

Урок 5. Реализуем добавление товаров в заказ

ОГЛАВЛЕНИЕ

[НАСТРАИВАЕМ ВЫБОР ТОВАРА ДЛЯ ЗАКАЗА 3](#_Toc79589113)

[ПРОСМОТР СОДЕРЖИМОГО ЗАКАЗА 12](#_Toc79589114)

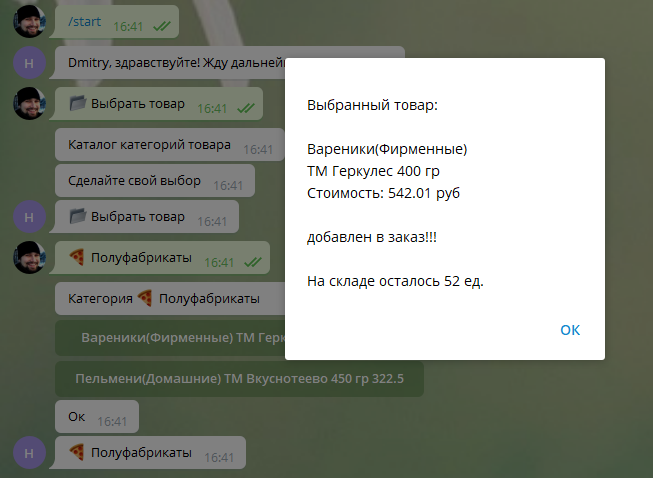
[ЗАКЛЮЧЕНИЕ 19](#_Toc79589115)

[КЛЮЧЕВЫЕ ВЫВОДЫ 19](#_Toc79589116)



# НАСТРАИВАЕМ ВЫБОР ТОВАРА ДЛЯ ЗАКАЗА

Итак, пользователь теперь может просматривать список товаров из каждой категории. Мы хотим сделать так, чтобы при нажатии на позицию (выбор товар из списка) этот товар добавлялся бы в текущий заказ пользователя и пользователь получал бы такое сообщение:



Для этого добавим в список наших модулей с обработчиками (в директорию **handlers**) модуль **handler\_inline\_query.py**.

**Листинг 1. step\_5/handlers/handler\_inline\_query.py**

*# импортируем класс родитель***from** handlers.handler **import** Handler  
*# импортируем сообщения пользователю***from** settings.message **import** MESSAGES  
  
  
**class** HandlerInlineQuery(Handler):  
 *"""  
 Класс обрабатывает входящие текстовые  
 сообщения от нажатия на инлайн-кнопоки  
 """* **def** \_\_init\_\_(self, bot):  
 super().\_\_init\_\_(bot)  
  
 **def** pressed\_btn\_product(self, call, code):  
 *"""  
 Обрабатывает входящие запросы на нажатие inline-кнопок товара  
 """  
 # создаем запись в БД по факту заказа* self.BD.\_add\_orders(1, code, 1)  
  
 self.bot.answer\_callback\_query(  
 call.id,  
 MESSAGES[**'product\_order'**].format(  
 self.BD.select\_single\_product\_name(code),  
 self.BD.select\_single\_product\_title(code),  
 self.BD.select\_single\_product\_price(code),  
 self.BD.select\_single\_product\_quantity(code)),  
 show\_alert=**True**)  
  
 **def** handle(self):  
 *# обработчик(декоратор) запросов от нажатия на кнопки товара.* @self.bot.callback\_query\_handler(func=**lambda** call: **True**)  
 **def** callback\_inline(call):  
 code = call.data  
 **if** code.isdigit():  
 code = int(code)  
  
 self.pressed\_btn\_product(call, code)

При нажатии на кнопку с товаром должна запускаться функция, обрабатывающая событие нажатия.

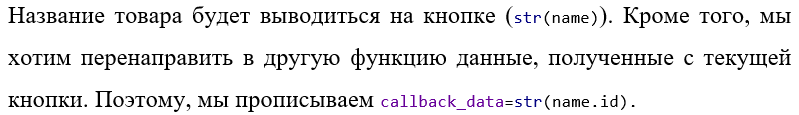
В нашем случае это функция:

**def** handle(self):  
 *# обработчик(декоратор) запросов от нажатия на кнопки товара.* @self.bot.callback\_query\_handler(func=**lambda** call: **True**)  
 **def** callback\_inline(call):  
 code = call.data  
 **if** code.isdigit():  
 code = int(code)  
  
 self.pressed\_btn\_product(call, code)

В разделе 4 курса мы познакомились с таким выражением:

callback\_data=str(name.id)

Дословно описание этого выражения звучало так:



Отправленный id мы принимаем в функции callback\_inline(call). Вот этот параметр:

code = call.data

Обработчик нажатия кнопки передает управление на функцию pressed\_btn\_product().

