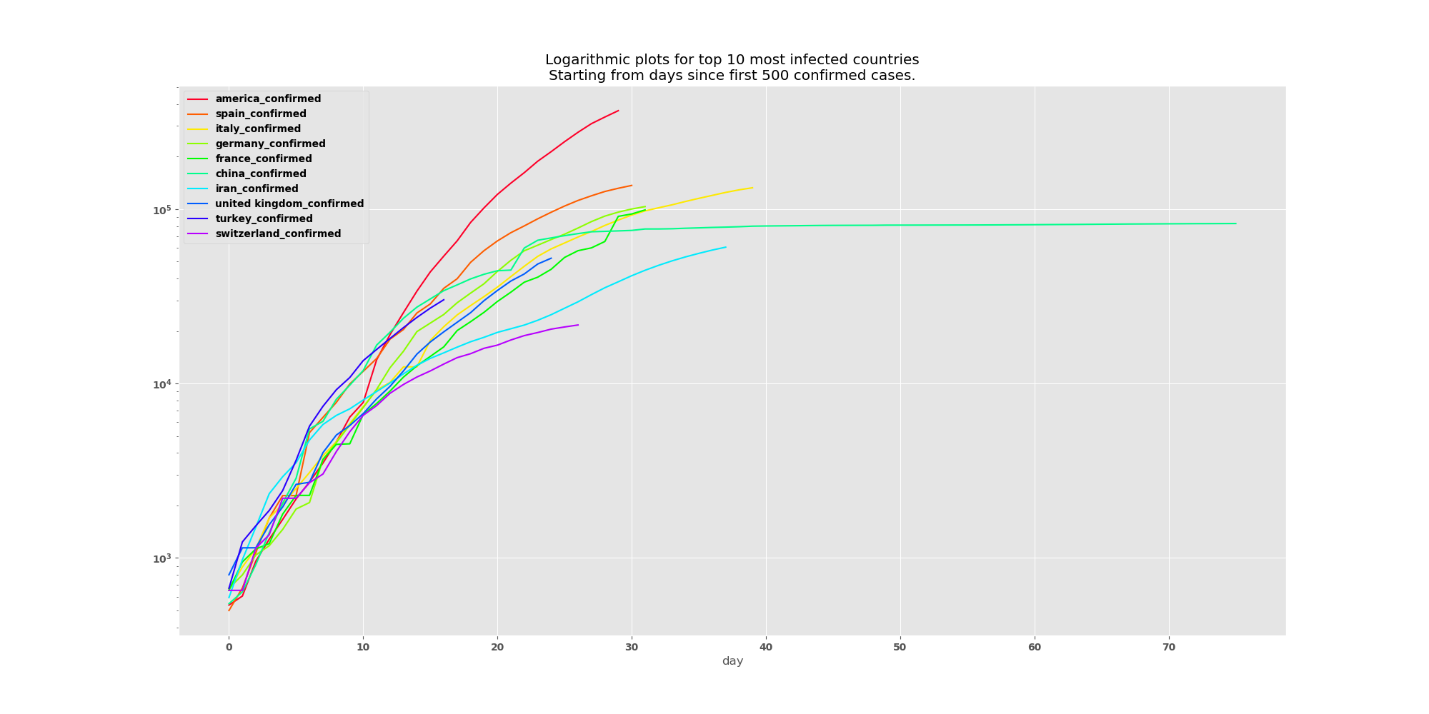


Figure 1 infected countries on a logarithmic scale.(AaronWard, 2020)

### 2.1.2 CASE 2

Covid-19 live visualization

API project which gives the layers data to the webapp(localeai, 2020). The webapp currently supports the layers that are picked up from periodically generated file from GitHub repository.

The GitHub api url for the repo holds the layers data. It is used to pick up the last updated date from the latest commit to the repo. It uses the GitHub API, along with the static repository URL.

The following figure gives an example of Covid-19 live visualization displaying covid-19 data.

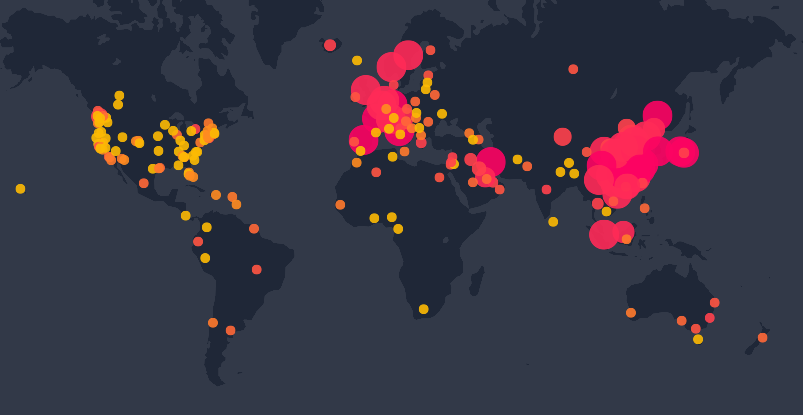


Figure 2 Covid-19 visualization(localeai, 2020)

### 2.1.3 CASE 3

Corona Virus Tracker Swift UI

The project gets current statistics of global total confirmed, deaths, recovered cases, statistics of countries and regions total cases count World map with annotation pin and description of affected regions Datasets is provided by ArcGIS Esri Corona Virus dataset(alfianlosari, 2020). Basic advice to prevent and handle virus, myth busters Q&A from WHO.

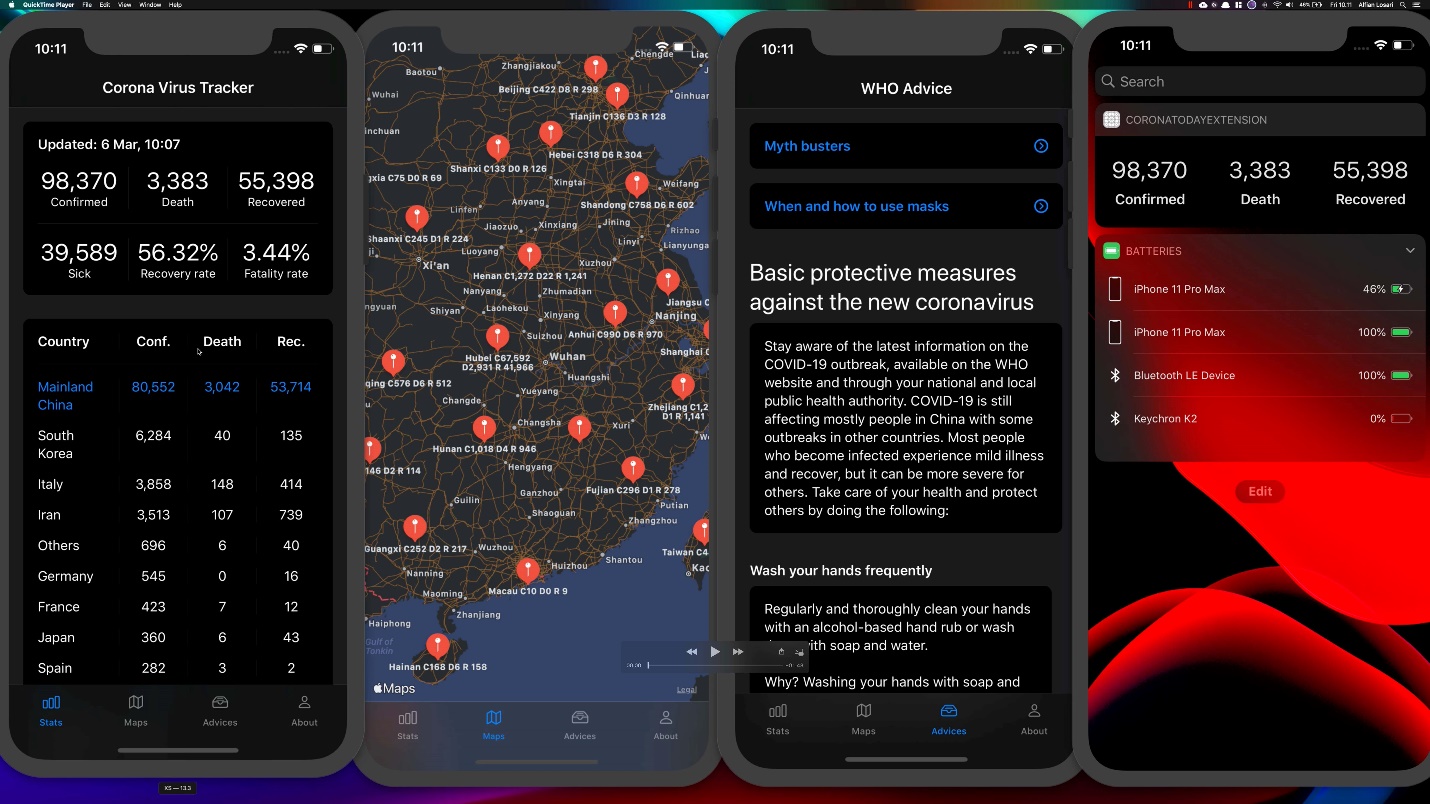


Figure 3 Corona Virus Tracker Swift UI(alfianlosari, 2020)

### 2.1.4 CASE 4

**Covid19-Dashboard**

This project was supported by a virtual team of open source collaborators that gather data, make visualizations and perform analysis regarding the COVID-19 epidemic(github, 2020).

