

Министерство образования и науки Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)

Институт радиоэлектроники и информационных технологий — РТФ

Кафедра «учебно-научный центр информационной безопасности»

Оценка

Руководитель курсового

проектирования Мирвода С.Г

Члены комиссии

Дата защиты

ОТЧЕТ

по курсовой работе

по теме: «Телеграмм-бот для администратора гостиничного комплекса»

по дисциплине: «Языки программирования»

Студенты: Ваулин Владислав Игоревич

Седельников Александр Алексеевич

Швецова Ксения Георгиевна

(ФИО) (Подпись)

Группа: РИ-311055

Екатеринбург

2024

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc155999501)

[**Цель проекта** 3](#_Toc155999502)

[**Видение проекта** 3](#_Toc155999503)

[**Функциональная диаграмма продукта (как он видится)** 4](#_Toc155999504)

[ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc155999505)

[ПРОЕКТИРОВАНИЕ 6](#_Toc155999506)

[**Создание и обоснование архитектуры продукта** 6](#_Toc155999507)

[**Требования к системному программному обеспечению, включающему в себя требования к операционной системе и СУБД** 6](#_Toc155999508)

[**Архитектурные диаграммы в составе** 7](#_Toc155999509)

[РАЗРАБОТКА 11](#_Toc155999510)

[**Диаграмма пакетов UML** 11](#_Toc155999511)

[**Диаграмма физической модели** 12](#_Toc155999512)

[ЗАКЛЮЧЕНИЕ 13](#_Toc155999513)

[ПРИЛОЖЕНИЕ А 14](#_Toc155999514)

[ПРИЛОЖЕНИЕ Б 28](#_Toc155999515)

[ПРИЛОЖЕНИЕ В 53](#_Toc155999516)

# ВВЕДЕНИЕ

На сегодняшний день функционирование гостиниц и гостиничных комплексов требует применения новых технологий. В обязательном порядке необходимо создание эффективной и функциональной информационной системы управления гостиницей, чтобы автоматизировать рабочие процессы, сконцентрировать огромный объем информации в одном месте, связать в одно целое разные службы отеля и повысить уровень и качество обслуживания клиентов.

## **Цель проекта**

Создать Телеграмм-бота, позволяющего администратору гостиничного комплекса вести работу по администрированию, осуществляя тем самым взаимодействие с базой данных комплекса, которую так же необходимо создать.

## **Видение проекта**

Телеграмм-бот с функциями регистрации посетителей, контроля брони и заказов необходим администратору гостиницы из-за своей мобильности (администратор может пользоваться им с любого устройства, имеющего доступ в сеть интернет), автоматизации процесса ведения учета, простого и понятного интерфейса, а также удобства взаимодействия с огромной базой данных, которая, в свою очередь, обеспечивает быстрый доступ и надежное хранение.

## **Функциональная диаграмма продукта (как он видится)**

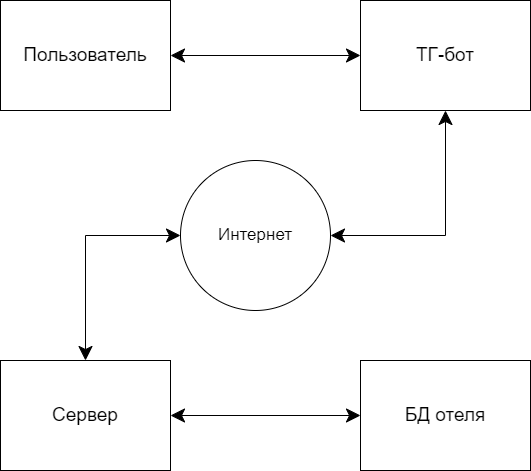


Рисунок 1 – Функциональная диаграмма

# ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Гостиничный комплекс состоит из нескольких зданий-гостиниц (корпусов). Каждый корпус имеет ряд характеристик, таких, как название, класс отеля, количество этажей в здании, общее количество комнат, комнат на этаже, местность номеров, наличие служб быта и развлечения: химчистка, питание, бассейн, сауна, бильярд и пр. От типа корпуса и местности номера зависит сумма оплаты за него. Все дополнительные услуги и развлечения оплачиваются отдельно. Ежедневная уборка номера, замена принадлежностей, работа электрика/сантехника (при необходимости) входят в стоимость проживания.