Эта функция первым делом создает заказ в базе данных:

self.BD.\_add\_orders(1, code, 1)

А вот большая функция, через которую мы работаем с заказом.

**Листинг 2. step\_5/data\_base/dbalchemy.py**

**def** \_add\_orders(self, quantity, product\_id, user\_id,):  
 *"""  
 Метод заполнения заказа  
 """  
 # получаем список всех product\_id* all\_id\_product = self.select\_all\_product\_id()  
 *# если данные есть в списке, обновляем таблицы заказа и продуктов* **if** product\_id **in** all\_id\_product:  
 quantity\_order = self.select\_order\_quantity(product\_id)  
 quantity\_order += 1  
 self.update\_order\_value(product\_id, **'quantity'**, quantity\_order)  
  
 quantity\_product = self.select\_single\_product\_quantity(product\_id)  
 quantity\_product -= 1  
 self.update\_product\_value(product\_id, **'quantity'**, quantity\_product)  
 **return** *# если данных нет, создаем новый объект заказа* **else**:  
 order = Order(quantity=quantity, product\_id=product\_id,  
 user\_id=user\_id, data=datetime.now())  
 quantity\_product = self.select\_single\_product\_quantity(product\_id)  
 quantity\_product -= 1  
 self.update\_product\_value(product\_id, **'quantity'**, quantity\_product)  
  
 self.\_session.add(order)  
 self.\_session.commit()  
 self.close()

Для начала получаем все идентификаторы товаров в заказе, нужно ведь проверить, возможно мы ранее уже добавили товар в заказ:

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

А вот и сама функция, обеспечивающая получение идентификаторов всех товаров из базы данных:

**Листинг 3. step\_5/data\_base/dbalchemy.py**

**def** select\_all\_product\_id(self):  
 *"""  
 Возвращает все id товара в заказе  
 """* result = self.\_session.query(Order.product\_id).all()  
 self.close()  
 *# конвертируем результат выборки в вид [1,3,5...]* **return** utility.\_convert(result)

Тут появляется несложный запрос:

result = self.\_session.query(Order.product\_id).all()

Он возвращает нам массив кортежей с идентификаторами товаров, такого вида: [(5,),(8,),...]. Формат не самый удобный, ведь нам, по сути, нужен просто набор идентификаторов, поэтому добавим в директорию **settings** нашего проекта модуль **utility.py**, где будем хранить полезные функции – утилиты.

Добавим в этот модуль первую функцию:

utility.\_convert(result)

Вот ее код:

**Листинг 4. step\_5/settings/utility.py**

**def** \_convert(list\_convert):  
  
 **return** [itm[0] **for** itm **in** list\_convert]

Благодаря этой функции набор идентификаторов товаров трансформируется из такого представления - [(5,),(8,),...] в такое [5, 8, …].

Вернемся обратно к функции \_add\_orders().

**if** product\_id **in** all\_id\_product:  
 quantity\_order = self.select\_order\_quantity(product\_id)  
 quantity\_order += 1  
 self.update\_order\_value(product\_id, **'quantity'**, quantity\_order)  
  
 quantity\_product = self.select\_single\_product\_quantity(product\_id)  
 quantity\_product -= 1  
 self.update\_product\_value(product\_id, **'quantity'**, quantity\_product)  
 **return**

Если мы заказывали товар ранее и выбираем товар вновь, то нужно получить текущее количество товара из заказа, изменить его на единицу и обновить базу данных. Вот этот код:

quantity\_order = self.select\_order\_quantity(product\_id)

quantity\_order += 1

self.update\_order\_value(product\_id, **'quantity'**, quantity\_order)

Функция select\_order\_quantity() находится в модуле **dbalchemy.py** и предназначена для получения существующего количества товара в заказе:

**Листинг 5. step\_5/data\_base/dbalchemy.py**

**def** select\_order\_quantity(self, product\_id):  
 *"""  
 Возвращает количество товара в заказе  
 """* result = self.\_session.query(Order.quantity).filter\_by(  
 product\_id=product\_id).one()  
 self.close()  
 **return** result.quantity

И функция обновления данных заказа:

**Листинг 6. step\_5/data\_base/dbalchemy.py**

**def** update\_order\_value(self, product\_id, name, value):  
 *"""  
 Обновляет данные указанной позиции заказа  
 в соответствии с номером товара - rownum  
 """* self.\_session.query(Order).filter\_by(  
 product\_id=product\_id).update({name: value})  
 self.\_session.commit()  
 self.close()

Вернемся обратно к функции \_add\_orders().

quantity\_product = self.select\_single\_product\_quantity(product\_id)

quantity\_product -= 1

self.update\_product\_value(product\_id, **'quantity'**, quantity\_product)

Т.к. мы заказали еще одну единицу некоторого товара, необходимо уменьшить количество товара на складе на эту самую единицу. Мы получаем товар по его id и уменьшаем количество. Далее выполняем обновление базы данных.

