

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ
ШКОЛА ЭКОНОМИКИ

Высшая Школа Бизнеса
Образовательная программа «Бизнес-Информатика»

Маниович Никита Андреевич

**РАЗРАБОТКА СЕРВИСА ДЛЯ ПОИСКА СОЦИАЛЬНЫХ СВЯЗЕЙ С
ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ БЕССЕРВЕРНЫХ ВЫЧИСЛЕНИЙ**

Курсовая работа студента 2 курса бакалавриата группы ББИ2006

Научный руководитель
Ю. И. Саночкин

Москва 2022

Оглавление

1. ЦЕЛИ И ЗАДАЧИ ПРОЕКТА	3
1.1 ИСПОЛЬЗУЕМЫЕ СЕРВИСЫ И ТЕХНОЛОГИИ	3
1.2 ОПИСАНИЕ ПРОЕКТА	4
2. ГРАФОВАЯ БАЗА ДАННЫХ	6
2.1 ХРАНИМАЯ ИНФОРМАЦИЯ	6
2.2 СЕРВЕР ДЛЯ СУБД	6
3. ИСПОЛЬЗУЕМЫЕ ФУНКЦИИ	8
3.1. ФУНКЦИЯ ДЛЯ ПОИСКА СВЯЗЕЙ	8
3.2. ФУНКЦИЯ ДЛЯ ОБРАБОТКИ СООБЩЕНИЙ ЧАТ-БОТА	19
3.3. ФУНКЦИЯ ДЛЯ ВЗАИМОДЕЙСТВИЙ С БД КЛИЕНТОВ	25
3.4. ФУНКЦИЯ ДЛЯ АВТОРИЗАЦИИ ПОЛЬЗОВАТЕЛЕЙ	26
3.5. ФУНКЦИЯ ДЛЯ ОБРАБОТКИ ПЛАТЕЖЕЙ	28
4. ЗАКЛЮЧЕНИЕ	30
5. СПИСОК ЛИТЕРАТУРЫ	31
6. ПРИЛОЖЕНИЯ	32

1. ЦЕЛИ И ЗАДАЧИ ПРОЕКТА

Целью моего проекта является создание сервиса для поиска социальных связей между пользователями социальной сети Вконтакте. Проект основывается на теории 6 рукопожатий, которая утверждает, что два случайных человека знакомы максимум через 6 рукопожатий. Сервис представляет из себя чат-бота, с помощью которого пользователи могут быстро и удобно найти цепочку знакомств между двумя людьми. Также проект использует технологии бессерверных вычислений, что позволяет сильно сэкономить на требуемой инфраструктуре и ускорить работу сервиса. Для реализации данного проекта необходимо разработать:

1. Графовую базу данных хранящую информацию о пользователях вконтакте и их дружеские связи.
2. Написание скрипта на языке Python, который получает входную информацию от пользователя, производит поиск между указанными людьми с помощью графовой базы данных и возвращает пользователю результат поиска.
3. Создание чат-бота во Вконтакте на языке Python.
4. Настройка и подключение дополнительных и основных, уже используемых, сервисов предоставляемых Yandex Cloud для обеспечения стабильной работы моего сервиса.

Подобный сервис не только интересен в разработке, но и выгоден как бизнес-проект, поэтому было решено настроить прием оплаты.

1.1 ИСПОЛЬЗУЕМЫЕ СЕРВИСЫ И ТЕХНОЛОГИИ

Для графовой базы данных используется СУБД Neo4j 4.4 Community Edition. Использование именно графовой СУБД обусловлено структурой данной, которую необходимо хранить, а именно люди и, главное, связи между ними. В сравнение с привычной СУБД, графовая позволяет сразу хранить сразу всю информацию в нужном виде, не настраивая логику взаимодействия таблиц для создания логики графов.

Все функции размещаются в сервисе Yandex Cloud Functions, которая позволяет использовать бессерверные вычисления, данная технология позволяет экономить на аренде полноценного сервера, достаточно создать функцию внутри сервера, которая будет вызываться каждый раз при обращении к ее публичному адресу HTTP-запросом, тем самым оплата производится только за фактическое время использования ресурсов, что позволяет сильно сэкономить на времени простоя привычного сервера, данная технология активно используются в ИТ-индустрии не только из-за

экономии, но и из-за отсутствия проблем с распределением нагрузки, так, если на обычном сервере приходится продумывать логику работы при пиковых нагрузках, то, используя данную технологию, автоматически выделяются ресурсы для запуска новых функций, при этом не уменьшая производительность остальных. Также используется сервис Yandex Message Queue для взаимодействия разных функций между друг-другом, с помощью нее настраивается 'общение' между функциями, а именно помогает запускать их последовательно. Сервис Yandex Databases, представляет из себя привычную СУБД, используется для хранения данных о пользователях чат-бота, его преимущество в поддержке безсерверные вычислений, а именно, аналогично Yandex Cloud Functions, оплата происходит за фактическое время использования базы данных, в нашем случае расчет идет на количество запросов, что также помогает экономить на времени простоя.

1.2 ОПИСАНИЕ ПРОЕКТА

Создаваемый сервис позволяет людям найти полезную для них информацию, которую вручную найти практически невозможно, потому что необходимо обработать очень большое количество информации. Пользователь взаимодействует с сервисом при помощи чат-бота. Интерфейс чат-бота позволяет клиенту легко авторизоваться внутри системы, изучить инструкцию, пополнить и проверить свой внутренний баланс, сделать запрос на поиск цепочек знакомств между двумя указанными им людьми. Инфраструктура проекта состоит из ряда функций, каждая из них выполняет свою специфическую задачу, например обработка сообщений присланных в чат-бота, запуск скрипта выполняющего непосредственно поиск социальных связей. Также используется сервер на операционной системе Debian, на нем развернута графовая СУБД. Графовая база данных постоянно расширяется, почти при каждой попытке поиска. Если в базе данных нет требуемого пути, то скрипт начинает искать таковой, добавляя друзей указанного пользователя в базу данных пока путь не будет найден или же пройдет максимальное время выполнения функции указанное в скрипте. Также важной для меня задачей стало соответствие всем правилам платформы Вконтакте. Во-первых, по правилам сообщества, есть ограничения по работе с API для одного пользователя, ограничивается количество запросов, поэтому была разработана система авторизации пользователя, благодаря которой я получаю токен VK пользователя и уже от его имени делаю запросы к Вконтакте.

Во-вторых, необходимо соблюдать конфиденциальность пользователей, так если у человека скрыты друзья от определенного пользователя, результатом запроса не должен стать путь содержащий такую связь, для этого была разработана логика проверки 'доступности' данных для пользователя отправляющего запрос. В-третьих, оплата должна приниматься в соответствии с законами РФ, поэтому мной был подключен интернет-эквайринг с онлайн-кассой для соответствия с ФЗ-54.

2. ГРАФОВАЯ БАЗА ДАННЫХ

Для поставленной задачи была выбрана именно графовая база данных поскольку социальные связи представляют из себя фактически граф, где люди это вершины графа, а их связи это ребра. Проект реализуется на основе социальной сети Вконтакте, у пользователей явно выделены их социальные связи списком друзей. Выбранная мною СУБД Neo4j использует язык Cypher для запросов, вместо привычного SQL. Для быстроты работы Neo4j очень важна правильная настройка конфига и индексации базы данных. В конфиге есть важные параметры, такие как Memory Heap Size, отвечающий за выделение физического объема памяти для обработки запросов Java Memory Machine, фактически использование для обработки любых транзакций БД, Memory Page Cache - объем физической памяти для хранения кеша базы данных, позволяет задействовать информацию быстрее, нежели каждый раз искать ее на диске. Индексация также очень важна, так как работа идет с действительно большим количеством данных, исходя из моей предметной области и атрибутов вершин, индексирование было решено производить по id пользователей.

2.1 ХРАНИМАЯ ИНФОРМАЦИЯ

Вконтакте присваивает каждому пользователю уникальный id, по которому можно найти страницу человека. Именно этот id и хранится в базе данных, как primary key у вершин графа, фактически остальная информация о пользователе нам не нужна, ее нет смысла хранить в графой базе данных, так как мы ее можем получить с помощью API запросов ко Вконтакте и при поиске она не используется. Ребра графа не имеют веса, поскольку мы не можем знать насколько близким другом приходится один пользователей другому. Таким образом ребра соединяют людей между собой, но никаких атрибутов, кроме уникального id, который также является primary key и задается уникальным автоматически при создание, не используется.