We saw that data scientists were struggling to share information with each other, so we used this project to aid in the sharing of data and information.

This project aims to offer an accessible way for data professionals to share dashboards (that are updated with new data automatically) without requiring any expertise in front end development.

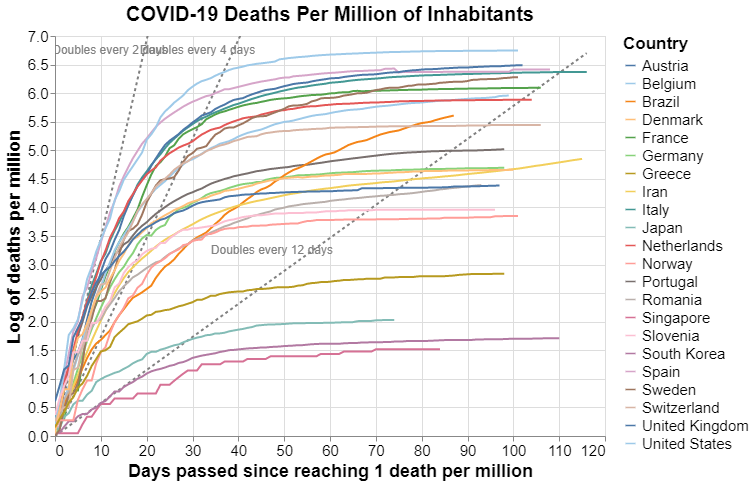


Figure 4 Deaths Per Million of Inhabitants(github, 2020)

# CHAPTER 3: METHODOLOGY

## 3.1 INTRODUCTION

This chapter deals with the research method that will be adopted and the analysis of the data collected. This study entails a descriptive survey design; the population that was the Kenya’s and the world population, sample design, data collection and analysis.

## 3.2 SYSTEM DEVELOPMENT LIFE CYCLE

The design and development of the bot will be followed by the waterfall model.

### The purpose of waterfall method is that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

It is easy to manage due to the rigidity of the model as each phase has specific deliverables and a review process and phases are processed and completed one at a time and they do not overlap. The waterfall model also works well for smaller projects like mine where requirements are very well understood.

The sample waterfall model is as shown in the diagram below.

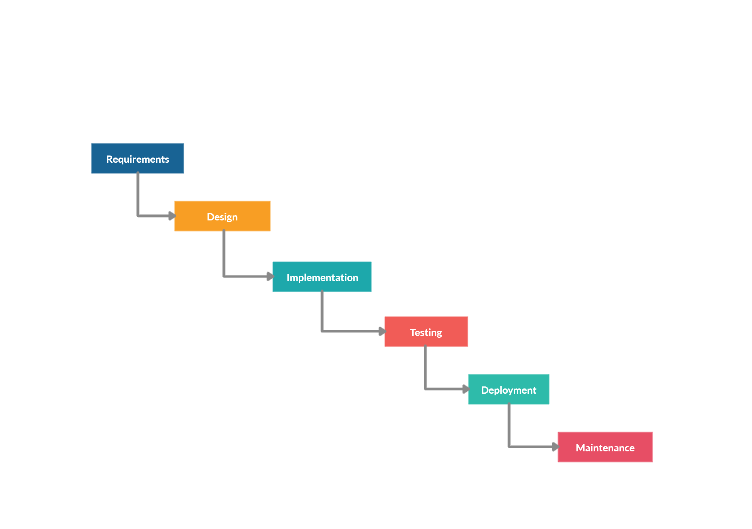


Figure 5 Waterfall model(creately.com, 2020)

### 3.2.1 REQUIREMENTS

In this phase I will focus on possible requirements of the system for the development. Requirements to be gathered subsequent to the end user consultation includes Google Cloud platform(cloud.google.com, 2020) , Heroku(heroku.com, 2020) ,Telegram generated bot with a working API , Internet connection ,personal computer , mobile phone , Visual Studio Code(code.visualstudio.com, 2020) ,Git(git-scm.com, n.d.) client and python environment installed on the computer.

### 3.2.2 RESEARCH DESIGN

Prior to beginning the actual coding, it is inevitable to understand what actions are to be taken and what they should like. The requirement specifications will be studied in detail in this phase and the design of the bot is prepared. The design specifications will be the base for the implementation and unit testing model phase.

Visual Studio Code will be used in the designing of the bot together with the bot to get the layout of the buttons to be used by the users and the texts to be displayed on the press of the button.

### IMPLEMENTATION

After receiving the system design documents, the work will be shared into various modules and the real coding is commenced by use of Visual Studio Code. The system will be developed into small coding files such as the API file and the layout file. These units will later be integrated in the subsequent phase. Every unit will be tested for its functionality to meet the bot functionality.

### 3.2.4 TESTING

The modules that will be divided into units will be integrated into a complete system and tested for proper coordination among modules and system behaves as per the specifications. The bot will be tested by being deployed to Google Cloud for its integration with the telegram bot API to start operational testing.

### 3.2.5 DEPLOYMENT

Once the testing is completed and the bot is found to meet the user requirements it will be deployed to Heroku via the git client as Heroku only supports deployment by use of git.Postress(data.heroku.com, 2020) will be used to connect the bot with the database accessible(NovelCOVID REST API (NovelCOVID, 2020)) of the covid-19 cases the to the bots API and displayed to the users.

### 3.2.6 MAINTENANCE

It is a never-ending phase. Once the system is running in production environment, problems may come up such as change of the database access URL. The issues that are related to the bot will be solved only after deployment of the system. The problems may arise from time to time and need to be solved.

## 3.3 TOTAL POPULATION

The total population to be considered it the world population which has already been counted for use by *Worldometer* that will be further broken down into Kenyan population and the top most affected countries in the world to see how other countries are doing with the corona virus.

## 3.4 SAMPLING PROCEDURE

### 3.4.1 Sample size

Kenyan population and the topmost affected countries in the world to see how other countries are doing with the corona virus.

### Sampling Design

Out of the world population data available in the API website ill sample Kenyan data to deal with and narrow down to the infected, deaths and recovered persons.

## 3.5 DATA COLLECTION METHOD

### 3.5.1 Documents and Records

I will examine data from documents containing relevant covid-19 cases such as journals and records containing covid-19 cases such as data of ArcGIS from GitHub spreadsheets

### 3.5.2 Internet Research

The Internet will be a great source of information and material, majority of the research to be carried out will be done here to avoid reinventing the wheel.

