**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» Кафедра «Системы обработки информации и управления»**

Лабораторная работа №6

**«**Разработка бота на основе конечного автомата для Telegram с использованием языка Python**»**

по предмету

«Базовые компоненты интернет-технологий»

Выполнил:

студент группы № ИУ5-31Б

Радченко Дмитрий

Проверил:

Преподаватель кафедры ИУ-5

Гапанюк Юрий

2023 г.

**Постановка задачи.**

Разработайте бота для Telegram. Бот должен реализовывать конечный автомат из трех состояний.

**Код программы:**

**Main.py**

# Copyright © 2022 mightyK1ngRichard <dimapermyakov55@gmail.com>  
  
*from* aiogram.utils *import* executor  
*from* create\_bot *import* dp  
*from* aiogram *import* types  
  
  
*async def* on\_startup(\_):  
 print('The bot was included.')  
  
  
*async def* handler\_start(message: types.Message):  
 *await* message.answer('Привет!')  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 *from* handlers *import* operations  
  
 operations.register\_handlers(dp)  
 executor.start\_polling(dp, skip\_updates=*True*, on\_startup=on\_startup)

**Creat\_bot.py**

# Copyright © 2022 mightyK1ngRichard <dimapermyakov55@gmail.com>  
*from* aiogram *import* Bot, Dispatcher  
*from* aiogram.contrib.fsm\_storage.memory *import* MemoryStorage  
*from* dotenv *import* load\_dotenv  
*import* os  
  
load\_dotenv()  
TOKEN = os.getenv('TOKEN')  
  
bot = Bot(TOKEN)  
storage = MemoryStorage()  
dp = Dispatcher(bot, storage=MemoryStorage())

**Utils.db\_management.py**

# Copyright © 2022 mightyK1ngRichard <dimapermyakov55@gmail.com>  
  
*import* sqlite3  
*from* random *import* choice  
  
  
*def* update\_name(id\_user: int, new\_name: str, filename: str = 'data/users\_old.db') -> *None*:  
 db = sqlite3.connect(filename)  
 cursor = db.cursor()  
 cursor.execute(f"UPDATE users SET name = '{new\_name}' WHERE id\_user = {id\_user}")  
 db.commit()  
 db.close()  
  
  
*def* get\_user\_data(id\_user: int, filename: str = 'data/users\_old.db') -> tuple | *None*:  
 db = sqlite3.connect(filename)  
 cursor = db.cursor()  
 cursor.execute(f"SELECT \* FROM users WHERE id\_user = {id\_user}")  
 res = cursor.fetchone()  
 db.commit()  
 db.close()  
 *return* res  
  
  
*def* get\_random\_user\_form(id\_user: int, filename: str = 'data/users\_old.db') -> tuple:  
 db = sqlite3.connect(filename)  
 cursor = db.cursor()  
 cursor.execute(f"SELECT id\_user FROM users WHERE id\_user <> {id\_user}")  
 id\_user: int = choice([el[0] *for* el *in* cursor.fetchall()])  
 res = get\_user\_data(id\_user, filename)  
 db.commit()  
 db.close()  
 *return* res  
  
  
*def* set\_user\_date(id\_user: int, data: dict, filename: str = 'data/users\_old.db'):  
 db = sqlite3.connect(filename)  
 cursor = db.cursor()  
 cursor.execute(  
 "INSERT INTO users VALUES ({}, '{}', {}, '{}', '{}', '{}', '{}', '{}')".format(id\_user, data['name'],  
 data['age'],  
 data['place'],  
 data['university'],  
 data['department'],  
 data['description'],  
 data['photo'])  
 )  
 db.commit()  
 db.close()  
  
  
*def* update\_user\_data(id\_user: int, data: dict, filename: str = 'data/users\_old.db'):  
 db = sqlite3.connect(filename)  
 cursor = db.cursor()  
 cursor.execute(  
 "UPDATE users SET name = '{}', age={}, place = '{}', university = '{}', department = '{}', description = '{}',"  
 "photo = '{}' WHERE id\_user = {}".format(data['name'], data['age'], data['place'], data['university'],  
 data['department'],  
 data['description'],  
 data['photo'],  
 id\_user))  
  
 db.commit()  
 db.close()  
  