2.2 СЕРВЕР ДЛЯ БАЗЫ ДАННЫХ

Для обеспечения стабильной работы сервиса нам необходимо разместить графовую базу данных на сервере. Для этого был выбран сервис Yandex Compute Cloud, для легкого взаимодействия с другими сервисами, которые

также размещены в Yandex Cloud. Графовая база данных содержит большой объем данных, так сейчас в ней уже больше 60 000 000 вершин и 400 000 000 ребер, сервер должен быть способен быстро обрабатывать запросы, поэтому сервер должен обладать достаточной мощностью. Neo4j требовательна к CPU, RAM и жесткому диску. Для начала сервер будет иметь следующие характеристики: 32GB RAM, платформа Intel Ice Lake с 4 ядрами, 164 GB SSD. Операционная система - Debian, она идеально подходит для СУБД, потому что не имеет лишних предустановленных сервисов и официально поддерживается Neo4j. 32 GB RAM необходимо из расчетов - для Memory Heap Size используем 5GB в соответствии с рекомендацией Neo4j, Memory Page Cache - Размер базы данных * 1.2, в моем случае это 17ГБ * 1.2 = 20.4ГБ, оставшееся память используется операционной системой. Размер SSD определялся мной исходя из количество IOPS, так yandex предоставляет больше IOPS при большем объеме диска, так 8000 IOPS выше стандартных показателей и доступно от 164 GB SSD, при необходимости диск можно увеличить без сбрасывания сервера.

3. ИСПОЛЬЗУЕМЫЕ ФУНКЦИИ

Все функции размещены на сервисе Yandex Cloud Functions, данный сервис позволяет не арендовать привычный сервер для работы функций, а запускать их только при поступлении запроса на определенный адрес с выделенными мощностями и максимальным временем исполнения. Такие бессерверные технологии все чаще используются во всех сферах IT, поскольку они не сложны в использовании, а также менее затратны, потому что оплата происходит за фактическое время использования, убирая затраты на время простоя сервера.

У таких функций есть особенности при написании кода. Во-первых, необходимо учитывать отведенное время исполнения функции, которое настраивается, но имеет ограничение в 600 секунд. Во-вторых, такие функции должны иметь точку входа, то-есть код начинает исполняться с указанной функции, ее стандартное название - `handle`, такая функция принимает два параметра - `event` и `context`. `Event` содержит данные о параметрах запроса, при вызове функции туда можно передать любую нужную информацию, `context` же содержит информацию и самой функции и ее настройках, там хранится информация о максимальном времени исполнения, оставшемся времени, используемых ресурсах.

По окончании выполнения работы функция возвращает с помощью `return` необходимую информацию.

В большинстве функций используется модуль `logging`, с помощью него составляю логи для легкого определения и исправления ошибок.

3.1 ФУНКЦИЯ ДЛЯ ПОИСКА СВЯЗЕЙ

```
from os import access
from time import process_time, perf_counter
import vk
import vk_api
import requests
import time
import logging as lg
import json
import os
import neo4j as n4
```

Используемые в функции поиска связей модули

Из указанных модулей внешними являются три: `vk`, `vk_api`, `neo4j`.

Модули vk и vk_api помогают легко взаимодействовать с API Вконтакте, так с помощью них можно отправлять сообщения в чат-бот и получать информацию о пользователях, библиотеки помогают формировать запрос автоматически, только подставляя параметры.

Данная функция вызывается из функции обработчика сообщений описанной в пункте 3.2 . Для работы функции нам передается три параметра: ID страницы Вконтакте пользователя вызвавшего функцию, ID первого и второго пользователя, между которыми необходимо найти цепочку знакомств.

```
def handler(event, context):
    try:
        if json.loads(event["body"])[ "type" ] == "confirmation" and json.loads(event["body"])[ 'group_id' ] == 210111570:
            confirmation_code = "41ecaf57"
            return {
                'statusCode': 200,
                'body': confirmation_code
            }
    except:
        pass
    try:
        message = json.loads(event['messages'])[0][ 'details' ][ 'message' ][ 'body' ])
        user_id = message["user_id"]
        entrance_id = message["entrance_id"]
        target_id = message["target_id"]
        logging.info("Input data: " + str(user_id) + " " + str(entrance_id) + " " + str(target_id))
        response = main(user_id, entrance_id, target_id, context)
        params = {
            "r_type": "update_query_status",
            "user_id": user_id,
            "new_status": False
        }
        queryStatus = (requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og",
            params=params))
        if "Поиск удался!" in response:
            echo(user_id, response)
        else:
            return_balance(user_id)
            echo(user_id, response)
        return {
            'statusCode': 200,
            'body': "OK"
        }
    except Exception as e:
        response = str(e)
        echo(335328970, response)

        return {
            'statusCode': 200,
            'body': 'OK'
        }
```

Функция handler в функции поиска связей

Первый блок try используется при подтверждение функции для Вконтакте, чтобы функция могла отправлять сообщения в чат-бот ее нужно подтвердить, так Вконтакте отправляет запрос на нашу функцию и она должна вернуть заранее заданный секретный код (confirmation_code в коде). С помощью json.loads берем из event нужные данные, а именно тип запроса и ID группы, сверяем для безопасности, иначе запрос могут подделать

злоумышленники. Если запрос не от ВК, то в блоке `except` мы просто продолжаем работу программы.

Во втором блоке `try` мы сначала получаем аналогично содержание `event`, записывая в соответствующие переменные необходимые входные данные.

Далее вызываем функцию `main`, в которую передаем входные данные и `context`, записывая возвращаемый ответ функции в переменную `response`.

После выполнения `main` меняем статус запроса в базе данных отправляя запрос на функцию описанную в 3.3 . В блоках `if-else` мы отправляем ответ пользователю, определяя удался ли запрос или нет. В случае ошибки срабатывает блок `except`, который отправляет информацию об ошибке мне.

Функция `echo` (приложение 1) предназначена для отправки сообщений чат-ботом, так она получает ID пользователя и текст, который необходимо отправить. Сначала создается сессия с помощью `vk_api` , а потом вызывается метод для отправки сообщения.

Функция `errors_catch` (приложение 3) используется для определения типа ошибок, их корректного логирования и генерации понятного для пользователя текста. Ошибки обрабатываются в соответствии с документацией ВК и их кодами ошибок.

Функции `return_balance` и `get_user_token` (приложение 4) помогают взаимодействовать с базой данных пользователей и получать их баланс и токен соответственно, отправляя запрос на другую функцию описанную в 3.3 .

```
def get_user_id(user_name):
    flist = vk_session.method("users.get", {"user_ids": user_name})
    return flist[0]["id"]

def get_user_name(user_id):
    flist = vk_session.method("users.get", {"user_ids": user_id})
    full_name = flist[0]["first_name"] + ' ' + flist[0]["last_name"]
    return full_name

def get_friend(user_id):
    flist = vk_session.method(
        "friends.get", {"user_id": user_id, "order": 'hints', "count": 10000})
    return flist["items"]
```

Функции `get_user_id`, `get_user_name`, `get_friend` в функции поиска связей

Вышепредставленные функции взаимодействует с API Вконтакте посредством методов модуля `vk_api`. С помощью `get_user_id` мы получаем ID пользователя по самостоятельно заданному идентификатору пользователя,

get_user_name производит обратную операцию, get_freind возвращает список ID друзей пользователя.

```
def add_lots_nodes(tx, root_friend, nodes):  
  
    query1 = f'MERGE (a:Person {{ id: "{root_friend}" }})'  
    query = ""  
    MATCH (b:Person {{id: $root_friend}})  
    UNWIND $nodes as row  
    MERGE (a:Person {{id: row.id}})  
    MERGE (a)-[:FRIENDS]-(b)  
    """.format(nodes=nodes, root_friend=root_friend)  
    tx.run(query1)  
    tx.run(query, nodes=nodes, root_friend=root_friend)
```

Функция add_lots_nodes в функции поиска связей

Функция add_lots_nodes используется для добавления списка пользователей в граф, она получает параметр tx, который содержит информацию о сессии подключения к графовой базе данных, root_friend это ID пользователя, чьих друзей мы добавляем, и nodes - список ID друзей root_friend пользователя. Составляем два Cypher запроса. Первый добавляет вершину root_friend в базу данных, а второй добавляет поочередно всех людей из списка nodes и добавляет ребра между root_friend и добавленными людьми.

