LISTING PROGRAM

1. bot.py

```
from telegram import ParseMode
from telegram.ext import Updater, CallbackQueryHandler, MessageHandler, Filters
import button
import configparser as cfg
import datetime
import logging
import response
import scraping
import os

def read_token(config_file):
        """Read token from config file"""
        parser = cfg.ConfigParser()
        parser.read(config_file)
        auth_token = parser.get("creds", "token")
        return auth_token

def error(update, context):
        """Log Errors caused by Updates"""
        logger.warning("Update '%s' caused error '%s'", update, context.error)

def reply(update, context):
        """Reply the user message"""
        message = update.message.text.lower()
        command = response.validate_message(message)

        if command:
        response.reply_message(update, context, command)
        else:
        update.message.reply_text("Perintah tidak dikenali.")

def callback_query_handler(update, context):
        """Handle callback query"""
        callback_query = update.callback_query
        callback_name = callback_query.data
        callback_reply = response.get_template(f"callback/{callback_name}.html")
        keyboard = button.create_button_from_callback(callback_name)

        if callback_name == "jadwal_callback":
        callback_query.message.edit_caption(caption=callback_reply, parse_mode=ParseMode.HTML)
        else:
        callback_query.edit_message_text(text=callback_reply, reply_markup=keyboard,
parse_mode=ParseMode.HTML)

def update_data(context):
        """Start scraping for update file"""
        scraping.start_scraping()

def main():
        """Start the bot"""
        token = read_token("config.cfg")
        # token = os.environ.get("TOKEN")
        updater = Updater(token)
```

```
    # Dispatcher for register handlers
    dispatcher = updater.dispatcher
    dispatcher.add_handler(MessageHandler(Filters.text, reply))
    dispatcher.add_handler(CallbackQueryHandler(callback_query_handler))
    dispatcher.add_error_handler(error)

    # Update data daily
    job = updater.job_queue
    job.run_daily(update_data, time=datetime.time(hour=00, minute=00, second=00))

    # Start bot
    updater.start_polling()
    print("Bot enabled... press CTRL+C to disabled")
    # Run until the process receives SIGINT, SIGTERM or SIGABRT
    updater.idle()

if __name__ == "__main__":
    # Enable logging
    logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
    )
    logger = logging.getLogger(__name__)

    main()
```

2. button.py

```
from telegram import InlineKeyboardButton, InlineKeyboardMarkup
import yaml

def get_calendar_download_link():
    yaml_path = "scrape_files/data/kalendar.yaml"
    data = yaml.load(open(yaml_path), Loader=yaml.FullLoader)
    calendar_url = data["url_file"]
    return calendar_url

def create_button(command, from_callback=False):
    template = {
    "start":        [("Lihat Dokumentasi","https://github.com/elmoallistair/gunadarma-telegram-
bot/blob/main/README.md"),
                    ("Lihat Source Code","https://github.com/elmoallistair/gunadarma-telegram-bot")],
    "help":         [("Lihat Dokumentasi","https://github.com/elmoallistair/gunadarma-telegram-
bot/blob/main/README.md")],
    "kalender":     [("Simpan sebagai PDF", f'{get_calendar_download_link()}')],
    "berita":       [("Kunjungi Berita BAAK", "https://berita.gunadarma.ac.id")],
    "jadwal":       [("Cara Membaca Jadwal", "")],
    "cuti":         [("Ketentuan dan Prosedur", ""),
                    ("  Formulir Cuti Akademik", "https://baak.gunadarma.ac.id/public/file/Administrasi
%20Akademik/F-CUTI%20New.doc")],
    "cek_nilai":    [("Ketentuan dan Prosedur", "")],
    "non_aktif":    [("Ketentuan dan Prosedur", ""),
                    ("  Formulir Tidak Aktif Kuliah", "https://baak.gunadarma.ac.id/public/file/Administrasi
%20Akademik/Formulir%20NonAktif.pdf")],
    "pindah_kelas":     [("Ketentuan dan Prosedur", ""),
                    ("  Formulir Pindah Kelas", "https://baak.gunadarma.ac.id/public/file/Administrasi
%20Akademik/F-PINKEL%20New.doc")],
    "pindah_jurusan":  [("Ketentuan dan Prosedur", ""),
                    ("  Formulir Pindah Jurusan Fakultas Ekonomi",
"https://baak.gunadarma.ac.id/public/file/Administrasi%20Akademik/pindah_jurusan_ekonomi.pdf"),
```

```
                    ("  Formulir Pindah Jurusan Fakultas Ilkom",
"https://baak.gunadarma.ac.id/public/file/Administrasi%20Akademik/pindah_jurusan_ilkom.pdf"),
                    ("  Formulir Pindah Jurusan Fakultas Teknologi Industri]",
"https://baak.gunadarma.ac.id/public/file/Administrasi%20Akademik/pindah_jurusan_ti.pdf")],
        "loker":            [("Kunjungi Career Center", "http://career.gunadarma.ac.id/")]
        }

        keyboard = []
        if command in template.keys():
        for i, (text,url) in enumerate(template[command]):
        keyboard.append([InlineKeyboardButton(text=text, callback_data=f"{command}_callback",
url=url)])
        return InlineKeyboardMarkup(keyboard)
        return None