**Листинг 7. step\_5/data\_base/dbalchemy.py**

**def** select\_single\_product\_quantity(self, rownum):  
 *"""  
 Возвращает количество товара на складе  
 в соответствии с номером товара - rownum  
 Этот номер определяется при выборе товара в интерфейсе.  
 """* result = self.\_session.query(  
 Products.quantity).filter\_by(id=rownum).one()  
 self.close()  
 **return** result.quantity

**Листинг 8. step\_5/data\_base/dbalchemy.py**

**def** update\_product\_value(self, rownum, name, value):  
 *"""  
 Обновляет количество товара на складе  
 в соответствии с номером товара - rownum  
 """* self.\_session.query(Products).filter\_by(  
 id=rownum).update({name: value})  
 self.\_session.commit()  
 self.close()

Вернемся обратно к функции \_add\_orders().

Если товар этот ранее мы не добавляли в заказ:

**else**:  
 order = Order(quantity=quantity, product\_id=product\_id,  
 user\_id=user\_id, data=datetime.now())  
 quantity\_product = self.select\_single\_product\_quantity(product\_id)  
 quantity\_product -= 1  
 self.update\_product\_value(product\_id, **'quantity'**, quantity\_product)  
  
 self.\_session.add(order)  
 self.\_session.commit()  
 self.close()

То создаем новый заказ с этим товаром, уменьшаем количество товара на складе и выполняем обновление базы данных. Т.к. мы в любом случае меняем содержимое таблицы с заказами, то подтверждаем все изменения.

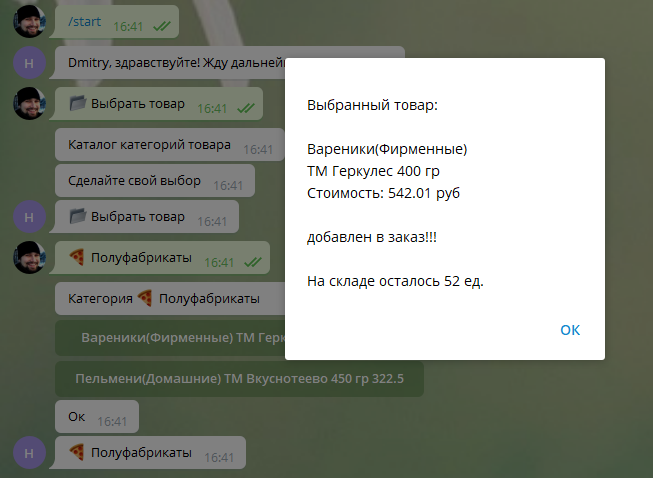
self.\_session.add(order)  
 self.\_session.commit()  
 self.close()

Вернемся к модулю **handler\_inline\_query.py.**

Нам нужно вывести сообщение, что товар добавлен в заказ.

self.bot.answer\_callback\_query(  
 call.id,  
 MESSAGES[**'product\_order'**].format(  
 self.BD.select\_single\_product\_name(code),  
 self.BD.select\_single\_product\_title(code),  
 self.BD.select\_single\_product\_price(code),  
 self.BD.select\_single\_product\_quantity(code)),  
 show\_alert=**True**)

Из модуля с сообщениями **messages.py** получаем текст сообщения и форматируем его, добавляя название товара, его торговую марку, цену и количество, оставшееся на складе. Т.е. по идее мы должны получить:



Всю работу выполняет функция **select\_single\_product\_name()**.

