

Урок 3. Создаем первые элементы интерфейса и обработчики для них

ОГЛАВЛЕНИЕ

[СОЗДАЕМ НАЧАЛЬНЫЙ ИНТЕРФЕЙС И ФУНКЦИОНАЛ 3](#_Toc79587583)

[ДОБАВЛЯЕМ ФУНКЦИОНАЛ «О МАГАЗИНЕ» 8](#_Toc79587584)

[ДОБАВЛЯЕМ ФУНКЦИОНАЛ «НАСТРОЙКИ» 16](#_Toc79587585)

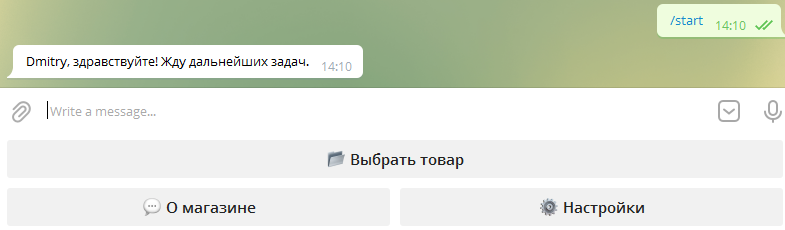
[ЗАКЛЮЧЕНИЕ 18](#_Toc79587586)

[КЛЮЧЕВЫЕ ВЫВОДЫ 18](#_Toc79587587)



# СОЗДАЕМ НАЧАЛЬНЫЙ ИНТЕРФЕЙС И ФУНКЦИОНАЛ

Представьте, мы хотим, чтобы при запуске бота отображался вот такой интерфейс:



Начнем его реализовывать.

**Листинг 1. step\_3/handlers/handler\_com.py**

|  |
| --- |
| *# импортируем класс родитель* **from** handlers.handler **import** Handler   **class** HandlerCommands(Handler):  *"""  Класс обрабатывает входящие команды /start и /help и т.п.  """* **def** \_\_init\_\_(self, bot):  super().\_\_init\_\_(bot)   **def** pressed\_btn\_start(self, message):  *"""  обрабатывает входящие /start команды  """* self.bot.send\_message(message.chat.id,  **f'{**message.from\_user.first\_name**},'  f' здравствуйте! Жду дальнейших задач.'**,  reply\_markup=self.keybords.start\_menu())   **def** handle(self):  *# обработчик(декоратор) сообщений,  # который обрабатывает входящие /start команды.* @self.bot.message\_handler(commands=[**'start'**])  **def** handle(message):  print(type(message))  **if** message.text == **'/start'**:  self.pressed\_btn\_start(message) |

При запуске бота пользователем срабатывает команда **/start**. Это событие мы перехватываем и запускаем обработчик команды – функцию **pressed\_btn\_start()**. Благодаря декоратору @self.bot.message\_handler(commands=[**'start'**]) осуществляется перехват ввода команды **/start** с передачей соответствующего этой команде объекта **message**. Это объект типа **<class 'telebot.types.Message'>**. В нем содержится много важных параметров, которые нам пригодятся.

Пример такого объекта **message** после запуска команды **/start**:

{'content\_type': 'text', 'message\_id': 11, 'from\_user': {'id': 474971924, 'is\_bot': False, 'first\_name': 'Dmitry', 'username': 'cdi999', 'last\_name': 'Chitalov', 'language\_code': 'ru'}, 'date': 1626782683, 'chat': {'type': 'private', 'last\_name': 'Chitalov', 'first\_name': 'Dmitry', 'username': 'cdi999', 'id': 474971924, 'title': None, 'all\_members\_are\_administrators': None, 'photo': None, 'description': None, 'invite\_link': None, 'pinned\_message': None, 'sticker\_set\_name': None, 'can\_set\_sticker\_set': None}, 'forward\_from\_chat': None, 'forward\_from': None, 'forward\_date': None, 'reply\_to\_message': None, 'edit\_date': None, 'media\_group\_id': None, 'author\_signature': None, 'text': '/start', 'entities': [<telebot.types.MessageEntity object at 0x000001E6E7A63188>], 'caption\_entities': None, 'audio': None, 'document': None, 'photo': None, 'sticker': None, 'video': None, 'video\_note': None, 'voice': None, 'caption': None, 'contact': None, 'location': None, 'venue': None, 'new\_chat\_member': None, 'new\_chat\_members': None, 'left\_chat\_member': None, 'new\_chat\_title': None, 'new\_chat\_photo': None, 'delete\_chat\_photo': None, 'group\_chat\_created': None, 'supergroup\_chat\_created': None, 'channel\_chat\_created': None, 'migrate\_to\_chat\_id': None, 'migrate\_from\_chat\_id': None, 'pinned\_message': None, 'invoice': None, 'successful\_payment': None, 'connected\_website': None, 'json': {'message\_id': 11, 'from': {'id': 474971924, 'is\_bot': False, 'first\_name': 'Dmitry', 'last\_name': 'Chitalov', 'username': 'cdi999', 'language\_code': 'ru'}, 'chat':

