

Урок 6. Реализуем возможность редактирования заказа

ОГЛАВЛЕНИЕ

[НАСТРАИВАЕМ ИЗМЕНЕНИЕ КОЛИЧЕСТВА ТОВАРА В ЗАКАЗЕ 3](#_Toc79589395)

[НАСТРАИВАЕМ ОТОБРАЖЕНИЕ КОЛИЧЕСТВА ТОВАРА В ЗАКАЗЕ 7](#_Toc79589396)

[НАСТРАИВАЕМ УДАЛЕНИЕ ТОВАРА ИЗ ЗАКАЗА 9](#_Toc79589397)

[НАСТРАИВАЕМ ПЕРЕМЕЩЕНИЕ МЕЖДУ ТОВАРАМИ ЗАКАЗА 14](#_Toc79589398)

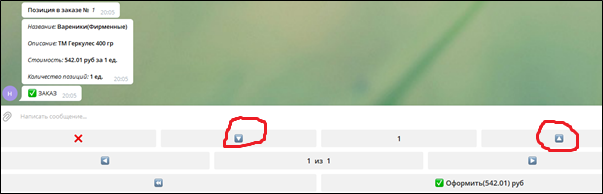
[ЗАКЛЮЧЕНИЕ 17](#_Toc79589399)

[КЛЮЧЕВЫЕ ВЫВОДЫ 17](#_Toc79589400)



# НАСТРАИВАЕМ ИЗМЕНЕНИЕ КОЛИЧЕСТВА ТОВАРА В ЗАКАЗЕ

Кнопки редактирования количества товара в заказе уже добавлены в интерфейс бота. Теперь нужно реализовать логику их работы.



Кнопки определены в модуле **markup.py**.

itm\_btn\_4 = self.set\_btn(**'UP'**, step, quantity)

itm\_btn\_2 = self.set\_btn(**'DOUWN'**,step,quantity)

Стоит напомнить, что step (шаг) – это, по сути, указание на номер товарной позиции в заказе.

Добавим в модуль **handler\_all\_text.py** выражение для обработки нажатия кнопки увеличения количества товара в заказе.

**if** message.text == config.KEYBOARD[**'UP'**]:  
 self.pressed\_btn\_up(message)

**Листинг 1. step\_6/handlers/handler\_all\_text.py**

**def** pressed\_btn\_up(self, message):  
 *"""  
 Обработка нажатия кнопки увеличения  
 количества определенного товара в заказе  
 """  
 # получаем список всех товаров в заказе* count = self.BD.select\_all\_product\_id()  
 *# получаем количество конкретной позиции в заказе* quantity\_order = self.BD.select\_order\_quantity(count[self.step])  
 *# получаем количество конкретной позиции в пролуктов* quantity\_product = self.BD.select\_single\_product\_quantity(  
 count[self.step])  
 *# если товар есть* **if** quantity\_product > 0:  
 quantity\_order += 1  
 quantity\_product -= 1  
 *# вносим изменения в БД orders* self.BD.update\_order\_value(count[self.step],  
 **'quantity'**, quantity\_order)  
 *# вносим изменения в БД product* self.BD.update\_product\_value(count[self.step],  
 **'quantity'**, quantity\_product)  
 *# отправляем ответ пользователю* self.send\_message\_order(count[self.step], quantity\_order, message)

Эта функция призвана обеспечить запись нового количества товара в заказе.

Первым делом получаем количество всех товарных позиций нашего заказа (select\_all\_product\_id()). Эту функцию мы уже реализовали. Получаем количество единиц текущей товарной позиции заказа (select\_order\_quantity()). Получаем количество данного товара на складе (select\_single\_product\_quantity()).

Если товар на складе еще есть, увеличиваем количество единиц товара в заказе. На складе количество товара уменьшится:

**if** quantity\_product > 0:  
 quantity\_order +=1  
 quantity\_product -=1

Далее вносим изменения в базу данных, а именно в таблицу с товаром на складе:

self.BD.update\_product\_value(count[self.step],**'quantity'**,quantity\_product)

И в таблицу с товаром в заказе:

self.BD.update\_order\_value(count[self.step],**'quantity'**,quantity\_order)

Т.е. количества товара на складе и в заказе меняются, и мы вносим эти изменения в базу данных.

