Пояснительная записка

Оглавление

[Введение 2](#_Toc155896423)

[Функциональная диаграмма продукта 3](#_Toc155896424)

[Проектирование 5](#_Toc155896425)

[Архитектурная диаграмма 5](#_Toc155896426)

[Диаграмма последовательности UML 5](#_Toc155896427)

[Концептуальная модель и физическая модель базы данных 7](#_Toc155896428)

[Требования к системному ПО 8](#_Toc155896429)

# Введение

На текущий момент существует множество различных мест, достопримечательностей, которые можно посетить в Санкт-Петербурге. Информацию о таких местах можно найти в сети Интернет с помощью браузера или с помощью онлайн-карт таких, как Яндекс.Карты.

При таком поиске пользователь тратит достаточно продолжительное время на выбор заведения, изучения информации о нем и т.д. Выборка мест и заведений будет происходит за счет наименования, а общая возможная заинтересованность пользователем в месте будет определяться отзывами о нем и его описанием, что также затрачивает время пользователя.

В качестве разрешения данной проблемы может выступать телеграмм бот, который будет выводить пользователю, основываясь на его предпочтениях и рекомендательной системе, структурированную информацию о местах и заведениях для посещения.

Целью работы является создание телеграмм бота, благодаря которому пользователи смогут получить информацию о местах и достопримечательностях города Санкт-Петербург на основе своих предпочтений.

Для выдачи информации пользователю необходимо воспользоваться следующими компонентами:

1. Тематики, предлагаемые телеграмм ботом и выбранные лично пользователем;
2. Рекомендательная система на основе схожих интересов между пользователями.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. спроектировать функциональную диаграмму продукта;
2. спроектировать и обосновать архитектуру продукта;
3. определить требования к системному ПО;

# Функциональная диаграмма продукта

Ниже представлена диаграмма функциональная диаграмма продукта в нотации IDEF0. Как видно из рисунка 1, есть несколько сущностей, влияющих на конечный проект:

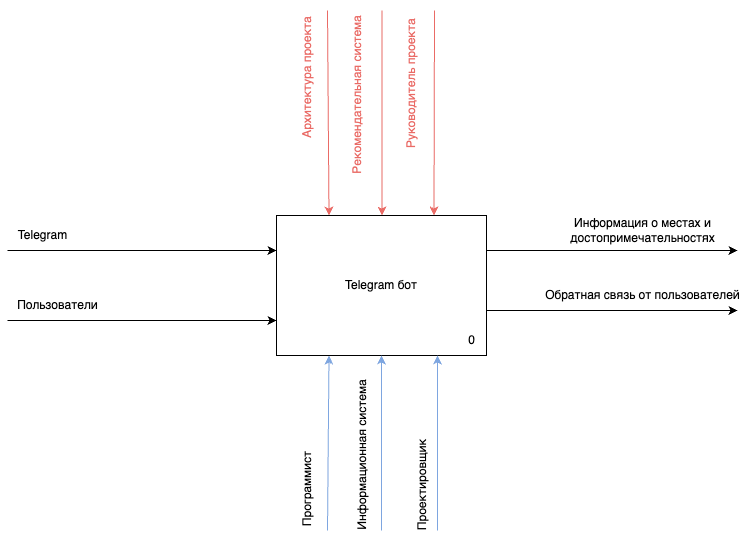
1. Входные данные определяются двумя сущностями: telegram и пользователи. Пользователи могут высылать информацию в telegram бот; telegram высылает различные сигналы в telegram бот.
2. Выходные данные: информация о местах и достопримечательностях, которая высылается пользователям, и обратная связь от пользователях, которая поставляется рекомендательной системе.
3. Контроль продукта составляют архитектура проекта, рекомендательная система и руководитель проекта.
4. ****За механизм продукта отвечают программист, информационная система и проектировщик.

Рисунок 1 – Функциональная диаграмма продукта

Ниже представлена функциональная диаграмма продукта, описывающая бизнес-процессы. Так существуют следующие бизнес-процессы:

1. Выбор пользователем категорий. В данном процессе подразумевается, что пользователь регистрируется в системе, выбирает интересующие его категории мест и достопримечательностей.
2. Процесс обработки рекомендательной системы. Рекомендательная система выбирает наиболее интересную для пользователя информацию, основываясь на собранной ранее информации о пользователях, их оценках о местах и достопримечательностях.
3. Публикация информации для пользователя. Как только рекомендательная система определила информацию о каком месте необходимо публиковать, telegram бот высылает информацию пользователю.
4. Сбор обратной связи. После того, как информация о месте выслана, telegram бот предоставляет возможность пользователю оценить информацию, место или достопримечательность. После информация об этом переходит в рекомендательную систему для использования в будущем.