{'id': 474971924, 'first\_name': 'Dmitry', 'last\_name': 'Chitalov', 'username': 'cdi999', 'type': 'private'}, 'date': 1626782683, 'text': '/start', 'entities': [{'offset': 0, 'length': 6, 'type': 'bot\_command'}]}}

Нам нужен параметр 'text': '/start'.

**def** handle(self):  
 *# обработчик(декоратор) сообщений,  
 # который обрабатывает входящие /start команды.* @self.bot.message\_handler(commands=[**'start'**])  
 **def** handle(message):  
 **if** message.text == **'/start'**:  
 self.pressed\_btn\_start(message)

Сработала стартовая команда и запускается привязанный к ней обработчик **pressed\_btn\_start()**.

**def** pressed\_btn\_start(self, message):  
 *"""  
 обрабатывает входящие /start команды  
 """* self.bot.send\_message(message.chat.id,  
 **f'{**message.from\_user.first\_name**},'  
 f' здравствуйте! Жду дальнейших задач.'**,  
 reply\_markup=self.keybords.start\_menu())

Запускается приветствие, где выводится имя пользователя Telegram-бота, а также запуск очень важной функции **start\_menu()**.

Но есть одно, но! Как мы запустим функцию **start\_menu()** если в модуле **handler\_com.py** ее вообще нет?! Эта функция вызывается от объекта **keyboards**. Но его тоже нет в этом модуле! Вспомним, что в проекте мы реализовали наследование класса **HandlerCommands** от класса **Handler**. Это значит, если у класса **Handler** есть какие-то атрибуты, мы можем ими воспользоваться. Смотрим на класс **Handler** и видим:

|  |
| --- |
| **class** Handler(metaclass=abc.ABCMeta):   **def** \_\_init\_\_(self, bot):  *# получаем объект бота* self.bot = bot  *# инициализируем разметку кнопок* self.keybords = Keyboards()  *# инициализируем менеджер для работы с БД* self.BD = DBManager()   @abc.abstractmethod  **def** handle(self):  **pass** |

А вот и наш атрибут **self.keyboards**. Он ссылается на объект класса **Keyboards()**. Этот класс мы ранее определили в модуле **markup.py**. Расширим интерфейс класса **Keyboards()**.

**Листинг 2. step\_3/markup/markup.py**

|  |
| --- |
| *# импортируем специальные типы телеграм бота для создания элементов интерфейса* **from** telebot.types **import** KeyboardButton, ReplyKeyboardMarkup *# импортируем настройки и утилиты* **from** settings **import** config *# импортируем класс-менеджер для работы с библиотекой* **from** data\_base.dbalchemy **import** DBManager   **class** Keyboards:  *"""  Класс Keyboards предназначен для создания и разметки интерфейса бота  """  # инициализация разметки* **def** \_\_init\_\_(self):  self.markup = **None** *# инициализируем менеджер для работы с БД* self.BD = DBManager()   **def** set\_btn(self, name, step=0, quantity=0):  *"""  Создает и возвращает кнопку по входным параметрам  """* **return** KeyboardButton(config.KEYBOARD[name])   **def** start\_menu(self):  *"""  Создает разметку кнопок в основном меню и возвращает разметку  """* self.markup = ReplyKeyboardMarkup(**True**, **True**)  itm\_btn\_1 = self.set\_btn(**'CHOOSE\_GOODS'**)  itm\_btn\_2 = self.set\_btn(**'INFO'**)  itm\_btn\_3 = self.set\_btn(**'SETTINGS'**)  *# рассположение кнопок в меню* self.markup.row(itm\_btn\_1)  self.markup.row(itm\_btn\_2, itm\_btn\_3)  **return** self.markup |

