## TELEGRAM FINANCE BOT

Для контроля ежедневных расходов используется телеграм бот, который учитывает расходы по заранее спланированным статьям расходов. Также есть возможность внести траты без статьи расхода, с пояснением о том, что и зачем было приобретено или оплачено в качестве предоставленных услуги.

Приложение написано на языке python с использованием библиотеки aiogram. В качестве базы данных для хранения пользовательской информации и статистики по расходам и доходам, используется на текущий момент сервис google sheets. Каждый внесенный запрос добавляет запись в таблицу на стороне google в определенные столбцы. Новый запрос начинается с новой строки.

**Планируется перевести сервис на postgresql 14, исключить google sheets в качестве основной БД и полностью отказаться от использования этого сервиса.**

**Также нужно перенести приложение бота с aiogram 2x версии на aiogram 3x версию.**

Репозиторий разработанного приложения находится на сервисе GitHub ***https://github.com/sdkinfotech/finance\_bot.git***

Сервис разработан в начале 2022 года. Поэтому требуется рефакторинг и существенные правки в структуре самого бота, начиная от каталога проекта и файловой структуры, до рефакторинга кода, классов, функций, названий переменных.

Также требуется продолжить разработку функционала контроля доходов. Сейчас бот умеет контролировать только ветку расходов.

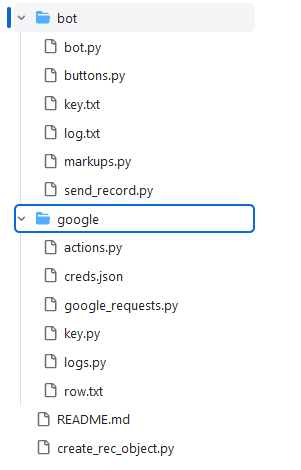
После реализации правильной работы накопления пользовательский данных, требуется составить мониторинг доходов, расходов, разбивка по статьям доходов и расходов, совместимые графики для контроля корреляции и сравнения данных. В общих целях – разработать дашборд, который будет предоставлять полную информацию по доходам и расходам. Разработка дашборда выходит за рамки рефакторинга. Его планирование и этапы разработки будут вынесены в отдельный документ.

## Структура проекта до внесения изменений.

До начала всех изменений требуется оценка того, с чем мы будем иметь дело, что потребуется переосмыслить, и уже опираясь на вводные данные, составить план поэтапного внедрения изменений в проект.

На текущий момент каталог проекта состоит из следующих файлов и каталогов.

###### Структура проекта



* **bot – Каталог, содержащий исходный код самого бота.**
* **google – Каталог с модулями взяимодействия с сервисом google sheets**
* **create\_rec\_object.py – Модуль, содержащий в себе класс создания запроса, сформированного UI telegram бота, для его даьнейшей отправки в google sheets**
* **README.md – Стандартное описание сервиса**

###### bot каталог

* **bot.py – Главный модуль бота**
* **buttons.py – Модуль с описанием кнопок бота**
* **key.txt – текстовый с токеном телеграм**
* **log.txt – файл лога**
* **markups.py – модуль с описанием маркапов**
* **send\_record.py – модуль для отправки записи в google sheets**

###### google каталог

* **actions.py – модуль содержит экземпляры классов для действий таких как выборка данных и запись данных в таблицу на стороне google sheets на основе google requests**
* **creds.json – содержит в себе токены для подключения к сервисам google**
* **google\_requests.py – описание взаимодействия с сервисами google**

**key.py - Модуль содержит в себе переменную api\_key в виде словаря. В словаре хранятся пути для авторизации google sheets и ID таблицы**

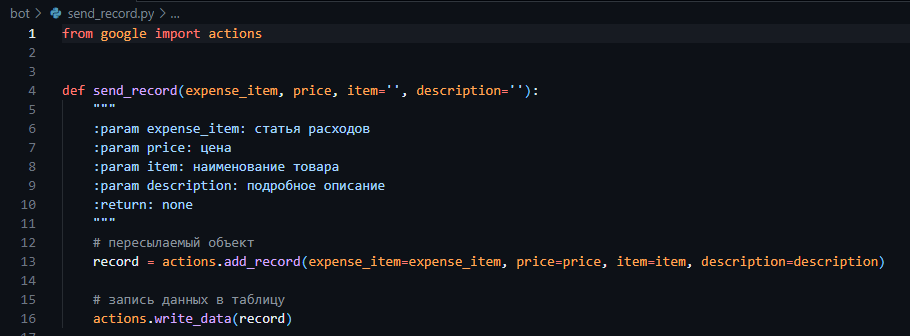
* **logs.py – модуль для описания системы логирования**
* **row.txt – хранит значение позиции курсора в таблице. Запоминает последнюю запись, созданную в таблице, а если быть точнее - значение номера строки.**

## **Удаление сервиса google sheets**

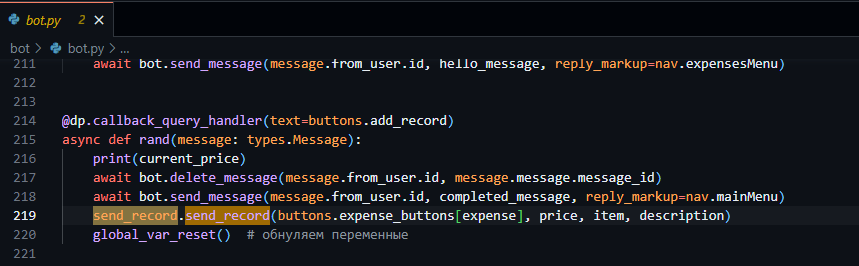
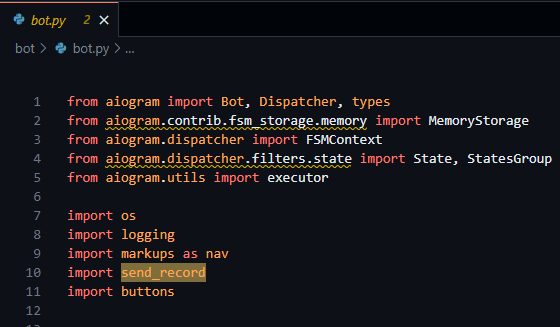
**Целью этого этапа работ, является получение правильного объекта из телеграм бота и вывод его отладочной информации в лог и в консоль.**

Первым этапом станет удаление сервиса google и всех вызовов, связанных с его работой.

