

Урок 7. Реализуем оформление заказа

ОГЛАВЛЕНИЕ

[НАСТРАИВАЕМ ВОЗМОЖНОСТЬ ОФОРМЛЕНИЯ ЗАКАЗА 3](#_Toc79589944)

[ДАЛЬНЕЙШИЕ ПЕРСПЕКТИВЫ ПРОЕКТА 9](#_Toc79589945)

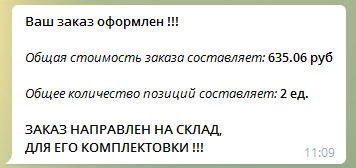
[ЗАКЛЮЧЕНИЕ 10](#_Toc79589946)

[КЛЮЧЕВЫЕ ВЫВОДЫ 10](#_Toc79589947)



# НАСТРАИВАЕМ ВОЗМОЖНОСТЬ ОФОРМЛЕНИЯ ЗАКАЗА

Мы хотим, чтобы при оформлении заказа выводился вот такой ответ:



Кнопка формирования заказа уже размещена в интерфейсе нашего проекта, (модуль **markup.py**):

itm\_btn\_8 = self.set\_btn(**'APPLAY'**, step, quantity)

К этой кнопке мы привяжем обработчик события нажатия:

**if** message.text == config.KEYBOARD[**'APPLAY'**]:

self.pressed\_btn\_apllay(message)  
*# иные нажатия и ввод данных пользователем***else**:  
 self.bot.send\_message(message.chat.id,message.text)

Добавим сам обработчик нажатия – функцию pressed\_btn\_apllay():

**Листинг 1. step\_7/handlers/handler\_all\_text.py**

**def** pressed\_btn\_apllay(self, message):  
 *"""  
 обрабатывает входящие текстовые сообщения  
 от нажатия на кнопку 'Оформить заказ'.  
 """  
 # отправляем ответ пользователю* self.bot.send\_message(message.chat.id,  
 MESSAGES[**'applay'**].format(  
 utility.get\_total\_coas(self.BD),  
  
 utility.get\_total\_quantity(self.BD)),  
 parse\_mode=**"HTML"**,  
 reply\_markup=self.keybords.category\_menu())  
 *# отчищаем данные с заказа* self.BD.delete\_all\_order()

Нам нужен простой ответ пользователю о том, что заказ сформирован. При этом, чтобы ответ содержал информацию об общей стоимости заказа и общем количестве единиц товаров в нем.

В приведенном коде мы видим новые для нас функции:

get\_total\_coas()

get\_total\_quantity()

Эти функции мы поместим в модуль **utility.py**.

**Листинг 2 step\_7/settings/utility.py**

**def** get\_total\_coas(BD):  
 *"""  
 Возвращает общую стоимость товара  
 """  
 # получаем список все product\_id заказа* all\_product\_id = BD.select\_all\_product\_id()  
 *# получаем список стоимость по всем позициям*

*# заказа в виде обычного списка* all\_price = [BD.select\_single\_product\_price(itm)

**for** itm **in** all\_product\_id]  
 *# получаем список количества по всем позициям заказа*

*# в виде обычного списка* all\_quantity = [BD.select\_order\_quantity(itm)

**for** itm **in** all\_product\_id]  
 *# Возвращает общую стоимость товара* **return** total\_coast(all\_quantity,all\_price)

Привычным образом получаем список идентификаторов всех товаров заказа:

all\_product\_id = BD.select\_all\_product\_id()

Теперь для каждого товара мы получаем цену и вносим ее в массив цен:

all\_price = [BD.select\_single\_product\_price(itm) **for** itm **in** all\_product\_id]

Аналогичным образом получаем список с количеством каждого товара в заказе:

all\_quantity = [BD.select\_order\_quantity(itm) **for** itm **in** all\_product\_id]

Теперь нам нужно посчитать общую стоимость заказа. Это обеспечит функция total\_coast(). Создадим ее в том же модуле **utility.py**.

**Листинг 3 step\_7/settings/utility.py**

*# Считает общую сумму заказа и возвращает результат***def** total\_coast(list\_quantity, list\_price):  
  
 order\_total\_cost = 0  
  
 **for** ind, itm **in** enumerate(list\_price):  
 order\_total\_cost += list\_quantity[ind]\*list\_price[ind]  
  
 **return** order\_total\_cost

Логика ее работы элементарна. Функция возвращает общую стоимость заказа.