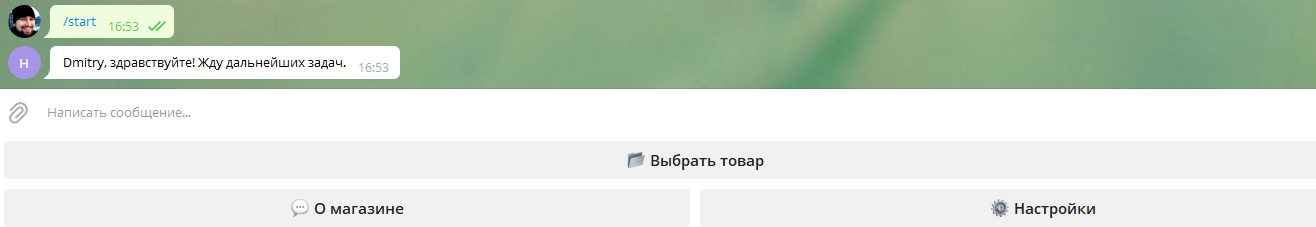
А вот и наша функция **start\_menu()**. Она обеспечит формирование необходимого интерфейса.

Класс **ReplyKeyboardMarkup** модуля **telebot.types** позволяет реализовать в интерфейсе настраиваемую клавиатуру с горизонтальными шаблонами разметки (row). В первый горизонтальный шаблон мы размещаем одну кнопку, во второй – две. Кнопки – это объекты класса **KeyboardButton**. Внешний вид кнопок мы уже настроили в модуле **config.py**. Под понятием «Клавиатура» здесь конечно понимается всего лишь набор элементов интерфейса бота.

Итак, мы создали наш обработчик **HandlerCommands**. Необходимо добавить его в центральное звено управления нашего проекта – модуль **handler\_main.py**. Мы добавим инициализацию обработчика и его запуск.

**Листинг 3. step\_3/handlers/handler\_main.py**

|  |
| --- |
| *# импортируем класс HandlerCommands обработка комманд* **from** handlers.handler\_com **import** HandlerCommands   **class** HandlerMain:  *"""  Класс-компоновщик  """* **def** \_\_init\_\_(self, bot):  *# получаем объект нашего бота* self.bot = bot  *# здесь будет инициализация обработчиков* self.handler\_commands = HandlerCommands(self.bot)   **def** handle(self):  *# здесь будет запуск обработчиков* self.handler\_commands.handle() |

Запустим проект и перейдем в бот.

Задача выполнена.

# ДОБАВЛЯЕМ ФУНКЦИОНАЛ «О МАГАЗИНЕ»

В интерфейсе нашего бота появились две кнопки «О магазине» и «Настройки». Пока они не работают, т.е. нажатие ничего не даст. Значит необходимо привязать к этим кнопкам обработчики события нажатия. Чтобы не хранить код всех обработчиков в одном файле, создадим еще один модуль и назовем его **handler\_all\_text.py**.

