Теоретические материалы к занятию 6

авайте используя уже полученные знания создадим простого бота онлайн магазина, с использованием базы данных.

```
import os
import logging
from aiogram import executor, types
from aiogram.types import ReplyKeyboardMarkup,
ReplyKeyboardRemove
from aiogram import Bot, Dispatcher, types
from aiogram.contrib.fsm_storage.memory import MemoryStorage
import config


bot = Bot(token=config.BOT_TOKEN,
parse_mode=types.ParseMode.HTML)
storage = MemoryStorage()
dp = Dispatcher(bot, storage=storage)
db = DatabaseManager('database.db')

WEBAPP_HOST = "0.0.0.0"
WEBAPP_PORT = int(os.environ.get("PORT", 5000))
user_message = 'Пользователь'
admin_message = 'Админ'


async def on_startup(dp):
   logging.basicConfig(level=logging.INFO)
   db.create_tables()

   await bot.delete_webhook()
   await bot.set_webhook(config.WEBHOOK_URL)


async def on_shutdown():
   logging.warning("Shutting down..")
   await bot.delete_webhook()
   await dp.storage.close()
   await dp.storage.wait_closed()
   logging.warning("Bot down")


if __name__ == '__main__':

      executor.start_polling(dp, on_startup=on_startup,
skip_updates=False)
```

Пропишем необходимые данные в config

```
BOT_TOKEN = 'Ваш токен'

PROJECT_NAME = 'store-bot-example'
ADMINS = [000000000, 1234567890]
```

Создадим класс базы данных и укажем необходимые поля и методы

```
class DatabaseManager(object):

   def __init__(self, path):
       self.conn = lite.connect(path)
       self.conn.execute('pragma foreign_keys = on')
       self.conn.commit()
       self.cur = self.conn.cursor()

   def create_tables(self):
       self.query(
           'CREATE TABLE IF NOT EXISTS products (idx text, title
text, body text, photo blob, price int, tag text)')
       self.query('CREATE TABLE IF NOT EXISTS orders (cid int,
usr_name text, usr_address text, products text)')
       self.query('CREATE TABLE IF NOT EXISTS cart (cid int, idx
text, quantity int)')
       self.query('CREATE TABLE IF NOT EXISTS categories (idx
text, title text)')
       self.query('CREATE TABLE IF NOT EXISTS wallet (cid int,
balance real)')
       self.query('CREATE TABLE IF NOT EXISTS questions (cid int,
question text)')

   def query(self, arg, values=None):
       if values == None:
           self.cur.execute(arg)
       else:
           self.cur.execute(arg, values)
       self.conn.commit()

   def fetchone(self, arg, values=None):
       if values == None:
           self.cur.execute(arg)
       else:
           self.cur.execute(arg, values)
       return self.cur.fetchone()

   def fetchall(self, arg, values=None):
       if values == None:
           self.cur.execute(arg)
       else:
           self.cur.execute(arg, values)
       return self.cur.fetchall()
```

```
    def __del__(self):
        self.conn.close()
```

В данном классе мы указали методы для создания таблиц, для запроса, а также выборки, как одного, так и всех элементов.

Отлично основа у нас уже есть. Давайте также сразу пропишем свои фильтры для проверки на администратора и пользователя

```
from aiogram.types import Message
from aiogram.dispatcher.filters import BoundFilter
from config import ADMINS

class IsUser(BoundFilter):
    async def check(self, message: Message):
        return message.from_user.id not in ADMINS

class IsAdmin(BoundFilter):
    async def check(self, message: Message):
        return message.from_user.id in ADMINS
```

В данных фильтрах мы опять же наследуемся от класса BoundFilter и возвращаем булево значение: есть ли пользователь в списке администраторов.

Основные приготовления готовы. Нам остается прописать хендлеры, клавиатуры и состояния. Да у нас пока не будет настоящей системы оплаты, но для примера нам пока это отлично подходит.

Давайте добавим обычную клавиатуру для меню.

```
from aiogram.types import ReplyKeyboardMarkup

back_message = '👈 Назад'
confirm_message = '✅ Подтвердить заказ'
all_right_message = '✅ Все верно'
cancel_message = '🚫 Отменить'

def confirm_markup():
    markup = ReplyKeyboardMarkup(resize_keyboard=True,
selective=True)
    markup.add(confirm_message)
    markup.add(back_message)

    return markup

def back_markup():
    markup = ReplyKeyboardMarkup(resize_keyboard=True,
selective=True)
    markup.add(back_message)
```

```
    return markup

def check_markup():
    markup = ReplyKeyboardMarkup(resize_keyboard=True,
selective=True)
    markup.row(back_message, all_right_message)

    return markup


def submit_markup():
    markup = ReplyKeyboardMarkup(resize_keyboard=True,
selective=True)
    markup.row(cancel_message, all_right_message)

    return markup
```

Назначения для нашей клавиатуры достаточно простое, поэтому не будем заострять на нем внимание.