Отправляем ответ пользователю:

self.send\_message\_order(count[self.step],quantity\_order,message)

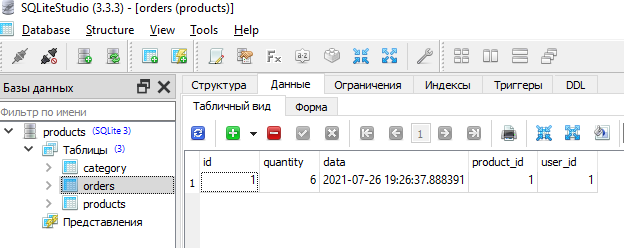
Теперь добавим в модуль **handler\_all\_text.py** выражение для обработки нажатия кнопки уменьшения количества товара в заказе:

**if** message.text == config.KEYBOARD[**'DOUWN'**]:  
 self.pressed\_btn\_douwn(message)

**Листинг 2. step\_6/handlers/handler\_all\_text.py**

**def** pressed\_btn\_douwn(self, message):  
 *"""  
 Обработка нажатия кнопки уменьшения  
 количества определенного товара в заказе  
 """  
 # получаем список всех товаров в заказе* count = self.BD.select\_all\_product\_id()  
 *# получаем количество конкретной позиции в заказе* quantity\_order = self.BD.select\_order\_quantity(count[self.step])  
 *# получаем количество конкретной позиции в пролуктов* quantity\_product = self.BD.select\_single\_product\_quantity(  
 count[self.step])  
 *# если товар в заказе есть* **if** quantity\_order > 0:  
 quantity\_order -= 1  
 quantity\_product += 1  
 *# вносим изменения в БД orders* self.BD.update\_order\_value(count[self.step],  
 **'quantity'**, quantity\_order)  
 *# вносим изменения в БД product* self.BD.update\_product\_value(count[self.step],  
 **'quantity'**, quantity\_product)  
 *# отправляем ответ пользователю* self.send\_message\_order(count[self.step], quantity\_order, message)

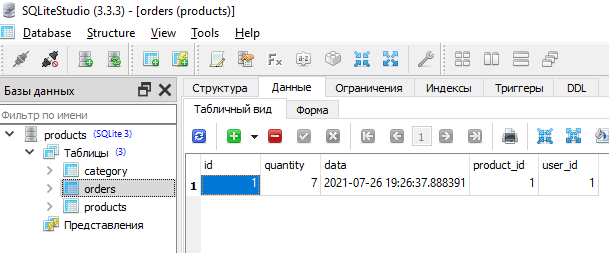
Логика работы этой функции похожа на предыдущий фрагмент, только в сторону уменьшения товара на складе и в заказе.

Проверим содержимое базы через приложение SQLiteStudio:

Проверим работу нового функционала. Увеличим количество товара:

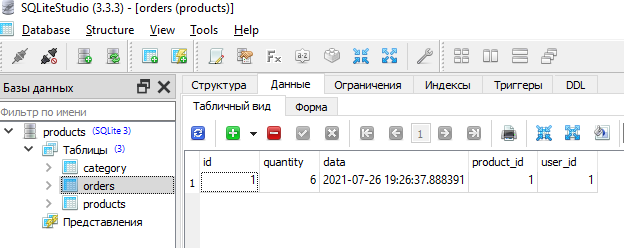


Работает.

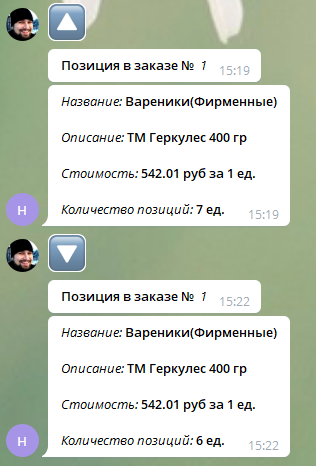


А теперь уменьшим количество товара.



