|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

Факультет «Информатика и системы управления»

ДИСЦИПЛИНА:

«БКИТ»

**Домашнее задание**

Студент Распашнов А.А. ИУ5Ц-54Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(И.О. Фамилия) (Группа) (Подпись, дата)

Преподаватель Гапанюк Ю.Е.  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(И.О. Фамилия) (Подпись, дата)

Оглавление

**Элементы оглавления не найдены.**

**Домашнее задание**

### Задание:

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).

# **ДЗ – код**



import unittest  
from bot import TOKEN  
from handlers.food import available\_burger\_names, available\_drink\_names, available\_fries\_names  
  
  
class TestBotWork(unittest.TestCase):  
 *# список еды, напитков и прочего может подгружаться из баз данных, поэтому важно знать, что список  
 # возможных продуктов не пустой* def test\_bot\_food\_not\_empty(self):  
 self.assertNotEqual(0, len(available\_burger\_names))  
  
 def test\_bot\_drinks\_not\_empty(self):  
 self.assertNotEqual(0, len(available\_drink\_names))  
  
 def test\_bot\_fries\_not\_empty(self):  
 self.assertNotEqual(0, len(available\_fries\_names))  
  
 *# важно знать, что токен бот корректен, можно проверять его длину* def test\_bot\_token\_exist(self):  
 self.assertEqual(len(TOKEN), 46)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()



import asyncio  
import logging  
import types  
  
from aiogram import Bot  
from aiogram.contrib.fsm\_storage.memory import MemoryStorage  
from aiogram.types import BotCommand  
  
from handlers.common import \*  
from handlers.food import \*  
  
  
class OrderFood(StatesGroup):  
 waiting\_for\_burger\_name = State()  
 waiting\_for\_fries\_name = State()  
 waiting\_for\_drink\_name = State()  
  
  
async def food\_start(message: types.Message):  
 keyboard = types.ReplyKeyboardMarkup(resize\_keyboard=True)  
 for name in available\_burger\_names:  
 keyboard.add(name)  
 await message.answer("Выберите, c чем вы хотите бургер:", reply\_markup=keyboard)  
 await OrderFood.waiting\_for\_burger\_name.set()  
  
  
async def food\_chosen(message: types.Message, state: FSMContext):  
 if message.text.lower() not in available\_burger\_names:  
 await message.answer("Пожалуйста, выберите мясо в бургере, используя клавиатуру ниже.")  
 return  
 await state.update\_data(chosen\_burger=message.text.lower())  
  
 keyboard = types.ReplyKeyboardMarkup(resize\_keyboard=True)  
 for name in available\_fries\_names:  
 keyboard.add(name)  
 await OrderFood.next()  
 await message.answer("Теперь выберите вид картошки:", reply\_markup=keyboard)  
  
  
async def food\_fries\_chosen(message: types.Message, state: FSMContext):  
 if message.text.lower() not in available\_fries\_names:  
 await message.answer("Пожалуйста, выберите вид картошки, используя клавиатуру ниже.")  
 return  
 await state.update\_data(chosen\_fries=message.text.lower())  
  
 keyboard = types.ReplyKeyboardMarkup(resize\_keyboard=True)  
 for name in available\_drink\_names:  
 keyboard.add(name)  
 await OrderFood.next()  
 await message.answer('Теперь выберите напиток:', reply\_markup=keyboard)  
  
  
async def food\_drink\_chosen(message: types.Message, state: FSMContext):  
 if message.text.lower() not in available\_drink\_names:  
 await message.answer("Пожалуйста, выберите напиток, используя клавиатуру ниже.")  
 return  
 user\_data = await state.get\_data()  
 await message.answer(f"Вы заказали бургер {user\_data['chosen\_burger'][2:]}, \n"  
 f"картофель-{user\_data['chosen\_fries']} и напиток - {message.text.lower()}. \n"  
 f"Спасибо за ваш заказ! ", reply\_markup=types.ReplyKeyboardRemove())  
 await state.finish()  
  
def register\_handlers\_food(dp: Dispatcher):  
 dp.register\_message\_handler(food\_start, commands="food", state="\*")  
 dp.register\_message\_handler(food\_chosen, state=OrderFood.waiting\_for\_burger\_name)  
 dp.register\_message\_handler(food\_fries\_chosen, state=OrderFood.waiting\_for\_fries\_name)  
 dp.register\_message\_handler(food\_drink\_chosen, state=OrderFood.waiting\_for\_drink\_name)  
  
  
logger = logging.getLogger(\_\_name\_\_)  
  
  
*# Регистрация команд, отображаемых в интерфейсе Telegram*async def set\_commands(bot: Bot):  
 commands = [  
 BotCommand(command="/food", description="Заказать блюда"),  
 BotCommand(command="/cancel", description="Отменить текущее действие")  
 ]  
 await bot.set\_my\_commands(commands)  
  
  
TOKEN = '5093952542:AAEfnITGgt2MSjOUOlQapYGUYWqiJdY2QJY'  
  
async def main():  
 logging.basicConfig(  
 level=logging.INFO,  
 format="%(asctime)s - %(levelname)s - %(name)s - %(message)s",  
 )  
 logger.error("Starting bot")  
  
  
 *# Объявление и инициализация объектов бота и диспетчера* bot = Bot(token=TOKEN)  
 dp = Dispatcher(bot, storage=MemoryStorage())  
  
 *# Регистрация хэндлеров* register\_handlers\_common(dp)  
  
 register\_handlers\_food(dp)  
  
 *# Установка команд бота* await set\_commands(bot)  
  
 *# Запуск поллинга* await dp.skip\_updates()  
 await dp.start\_polling()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 asyncio.run(main())

**Результат ДЗ:**

C:\Users\arsec\OneDrive\Desktop\DZ\DZ\venv\Scripts\python.exe "S:\Pyh\PyCharm 2021.3\plugins\python\helpers\pycharm\\_jb\_unittest\_runner.py" --path C:/Users/arsec/OneDrive/Desktop/DZ/DZ/test\_bot.py

Testing started at 17:07 ...

Launching unittests with arguments python -m unittest C:/Users/arsec/OneDrive/Desktop/DZ/DZ/test\_bot.py in C:\Users\arsec\OneDrive\Desktop\DZ\DZ

Ran 4 tests in 0.010s

OK

Process finished with exit code 0