def create_button_from_callback(callback_name):
        template = {
        "cuti_callback":    [("  Formulir Cuti Akademik",
"https://baak.gunadarma.ac.id/public/file/Administrasi%20Akademik/F-CUTI%20New.doc")],
        "non_aktif_callback":        [("  Formulir Tidak Aktif Kuliah",
"https://baak.gunadarma.ac.id/public/file/Administrasi%20Akademik/Formulir%20NonAktif.pdf")],
        "pindah_kelas_callback":    [("  Formulir Pindah Kelas",
"https://baak.gunadarma.ac.id/public/file/Administrasi%20Akademik/F-PINKEL%20New.doc")],
        "pindah_jurusan_callback":  [("  Formulir Pindah Jurusan Fakultas Ekonomi",
"https://baak.gunadarma.ac.id/public/file/Administrasi%20Akademik/pindah_jurusan_ekonomi.pdf"),
                    ("  Formulir Pindah Jurusan Fakultas Ilkom",
"https://baak.gunadarma.ac.id/public/file/Administrasi%20Akademik/pindah_jurusan_ilkom.pdf"),
                    ("  Formulir Pindah Jurusan Fakultas Teknologi Industri]",
"https://baak.gunadarma.ac.id/public/file/Administrasi%20Akademik/pindah_jurusan_ti.pdf")]
        }

        keyboard = []
        if callback_name in template.keys():
        for i, (text,url) in enumerate(template[callback_name]):
        keyboard.append([InlineKeyboardButton(text=text, callback_data=callback_name, url=url)])
        return InlineKeyboardMarkup(keyboard)
        return None
```

3. requirements.txt

```
selenium==3.141.0
pyaml==20.4.0
python-telegram-bot==13.4.1
```

4. response.txt

```
from collections import OrderedDict
from datetime import datetime as dt
from operator import getitem
from pathlib import Path
from telegram import ParseMode
from textwrap import dedent
import scraping
import button
import yaml

def validate_message(message):
        """Check and return command if message contains valid command"""
        list_of_commands = ["/start", "/help", "/kalender", "/berita", "/jadwal",
                "/jam", "/cuti", "/non_aktif", "/cek_nilai",
                "/pindah_kelas", "/pindah_jurusan", "/loker"]
```

```
            for word in message.split():
            if word in list_of_commands:
            return word[1:]
            return None


def load_data(command):
            """Load data from yaml file"""
            yaml_path = f"scrape_files/data/{command}.yaml"
            data = yaml.load(open(yaml_path), Loader=yaml.FullLoader)
            return data

def sort_data(data):
            """Sort data in dict by date in descending order"""
            key = lambda x: dt.strptime(getitem(x[1], "date"), "%d/%m/%Y")
            sorted_data = OrderedDict(sorted(data.items(), key=key, reverse=True))
            return sorted_data

def get_template(path):
            """Get reply template from html file"""
            file_path = f"response_templates/{path}"
            message_template = Path(file_path).read_text()
            return message_template

def send_image(context, chat_id, image_path, keyboard=None, caption=None):
            """Send image to user"""
            image = open(image_path, "rb")
            x = context.bot.sendPhoto(chat_id=chat_id,
                        photo=image,
                        reply_markup=keyboard,
                        caption=caption,
                        parse_mode=ParseMode.HTML)
            print("----------\n", x, "----------\n")

def send_text(update, text, keyboard=None):
            """Send text to user"""
            x = update.message.reply_text(text=dedent(text),
                        reply_markup=keyboard,
                        parse_mode=ParseMode.HTML,
                        disable_web_page_preview=True)
            print("----------\n", x, "----------\n")

def reply_message(update, context, command):
            """Creating reply to user"""
            chat_id = update.message.chat.id

            # Get response text template
            try:
            text = get_template(f"command/{command}.html")
            except:
            text = None

            # Create button based on command
            try:
            keyboard = button.create_button(command)
            except:
            keyboard = None

            # Create and send reply for specific command
            if command == "kalender":
            caption, image_path, _ = load_data(command).values()
```

```
send_image(context, chat_id, image_path, keyboard, f"🗓 {caption}")
elif command == "jam":
image_path = 'response_templates/command/jam.png'
send_image(context, chat_id, image_path)
elif command == "loker":
data = load_data(command)
content = ""
sorted_data = sort_data(data)
for id in sorted_data:
date, title, url = sorted_data[id].values()
content += f"🔖 <b>({date}) <a href='{url}'>{title}</a></b>\n"
text = text.format(content)
elif command == "jadwal":
keyboard = None
message = update.message.text.lower()
try:
message_split = message.split(" ", 1) # Split command and query
cmd, query = message_split
assert cmd == "/jadwal" # Must be in '/jadwal [KELAS_ATAU_DOSEN]' format
try:
assert len(query) >= 5 # Query must be 5 characters long
# Scraping data
try:
send_text(update, f"Mencari jadwal untuk input: <b>{query}</b> ...")
