**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: | Проверил: |
| студент группы РТ5-31Б | преподаватель каф. ИУ5 |
| Агеев Алексей | Гапанюк Ю.Е. |
| Подпись и дата: | 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')

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/c".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℃. Ощущается как: -16℃\nСкорость ветра: 6 м/c')

        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/c"

**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