**Листинг 9. step\_5/data\_base/dbalchemy.py**

**def** select\_single\_product\_name(self, rownum):  
 *"""  
 Возвращает название товара  
 в соответствии с номером товара - rownum  
 """* result = self.\_session.query(Products.name).filter\_by(id=rownum).one()  
 self.close()  
 **return** result.name  
  
**def** select\_single\_product\_title(self, rownum):  
 *"""  
 Возвращает торговую марку товара  
 в соответствии с номером товара - rownum  
 """* result = self.\_session.query(Products.title).filter\_by(id=rownum).one()  
 self.close()  
 **return** result.title  
  
**def** select\_single\_product\_price(self, rownum):  
 *"""  
 Возвращает цену товара  
 в соответствии с номером товара - rownum  
 """* result = self.\_session.query(Products.price).filter\_by(id=rownum).one()  
 self.close()  
 **return** result.price

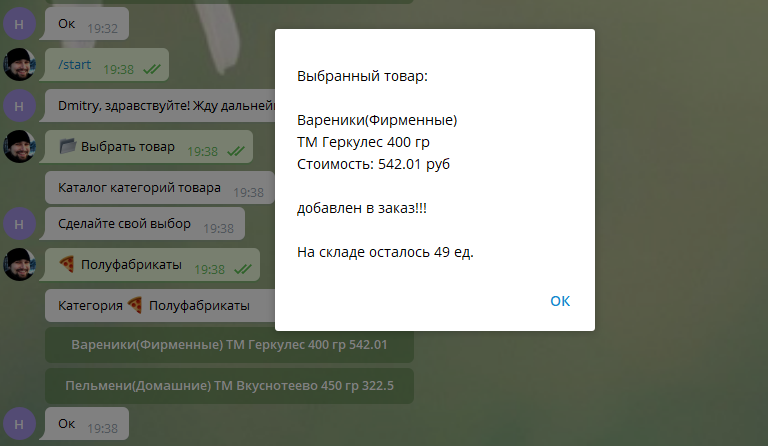
Можно запускать? Нет! Мы еще не подключили модуль **handler\_inline\_query.py** к нашему проекту.

Подключим его:

**Листинг 10. step\_5/handlers/handler\_main.py**

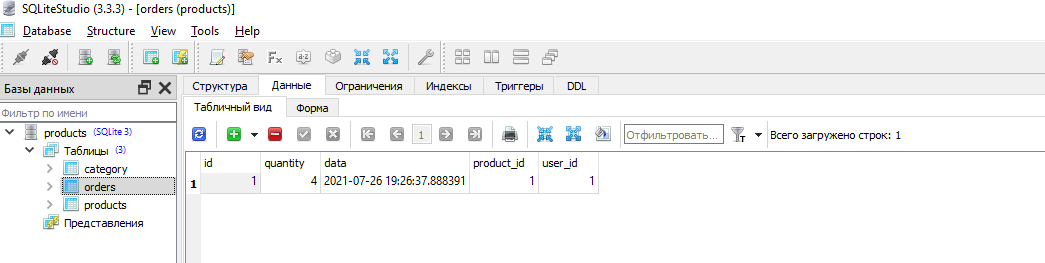
*# импортируем класс HandlerCommands обработка комманд***from** handlers.handler\_com **import** HandlerCommands  
*# импортируем класс HandlerAllText обработка нажатия на кнопки и иные сообщения***from** handlers.handler\_all\_text **import** HandlerAllText  
*# импортируем класс HandlerInlineQuery обработка нажатия на кнопки инлайн***from** handlers.handler\_inline\_query **import** HandlerInlineQuery  
  
  
**class** HandlerMain:  
 *"""  
 Класс компоновщик  
 """* **def** \_\_init\_\_(self, bot):  
 *# получаем нашего бота* self.bot = bot  
 *# здесь будет инициализация обработчиков* self.handler\_commands = HandlerCommands(self.bot)  
 self.handler\_all\_text = HandlerAllText(self.bot)  
 self.handler\_inline\_query = HandlerInlineQuery(self.bot)  
  
 **def** handle(self):  
 *# здесь будет запуск обработчиков* self.handler\_commands.handle()  
 self.handler\_all\_text.handle()  
 self.handler\_inline\_query.handle()

А теперь запустим проект и проверим его работу:

****

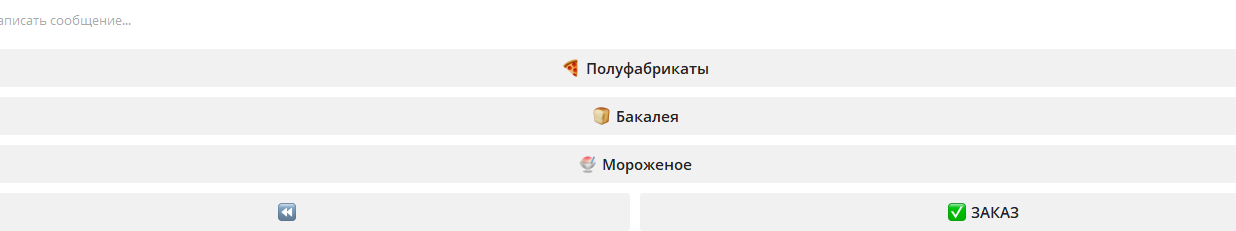
Если еще несколько раз нажать на кнопку с товаром, то количество товара в заказе увеличится. То же самое будет, если остановить работу бота и запустить его повторно.