Функция graph_maker (приложение 4) используется для дальнейшего вызова add_lots_nodes, сначала из списка ID пользователей с помощью convert (приложение 4) делаем список словарей, в таком формате нужно передавать массив данных в cypher запрос, далее открываем сессию подключения к графовой базе данных и вызываем add_lots_nodes для добавления в нее новой информации.

serach_for_friend (приложение 5) проверяет есть ли в списке друзей пользователя с ID передаваемом в переменной second_user_id пользователей с ID first_user_Id.

```

def shortest_path_query(tx, first_id, second_id):
    res = tx.run(
        f'MATCH p = ShortestPath((:Person {{ id: "{first_id}" }})-[*..(max_path_len-1)]-(:Person {{ id: "{second_id}" }})) RETURN p')
    rec = res.single()
    if rec == None:
        raise ValueError("No Path")
    nodes = rec[0].nodes
    list_to_return = []
    for node in nodes:
        list_to_return.append(node["id"])
    return list_to_return

def delete_relation_query(tx, first_id, second_id):
    tx.run(f'MATCH (:Person {{ id: "{first_id}" }})-[a:FRIENDS]-(:Person {{ id: "{second_id}" }}) DELETE a')

def graph_path(first_id, second_id):

    with driver.session() as session:
        path_test = session.write_transaction(shortest_path_query, first_id, second_id)

    for i in range(len(path_test) - 1):
        if not search_for_friend(int(path_test[i]), int(path_test[i + 1])):
            with driver.session() as session:
                session.write_transaction(delete_relation_query, path_test[i], path_test[i + 1])
            #ComG.remove_edge(str(path_test[i]), str(path_test[i + 1]))
            with driver.session() as session:
                path_test = session.write_transaction(all_paths_query, first_id, second_id)
            try:
                path_test = path_test[0]
                i = len(path_test)
                break
            except:
                raise ValueError("Not actual info and no other ways")

    path = path_test
    if len(path) <= max_path_len:
        return path
    else:
        raise ValueError("Longer than needed")

```

Функции `shortest_path_query`, `delete_realtion_query`, `graph_path` в функции поиска связей

Данные функции являются основными для поиска цепочек связей, составляют Cypher запросы в графовую базу данных. Так `shortest_path_query` возвращает кратчайший путь между двумя пользователями, `delete_relation_query` удаляет ребро между вершинами, `graph_path` сначала делает запрос на получение всех путей между двумя пользователями, а далее проверяет актуальность найденных путей, а именно проверяет, что пользователи во всех случаях еще находятся в друзьях у друг-друга, а также проверяет, что их страницы открыты для пользователя отправившего запрос. В случае если таких путей нет - возвращает ошибку, которая обрабатывается try catch блоком.

Функция (приложение 6) `all_paths_query` создает Cypher запрос для поиска всех возможных путей между пользователями максимальной указанной длины и преобразует ответ в лист ID пользователей, она используется если уже найден ранее один путь, для проверки наличия большего количества перед возвратом ответа.

Функция (приложение 7) `add_second_user_lvl2` добавляет друзей второго уровня пользователя в базу данных.

```

def main(user_id, entrance_id, target_id,
         context):
    global graph_file_pth
    try:

        logging.info("Session has started")

        global path, friends_max, all_friends, found, ifstop, max_path_len, vk_session, first_id, second_id, second_friends

        path = []
        friends_max = 10000
        all_friends = []
        found = False
        ifstop = False
        max_path_len = 6

        access_token = get_user_token(user_id=user_id)
        #access_token = "0f36f60a6628d9d00243166fa8f6820cb34a54554be81a1485e56a26c90f702f7d7257d9a2e186f9db27f"

        # токен принимается из бд
        vk_session = vk_api.VkApi(
            token=access_token)
        vk_session.get_api()

        logging.info("access token has been obtained -- " + access_token)
        time.sleep(2)
        echo(user_id,
            "✅Ваш запрос принят!\n🔍 Запрос обрабатывается до 15 минут, пожалуйста, дождитесь ответа перед тем как отправлять новый!")

        logging.info("Подключение к Neo4j")

        global driver

        driver = n4.GraphDatabase.driver("bolt://46.39.255.77:7766", auth=('neo4j', 'master'))

        first_guy = entrance_id
        second_guy = target_id

        logging.info("Getting ids of selected users!")

```

Функция main в функции поиска связей. Часть 1

Функция main - основная функция, аналог блока кода в обычном python-скрипте, где вход начинается с первых строк, после блока модулей. Для начала мы задаем необходимые переменные, создаем сессию подключения VK, получаем token пользователя и задаем параметры подключения к Neo4j базе данных.

```

first_guy = entrance_id
second_guy = target_id

logging.info("Getting ids of selected users!")

try:
    first_id = get_user_id(first_guy)
    second_id = get_user_id(second_guy)
except Exception as e:
    if 'list' in str(e):
        return "❗ Не удается получить информацию об одном из указанных пользователей! Проверьте корректность запроса!"
    else:
        raise e

logging.info("Getting friends of both ids")

first_friends = get_friend(first_id)
first_friends = first_friends[:friends_max]
second_friends = get_friend(second_id)
second_friends = second_friends[:friends_max]

logging.info("Got it")

if (second_friends == []) or (first_friends == []):
    return "У одного из указанных людей невозможно получить список друзей, возможно скрыт профиль или список друзей"

graph_maker(first_friends, first_id)
graph_maker(second_friends, second_id)

logging.info("Added to graph")

search_res = search_f(first_friends,
                      context
                      )

```

Функция main в функции поиска связей. Часть 2

Пытаемся получить ID и список друзей первого и второго пользователя, в случае ошибки возвращаем пользователю информацию, о том, что у указанных пользователей невозможно получить список друзей или их ID. Далее добавляем их и их друзей в базу данных и запускаем функцию поиска цепочки знакомств `search_f`, записывая ее результат в переменную `search_res`.

```

if (search_res == True) or ((len(path) > 0) and (len(path) <= max_path_len)):
    logging.info("I know the WAY!!!!!!")

    res = "🔍 Поиск удался! ✅\n"

    with driver.session() as session:
        paths = session.write_transaction(all_paths_query, first_id, second_id)

    count = 1

    if len(paths) == 0:
        return ""

    if len(paths) == 1:
        for user in paths[0]:
            name = get_user_name(user)
            res = res + f'[id{user}|{name}]' + '\n'
        res = res + "\n"

    else:
        for path_now in paths:
            res = res + f"Вариант №{count}: \n"
            count = count + 1
            for user in path_now:
                name = get_user_name(user)
                res = res + f'[id{user}|{name}]' + '\n'
            res = res + "\n"

return res

```

Функция main в функции поиска связей. Часть 3

Проверяем удалось ли найти путь в результате работы функции search_f, если да, то формируем ответ пользователю, в котором будет содержаться все найденные пути.

```

else:
    logging.info("Nothing has been found!")

    if search_res == False:
        return "😞 Путь не найден, попробуйте других людей"
    elif search_res == 'day limit':
        return "⚠️ Вы достигли лимита запросов, это ограничение от VK, к сожалению, избежать его нельзя. Обычно лимит сбрасывается через час - день."
    elif search_res == 'token error':
        return "⚠️ Возникла ошибка авторизации, пожалуйста, попробуйте авторизоваться заново!"
    else:
        return "⚠️ Возникла неизвестная ошибка, если эта ошибка повторится у Вас, пожалуйста, напишите администратору группы."

except Exception as e:
    return errors_catch(e)

```

Функция main в функции поиска связей. Часть 4

Если путь не найден, то мы определяем причину этого и возвращаем соответствующую информацию. Если в ходе main возникла ошибка то вызываем функцию errors_catch для ее обработки.

```
def search_f(friends_list,
            context
            ):

    global path
    lvl_friends = []
    count = 0
    alreadyTriedToFindPath = False

    for i in range(10):
        try:

            # TimeoutHandler
            if float(context.get_remaining_time_in_millis() / 1000) < 120:
                return False

            graph_path(first_id, second_id)
            return True
        except Exception as e:

            alreadyTriedToFindPath = True

            logging.warn(str(e) + " -- Error while finding path in the main cycle")

            if (count == 0) and (i == 0):
                print("добавляю друзей второго уровня второго пользователя")
                add_second_user_lvl2(second_friends[:100])
                print("Готово")
```