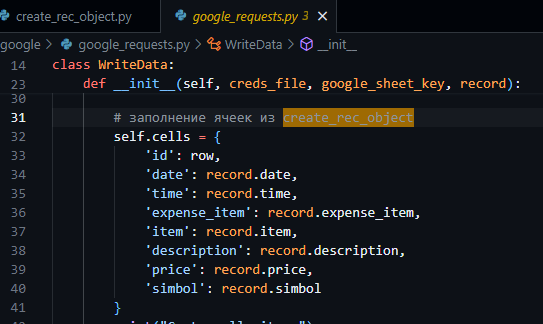
Начнем с каталога **bot**



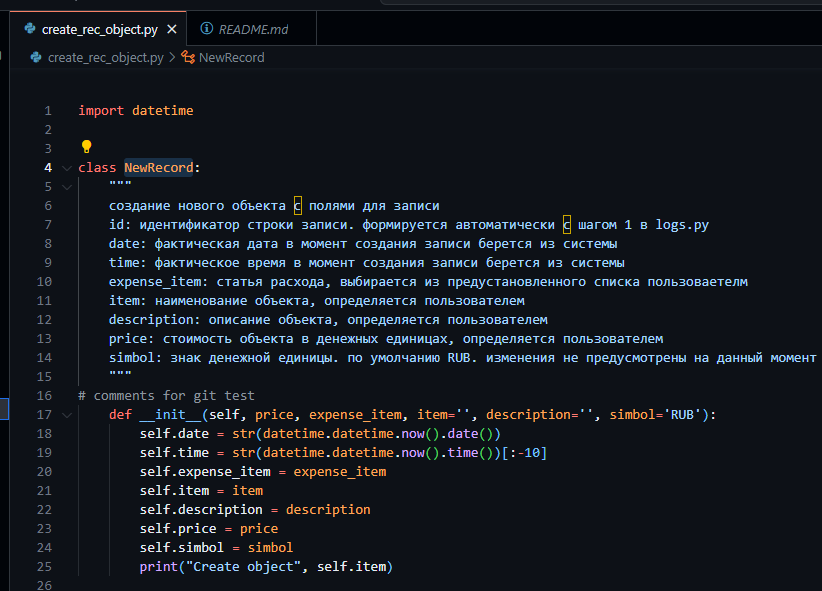
В нем присутствует молуль **send\_record.py** и **метод send\_record** который раскладывает данные пользователя, получаемые из телеграм, и перенаправляет их на запрос для google sheets. Для начала его нужно переработать для взаимодействия с postgresql.



**send\_record**  передавался в класс **WriteData** в аргументе **record**



Cам **record** является экземляром класса **NewRecord** из модуля **create\_rec\_object.py**



Он формирует как раз всю строку данных, которая дальше передавалась **в google** Аргументы, которые имеют значения по умолчанию, не передавались, если в боте пропускались эти поля. И переопределялись если пользователь что-либо писал в эти поля.

**Новая схема формирования должна выглядеть следующим образом:**

Сразу подготовим два новых файла

Создать модуль отправки данных  **db.py**, чтобы было понятно, что этот модуль работает с БД и содержит в себе классы с методами различных запросов.

Создать файл **db\_init.sql**, в котором будет описана развертка базы данных, и создание необходимой структуры таблиц. К ней вернемся в следующем разделе.

Порядок изменений:

1. Модуль **send\_record.py** переименовать **в operations.py**
2. Сам класс **NewRecord** перенести в **operations.py**
3. **NewRecord** формирует объект запроса, с заполненными полями, полученными из телеграм бота при заполнении пользователем,
4. Из **NewRecord**  удалить строки

**self.date = str(datetime.datetime.now().date())**

**self.time = str(datetime.datetime.now().time())[:-10]**

Дата и время записи будут созданы на стороне postgres

5.Метод **send\_record** сделать частью класса **NewRecord** и описать в нем отправку в БД.

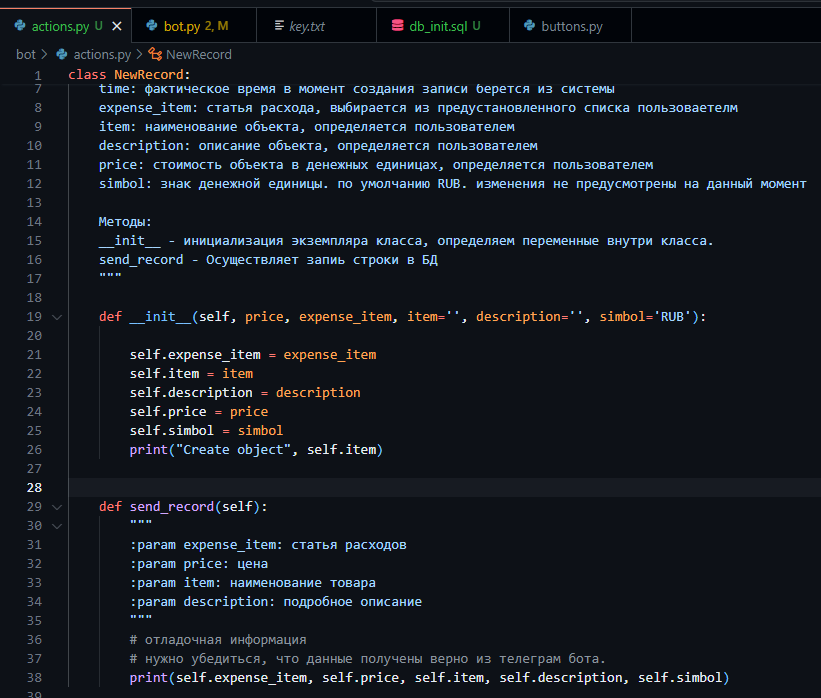
6.Весь каталог **google** нужно полностью удалить.

7. Провести тестирование, проверить отладочную информацию, пока работаем без базы.