Проверим содержимое таблицы с заказами через приложение SQLiteStudio.

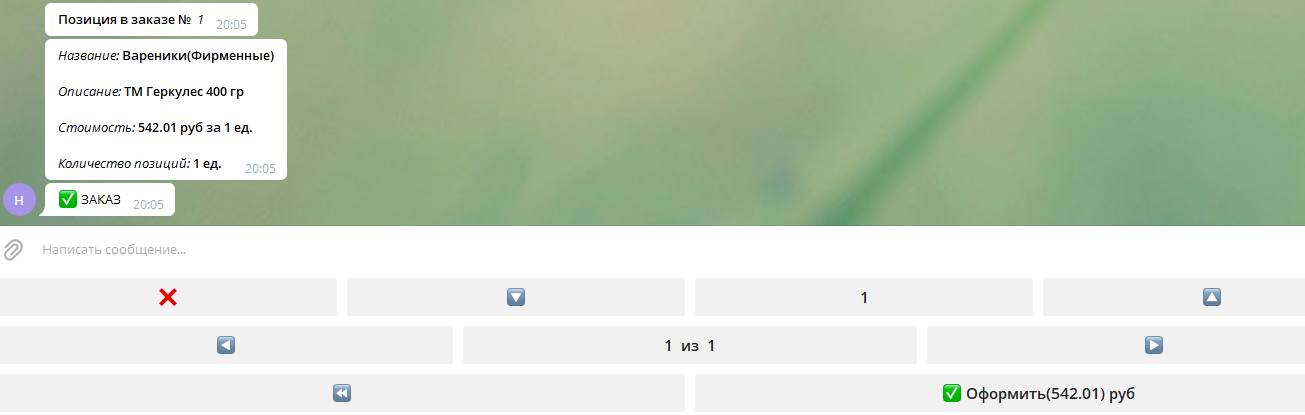


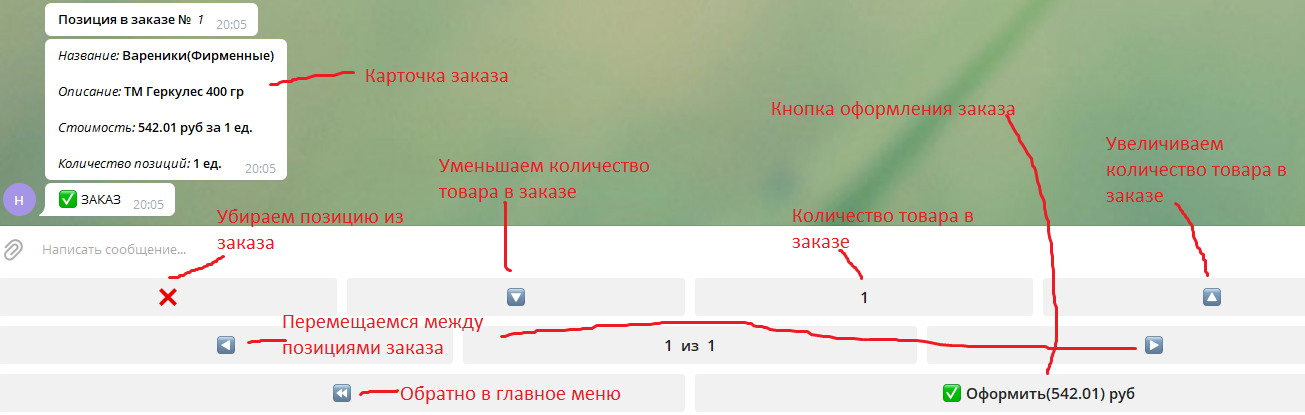
# ПРОСМОТР СОДЕРЖИМОГО ЗАКАЗА

Вспомним, что в интерфейсе нашего бота после перехода в меню выбора товара появляется такое окно:



Справа кнопка получения информации о заказе. Пока она не работает, но один заказ мы уже сделали, а значит хотим посмотреть информацию о нем.