Функция search_f в функции поиска связей. Часть 1

Функция search_f производит поиск пути. В блоке TimeoutHandler проверяем сколько осталось из выделенного времени на выполнение функции с помощью параметра context. Далее в цикле мы периодически пытаемся найти путь с помощью функции graph_path, также при первом запуске мы вызываем функцию add_second_user_lvl2, чтобы добавить у второго пользователя друзей второго уровня, зачастую это очень сильно ускоряет процесс поиска пути.


```

for user in friends_list:
    if count >= friends_max:
        break

    if ((count % 90) == 0) and (alreadyTriedToFindPath == False):
        try:
            graph_path(first_id, second_id)
            return True
        except Exception as e1:
            logging.warn(str(e1) + " -- Error while finding path in the submain cycle")

    alreadyTriedToFindPath = False

try:
    # TimeoutHandler
    if float(context.get_remaining_time_in_millis() / 1000) < 120:
        return False

    new_list = get_friend(user)

    if (second_id in new_list):
        graph_maker(new_list, user)
        graph_path(first_id, second_id)
        return True

    if any(item in new_list for item in second_friends):
        new_list = list(
            set(new_list).intersection(second_friends))
        graph_maker(user, new_list[0])
        graph_maker(new_list[0], second_id)
        #ComG.add_edge(str(user), str(new_list[0]))
        #ComG.add_edge(str(new_list[0]), str(second_id))
        graph_path(first_id, second_id)
        return True

    new_list = new_list[:friends_max]
    graph_maker(new_list, user)
    count = count + 1
    lvl_friends.extend(new_list)

```

Функция search_f в функции поиска связей. Часть 2

Далее мы производим аналог поиска в глубину, вы добавляем друзей-друзей первого пользователя и так далее пока не найдем связь между первым и конечным пользователями.

```

except vk_api.AuthError as e2:
    logging.warning(str(e2) + "AuthError")
    continue
except vk_api.ApiError as e2:
    if ('18' in str(e2)) or ('30' in str(e2)):
        continue
    elif '29' in str(e2):

        logging.error(str(e2) + "DAY LIMIT REACHED !!! TRY TOMORROW! in fun")

        return 'day limit'
        continue
    elif ('5' in str(e2)) and ('15' not in str(e2)):

        logging.error(str(e2) + "ACCESS TOKEN DENIED OR EXPIRED! in fun")

        return 'token error'
        continue

    else:
        logging.error(str(e2) + "UNKNOWN ApiError")
        continue
except vk_api.VkApiError as e2:
    logging.warn(str(e2) + "VkApiError")
    continue
except vk_api.ApiHttpError as e2:
    logging.warn(str(e2) + "ApiHttp")
    continue
except Exception as e2:
    logging.warn(str(e2) + "OTHER ERROR")
    continue

friends_list = lvl_friends
lvl_friends = []
count = 0

```

Функция search_f в функции поиска связей. Часть 3

В данном блоке мы обрабатываем возможные ошибки, а в случае их отсутствия переходим на следующий уровень друзей по удаленности от первого пользователя.

Настраиваем в сервисе Yandex Cloud Functions данную функцию (приложение 8). Так как поиск может длиться долго, выставляем максимальный таймаут в 600 секунд, функция во время работы оперирует весьма большим количеством данных и поэтому RAM нам потребуется около 256, также задаем переменные окружения, в данном случае это токен для отправки сообщений от лица чат-бота.

3.2 ФУНКЦИЯ ДЛЯ ОБРАБОТКИ СООБЩЕНИЙ ЧАТ-БОТА

Данная функция предназначена для обработки поступающих сообщений в чат-бот во Вконтакте и ответа на них.

Из дополнительных модулей данная функция использует `vk`, описанную ранее, `requests`, для отправки запросов и `boto3` для удобных запросов к сервису Yandex Message Queue (приложение 9).

```
def handler(event, context):
    try:
        if json.loads(event["body"])["type"] == "auth":
            session = vk.Session()
            api = vk.API(session, v=5.131)
            api.messages.send(access_token=str(os.getenv('token')), user_id=json.loads(event["body"])["user_id"],
                              message="✅ Успешная авторизация!", random_id=int(
                                  time.time()) * 10000000)
            return {
                'statusCode': 200,
                'body': "OK"
            }

    except Exception as e:
        pass

    try:
        if json.loads(event["body"])["type"] == "confirmation" and json.loads(event["body"])["group_id"] == 210111570:
            data = "b98cc701"
            return {
                'statusCode': 200,
                'body': data
            }

        if json.loads(event["body"])["secret"] == "s1979zfs21ds53dF93sdLM":
            vk_callback(event["body"], os.getenv('token'))

        return {
            'statusCode': 200,
            'body': "OK"
        }

    except Exception as e:
        rep = str(e)
        return {
            'statusCode': 200,
            'body': str(e)
        }
```

Функция handler функции обработчика чат-бота.

Первый блок `try` обрабатывает запросы от функции описанной в 3.4. Аналогично функции поиска связей, второй `try` блок ответственный для авторизации функции во Вконтакте и также вызова обработчика сообщений `vk_callback` функции.

Функция `links_to_id` преобразует переданное сообщение с двумя пользователями, между которыми нужно найти цепочку связи, в ID их страниц, а также проверяет, что передано ровно два человека.

vk callback - основная функция обрабатывающая сообщения пользователей присланные в чат-бот. Сначала записываю в переменные тексты для приветствия и инструкции (приложение 10).

Задаем в виде словарей клавиатуры в формате соответствующему требованиям VK (приложение 11). keyboard_new - основная клавиатура, которая показывается пользователю вместо клавиатуры в чат-боте. keyboard_auth - inline клавиатура вызываемая при прохождении авторизации, прикрепляется к сообщению от бота. instuctionKeyboardButton - команда для вывода кнопки с инструкцией, прикрепляется к сообщения бота.

```
session = vk.Session()
api = vk.API(session, v=5.131)
event = json.loads(lambda_event)

if "type" not in event:
    return
if event['type'] == 'message_new':
    msg_text = event['object']['message']['text']
    user_id = event['object']['message']['from_id']
    if (msg_text == 'Начать') or (msg_text == 'начать'):
        api.messages.send(access_token=token, user_id=str(user_id), message=start_text, random_id=int(
            time.time()) * 10000000, keyboard=json.dumps(keyboard_new))
    elif (msg_text == 'Баланс') or (msg_text == 'баланс'):
        params = {
            "r_type": "get_user_balance",
            "user_id": user_id
        }
        balance = requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og",
                               params=params)
        if (json.loads(balance.text) == "User not found"):
            api.messages.send(access_token=token, user_id=str(user_id),
                              message="🔒 Сначала вам нужно авторизоваться!", random_id=int(
                                  time.time()) * 10000000, keyboard=json.dumps(keyboard_auth))
            return

        api.messages.send(access_token=token, user_id=str(user_id),
                          message="Ваш баланс: " + balance.text + " запросов. 🔍", random_id=int(
                              time.time()) * 10000000, keyboard=json.dumps(keyboard_new))
    elif (msg_text == 'Пополнить баланс') or (msg_text == 'пополнить баланс'):
        params = {
            "r_type": "get_user_balance",
            "user_id": user_id
        }
        balance = requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og",
                               params=params)
        if (json.loads(balance.text) == "User not found"):
            api.messages.send(access_token=token, user_id=str(user_id),
                              message="🔒 Сначала вам нужно авторизоваться!", random_id=int(
                                  time.time()) * 10000000, keyboard=json.dumps(keyboard_auth))
            return
        api.messages.send(access_token=token, user_id=str(user_id),
                          message="Выберите сколько запросов вы хотите купить:", random_id=int(
                              time.time()) * 10000000, keyboard=json.dumps(keyboard_new_pay))
```

Функция vk_callback функции обработчика чат-бота. Часть 3.

Создаем сессию подключения к vk. С помощью блока if определяем тип присланного сообщения.

Если 'начать' - отправляем приветственный текст, если 'баланс' - получаем баланс с помощью функции описанной в 3.3 получаем баланс и присылаем соответствующее сообщение пользователю, если 'пополнить баланс' - проверяем авторизован ли пользователь, если нет - предлагаем авторизоваться, если да - присылаем клавиатуру с интерактивными кнопками-ссылками для перехода на страницу оплаты.

Команда `god_give_me_money` (приложение 12) - администраторская команда, которая позволяет пополнить баланс без оплаты, мы получаем сначала актуальный баланс , добавляем к нему 10 и записываем новый баланс в базу данных.

По сообщению 'инструкция' пользователю присылается сообщение с текстом инструкции по использованию бота.