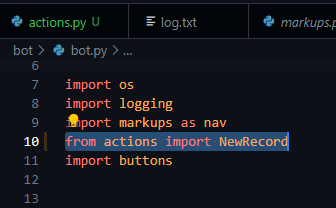
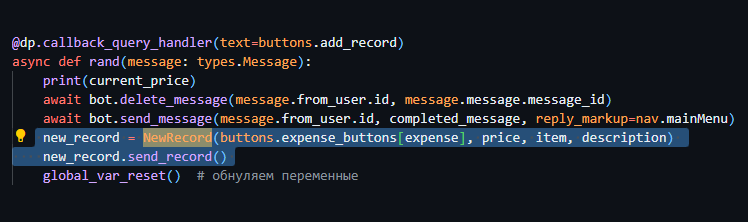
Предполагается, что удаление каталога google вызовет ошибки, которые невозможно проанализировать позже. на этапе планирования, все они будут обработаны

###### Представление модуля actions.py после внесения изменений.

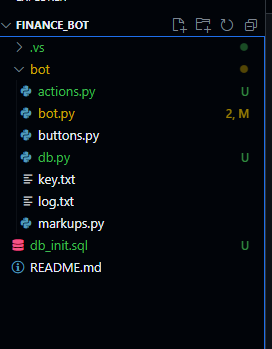
Тут предполагается, что send\_record будет выводить отладочное сообщение в консоль.После добавления и настройки БД коннектора, тут будет выполняться отправка записи в таблицы.



Соответствующие изменения требуется внести в **bot.py**



После удаления каталога google sheets и всех упоминаний о нем, дерево выглядит так:



## Миграция на aiogram 3.x

## Подготовка, первичный анализ.

*На данном этапе будут рассмотрены только ключевые идеи и ошибки, возникающие в процессе миграции.*

*Правок достаточно много и они неуместны в текущем документе,*

*все изменения можно отследить потом в ветке migration в git*

Требуется внести ряд доработок в код, изменить импорты, так как в новой версии aiogram многое поменялось, не используется, или перенесено в другие разделы библиотеки.

Изменение импортов **bot.py**

**~~from aiogram.contrib.fsm\_storage.memory import MemoryStorage~~**

**from aiogram.fsm.storage.memory import MemoryStorage**

**~~from aiogram.dispatcher.filters.state import State, StatesGroup~~**

**from aiogram.fsm.state import State, StatesGroup**

**~~from aiogram.dispatcher import FSMContext~~**

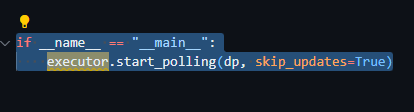
**from aiogram.fsm.context import FSMContext**

**~~from aiogram import executor~~**

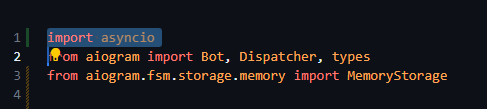
**~~executor~~ Убран совасем в aiogram 3.x**

По этой причине следует переработать запуск бота.

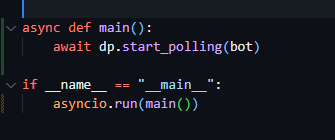
Сейчас старт бота выглядит так:



потребуется импорт asyncio



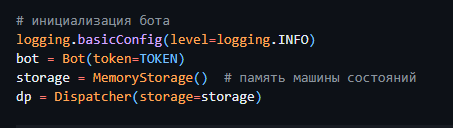
Также добавим def main и перепишем вызов старта



Также Dispatcher тоже нуждается в переработке в него больше не нужно передавать **bot**

**~~dp = Dispatcher(bot, storage=storage)~~**

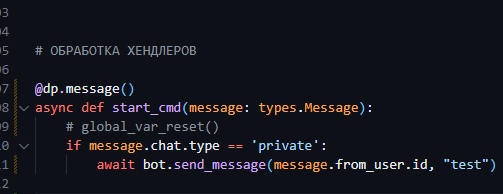
**dp = Dispatcher(bot, storage=storage)**



Хендлеры тоже претерпели изменения. Для тестового запуска используем start

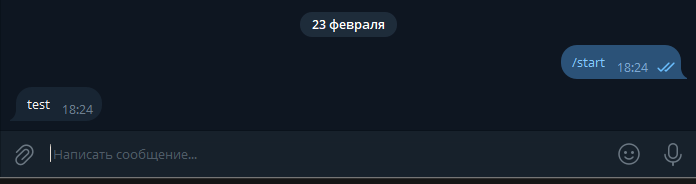
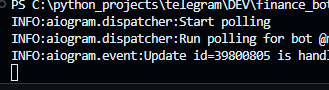


Переработаем его таким образом



Все функции и остальной код нужно временно закомментировать, для того чтобы провести первый тестовый запуск на aiogram 3.x после отладки всех ошибок, поэтапно можно переходить к следующим шагам отладки и внесения изменений миграции.