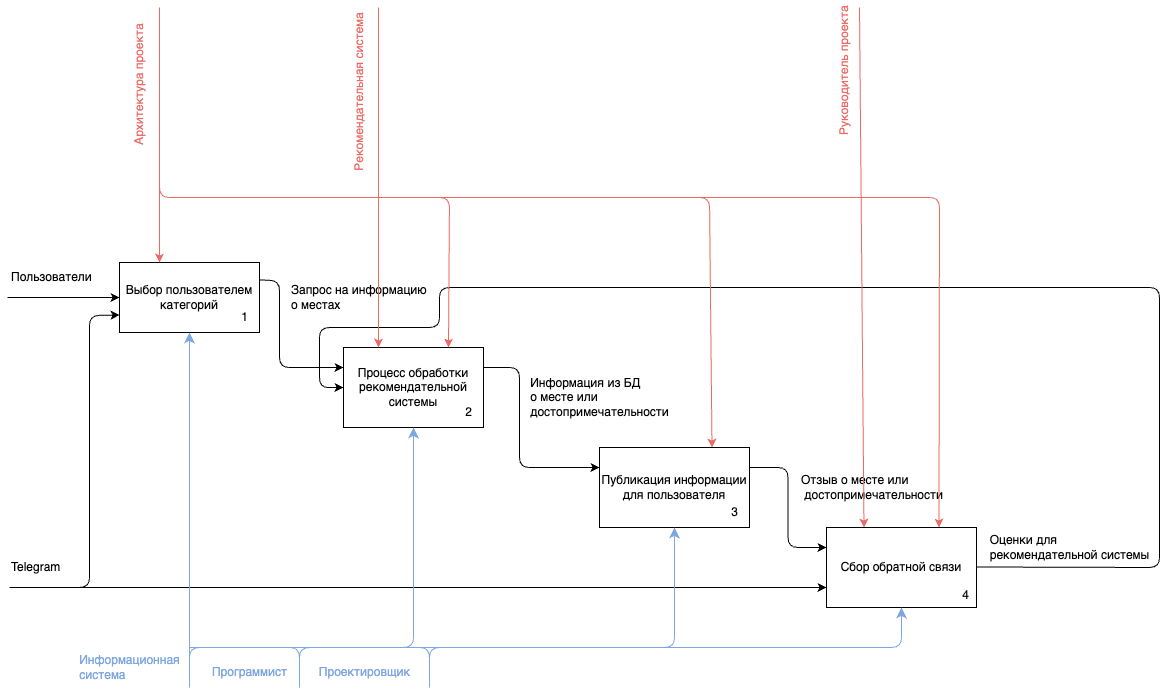
****

Рисунок 2 – Функциональная диаграмма продукта (бизнес-процессы)

# Проектирование

## Архитектурная диаграмма

На рисунке 3 ниже представлена архитектурная диаграмма продукта. На диаграмме присутствуют четыре сущности:

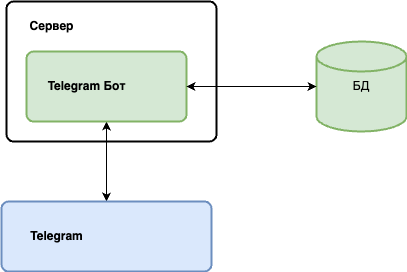
1. сервер, на котором запущен telegram бот;
2. telegram бот – скрипты, которые осуществляют всю необходимую работу для функционирования telegram бота;
3. база данных, которая хранит информацию о пользователях, рекомендательной системы и о местах и достопримечательностях;
4. telegram, к которому высылается вся информация от telegram бота.

Рисунок 3 – Архитектурная диаграмма

В случае реализации MVP данная архитектура является достаточной, если учитывать время и ресурсы, которыми обладает команда для реализации продукта.

В дальнейшем при анализе готового решения текущая архитектура позволяет производить масштабирование системы, а в каких-то системных изменений – команда не понесет значительных временных и иных ресурсов для внесения изменений.

## Диаграмма последовательности UML

Ниже представлена диаграмма последовательностей UML. Диаграмма отражает в другом виде функциональную диаграмму IDEF0. Данная диаграмма указывает дополнительные сведения о том, что в определенные процессы происходит обращение к базе данных (БД) и к рекомендательной системе.