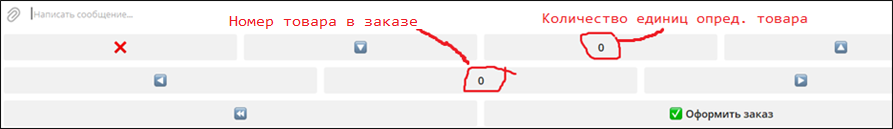
Работает.

В карточке товара также отслеживаются все изменения.



# НАСТРАИВАЕМ ОТОБРАЖЕНИЕ КОЛИЧЕСТВА ТОВАРА В ЗАКАЗЕ

Обратим внимание, что у нас уже есть один товар в заказе, но и в количестве единиц товара, и в номере товара в заказе значатся цифры 0, как будто товара вообще нет.



Нужно настроить обработчики для отображения корректных значений этих параметров.

Элементы управления количеством единиц товара определенной номенклатурной позиции и номером в заказе определены в модуле **markup.py**:

itm\_btn\_3 = self.set\_btn(**'AMOUNT\_PRODUCT'**, step, quantity)

itm\_btn\_6 = self.set\_btn(**'AMOUNT\_ORDERS'**, step, quantity)

В этом же файле у нас есть функция set\_btn(). В этой функции создавался и возвращался объект каждой кнопки. Но в настоящее время мы говорим, по сути, не о кнопках, а о надписях в нашем интерфейсе.

Поэтому в функцию set\_btn() мы добавим по факту просто формирование соответствующих надписей:

**Листинг 3. step\_6/markup/markup.py**

**if** name == **"AMOUNT\_ORDERS"**:  
 config.KEYBOARD[**"AMOUNT\_ORDERS"**] = **"{} {} {}"**.format(step + 1,  
 **' из '**, str(  
 self.BD.count\_rows\_order()))  
  
**if** name == **"AMOUNT\_PRODUCT"**:  
 config.KEYBOARD[**"AMOUNT\_PRODUCT"**] = **"{}"**.format(quantity)

Этот фрагмент:

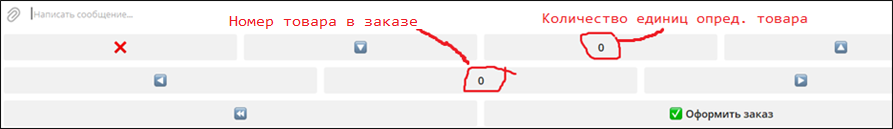
**if** name == **"AMOUNT\_PRODUCT"**:  
 config.KEYBOARD[**"AMOUNT\_PRODUCT"**] = **"{}"**.format(quantity)

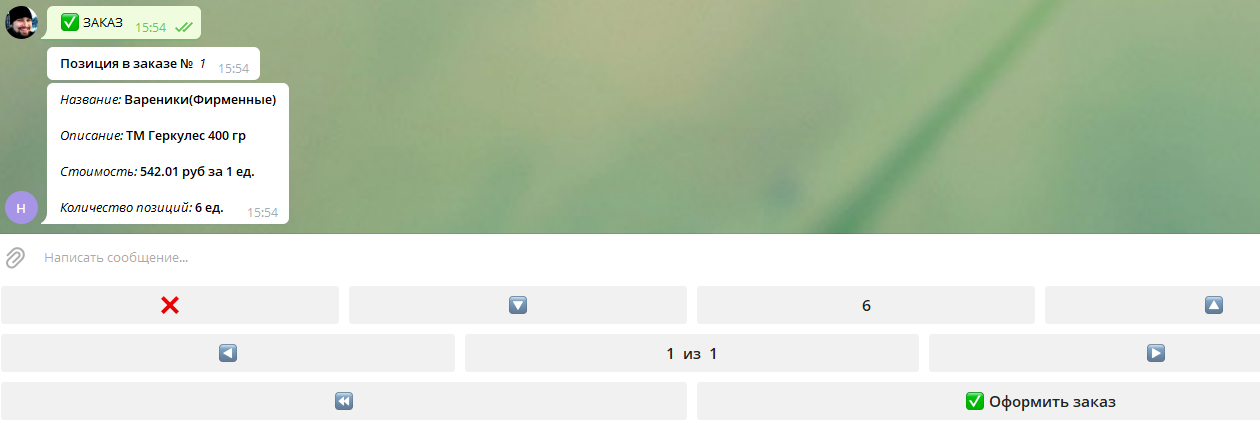
Обеспечивает вывод количества конкретного товара в заказе.

