Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №5

Технології розроблення програмного забезпечення

ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Варіант 24

Виконала:
студентка групи ІА-14
Шеліхова А.О

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема: ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.

2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

3. Застосування одного з розглянутих шаблонів при реалізації програми.

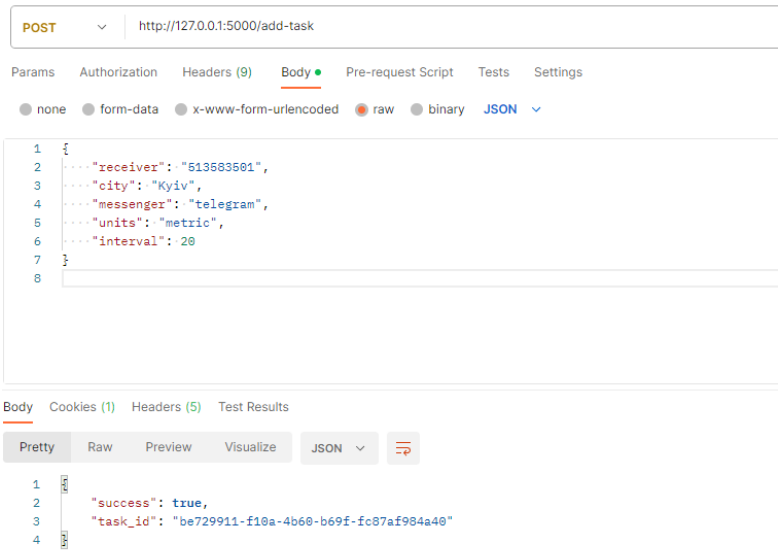
Виконання роботи:

У ході роботи було реалізовано API для створення, редагування та видалення завдань для регулярної відправки повідомлень про погоду у вибраному місті у вибраний мессенджер.

API має три endpoint-и:

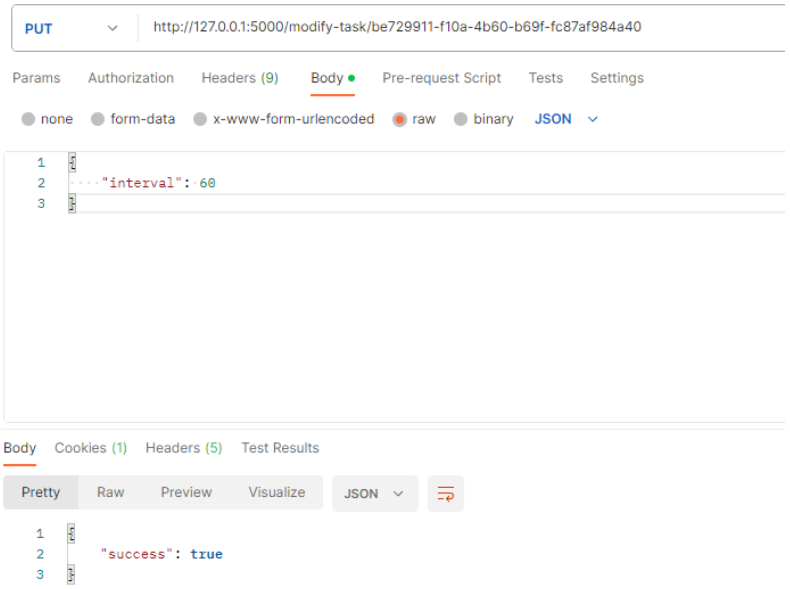
POST /add-task:

Додає завдання для відправки повідомлень. Повертає id завдання.



