Московский государственный технический университет им. Н.Э. Баумана

Факультет «Радиотехнический»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчёт по домашнему заданию.

Выполнил:
студент группы РТ5-31Б
Агеев Алексей
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва, 2021 г

Описание задания

1. Модифицируйте код лабораторной работы №6 таким образом, чтобы он был пригоден для модульного тестирования.
2. Используя материалы лабораторной работы №4 создайте модульные тесты с применением TDD - фреймворка (2 теста) и BDD - фреймворка (2 теста).

Текст программы

Bot.py

```
import logging
from aiogram.dispatcher import storage
from aiogram.dispatcher.filters import state

from tkinter import font

from Crypto import crypto_rate_request
from course import valute_rate_request
from weather import weather_request

from aiogram.dispatcher import FSMContext
from aiogram import Bot, Dispatcher, executor, types
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from aiogram.types import KeyboardButton, ReplyKeyboardMarkup,
InlineKeyboardMarkup


from aiogram.dispatcher.filters.state import State, StatesGroup


API_TOKEN = “”

# Configure logging
logging.basicConfig(level=logging.INFO)

# Initialize bot and dispatcher
bot = Bot(token=API_TOKEN)
dp = Dispatcher(bot, storage=MemoryStorage())

"""Приветствие ////////////////////////////////////////////////"""

greeting = "Хай! \U0001F44B"

greet_but = KeyboardButton(greeting)

greet_kb = ReplyKeyboardMarkup(resize_keyboard=True, one_time_keyboard=True)
greet_kb.add(greet_but)

@dp.message_handler(commands='start')
```

```python
async def send_welcome(message: types.Message):
    """
    This handler will be called when user sends `/start` command
    """
    await message.answer("Привет!", reply_markup=greet_kb)
    await OrderDir.wait_funcs.set()

"""Функции бота ////////////////////////////////////////////"""
valute_course = "Курс доллара \U0001F4B2"
weather = "Погода в Москве ☂"
crypto_course = "Курс криптовалют"

funcs = [valute_course, weather, crypto_course]

funcs_but = [KeyboardButton(i) for i in funcs]

funcs_kb = ReplyKeyboardMarkup(resize_keyboard=True)

class OrderDir(StatesGroup):
    wait_funcs = State()
    funcs_chosen = State()
    wait_crypto_parameter = State()
    wait_crypto_count = State()

for temp in funcs_but:
    funcs_kb.row(temp)

@dp.message_handler(state = OrderDir.wait_funcs)
async def send_bot_functions(message: types.Message):
    """
    Handler after greeting
    """
    await OrderDir.funcs_chosen.set()
    await message.answer("Выберите необходимую функцию", reply_markup=funcs_kb)

@dp.message_handler(regexp=valute_course, state = OrderDir.funcs_chosen) #Вызов
списка функций бота после запроса курса
async def valute_course(message: types.Message):
    """
    Handler after response for valute_course
    """
    await OrderDir.funcs_chosen.set()
    await message.answer(valute_rate_request(), reply_markup=funcs_kb)

@dp.message_handler(regexp=weather, state = OrderDir.funcs_chosen) #Вызов списка
функций бота после запроса курса
async def weather(message: types.Message):
    """
    Handler after response for weather
    """
    await OrderDir.funcs_chosen.set()
    await message.answer(weather_request(), reply_markup=funcs_kb)
```

```
"""//////////////////////////////////////////////////////////"""


crypto_parameters_var = {'По капитализации' : 'market_cap', 'По росту за последние
24 часа' : 'percent_change_24h', 'По стоимости' : 'price'}

crypto_parameters_kb = ReplyKeyboardMarkup()

crypto_parameters_but = [KeyboardButton(temp) for temp in crypto_parameters_var]

crypto_parameters_greet = "Выберите по какому параметру отобрать криптовалюты"

crypto_parameters_warning = "Введите корректный параметр"

for temp in crypto_parameters_but:
    crypto_parameters_kb.row(temp)

@dp.message_handler(regexp=crypto_course, state = OrderDir.funcs_chosen)
async def crypto_parameters(message: types.Message):
    """
    Handler after response for crypto_course
    """
    await message.answer(crypto_parameters_greet,
reply_markup=crypto_parameters_kb)
    await OrderDir.wait_crypto_parameter.set()

crypto_count_var = ['3', '5', '10']

crypto_count_kb = ReplyKeyboardMarkup()

crypto_count_but = [KeyboardButton(temp) for temp in crypto_count_var]

crypto_count_greet = "Выберите количество криптовалют"

crypto_count_warning = "Введите корректное значение"

class crypto_parameter:
    param = "Параметр"
    def set_data(parameter):
        crypto_parameter.param = parameter
    def get_data():
        return crypto_parameter.param

for temp in crypto_count_but:
    crypto_count_kb.row(temp)

@dp.message_handler(state = OrderDir.wait_crypto_parameter)
async def crypto_count(message: types.Message):
    """
    Handler after response for crypto_parameters
    """
    if message.text not in crypto_parameters_var:
```

```
        await message.answer(crypto_parameters_warning,
reply_markup=crypto_parameters_kb)
        return
    crypto_parameter.set_data(message.text)
    await OrderDir.wait_crypto_count.set()
    await message.answer(crypto_count_greet, reply_markup=crypto_count_kb)

@dp.message_handler(state = OrderDir.wait_crypto_count)
async def crypto_corse(message: types.Message):
    """
    Handler after response for crypto_parameters
    """
    if message.text not in crypto_count_var:
        await message.answer(crypto_count_warning, reply_markup=crypto_count_kb)
        return
    await OrderDir.funcs_chosen.set()
    await
message.answer(crypto_rate_request(crypto_parameters_var[crypto_parameter.get_data
()], int(message.text)), reply_markup=funcs_kb)



"""////////////////////////////////////////////////////////////"""

if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

course.py