**Листинг 4. step\_3/handlers/handler\_all\_text.py**

|  |
| --- |
| *# импортируем ответ пользователю* **from** settings.message **import** MESSAGES **from** settings **import** config *# импортируем класс-родитель* **from** handlers.handler **import** Handler   **class** HandlerAllText(Handler):  *"""  Класс обрабатывает входящие текстовые сообщения от нажатия на кнопки  """* **def** \_\_init\_\_(self, bot):  super().\_\_init\_\_(bot)  *# шаг в заказе* self.step = 0   **def** pressed\_btn\_info(self, message):  *"""  обрабатывает входящие текстовые сообщения  от нажатия на кнопоку 'О магазине'.  """* self.bot.send\_message(message.chat.id, MESSAGES[**'trading\_store'**],  parse\_mode=**"HTML"**,  reply\_markup=self.keybords.info\_menu())   **def** pressed\_btn\_settings(self, message):  *"""  обрабатывает входящие текстовые сообщения  от нажатия на кнопоку 'Настройки'.  """* self.bot.send\_message(message.chat.id, MESSAGES[**'settings'**],  parse\_mode=**"HTML"**,  reply\_markup=self.keybords.settings\_menu())   **def** pressed\_btn\_back(self, message):  *"""  обрабатывает входящие текстовые сообщения*  *от нажатия на кнопку 'Назад'.  """* self.bot.send\_message(message.chat.id, **"Вы вернулись назад"**,  reply\_markup=self.keybords.start\_menu())   **def** handle(self):  *# обработчик(декоратор) сообщений,  # который обрабатывает входящие текстовые сообщения*  *# от нажатия кнопок.* @self.bot.message\_handler(func=**lambda** message: **True**)  **def** handle(message):  *# \*\*\*\*\*\*\*\*\*\* меню \*\*\*\*\*\*\*\*\*\* #* **if** message.text == config.KEYBOARD[**'INFO'**]:  self.pressed\_btn\_info(message)   **if** message.text == config.KEYBOARD[**'SETTINGS'**]:  self.pressed\_btn\_settings(message)   **if** message.text == config.KEYBOARD[**'<<'**]:  self.pressed\_btn\_back(message) |

Наш проект становится сложнее, поэтому будем всем разбирать не торопясь.

Итак, у нас есть две кнопки. Они размечены в файле **markup.py**:

itm\_btn\_2 = self.set\_btn(**'INFO'**)  
itm\_btn\_3 = self.set\_btn(**'SETTINGS'**)

При нажатии кнопок происходит перехват события нажатия. Срабатывает декоратор @self.bot.message\_handler(func=**lambda** message: **True**) , который передает управление методу **handle()** c передачей в этот метод важного объекта **message**. О нем мы уже говорили. Он содержит много важных параметров.

Например, сейчас он может дать нам информацию о том, какая кнопка была нажата:

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

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

и т.д.

Зная, какая кнопка нажата, мы можем запустить соответствующий обработчик.

Например, хотим получить информацию о магазине. При этом запускается обработчик **self.pressed\_btn\_info(message).**

Код обработчика также представлен в листинге 4 (см. выше).

**def** pressed\_btn\_info(self, message):

*"""  
 обрабатывает входящие текстовые сообщения*

*от нажатия на кнопку ‘О магазине’.  
 """*

self.bot.send\_message(message.chat.id, MESSAGES[**'trading\_store'**],  
 parse\_mode=**"HTML"**,  
 reply\_markup=self.keybords.info\_menu())

Мы хотим, чтобы наш бот как-то отреагировал на нажатие кнопки, т.е., например, отправил нам какое-то сообщение. Благодаря возможностям **python-telegram-bot API** мы можем вызвать встроенную функцию **send\_message ()**. При этом мы должны знать кому отправить сообщение и знать, что должно быть в этом сообщении.

Мы указываем параметр **message.chat.id.** Он ссылается на уникальный идентификатор чата, т.е. мы знаем, что сообщение придет именно нам, в наш текущий канал (текущий экземпляр бота). Также мы указываем, что будет в ответе. Параметр – MESSAGES[**'trading\_store'**]. Вот здесь остановимся подробнее.

В листинге 4 мы видим такой импорт:

**from** settings.message **import** MESSAGES

Этого модуля у нас пока не было. Давайте создадим его в директории **settings** и сразу на будущее заполним необходимым кодом. Это код вам покажется сложным. Но на самом деле здесь просто хранятся тексты наших сообщений.