Теперь по функции get\_total\_quantity():

**Листинг 4 step\_6/settings/utility.py**

**def** get\_total\_quantity(BD):  
 *"""  
 Возвращает общее количество заказанной единицы товара  
 """  
 # получаем список все product\_id заказа* all\_product\_id = BD.select\_all\_product\_id()  
 *# получаем список количества по всем позициям*

*# заказа в виде обычного списка* all\_quantity = [BD.select\_order\_quantity(itm)

**for** itm **in** all\_product\_id]  
 *# Возвращает количество товарных позиций* **return** total\_quantity(all\_quantity)

Первые два выражения мы уже разобрали. Список с количествами товаров передаем в функцию total\_quantity().

**Листинг 5 step\_6/settings/utility.py**

*# Считает общее количество заказанной единицы товара и возвращает результат***def** total\_quantity(list\_quantity):  
  
 order\_total\_quantity = 0  
  
 **for** itm **in** list\_quantity:  
 order\_total\_quantity += itm  
  
 **return** order\_total\_quantity

Функция также очень простая и возвращает количество товарных позиций в заказе.

Ну и конечно после оформления заказа необходимо удалить его из базы данных. Поэтому в модуле **handler\_all\_text.py** мы видим еще команду:

self.BD.delete\_all\_order()

Добавим эту функцию в модуль с менеджером базы данных:

**Листинг 6 step\_6/data\_base/dbalchemy.py**

**def** delete\_all\_order(self):  
 *"""  
 Удаляет данные всего заказа  
 """* all\_id\_orders = self.select\_all\_order\_id()  
  
 **for** itm **in** all\_id\_orders:  
 self.\_session.query(Order).filter\_by(id=itm).delete()  
 self.\_session.commit()  
 self.close()

Т.к. заказ уходит на оформление, нужно удалить данные по нему из таблицы с заказами.

С помощью функции select\_all\_order\_id()мы получаем все товары из заказа:

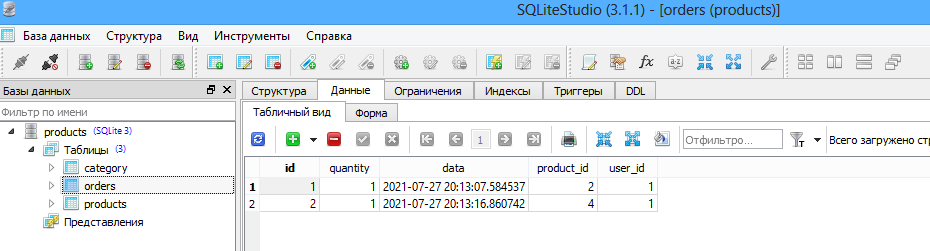
**Листинг 7 step\_6/data\_base/dbalchemy.py**

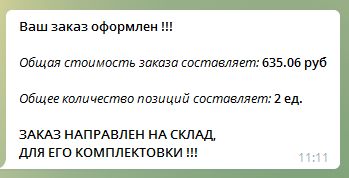
**def** select\_all\_order\_id(self):  
 *"""  
 Возвращает все id заказа  
 """* result = self.\_session.query(Order.id).all()  
 self.close()  
 **return** utility.\_convert(result)

А затем в цикле обходим элементы полученного массива и выполняем удаление:

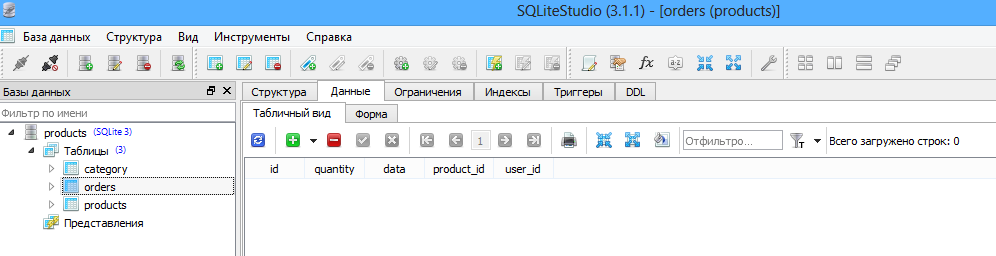
**for** itm **in** all\_id\_orders:  
 self.\_session.query(Order).filter\_by(id=itm).delete()  
 self.\_session.commit()

Проверим, что у нас получилось. Откроем файл с базой данных. Данные о заказе пока существуют.



Проверим, что у нас получилось. Попробуем оформить заказ.

Все работает. Заказ оформлен и в базе данных его больше нет.