Мы хотим увидеть вот такую картинку для управления заказом. На первый взгляд это сложный интерфейс. На самом деле, в нем нет ничего сложного.

Настроим формирование такого интерфейса.

В модуле **markup.py** у нас есть вот такая команда:

self.markup.row(self.set\_btn(**'<<'**), self.set\_btn(**'ORDER'**))

В этой команде мы видим кнопку с переходом в заказ.

Нужно настроит обработку нажатия на эту кнопку.

Добавим в модуль **handler\_all\_text.py** следующий фрагмент:

**Листинг 11. step\_5/handlers/handler\_all\_text.py**

**if** message.text == config.KEYBOARD[**'ORDER'**]:  
 *# если есть заказ* **if** self.BD.count\_rows\_order() > 0:  
 self.pressed\_btn\_order(message)  
 **else**:  
 self.bot.send\_message(message.chat.id,MESSAGES[**'no\_orders'**],  
 parse\_mode=**"HTML"**,  
 reply\_markup=self.keybords.category\_menu())

В этом фрагменте есть вызов следующего метода:

self.BD.count\_rows\_order()

Этим методом мы проверяем есть ли в базе данных заказ со списком товаров.

Добавим в модуль **dbalchemy.py** функцию:

**Листинг 12. step\_5/data\_base/dbalchemy.py**

**def** count\_rows\_order(self):  
 *"""  
 Возвращает количество позиций в заказе  
 """* result = self.\_session.query(Order).count()  
 self.close()  
 **return** result

Если заказ с товарами существует, мы выполняем вызов обработчика:

self.pressed\_btn\_order(message)

Добавляем в модуль **handler\_all\_text.py** код этого обработчика:

**Листинг 13. step\_5/handlers/handler\_all\_text.py**

**def** pressed\_btn\_order(self, message):  
 *"""  
 Обрабатывает входящие текстовые сообщения от нажатия на кнопку 'Заказ'.  
 """  
 # обнуляем данные шага* self.step = 0  
 *# получаем список всех товаров в заказе* count = self.BD.select\_all\_product\_id()  
 *# получаем количество по каждой позиции товара в заказе* quantity = self.BD.select\_order\_quantity(count[self.step])  
  
 *# отправляем ответ пользователю* self.send\_message\_order(count[self.step], quantity, message)

Эта функция должна сформировать карточку заказа.

Сперва мы получаем все товары в заказе:

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

Функцию **select\_all\_product\_id()** мы уже написали. Она находится в модуле **dbalchemy.py**.

Теперь мы получаем количество для каждого товара в заказе.

quantity = self.BD.select\_order\_quantity(count[self.step])  
  
**Листинг 14. step\_5/data\_base/dbalchemy.py**

**def** select\_order\_quantity(self, product\_id):  
 *"""  
 Возвращает количество товара из заказа  
 в соответствии с номером товара - rownum  
 """* result = self.\_session.query(Order.quantity).filter\_by(  
 product\_id=product\_id).one()  
 self.close()  
 **return** result.quantity

Вы скажете, но ведь в функции pressed\_btn\_order() нет цикла. Как тогда мы выведем карточку для каждой позиции заказа. А мы сначала выведем карточку первого товара:

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

Ведь self.step = 0

А потом будем менять значение шага. При переходе к новым позициям заказа. Меняется счетчик, значит, в массиве count мы будем каждый раз извлекать элемент по индексу на единицу больше и тем самым будем обращаться к следующему товару и т.д.

Вернемся к файлу **handler\_all\_text.py**

Отправляем ответ пользователю:

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

**Листинг 15. step\_5/handlers/handler\_all\_text.py**

**def** send\_message\_order(self, product\_id, quantity, message):  
 *"""  
 Отправляет ответ пользователю при выполнении различных действий  
 """* self.bot.send\_message(message.chat.id,MESSAGES[**'order\_number'**].format(  
 self.step+1), parse\_mode=**"HTML"**)  
 self.bot.send\_message(message.chat.id,  
 MESSAGES[**'order'**].  
 format(self.BD.select\_single\_product\_name(  
 product\_id),  
 self.BD.select\_single\_product\_title(  
 product\_id),  
 self.BD.select\_single\_product\_price(  
 product\_id),  
 self.BD.select\_order\_quantity(  
 product\_id)),  
 parse\_mode=**"HTML"**,  
 reply\_markup=self.keybords.orders\_menu(  
 self.step, quantity))

Команда:

self.bot.send\_message(message.chat.id,MESSAGES[**'order\_number'**].format(self.step+1),parse\_mode=**"HTML"**)