На этом этапе все должно заработать для тестового запуска

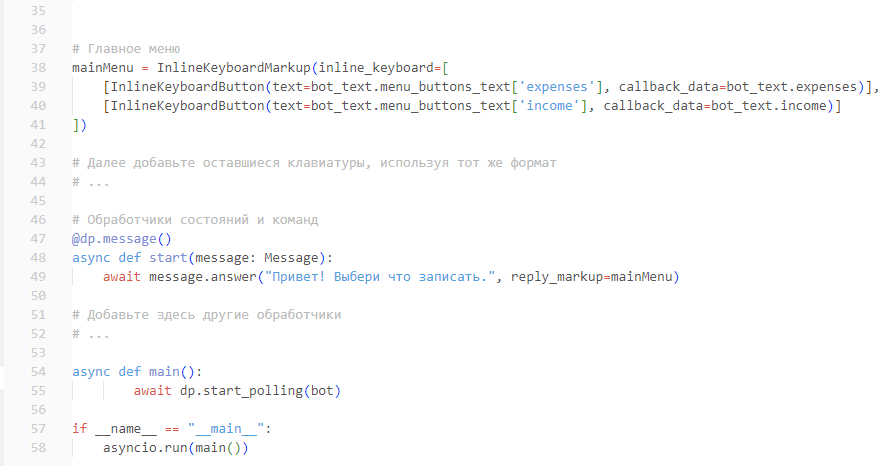
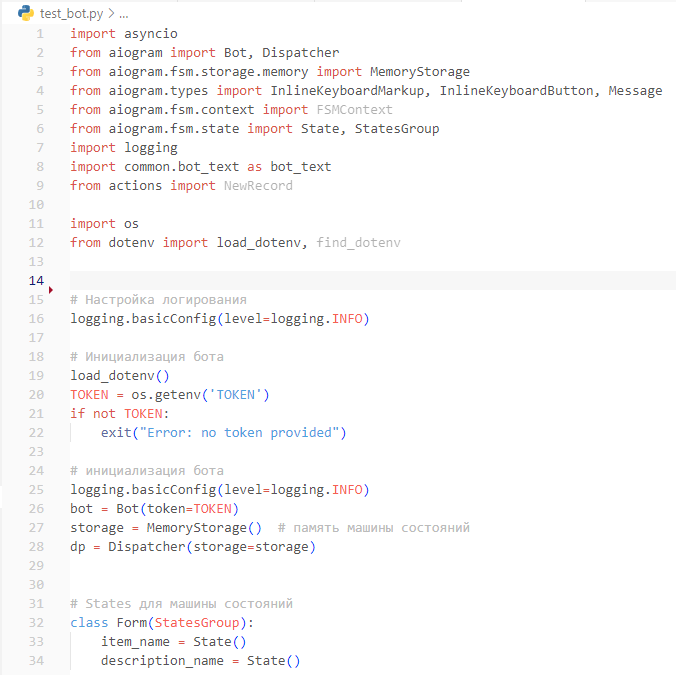


## Миграция на aiogram 3.x

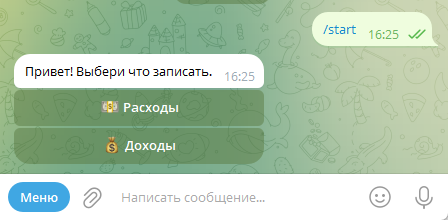
## Подготовка стендового модуля

Для тестирования инлайн кнопок и клавиатур, которые тоже подверглись изменениям в aiogram 3x лучше всего создать тестовый стенд, для того чтобы отладить существующий код. Кроме того, это все тесно взаимодействует с хендлерами и роутерами, что сильно затруднит отладку при миграции на новую версию фреймворка.

Для тестов создадим **test\_bot.py. С**разу нужно создать **в корне .env** файл в котором создать TOKEN переменную поместить токен бота в нее. Добавить в корнефайл **requirements** и в нем указать зависимости, **aiogram** и **python-dotenv Полный код test\_bot.py**



Убедимся что все работает запустив **test\_bot.py**



Тестовый модуль нам понадобится для экспериментов с кнопками и клавиатурами, так как изначально этот модуль был написан на **aiogram 3.x.** Все что будет в нем написано будет сравниваться со старой версией, затем будут частями блочно вноситься корректировки. Те участки кода, которые будут вызывать ошибки, комментируются временно до внесения изменений, либо полностью удаляются. Иногда потребуется заново переписать весь функционал в некоторых участках кода. Это намного проще, чем пытаться адаптировать старый код.

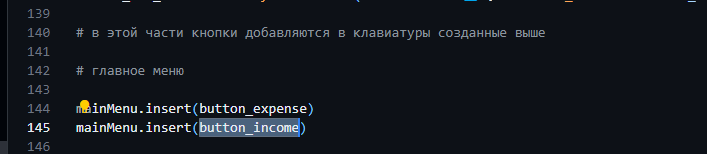
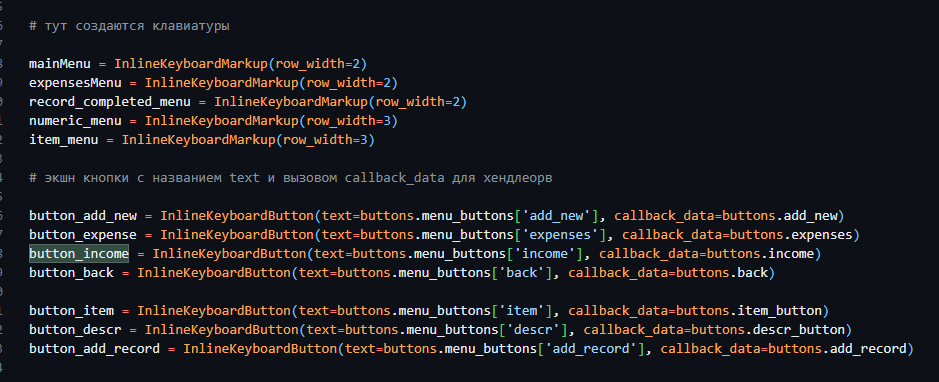
# Миграция на aiogram 3.x

# Роутеры, хендлеры, клавиатуры и кнопки

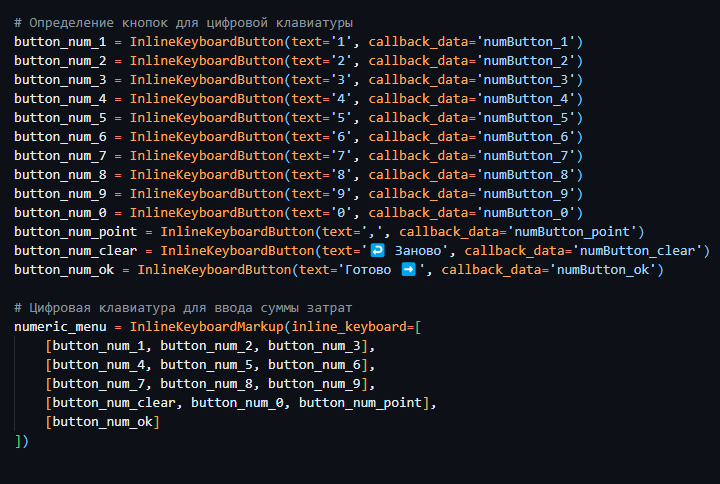
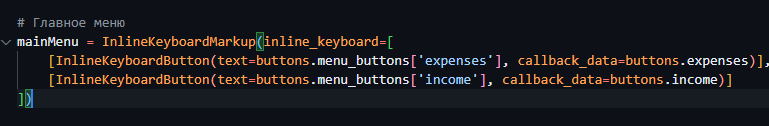
Новый подход обработки хендлеров подразумевает что будет выполнен список в списке.