## DATA ANALYSIS

### 3.6.1 Test for Covidly functionality

# CHAPTER FOUR: DESIGN AND ANALYSIS

## 4.1 INTRODUCTION

## 4.2 REQUIREMENT ANALYSIS

### 4.2.1 Functional Requirements

### 4.2.2 Non-Functional Requirements

## 4.3 DATA PROCESSING AND ANALYSIS

### 4.3.1 Questionnaire

### 4.3.2 Charts and Graphs

# 4.4 SYSTEM ANALYSIS

### 4.4.1 INTRODUCTION

### 4.4.2 Sequence Diagram

### 4.4.3 Use Case Diagram

# CHAPTER FIVE: DISCUSSION, CONCLUSION AND RECOMMENDATIONS

## 5.1 INTRODUCTION

This chapter presents a summary of…

## 5.2 DISCUSSIONS

## 5.3 CONCLUSIONS

## 5.4 RECOMMENDATION

## 5.5 LIMITATIONS OF THE STUDY

# References

AaronWard. (2020). *covidify.*

alfianlosari. (2020). *CoronaVirusTrackerSwiftUI.*

cloud.google.com. (2020). *cloud.google.com.*

code.visualstudio.com. (2020). *code.visualstudio.com.*

creately.com. (2020). *creately.com.*

data.heroku.com. (2020). *data.heroku.com.*

de Groot, R. J., Baker, S. C., Baric, R. S., Brown, C. S., Drosten, C., Enjuanes, L., Fouchier, R. A. M., Galiano, M., Gorbalenya, A. E., Memish, Z. A., Perlman, S., Poon, L. L. M., Snijder, E. J., Stephens, G. M., Woo, P. C. Y., Zaki, A. M., Zambon, M., & Ziebuhr, J. (2013). Middle East Respiratory Syndrome Coronavirus (MERS-CoV): Announcement of the Coronavirus Study Group. *Journal of Virology*, *87*(14), 7790–7792. https://doi.org/10.1128/jvi.01244-13

git-scm.com. (n.d.). *git-scm.com.*

github. (2020). *covid19-dashboard.*

heroku.com. (2020). *heroku.com.*

Jester, B. J., Uyeki, T. M., Patel, A., Koonin, L., & Jernigan, D. B. (2018). 100 Years of Medical Countermeasures and Pandemic Influenza Preparedness. *American Journal of Public Health*, *108*(11), 1469–1472. https://doi.org/10.2105/AJPH.2018.304586

Jester, B., Uyeki, T. M., Jernigan, D. B., & Tumpey, T. M. (2019). Historical and clinical aspects of the 1918 H1N1 pandemic in the United States. *Virology*, *527*(May 2018), 32–37. https://doi.org/10.1016/j.virol.2018.10.019

localeai. (2020). *covid19-live-visualization.*

MOH\_Kenya. (2020). *#KomeshaCorona.*

NovelCOVID. (2020). *API.*

Organization, W. H., & others. (2020). *Coronavirus disease 2019 (COVID-19): situation report, 72*.

Rajendran, D. K., Rajagopal, V., Alagumanian, S., Santhosh Kumar, T., Sathiya Prabhakaran, S. P., & Kasilingam, D. (2020). Systematic literature review on novel corona virus SARS-CoV-2: a threat to human era. *VirusDisease*. https://doi.org/10.1007/s13337-020-00604-z

Shereen, M. A., Khan, S., Kazmi, A., Bashir, N., & Siddique, R. (2020). COVID-19 infection: Origin, transmission, and characteristics of human coronaviruses. *Journal of Advanced Research*, *24*, 91–98. https://doi.org/10.1016/j.jare.2020.03.005

telegram.org. (2020). *telegram.org.*

Worldometers. (2020). *COVID-19 Coronavirus Pandemic.*

Zhang, C., Zheng, W., Huang, X., Bell, E. W., Zhou, X., & Zhang, Y. (2020). Protein Structure and Sequence Reanalysis of 2019-nCoV Genome Refutes Snakes as Its Intermediate Host and the Unique Similarity between Its Spike Protein Insertions and HIV-1. *Journal of Proteome Research*, *19*(4), 1351–1360. https://doi.org/10.1021/acs.jproteome.0c00129

Zhang, L., & Liu, Y. (2020). Potential interventions for novel coronavirus in China: A systematic review. In *Journal of Medical Virology* (Vol. 92, Issue 5). https://doi.org/10.1002/jmv.25707

# Appendix A

## A1 PYTHON LANGUAGE GRAMMER SPECIFICATIONS

*# Grammar for Python*

*# NOTE WELL: You should also follow all the steps listed at*

*# https://devguide.python.org/grammar/*

*# Start symbols for the grammar:*

*# single\_input is a single interactive statement;*

*# file\_input is a module or sequence of commands read from an input file;*

*# eval\_input is the input for the eval() functions.*

*# func\_type\_input is a PEP 484 Python 2 function type comment*

*# NB: compound\_stmt in single\_input is followed by extra NEWLINE!*

*# NB: due to the way TYPE\_COMMENT is tokenized it will always be followed by a NEWLINE*

single\_input: NEWLINE | simple\_stmt | compound\_stmt NEWLINE

file\_input: (NEWLINE | stmt)\* ENDMARKER

eval\_input: testlist NEWLINE\* ENDMARKER

decorator: '@' dotted\_name [ '(' [arglist] ')' ] NEWLINE

decorators: decorator+

decorated: decorators (classdef | funcdef | async\_funcdef)

async\_funcdef: ASYNC funcdef

funcdef: 'def' NAME parameters ['->' test] ':' [TYPE\_COMMENT] func\_body\_suite

parameters: '(' [typedargslist] ')'

*# The following definition for typedarglist is equivalent to this set of rules:*

*#*

*# arguments = argument (',' [TYPE\_COMMENT] argument)\**

*# argument = tfpdef ['=' test]*

*# kwargs = '\*\*' tfpdef [','] [TYPE\_COMMENT]*

*# args = '\*' [tfpdef]*

*# kwonly\_kwargs = (',' [TYPE\_COMMENT] argument)\* (TYPE\_COMMENT | [',' [TYPE\_COMMENT] [kwargs]])*

*# args\_kwonly\_kwargs = args kwonly\_kwargs | kwargs*

*# poskeyword\_args\_kwonly\_kwargs = arguments ( TYPE\_COMMENT | [',' [TYPE\_COMMENT] [args\_kwonly\_kwargs]])*

*# typedargslist\_no\_posonly = poskeyword\_args\_kwonly\_kwargs | args\_kwonly\_kwargs*

*# typedarglist = (arguments ',' [TYPE\_COMMENT] '/' [',' [[TYPE\_COMMENT] typedargslist\_no\_posonly]])|(typedargslist\_no\_posonly)"*

*#*

*# It needs to be fully expanded to allow our LL(1) parser to work on it.*

typedargslist: (

(tfpdef ['=' test] (',' [TYPE\_COMMENT] tfpdef ['=' test])\* ',' [TYPE\_COMMENT] '/' [',' [ [TYPE\_COMMENT] tfpdef ['=' test] (

',' [TYPE\_COMMENT] tfpdef ['=' test])\* (TYPE\_COMMENT | [',' [TYPE\_COMMENT] [

'\*' [tfpdef] (',' [TYPE\_COMMENT] tfpdef ['=' test])\* (TYPE\_COMMENT | [',' [TYPE\_COMMENT] ['\*\*' tfpdef [','] [TYPE\_COMMENT]]])

| '\*\*' tfpdef [','] [TYPE\_COMMENT]]])

| '\*' [tfpdef] (',' [TYPE\_COMMENT] tfpdef ['=' test])\* (TYPE\_COMMENT | [',' [TYPE\_COMMENT] ['\*\*' tfpdef [','] [TYPE\_COMMENT]]])

| '\*\*' tfpdef [','] [TYPE\_COMMENT]]] )

| (tfpdef ['=' test] (',' [TYPE\_COMMENT] tfpdef ['=' test])\* (TYPE\_COMMENT | [',' [TYPE\_COMMENT] [

'\*' [tfpdef] (',' [TYPE\_COMMENT] tfpdef ['=' test])\* (TYPE\_COMMENT | [',' [TYPE\_COMMENT] ['\*\*' tfpdef [','] [TYPE\_COMMENT]]])

| '\*\*' tfpdef [','] [TYPE\_COMMENT]]])

| '\*' [tfpdef] (',' [TYPE\_COMMENT] tfpdef ['=' test])\* (TYPE\_COMMENT | [',' [TYPE\_COMMENT] ['\*\*' tfpdef [','] [TYPE\_COMMENT]]])

| '\*\*' tfpdef [','] [TYPE\_COMMENT])

)

tfpdef: NAME [':' test]

*# The following definition for varargslist is equivalent to this set of rules:*

*#*

*# arguments = argument (',' argument )\**

*# argument = vfpdef ['=' test]*

*# kwargs = '\*\*' vfpdef [',']*

*# args = '\*' [vfpdef]*

*# kwonly\_kwargs = (',' argument )\* [',' [kwargs]]*

*# args\_kwonly\_kwargs = args kwonly\_kwargs | kwargs*

*# poskeyword\_args\_kwonly\_kwargs = arguments [',' [args\_kwonly\_kwargs]]*

*# vararglist\_no\_posonly = poskeyword\_args\_kwonly\_kwargs | args\_kwonly\_kwargs*

*# varargslist = arguments ',' '/' [','[(vararglist\_no\_posonly)]] | (vararglist\_no\_posonly)*

*#*

*# It needs to be fully expanded to allow our LL(1) parser to work on it.*

varargslist: vfpdef ['=' test ](',' vfpdef ['=' test])\* ',' '/' [',' [ (vfpdef ['=' test] (',' vfpdef ['=' test])\* [',' [