Этот фрагмент:

**if** name == **"AMOUNT\_ORDERS"**:  
 config.KEYBOARD[**"AMOUNT\_ORDERS"**] = **"{} {} {}"**.format(step + 1,  
 **' из '**, str(  
 self.BD.count\_rows\_order()))

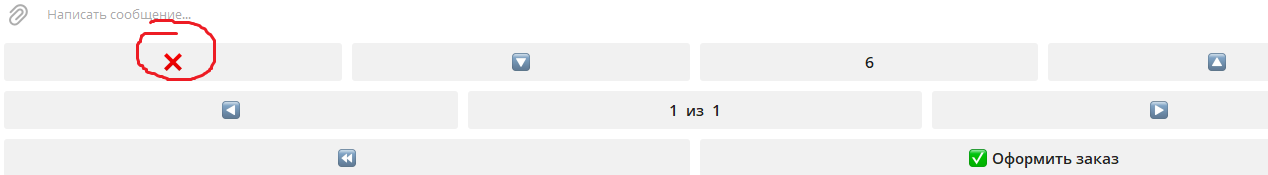
Обеспечивает вывод номера товара в заказе.



Все работает. Видно, что у нас в заказе один товар с количеством 6 единиц.

# НАСТРАИВАЕМ УДАЛЕНИЕ ТОВАРА ИЗ ЗАКАЗА

Соответствующая кнопка уже размещена в интерфейсе:



Данный элемент управления задан с помощью выражения:

itm\_btn\_1 = self.set\_btn(**'X'**, step, quantity)

А вот и привязка соответствующего обработчика в модуле **handler\_all\_text.py**:

**if** message.text == config.KEYBOARD[**'X'**]:  
 self.pressed\_btn\_x(message)

Обработчик нажатия кнопки:

**Листинг 4. step\_6/handlers/handler\_all\_text.py**

**def** pressed\_btn\_x(self, message):  
 *"""  
 Обрабатывает нажатие кнопки удаления товара в заказе  
 """  
 # получаем список всех product\_id заказа* count = self.BD.select\_all\_product\_id()  
 *# если список не пуст* **if** count.\_\_len\_\_() > 0:  
 *# получаем количество конкретной позиции в заказе* quantity\_order = self.BD.select\_order\_quantity(count[self.step])  
 *# получаем количество товара конкретной позиции на складе* quantity\_product = self.BD.select\_single\_product\_quantity(  
 count[self.step])  
 quantity\_product += quantity\_order  
 *# вносим изменения в БД orders* self.BD.delete\_order(count[self.step])  
 *# вносим изменения в БД product* self.BD.update\_product\_value(count[self.step], **'quantity'**,  
 quantity\_product)  
 *# уменьшаем шаг* self.step -= 1  
  
 **else**:  
 *# если товара нет в заказе отправляем сообщение* self.bot.send\_message(message.chat.id, MESSAGES[**'no\_orders'**],  
 parse\_mode=**"HTML"**,  
 reply\_markup=self.keybords.category\_menu())

Почему у этого обработчика такой сложный объемный код?

Если пользователь передумал оформлять заказ, нам нужно не только удалить сам заказ, но еще и «вернуть» товары из заказа на склад.

Для начала мы привычным образом получаем список идентификаторов всех товарных позиций заказа:

count = self.BD.select\_all\_product\_id()

Если список не пуст, значит товары в заказе есть.