В брони указывается класс отеля, этаж, количество комнат и общее количество людей. Бронь может быть отменена за неделю до заселения, в таком случае бронь удаляется из базы данных. Также исследуется мнение жильцов о ценах и сервисе. Жалобы фиксируются и исследуются. Изучается статистика популярности номеров.

# ПРОЕКТИРОВАНИЕ

## **Создание и обоснование архитектуры продукта**

Для того, чтобы спроектировать телеграмм-бот и оптимизировать его так, чтобы запросы обрабатывались корректно и быстро, необходимо привести ряд требований, при которых продукт будет работать стабильно и точно.

Таким образом, продукт должен состоять из следующих частей: база данных, телеграмм-бот, а также серверная часть, включающая в себя работу тг-бота, написанного на определенном языке программирования, а также обработку запросов к БД.

Телеграмм-бот представляет собой интерфейс, позволяющий пользователю взаимодействовать непосредственно с базой данных.

Серверная часть необходима для обработки и преобразования простых запросов пользователя в запросы для БД.

## **Требования к системному программному обеспечению, включающему в себя требования к операционной системе и СУБД**

Для того, чтобы реализовать данный механизм внедрения системы телеграмм-бота для гостиничного комплекса, необходимо использовать корректное системное программное обеспечение – такой набор программ, которые управляют компонентами вычислительной системы, чтобы обеспечить её полное функционирование и работоспособность в целом.

Получаем, что необходимо использовать операционную систему Windows 7 и выше, так как используемый сервер является устройством, использующем именно эту операционную систему. Необходима возможность беспроблемного переноса системы на ОС Linux для последующего расширения

При создании базы данных использовалась объектно-реляционная система управления БД – PostgreSQL, которая позволяет создавать, хранить базы данных и работать с данными с помощью запросов на языке SQL. Такая система универсальная, а также динамично развивается в России. Кроме того, PostgreSQL выпускается для актуальных версий Windows и Linux.

## **Архитектурные диаграммы в составе**

1. Архитектурная диаграмма

Изображение выглядит как текст, снимок экрана, диаграмма, дизайн

Автоматически созданное описание

Рисунок 2 – Архитектурная диаграмма

Архитектурная диаграмма показывает процесс создания визуальных представлений компонентов программной системы.

1. Функциональная диаграмма UML

Изображение выглядит как текст, диаграмма, линия, рисунок

Автоматически созданное описание

Рисунок 3 – Функциональная диаграмма UML

Функциональная диаграмма отражает границы системы и ее взаимодействие с внешним миром, а также «актеров» и их связь с прецедентами.

1. Диаграмма последовательности UML (для одной выбранной ключевой функции)

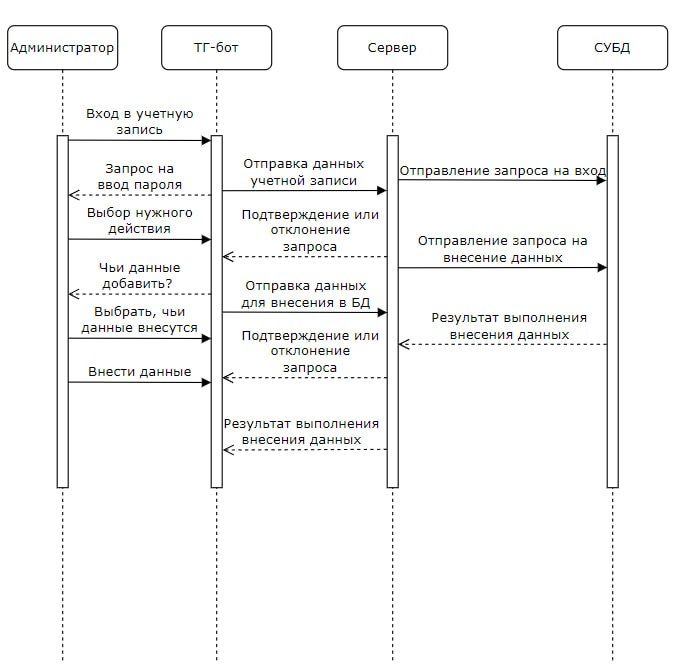


Рисунок 4 – Диаграмма последовательности

Такая диаграмма описывает один сценарий, при котором экземпляры объектов и сообщения, которыми обмениваются объекты в рамках одного прецедента. В качестве прецедента была выбрана одна из ключевых функций – внесение данных профиля клиента.