'\*' [vfpdef] (',' vfpdef ['=' test])\* [',' ['\*\*' vfpdef [',']]]

| '\*\*' vfpdef [',']]]

| '\*' [vfpdef] (',' vfpdef ['=' test])\* [',' ['\*\*' vfpdef [',']]]

| '\*\*' vfpdef [',']) ]] | (vfpdef ['=' test] (',' vfpdef ['=' test])\* [',' [

'\*' [vfpdef] (',' vfpdef ['=' test])\* [',' ['\*\*' vfpdef [',']]]

| '\*\*' vfpdef [',']]]

| '\*' [vfpdef] (',' vfpdef ['=' test])\* [',' ['\*\*' vfpdef [',']]]

| '\*\*' vfpdef [',']

)

vfpdef: NAME

stmt: simple\_stmt | compound\_stmt

simple\_stmt: small\_stmt (';' small\_stmt)\* [';'] NEWLINE

small\_stmt: (expr\_stmt | del\_stmt | pass\_stmt | flow\_stmt |

import\_stmt | global\_stmt | nonlocal\_stmt | assert\_stmt)

expr\_stmt: testlist\_star\_expr (annassign | augassign (yield\_expr|testlist) |

[('=' (yield\_expr|testlist\_star\_expr))+ [TYPE\_COMMENT]] )

annassign: ':' test ['=' (yield\_expr|testlist\_star\_expr)]

testlist\_star\_expr: (test|star\_expr) (',' (test|star\_expr))\* [',']

augassign: ('+=' | '-=' | '\*=' | '@=' | '/=' | '%=' | '&=' | '|=' | '^=' |

'<<=' | '>>=' | '\*\*=' | '//=')

*# For normal and annotated assignments, additional restrictions enforced by the interpreter*

del\_stmt: 'del' exprlist

pass\_stmt: 'pass'

flow\_stmt: break\_stmt | continue\_stmt | return\_stmt | raise\_stmt | yield\_stmt

break\_stmt: 'break'

continue\_stmt: 'continue'

return\_stmt: 'return' [testlist\_star\_expr]

yield\_stmt: yield\_expr

raise\_stmt: 'raise' [test ['from' test]]

import\_stmt: import\_name | import\_from

import\_name: 'import' dotted\_as\_names

*# note below: the ('.' | '...') is necessary because '...' is tokenized as ELLIPSIS*

import\_from: ('from' (('.' | '...')\* dotted\_name | ('.' | '...')+)

'import' ('\*' | '(' import\_as\_names ')' | import\_as\_names))

import\_as\_name: NAME ['as' NAME]

dotted\_as\_name: dotted\_name ['as' NAME]

import\_as\_names: import\_as\_name (',' import\_as\_name)\* [',']

dotted\_as\_names: dotted\_as\_name (',' dotted\_as\_name)\*

dotted\_name: NAME ('.' NAME)\*

global\_stmt: 'global' NAME (',' NAME)\*

nonlocal\_stmt: 'nonlocal' NAME (',' NAME)\*

assert\_stmt: 'assert' test [',' test]

compound\_stmt: if\_stmt | while\_stmt | for\_stmt | try\_stmt | with\_stmt | funcdef | classdef | decorated | async\_stmt

async\_stmt: ASYNC (funcdef | with\_stmt | for\_stmt)

if\_stmt: 'if' namedexpr\_test ':' suite ('elif' namedexpr\_test ':' suite)\* ['else' ':' suite]

while\_stmt: 'while' namedexpr\_test ':' suite ['else' ':' suite]

for\_stmt: 'for' exprlist 'in' testlist ':' [TYPE\_COMMENT] suite ['else' ':' suite]

try\_stmt: ('try' ':' suite

((except\_clause ':' suite)+

['else' ':' suite]

['finally' ':' suite] |

'finally' ':' suite))

with\_stmt: 'with' with\_item (',' with\_item)\* ':' [TYPE\_COMMENT] suite

with\_item: test ['as' expr]

*# NB compile.c makes sure that the default except clause is last*

except\_clause: 'except' [test ['as' NAME]]

suite: simple\_stmt | NEWLINE INDENT stmt+ DEDENT

namedexpr\_test: test [':=' test]

test: or\_test ['if' or\_test 'else' test] | lambdef

test\_nocond: or\_test | lambdef\_nocond

lambdef: 'lambda' [varargslist] ':' test

lambdef\_nocond: 'lambda' [varargslist] ':' test\_nocond

or\_test: and\_test ('or' and\_test)\*

and\_test: not\_test ('and' not\_test)\*

not\_test: 'not' not\_test | comparison

comparison: expr (comp\_op expr)\*

*# <> isn't actually a valid comparison operator in Python. It's here for the*

*# sake of a \_\_future\_\_ import described in PEP 401 (which really works :-)*

comp\_op: '<'|'>'|'=='|'>='|'<='|'<>'|'!='|'in'|'not' 'in'|'is'|'is' 'not'

star\_expr: '\*' expr

expr: xor\_expr ('|' xor\_expr)\*

xor\_expr: and\_expr ('^' and\_expr)\*

and\_expr: shift\_expr ('&' shift\_expr)\*

shift\_expr: arith\_expr (('<<'|'>>') arith\_expr)\*

arith\_expr: term (('+'|'-') term)\*

term: factor (('\*'|'@'|'/'|'%'|'//') factor)\*

factor: ('+'|'-'|'~') factor | power

power: atom\_expr ['\*\*' factor]

atom\_expr: [AWAIT] atom trailer\*

atom: ('(' [yield\_expr|testlist\_comp] ')' |

'[' [testlist\_comp] ']' |

'{' [dictorsetmaker] '}' |

NAME | NUMBER | STRING+ | '...' | 'None' | 'True' | 'False')

testlist\_comp: (namedexpr\_test|star\_expr) ( comp\_for | (',' (namedexpr\_test|star\_expr))\* [','] )

trailer: '(' [arglist] ')' | '[' subscriptlist ']' | '.' NAME

subscriptlist: subscript (',' subscript)\* [',']

subscript: test | [test] ':' [test] [sliceop]

sliceop: ':' [test]

exprlist: (expr|star\_expr) (',' (expr|star\_expr))\* [',']

testlist: test (',' test)\* [',']

dictorsetmaker: ( ((test ':' test | '\*\*' expr)

(comp\_for | (',' (test ':' test | '\*\*' expr))\* [','])) |

((test | star\_expr)

(comp\_for | (',' (test | star\_expr))\* [','])) )

classdef: 'class' NAME ['(' [arglist] ')'] ':' suite

arglist: argument (',' argument)\* [',']

*# The reason that keywords are test nodes instead of NAME is that using NAME*

*# results in an ambiguity. ast.c makes sure it's a NAME.*

*# "test '=' test" is really "keyword '=' test", but we have no such token.*

*# These need to be in a single rule to avoid grammar that is ambiguous*

*# to our LL(1) parser. Even though 'test' includes '\*expr' in star\_expr,*

*# we explicitly match '\*' here, too, to give it proper precedence.*

*# Illegal combinations and orderings are blocked in ast.c:*

*# multiple (test comp\_for) arguments are blocked; keyword unpackings*

*# that precede iterable unpackings are blocked; etc.*

argument: ( test [comp\_for] |

test ':=' test |

test '=' test |

'\*\*' test |

'\*' test )

comp\_iter: comp\_for | comp\_if

sync\_comp\_for: 'for' exprlist 'in' or\_test [comp\_iter]

comp\_for: [ASYNC] sync\_comp\_for

comp\_if: 'if' test\_nocond [comp\_iter]

*# not used in grammar, but may appear in "node" passed from Parser to Compiler*

encoding\_decl: NAME

yield\_expr: 'yield' [yield\_arg]

yield\_arg: 'from' test | testlist\_star\_expr

*# the TYPE\_COMMENT in suites is only parsed for funcdefs,*

*# but can't go elsewhere due to ambiguity*

func\_body\_suite: simple\_stmt | NEWLINE [TYPE\_COMMENT NEWLINE] INDENT stmt+ DEDENT

func\_type\_input: func\_type NEWLINE\* ENDMARKER

func\_type: '(' [typelist] ')' '->' test

*# typelist is a modified typedargslist (see above)*

typelist: (test (',' test)\* [','

['\*' [test] (',' test)\* [',' '\*\*' test] | '\*\*' test]]

| '\*' [test] (',' test)\* [',' '\*\*' test] | '\*\*' test)

## A2 JSON LANGUAGE GRAMMER SPECIFICATIONS

{

    "links": {

      "self": "http://example.com/articles",

      "next": "http://example.com/articles?page[offset]=2",

      "last": "http://example.com/articles?page[offset]=10"

    },

    "data": [{

      "type": "articles",

      "id": "1",

      "attributes": {

        "title": "JSON:API paints my bikeshed!"