**Листинг 5. step\_3/settings/message.py**

|  |
| --- |
| *# импортируем настройки для отражения эмоджи* **from** .config **import** KEYBOARD, VERSION, AUTHOR  *# ответ пользователю при посещении блока "О магазине"* trading\_store = **"""  <b>Добро пожаловать в приложение  GroceryStore !!!</b>  Данное приложение разработано  специально для торговых представителей, далее <i>(ТП/СВ)</i>,а также для кладовщиков,  коммерческих организаций осуществляющих оптово-розничную торговлю.  ТП используя приложение GroceryStore, в удобной интуитивной форме смогут без особого труда принять заказ от клиента. GroceryStore поможет сформировать заказ и в удобном виде адресует кладовщику  фирмы для дальнейшего комплектования заказа.   """** *# ответ пользователю при посещении блока "Настройки"* settings = **""" <b>Общее руководство приложением:</b>  <i>Навигация:</i>  -<b>({}) - </b><i>назад</i> -<b>({}) - </b><i>вперед</i> -<b>({}) - </b><i>увеличить</i> -<b>({}) - </b><i>уменьшить</i> -<b>({}) - </b><i>следующий</i> -<b>({}) - </b><i>предыдующий</i>  <i>Специальные кнопки:</i>  -<b>({}) - </b><i>удалить</i> -<b>({}) - </b><i>заказ</i> -<b>({}) - </b><i>Оформить заказ</i>  <i>Общая информация:</i>  -<b>версия программы: - </b><i>({})</i> -<b>разработчик: - </b><i>({})</i>   <b>{}Ваше имя</b>  """**.format(  KEYBOARD[**'<<'**],  KEYBOARD[**'>>'**],  KEYBOARD[**'UP'**],  KEYBOARD[**'DOUWN'**],  KEYBOARD[**'NEXT\_STEP'**],  KEYBOARD[**'BACK\_STEP'**],  KEYBOARD[**'X'**],  KEYBOARD[**'ORDER'**],  KEYBOARD[**'APPLAY'**],  VERSION,  AUTHOR,  KEYBOARD[**'COPY'**], ) *# ответ пользователю при добавлении товара в заказ* product\_order = **""" Выбранный товар:  {} {} Cтоимость: {} руб  добавлен в заказ!!!  На складе осталось {} ед.  """** *# ответ пользователю при посещении блока с заказом* order = **"""  <i>Название:</i> <b>{}</b>  <i>Описание:</i> <b>{}</b>  <i>Cтоимость:</i> <b>{} руб за 1 ед.</b>  <i>Количество позиций:</i> <b>{} ед.</b>  """** order\_number = **"""  <b>Позиция в заказе № </b> <i>{}</i>  """** *# ответ пользователю, когда заказа нет* no\_orders = **""" <b>Заказ отсутствует !!!</b> """** *# ответ пользователю при подтверждении оформления заказа* applay = **""" <b>Ваш заказ оформлен !!!</b>  <i>Общая стоимость заказа составляет:</i> <b>{} руб</b>  <i>Общее количнсктво пзиций составляет:</i> <b>{} ед.</b>  <b>ЗАКАЗ НАПРАВЛЕН НА СКЛАД, ДЛЯ ЕГО КОМПЛЕКТОВКИ !!!</b> """** *# словарь ответов пользователю* MESSAGES = {  **'trading\_store'**: trading\_store,  **'product\_order'**: product\_order,  **'order'**: order,  **'order\_number'**: order\_number,  **'no\_orders'**: no\_orders,  **'applay'**: applay,  **'settings'**: settings } |

Пока не будем изучать его в подробностях. Изучим постепенно. Итак, у нас есть параметр – MESSAGES[**'trading\_store'**]. В приведенном выше листинге такой словарь есть и по ключу **'trading\_store'** мы получим вот такой интересный фрагмент:

trading\_store = **"""  
  
<b>Добро пожаловать в приложение  
 GroceryStore !!!</b>  
  
Данное приложение разработано   
специально для торговых представителей,  
далее <i>(ТП/СВ)</i>,а также для кладовщиков,   
коммерческих организаций осуществляющих  
оптово-розничную торговлю.  
  
ТП используя приложение GroceryStore,  
в удобной интуитивной форме смогут без  
особого труда принять заказ от клиента.  
GroceryStore поможет сформировать заказ  
и в удобном виде адресует кладовщику   
фирмы для дальнейшего комплектования заказа.   
"""**

Похоже на строку, в которой есть HTML-разметка. Создавая наш бот, мы можем реализовать форматирование с помощью HTML-тегов и атрибутов (параметр parse\_mode=**"HTML"**).

Остался еще один непонятный параметр: reply\_markup=self.keybords.info\_menu(). Этот параметр отвечает за быстрый ответ. Мы хотим, чтобы, перейдя на страницу с информацией о магазине, была возможность вернуться обратно. А значит должна быть кнопка возврата. Метод **info\_menu()** как раз и отвечает за формирование кнопки возврата. Но этот метод мы пока не создавали. Значит нужно создать.