# ДАЛЬНЕЙШИЕ ПЕРСПЕКТИВЫ ПРОЕКТА

Если вы дошли до текущего шага работы над проектом, поздравляю вас. Вы проделали большую и сложную, но важную и плодотворную работу. Теперь вы знаете, что на основе Python и дополнительных библиотек вы можете создавать не только настольные и веб-приложения, но Telegram-боты. Вы сделали хороший профессиональный рывок. Самое время его закрепить. Для этого я предлагаю сделать передышку и добавить в проект:

1. Сейчас у нас существует только роль покупателя и соответствующий интерфейс. Сделайте возможность выбора роли: покупатель или менеджер. Для менеджера реализуйте интерфейс работы с таблицей заказов. У нас сейчас заказ уходит в никуда. Сделайте так, чтобы после отправки заказа у менеджера всплывала метка, что есть новый заказ, который нужно собрать и передать в транспортную компанию. Можно сделать еще одну таблицу и соответствующий класс-модель специально для менеджера. В этой таблице будут храниться заказы для сборки и отправки менеджером. После сборки и отправки он будет нажимать кнопку «Подтвердить» и заказ будет удаляться из таблицы. При этом будет создаваться отчет о заказе и записываться в соответствующую таблицу (эту таблицу тоже нужно создать).
2. Т.е. у вас получится две новые таблицы и два новых класса-модели.
3. Да, еще третья таблица и класс для хранения ролей пользователей: покупатель и менеджер.
4. Важно, что в процессе работы над улучшением проекта вы столкнетесь с сложностями и вопросами, но вы обязательно их преодолеете и получите отличный рост знаний и умений.
5. Можно также реализовать функционал для заполнения каталога товара админом через интерфейс бота.

# Вид сверху на рукопожатие над деловыми документамиЗАКЛЮЧЕНИЕ

Первоначальная версия нашего Telegram-бота завершена. Проект можно улучшать и дорабатывать бесконечно. Попытайтесь дополнить приложение функционалом, который предложил реализовать преподаватель. Вы создали MVP-версию чат-бота, которую нужно обязательно добавить в свое портфолио, поскольку «наполненность» портфолио играет большую роль при поиске работы. Компании при поиске специалистов обращают большое внимание на то, в каким проектах принимал участие соискатель. Поэтому портфолио нужно наполнять, пусть даже это буду учебные проекты, стартапы.

## КЛЮЧЕВЫЕ ВЫВОДЫ

* Наш бот имеет полноценный графический интерфейс благодаря возможностям библиотеки pyTelegramBotApi и платформе Telegram.
* Пользователь бота имеет возможность просмотра каталога товаров, добавления товара в заказ, редактирования параметров заказа, подтверждения оформления заказа.
* Мы изучили возможности ORM-библиотеки SQLAlchemy и реализовали бэкенд для взаимодействия с базой данных проекта.
* Мы научились связывать элементы управления интерфейса с обработчиками.

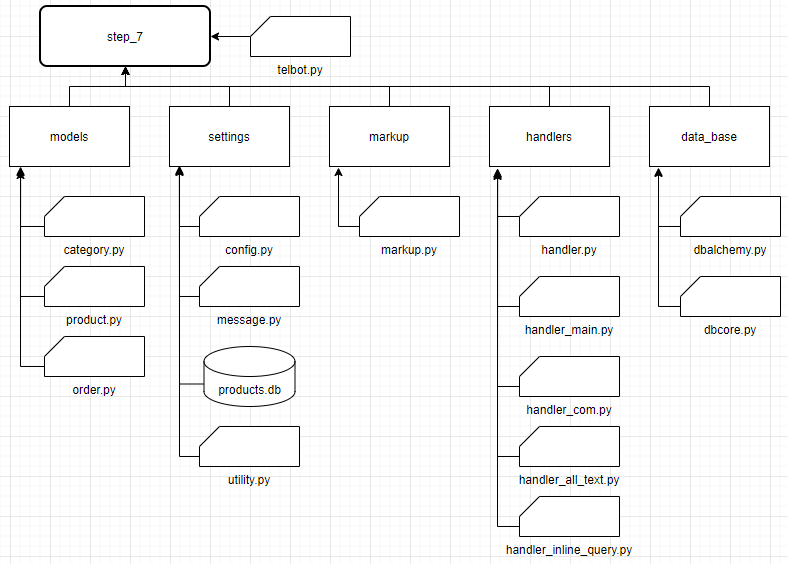
Текущая структура разрабатываемого Telegram-бота приведена на на рис. 1.

Рис. 1. Структура проекта