  
*def* update\_user\_description(id\_user: int, description: str, filename: str = 'data/users\_old.db'):  
 db = sqlite3.connect(filename)  
 cursor = db.cursor()  
 cursor.execute(f"UPDATE users SET description = '{description}' WHERE id\_user = {id\_user}")  
 db.commit()  
 db.close()  
  
  
*def* update\_user\_photo(id\_user: int, photo: str, filename: str = 'data/users\_old.db'):  
 db = sqlite3.connect(filename)  
 cursor = db.cursor()  
 cursor.execute(f"UPDATE users SET photo = '{photo}' WHERE id\_user = {id\_user}")  
 db.commit()  
 db.close()  
  
  
*def* user\_presents(id\_user: int, filename: str = 'data/users\_old.db') -> bool:  
 *"""Checking, is user in data or is not.* ***:param*** *id\_user: id of the checking user.* ***:param*** *filename: way and name for file.* ***:return****: user status True/False.  
 """* db = sqlite3.connect(filename)  
 cursor = db.cursor()  
 cursor.execute(f"SELECT id\_user FROM users WHERE id\_user = {id\_user}")  
 lst = cursor.fetchall()  
 db.commit()  
 db.close()  
 *return False if* len(lst) == 0 *else True*

**Keyabords.menu\_bts.py**

# Copyright © 2022 mightyK1ngRichard <dimapermyakov55@gmail.com>  
*from* aiogram.types *import* ReplyKeyboardMarkup, KeyboardButton  
  
start\_markup = ReplyKeyboardMarkup(resize\_keyboard=*True*)  
buttons = (  
 'Заполнить анкету заново',  
 'Изменить фото',  
 'Изменить текст анкеты',  
 'Искать друзей 🤝'  
)  
  
start\_markup.row(\*(el *for* el *in* buttons[:2])).row(\*(el *for* el *in* buttons[2:4]))

**Keyabords.search\_lib.py**

# Copyright © 2022 mightyK1ngRichard <dimapermyakov55@gmail.com>  
*from* aiogram.types *import* InlineKeyboardButton, InlineKeyboardMarkup  
  
search\_markup = InlineKeyboardMarkup(resize\_keyboard=*True*)  
buttons = (  
 ('👍', 'like'),  
 ('📝', 'write'),  
 ('👎', 'dislike'),  
 ('💤', 'info')  
)  
search\_markup.row(\*(InlineKeyboardButton(text, callback\_data=data) *for* text, data *in* buttons))

**Handlers.operations**

# Copyright © 2022 mightyK1ngRichard <dimapermyakov55@gmail.com>  
*from* aiogram *import* Dispatcher, types  
*from* aiogram.dispatcher.filters.state *import* State, StatesGroup  
*from* aiogram.dispatcher *import* FSMContext  
*import* aiogram.utils.markdown *as* md  
*from* keyboards.menu\_bts *import* start\_markup  
*from* keyboards.search\_ib *import* search\_markup  
*from* utils.db\_management *import* \*  
*from* create\_bot *import* dp  
  
  
*class* FSMUsers(StatesGroup):  
 id\_user = State()  
 photo = State()  
 name = State()  
 age = State()  
 place = State()  
 university = State()  
 department = State()  
 description = State()  
  
  
*class* FSMChangeUserData(StatesGroup):  
 photo = State()  
  
  
*async def* handler\_start(message: types.Message):  
 # Если пользователь уже зарегистрирован.  
 user\_data = get\_user\_data(message.chat.id)  
 *if* user\_data *is not None*:  
 text = "Класс! Вы уже зарегистрированы!"  
 *await* message.answer(text=text)  
 *await* message.answer\_photo(  
 photo=user\_data[7],  
 caption=md.text(  
 md.text('Так выглядит Ваша анкета:\n'),  
 md.text(f'{user\_data[1]}, {user\_data[2]}, {user\_data[3]}'),  
 md.text(f'{user\_data[4]}, {user\_data[5]}'),  
 md.text('О себе:'),  
 md.text(user\_data[6]),  
 sep='\n',  
 )  
 )  
 *await* message.answer('Выберите дальнейший сценарий:', reply\_markup=start\_markup)  
  
 *else*:  
 *await* message.answer("О, новичок!\nДавайте Вас зарегистрируем!")  
 *await* FSMUsers.photo.set()  
 *await* message.answer("Отправьте ваше фото:")  
  
  
*async def* catch\_photo(message: types.Message, state: FSMContext):  
 *async with* state.proxy() *as* data:  
 data['photo'] = message.photo[0].file\_id  
  