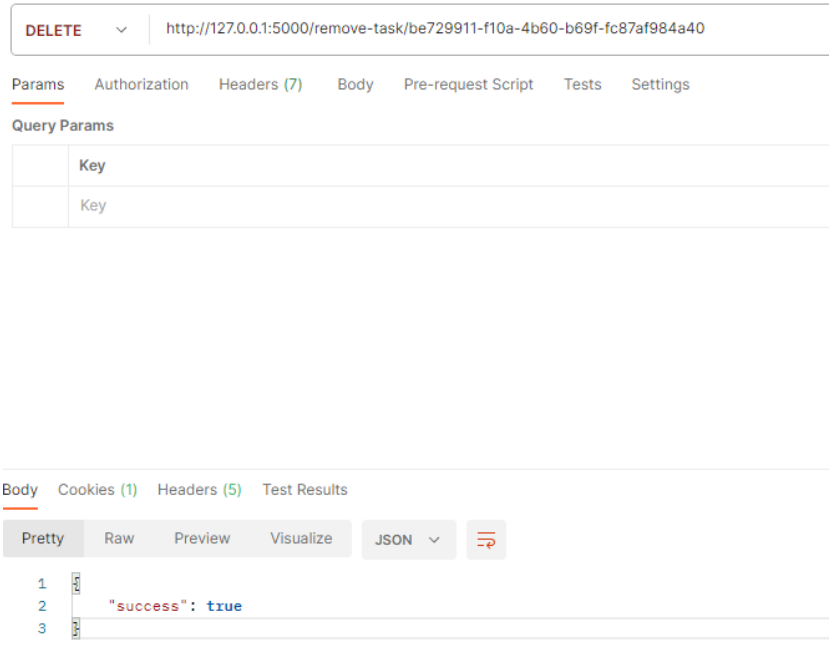
POST /modify-task:

Редагує інтервал для завдання по указаному id.



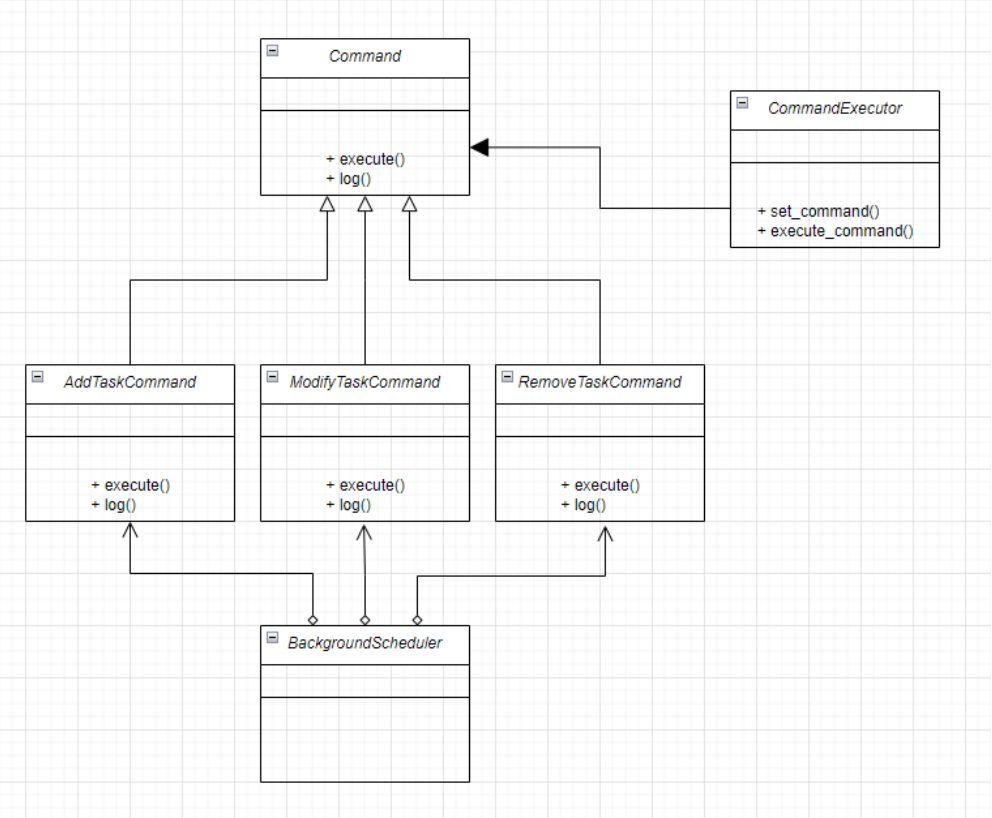
DELETE /remove-task:

Видаляє завдання по указаному id.



Виконання команд реалізовано за допомогою шаблону “Command”.