      },

      "relationships": {

        "author": {

          "links": {

            "self": "http://example.com/articles/1/relationships/author",

            "related": "http://example.com/articles/1/author"

          },

          "data": { "type": "people", "id": "9" }

        },

        "comments": {

          "links": {

            "self": "http://example.com/articles/1/relationships/comments",

            "related": "http://example.com/articles/1/comments"

          },

          "data": [

            { "type": "comments", "id": "5" },

            { "type": "comments", "id": "12" }

          ]

        }

      },

      "links": {

        "self": "http://example.com/articles/1"

      }

    }],

    "included": [{

      "type": "people",

      "id": "9",

      "attributes": {

        "firstName": "Dan",

        "lastName": "Gebhardt",

        "twitter": "dgeb"

      },

      "links": {

        "self": "http://example.com/people/9"

      }

    }, {

      "type": "comments",

      "id": "5",

      "attributes": {

        "body": "First!"

      },

      "relationships": {

        "author": {

          "data": { "type": "people", "id": "2" }

        }

      },

      "links": {

        "self": "http://example.com/comments/5"

      }

    }, {

      "type": "comments",

      "id": "12",

      "attributes": {

        "body": "I like XML better"

      },

      "relationships": {

        "author": {

          "data": { "type": "people", "id": "9" }

        }

      },

      "links": {

        "self": "http://example.com/comments/12"

      }

    }]

  }

# Appendix B

# Sample source code and output listing

## B1 COMMANDS CODE FOR THE BOT AND REPLIES

{

    "start": [

        "Greetings {}!",

        "",

        "You just found @Coronaviruskenyabot \uD83E\uDDA0. This bot gives you up-to-date information on Kenya and worldwide spread of the Covid-19 disease.",

        "",

        "To get a quick overview of the current status, try /today ,/kenya or /world.",

        "You can view detailed statistics of your country by using the /list command.",

        "Subscribe to daily updates with /subscribe for automatic notifications.",

        "\*MAKE SURE TO USE THE /setcountry COMMAND TO SET YOUR COUNNTRY FOLLOWED BY /subscribe TO GET DAILY NOTIFICATIONS\*",

        "",

        "\*To get the FAQs about corona virus disease\* use /faqs",

        "",

        "\*You have no idea what COVID-19 is or having doubts\* use /faqs",

        "",

        "For more, see /help."

    ],

    "donate":[

        "\uD83E\uDDA0 @Coronaviruskenyabot  \*Donate\*  \*#STAYSAFE\*",

        "Donate to this project for  more exclusive platforms such as WhatsApp.\n",

        "\*If you'd like to donate do it\* [HERE](https://mikeowino.com/donation)"

    ],

    "help": [

        "\uD83E\uDDA0 @Coronaviruskenyabot Help \*#STAYSAFE\*",

        "",

        "This bot gives you up-to-date information on the worldwide spread of the COVID-19 disease.",

        "",

        "\*/world\*  \u2022  Worldwide case statistics.",

        "\*/today\*  \u2022  Summary of today's cases.",

        "\*/list\*  \u2022  List of countries ordered by number of cases.",

        "\*/subscribe\*  \u2022  Subscribe to daily status updates with new case statistics.",

        "\*/help\*  \u2022  Show this help.",

        "",

        "You can use this bot in any chat or group without adding it by typing `@Coronaviruskenyabot [country]`.",

        "",

        "—",

        "\u2139\uFE0F For issues and feedback, [Contact Me](https://t.me/decipha).\n",

        "Sources: [WORLDOMETERS](https://www.worldometers.info/coronavirus/country/kenya/), [WHO](https://covid19.who.int), [MOH](https://twitter.com/MOH\_Kenya).\n",

        "Data is accessed via [NovelCOVID API](https://github.com/NovelCOVID/API)."

    ],

    "faqs": [

        "\uD83E\uDDA0 @Coronaviruskenyabot FAQs \*#STAYSAFE\*",

        "",

        "\*From the Government\*",

        "",

        "\*IF YOU FEEL UNWELL, HAVE A COLD, A COUGH OR FEVER, PLEASE CALL\* 719 \*OR TEXT\* 719",

        "",

        "\*What is a Coronavirus ?\*",

        "Coronaviruses are a large family of viruses that are known to cause illness ranging from the common cold to more severe diseases such as Middle East Respiratory Syndrome (MERS) and Severe Acute Respiratory Syndrome (SARS).",

        "",

        "\*What is novel coronavirus ?\*",

        "A novel coronavirus (CoV) is a new strain of coronavirus that has not been previously identified in humans.",

        "",

        "\*Can humans get infected from novel coronavirus from animal source ?\*",

        "Coronaviruses are a large family of viruses that are common in animals. Occasionally, people get infected with these viruses which may then spread to other people.Possible animal sources of COVID-19 have not yet been confirmed.",

        "To protect yourself, such as when visiting live animal markets, avoid direct contact with animals and surfaces in contact with animals. Ensure good food safety practices at all times. Handle raw meat, milk or animal organs with care to avoid contamination of uncooked foods and avoid consuming raw or undercooked animal products.",

        "",

        "\*What are the symptoms of COVID-19 ?\*",

        "common signs include respiratory \*symptoms, fever, cough, shortness of breath, and breathing difficulties.\* In more severe cases, infection can cause \*pneumonia, severe acute respiratory syndrome, kidney failure and even death.\*",

        "",

        "\*Can the virus that causes COVID-19 be transmitted through the air ?\*",

        "Studies to date suggest that the virus that causes COVID-19 is mainly transmitted through contact with respiratory droplets rather than through the air",

        "",

        "\*Is there a vaccine for novel coronavirus ?\*",

        "Not yet. To date, there is no vaccine and no specific antiviral medicine to prevent or treat COVID-2019. However, those affected should receive care to relieve symptoms. People with serious illness should be hospitalized. Most patients recover thanks to supportive care."

    ],

    "today": [

        "\uD83D\uDCDD Status on {:%Y-%m-%d} at {:%H:%m} UTC \*#STAYSAFE\*",

        "",

        "\uD83C\uDF10 Worldwide, `{:,}` COVID-19 infections have been reported. `{:,}` infected people died.",

        "\uD83E\uDDA0 Today, there have been `{:,}` new cases.",

        "\u26B0\uFE0F The number of deaths since 0:00 UTC is `{:,}`.",

        "More: /world",

        ""

    ],

    "today\_country": [

        "{} In {}, `{:,}` infections have been reported. `{:,}` infected people died.",

        "\uD83E\uDDA0 Today, there have been `{:,}` new cases.",

        "\u26B0\uFE0F The number of deaths since 0:00 UTC is `{:,}`.",

        "More: /{}",

        ""

    ],

    "today\_footer": "/list  \u2022  /setcountry  \u2022  /help",

    "stats\_table": [

        "COVID-19 Statistics for \*{}\* {}",

        "",

        "\uD83E\uDDA0  `{:,}`  estimated cases",

        "\uD83C\uDF21  `{:,}`  active (`{:.1%}`)",

        "\uD83D\uDD06  `{:,}`  recovered (`{:.1%}`)",

        "\u26B0\uFE0F  `{:,}`  deaths (`{:.1%}`)",

        "",

        "\_Today\_ \uD83D\uDCC6",

        "\uD83E\uDDA0  `+ {:,}`  new cases",

        "\u26B0\uFE0F  `+ {:,}`  deaths",

        ""

    ],

    "stats\_table\_more": [

        "\uD83D\uDC65  `{:,}`  cases per million people",

        "\uD83D\uDC65  `{:,}` deaths per million people",

        ""

    ],

    "stats\_table\_simple": [

        "COVID-19 Statistics for \*{}\* {}",

        "",

        "\uD83E\uDDA0  `{:,}`  estimated cases",

        "\u26B0\uFE0F  `{:,}`  deaths (`{:.1%}`)",

        ""

    ],

    "stats\_updated": "\_Updated: {:%Y-%m-%d %H:%m} UTC\_",

    "stats\_more": "More",

    "stats\_less": "Less",

    "more": "More: @Coronaviruskenyabot",

    "page\_left": "\u25C0\uFE0F Page {}",

    "page\_right": "Page {} \u25B6\uFE0F",

    "to\_start": "\u23EA To Start",

    "to\_end": "To End \u23E9",

    "list\_header": "\uD83D\uDCCA Countries by \*{}\*\n",

    "no\_data": "Sorry, no data available! Try again later.",

    "unknown\_place": "Please use the available commands to get a reply.",

    "no\_country\_set": "IFf you have not configured your country. Use /setcountry to configure it.",

    "setcountry\_start": [

        "To set your country, send your country name.",

        "Send /cancel to cancel."