 # print(len(data))  
 *await* FSMUsers.next()  
 *await* message.reply('Теперь введите Ваше имя:')  
  
  
*async def* catch\_name(message: types.Message, state: FSMContext):  
 *async with* state.proxy() *as* data:  
 data['name'] = message.text  
 *await* FSMUsers.next()  
 *await* message.reply('Введите ваш возраст:')  
  
  
*async def* catch\_age(message: types.Message, state: FSMContext):  
 *async with* state.proxy() *as* data:  
 data['age'] = message.text  
 *await* FSMUsers.next()  
 *await* message.reply('Введите ваш город:')  
  
  
*async def* catch\_place(message: types.Message, state: FSMContext):  
 *async with* state.proxy() *as* data:  
 data['place'] = message.text  
 *await* FSMUsers.next()  
 *await* message.reply('Введите название вашего университета:')  
  
  
*async def* catch\_university(message: types.Message, state: FSMContext):  
 *async with* state.proxy() *as* data:  
 data['university'] = message.text  
 *await* FSMUsers.next()  
 *await* message.reply('Введите название вашего факультета:')  
  
  
*async def* catch\_department(message: types.Message, state: FSMContext):  
 *async with* state.proxy() *as* data:  
 data['department'] = message.text  
 *await* FSMUsers.next()  
 *await* message.reply('Расскажите что-то о себе:')  
  
  
*async def* catch\_description(message: types.Message, state: FSMContext):  
 *async with* state.proxy() *as* data:  
 data['description'] = message.text  
 *async with* state.proxy() *as* data:  
 *try*:  
 *await* message.answer\_photo(  
 photo=data['photo'],  
 caption=md.text(  
 md.text('Так выглядит Ваша анкета:\n'),  
 md.text(f"{data['name']}, {data['age']}, {data['place']}"),  
 md.text(f"{data['university']}, {data['department']}"),  
 md.text('О себе:'),  
 md.text(data['description']),  
 sep='\n',  
 )  
 )  
 # Добавляет в SQL таблицу.  
 *if* user\_presents(message.chat.id):  
 update\_user\_data(message.chat.id, dict(data))  
 *else*:  
 set\_user\_date(message.chat.id, dict(data))  
 *except* KeyError:  
 temp\_id = message.chat.id  
 update\_user\_description(temp\_id, message.text)  
 new\_data = get\_user\_data(temp\_id)  
 *await* message.answer\_photo(  
 photo=new\_data[7],  
 caption=md.text(  
 md.text('Так выглядит Ваша анкета:\n'),  
 md.text(f'{new\_data[1]}, {new\_data[2]}, {new\_data[3]}'),  
 md.text(f'{new\_data[4]}, {new\_data[5]}'),  
 md.text('О себе:'),  
 md.text(new\_data[6]),  
 sep='\n',  
 )  
 )  
 *await* state.finish()  
 *await* message.answer('Выберите новый сценарий:', reply\_markup=start\_markup)  
  
  
*async def* all\_msg\_handler(message: types.Message):  
 button\_text = message.text  
  
 *if* button\_text == 'Заполнить анкету заново':  
 *await* FSMUsers.photo.set()  
 *await* message.answer("Отправьте ваше фото:", reply\_markup=types.ReplyKeyboardRemove())  
  
 *elif* button\_text == 'Изменить фото':  
 *await* FSMChangeUserData.photo.set()  
 *await* message.reply("Отправьте ваше новое фото:", reply\_markup=types.ReplyKeyboardRemove())  
  
 *elif* button\_text == 'Изменить текст анкеты':  
 *await* FSMUsers.description.set()  
 *await* message.reply("Расскажите что-то о себе:", reply\_markup=types.ReplyKeyboardRemove())  
  
 *elif* button\_text == 'Искать друзей 🤝':  
 user\_data = get\_random\_user\_form(message.chat.id)  
 *await* message.answer('Поиск друзей!', reply\_markup=types.ReplyKeyboardRemove())  
 *await* message.delete()  
 *await* message.answer\_photo(  
 photo=user\_data[7],  
 caption=md.text(  
 md.text(f"{user\_data[1]}, {user\_data[2]}, {user\_data[3]}"),  
 md.text(f"{user\_data[4]}, {user\_data[5]}"),  
 md.text('О себе:'),  
 md.text(user\_data[6]),  
 sep='\n',  
 ),  
 reply\_markup=search\_markup  
 )  
 # await message.reply(reply\_text)  
  