Класс **Keyboards()** у нас находится в модуле **markup.py**. Дополним интерфейс классами следующими методами:

**Листинг 6. step\_3/markup/markup.py**

**def** info\_menu(self):  
 *"""  
 Создает разметку кнопок в меню 'О магазине'  
 """* self.markup = ReplyKeyboardMarkup(**True**, **True**)  
 itm\_btn\_1 = self.set\_btn(**'<<'**)  
 *# рассположение кнопок в меню* self.markup.row(itm\_btn\_1)  
 **return** self.markup  
  
**def** settings\_menu(self):  
 *"""  
 Создает разметку кнопок в меню 'Настройки'  
 """* self.markup = ReplyKeyboardMarkup(**True**, **True**)  
 itm\_btn\_1 = self.set\_btn(**'<<'**)  
 *# рассположение кнопок в меню* self.markup.row(itm\_btn\_1)  
 **return** self.markup

А вот и метод **info\_menu()** с кнопкой возврата. Но как бот узнает, что ему нужно вернуться обратно?

Нужно обработать нажатие.

В модуле **handler\_all\_text.py** мы добавим еще одну ветвь:

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

И обработчик:

**def** pressed\_btn\_back(self, message):  
 *"""  
 обрабатывает входящие текстовые сообщения от нажатия на кнопку ‘Назад’.  
 """* self.bot.send\_message(message.chat.id, **"Вы вернулись назад"**,  
 reply\_markup=self.keybords.start\_menu())

Мы вернемся назад, получим сообщение **"Вы вернулись назад"** и вновь окажемся на главном меню (отображение главного меню обеспечит вызов метода **start\_menu()**).

Вот такой получился подробный разбор!

Но это еще не все. Мы создали новый модуль с обработчиками **handler\_all\_text.py**. При этом, важно вспомнить, что все модули в нашем проекте привязаны к одному центральному модулю **handler\_main.py**. Нужно дополнить его некоторыми важными командами:

**from** handlers.handler\_all\_text **import** HandlerAllText

…

self.handler\_all\_text = HandlerAllText(self.bot)

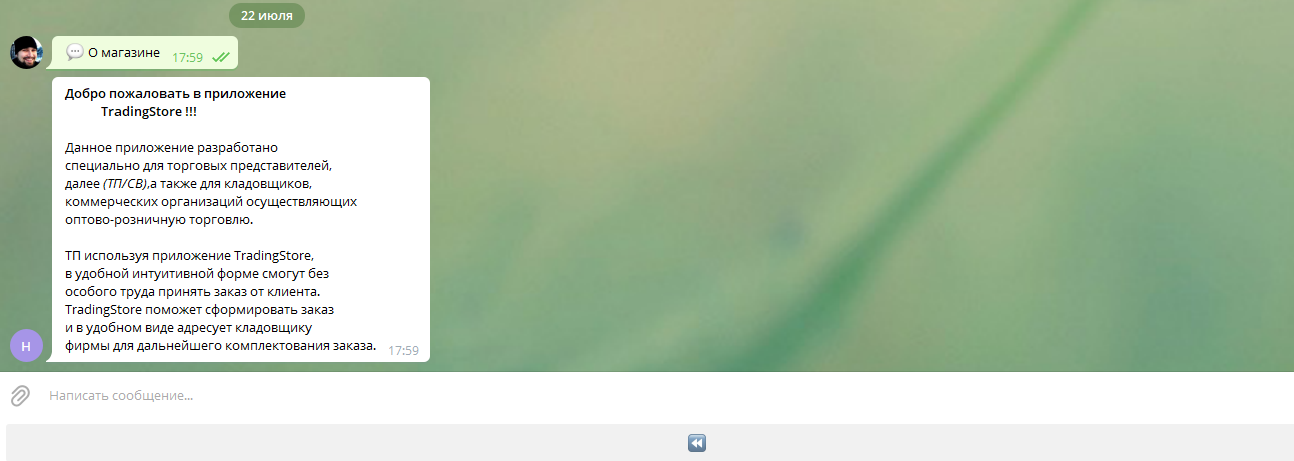
self.handler\_all\_text.handle()

И теперь его код будет выглядеть следующим образом:

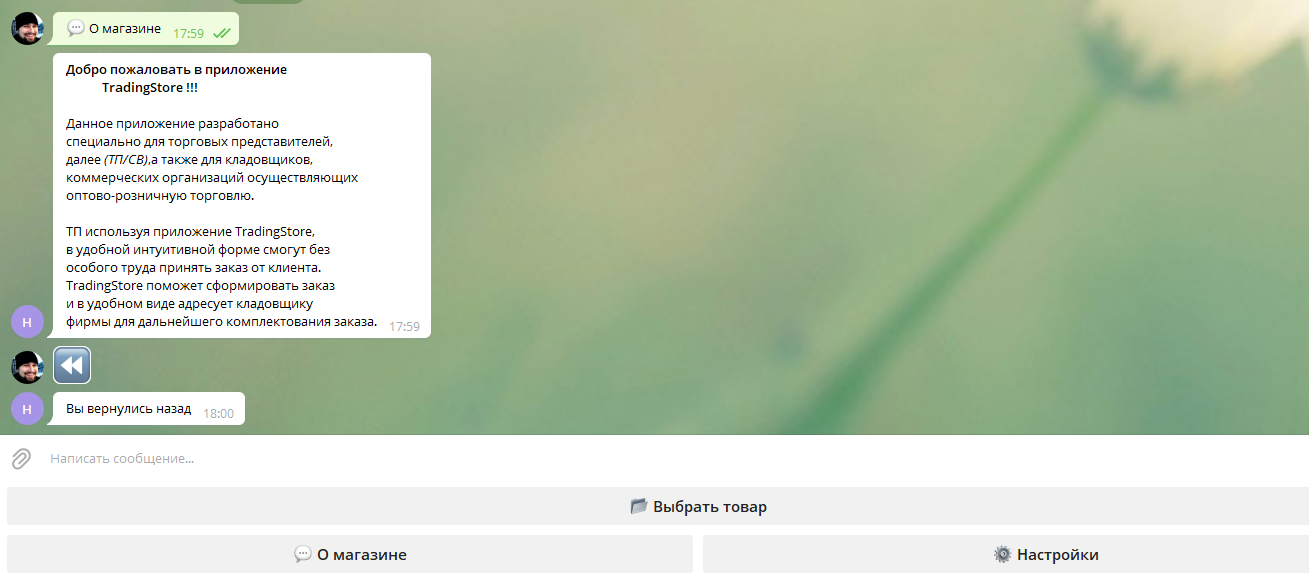
**Листинг 7. step\_3/handlers/handler\_main.py**

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

Запустим проект и сразу перейдем в раздел «О магазине»:



Работает. А теперь попробуем вернуться назад:



Тоже работает.

# ДОБАВЛЯЕМ ФУНКЦИОНАЛ «НАСТРОЙКИ»

Теперь нам по такому же алгоритму необходимо добавить функционал для отображения настроек нашего бота.

В модуль **handler\_all\_text.py** обработка нажатия соответствующей кнопки уже добавлена ранее:

…

**def** pressed\_btn\_settings(self, message):  
 *"""  
 обрабатывает входящие текстовые сообщения*

*от нажатия на кнопку ‘Настройки’.  
 """* self.bot.send\_message(message.chat.id, MESSAGES[**'settings'**],  
 parse\_mode=**"HTML"**,  
 reply\_markup=self.keybords.settings\_menu())