    ],

    "cancel": "Cancelled.",

    "setcountry\_success": "Successfully set your country to \*{}\*.",

    "subscribe": [

        "You have subscribed to daily case updates.",

        "To unsubscribe, send /unsubscribe."

    ],

    "unsubscribe": [

        "You have unsubscribed from daily updates.",

        "To re-subscribe, send /subscribe."

    ],

    "sort\_order": "Sort order \uD83D\uDCCA",

    "back": "\u25C0\uFE0F Back",

    "sort\_order\_cases": "\uD83E\uDDA0 total",

    "sort\_order\_casesPerOneMillion": "\uD83E\uDDA0 / million",

    "sort\_order\_todayCases": "\uD83E\uDDA0 today",

    "sort\_order\_deaths": "\u26B0\uFE0F total",

    "sort\_order\_deathsPerOneMillion": "\u26B0\uFE0F / million",

    "sort\_order\_todayDeaths": "\u26B0\uFE0F today"

}

## B2 CODE FOR ACCESSING THE COVID-19 API SERVERS

import requests

BASE\_URL="https://corona.lmao.ninja/v2/"

class CovidApi:

    """A simple wrapper for the NovelCOVID API (https://github.com/NovelCOVID/API).

    """

    def \_\_init\_\_(self):

        self.countries = self.\_all\_countries()

        self.name\_map = self.\_build\_name\_map(self.countries)

        self.us\_states = self.\_all\_us\_states()

        self.de\_states = self.\_all\_de\_states()

    def \_clean(self, s):

        s = s.replace('\xad', '')

        s = s.replace('\n', '')

        return s

    def \_build\_name\_map(self, countries):

        name\_map = {}

        for iso2, country in countries.items():

            name\_map[country['iso2'].lower()] = iso2

            name\_map[country['iso3'].lower()] = iso2

            name\_map[country['name'].lower()] = iso2

        return name\_map

    def \_all\_countries(self):

        response = requests.get(BASE\_URL+"countries")

        if response.status\_code == 200:

            countries = {}

            for item in response.json():

                iso2 = item['countryInfo']['iso2']

                if iso2:

                    countries[iso2] = item['countryInfo']

                    countries[iso2]['name'] = item['country']

            return countries

        else:

            return {}

    def \_all\_us\_states(self):

        response = requests.get(BASE\_URL+"states")

        if response.status\_code == 200:

            countries = []

            for item in response.json():

                countries.append(item['state'])

            return countries

        else:

            return []

    def \_all\_de\_states(self):

        response = requests.get(BASE\_URL+"gov/de")

        if response.status\_code == 200:

            countries = []

            for item in response.json():

                if item['province'].lower() != 'total':

                    countries.append(self.\_clean(item['province']))

            return countries

        else:

            return []

    def cases\_world(self):

        response = requests.get(BASE\_URL+"all")

        if response.status\_code == 200:

            return response.json()

        else:

            return None

    def cases\_country\_list(self, sort\_by="cases"):

        response = requests.get(BASE\_URL+"countries", params={'sort': sort\_by})

        if response.status\_code == 200:

            return [item for item in response.json() if item['countryInfo']['iso2']]

        else:

            return []

    def cases\_country(self, country):

        country\_code = self.name\_map[country.lower()]

        response = requests.get(BASE\_URL+"countries/{}".format(country\_code))

        if response.status\_code == 200:

            data = response.json()

            del data['countryInfo']

            return data

        else:

            return None

    def cases\_us\_state(self, state):

        response = requests.get(BASE\_URL+"states/{}".format(state))

        if response.status\_code == 200:

            data = response.json()

            # additions to unify format with countries

            data['recovered'] = data['cases'] - data['active'] - data['deaths']

            return data

        else:

            return None

    def cases\_de\_state(self, state):

        response = requests.get(BASE\_URL+"gov/de")

        if response.status\_code == 200:

            data = response.json()

            filtered = [item for item in data if self.\_clean(item['province'].lower()) == state.lower()]

            return filtered[0] if len(filtered) > 0 else None

        else:

            return None

## B3 MAIN SOURCE CODE OF THE BOT INCLUDING ACCESS TO THE BOT API TOKEN AND NOTIFY TIME FOR SENDING THE DAILY NOTIFICATIONS

#!/usr/bin/env python3

from datetime import datetime

import json

import logging

import re

import requests

from time import sleep

from telegram import ParseMode

from telegram import InlineKeyboardButton, InlineKeyboardMarkup, InlineQueryResultArticle, InputTextMessageContent

from telegram.ext import Updater, CommandHandler, CallbackQueryHandler, MessageHandler, Filters, InlineQueryHandler

from telegram.ext import PicklePersistence, ConversationHandler, DispatcherHandlerStop

from telegram.error import TelegramError

from statistics\_api import CovidApi

import wikidata

from resources.resolver import resolve

from utils import \*

CONFIG\_FILE="config.json"

logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

                    level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

WORLD\_IDENT="world"

api = CovidApi()

# command /start

def command\_start(update, context):

    update.message.reply\_markdown(resolve('start', lang(update), update.message.from\_user.first\_name))

#command /faqs

def command\_faqs(update, context):

    update.message.reply\_markdown(resolve('faqs', lang(update)), disadle\_web\_page\_preview=True)

#command /donate

def command\_donate(update, context):

    update.message.reply\_markdown(resolve('donate', lang(update)), disable\_web\_page\_preview=True)

# command /help

@handler\_decorator

def command\_help(update, context):

    update.message.reply\_markdown(resolve('help', lang(update)), disable\_web\_page\_preview=True)

### World & country stats + status report ###

def format\_stats(update, code, data, icon=None, detailed=False):

    if code in api.countries:

        name = api.countries[code]['name']

    elif code == WORLD\_IDENT:

        name = "the World"

        icon = '\U0001f310'

    else:

        name = code

    if not icon:

        icon = flag(code)

    p\_dead = data['deaths'] / data['cases']

    if 'active' in data and 'todayCases' in data: # we have detailed data, so use more detailed view

        p\_active = data['active'] / data['cases']

        p\_recov = data['recovered'] / data['cases']

        text = resolve('stats\_table', lang(update), name, icon, data['cases'],

                data['active'], p\_active, data['recovered'], p\_recov, data['deaths'], p\_dead,

                data['todayCases'], data['todayDeaths'])

        if detailed:

            text += '\n'+resolve('stats\_table\_more', lang(update), data['casesPerOneMillion'] or 0,

                            data['deathsPerOneMillion'] or 0)

    else: # we only have limited data

        text = resolve('stats\_table\_simple', lang(update), name, icon, data['cases'], data['deaths'], p\_dead)

    text += '\n'+resolve('stats\_updated', lang(update), datetime.utcfromtimestamp(data['updated'] / 1e3))

    return text

def get\_stats\_keyboard(update, country\_code, is\_detailed=False):

    keyboard = [[]]

    caption\_key = 'stats\_more' if not is\_detailed else 'stats\_less'

    caption = resolve(caption\_key, lang(update))

    keyboard[0].append(InlineKeyboardButton(caption,

                    callback\_data="stats {} {}".format(country\_code, 1 if is\_detailed else 0)))

    return InlineKeyboardMarkup(keyboard)

# the text used for daily notifications and /today

def get\_status\_report(country\_code=None, lang="en"):

    data = api.cases\_world()

    if data:

        dt = datetime.utcfromtimestamp(data['updated'] / 1e3)

        text = resolve('today', lang,

                dt, dt, data['cases'], data['deaths'], data['todayCases'], data['todayDeaths'])

        # fetch data of home country if set

        if country\_code:

            country\_data = api.cases\_country(country\_code)

            text += '\n'+resolve('today\_country', lang, flag(country\_code),

                            api.countries[country\_code]['name'], country\_data['cases'], country\_data['deaths'],