```
import requests
from requests.models import HTTPError

def valute_rate_request(Valute = "USD"):
    try:
        data = requests.get("https://www.cbr-xml-daily.ru/daily_json.js").json()
    except HTTPError as http_err:
        print(f"HTTP error occured: {http_err}")
        return -1
    except Exception as err:
        print(f"Exception occured: {err}")
        return -1
    Valute_data = data["Valute"][Valute]
    Valute_data = f"1 {Valute_data['CharCode']} = {Valute_data['Value']} RUB"
    return Valute_data

print(valute_rate_request("USD"))

# print("//////////////////////")

# print(rate_request("EUR"))
```

Weather.py

```
from aiogram.types.base import String
import requests
from requests.models import HTTPError

API_key = '11e360048e96c001813146fe2205703f'

def weather_request(city = 'Moscow'):
    try:
        data =
requests.get(f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_
key}&lang=ru&units=metric").json()
    except HTTPError as http_err:
        print(f"HTTP error occured: {http_err}")
        return -1
    except Exception as err:
        print(f"Exception occured: {err}")
        return -1
    weather = "{}.\nТемпература: {:.0f}\U00002103. Ощущается как:
{:.0f}\U00002103\nСкорость ветра: {:.0f}
м/с".format(data['weather'][0]['description'].capitalize(),
round(data['main']['temp'], 0), round(data['main']['feels_like'], 0),
round(data['wind']['speed'], 1))
    return weather

print(weather_request())
```

Crypto.py

```
from os import sep
from typing import Text
from warnings import resetwarnings
from requests import Request, Session
from requests.exceptions import ConnectionError, Timeout, TooManyRedirects
import json

API_KEY = "b3274e01-fd44-48db-8f86-b5d77c17a1a2"

url = 'https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest'
parameters = {
'start':'1',
'limit':'5000',
'convert':'USD'
}
headers = {
'Accepts': 'application/json',
```

```
'X-CMC_PRO_API_KEY': API_KEY,
}

def crypto_rate_request(parameter_token : str, num : int):

    session = Session()
    session.headers.update(headers)

    try:
        response = session.get(url, params=parameters)
        data = json.loads(response.text)['data']
        sort_data = []
        for temp in data:
            sort_data.append([temp['name'],temp['quote']['USD'][parameter_token]])
        sort_data = sorted(sort_data, key = lambda x: x[1], reverse=True)
        col_width = max(max(len(temp[0]),len(f"{temp[1]:0f}")) for temp in
sort_data[0:num]) + 2
    except (ConnectionError, Timeout, TooManyRedirects) as err:
        print(err)
    result = 'Название'.ljust(col_width, '\U00002003') + str(parameter_token)
    for name, value in sort_data[0:num]:
        result += '\n' + name.ljust(col_width, '\U00002003') + f"{value:0.1f}"
    print(result)
    return result


#print(crypto_rate_request('price', 3))
```

TDD.py

```
import unittest
from course import valute_rate_request
from weather import weather_request

class test(unittest.TestCase):
    def test(self):
        self.assertEqual(weather_request(),'Облачно с прояснениями.\nТемпература:
-9°C. Ощущается как: -16°C\nСкорость ветра: 6 м/с')
        self.assertEqual(valute_rate_request(),"1 USD = 73.1886 RUB")

if __name__ == "__main__":
    unittest.main()
```

Course_feat.feature

```
Feature: My first feature file using radish
    In order to test my awesome software
    I need an awesome BDD tool like radish
    to test my software.

    Scenario: get valute course
```

```
        Given telegram chatbot: 321321
        When the user searches for "valute"
        Then expect result to be 1 USD = 73.1886 RUB

     Scenario: get weather
        Given telegram chatbot: 321321
        When the user searches for "wheather"
        Then expect result to be "Облачно с прояснениями.\nТемпература: -9℃.
Ощущается как: -14℃\nСкорость ветра: 2 м/с"
```

BDD.py

```
from radish import given, when, then

import Bot

from course import valute_rate_request
from weather import weather_request

@given("telegram chatbot {API_KEY:g}")
def have_bot(step, API_KEY):
    step.context.bot_api = {API_KEY}

@when("the user searches for {valute:g}")
def exec_func(step, valute):
    if(valute == "valute"):
        step.context.func = valute_rate_request
    else:
        step.context.func = weather_request

@then("the user expects result to be {result:g}")
def expect_result(step, valute):
    assert step.context.func() == result
```