На сообщение 'авторизация' присылаем соответствующую клавиатуру, содержащую кнопку-ссылку , она перенаправляет на страницу авторизации.

На сообщение 'Поиск' присылаем краткую инструкцию и кнопку для просмотра полной.

```

elif (msg_text[:5] == 'Поиск') or (msg_text[:5] == 'поиск'):
    params = {
        "r_type": "get_query_status",
        "user_id": user_id
    }
    queryStatus = (requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og",
        params=params))

    if queryStatus.text == 'True':
        api.messages.send(access_token=token, user_id=str(user_id),
            message="🔍 У вас уже запущен запрос, подождите его результата и попробуйте снова!", random_id=int(
                time.time()) * 10000000)
        return

    params = {
        "r_type": "get_user_balance",
        "user_id": user_id
    }
    balance = (requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og",
        params=params))
    if (json.loads(balance.text) == "User not found"):
        api.messages.send(access_token=token, user_id=str(user_id),
            message="🔑 Сначала вам нужно авторизоваться!", random_id=int(
                time.time()) * 10000000, keyboard=json.dumps(keyboard_auth))
        return
    balance = int(balance.text)
    if (balance < 1):
        api.messages.send(access_token=token, user_id=str(user_id),
            message=f"Недостаточно средств на балансе! Ваш баланс: {balance} руб. 💸",
            random_id=int(
                time.time()) * 10000000, keyboard=json.dumps(keyboard_new))
        return

    ids = links_to_id(msg_text[6:])
    if ids != 'WRONG_SEARCH_FUNC':
        if (ids[0] == 'Я') or (ids[0] == 'я'):
            ids[0] = user_id
        elif (ids[1] == 'Я') or (ids[1] == 'я'):
            ids[1] = user_id
        new_balance = balance - 1
        params = {
            "r_type": "update_user_balance",
            "user_id": user_id,
            "new_balance": new_balance
        }
        requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og",
            params=params)
        api.messages.send(access_token=token, user_id=str(user_id),
            message=f"🔥 Запрос на поиск между {ids[0]} и {ids[1]}", random_id=int(
                time.time()) * 10000000, keyboard=json.dumps(keyboard_new))
        datatosend = {
            "user_id": int(user_id),
            "entrance_id": ids[0],
            "target_id": ids[1]
        }
    }
    try:
        client = boto3.client(
            service_name='sqs',
            endpoint_url='https://message-queue.api.cloud.yandex.net',
            region_name='ru-central1'
        )

        params = {
            "r_type": "update_query_status",
            "user_id": user_id,
            "new_status": True
        }
        queryStatus = (requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og",
            params=params))

        # Send message to queue
        client.send_message(
            QueueUrl="https://message-queue.api.cloud.yandex.net/b1g96b95lu5trfs7poul/dj6000000005ifbb05g9/vkbot-power",
            MessageBody=json.dumps(datatosend)
        )

```

Функция vk_callback функции обработчика чат-бота. Часть 5.

Если пользователь присылает 'Поиск ссылка 1 ссылка 2', то мы сначала проверяем нет ли у него активных запросов, если есть, то присылаем сообщения, в котором уведомляем о невозможности запускать одновременно нескольких запросов из-за ограничений VK. Далее проверяем, что на балансе есть средства и он авторизован, в случае если одно из условий не выполняется, присылаем предупреждающее сообщение. Если все нормально, то обновляем баланс и преобразуем сообщение в список из двух ID с помощью функции `links_to_id` и отправляем запрос на функцию по поиску связей, а также устанавливаем в базе данных статус активного запроса. Запрос отправляется через сервис Yandex Message Queue, который помогает избежать перегрузку функции, в случае большого количества запроса, новые отправляются в очередь и вызываются после завершения старых.

```
    )

    except Exception as e:
        api.messages.send(access_token=token, user_id=str(user_id),
                          message='Не получилось', random_id=int(
                              time.time()) * 10000000, keyboard=json.dumps(keyboard_new))

    else:
        api.messages.send(access_token=token, user_id=str(user_id),
                          message="⚠ Неверный формат запроса! Пожалуйста, проверьте Ваш запрос или прочитайте инструкцию.",
                          random_id=int(
                              time.time()) * 10000000, keyboard=json.dumps(keyboard_new))

    else:
        api.messages.send(access_token=token, user_id=str(user_id),
                          message="⚠ Я не могу понять ваш запрос :( \n Напишите 'Инструкция', если хотите ознакомиться с функционалом",
                          random_id=int(time.time()) * 10000000, keyboard=json.dumps(keyboard_new))
```

Функция `vk_callback` функции обработчика чат-бота. Часть 6.

Обрабатываем исключения, если запрос не правильный или сообщение не подходит ни под один сценарий или же возникла другая ошибка при выполнении кода.

```

def generate_link(pay_amount, user_id):
    user = "admin"
    password = "5bfd5bf2cc06"

    server_paykeeper = "https://social-navigato-vk.server.paykeeper.ru"

    uri_token = "/info/settings/token/"
    uri_invoice = "/change/invoice/preview/"

    payment_data = {
        "pay_amount": pay_amount,
        "clientid": str(user_id)
    }

    auth = HTTPBasicAuth(user, password)

    response = requests.get(server_paykeeper + uri_token, auth=auth, data=payment_data)

    response_json = response.json()

    if "token" in response_json:
        token = response_json["token"]
    else:
        return

    print(token)

    payment_data["token"] = token

    response = requests.post(server_paykeeper + uri_invoice, data = payment_data, auth=auth)

    response_json = response.json()

    if "invoice_id" in response_json:
        invoice_id = response_json["invoice_id"]
    else:
        return

    link = f"{server_paykeeper}/bill/{invoice_id}/"

    return link

```

Функция generate_link функции обработчика чат-бота.

Данная функция отвечает за генерацию ссылки на оплату. Сначала мы получаем токен для авторизации в системе эквайринга счета, в счет мы включаем выбранную сумму пополнения и также ID пользователя во Вконтакте, далее получив токен мы запрашиваем номер созданного счета и создаем ссылку по шаблону. Авторизации всех запросов происходит по системе HTTPBasicAuth.

Данный код не ресурсозатратен, поэтому выставляем маленький таймаут в 20 секунд и минимальное количество RAM, также указываем переменные окружения, а именно ключ аккаунта Yandex Cloud и token для запросов в VK (приложение 13).

3.3 ФУНКЦИЯ ДЛЯ ВЗАИМОДЕЙСТВИЙ С БАЗОЙ ДАННЫХ КЛИЕНТОВ

База данных развернута на Yandex Databases и содержит одну таблицу `user_info`, в которой есть 4 атрибута: `user_id` (PK, `uint32`) ID пользователя во Вконтакте, `access_token` (string) токен получаемый при авторизации пользователя в чат-боте, `active_query` (bool) есть ли активный запрос на поиск, `balance` (`uint32`) внутренний баланс пользователя в чат-боте.

Из нестандартных модулей используется только `ydb`, этот модуль предназначен для работы с Yandex Databases (приложение 14). В первых строчках мы устанавливаем соединение с базой данных.

```
def handler(event, context):
    try:
        r_type = event['queryStringParameters']['r_type']
        user_id = event['queryStringParameters']['user_id']

        if not user_id.isdecimal():
            return {'statusCode': 200,
                    'body': json.dumps("id must be int")}

        if not user_exists(session = session, user_id = user_id):
            return {'statusCode': 200,
                    'body': json.dumps("User not found")}

        if r_type == "get_user_token":
            response = get_user_token(session=session, user_id=user_id)
        elif r_type == "get_user_balance":
            response = get_user_balance(session=session, user_id=user_id)
        elif r_type == "update_user_balance":
            new_balance = event['queryStringParameters']['new_balance']
            response = update_user_balance(session = session, user_id = user_id, new_balance = new_balance)
        elif r_type == "get_query_status":
            response = get_query_status(session = session, user_id = user_id)
        elif r_type == "update_query_status":
            new_status = event['queryStringParameters']['new_status']
            response = update_query_status(session = session, user_id = user_id, new_status = new_status)
        elif r_type == "if_user":
            response = str(user_exists(session = session, user_id = user_id))

    except Exception as e:
        response = str(e)

    return {
        'statusCode': 200,
        'body': str(response)
    }
```

Функция handler функции для работы с БД.