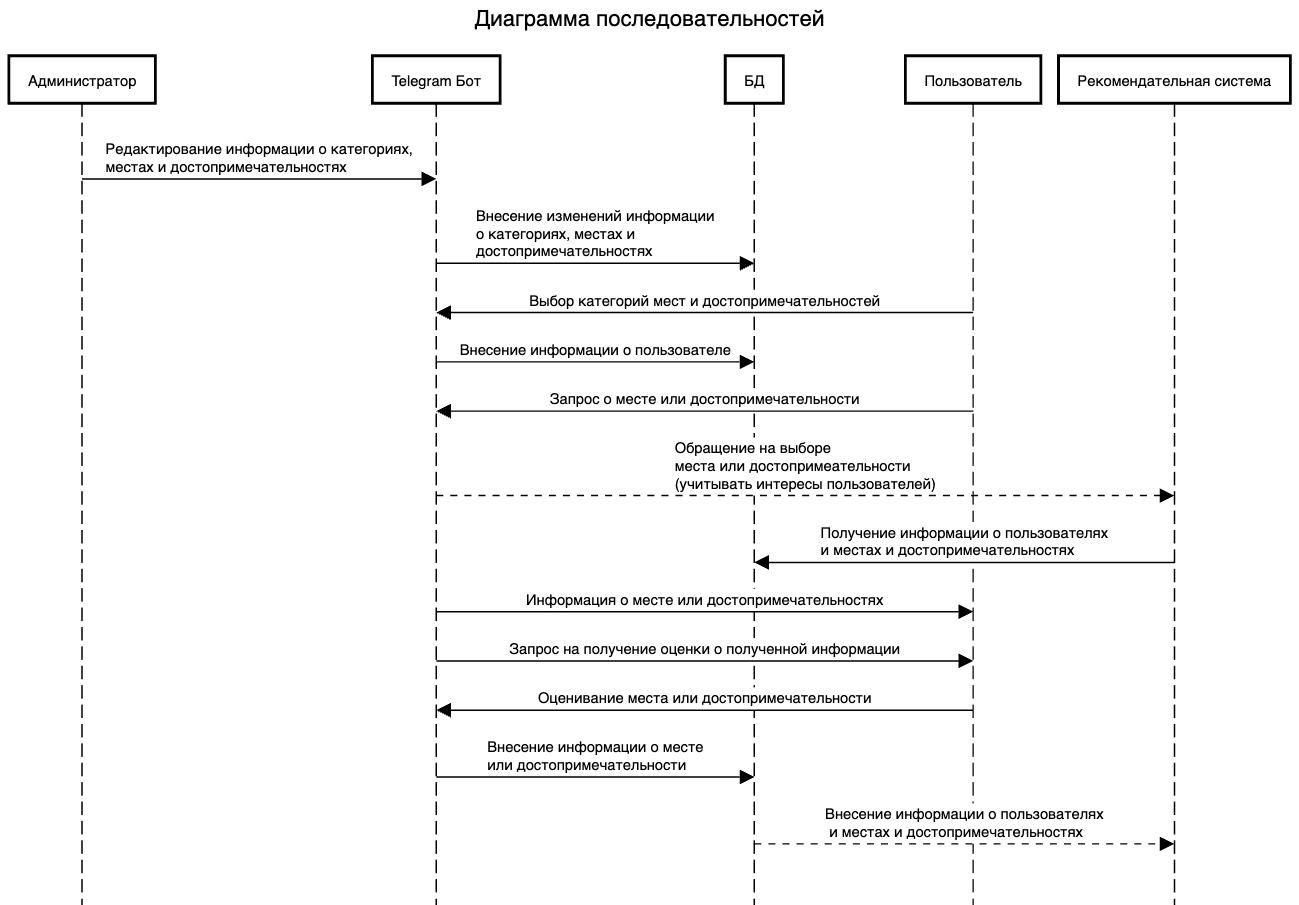


Рисунок – Диаграмма последовательностей UML

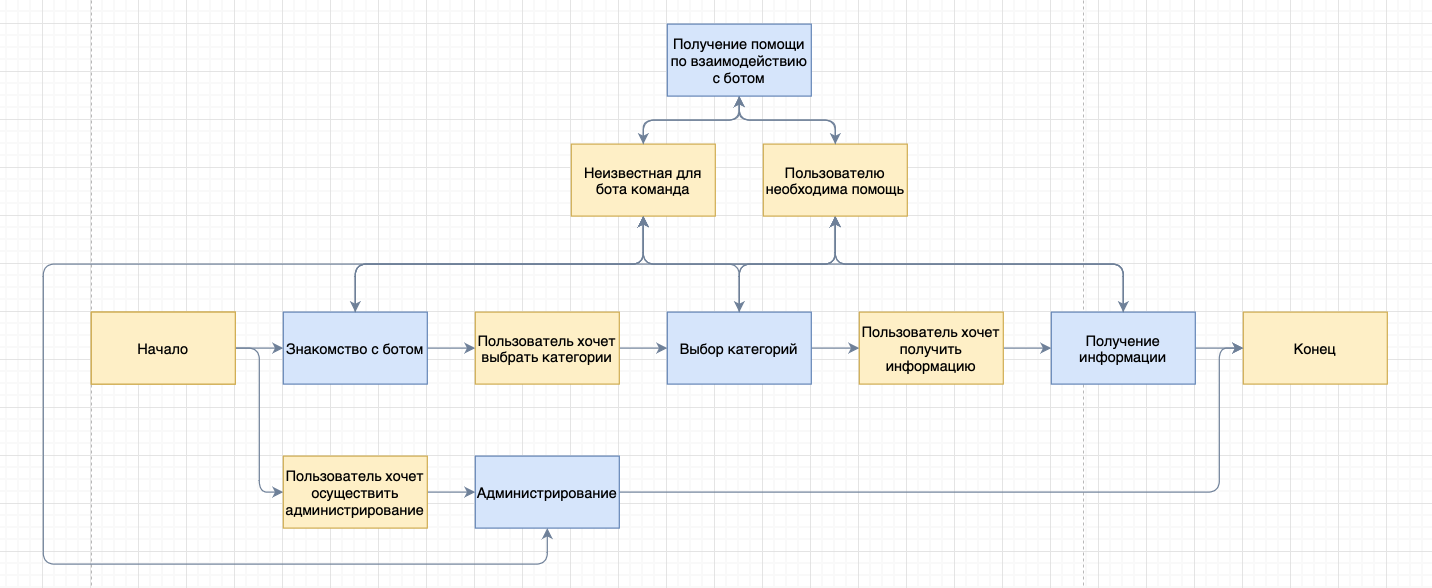
Дополнительно была реализована карта сценариев, которая отражает диалог между пользователем и telegram ботом. Диаграмма представлена рисунком 5 ниже:

Рисунок 5 – Карта сценариев

## Концептуальная модель и физическая модель базы данных

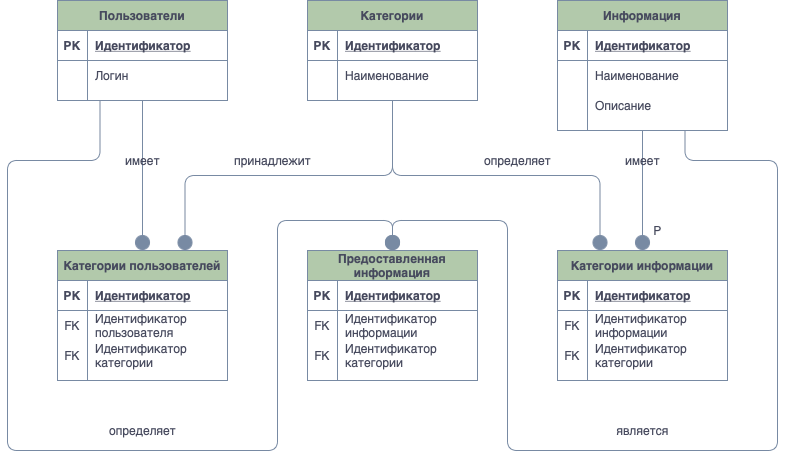
Концептуальная модель и физическая модель базы данных представлена в виде логической модели базы данных. На нижеуказанной модели (рисунок 6) можно увидеть сущности базы данных, которые будут реализованы, их поля, а и также отношения и их мощности.

Рисунок 6 – Логическая модель базы данных

# Требования к системному ПО

Продукт, телеграм-бот, располагается на сервере. Для работы сервера вводятся минимальные требования:

1. Операционная система – Ubuntu 16.04, Debian 10;
2. Процессор – i5-3370;
3. ОЗУ – 4 GB;
4. HHD или SSD объемом 20 GB.

Telegram бот написан с использованием:

1. Python 3.8;
2. Библиотек Python 3.8:
   1. telegram;
   2. sqlite3.

# Приложение А

Код программы

***main.py***

import random

import telebot

from telebot.types import KeyboardButton

from config import \*

from database import Databas

# < Константы >

bot = telebot.TeleBot(TOKEN, parse\_mode='Markdown')

db = Database(DB\_PATH)

interests = db.get\_all\_categories()

interests\_dict = {name.lower(): id for id, name in interests}

start\_commands = ['/start', 'привет', 'кто ты?', 'кто ты', 'старт']

select\_interests\_commands = [

'/add', 'выбрать категории', 'выбрать категории ⚙️', 'категории']

help\_commands = ['/help', 'помощь 💊', 'помоги мне',

'помоги', 'как с тобой работать', 'что мне делать']

get\_info\_commands = ['/info', 'узнать интересную информацию 👀',

'интересная информация', 'интересные факты', 'информация',

'факты', 'получить факты']

reset\_commands = ['/reset', 'сбросить категории', 'сбросить']

admins\_commands = ['Посмотреть список информации', 'Посмотреть категории',

'Показать описание информации',

'Добавить информацию', 'Добавить категорию',

'Удалить информацию', 'Удалить категорию',

'Добавить категорию у информации',

'Сбросить категории у информации', 'Назад']