Отвечает за вывод:

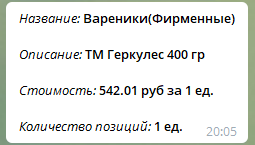


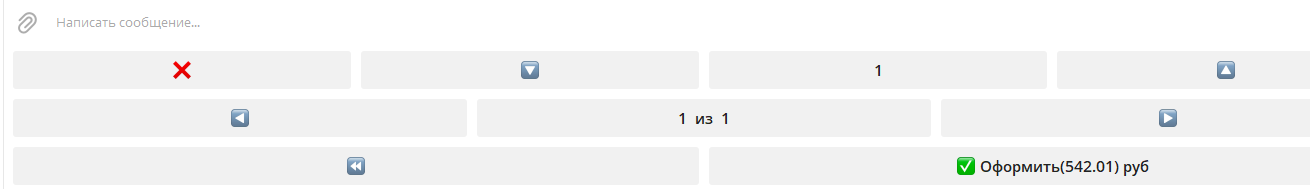
Вот здесь мы как раз меняем значение шага (self.step+1). Тем самым будем переходить к следующим позициям в заказе.

Блок:

self.bot.send\_message(message.chat.id,  
 MESSAGES[**'order'**].format(self.BD.select\_single\_product\_name(product\_id),  
 self.BD.select\_single\_product\_title(product\_id),  
 self.BD.select\_single\_product\_price(product\_id),  
 self.BD.select\_order\_quantity(product\_id)),  
 parse\_mode=**"HTML"**,

Отвечает за вывод:



А теперь, кто же нам покажет вот это сложное меню:

Вот это выражение:

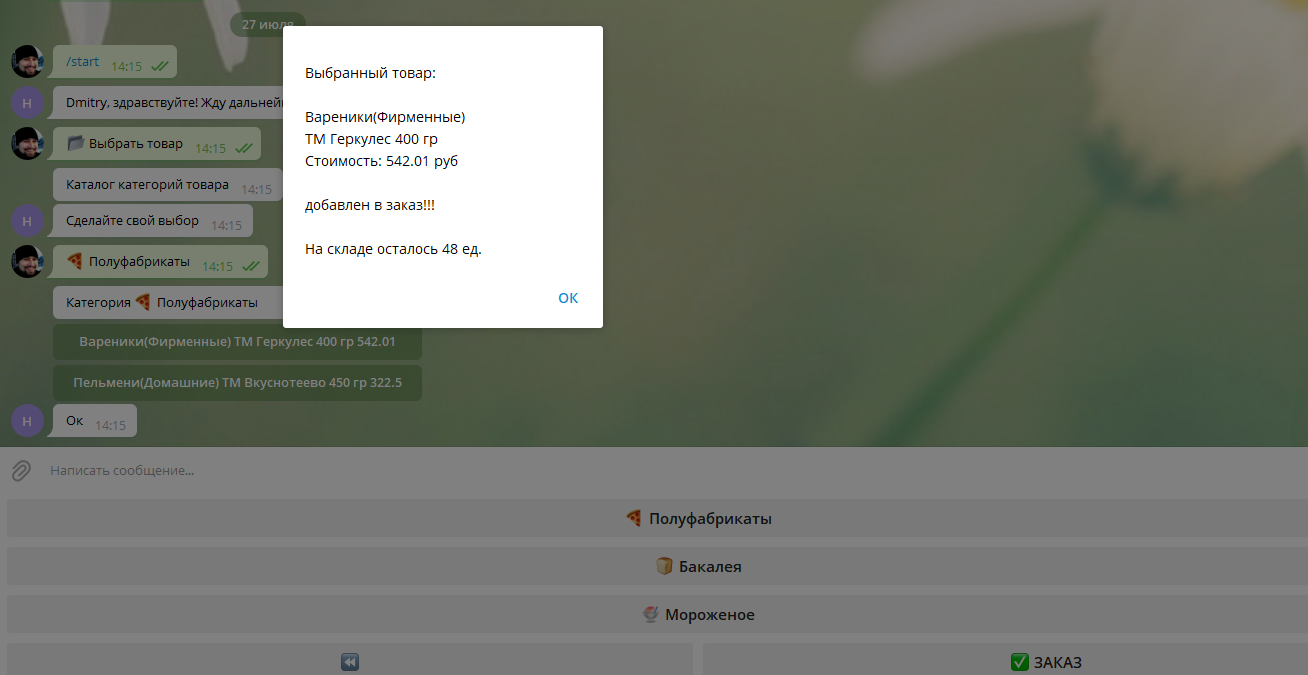
reply\_markup=self.keybords.orders\_menu(self.step,quantity))