Нужно привести все клавиатуры к подобной структуре.

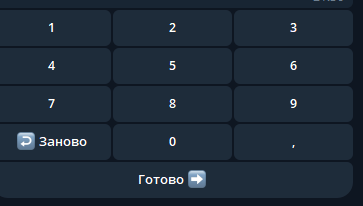
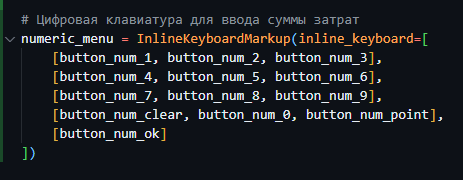
Раньше было вот так.



Теперь мы описываем клавиатуру в виде матричной сетки.

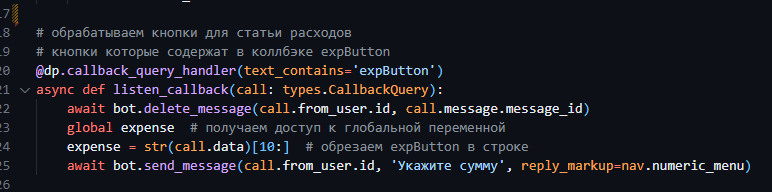


На мой взгляд таким образом создавать клавиатуры гораздо удобнее и можно проектировать в коде клавиатуру и довольно точно представлять, как она будет выглядеть после запуска

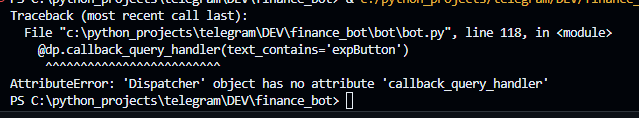


Далее можно рассмотреть динамически создаваемые клавиатуры через билдер и распаковку через циклы. Но для миграции и отладки функционала это пока не требуется.

Если мы попробуем запустить бота, раскомментировав хотя бы один хендлер с коллбэком



То получим ошибку



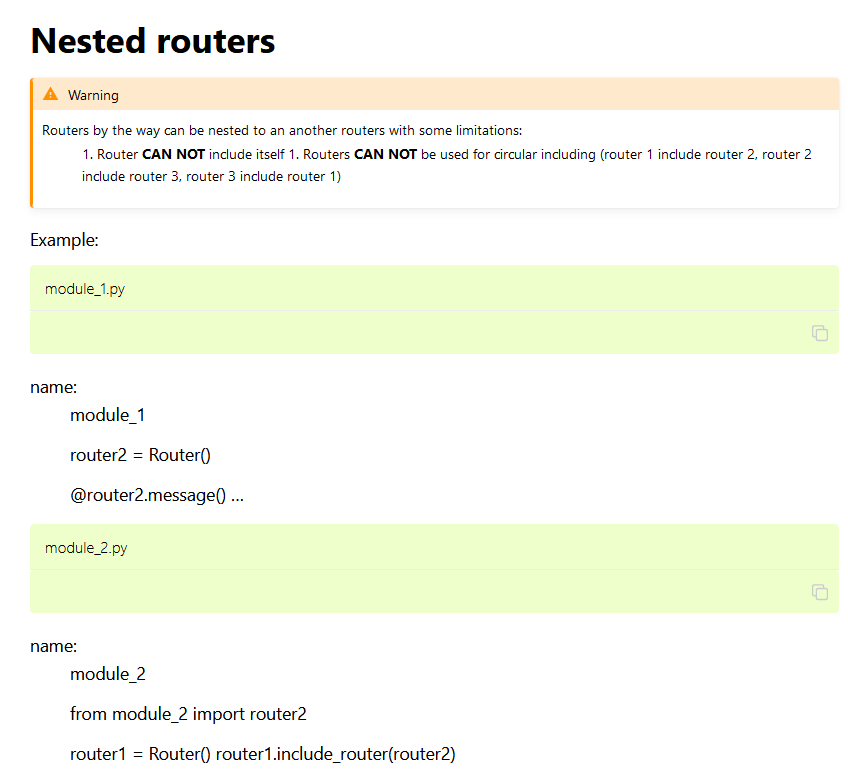
В документации **aiogram 3.4** написано, что теперь **Dispatcher** не может использовать **callback\_query\_handler**



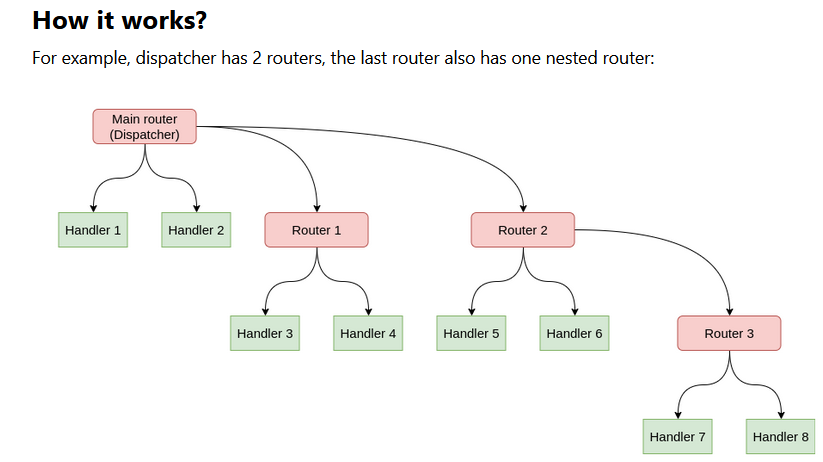
**Ссылка на документацию :** [**https://docs.aiogram.dev/en/dev-3.x/dispatcher/router.html**](https://docs.aiogram.dev/en/dev-3.x/dispatcher/router.html)

Теперь эту работу выполняет **Router**

Роутеров может быть много, и они могут быть разнесены по разным модулям. Что существенно нам поможет вынести все в разные модули и не загромождать главный файл.