Далее получаем количество выбранной товарной позиции заказа:

quantity\_order = self.BD.select\_order\_quantity(count[self.step])  
  
Получаем количество товара на складе:

quantity\_product =self.BD.select\_single\_product\_quantity(count[self.step])

Возвращаем количество товара из заказа обратно на склад:

quantity\_product += quantity\_order

Удаляем товар из заказа:

self.BD.delete\_order(count[self.step])

Обновляем количество товара на складе:

self.BD.update\_product\_value(count[self.step],**'quantity'**,quantity\_product)

Уменьшаем шаг:

self.step -= 1

Т.е., по сути, мы уменьшаем количество товарных позиций в заказе.

Отправляем сообщение пользователю:

self.send\_message\_order(count[self.step],quantity\_order,message)

Если товара в заказе нет, отправляем соответствующее сообщение:

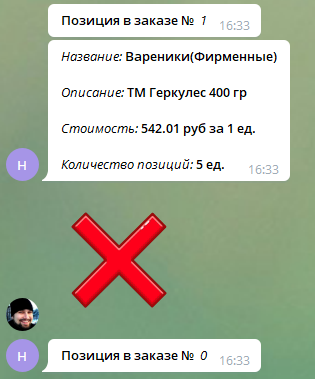
self.bot.send\_message(message.chat.id,MESSAGES[**'no\_orders'**],  
 parse\_mode=**"HTML"**,  
 reply\_markup=self.keybords.category\_menu())

Добавим в класс-менеджер базы данных сам обработчик удаления товара в заказе:

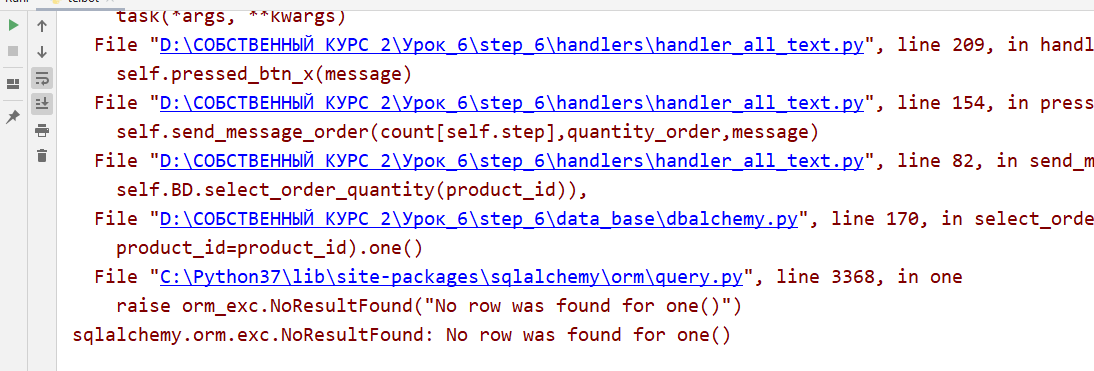
**Листинг 5. step\_6/data\_base/dbalchemy.py**

**def** delete\_order(self, product\_id):  
 *"""  
 Удаляет данные указанной строки заказа  
 """* self.\_session.query(Order).filter\_by(product\_id=product\_id).delete()  
 self.\_session.commit()  
 self.close()

Проверим работу проекта:



Не совсем понятный результат, да еще в потоке вывода в PyCharm ошибка:



В функции pressed\_btn\_x() у нас есть команда отправки сообщения, где мы сообщаем пользователю информацию о том, что сейчас находится в заказе:

self.send\_message\_order(count[self.step],quantity\_order,message)

А если у нас в заказе всего один товар и мы его удалили?