                            country\_data['todayCases'], country\_data['todayDeaths'], country\_code.lower())

        else:

            text += '\n\_'+resolve('no\_country\_set', lang)+'\_\n'

        text += '\n'+resolve('today\_footer', lang)

    else:

        text = resolve('no\_data',lang)

    return text

# command /today

@handler\_decorator

def command\_today(update, context):

    if 'country' in context.chat\_data:

        country\_code = context.chat\_data['country']

    else:

        country\_code = None

    text = get\_status\_report(country\_code, lang(update))

    update.message.reply\_markdown(text)

def format\_list\_item(data, order):

    code = data['countryInfo']['iso2'].lower()

    icon = resolve('sort\_order\_'+order, None).split(' ')[0]

    number = data[order]

    text = """

{} \*{}  -  {}\*  -  {} `{:,}`

    """.format(flag(code), data['country'], '/'+code, icon, number)

    return text

def get\_list\_keyboard(update, current\_index, limit, last=False):

    keyboard = [[]]

    if current\_index > 0:

        keyboard[0].append(InlineKeyboardButton(resolve('page\_left', lang(update), current\_index),

                                callback\_data="list {} {}".format(current\_index-1, limit)))

    if not last:

        keyboard[0].append(InlineKeyboardButton(resolve('page\_right', lang(update), current\_index+2),

                                callback\_data="list {} {}".format(current\_index+1, limit)))

    if current\_index > 0:

        keyboard.append([

            InlineKeyboardButton(resolve('to\_start', lang(update)), callback\_data="list 0 {}".format(limit))])

    else:

        keyboard.append([

            InlineKeyboardButton(resolve('to\_end', lang(update)), callback\_data="list -1 {}".format(limit))])

    keyboard.append([

        InlineKeyboardButton(resolve('sort\_order', lang(update)),

                callback\_data="list\_order\_menu 1 ({} {} {})".format(current\_index, limit, int(last)))

    ])

    return InlineKeyboardMarkup(keyboard)

SORT\_ORDERS = ['cases', 'deaths', 'casesPerOneMillion', 'deathsPerOneMillion', 'todayCases', 'todayDeaths']

def get\_list\_order\_keyboard(update, current\_index, limit, last=False):

    keyboard = []

    for i, sort\_order in enumerate(SORT\_ORDERS):

        button = InlineKeyboardButton(resolve("sort\_order\_"+sort\_order, lang(update)), callback\_data="list\_order {} {}".format(sort\_order, limit))

        if i % 2 == 0:

            l = [button]

        else:

            l.append(button)

            keyboard.append(l)

    keyboard.append([InlineKeyboardButton(resolve('back', lang(update)),

                callback\_data="list\_order\_menu 0 ({} {} {})".format(current\_index, limit, int(last)))])

    return InlineKeyboardMarkup(keyboard)

# command /world

@handler\_decorator

def command\_world(update, context):

    photo\_file = wikidata.cases\_world\_map()

    data = api.cases\_world()

    if data:

        text = format\_stats(update, WORLD\_IDENT, data)

        update.message.reply\_photo(photo=photo\_file, caption=text, parse\_mode=ParseMode.MARKDOWN,

                                   reply\_markup=get\_stats\_keyboard(update, WORLD\_IDENT))

    else:

        update.message.reply\_text(resolve('no\_data', lang(update)))

# command /[country]

@handler\_decorator

def command\_country(update, context, country\_code):

    photo\_file = wikidata.cases\_country\_map(country\_code)

    data = api.cases\_country(country\_code)

    if data:

        text = format\_stats(update, country\_code, data)

        if photo\_file:

            update.message.reply\_photo(photo=photo\_file, caption=text, parse\_mode=ParseMode.MARKDOWN,

                                       reply\_markup=get\_stats\_keyboard(update, country\_code))

        else:

            update.message.reply\_markdown(text, reply\_markup=get\_stats\_keyboard(update, country\_code))

    else:

        update.message.reply\_text(resolve('no\_data', lang(update)))

def command\_us\_state(update, context, state):

    data = api.cases\_us\_state(state)

    if data:

        text = format\_stats(update, state.title(), data, icon='\uD83C\uDDFA\uD83C\uDDF8')

        update.message.reply\_markdown(text)

    else:

        update.message.reply\_text(resolve('no\_data', lang(update)))

def command\_de\_state(update, context, state):

    data = api.cases\_de\_state(state)

    if data:

        text = format\_stats(update, state.title(), data, icon='\uD83C\uDDE9\uD83C\uDDEA')

        update.message.reply\_markdown(text)

    else:

        update.message.reply\_text(resolve('no\_data', lang(update)))

def callback\_stats(update, context):

    query = update.callback\_query

    country\_code = context.match.group(1)

    # True if we want to \*show\* detailed mode, False otherwise

    is\_detailed = not bool(int(context.match.group(2)))

    if country\_code == WORLD\_IDENT:

        data = api.cases\_world()

    else:

        data = api.cases\_country(country\_code)

    if data:

        text = format\_stats(update, country\_code, data, detailed=is\_detailed)

        if query.message.caption:

            query.edit\_message\_caption(caption=text, parse\_mode=ParseMode.MARKDOWN,

                                       reply\_markup=get\_stats\_keyboard(update, country\_code, is\_detailed))

        else:

            query.edit\_message\_text(text=text, parse\_mode=ParseMode.MARKDOWN,

                                    reply\_markup=get\_stats\_keyboard(update, country\_code, is\_detailed))

### Country list ###

# command /list

@handler\_decorator

def command\_list(update, context):

    # set or retrieve sort order

    if len(context.args) > 0:

        order = context.args[0]

        context.chat\_data['order'] = order

    elif 'order' in context.chat\_data:

        order = context.chat\_data['order']

    else:

        # use first possible order as default

        order = SORT\_ORDERS[0]

        context.chat\_data['order'] = order

    # by default, return 7 items. min 2 and max 20.

    limit = int(context.args[1]) if len(context.args) > 1 else 7

    limit = min(max(2, limit), 20)

    case\_list = api.cases\_country\_list(sort\_by=order)[:limit]

    if len(case\_list) > 0:

        text = resolve('list\_header', lang(update), resolve("sort\_order\_"+order, lang(update)))

        for item in case\_list:

            text += format\_list\_item(item, order)

        update.message.reply\_markdown(text, reply\_markup=get\_list\_keyboard(update, 0, limit))

    else:

        update.message.reply\_text(resolve('no\_data', lang(update)))

def callback\_list\_pages(update, context):

    query = update.callback\_query

    order = context.chat\_data.get('order', SORT\_ORDERS[0]) # for backward comp

    page, limit = int(context.match.group(1)), int(context.match.group(2))

    case\_list = api.cases\_country\_list(sort\_by=order)

    if page >= 0:

        case\_list = case\_list[page\*limit:(page+1)\*limit]

    else:

        # if the given page number is negative, we want to access the last page

        page = len(case\_list) // limit

        offset = len(case\_list) % limit

        case\_list = case\_list[-offset:]

    query.answer()

    if len(case\_list) > 0:

        text = resolve('list\_header', lang(update), resolve("sort\_order\_"+order, lang(update)))

        for item in case\_list:

            text += format\_list\_item(item, order)

        query.edit\_message\_text(text=text, parse\_mode=ParseMode.MARKDOWN,

                                reply\_markup=get\_list\_keyboard(update, page, limit, len(case\_list) < limit))

    else:

        query.edit\_message\_text(resolve('no\_data', lang(update)),

                                reply\_markup=get\_list\_keyboard(update, page, limit, len(case\_list) < limit))

def callback\_list\_order\_menu(update, context):

    query = update.callback\_query

    on = int(context.match.group(1))

    payload = [int(g) for g in context.match.group(2).split(" ")]

    query.answer()

    if on:

        query.edit\_message\_reply\_markup(reply\_markup=get\_list\_order\_keyboard(update, \*payload))

    else:

        query.edit\_message\_reply\_markup(reply\_markup=get\_list\_keyboard(update, \*payload))

def callback\_list\_order(update, context):

    query = update.callback\_query

    order = context.match.group(1)

    # save the selected order

    context.chat\_data['order'] = order

    limit = int(context.match.group(2))

    case\_list = api.cases\_country\_list(sort\_by=order)[:limit]

    query.answer()

    if len(case\_list) > 0:

        text = resolve('list\_header', lang(update), resolve("sort\_order\_"+order, lang(update)))

        for item in case\_list:

            text += format\_list\_item(item, order)

        query.edit\_message\_text(text=text, parse\_mode=ParseMode.MARKDOWN,

                                reply\_markup=get\_list\_keyboard(update, 0, limit, len(case\_list) < limit))

    else:

        query.edit\_message\_text(resolve('no\_data', lang(update)),

                                reply\_markup=get\_list\_keyboard(update, 0, limit, len(case\_list) < limit))