# < Callback - Функции бота >

@bot.message\_handler(commands=['start'])

def start\_bot(message):

log.info(f'Start chatting with ({message.from\_user.username})')

if not db.is\_user\_in\_db(message.from\_user.username):

keyboard = get\_main\_keyboard()

bot.send\_message(message.chat.id,

f'\*Здравствуйте, {message.from\_user.username}!\* 🙃 '

f'\n\nЯ смотрю, Вы здесь первый раз.\n '>>

f'Давайте я Вам все расскажу! \n\nЯ - бот, который '

f'помогает найти интересную информацию для Вас на '

f'основе ваших категорий. \n\nДля начала мне нужно '

f'узнать категории. Нажимайте кнопочки под чатом, '

f'чтобы выбрать интересующую Вас категорию. \nПосле '

f'выбора всех интересующих Вас категорий нажмите '

f'кнопку "Закончить выбор", чтобы я начал вам '

f'выдавать интересную информацию. \n\nТакже '

f'перечисляю ниже свои команды: \n1. /info - Выдать '

f'интересную информацию \n2. /add - Добавить '

f'интересы\n3. /help - Помощь\n4. /reset - '

f'Перевыбрать интересы\n 5. /admin - '

f'Администрирование',

reply\_markup=keyboard)

db.add\_user(message.from\_user.username)

else:

send\_main\_message(message)

@bot.message\_handler(commands=['help'])

def get\_help\_bot(message):

log.info(f'Send help to ({message.from\_user.username})')

bot.send\_message(message.chat.id,

'Я - бот, который помогает найти интересную информацию '

'для Вас на основе ваших категорий.\nЯ обладаю '

'следующими командами:\n1. /start - Начать диалог\n2. '

'/help - Помощь по боту\n3. /info - Выдать интересную '

'информацию\n4. /reset - Перевыбрать интересы \n5. '

'/admin – Администрирование')

@bot.message\_handler(commands=['reset'])

def reset\_categories\_bot(message):

log.info(f'Reset categories of ({message.from\_user.username})')

db.remove\_all\_categories(message.from\_user.username)

db.remove\_user\_info\_provided(message.from\_user.username)

bot.send\_message(message.chat.id, 'Ваши категории успешно сброшены!')

@bot.message\_handler(commands=['add'])

def select\_categories\_bot(message):

categories = db.get\_no\_user\_categories(message.from\_user.username)

if len(categories) > 0:

keyboard = get\_keybord\_with\_categories(categories)

bot.send\_message(

message.chat.id,

'Выберите соответствущие категории, которые Вам подходят, нажимая '

'на кнопки.',

reply\_markup=keyboard)

else:

send\_all\_categories\_added(message)

@bot.message\_handler(commands=['admin'])

def show\_admin\_panel(message):

keyboard = get\_admin\_keyboard()

bot.send\_message(

message.chat.id,

'Для администрирования выберите одну из функций с помощью кнопок.',

reply\_markup=keyboard)

@bot.message\_handler(commands=['info'])

def get\_info\_bot(message):

log.info(f'Send some info to ({message.from\_user.username})')

info = get\_list\_info(message.from\_user.username)

no\_shown = [i for i in info if not i[2]]

if len(no\_shown):

info\_id = no\_shown[0][0]

else:

info\_id = info[random.randint(0, len(info) - 1)][0]

info = db.get\_info(info\_id)

bot.send\_message(

message.chat.id, f'{info[0]}\n\n{info[1]}')

db.insert\_provided\_info(message.from\_user.username, info\_id)

# < Дополнительные функции для упрощения работы с ботом >

def show\_categories\_admin(message):

categories = db.get\_all\_categories()

msg = 'Список категорий (индентификаторы и названия): \n\n'

for idx, teg in enumerate(categories):

msg += f'{idx+1}. {teg[1]}\n'

bot.send\_message(message.chat.id, msg)

def show\_info\_admin(message):

info = db.get\_all\_info()

msg = 'Список информации (индентификаторы, названия и теги): \n\n'

for doc in info:

tegs = ', '.join(doc[2])

msg += f'{doc[0]}. {doc[1]} '

if len(tegs) > 0:

msg += f'({tegs}) \n'

else:

msg += '\n'

bot.send\_message(message.chat.id, msg)

def show\_desc\_of\_info\_admin(message):

bot.send\_message(

message.chat.id,

'Напишите идентификатор информации, о котором хотите получить '