1. При наличии БД концептуальную модель БД

Изображение выглядит как диаграмма, линия, оригами, шаблон

Автоматически созданное описание

Рисунок 5 – Концептуальная модель БД

Подобная диаграмма показывает связь между объектами и их характеристиками. Концептуальная модель нужна для дальнейшего проектирования базы данных.

# РАЗРАБОТКА

Согласно поставленным задачам, был выбран стек технологий, а именно:

* PyCharm – среда разработки, специализирующаяся на языке python. Поддерживает интеграцию Git;
* Python 3.11 – язык программирования для реализации функционала бота и взаимодействия с базой данных;
* pyTelegramBotAPI (telebot) – библиотека, которая позволяет создать бота, отвечающего на сообщения пользователя, выполнять команды;
* RE – библиотека регулярных выражений для Python. Используется для разделения слов при внесении;
* Datetime – библиотека, которая предоставляет классы для обработки времени и даты разными способами. Используется для внутреннего логирования действий пользователя бота;
* PGAdmin 4.0 – это платформа с открытым исходным кодом для администрирования и разработки для PostgreSQL и связанных с ней систем управления базами данных
* PostgreSQL 15.2 – реляционная система управления базами данных;
* psycopg2 – библиотека для работы с базами данных PostgreSQL в языке программирования Python;