Ну что ж давайте перейдем к инлайн клавиатурам и добавим необходимые. Они у нас также будут выполнять роль администраторского меню(добавление товара и т.д)
Начнем с клавиатуры для категорий.

```
from aiogram.types import InlineKeyboardMarkup,
InlineKeyboardButton
from aiogram.utils.callback_data import CallbackData


category_cb = CallbackData('category', 'id', 'action')


def categories_markup():
    global category_cb

    markup = InlineKeyboardMarkup()
    for idx, title in db.fetchall('SELECT * FROM categories'):
        markup.add(InlineKeyboardButton(title,
callback_data=category_cb.new(id=idx, action='view')))

    return markup
```

Данная клавиатура делает запрос в базу данных для выборки всех категории и создании нужного количества кнопок.

Перейдем к клавиатуре имитирующей корзину покупателя, где он может выбрать необходимое количество товара.

```
product_cb = CallbackData('product', 'id', 'action')

def product_markup(idx, count):

   global product_cb

   markup = InlineKeyboardMarkup()
   back_btn = InlineKeyboardButton('⬅️',
callback_data=product_cb.new(id=idx, action='decrease'))
   count_btn = InlineKeyboardButton(count,
callback_data=product_cb.new(id=idx, action='count'))
   next_btn = InlineKeyboardButton('➡️',
callback_data=product_cb.new(id=idx, action='increase'))
   markup.row(back_btn, count_btn, next_btn)

   return markup
```

Пользователю доступно в данной клавиатуре возможность увеличения/уменьшения количества товара с помощью кнопок.
Ну и конечно же создадим клавиатуру для добавления товара в корзину

```
product_cb = CallbackData('product', 'id', 'action')


def product_markup(idx='', price=0):

   global product_cb

   markup = InlineKeyboardMarkup()
   markup.add(InlineKeyboardButton(f'Добавить в корзину -
{price}₽', callback_data=product_cb.new(id=idx, action='add')))

   return markup
```

Отлично! Теперь можно добавить машину состояний и переходить к хендлерам

```
from aiogram.dispatcher.filters.state import StatesGroup, State


class CheckoutState(StatesGroup):
   check_cart = State()
   name = State()
```

```
    address = State()
    confirm = State()


class ProductState(StatesGroup):
    title = State()
    body = State()
    image = State()
    price = State()
    confirm = State()


class CategoryState(StatesGroup):
    title = State()


class SosState(StatesGroup):
    question = State()
    submit = State()


class AnswerState(StatesGroup):
    answer = State()
    submit = State()
```

Все необходимые состояния созданы, время реализовать хендлеры. Здесь для удобства создадим папку хендлер и в ней папки user и admin. Создадим в папке admin файл add, который будет обрабатывать возможность добавления нового товара.

```
@dp.message_handler(IsAdmin(), lambda message: message.text.isdigit(),
state=ProductState.price)
async def process_price(message: Message, state: FSMContext):

    async with state.proxy() as data:

        data['price'] = message.text

        title = data['title']
        body = data['body']
        price = data['price']

        await ProductState.next()
        text = f'<b>{title}</b>\n\n{body}\n\nЦена: {price} рублей.'

        markup = check_markup()

        await message.answer_photo(photo=data['image'],
                                   caption=text,
                                   reply_markup=markup)
```

```
@dp.message_handler(IsAdmin(), lambda message: message.text not in
[back_message, all_right_message], state=ProductState.confirm)
async def process_confirm_invalid(message: Message, state: FSMContext):
   await message.answer('Такого варианта не было.')


@dp.message_handler(IsAdmin(), text=back_message,
state=ProductState.confirm)
async def process_confirm_back(message: Message, state: FSMContext):

   await ProductState.price.set()

   async with state.proxy() as data:

       await message.answer(f"Изменить цену с <b>{data['price']}</b>?",
reply_markup=back_markup())


@dp.message_handler(IsAdmin(), text=all_right_message,
state=ProductState.confirm)
async def process_confirm(message: Message, state: FSMContext):

   async with state.proxy() as data:

       title = data['title']
       body = data['body']
       image = data['image']
       price = data['price']

       tag = db.fetchone(
           'SELECT title FROM categories WHERE idx=?',
(data['category_index'],))[0]
       idx = md5(' '.join([title, body, price, tag]
                          ).encode('utf-8')).hexdigest()

       db.query('INSERT INTO products VALUES (?, ?, ?, ?, ?, ?)',
                (idx, title, body, image, int(price), tag))

   await state.finish()
   await message.answer('Готово!', reply_markup=ReplyKeyboardRemove())
   await process_settings(message)


# delete product


@dp.callback_query_handler(IsAdmin(), product_cb.filter(action='delete'))
async def delete_product_callback_handler(query: CallbackQuery,
callback_data: dict):

   product_idx = callback_data['id']
   db.query('DELETE FROM products WHERE idx=?', (product_idx,))
   await query.answer('Удалено!')
   await query.message.delete()


async def show_products(m, products, category_idx):

   await bot.send_chat_action(m.chat.id, ChatActions.TYPING)
```

```
   for idx, title, body, image, price, tag in products:

       text = f'<b>{title}</b>\n\n{body}\n\nЦена: {price} рублей.'

       markup = InlineKeyboardMarkup()
       markup.add(InlineKeyboardButton(
           '🗑️ Удалить', callback_data=product_cb.new(id=idx,
action='delete')))

       await m.answer_photo(photo=image,
                            caption=text,
                            reply_markup=markup)

   markup = ReplyKeyboardMarkup()
   markup.add(add_product)
   markup.add(delete_category)

   await m.answer('Хотите что-нибудь добавить или удалить?',
reply_markup=markup)
```

