Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по домашнему заданию

Выполнил:
студент группы ИУ5-32Б
Еремихин Владислав

Москва, 2021 г.

Описание задания

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

Текст программы

Файл **bot.py**:

```
from resources.config import TOKEN
from resources.messages import MESSAGES
import logging
from aiogram import Bot, Dispatcher, executor, types
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from handlers.common import register_message_handlers
from handlers.callback_queries import register_callback_query_handlers


logging.basicConfig(level=logging.INFO)

storage = MemoryStorage()
bot = Bot(token=TOKEN)
dp = Dispatcher(bot, storage=storage)


if __name__ == "__main__":
    register_message_handlers(dp)
    register_callback_query_handlers(dp)
    dp.bot.set_my_commands([
        types.BotCommand("start", "Запустить бота"),
        types.BotCommand("help", "Помощь"),
        types.BotCommand("cancel", "Отменить текущую операцию")
    ])
    executor.start_polling(dp, skip_updates=True)
```

Файл **markups.py**:

```
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton

start_menu =InlineKeyboardMarkup(row_width=1)
btnStart = InlineKeyboardButton(text="☑Начать", callback_data="btnStart")

start_menu.insert(btnStart)

from_menu = InlineKeyboardMarkup(row_width=2)
btn_BTC = InlineKeyboardButton(text="₿Bitcoin", callback_data="btn_BTC")
```

```
btn_ETH = InlineKeyboardButton(text="💰Ethereum", callback_data="btn_ETH")
btn_XRP = InlineKeyboardButton(text="💸XRP", callback_data="btn_XRP")
btn_SOL = InlineKeyboardButton(text="🤑Solana", callback_data="btn_SOL")

from_menu.insert(btn_BTC)
from_menu.insert(btn_ETH)
from_menu.insert(btn_XRP)
from_menu.insert(btn_SOL)

to_menu = InlineKeyboardMarkup(row_width=2)
btn_USD = InlineKeyboardButton(text="💵USD", callback_data="btn_USD")
btn_RUB = InlineKeyboardButton(text="🇷🇺RUB", callback_data="btn_RUB")
btn_EUR = InlineKeyboardButton(text="💶EUR", callback_data="btn_EUR")
btn_UAH = InlineKeyboardButton(text="🇺🇦UAH", callback_data="btn_UAH")
btn_back_from = InlineKeyboardButton(text="Назад", callback_data="btn_back_from")

to_menu.insert(btn_USD)
to_menu.insert(btn_RUB)
to_menu.insert(btn_EUR)
to_menu.insert(btn_UAH)
to_menu.insert(btn_back_from)

choice_menu = InlineKeyboardMarkup(row_width=1)
btn_back_crypto = InlineKeyboardButton(text="Назад к выбору криптовалюты",
callback_data="btn_crypto")
btn_back_curr =  InlineKeyboardButton(text="Назад к выбору валюты для конвертации",
callback_data="btn_curr")
btn_result = InlineKeyboardButton(text="Показать результат",
callback_data="btn_result")

choice_menu.insert(btn_result)
choice_menu.insert(btn_back_crypto)
choice_menu.insert(btn_back_curr)

final_menu = InlineKeyboardMarkup(row_width=1)
btn_retry = InlineKeyboardButton(text="Обновить курс", callback_data="btn_retry")
btn_finish =  InlineKeyboardButton(text="Завершить/Начать заново",
callback_data="btn_finish")

final_menu.insert(btn_retry)
final_menu.insert(btn_finish)

help_menu = InlineKeyboardMarkup(row_width=1)
btn_accept = InlineKeyboardButton(text="Понятно", callback_data="accept")

help_menu.insert(btn_accept)
```

Файл **request.py**:

```python
from requests import Session
from requests.exceptions import ConnectionError, Timeout, TooManyRedirects
from resources.config import CRYPTO_API_TOKEN
import json


def get_exchange_rate(crypto, curr):
    url = "https://min-api.cryptocompare.com/data/price"
    parameters = {"fsym": crypto, "tsyms": curr, "api_key": CRYPTO_API_TOKEN}

    session = Session()

    try:
        response = session.get(url, params=parameters)
        data = json.loads(response.text)
        return data[curr]
    except (ConnectionError, Timeout, TooManyRedirects) as e:
        print(e)
    finally:
        session.close()
```

Файл **resources/config.py**:

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

TOKEN = "***"

CRYPTO_API_TOKEN = "***"

db_file = "resources/db.vdb"

class States(StatesGroup):
    state_start = State()
    state_from_currency = State()
    state_to_currency = State()
    state_choice = State()
    state_result = State()
```

Файл **resources/messages.py**:

```python
start_message = """
*Привет, пользователь!*

Этот бот позволяет узнать текущий курс самых популярных криптовалют
(💎_Bitcoin_, 💰_Ethereum_ и др.).