Мы из запроса определяем его тип и ID пользователя, для кого нужен запрос. Далее проверяем наличие пользователя в БД, если его нет, то возвращаем ошибку, в ином случае переходим к блоку `if` и в зависимости от типа запроса выполняем одну из функций.

get_user_token (приложение 15) - генерирует SQL запрос на получение токена пользователя, user_exists - проверяет есть ли пользователь в БД с помощью SQL запроса и сверки результата SELECT, если он возвращает 0 элементов - пользователя нет, get_user_balance выполняет SQL запрос в базу данных для получения баланса пользователя.

```
def update_user_balance(session, user_id, new_balance):
    data = session.transaction(ydb.SerializableReadWrite()).execute(
        "UPDATE user_info SET balance = {new_balance} WHERE user_id = {user_id}".format(new_balance=new_balance, user_id = user_id),
        commit_tx=True,
    )
    return "Done"

def update_query_status(session, user_id, new_status):
    data = session.transaction(ydb.SerializableReadWrite()).execute(
        "UPDATE user_info SET active_query = {new_status} WHERE user_id = {user_id}".format(new_status=new_status, user_id = user_id),
        commit_tx=True,
    )
    return "Done"

def get_query_status(session, user_id):
    data = session.transaction(ydb.SerializableReadWrite()).execute(
        "SELECT active_query FROM `user_info` WHERE `user_id` = {user_id};".format(user_id=str(user_id)),
        commit_tx=True,
    )
    return data[0].rows[0].active_query
```

Функции update_user_balance, update_query_status, get_query_status функции для работы с БД.

update_user_balance - генерирует SQL запрос для обновления баланса его в базе данных, update_query_status - меняет статус active_query атрибута у указанного пользователя, get_query_status получает значение active_query атрибута у указанного пользователя.

Эта функция выполняет простейшие запросы в БД и требует ресурсов, поэтому выставяем минимальные настройки. Переменные окружения, в них храним параметры для подключения к БД (приложение 18).

3.4 ФУНКЦИЯ ДЛЯ АВТОРИЗАЦИИ ПОЛЬЗОВАТЕЛЕЙ

Данная функция предназначена для обработки авторизации пользователя и получения его токена, который позволяет сервису видеть, какая информация доступна пользователю.

Аналогично прошлой главе подключаемся к БД (приложение 17).

```

def handler(event, context):
    try:
        code = event["queryStringParameters"]['code']
        response = create_access_token(code=code)
        if not user_exists(session = session, user_id = response["id"]):
            new_user(session = session, user_info = response)
        else:
            response = update_user(session = session, user_info = response)

        response = {}
        response["statusCode"] = 302
        response["headers"] = {'Location': 'https://vk.com/im?sel=-210111570'}
        data = {}
        response["body"] = json.dumps(data)
    except Exception as e:
        response = str(e)

    return response

```

Функция handler функции пользовательской авторизации.

Сначала получаем код передаваемый ВК при авторизации пользователя, получаем токен используя код, далее проверяем авторизован ли уже пользователь, если нет, то создаем новую запись в БД функцией new_user, иначе обновляем токен у юзера в БД с помощью функции update_user. После этого возвращаем пользователя на страницу чат-бота и отправляем ему сообщение используя функцию описанную в 3.2 отправляя на нее соответствующий запрос.

```

def user_exists(session, user_id):
    data = session.transaction(ydb.SerializableReadWrite()).execute(
        "SELECT 'user_id' FROM 'user_info' WHERE 'user_id' = {user_id};".format(user_id=str(user_id)),
        commit_tx=True,
    )
    if data[0].rows == []:
        return False
    else:
        return True

def new_user(session, user_info):
    session.transaction(ydb.SerializableReadWrite()).execute(
        "INSERT INTO 'user_info' ('user_id', 'access_token', 'balance', 'active_query') VALUES ({user_id}, '{access_token}', {balance}, {status_to_set});".format(user_id=int(user_info["id"]),
        access_token=str(user_info["access_token"]), balance=1, status_to_set=False),
        commit_tx=True,
    )
    tobotdata = {
        "type": "auth",
        "user_id": str(user_info["id"])
    }
    requests.post("https://functions.yandexcloud.net/d4euu8qh1474ba3ii2ia", json = {"type": "auth", "user_id": str(user_info["id"])})

```

Функции user_exists и new_user функции пользовательской авторизации.

Также как в функции описанной в 3.3 мы проверяем наличие пользователя в БД функцией user_exists, функция new_user добавляет нового пользователя в базу данных.

update_user (приложение 18) - отправляет SQL запрос с новым токеном в БД обновляя запись пользователя в ней. create_access_token - отправляем

GET запрос в ВК для получения токена пользователя, получая ответ записываем его в переменную и возвращаем вместе с ID пользователя в формате словаря.

Настройки данной функции полностью аналогичны функции представленной в параграфе 3.3 (приложение 19) .

3.5 ФУНКЦИЯ ДЛЯ ОБРАБОТКИ ПЛАТЕЖЕЙ

Прием платежей производится с помощью интернет-эквайринга от Альфа-Банка, при участии их партнера PayKeeper. Фискализация автоматическая благодаря решению от PayKeeper совместно с Businesskassa. После успешной оплаты, PayKeeper отправляет на мою функцию-обработчик POST уведомление, которое и позволяет принимать оплату автоматически. Тут используются все уже ранее описанные модули (приложение 20).

Функция `add_balance` (приложение 21) используется для добавления пользователю средств на счет в случае успешной оплаты, сама функция полностью аналогична по добавлению баланса в функции по работе с БД пользователей.

Функция `echo` (приложение 21), как и в функции поиска, используется для отправки сообщений пользователям.

```

def handler(event, context):

    #echo(335328970, base64.b64decode(event["body"]).decode("utf-8"))

    decoded_body = base64.b64decode(event["body"]).decode("utf-8")
    body_dict = dict(parse.parse_qsl(decoded_body))

    secret_seed = os.getenv('pass');
    payment_id = body_dict['id'];
    order_sum = body_dict['sum'];
    clientid = body_dict['clientid'];
    key = body_dict['key'];

    sig_val = hashlib.md5((payment_id + order_sum + clientid + secret_seed).encode("utf-8")).hexdigest()

    if key != sig_val:
        return {
            'statusCode': 200,
            'body': 'No hash match'
        }

    key_to_return = hashlib.md5((payment_id + secret_seed).encode("utf-8")).hexdigest()

    sum_as_int = int(float(order_sum))

    try:
        intclientid = int(clientid)
    except Exception as e:
        echo(335328970, str(e) + " " + str(payment_id))
        return {
            'statusCode': 200,
            'body': 'OK {key}'.format(key=key_to_return)
        }

```

Функция handler в функции обработки платежей. Часть 1.

Основная функция содержит в себе логику обработки POST уведомления от эквайринга. Для начала мы декодируем переданные параметры с помощью base64 декодера и преобразуем это в словарь, для удобной работы с параметрами. Сверяем сигнатурное значение с ключом. Сигнатурное значение получаем с помощью хеширования параметров по правилам MD5 хеша. Генерируем аналогично ключ, который надо вернуть эквайрингу, чтобы утвердить обработку нами уведомления. Проверяем является ли clientid числом, если нет - то платеж тестовый, потому что во Вконтакте все id - числа, а значит не надо добавлять никому баланс, нужно лишь вернуть эквайрингу ответ.

Если clientid число, то определяем по сумме платежа количество токенов, который нужно добавить клиенту на счет и вызываем функцию add_balance с соответствующими параметрами. Отправляем сообщение об успешной оплате клиенту и возвращаем эквайринг подтверждение обработки нами платежа (приложение 22).

Настройки стандартные, pass - пароль настраиваемый в эквайринги для хеширования сигнатурного значения и token для отправки сообщений от имени бота (приложение 23).

4. ЗАКЛЮЧЕНИЕ

Мною была проделана работа по созданию сервиса по поиску социальных связей с использованием технологии бессерверных вычислений. Были освоены разные технологии для построения ИТ-инфраструктуры, изучены возможности графовых баз данных и их преимуществ перед обычными в особых сценариях. Получился полезный сервис с приятным и удобным интерфейсом, была создана большая инфраструктура для данного проекта задействующая Yandex Cloud Functions, Yandex Message Queue, Yandex Databases, Yandex Compute Cloud и Neo4j 4.4. Комбинация данных сервисов помогла достичь максимальной отказоустойчивости сервиса и скорости его работы. В дальнейшем этот проект можно монетизировать и развивать добавляя новый функционал и улучшая существующий. Сервис соответствует всем правилам Вконтакте, а также оплата производится в соответствии с законодательством РФ. База данных уже содержит в себе 60 миллионов пользователей Вконтакте и 400 миллионов записей о их связях, благодаря выбранным мною технологиям, обслуживание и содержание инфраструктуры обходится дешевле, чем альтернативные способы возможной реализации такого проекта. Также стоит заметить, что проект соответствует правилам SOLID, все блоки максимально разбиты по зонам ответственности, это позволяет в дальнейшем добавлять новый функционал не внося изменения в уже существующие модули, а также помогает быстро находить и устранять неполадки при работе сервиса.