файл orders для заказов

```
from aiogram.types import Message
from bot import dp, db, IsAdmin
from handlers.user.menu import orders



@dp.message_handler(IsAdmin(), text=orders)
async def process_orders(message: Message):
   orders = db.fetchall('SELECT * FROM orders')

   if len(orders) == 0:
       await message.answer('У вас нет заказов.')
   else:
       await order_answer(message, orders)


async def order_answer(message, orders):
   res = ''

   for order in orders:
       res += f'Заказ <b>№{order[3]}</b>\n\n'

   await message.answer(res)
```

и файл questions для обработки вопросов от пользователей

```
from handlers.user.menu import questions
from aiogram.dispatcher import FSMContext
from aiogram.utils.callback_data import CallbackData
```

```
from bot import all_right_message, cancel_message, submit_markup,
AnswerState, dp, db, bot, IsAdmin
from aiogram.types import Message, CallbackQuery,
InlineKeyboardMarkup, InlineKeyboardButton, ReplyKeyboardRemove
from aiogram.types.chat import ChatActions


question_cb = CallbackData('question', 'cid', 'action')


@dp.message_handler(IsAdmin(), text=questions)
async def process_questions(message: Message):

   await bot.send_chat_action(message.chat.id,
ChatActions.TYPING)
   questions = db.fetchall('SELECT * FROM questions')

   if len(questions) == 0:

       await message.answer('Нет вопросов.')

   else:

       for cid, question in questions:

           markup = InlineKeyboardMarkup()
           markup.add(InlineKeyboardButton(
               'Ответить', callback_data=question_cb.new(cid=cid,
action='answer')))

           await message.answer(question, reply_markup=markup)


@dp.callback_query_handler(IsAdmin(),
question_cb.filter(action='answer'))
async def process_answer(query: CallbackQuery, callback_data:
dict, state: FSMContext):

   async with state.proxy() as data:
       data['cid'] = callback_data['cid']

   await query.message.answer('Напиши ответ.',
reply_markup=ReplyKeyboardRemove())
   await AnswerState.answer.set()


@dp.message_handler(IsAdmin(), state=AnswerState.answer)
async def process_submit(message: Message, state: FSMContext):

   async with state.proxy() as data:
       data['answer'] = message.text

   await AnswerState.next()
   await message.answer('Убедитесь, что не ошиблись в ответе.',
reply_markup=submit_markup())
```

```
@dp.message_handler(IsAdmin(), text=cancel_message,
state=AnswerState.submit)
async def process_send_answer(message: Message, state:
FSMContext):
   await message.answer('Отменено!',
reply_markup=ReplyKeyboardRemove())
   await state.finish()


@dp.message_handler(IsAdmin(), text=all_right_message,
state=AnswerState.submit)
async def process_send_answer(message: Message, state:
FSMContext):

   async with state.proxy() as data:

       answer = data['answer']
       cid = data['cid']

       question = db.fetchone(
           'SELECT question FROM questions WHERE cid=?',
(cid,))[0]
       db.query('DELETE FROM questions WHERE cid=?', (cid,))
       text = f'Вопрос: <b>{question}</b>\n\nОтвет:
<b>{answer}</b>'

       await message.answer('Отправлено!',
reply_markup=ReplyKeyboardRemove())
       await bot.send_message(cid, text)

   await state.finish()
```