'описание. \nДля отмены действия напишите команду /break')

bot.send\_message(

message.chat.id,

'Введите название информации, а затем с новой строки его описание.')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_show\_desc\_of\_info\_admin)

def c\_show\_desc\_of\_info\_admin(message):

if message.text not in ['/break', 'Назад']:

if message.text.isdigit():

desc = db.get\_info\_for\_admin(int(message.text))[1]

bot.send\_message(message.chat.id, f'{desc}')

else:

bot.send\_message(message.chat.id, 'Некорректный ввод данных!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def add\_info\_admin(message):

bot.send\_message(message.chat.id,

'Давайте добавим новую информацию! \nОбратите внимание, '

'если название совпадет с названием уже существующей '

'информации, то таким образом вы лишь обновите '

'существующую информацию \nДля отмены действия напишите '

'команду /break')

bot.send\_message(

message.chat.id,

'Введите название информации, а затем с новой строки ее описание.')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_add\_info\_admin)

def c\_add\_info\_admin(message):

if message.text not in ['/break', 'Назад']:

try:

title, desc = message.text.split('\n', maxsplit=1)

db.add\_info(title, desc)

bot.send\_message(message.chat.id, 'Информация успешно добавлена!')

except Exception as e:

bot.send\_message(message.chat.id, 'Некорректный ввод данных!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def add\_category\_user(message):

is\_added = db.add\_user\_category(

message.from\_user.username, interests\_dict.get(message.text.lower()))

if is\_added:

send\_added\_category\_message(message)

else:

send\_no\_added\_category\_message(message.chat.id)

def add\_category\_admin(message):

bot.send\_message(

message.chat.id,

'Давайте добавим новую категорию! \nДля отмены действия напишите '

'команду /break')

bot.send\_message(

message.chat.id, 'Введите название категории.')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_add\_category\_admin)

def c\_add\_category\_admin(message):

if message.text not in ['/break', 'Назад']:

db.add\_category(message.text)

bot.send\_message(

message.chat.id, 'Категория успешно добавлена!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def remove\_info\_admin(message):

bot.send\_message(

message.chat.id,

'Введите уникальный идентификатор информации. \nДля отмены действия '

'напишите команду /break')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_remove\_info\_admin)

def c\_remove\_info\_admin(message):

if message.text not in ['/break', 'Назад']:

if message.text.isdigit():

db.remove\_info(int(message.text))

bot.send\_message(message.chat.id, 'Информация успешно удалена!')

else:

bot.send\_message(message.chat.id, 'Некорректный ввод данных!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def remove\_category\_admin(message):

bot.send\_message(

message.chat.id,

'Введите уникальный идентификатор категории. \nДля отмены действия '

'напишите команду /break')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_remove\_category\_admin)

def c\_remove\_category\_admin(message):

if message.text not in ['/break', 'Назад']:

if message.text.isdigit():

remove\_category\_id = int(message.text) - 1

categories = db.get\_all\_categories()

if 0 <= remove\_category\_id < len(categories):

db\_category\_id = categories[remove\_category\_id][0]

db.remove\_category(db\_category\_id)

bot.send\_message(

message.chat.id, 'Категория успешно удалена!')

else:

bot.send\_message(message.chat.id, 'Произошла ошибка!')

else:

bot.send\_message(message.chat.id, 'Произошла ошибка!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def add\_categories\_to\_info\_admin(message):

bot.send\_message(

message.chat.id,

'Введите через пробел идентификатор информации и идентификатор '

'категории, которую хотите присвоить факту. \nДля отмены действия '

'напишите команду /break')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_add\_categories\_to\_info\_admin)

def c\_add\_categories\_to\_info\_admin(message):

if message.text not in ['/break', 'Назад']:

try:

info\_id, category\_id = message.text.split(' ', 1)

db.add\_category\_to\_info(info\_id, category\_id)

bot.send\_message(

message.chat.id, 'Категория у информации успешно удалена!')

except Exception as e:

bot.send\_message(message.chat.id, 'Произошла ошибка!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!'

def reset\_categories\_of\_info\_admin(message):

bot.send\_message(

message.chat.id,

'Введите идентификатор информации, у которого Вы хотите сбросить '

'категории. \nДля отмены действия напишите команду /break')

bot.register\_next\_step\_handler\_by\_chat\_id(

message.chat.id, c\_reset\_categories\_of\_info\_admin)

def c\_reset\_categories\_of\_info\_admin(message):

if message.text not in ['/break', 'Назад']:

if message.text.isdigit():

db.reset\_categories\_of\_info(int(message.text))

bot.send\_message(

message.chat.id, 'Категория интереса у факта успешно удалена!')