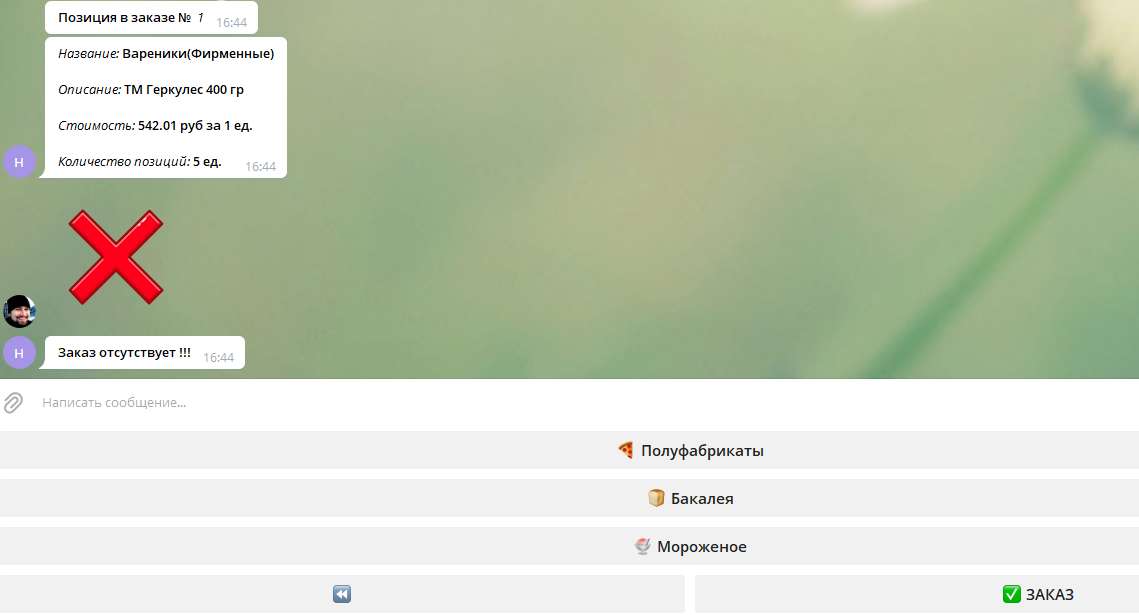
Тогда выражение выше даст нам ошибку, поэтому нужно вызывать это выражение только, если в заказе есть товар.

Поэтому функция pressed\_btn\_x() должна выглядеть так:

**def** pressed\_btn\_x(self, message):  
 *"""  
 Обрабатывает нажатие кнопки удаления товара в заказе  
 """  
 # получаем список всех product\_id заказа* count = self.BD.select\_all\_product\_id()  
 *# если список не пуст* **if** count.\_\_len\_\_() > 0:  
 *# получаем количество конкретной позиции в заказе* quantity\_order = self.BD.select\_order\_quantity(count[self.step])  
 *# получаем количество товара конкретной позиции на складе* quantity\_product = self.BD.select\_single\_product\_quantity(  
 count[self.step])  
 quantity\_product += quantity\_order  
 *# вносим изменения в БД orders* self.BD.delete\_order(count[self.step])  
 *# вносим изменения в БД product* self.BD.update\_product\_value(count[self.step], **'quantity'**,  
 quantity\_product)  
 *# уменьшаем шаг* self.step -= 1  
  
 count = self.BD.select\_all\_product\_id()  
 *# если список не пуст* **if** count.\_\_len\_\_() > 0:  
  
 quantity\_order = self.BD.select\_order\_quantity(count[self.step])  
 *# отправляем пользователю сообщение* self.send\_message\_order(count[self.step], quantity\_order, message)  
  
 **else**:  
 *# если товара нет в заказе отправляем сообщение* self.bot.send\_message(message.chat.id, MESSAGES[**'no\_orders'**],  
 parse\_mode=**"HTML"**,  
 reply\_markup=self.keybords.category\_menu())

Мы добавили блок:

count = self.BD.select\_all\_product\_id()  
 *# если список не пуст* **if** count.\_\_len\_\_() > 0:  
  
 quantity\_order = self.BD.select\_order\_quantity(count[self.step])  
 *# отправляем пользователю сообщение* self.send\_message\_order(count[self.step],quantity\_order,message)

Теперь при удалении все в порядке. Заказ удален.