data = scraping.scraping_jadwal_kuliah(query)
if data: # Successful scraping
        img_path, caption = data
        keyboard = button.create_button(command)
        send_image(context, chat_id, img_path, keyboard, dedent(caption))
else: # Query not found
        text = get_template("error/jadwal_not_found.html").format(query)
except Exception as err: # Failed scraping data
text = get_template("error/jadwal_failed_scraping.html").format(err)
except: # Query length < 5
text = get_template("error/jadwal_short_query.html")
except: # Wrong format
text = get_template("error/jadwal_wrong_format.html")
elif command == "berita":
data = load_data(command)
message = update.message.text.lower()
try:
# Send news content by id
message_split = message.split(" ", 1) # Split command and query
cmd, query = message_split
if cmd == "/berita": # Must be in '/berita [ID]' format
try:
content, _, title, _ = data[query].values()
text = f"<b>{title.upper()}</b>\n\n{content}"
except: # ID not found
keyboard = None
text = get_template("error/berita_not_found.html").format(query)
else: # Wrong format
keyboard = None
text = get_template("error/berita_wrong_format.html")
except:
# Send list of news
content = ""
sorted_data = sort_data(data)
for id in sorted_data:
_, date, title, url = sorted_data[id].values()
content += f"🔖 <b>({date} - {id}) <a href='{url}'>{title}</a></b>\n"
```

```
                text = text.format(content)

                if text:
                send_text(update, text, keyboard)
```

5. scraping_metadata.yaml

```
kalender:
 url: "https://baak.gunadarma.ac.id"
 xpath:
                title: "//div[@class='cell-sm-6 cell-md-6']/h3"
                table: "//table[@class='table table-custom table-primary bordered-table table-striped table-fixed
stacktable large-only']"
                file_url: "//p[@class='text-primary']/a"
jadwal_kuliah:
 url: "https://baak.gunadarma.ac.id/jadwal/cariJadKul"
 xpath:
                form_input: "//input[@class='form-search-input form-control']"
                form_submit: "//button[@class='form-search-submit']"
                table: "//table[@class='table table-custom table-primary table-fixed bordered-table stacktable large-
only']"
                title: "//h3[@class='veil reveal-sm-block']"
                valid_from: "//p[@class='text-md-left']"
berita:
 url: "https://baak.gunadarma.ac.id/berita"
 xpath:
                title_and_url: "//div[@class='post-news-body']/h6/a"
                date: "//span[@class='text-middle inset-left-10 text-italic text-black']"
                page_content: "//div[@class='offset-top-30']"
loker:
 url: "http://career.gunadarma.ac.id/"
 xpath:
                title_and_url: "//div[@class='views-field views-field-title']/span/a"
```

6. scraping.py

```
from selenium import webdriver
import os
import re
import yaml

def set_driver():
                """Preparing chromedriver"""
                options = webdriver.ChromeOptions()
                options.add_argument("--headless")
                options.add_argument("--disable-dev-shm-usage")
                options.add_argument("--no-sandbox")
                options.binary_location = os.environ.get("GOOGLE_CHROME_BIN")

                driver = webdriver.Chrome(executable_path=os.environ.get("CHROMEDRIVER_PATH"),
options=options)
                # driver = webdriver.Chrome(options=options)
                driver.set_window_size(1920, 1080)
                return driver

def open_website(driver, url):
                """Open the website"""
                driver.get(url)
                if driver.title == "":
                raise AssertionError("Error accessing '{url}'")
                return driver
```

```
def get_metadata(key):
        """Get url and xpath location from yaml file"""
        data = yaml.load(open("scraping_metadata.yaml"), Loader=yaml.FullLoader)
        return data[key].values()

def write_to_yaml(data, filename):
        """Save scraping result to yaml file"""
        yaml_path = f"scrape_files/data/{filename}.yaml"
        with open(yaml_path, "w") as yaml_file:
        yaml.dump(data, yaml_file)

def screenshot_element(driver, element, img_path):
        """Screenshot element from web page"""
        driver.execute_script("window.scrollTo(0, 475)") # scroll page
        screenshot = element.screenshot_as_png
        with open(img_path, "wb") as file:
        file.write(screenshot)

def scraping_kalendar_akademik():
        driver = set_driver()
        url, xpath = get_metadata("kalendar")