else:

bot.send\_message(message.chat.id, 'Произошла ошибка!')

else:

bot.send\_message(message.chat.id, 'Действие успешно отменено!')

def send\_main\_message(message):

keyboard = get\_main\_keyboard()

bot.send\_message(

message.chat.id,

'Для того, чтобы начать работать, выберите одну из функций с помощью '

'кнопок.',

reply\_markup=keyboard)

def send\_no\_added\_category\_message(message):

bot.send\_message(message.chat.id, f'Вы уже добавили данную категорию!')

def send\_added\_category\_message(message):

keyboard = get\_keybord\_with\_categories(

db.get\_no\_user\_categories(message.from\_user.username))

bot.send\_message(

message.chat.id,

f'Вы успешно выбрали новую категорию – {message.text}!\nВы можете '

f'продолжить выбор категорий, либо перейти в главное меню нажатием '

f'кнопки "Назад".',

reply\_markup=keyboard)

def send\_all\_categories\_added(message):

bot.send\_message(

message.chat.id,

'Вы выбрали все доступные категории!\nСбросьте свой список интересов '

'(/reset), чтобы выбрать их заново.')

def send\_error(message):

bot.send\_message(message.chat.id,

'Я не знаю такую команду! 🙁\nВоспользуйтесь командой '

'/help для того, чтобы узнать мои команды.\n')

def get\_main\_keyboard():

keyboard = telebot.types.ReplyKeyboardMarkup(True)

main\_list = ['Выбрать категории ⚙️', 'Сбросить категории',

'Помощь 💊', 'Узнать интересную информацию 👀',

'Администрирование']

btns = [KeyboardButton(name) for name in main\_list]

keyboard.add(\*btns)

return keyboard

def get\_keybord\_with\_categories(categories):

keyboard = telebot.types.ReplyKeyboardMarkup(True)

btns = [KeyboardButton(item[1]) for item in categories]

btns.append(get\_back\_button())

keyboard.add(\*btns)

return keyboard

def get\_admin\_keyboard():

keyboard = telebot.types.ReplyKeyboardMarkup(True, row\_width=2)

\_btns = [KeyboardButton(command) for command in admins\_commands]

keyboard.add(\*\_btns)

return keyboard

def get\_back\_button():

return KeyboardButton('Назад')

def get\_list\_info(username):

unprovided = db.get\_unprovided\_info(username)

provided = db.get\_provided\_info(username)

result = []

for id in unprovided:

result.append((id, db.get\_similarity\_categories(username, id), False))

for id in provided:

result.append((id, db.get\_similarity\_categories(username, id), True))

result.sort(key=lambda x: x[1], reverse=True)

return result

# < Массив: [команды], функция >

admin\_triggers = [

('Посмотреть список информации', show\_info\_admin),

('Посмотреть категории', show\_categories\_admin),

('Показать описание информации', show\_desc\_of\_info\_admin),

('Добавить информацию', add\_info\_admin),

('Добавить категорию', add\_category\_admin),

('Удалить информацию', remove\_info\_admin),

('Удалить категорию', remove\_category\_admin),

('Добавить категории у информации', add\_categories\_to\_info\_admin),

('Сбросить категории у информации', reset\_categories\_of\_info\_admin),

('Назад', send\_main\_message)

@bot.message\_handler(commands=['admin'])

def admin\_panel\_bot(message):

text = message.text.lower()

for command, func in admin\_triggers:

if text == command.lower():

func(message)

triggers = [

(start\_commands, start\_bot),

(select\_interests\_commands, select\_categories\_bot),

(help\_commands, get\_help\_bot),

(get\_info\_commands, get\_info\_bot),

(reset\_commands, reset\_categories\_bot),

([command.lower() for command in admins\_commands], admin\_panel\_bot),

(['назад'], send\_main\_message),

(['администрирование'], show\_admin\_panel),

]

@bot.message\_handler(content\_types=['text'])

def process\_triggers\_bot(message):

log.info(

f'Got a message from ({message.from\_user.username}) : ({message.text})')

text = message.text.lower()

for commands, func in triggers:

if text in commands:

func(message)

return

if interests\_dict.get(text) is not None:

add\_category\_user(message)

else:

send\_error(message)

bot.polling()

***database.py***

import logging as log

from os import name

import sqlite3

class Database:

def \_\_init\_\_(self, db\_path) -> None:

self.db\_path = db\_path

def is\_user\_in\_db(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT COUNT(\*) FROM users WHERE login=?',