Сами роутеры включаются в **root Router** то есть **Dispatcher**

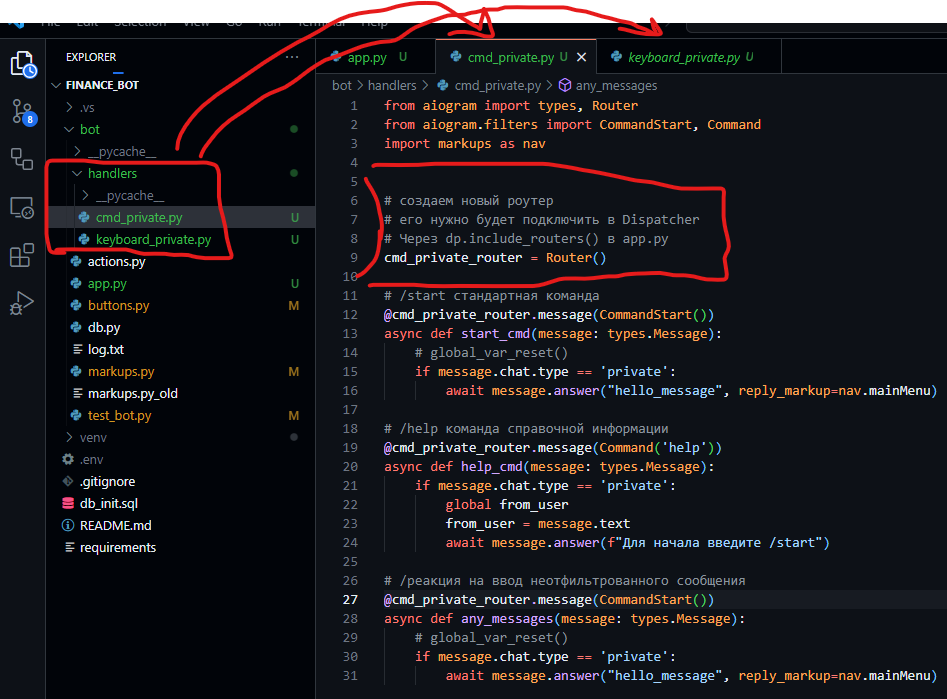


**Пример использования Router включенного в Dispatcher**



Теперь, когда известно, что нужно изменить в коде, переработаем все хендлены, начав с первого раскомментированного. Затем протестируем его работу. Нам потребуется импортировать класс **Route**r и создать его экземпляр. Новый **router** нужно будет включить в **Dispatcher**

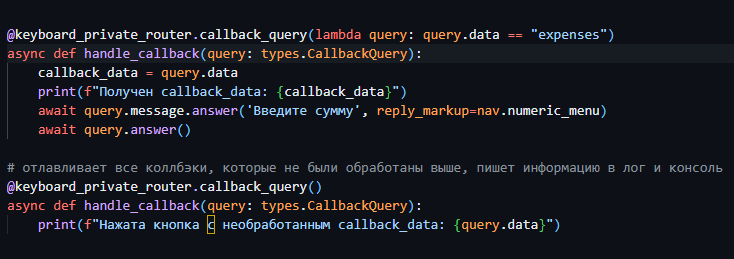
Далее, проведем рефакторинг, требуется вынести все хендлеры в отдельные модули, расположенные в каталоге **handlers.** Предварительно создадим модуль **cmd\_private.py** и **keyboard\_private.py**



**cmd\_private.py будет** использоваться для команд

**keyboard\_private.py** будет использоваться для коллбэков клавиатуры

Вот пример того , как будут выглядеть хендлеры теперь



После всех изменений нужно протестировать работу бота на новом фреймворке и перейти к более тщательному рефакторингу. Также можно частично совместить рефакторинг с миграцией для удобства. В конце всех тестов, мы лолжны получить print() строку сформированную ботом, которая в дальнейшем будет записываться в базу данных.

Далее все изменения будут описаны в коде в ветке **migration**

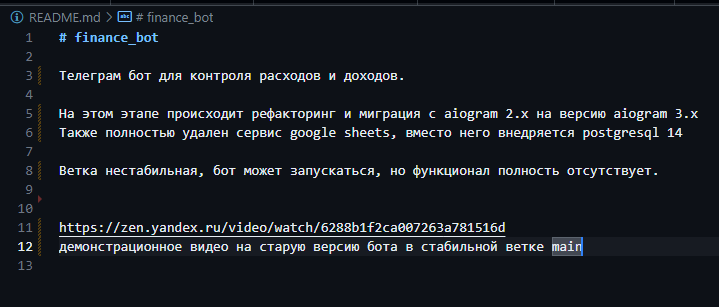
# Миграция на aiogram 3.x

# Git - ветка migration

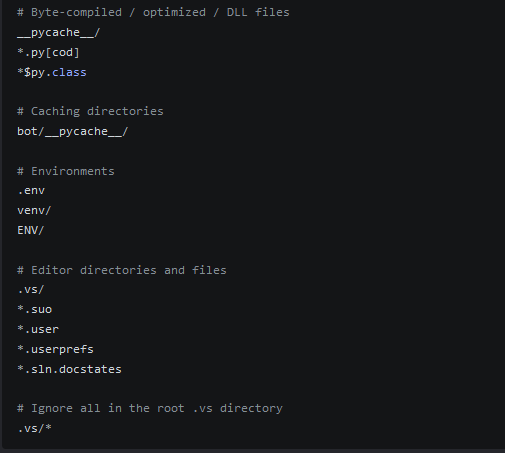
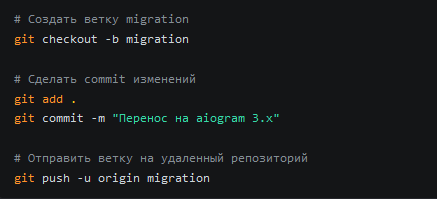
Проделан уже внушительный объем работы, поэтому пора создать ветку migration в git и сделать commit и push для сохранения изменений на текущем этапе

Перед тем как отправить файлы в удаленный репозиторий, следует добавить файл **.gitignore** и внести туда все файлы и папки, которые не следует туда отправлять, например **.env** с ключами доступа, каталоги **bot/\_\_pycache\_\_/** или **.vs/, venv/**

Также следует указать новое описание для ветки



Для **.gitignore** можно использовать следующий шаблон.