# НАСТРАИВАЕМ ПЕРЕМЕЩЕНИЕ МЕЖДУ ТОВАРАМИ ЗАКАЗА

Кнопки перемещения между товарами заказы определены в модуле **markup.py**.

itm\_btn\_5 = self.set\_btn(**'BACK\_STEP'**,step,quantity)

itm\_btn\_7 = self.set\_btn(**'NEXT\_STEP'**,step,quantity)

Привяжем обработчики к событиям нажатия соответствующих кнопок (модуль **handler\_all\_text.py**):

**if** message.text == config.KEYBOARD[**'BACK\_STEP'**]:  
 self.pressed\_btn\_back\_step(message)  
  
**if** message.text == config.KEYBOARD[**'NEXT\_STEP'**]:  
 self.pressed\_btn\_next\_step(message)

А теперь непосредственно подготовим обработчики событий:

**Листинг 6. step\_6/handlers/handler\_all\_text.py**

**def** pressed\_btn\_back\_step(self, message):  
 *"""  
 Обрабатывает нажатие кнопки перемещения  
 на предыдущую позицию товара в заказе  
 """  
 # уменьшаем шаг пока шаг не будет равет "0"* **if** self.step > 0:  
 self.step -= 1  
 *# получаем список всех товаров в заказе* count = self.BD.select\_all\_product\_id()  
 quantity = self.BD.select\_order\_quantity(count[self.step])  
  
 *# отправляем ответ пользователю* self.send\_message\_order(count[self.step], quantity, message)

Переменная step – это шаг перемещения между товарными позициями в заказе.



Нажимая на эту кнопку, мы будем двигаться к более ранним товарам в заказе, пока значение переменной step не станет равным 0.

Привычным образом мы получаем список идентификаторов всех товаров в заказе:

count = self.BD.select\_all\_product\_id()  
Получаем количество текущего товара в заказе:

quantity = self.BD.select\_order\_quantity(count[self.step])

Отправляем ответ пользователю:

self.send\_message\_order(count[self.step],quantity,message)

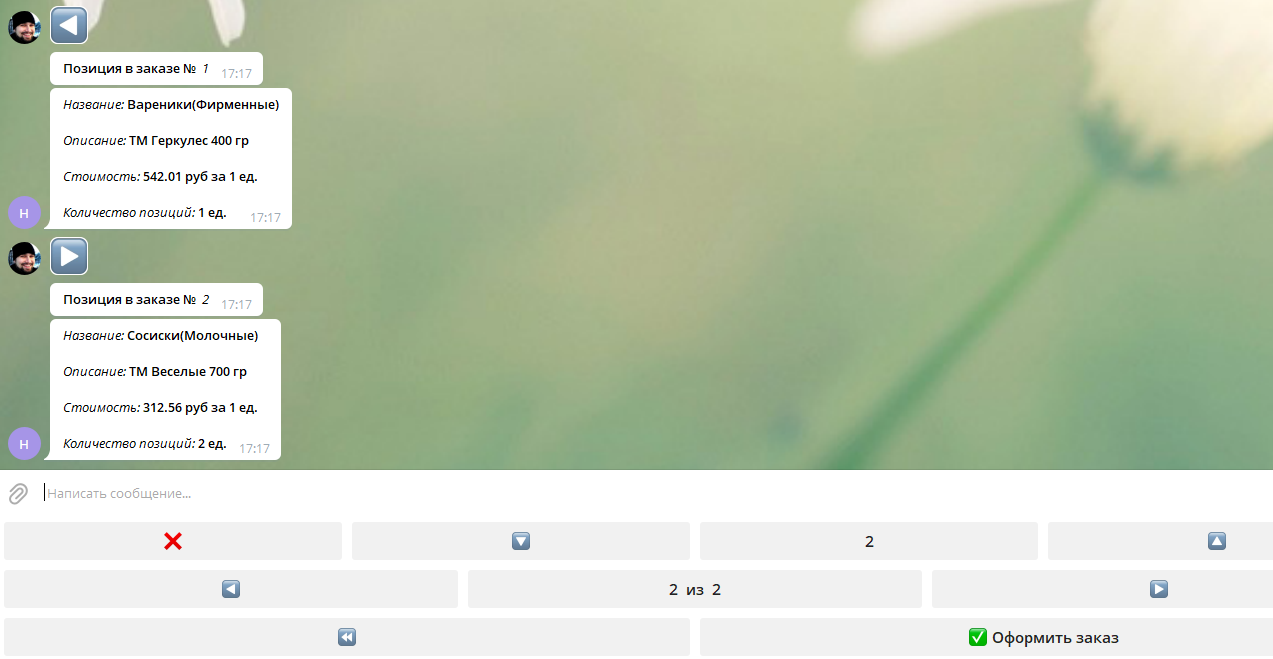
**Листинг 7. step\_6/handlers/handler\_all\_text.py**