**Листинг 16. step\_5/markup/markup.py**

**def** orders\_menu(self, step, quantity):  
 *"""  
 Создает разметку кнопок в заказе товара и возвращает разметку  
 """* self.markup = ReplyKeyboardMarkup(**True**, **True**)  
 itm\_btn\_1 = self.set\_btn(**'X'**, step, quantity)  
 itm\_btn\_2 = self.set\_btn(**'DOUWN'**, step, quantity)  
 itm\_btn\_3 = self.set\_btn(**'AMOUNT\_PRODUCT'**, step, quantity)  
 itm\_btn\_4 = self.set\_btn(**'UP'**, step, quantity)  
  
 itm\_btn\_5 = self.set\_btn(**'BACK\_STEP'**, step, quantity)  
 itm\_btn\_6 = self.set\_btn(**'AMOUNT\_ORDERS'**, step, quantity)  
 itm\_btn\_7 = self.set\_btn(**'NEXT\_STEP'**, step, quantity)  
 itm\_btn\_8 = self.set\_btn(**'APPLAY'**, step, quantity)  
 itm\_btn\_9 = self.set\_btn(**'<<'**, step, quantity)  
 *# рассположение кнопок в меню* self.markup.row(itm\_btn\_1, itm\_btn\_2, itm\_btn\_3, itm\_btn\_4)  
 self.markup.row(itm\_btn\_5, itm\_btn\_6, itm\_btn\_7)  
 self.markup.row(itm\_btn\_9, itm\_btn\_8)  
  
 **return** self.markup

Запустим нашего бота и проверим реализованный функционал.

Все работает.



# Вид сверху на рукопожатие над деловыми документамиЗАКЛЮЧЕНИЕ

Мы реализовали возможность выбора товара и добавления его в заказ. При добавлении отображается карточка товара в заказе с указанием наименования, торговой марки, стоимости, оставшегося количества товара на складе. Все изменения фиксируются в базе данных. Теперь пользователь может добавить в заказ любой товар.

## КЛЮЧЕВЫЕ ВЫВОДЫ

* Все, что мы реализовали в рамках текущего раздела, вам уже знакомо. Это и формирование элементов интерфейса для работы с заказом, взаимодействие с базой данных посредством ORM и соответствующих запросов.
* В интерфейс бота мы добавили возможность выбора пользователем товаров каждой номенклатурной группы и подключили соответствующий обработчик.
* Нам не хватает возможности редактирования содержимого заказа, а также запуска процедуры оформления заказа.
* Проект «обрастает» бэкендом и фронтендом. Он станоится сложнее, а значит нужно тратить больше времени на проработку материала. При этом наш бот становится все больше похож на привычный магазин, где можно просмотреть товар и добавить его в заказ.

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

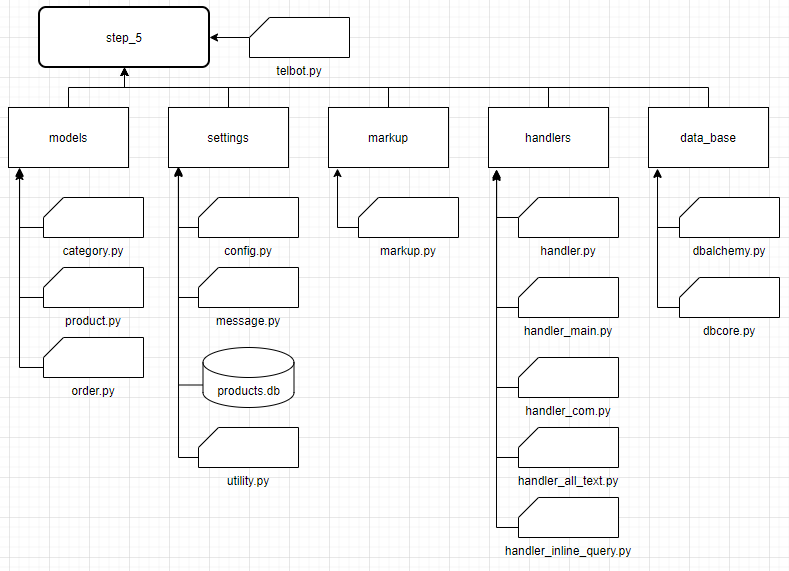


Рис. 1. Структура проекта