Для получения курса нажмите на кнопку "Начать" ниже:
"""

from_currency_message = """
```

```
Выберите криптовалюту нажатием кнопки ниже:
"""

err_message = """
*Нет такой команды!*
"""


to_currency_message = """
Выбранная криптовалюта: *{}*

Выберите валюту для конвертации нажатием кнопки ниже:
"""


choice_message = """
Выбранная пара для конвертации: *{}* -> *{}*

Выберите действие нажатием кнопки ниже:
"""

result_message = """
_Текущий курс:_

    *1* *{}* -> *{}*

Выберите действие нажатием кнопки ниже:
"""

cancel_message = '''
*Текущая операция отменена*

Введите /help или /start чтобы начать заново:
'''

help_message = '''
Для отмены текущей операции напишите /cancel

Для продолжения нажмите "Понятно" ниже:
'''

MESSAGES = {
    'States:state_start': start_message,
    'States:state_from_currency': from_currency_message,
    'States:state_to_currency': to_currency_message,
    'States:state_choice': choice_message,
    'States:state_result': result_message,
    'err': err_message,
    'cancel':cancel_message,
    'help': help_message
```

```
}
```

Файл **handlers/callback_queries.py**:

```
from aiogram.dispatcher.dispatcher import Dispatcher
from aiogram.dispatcher.storage import FSMContext
from resources.config import States
import dbworker
import markups
from aiogram.dispatcher.filters import Text
from aiogram import types
from resources.messages import MESSAGES
from request import get_exchange_rate
from number_format import cformat


# Обрабатываем /help
async def handle_help(call: types.CallbackQuery, state: FSMContext):
    await call.answer()
    bot = call.bot
    await bot.delete_message(call.from_user.id, call.message.message_id)

# Начинаем ввод данных (STATE_START -> STATE_FROM_CURRENCY)
async def handle_start(call: types.CallbackQuery, state: FSMContext):
    await call.answer()
    await state.finish()
    bot = call.bot
    await bot.delete_message(call.from_user.id, call.message.message_id)
    await States.state_from_currency.set()
    print(await state.get_state())
    await bot.send_message(call.from_user.id, text=MESSAGES[await
state.get_state()], reply_markup=markups.from_menu, parse_mode="Markdown")


# Выбираем криптовалюту (STATE_FROM_CURRENCY -> STATE_TO_CURRENCY)
async def handle_crypto(call: types.CallbackQuery, state: FSMContext):
    await call.answer()
    bot = call.bot
    await bot.delete_message(call.from_user.id, call.message.message_id)
    from_currency = call.data.split('_')[1]
    dbworker.set(dbworker.make_key(call.from_user.id, "FROM_CURRENCY"),
from_currency)
    print(await state.get_state())
    await States.state_to_currency.set()
    print(await state.get_state())
    await bot.send_message(
        call.from_user.id,
        text=MESSAGES[await
state.get_state()].format(dbworker.get(dbworker.make_key(call.from_user.id,
"FROM_CURRENCY"))),
```

```
        parse_mode="Markdown",
        reply_markup=markups.to_menu
    )


# Выбираем валюту (STATE_TO_CURRENCY -> STATE_CHOICE)
async def handle_curr(call: types.CallbackQuery, state: FSMContext):
    await call.answer()
    bot = call.bot
    await bot.delete_message(call.from_user.id, call.message.message_id)
    to_currency = call.data.split('_')[1]
    print(await state.get_state())
    if to_currency != "back":
        dbworker.set(dbworker.make_key(call.from_user.id, "TO_CURRENCY"),
to_currency)
        await States.state_choice.set()
        await bot.send_message(
            call.from_user.id,
            text=MESSAGES[await state.get_state()].format(
                dbworker.get(dbworker.make_key(call.from_user.id,
"FROM_CURRENCY")),
                dbworker.get(dbworker.make_key(call.from_user.id, "TO_CURRENCY"))
            ),
            reply_markup=markups.choice_menu,
            parse_mode="Markdown"
        )
    else:
        print(await state.get_state())
        await States.state_from_currency.set()
        await bot.send_message(
            call.from_user.id,
            text="Выберите криптовалюту: ",
            reply_markup=markups.from_menu,
            parse_mode="Markdown"
        )