 *else*:  
 reply\_text = "Keep calm... Everything is fine, you just a silly"  
 *await* message.reply(reply\_text)  
 *await* message.delete()  
  
  
*async def* update\_user\_photo\_fsm(message: types.Message, state: FSMContext):  
 *async with* state.proxy() *as* data:  
 data['photo'] = message.photo[0].file\_id  
 temp\_id = message.chat.id  
 update\_user\_photo(temp\_id, data['photo'])  
 new\_data = get\_user\_data(temp\_id)  
 *await* message.answer\_photo(  
 photo=new\_data[7],  
 caption=md.text(  
 md.text('Так выглядит Ваша анкета:\n'),  
 md.text(f'{new\_data[1]}, {new\_data[2]}, {new\_data[3]}'),  
 md.text(f'{new\_data[4]}, {new\_data[5]}'),  
 md.text('О себе:'),  
 md.text(new\_data[6]),  
 sep='\n',  
 )  
 )  
 *await* state.finish()  
 *await* message.answer('Выберите новый сценарий:', reply\_markup=start\_markup)  
  
  
@dp.callback\_query\_handler(text=['like', 'write', 'dislike', 'info'])  
*async def* inline\_kb\_answer\_callback\_handler(query: types.CallbackQuery):  
 answer\_data = query.data  
  
 *if* answer\_data == 'like':  
 text = 'В разработке!'  
 *await* query.message.answer(text)  
  
 *elif* answer\_data == 'write':  
 text = 'В разработке!'  
 *await* query.message.answer(text)  
  
 *elif* answer\_data == 'dislike':  
 user\_data = get\_random\_user\_form(query.message.chat.id)  
 *await* query.message.answer\_photo(  
 photo=user\_data[7],  
 caption=md.text(  
 md.text(f"{user\_data[1]}, {user\_data[2]}, {user\_data[3]}"),  
 md.text(f"{user\_data[4]}, {user\_data[5]}"),  
 md.text('О себе:'),  
 md.text(user\_data[6]),  
 sep='\n',  
 ),  
 reply\_markup=search\_markup  
 )  
  
 *elif* answer\_data == 'info':  
 user\_data = get\_user\_data(query.message.chat.id)  
 *await* query.message.answer\_photo(  
 photo=user\_data[7],  
 caption=md.text(  
 md.text('Так выглядит Ваша анкета:\n'),  
 md.text(f'{user\_data[1]}, {user\_data[2]}, {user\_data[3]}'),  
 md.text(f'{user\_data[4]}, {user\_data[5]}'),  
 md.text('О себе:'),  
 md.text(user\_data[6]),  
 sep='\n',  
 )  
 )  
 *await* query.message.answer('Выберите дальнейший сценарий:', reply\_markup=start\_markup)  
  
 *else*:  
 text = f'Unexpected callback data {answer\_data!r}!'  
 *await* query.message.answer(text)  
  
  
*def* register\_handlers(dp\_main: Dispatcher):  
 dp\_main.register\_message\_handler(handler\_start, commands=['start'])  
 dp\_main.register\_message\_handler(catch\_photo, content\_types=['photo'], state=FSMUsers.photo)  
 dp\_main.register\_message\_handler(update\_user\_photo\_fsm, content\_types=['photo'], state=FSMChangeUserData.photo)  
 dp\_main.register\_message\_handler(catch\_name, state=FSMUsers.name)  
 dp\_main.register\_message\_handler(catch\_age, *lambda* message: message.text.isdigit(), state=FSMUsers.age)  
 dp\_main.register\_message\_handler(catch\_place, state=FSMUsers.place)  
 dp\_main.register\_message\_handler(catch\_university, state=FSMUsers.university)  
 dp\_main.register\_message\_handler(catch\_department, state=FSMUsers.department)  
 dp\_main.register\_message\_handler(catch\_description, state=FSMUsers.description)  
 # dp\_main.register\_message\_handler(inline\_kb\_answer\_callback\_handler, text=['like', 'write', 'dislike', 'info'])  
 # Last point!! Important!  
 dp\_main.register\_message\_handler(all\_msg\_handler)

**Результат работы программы:**

База данных на основе SQLite3