(username,))

data = cur.fetchall()

return data[0][0] != 0

def add\_user(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('INSERT INTO users (login) VALUES (?)', (username,))

con.commit()

log.info(f'Added new user({username})')

def add\_user\_category(self, username, category\_id):

user\_id = self.get\_user\_id(username)

user\_category = self.get\_user\_categories(username)

for \_, id in user\_category:

if category\_id == id:

return False

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'INSERT INTO user\_categories (user\_id, category\_id) VALUES (?, ?)', (str(user\_id), str(category\_id)))

con.commit()

return True

def remove\_all\_categories(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

DELETE FROM user\_categories WHERE user\_id=(

SELECT id from users WHERE login=?)""", (username,))

con.commit()

def remove\_user\_info\_provided(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""DELETE FROM info\_provided WHERE user\_id=(

SELECT id from users WHERE login=?)""", (username,))

con.commit()

def get\_user\_categories(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT id, category\_id FROM user\_categories WHERE user\_id=

(SELECT users.id FROM users WHERE users.login=?)""", (username,))

data = cur.fetchall()

return data

def get\_no\_user\_categories(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT id, name FROM categories WHERE id NOT IN

(SELECT category\_id FROM user\_categories WHERE user\_id=

(SELECT users.id FROM users WHERE users.login=?))""", (username,))

data = cur.fetchall()

return data

def get\_all\_categories(self):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT \* FROM categories')

data = cur.fetchall()

return data

def get\_user\_id(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT id FROM users WHERE login=?', (username,))

data = cur.fetchall()

return data[0][0]

def get\_info(self, info\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'SELECT title, description FROM info WHERE id=?', (info\_id,))

data = cur.fetchall()

return data[0]

def get\_unprovided\_info(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT id

FROM info

WHERE id NOT IN (

SELECT info\_id

FROM info\_provided

WHERE user\_id=(

SELECT id

FROM users

WHERE login=?

)

)

""", (username,))

data = cur.fetchall()

result = [d[0] for d in data]

return result

def get\_provided\_info(self, username):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT id

FROM info

WHERE id IN (

SELECT info\_id

FROM info\_provided

WHERE user\_id=(

SELECT id

FROM users

WHERE login=?

)

)

""", (username,))

data = cur.fetchall()

result = [d[0] for d in data]

return result

def get\_similarity\_categories(self, username, info\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

SELECT COUNT(\*)

FROM info\_categories

WHERE category\_id IN (

SELECT category\_id

FROM user\_categories

WHERE user\_id=(

SELECT id FROM users WHERE login=?

)

) AND info\_id=?

""", (username, info\_id))

data = cur.fetchall()

return data[0][0]

def get\_all\_info(self):

result = []

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT id, title FROM info')

data = cur.fetchall()

for idx, name in data:

cur.execute(

'SELECT name FROM categories WHERE id IN (SELECT category\_id FROM info\_categories WHERE category\_id=?)', (idx,))

tegs = cur.fetchall()

\_tegs = [t[0] for t in tegs]

result.append((idx, name, \_tegs))

return result

def get\_info\_for\_admin(self, info\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT \* FROM info WHERE id=?', (info\_id,))

data = cur.fetchall()

return data[0]

def get\_category\_for\_admin(self, category\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('SELECT \* FROM categories WHERE id=?', (category\_id,))

data = cur.fetchall()

return data[0]

def remove\_info(self, info\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('DELETE FROM info WHERE id=?', (info\_id,))

con.commit()

def remove\_category(self, category\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute('DELETE FROM categories WHERE id=?', (category\_id,))

con.commit()

def add\_info(self, title, desc):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'INSERT OR REPLACE INTO info (title, description) VALUES (?, ?)', (title, desc,))

con.commit()

def add\_category\_to\_info(self, info\_id, category\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'INSERT INTO tegs (info\_id, category\_id) VALUES (?, ?)', (info\_id, category\_id,))

con.commit()

def reset\_categories\_of\_info(self, info\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'DELETE FROM info\_categories WHERE info\_id=?', (info\_id,))

con.commit()

def add\_category(self, name):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute(

'INSERT OR REPLACE INTO categories (name) VALUES (?)', (name,))

con.commit()

def insert\_provided\_info(self, username, info\_id):

with sqlite3.connect(self.db\_path) as con:

cur = con.cursor()

cur.execute("""

INSERT INTO info\_provided (info\_id, user\_id) VALUES

(?, (SELECT id FROM users u WHERE u.login=?))

""", (info\_id, username,))

con.commit()

***config.py***

import logging as log

# Токен для управления ТГ ботом

TOKEN = ''

DB\_PATH = 'data.db'

log.basicConfig(filename='logfile.log',

format='[%(asctime)s] > %(message)s', level=log.INFO)