**def** pressed\_btn\_next\_step(self, message):  
 *"""  
 Обрабатывает нажатие кнопки перемещения  
 на следующую позицию товара в заказе  
 """  
 # увеличиваем шаг пока шаг не будет равет количеству строк  
 # заказа с расчетом цены деления начиная с "0"* **if** self.step < self.BD.count\_rows\_order()-1:  
 self.step += 1  
 *# получаем список всех товаров в заказе* count = self.BD.select\_all\_product\_id()  
 *# получаем количество конкретного товара  
 # в соответствие с шагом выборки* quantity = self.BD.select\_order\_quantity(count[self.step])  
  
 *# отправляем ответ пользователю* self.send\_message\_order(count[self.step], quantity, message)



Логика работы похожа, только перемещаемся мы к более поздним товарам в заказе.

Если запустить проект, то все работает как надо. Перемещение между позициями заказа происходит, как и редактирование количества каждого товара в заказе.



# Вид сверху на рукопожатие над деловыми документамиЗАКЛЮЧЕНИЕ

В нашем проекте почти все готово. Мы можем уверенно добавлять товары в заказ, редактировать товарные позиции заказа: изменять количество, удалять товар из заказа, а также перемещаться между выбранными товарными позициями. Мы убедились, что возможности языка программирования Python, библиотеки Telegram Bot Api и библиотеки ORM позволяют создавать полноценные Телеграм-боты для реализации любых задач: учета, автоматизации, торговли и т.д.

## КЛЮЧЕВЫЕ ВЫВОДЫ

* В нашем проекте не хватает важного шага – оформления заказа и передачи его в работу.
* Также нужно поработать с отладкой, проверить работу проекта, привести код в точное соответствие положениям PEP-8.
* Также нужно подвести итоги, наметить дальнейшие перспективы разработки.

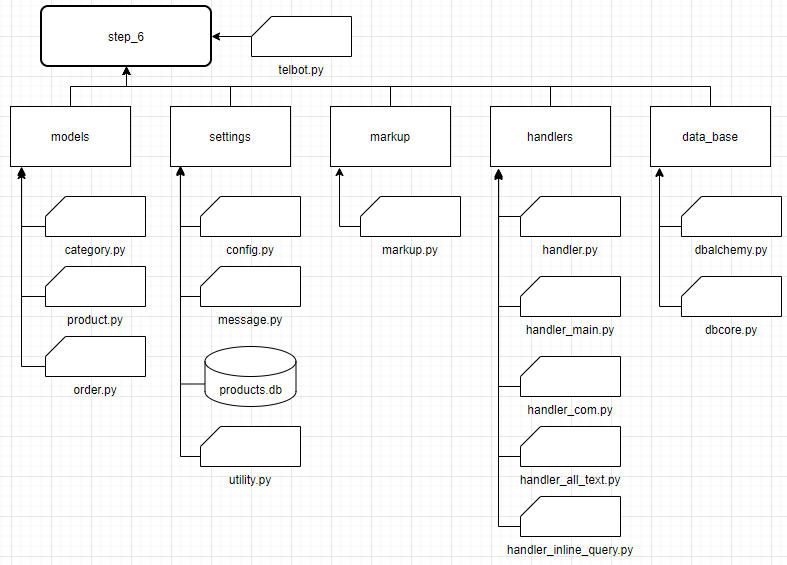
Текущая структура разрабатываемого Telegram-бота приведена на на рис. 1.

Рис. 1. Структура проекта