Шаблон “Command”:



Код програми:

Код, який реалізує шаблон “Command”:

```
import uuid

from datetime import datetime as dt
from abc import ABC, abstractmethod

from messengers import MESSENGERS
from utils import get_weather
from config import API_KEYS, OPENWEATHERAPP_API_KEY, WEATHER_UNITS


class Command(ABC):

    @abstractmethod
    def execute(self):
        pass

    @abstractmethod
    def log(self):
        pass


class AddTaskCommand(Command):

    def __init__(self, scheduler, receiver, city, messenger, units, interval):

        self.__scheduler = scheduler

        self.__receiver = receiver
        self.__city = city
        self.__messenger = messenger.lower()
        self.__units = units.lower()
        self.__interval = interval
```

```
        self.__success = None
        self.__error_message = None

    def execute(self):

        try:

            if self.__messenger not in API_KEYS:
                raise Exception(f"Invalid messenger \"{self.__messenger}\"")

            if self.__units not in WEATHER_UNITS:
                raise Exception(f"Invalid units \"{self.__units}\"")

            def function():

                weather_result = get_weather(OPENWEATHERAPP_API_KEY, self.__city, self.__units)

                if not weather_result['success']:
                    print(f"[{dt.now().strftime('%d-%m-%Y %H:%M:%S')}] {weather_result['error']}")
                    return

                weather = weather_result['weather']

                message = f"Weather in {self.__city.title()}:\n\n" \
                    f"Temperature: {weather['temperature']}
{WEATHER_UNITS[self.__units]['temperature']}\n" \
                    f"Pressure: {weather['pressure']} hPa\n" \
                    f"Humidity: {weather['humidity']} %\n" \
                    f"Wind Speed: {weather['wind_speed']} {WEATHER_UNITS[self.__units]['wind_speed']}"

                success = MESSENGERS[self.__messenger].send_message(API_KEYS[self.__messenger],
self.__receiver, message)

                print(f"[{dt.now().strftime('%d-%m-%Y %H:%M:%S')}] {'Success' if success else
'Failed'}.")

            task_id = str(uuid.uuid4())

            self.__scheduler.add_job(func=function, trigger='interval', seconds=self.__interval,
id=task_id)

            self.__success = True

            return {
                'success': self.__success,
                'task_id': task_id
            }

        except Exception as exception:

            self.__success = False

            self.__error_message = f"{type(exception).__name__}: {exception}"

            return {
                'success': self.__success,
                'error': {
                    'type': type(exception).__name__,
                    'message': str(exception)
                }
            }

    def log(self):

        if self.__success:
            print(f"Task added successfully for {self.__city.title()} every {self.__interval} seconds")

        elif not self.__success:
            print(f"Task not added for {self.__city.title()}. Error: {self.__error_message}")

        else:
            print("Add task command not executed")
```

```
class ModifyTaskCommand(Command):

    def __init__(self, scheduler, task_id, interval):

        self.__scheduler = scheduler

        self.__task_id = task_id
        self.__interval = interval

        self.__success = None
        self.__error_message = None

    def execute(self):

        try:

            self.__scheduler.reschedule_job(self.__task_id, trigger='interval', seconds=self.__interval)

            self.__success = True

            return {
                'success': self.__success
            }

        except Exception as exception:

            self.__success = False

            self.__error_message = f"{type(exception).__name__}: {exception}"

            return {
                'success': self.__success,
                'error': {
                    'type': type(exception).__name__,
                    'message': str(exception)
                }
            }

    def log(self):

        if self.__success:
            print(f"Task {self.__task_id} modified successfully")

        elif not self.__success:
            print(f"Task {self.__task_id} not modified. Error: {self.__error_message}")

        else:
            print("Modify task command not executed")


class RemoveTaskCommand(Command):

    def __init__(self, scheduler, task_id):

        self.__scheduler = scheduler

        self.__task_id = task_id

        self.__success = None
        self.__error_message = None

    def execute(self):

        try:

            self.__scheduler.remove_job(self.__task_id)

            self.__success = True

            return {
                'success': self.__success
            }

        except Exception as exception:
```

```
            self.__success = False

            self.__error_message = f"{type(exception).__name__}: {exception}"

            return {
                'success': self.__success,
                'error': {
                    'type': type(exception).__name__,
                    'message': str(exception)
                }
            }

    def log(self):

        if self.__success:
            print(f"Task {self.__task_id} removed successfully")

        elif not self.__success:
            print(f"Task {self.__task_id} not removed. Error: {self.__error_message}")

        else:
            print("Remove task command not executed")


class CommandExecutor:

    def set_command(self, command):
        self.__command = command

    def execute_command(self):

        result = self.__command.execute()

        self.__command.log()

        return result
```

Головний код програми:

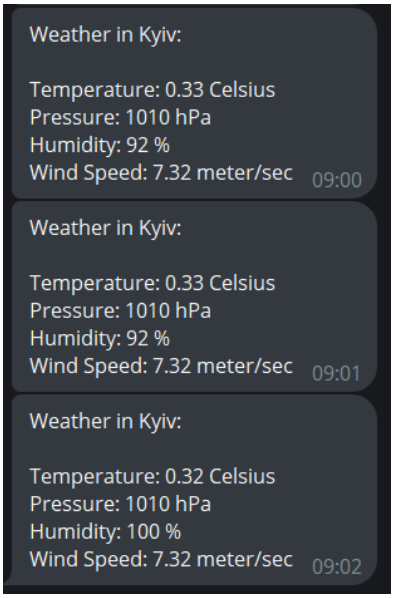
```
from flask import Flask, jsonify, request

from apscheduler.schedulers.background import BackgroundScheduler

from commands import AddTaskCommand, ModifyTaskCommand, RemoveTaskCommand, CommandExecutor


app = Flask(__name__)


scheduler = BackgroundScheduler()

scheduler.start()

command_executor = CommandExecutor()


@app.route('/')
def index():
    return jsonify({
        'success': True,
        'message': "This is application for managing tasks"
    })


@app.route('/add-task', methods=['POST'])
def add_task():

    try:

        data = request.get_json()
```

```
        for filed in ['receiver', 'city', 'messenger', 'units', 'interval']:
            if filed not in data:
                raise Exception(f"Invalid request data. Field \"{filed}\" is required.")

        receiver = data['receiver']
        city = data['city']
        messenger = data['messenger']
        units = data['units']
        interval = data['interval']

        if not isinstance(interval, int):
            raise TypeError(f"Invalid request data. Field \"interval\" must be integer.")

        if interval <= 0:
            raise ValueError(f"Invalid request data. Field \"interval\" must be positive integer.")

        command = AddTaskCommand(scheduler, receiver, city, messenger, units, interval)

        command_executor.set_command(command)

        result = command_executor.execute_command()

        return jsonify(result)

    except Exception as exception:
        return jsonify({
            'success': False,
            'error': {
                'type': type(exception).__name__,
                'message': str(exception)
            }
        })


@app.route('/modify-task/<task_id>', methods=['PUT'])
def modify_task(task_id):

    try:

        data = request.get_json()

        for filed in ['interval']:
            if filed not in data:
                raise Exception(f"Invalid request data. Field \"{filed}\" is required.")

        interval = data['interval']

        if not isinstance(interval, int):
            raise TypeError(f"Invalid request data. Field \"interval\" must be integer.")

        if interval <= 0:
            raise ValueError(f"Invalid request data. Field \"interval\" must be positive integer.")

        command = ModifyTaskCommand(scheduler, task_id, interval)

        command_executor.set_command(command)

        result = command_executor.execute_command()

        return jsonify(result)

    except Exception as exception:
        return jsonify({
            'success': False,
            'error': {
                'type': type(exception).__name__,
                'message': str(exception)
            }
        })


@app.route('/remove-task/<task_id>', methods=['DELETE'])
def remove_task(task_id):
```

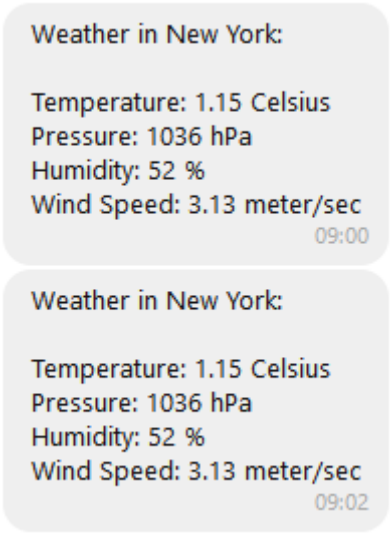
```
    try:

        command = RemoveTaskCommand(scheduler, task_id)

        command_executor.set_command(command)

        result = command_executor.execute_command()

        return jsonify(result)

    except Exception as exception:
        return jsonify({
            'success': False,
            'error': {
                'type': type(exception).__name__,
                'message': str(exception)
            }
        })


if __name__ == '__main__':
    app.run()
```

Результати:

Погода в Києві у одиницях вимірювання metric (Telegram) кожні 60 секунд:



Погода в Нью Йорку у одиницях вимірювання imperial (Viber) кожні 120 секунд:



Висновки:

При виконанні даної лабораторної роботи було вивчено шаблон проектування «Command». У ході роботи було реалізовано API для додавання, редагування та видалення завдань для відправки повідомлень про погоду у вибраний мессенджер. Виконання команд реалізовано за допомогою шаблону “Command”.