# Выбираем действие (STATE_CHOICE -> STATE_RESULT)
async def handle_choice(call: types.CallbackQuery, state: FSMContext):
    await call.answer()
    bot = call.bot
    await bot.delete_message(call.from_user.id, call.message.message_id)
    choice = call.data.split('_')[1]
    if choice == "result":
        from_currency = dbworker.get(dbworker.make_key(call.from_user.id,
"FROM_CURRENCY"))
        to_currency = dbworker.get(dbworker.make_key(call.from_user.id,
"TO_CURRENCY"))
        result = cformat(float(get_exchange_rate(from_currency, to_currency)),
to_currency)
        await States.state_result.set()
```

```
        await bot.send_message(
            call.from_user.id,
            text=MESSAGES[await state.get_state()].format(
                from_currency,
                result
            ),
            reply_markup=markups.final_menu,
            parse_mode="Markdown"
        )
    else:
        if choice == "crypto":
            await States.state_from_currency.set()
            await bot.send_message(
                call.from_user.id,
                text=MESSAGES[await state.get_state()],
                reply_markup=markups.from_menu,
                parse_mode="Markdown"
            )
        else:
            await States.state_to_currency.set()
            await bot.send_message(
                call.from_user.id,
                text=MESSAGES[await
state.get_state()].format(dbworker.get(dbworker.make_key(call.from_user.id,
"FROM_CURRENCY"))),
                reply_markup=markups.to_menu,
                parse_mode="Markdown"
            )

# Финальный выбор
async def handle_result(call: types.CallbackQuery, state: FSMContext):
    await call.answer()
    bot = call.bot
    await bot.delete_message(call.from_user.id, call.message.message_id)
    choice = call.data.split('_')[1]
    if choice == "retry":
        from_currency = dbworker.get(dbworker.make_key(call.from_user.id,
"FROM_CURRENCY"))
        to_currency = dbworker.get(dbworker.make_key(call.from_user.id,
"TO_CURRENCY"))
        result = cformat(float(get_exchange_rate(from_currency, to_currency)),
to_currency)
        await bot.send_message(
            call.from_user.id,
            text=MESSAGES[await state.get_state()].format(
                from_currency,
                result
            ),
            reply_markup=markups.final_menu,
            parse_mode="Markdown"
```

```
        )
    else:
        await States.state_start.set()
        await bot.send_message(
            call.from_user.id,
            text=MESSAGES[await state.get_state()],
            reply_markup=markups.start_menu,
            parse_mode="Markdown"
        )
```

```
def register_callback_query_handlers(dp: Dispatcher):
    dp.register_callback_query_handler(handle_start, text="btnStart",
state=States.state_start)
    dp.register_callback_query_handler(handle_crypto, Text(startswith="btn"),
state=States.state_from_currency)
    dp.register_callback_query_handler(handle_curr, Text(startswith="btn"),
state=States.state_to_currency)
    dp.register_callback_query_handler(handle_choice, Text(startswith="btn"),
state=States.state_choice)
    dp.register_callback_query_handler(handle_result, Text(startswith="btn"),
state=States.state_result)
    dp.register_callback_query_handler(handle_help, text="accept", state='*')
```

Файл **handlers/common.py**:

```
from aiogram import types
from aiogram.dispatcher.dispatcher import Dispatcher
from aiogram.dispatcher.storage import FSMContext
from resources.config import States
import dbworker
import markups
from resources.messages import MESSAGES
from request import get_exchange_rate
import babel.numbers as bab


async def cmd_start(message: types.Message, state: FSMContext):
    await States.state_start.set()
    await message.answer(MESSAGES[await state.get_state()], parse_mode="Markdown",
reply_markup=markups.start_menu)

async def cmd_cancel(message: types.Message, state: FSMContext):
    if await state.get_state() == None:
        return
    else:
        await state.finish()
        await message.answer(MESSAGES['cancel'], parse_mode="Markdown")
```