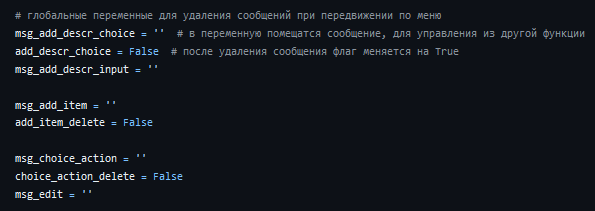
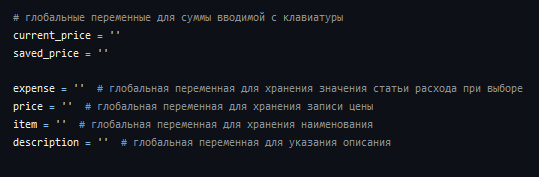
И отправить все на удаленный репозиторий

# Миграция на aiogram 3.x

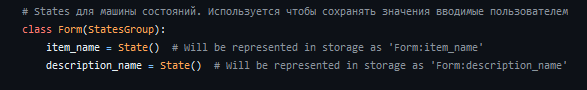
# Машина состояний

Машина состояний также нуждается в переработке. Кроме того, требуется использовать базу данных, для хранения значений некоторых переменных.

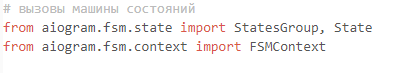
В старой версии был использован подход глобальных переменных, которые содержали в себе промежуточные значения, хранящиеся в оперативной памяти. Их также следует убрать и перевести все на работу через машину состояний.



При этом, код предусматривал работу машины состояний, частично.

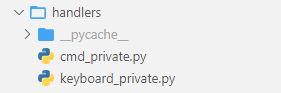


Строки импорта тоже изменились, теперь правильный импорт выглядит следующим образом,



В отличии от aiogram 2.x не требуется передавать тип памяти по умолчанию в диспетчер.

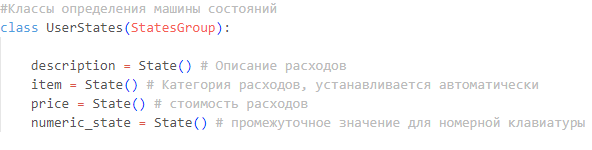
Импорт машины состояний требуется определить в модулях с роутерами



В модуле **cmd\_private.py** потребуется обработать сброс состояний, при вызове команды **/start**

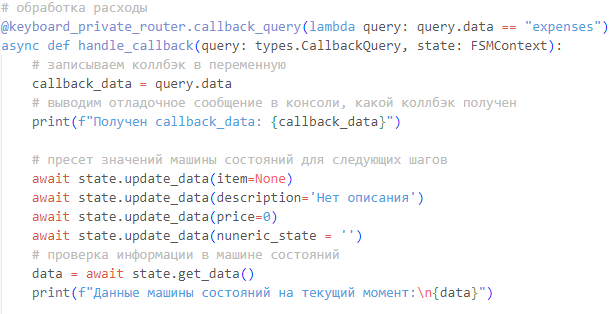


На текущий момент реализована работа машины состояний, которая отслеживает статус работы пользователя и сохраняет значения для дальнейшего использования. Весь функционал полностью асинхронный. Никаких глобальных переменных.

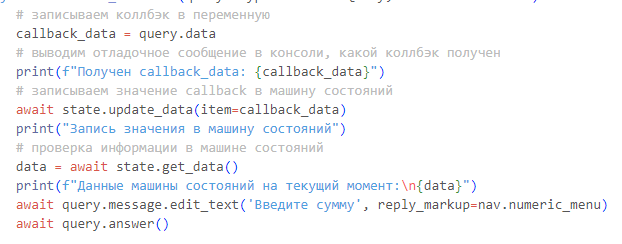


В машине состояний используются переменные для сохранения значений категории – **item**, отлавливаемой через коллбэки при выборе одной из кнопок, описание **description**, которое вводится пользователем с клавиатуры, при этом машина состояний ожидает получение ввода. Значение цены **price**, которое вводится номерной клавиатурой, состоящей из инлайн кнопок, в процессе ее работы строка выше клавиатуры переписывается при каждом нажатии очередной кнопки. Для этого используется **numeric\_state**. Далее, после окончательного ввода **numeric\_state** проверяется на валидность и конвертируется в значение с плавающей точкой и записывается в **price**.

Состояния должны быть сброшены после каждого вызова **/start,** возврата в главное меню, с помощью кнопки **Домой,** или после выполнения записи в базу, так как задача выполнена.



Обработка коллбэков не требует перехода по состояниям, достаточно просто записывать значениие пойманного коллбэка



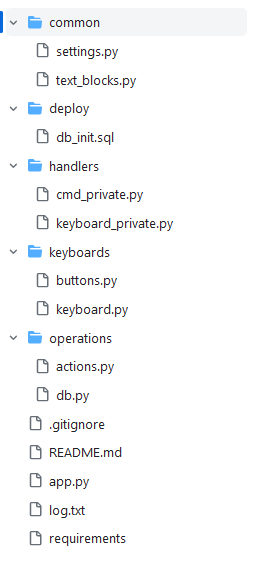
# Миграция на aiogram 3.x

# Заключительный этап. Тестирование

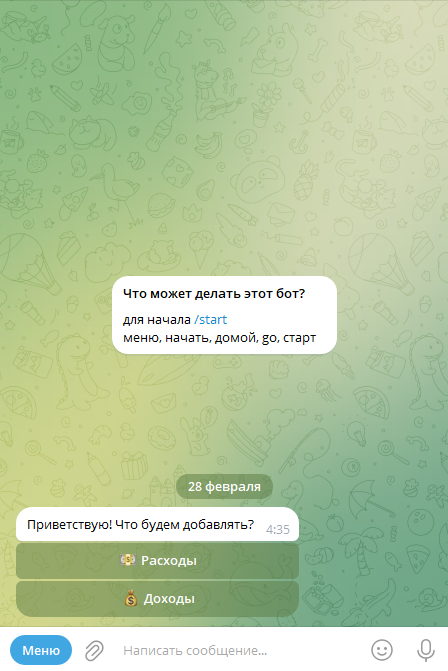
Для формирования запроса существует модуль **actions.py**, который был создан ранее, в нем есть дебаг конструкция, которая выводит полностью сформированную ботом строку в консоль.