### Free text & inline ###

# free text input

@handler\_decorator

def handle\_text(update, context):

    query\_string = update.message.text.lower()

    if query\_string in api.name\_map:

        command\_country(update, context, api.name\_map[query\_string])

    elif WORLD\_IDENT in query\_string:

        command\_world(update, context)

    elif check\_flag(query\_string):

        code = code\_from\_flag(query\_string).lower()

        if code in api.name\_map:

            command\_country(update, context, api.name\_map[code])

    elif query\_string.title() in api.us\_states:

        command\_us\_state(update, context, query\_string)

    elif query\_string.title() in api.de\_states:

        command\_de\_state(update, context, query\_string)

    else:

        update.message.reply\_text(resolve('unknown\_place', lang(update)))

# inline queries

def handle\_inlinequery(update, context):

    inline\_query = update.inline\_query

    query\_string = inline\_query.query.lower()

    if not query\_string:

        return

    results = []

    # a special case matching 'world'

    if WORLD\_IDENT.startswith(query\_string):

        results.append((WORLD\_IDENT, WORLD\_IDENT))

    for name in api.name\_map.keys():

        if name.startswith(query\_string):

            results.append((name, "country"))

        # limit to the first threee results

        if len(results) >= 3:

            break

    if len(results) < 3:

        for state in api.us\_states:

            if state.lower().startswith(query\_string):

                results.append((state.lower(), "us\_state"))

            if len(results) >= 3:

                break

    if len(results) < 3:

        for state in api.de\_states:

            if state.lower().startswith(query\_string):

                results.append((state.lower(), "de\_state"))

            if len(results) >= 3:

                break

    query\_results = []

    for i,(s, t) in enumerate(results):

        if t == WORLD\_IDENT:

            data = api.cases\_world()

            text = format\_stats(update, WORLD\_IDENT, data, detailed=True)

        elif t == "us\_state":

            data = api.cases\_us\_state(s)

            text = format\_stats(update, s.title(), data, icon='\uD83C\uDDFA\uD83C\uDDF8')

        elif t == "de\_state":

            data = api.cases\_de\_state(s)

            text = format\_stats(update, s.title(), data, icon='\uD83C\uDDE9\uD83C\uDDEA')

        else:

            country\_code = api.name\_map[s]

            data = api.cases\_country(country\_code)

            text = format\_stats(update, country\_code, data, detailed=True)

        text+='\n'+resolve('more', lang(update))

        result\_content = InputTextMessageContent(text, parse\_mode=ParseMode.MARKDOWN)

        query\_results.append(

            InlineQueryResultArticle(id=i, title=s, input\_message\_content=result\_content)

        )

    inline\_query.answer(query\_results)

### Set country ###

# command /setcountry

@handler\_decorator

def handle\_setcountry\_start(update, context):

    update.message.reply\_markdown(resolve('setcountry\_start', lang(update)))

    return 1

def handle\_setcountry\_input(update, context):

    query\_string = update.message.text.lower()

    if query\_string in api.name\_map:

        code = api.name\_map[query\_string]

        context.chat\_data['country'] = code

        update.message.reply\_markdown(

                resolve('setcountry\_success', lang(update), api.countries[code]['name']))

        return ConversationHandler.END

    else:

        update.message.reply\_text(resolve('unknown\_place', lang(update)))

def handle\_setcountry\_cancel(update, context):

    update.message.reply\_text(resolve('cancel', lang(update)))

    return ConversationHandler.END

### Notification subscription ###

@handler\_decorator

def command\_subscribe(update, context):

    if not 'subscribers' in context.bot\_data:

        context.bot\_data['subscribers'] = [update.message.chat.id]

    elif not update.message.chat.id in context.bot\_data['subscribers']:

        context.bot\_data['subscribers'].append(update.message.chat.id)

    update.message.reply\_markdown(resolve('subscribe', lang(update)))

@handler\_decorator

def command\_unsubscribe(update, context):

    if 'subscribers' in context.bot\_data:

        if update.message.chat.id in context.bot\_data['subscribers']:

            context.bot\_data['subscribers'].remove(update.message.chat.id)

    update.message.reply\_markdown(resolve('unsubscribe', lang(update)))

# runs the status notification job once per day

def run\_notify(context):

    if not 'subscribers' in context.bot\_data:

        logger.warn("No subscribers list specified.")

        return

    count = 0

    for chat\_id in context.bot\_data['subscribers']:

        try:

            country\_code = context.dispatcher.chat\_data[chat\_id].get('country', None)

            text = get\_status\_report(country\_code=country\_code) # TODO always English

            context.bot.send\_message(chat\_id=chat\_id, text=text, parse\_mode=ParseMode.MARKDOWN)

            count+=1

            sleep(0.05) # try to avoid flood limits

        except Exception as ex:

            # remove user from subscribers if he blocked or kicked the bot

            if isinstance(ex, TelegramError) and ex.message.startswith("Forbidden: "):

                context.bot\_data['subscribers'].remove(chat\_id)

            logger.error("Failed to send daily notification to {}".format(chat\_id), exc\_info=True)

    logger.info("Successfully sent daily notification to {} users.".format(count))

def error(update, context):

    try:

        raise context.error

    except TelegramError:

        logger.warning('Update {} caused error "{}"'.format(update, context.error))

def main(config):

    persistence = PicklePersistence("database.pkl")

    updater = Updater(config['token'], persistence=persistence, use\_context=True)

    # add commands

    dp = updater.dispatcher

    dp.add\_handler(CommandHandler("start", command\_start))

    dp.add\_handler(CommandHandler("help", command\_help))

    dp.add\_handler(CommandHandler("faqs", command\_faqs))

    dp.add\_handler(CommandHandler("donate", command\_donate))

    dp.add\_handler(CommandHandler("today", command\_today))

    dp.add\_handler(CommandHandler("world", command\_world))

    dp.add\_handler(CommandHandler("list", command\_list))

    # callbacks for page buttons in list

    dp.add\_handler(CallbackQueryHandler(callback\_list\_pages, pattern=r"list (-?\d+) (\d+)"))

    dp.add\_handler(CallbackQueryHandler(callback\_list\_order\_menu, pattern=r"list\_order\_menu (\d+) \(([\d\s]+)\)"))

    dp.add\_handler(CallbackQueryHandler(callback\_list\_order, pattern=r"list\_order (\w+) (\d+)"))

    # for every country, add a command for the iso2 and iso3 codes and the name

    for iso, country in api.countries.items():

        callback = lambda update, context, code=iso: command\_country(update, context, code)

        dp.add\_handler(CommandHandler(iso, callback))

        if country['iso3']:

            dp.add\_handler(CommandHandler(country['iso3'], callback))

        name\_normal = re.sub(r"[^a-z]", "\_", country['name'].lower())

        dp.add\_handler(CommandHandler(name\_normal, callback))

    # country detailed button callback

    dp.add\_handler(CallbackQueryHandler(callback\_stats, pattern=r"stats (\w+) (\d)"))

    # set country (this has to be added before the free text handler)

    dp.add\_handler(ConversationHandler(

        entry\_points=[CommandHandler("setcountry", handle\_setcountry\_start)],

        states={

            1: [MessageHandler(Filters.text & ~Filters.command, handle\_setcountry\_input)]

        },

        fallbacks=[CommandHandler("cancel", handle\_setcountry\_cancel)],

        conversation\_timeout=60\*10 # = 10 minutes

    ))

    # subscription

    dp.add\_handler(CommandHandler("subscribe", command\_subscribe))

    dp.add\_handler(CommandHandler("unsubscribe", command\_unsubscribe))

    # subscription job

    job\_queue = updater.job\_queue

    if 'notify\_time' in config:

        job\_queue.run\_daily(run\_notify, datetime.strptime(config['notify\_time'], '%H:%M').time())

    # free text input

    dp.add\_handler(MessageHandler(Filters.text & ~Filters.command, handle\_text))

    dp.add\_handler(InlineQueryHandler(handle\_inlinequery))

    dp.add\_error\_handler(error)

    # start the bot

    updater.start\_polling()

    updater.idle()

if \_\_name\_\_ == "\_\_main\_\_":

    with open(CONFIG\_FILE, 'r') as f:

        config = json.load(f)

    main(config)

## B4 JSON CODE CONTAINING BOT TOKEN AND TIMER

{

    "token": "1098397543:pJm4JTpTTI\_l7o5IasqeCj7qw",

    "notify\_time": "17:00"

}