```
async def cmd_help(message: types.Message, state: FSMContext):
    await message.answer(MESSAGES['help'], parse_mode="Markdown",
reply_markup=markups.help_menu)


async def err_handle(message: types.Message, state: FSMContext):
    print()
    print(await state.get_state())
    print()
    if await state.get_state() == None:
        return
    current_state = await state.get_state()
    reply_markup = None
    reply_msg = ''
    if current_state == 'States:state_start':
        reply_msg = MESSAGES['err'] + MESSAGES[current_state]
        reply_markup = markups.start_menu
    elif current_state == 'States:state_from_currency':
        reply_msg = MESSAGES['err'] + MESSAGES[current_state]
        reply_markup = markups.from_menu
    elif current_state == 'States:state_to_currency':
        reply_msg = MESSAGES['err'] +
MESSAGES[current_state].format(dbworker.get(dbworker.make_key(message.from_user.id,
"FROM_CURRENCY")))
        reply_markup = markups.to_menu
    elif current_state == 'States:state_choice':
        reply_msg = MESSAGES['err'] + MESSAGES[current_state].format(
            dbworker.get(dbworker.make_key(message.from_user.id, "FROM_CURRENCY")),
            dbworker.get(dbworker.make_key(message.from_user.id, "TO_CURRENCY"))
        )
        reply_markup = markups.choice_menu
    elif current_state == 'States:state_result':
        from_currency = dbworker.get(dbworker.make_key(message.from_user.id,
"FROM_CURRENCY"))
        to_currency = dbworker.get(dbworker.make_key(message.from_user.id,
"TO_CURRENCY"))
        result = bab.format_currency(float(get_exchange_rate(from_currency,
to_currency)), to_currency, locale="ru_RU")

        reply_msg = MESSAGES['err'] + MESSAGES[current_state].format(
            from_currency,
            result
        )
        reply_markup = markups.final_menu


    await message.answer(reply_msg, parse_mode="Markdown",
reply_markup=reply_markup)

def register_message_handlers(dp: Dispatcher):
```

```
    dp.register_message_handler(cmd_start, commands=['start', 'help'])
    dp.register_message_handler(cmd_cancel, commands=['cancel'], state="*")
    dp.register_message_handler(cmd_help, commands=['help'], state="*")
    dp.register_message_handler(err_handle, state="*")
```

Файл **handlers/db_worker.py**:

```
from vedis import Vedis
from resources import config


def get(key):
    with Vedis(config.db_file) as db:
        try:
            return db[key].decode()
        except KeyError:
            # в случае ошибки значение по умолчанию - начало диалога
            return config.States.S_START.value


def set(key, value):
    with Vedis(config.db_file) as db:
        try:
            db[key] = value
            return True
        except:
            return False


def make_key(chatid, keyid):
    res = str(chatid) + '__' + str(keyid)
    return res
```

Файл **tests/test_request.py**:

```
import babel.numbers as bab
import unittest
from requests import Session
from requests.exceptions import ConnectionError, Timeout, TooManyRedirects
from config import CRYPTO_API_TOKEN
import json

class TestBotUtilities(unittest.TestCase):
    def test_cformat(self):
        self.assertEqual(cformat(1000, "RUB"), "1 000,00 ₽")
        self.assertEqual(cformat(1000, "USD"), "1 000,00 $")

    def test_get_exchange_rate(self):
        self.assertGreater(get_exchange_rate("BTC", "USD"), 30000)
        self.assertLess(get_exchange_rate("BTC", "USD"), 70000)
```

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

Файл **tests/features/cformat_test.feature**:

```
Feature: Right formatting of currency

  Scenario: 1000 RUB
    Given Number is 1000, currency is RUB
    Then Result must be _1 000,00 ₽_