        try:
        open_website(driver, url)
        caption = driver.find_element_by_xpath(xpath["title"]).text
        table = driver.find_element_by_xpath(xpath["table"])
        url_file = driver.find_element_by_xpath(xpath["file_url"]).get_attribute("href")
        img_path = "scrape_files/img/kalendar_akademik.png"
        data = {"caption":f"<b>{caption}</b>",
        "img_path":img_path,
        "url_file":url_file}
        screenshot_element(driver, table, img_path)
        write_to_yaml(data, "kalendar")
        print("Successfull scraping 'kalendar'")
        driver.quit()
        except Exception as e:
        print(f"Failed scraping 'kalendar': ({e})")


def scraping_jadwal_kuliah(class_or_lecturer):
        driver = set_driver()
        url, xpath = get_metadata("jadwal_kuliah")
        try:
        open_website(driver, url)
        form_input = driver.find_element_by_xpath(xpath["form_input"])
        form_submit = driver.find_element_by_xpath(xpath["form_submit"])
        form_input.send_keys(class_or_lecturer)
        form_submit.click()
        try:
        table = driver.find_element_by_xpath(xpath["table"])
        except:
        return None
        title = driver.find_elements_by_xpath(xpath["title"])[0].text
        valid_from = driver.find_element_by_xpath(xpath["valid_from"]).text
        filename = re.sub(r'[ /]', '_', title).lower()
        caption = f"<b>{title}</b>\n\nUntuk Input : <b>{class_or_lecturer.upper()}</b>\n{valid_from}"
        img_path = f"scrape_files/img/{class_or_lecturer.replace(' ','_')}_jadwal.png"
        driver.execute_script("window.scrollTo(0, 475)")
        screenshot_element(driver, table, img_path)
        driver.quit()
        return img_path, caption
```

```
        except Exception as e:
        print(f"Failed scraping 'jadwal_kuliah': {e}")
        print(f"err: {e}")

def scraping_berita():
        driver = set_driver()
        url, xpath = get_metadata("berita")
        try:
        open_website(driver, url)
        title_url = driver.find_elements_by_xpath(xpath["title_and_url"])
        date = driver.find_elements_by_xpath(xpath["date"])
        post_title = [post.text for post in title_url]
        post_url = [post.get_attribute("href") for post in title_url]
        post_id = [re.search("berita/(\d+)", post).group(1) for post in post_url]
        post_date = [post.text for post in date]
        post_content = []
        for url in post_url: # scrape every post
        driver.get(url)
        page_content = driver.find_elements_by_xpath(xpath["page_content"])
        content = page_content[0].text
        content = re.sub("[ \-\w\(\)\d]+(?:.doc|.pdf)", "", content).strip()
        post_content.append(content)
        contents = zip(post_id, post_title, post_url, post_date, post_content)
        data = {}
        for id, title, url, date, content in contents:
        data[id] = {"title":title, "url":url, "date":date, "content":content}
        write_to_yaml(data, "berita")
        driver.quit()
        print("Successfull scraping 'berita'")
        except Exception as e:
        print(f"Failed scraping 'berita': {e}")

def scraping_loker():
        driver = set_driver()
        url, xpath = get_metadata("loker")
        try:
        open_website(driver, url)
        elements = driver.find_elements_by_xpath(xpath["title_and_url"])
        post_title = [element.text for element in elements]
        post_url = [element.get_attribute("href") for element in elements]
        post_id = [re.search("node/(\d+)", url).group(1) for url in post_url]
        post_date = []
        for url in post_url: # scrape every post
        driver.get(url)
        elements = driver.find_elements_by_xpath("//span[@class='meta submitted']")
        date_posted = re.search("\d{2}/\d{2}/\d{4}", elements[0].text)[0]
        post_date.append(date_posted)
        contents = zip(post_id, post_title, post_url, post_date)
        data = {}
        for id, title, url, date in contents:
        data[id] = {"date":date, "title":title, "url":url}
        write_to_yaml(data, 'loker')
        print("Successfull scraping 'loker'")
        driver.quit()
        except Exception as e:
        print(f"Failed scraping 'loker': \n{e}")

def update_all():
        """Update all data"""
        scraping_berita()
        scraping_loker()
```

```
        scraping_kalendar_akademik()

if __name__ == "__main__":
        print("Testing scraping...")
        update_all()
        scraping_jadwal_kuliah("3ka17")
```