5. СПИСОК ЛИТЕРАТУРЫ

1. Библиотека vk Python - <https://pypi.org/project/vk/>
2. Библиотека vk_api Python - https://github.com/python273/vk_api
3. Библиотека ydb-python-sdk Python -
<https://github.com/ydb-platform/ydb-python-sdk>
4. Библиотека neo4j Python - <https://pypi.org/project/neo4j/>
5. Yandex Cloud Functions - Разработка на Python -
<https://cloud.yandex.ru/docs/functions/lang/python/>
6. Yandex Databases - Документация - <https://cloud.yandex.ru/docs/ydb/>
7. Yandex Message Queue - Документация -
<https://cloud.yandex.ru/docs/compute/>
8. VK API - Описание методов API - <https://dev.vk.com/method>
9. Neo4j - Neo4j 4.4 Cypher Manual -
<https://neo4j.com/docs/cypher-manual/current/>
10. ФЗ-54 - О применение контрольно-кассовой техники при
осуществление расчетов в РФ -
http://www.consultant.ru/document/cons_doc_LAW_42359/

6. ПРИЛОЖЕНИЯ

Приложение №1

```
def echo(user_id, response):
    session = vk.Session()
    api = vk.API(session, v=5.131)
    logging.info("отправил")
    api.messages.send(
        access_token=os.getenv('token'),
        user_id=user_id, message=response,
        random_id=int(time.time()) * 10000000)
```

Приложение №2

```
def errors_catch(error):
    string_error = str(error)
    if ('5' in string_error) and ('15' not in string_error):
        logging.critical("Error while auth mb something wrong with the access token --- " + string_error)
        return "❗ Возникла ошибка авторизации, пожалуйста, попробуйте авторизоваться заново!"
    elif ('18' in string_error) or ('30' in string_error):
        logging.error("18 or 30 error--- " + string_error)
        return "❗ У вас нет доступа к одной из страниц."
    elif ('29' in string_error):
        logging.error("DAY LIMIT REACHED !!! TRY TOMORROW!--- " + string_error)
        return "❗ Вы достигли лимита запросов, это ограничение от VK, к сожалению, избежать его нельзя. Обычно лимит сбрасывается через час - день."
    else:
        logging.error("UNKNOWN ApiError--- " + string_error)
        return "❗ Возникла неизвестная ошибка, если эта ошибка повторяется у Вас, пожалуйста, напишите администратору группы." + string_error
```

Приложение №3

```
def return_balance(user_id):
    params = {
        "r_type": "get_user_balance",
        "user_id": user_id
    }
    balance = int((requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og", params=params)).text)
    new_balance = balance + 1
    params = {
        "r_type": "update_user_balance",
        "user_id": user_id,
        "new_balance": new_balance
    }
    requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og", params=params)

def get_user_token(user_id):
    params = {
        "r_type": "get_user_token",
        "user_id": user_id
    }
    access_token = (requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og", params=params)).text
    logging.info(access_token)
    return access_token
```


Приложение №4

```
def convert(lst):
    res = []
    if isinstance(lst, int):
        res.append({'id': f'{str(lst)}'})
        return res

    for x in lst:
        res.append({'id': f'{str(x)}'})
    return res

def graph_maker(new_users, root_friend):

    nodes = convert(new_users)

    with driver.session() as session:
        session.write_transaction(add_lots_nodes, str(root_friend), nodes)
```

Приложение №5

```
def search_for_friend(first_user_id, second_user_id):
    try:
        if first_user_id in get_friend(second_user_id):
            return True
        else:
            return False
    except Exception as e:
        logging.warn(str(e) + " -- search_for_friend")
        return False
```

Приложение №6

```
def all_paths_query(tx, first_us, second_us):
    res = tx.run(
        f'MATCH p = allShortestPaths((:Person {{id: "{first_us}"}})-[*..5]-(:Person {{id: "{second_us}"}})) RETURN p LIMIT 4')
    rec = res.values()
    if rec[0] == None:
        return "NoPath"
    paths_to_return = []
    for path in rec:
        nodes = path[0].nodes
        list_to_add = []
        for node in nodes:
            list_to_add.append(node["id"])
        paths_to_return.append(list_to_add)

    for path in paths_to_return:
        for i in range(len(path) - 1):
            if not search_for_friend(int(path[i]), int(path[i + 1])):
                paths_to_return.remove(path)

    return paths_to_return
```

Приложение №7

```
def add_second_user_lvl2(friends_list):
    for user in friends_list:
        try:
            new_list = get_friend(user)
            graph_maker(new_list[:100], user)
        except:
            pass
```

Приложение №8

Точка входа ?

Параметры

Таймаут, с

Память
128 МБ 4096 МБ

Сервисный аккаунт или

Переменные окружения ? = ×

=

Приложение №9

```
import random
import vk
import json
import os
import time
import requests
import boto3
import hashlib
from urllib import parse
from urllib.parse import urlparse
```

Приложение №10

```
def vk_callback(lambda_event, token):
    start_text = "Привет! 🤖 Рады видеть тебя у нас в сообществе!\n\nПожалуйста, внимательно прочитай информацию!\n\n❌ Наш " +
    "бот поможет тебе найти цепочку знакомств между двумя людьми во вконтакте.\n\n? Может быть ты слышал что-то о теории 6 рукопожатий? Да-да именно" +
    " это мы здесь и проверяем!\n\nДля начала тебе нужно авторизоваться нажав на соответствующую кнопку в меню бота, далее ты сможешь пополнить баланс и найти цепочку" +
    " знакомств между двумя пользователями.\n\nПоиск идет до 15 минут, если за это время не получится найти цепочку, деньги вернутся на баланс бота! ✅\n\n🔔 Подробнее" +
    " о команде поиска можно прочитать в инструкции!\n\n👋 Приятного пользования!"

    insrt_text = "🔍 Команда для поиска цепочки знакомств между двумя пользователями:\n\n📩 Поиск (Ссылка или ID первого пользователя) (Ссылка или ID второго пользователя)\n\n"+
    "Вместо одного из пользователей можно указать 'я', так обозначается ваш ID\n\nПример запроса:\n\n📩 Поиск id11111 id22222\n\n📩 Поиск vk.com/firstuser vk.com/seconduser\n\n📩 "+
    "Поиск я dipov\n\nВконтакте есть свои ограничения, которые мы нарушать не хотим и не можем, если вы столкнулись с ограничением на количество запросов, пожалуйста, подождите "+
    "сутки и попробуйте еще раз\n\n📩 Со всеми вопросами вы можете написать администратору сообщества"
```

Приложение №11

```

keyboard_new = {
  "one_time": False,
  "buttons": [
    [
      {
        "action": {
          "type": "text",
          "payload": "{\\"button\\": \\"1\\"}",
          "label": "Инструкция"
        },
        "color": "negative"
      },
      {
        "action": {
          "type": "text",
          "payload": "{\\"button\\": \\"1\\"}",
          "label": "Авторизация"
        },
        "color": "negative"
      }
    ],
    [
      {
        "action": {
          "type": "text",
          "payload": "{\\"button\\": \\"2\\"}",
          "label": "Пополнить баланс"
        },
        "color": "primary"
      },
      {
        "action": {
          "type": "text",
          "payload": "{\\"button\\": \\"3\\"}",
          "label": "Баланс"
        },
        "color": "primary"
      }
    ],
    [
      {
        "action": {

```

```

keyboard_auth = {
  "one_time": False,
  "inline": True,
  "buttons": [
    [
      {
        "action": {
          "type": "open_link",
          "payload": "{\\"button\\": \\"2\\"}",
          "link": "https://oauth.vk.com/authorize?client_id=8049680&d",
          "label": "Авторизоваться"
        }
      }
    ]
  ]
}

instructionKeyboardButton = {
  "one_time": False,
  "inline": True,
  "buttons": [
    [
      {
        "action": {
          "type": "text",
          "payload": "{\\"button\\": \\"2\\"}",
          "label": "Инструкция"
        }
      }
    ]
  ]
}