  Scenario: 1000 USD
    Given Number is 1000, currency is USD
    Then Result must be _1 000,00 $_
```

Файл **tests/get_exchange_rate_test.feature**:

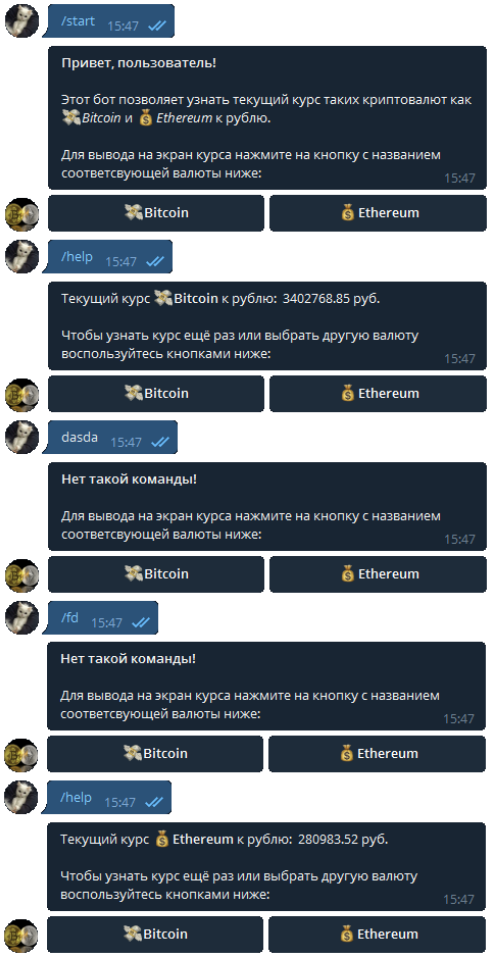
```
Feature: Right Currency Choosing

  Scenario: XRP TO RUB
    Given Crypto currency is XRP, to currency is RUB
    Then Result must be between 60 and 90

  Scenario: BTC TO USD
    Given Crypto currency is BTC, to currency is USD
    Then Result must be between 30000 and 60000
```

Файл **tests/features/steps/test_bot_utils_bdd.py**:

```
from behave import given, then
import babel.numbers as bab
from requests import Session
from requests.exceptions import ConnectionError, Timeout, TooManyRedirects
from config import CRYPTO_API_TOKEN
import json


@given("Crypto currency is {crypto}, to currency is {curr}")
def have_convert_params(context, crypto, curr):
    context.crypto = crypto
    context.curr = curr


@then("Result must be between {from_curr} and {to}")
def expect_result(context, from_curr, to):
    assert get_exchange_rate(context.crypto, context.curr) > float(from_curr)
    assert get_exchange_rate(context.crypto, context.curr) < float(to)
```

```python
@given("Number is {number}, currency is {curr}")
def have_convert_params(context, number, curr):
    context.number = number
    context.curr = curr


@then("Result must be _{result}_")
def expect_result(context, result):
    assert cformat(float(context.number), context.curr) == result
```

Пример выполнения программы



Результаты тестов:

```
=============
TDD
=============

..
----------------------------------------------------------------------
Ran 2 tests in 4.052s

OK


=============
BDD
=============

Feature: Right formatting of currency # features/cformat_test.feature:1

  Scenario: 1000 RUB                     # features/cformat_test.feature:3
    Given Number is 1000, currency is RUB # features/steps/test_bot_utils_bdd.py:40
    Then Result must be _1 000,00 ₽_      # features/steps/test_bot_utils_bdd.py:46

  Scenario: 1000 USD                     # features/cformat_test.feature:7
    Given Number is 1000, currency is USD # features/steps/test_bot_utils_bdd.py:40
    Then Result must be _1 000,00 $_      # features/steps/test_bot_utils_bdd.py:46

Feature: Right Currency Choosing # features/get_exchange_rate_test.feature:1

  Scenario: XRP TO RUB                               # features/get_exchange_rate_test.feature:3
    Given Crypto currency is XRP, to currency is RUB # features/steps/test_bot_utils_bdd.py:29
    Then Result must be between 60 and 90            # features/steps/test_bot_utils_bdd.py:35

  Scenario: BTC TO USD                               # features/get_exchange_rate_test.feature:7
    Given Crypto currency is BTC, to currency is USD # features/steps/test_bot_utils_bdd.py:29
    Then Result must be between 30000 and 60000      # features/steps/test_bot_utils_bdd.py:35

2 features passed, 0 failed, 0 skipped
4 scenarios passed, 0 failed, 0 skipped
8 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m7.039s
```