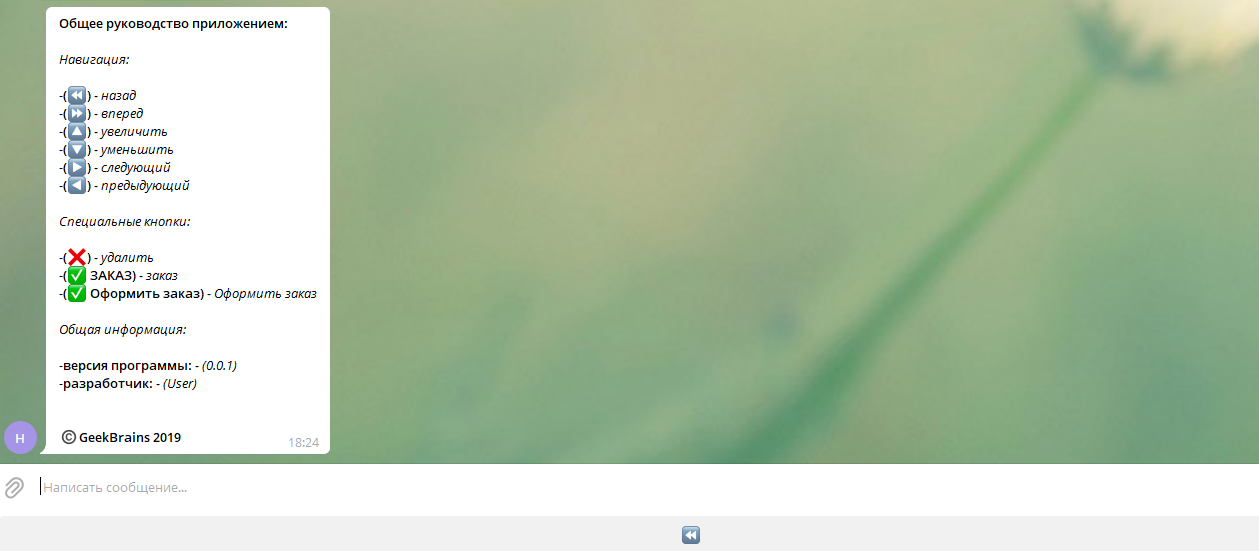
…

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

self.pressed\_btn\_settings(message)

В модуле **message.py** все необходимое уже есть, **markup.py** мы также заполнили.

Вновь выполним запуск и перейдем в раздел настроек.

Все работает.

При необходимости можно легко вернуться назад.

Здесь прошу вас сделать паузу и не спеша, как следует, еще раз пройти весь представленный материал.

Вначале все покажется сложным, но когда вы пройдете материал несколько раз и погрузитесь в код, поймаете ошибки, исправите их, сделаете отладку, постепенно все хорошо усвоится

# Вид сверху на рукопожатие над деловыми документамиЗАКЛЮЧЕНИЕ

Возможности Telegram Bot API позволяют настраивать обработку событий вашего бота. Общий принцип создания обработчиков несложный, но нужно знать какие для этого используются команды и какие они принимают параметры. Правильным будет подробное изучение возможностей Telegram Bot API по документации.

## КЛЮЧЕВЫЕ ВЫВОДЫ

* Возможность создания привычных графических интерфейсов позволяет создавать для бота традиционные средства управления.
* Мы можем реализовать форматирование элементов интерфейса даже посредством HTML-тегов и атрибутов.
* Задача разработчика – создавать элементы управления, настраивать их внешний вид, далее создавать и привязывать обработчики с необходимым функционалом.
* Представленный алгоритм похож на написание PyQt-приложений, когда мы создаем окно, размещаем в нем кнопки, отлавливаем событие нажатия и запускаем привязанные к этим событиям обработчики.

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

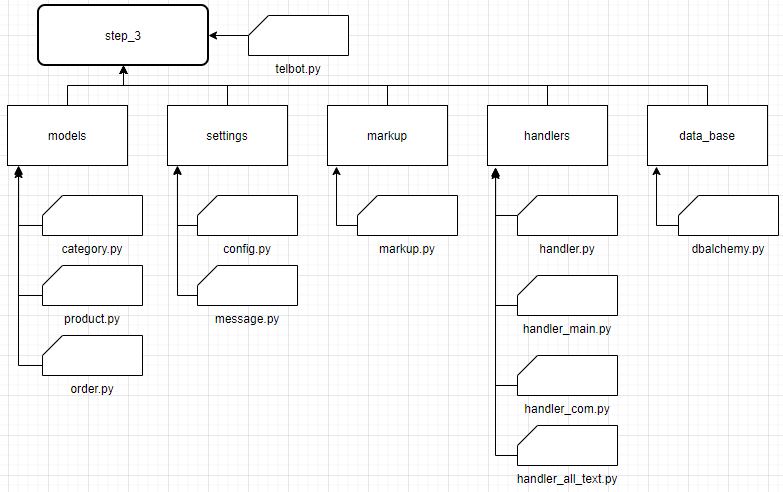


Рис. 1. Структура проекта