```

```

elif (msg_text == 'god_give_me_money'):
    params = {
        "r_type": "get_user_balance",
        "user_id": user_id
    }
    balance = (requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og",
        params=params))
    if (balance.text == "User not found"):
        api.messages.send(access_token=token, user_id=str(user_id),
            message="👤 Сначала вам нужно авторизоваться!", random_id=int(
                time.time()) * 10000000, keyboard=json.dumps(keyboard_auth))
        return
    new_balance = int(balance.text) + 10
    params = {
        "r_type": "update_user_balance",
        "user_id": user_id,
        "new_balance": new_balance
    }
    requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og", params=params)
    api.messages.send(access_token=token, user_id=str(user_id), message="Бор вас услышал!", random_id=int(
        time.time()) * 10000000, keyboard=json.dumps(keyboard_new))
elif (msg_text == 'Инструкция') or (msg_text == 'инструкция'):
    api.messages.send(access_token=token, user_id=str(user_id), message=insrt_text, random_id=int(
        time.time()) * 10000000, keyboard=json.dumps(keyboard_new))
elif (msg_text == 'Авторизация') or (msg_text == 'авторизация'):
    api.messages.send(access_token=token, user_id=str(user_id), message="Для регистрации перейдите по ссылке",
        random_id=int(
            time.time()) * 10000000, keyboard=json.dumps(keyboard_auth))

elif (msg_text == 'Поиск'):
    api.messages.send(access_token=token, user_id=str(user_id), message="Для поиска введите\n'Поиск Ссылка1 Ссылка2'\n✅ Подробнее в инструкции! ",
        random_id=int(
            time.time()) * 10000000, keyboard=json.dumps(instructionKeyboardButton))

```

Приложение №13

Точка входа ?

Параметры

Таймаут, с

Память
128 МБ 4096 МБ

Сервисный аккаунт или

Переменные окружения ?

<input type="text" value="AWS_ACCESS_"/>	=	<input type="text" value="YCAJEPKkWe5AisXI8MC
XyQvTM"/>	<input type="button" value="X"/>
<input type="text" value="AWS_SECRET_"/>	=	<input type="text" value="YCOQ326hY4zd0biCNigi
fubnAU9kxZZ09XhCauzJ"/>	<input type="button" value="X"/>
<input type="text" value="pass1"/>	=	<input type="text" value="ppkvY6h2J4Pu6mIM9Wt
U"/>	<input type="button" value="X"/>
<input type="text" value="token"/>	=	<input type="text" value="b142cdcbcfefbfd91e879
234605219c90769576b4
6ca31b1d8fd58c450427
6f94e3b5513f1b6d3cf89
413"/>	<input type="button" value="X"/>
<input type="text" value="Ключ"/>	=	<input type="text" value="Значение"/>	

Приложение №14

```
import json
import os
import ydb as ydb
```

```
driver = ydb.Driver(endpoint=os.getenv('YDB_ENDPOINT'), database=os.getenv('YDB_DATABASE'))
driver.wait(fail_fast=True, timeout=5)
session = driver.table_client.session().create()
```

Приложение №15

```
def get_user_token(session, user_id):
    # create the transaction and execute query.
    data = session.transaction(ydb.SerializableReadWrite()).execute(
        "SELECT `access_token` FROM `user_info` WHERE `user_id` = {user_id};".format(user_id=str(user_id)),
        commit_tx=True,
    )
    if data[0].rows[0].access_token == None:
        return "This user doesnt have token yet"

    return data[0].rows[0].access_token.decode('utf-8')

def user_exists(session, user_id):
    data = session.transaction(ydb.SerializableReadWrite()).execute(
        "SELECT `user_id` FROM `user_info` WHERE `user_id` = {user_id};".format(user_id=str(user_id)),
        commit_tx=True,
    )
    if data[0].rows == []:
        return False
    else:
        return True

def get_user_balance(session, user_id):
    data = session.transaction(ydb.SerializableReadWrite()).execute(
        "SELECT * FROM `user_info` WHERE `user_id` = {user_id};".format(user_id=str(user_id)),
        commit_tx=True,
    )
    balance = data[0].rows[0].balance
    if balance == None:
        return "0"
    return balance
```

Приложение №16

Точка входа ?

Параметры

Таймаут, с

Память 128 МБ 4096 МБ

Сервисный аккаунт или

Переменные окружения ?

YDB_DATABASES = /ru-central1/b1g96b95lu5trfs7poul/etn52fche4gkq6c cs3ge ×

YDB_ENDPOINT = grpc://ydb.serverless.yandexcloud.net:2135 ×

Ключ = Значение

Приложение №17

```
import json
import requests
import os
import ydb as ydb

driver = ydb.Driver(endpoint=os.getenv('YDB_ENDPOINT'), database=os.getenv('YDB_DATABASE'))
driver.wait(fail_fast=True, timeout=5)
session = driver.table_client.session().create()
```

Приложение №18

```
def update_user(session, user_info):
    session.transaction(ydb.SerializableReadWrite()).execute(
        "UPSERT INTO `user_info` (`user_id`, `access_token`) VALUES ({user_id}, '{access_token}');".format(user_id = int(user_info["id"]), access_token = str(user_info["access_token"])),
        commit_tx=True,
    )
    tobotdata = {
        "type": "auth",
        "user_id": str(user_info["id"])
    }
    requests.post("https://functions.yandexcloud.net/d4euu8qh1474bajii21a", json = {"type": "auth", "user_id": str(user_info["id"])})

def create_access_token(code):
    params = {
        "client_id": "8049688",
        "client_secret": "M4GpSCFkonIOzeymTNk",
        "redirect_uri": "https://functions.yandexcloud.net/d4epbm3dn86epcb1k1n",
        "code": code
    }
    r = requests.get("https://oauth.vk.com/access_token", params=params)
    response_data = r.json()
    user_info = {
        "id": response_data["user_id"],
        "access_token": response_data["access_token"]
    }
    return user_info
```

Приложение №19

Точка входа ?

index.handler

Параметры

Таймаут, с

3

Память

128 МБ

128 МБ

4096 МБ

Сервисный аккаунт

default



или

Создать новый

Переменные окружения ?

YDB_DATABAS

=

/ru-central1/b1g96b95lu5trf
s7poul/etn52fche4gkq6c
cs3ge



YDB_ENDPOIN

=

grpc://ydb.serverless.y
andexcloud.net:2135



Ключ

=

Значение

Добавить

Приложение №20

```

import vk
import os
import time
import json
import hashlib
import requests
import base64
from urllib import parse

```

Приложение №21

```

def add_balance(amn_to_add, user_id):
    params = {
        "r_type": "get_user_balance",
        "user_id": user_id
    }
    balance = (requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og", params=params))
    new_balance = int(balance.text) + amn_to_add
    params = {
        "r_type": "update_user_balance",
        "user_id": user_id,
        "new_balance": new_balance
    }
    requests.get("https://functions.yandexcloud.net/d4e15rdqtnakc2noq0og", params=params)

def echo(user_id, response):
    session = vk.Session()
    api = vk.API(session, v=5.131)
    api.messages.send(
        access_token=os.getenv('token'),
        user_id=user_id, message=response,
        random_id=int(time.time()) * 10000000)

```

Приложение №22

```

if sum_as_int == 59:
    add_balance(1, clientid)
elif sum_as_int == 99:
    add_balance(2, clientid)
elif sum_as_int == 199:
    add_balance(5, clientid)
elif sum_as_int == 449:
    add_balance(10, clientid)
else:
    echo(clientid, "Неверная сумма пополнения, обратитесь к администратору сообщества!")
    return {
        'statusCode': 200,
        'body': 'OK {key}'.format(key=key_to_return)
    }

echo(clientid, "✅ Баланс успешно пополнен!")

return {
    'statusCode': 200,
    'body': 'OK {key}'.format(key=key_to_return)
}

```

Приложение №23

Параметры

Таймаут, с

Память

128 МБ

4096 МБ

Сервисный аккаунт

Не выбрано



или

Создать новый

Переменные окружения ?

pass

=

e9qRhK(Lc5qERF4



token

=

b142cdcbcfefbfd91e879
234605219c90769576b4
6ca31b1d8fd58c450427
6f94e3b5513f1b6d3cf89
413



Ключ

=

Значение

Добавить