Таким образом, можно сказать что весь базовый функционал мигрирован на aiogram 3.x

В итоге структура проекта после проделанной работы выглядит вот так



Приложение формирует вывод строки с наименованием категории, описанием, ценой и отдает строку для следующих манипуляций.



При нажатии на **/start** мы попадаем в главное меню бота, там нам предлагается выбрать дальнейшие действия.

Реализован функционал меню **Расходы**

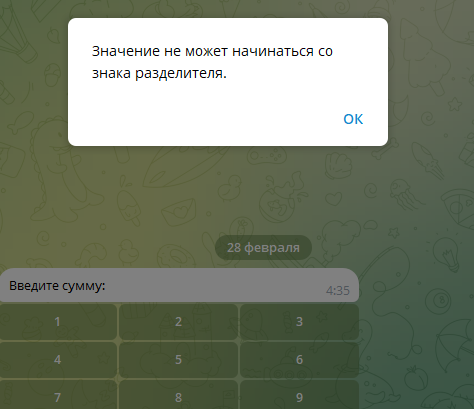
Здесь мы можем выбрать одну из категорий расходов, я делал ее исключительно по своему пожеланию и усмотрению. Возможно в дальнейшем я реализую пользовательскую настройку новых категорий или вообще изменение всего списка для каждого пользователя.



После выбора категории расходов пользователь попадает на клавиатуру ввода цены



Здесь требуется ввести значение цены. Все попытки выполнить неправильный ввод, такой как два нуля в строке, двойной знак разделителя в строке, строка начинается не с цифры, обрабатываются и отлавливаются ботом. При этом всплывает информационное сообщение об ошибке ввода и ввод нужно повторить заново.



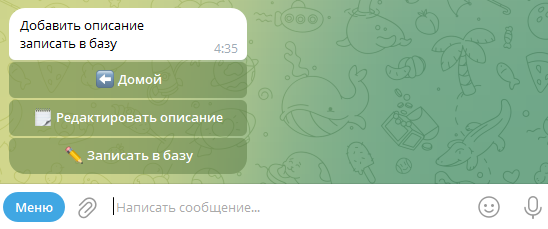
При вводе значения отображаются в верхнем сообщении над клавиатурой:



Кнопка **Заново**, сбрасывает значение и можно начать вводить по новой.

Кнопка **Домой** отправляет в главное меню.

Кнопка **Готово** сохраняет введенное значение в машине состояний и переводит пользователя к следующему шагу.



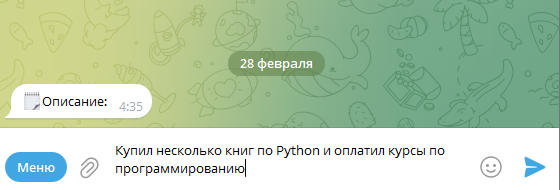
На этом этапе можно добавить описание к текущей позиции, но это необязательный параметр.

Рекомендуется заполнять его, чтобы более гибко отслеживать затраты.

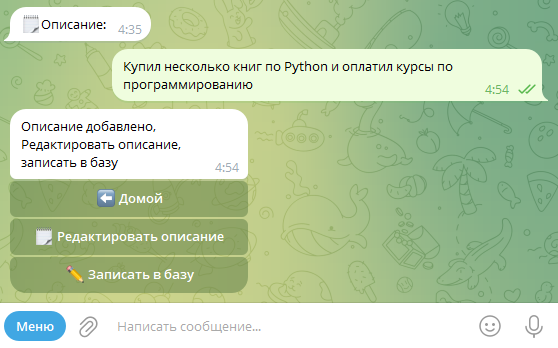
Кнопка **Редактировать описание** переводит пользователя в режим ввода текста, где требуется ввести описание. Машина состояний ожидает получение строкового значения, которое будет сохранено.

Кнопка **Записать в базу**  на данном этапе разработки отправляет дебаг сообщение в консоль со всеми данными от пользователя. В дальнейшем эта строка будет направлена в базу данных

Кнопка **Домой** возвращает пользователя в главное меню. Все состояния сбрасываются.

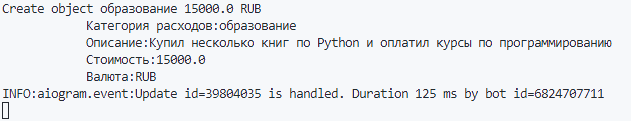


Во время редактирования описания инлайн клавиатура недоступна, и нужно ввести описание со стандартной клавиатуры ввода в текстовое поле. И нажать на отправку.



Теперь можно нажать **Записать в базу,** перед этим можно отредактировать описание еще раз, если нужно.

После записи мы получаем всю информацию в дебаг сообщении



# **Создание тестовой БД и таблицы в СУБД postgresql**

Теперь у нас есть все условия для того чтобы продолжить работу и сформировать шаблон записи в базу данных.Но перед этим, базу необходимо создать.

Требуется развернуть **postgreql 14** в контейнере **Docker** и подключиться к ней по **localhost** предварительно пробросив порты при создании контейнера.

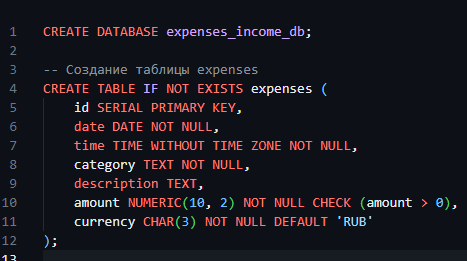
В ней должна создаваться БД **expenses\_income\_db** содержащая в себе таблицы **expenses** и **income.**

**Income** пока использовать не будем, просто чтобы не забыть о ней в будущем при разработке и тестировании функционала учета доходов.

Сценарий с созданием SQL запроса на создание БД и таблицы будет находиться в

**init\_db.sql**

В дальнейшем, таблица может быть переработана и будет состоять из склейки нескольких таблиц. Сейчас это простая талица для отладки приложения.



В дальнейшем, после тестирования, база данных будет переработана, для более гибкого взаимодействия с ней. На данном этапе такая расстановка является приемлемой.

Также дополнительно следует обработать добавление даты и времени при каждой новой записи