## **Диаграмма пакетов UML**

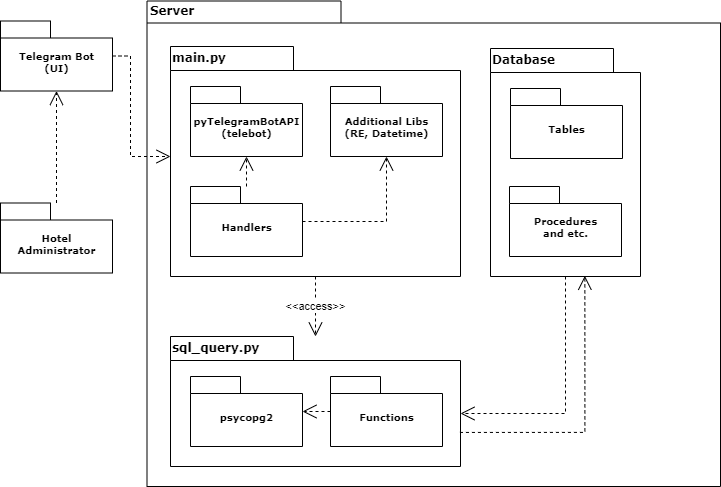


Рисунок 6 – Диаграмма пакетов

## **Диаграмма физической модели**

Используя открытый конструктор БД DbDesigner.net была построена ER-диаграмма

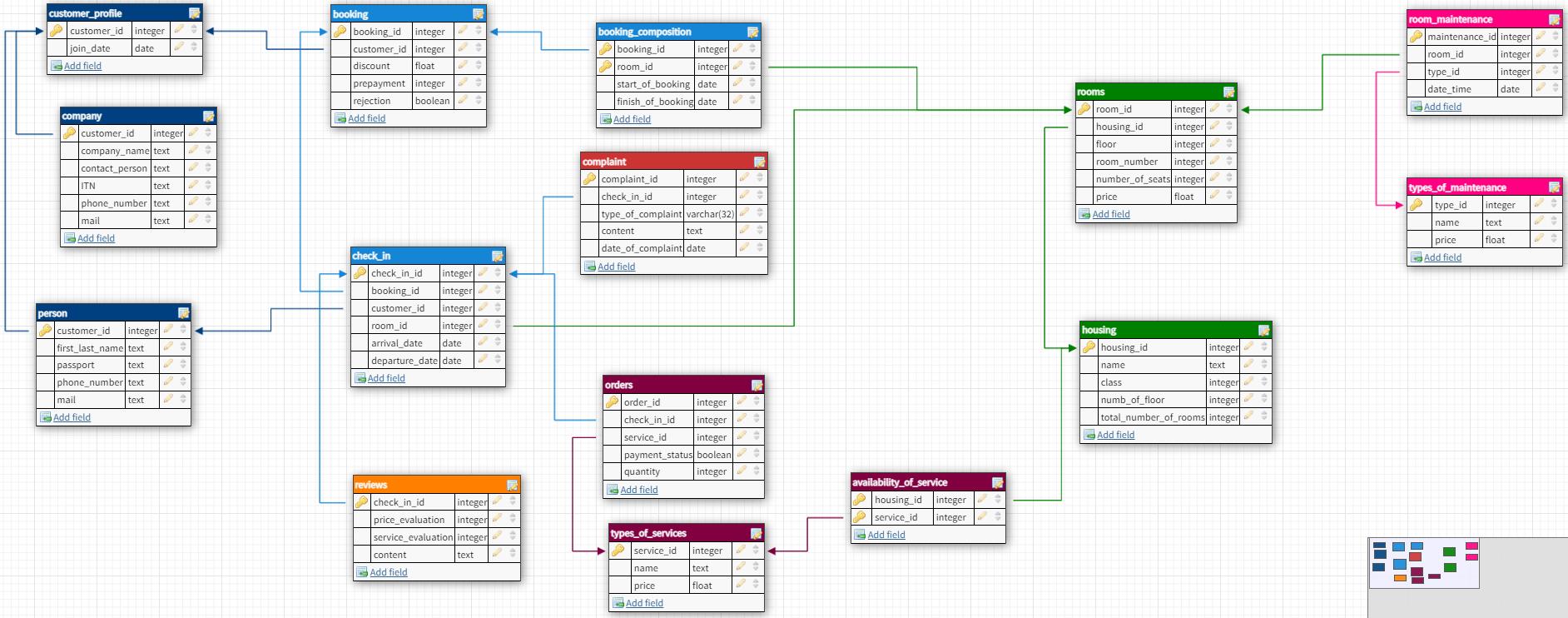


Рисунок 7 – Физическая модель БД

# ЗАКЛЮЧЕНИЕ

Итогами выполнения курсовой работы можно считать полностью удовлетворяющую требованиям систему администрирования, спроектированную в рамках предметной области «Гостиничный комплекс» и разработанную с помощью системы управления базами данных PostgreSQL посредством TG-бота и собственного сервера.

В процессе реализации системы были изучены предметная область, особенности методов и средств проектирования информационных систем, создания и модификации баз данных, установления связей между ними с помощью первичных ключей, а также управления ими посредством запросов SQL.

# ПРИЛОЖЕНИЕ А

**Создание базы данных**

-- Creating schema in DB

CREATE SCHEMA IF NOT EXISTS hotel;

-- Adding tables to schema

SET search\_path TO hotel;

CREATE TABLE "customer\_profile" (

"customer\_id" serial NOT NULL,

"join\_date" DATE NOT NULL,

CONSTRAINT "customer\_profile\_pk" PRIMARY KEY ("customer\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "company" (

"customer\_id" integer NOT NULL,

"company\_name" TEXT NOT NULL,

"contact\_person" TEXT NOT NULL,

"ITN" TEXT NOT NULL,

"phone\_number" TEXT NOT NULL,

"mail" TEXT NOT NULL,

CONSTRAINT "company\_pk" PRIMARY KEY ("customer\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "person" (

"customer\_id" integer NOT NULL,

"first\_last\_name" TEXT NOT NULL,

"passport" TEXT NOT NULL,

"phone\_number" TEXT NOT NULL,

"mail" TEXT NOT NULL,

CONSTRAINT "person\_pk" PRIMARY KEY ("customer\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "booking" (

"booking\_id" serial NOT NULL,

"customer\_id" integer NOT NULL,

"discount" FLOAT NOT NULL,

"prepayment" integer NOT NULL,

"rejection" BOOLEAN NOT NULL,

CONSTRAINT "booking\_pk" PRIMARY KEY ("booking\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "check\_in" (

"check\_in\_id" serial NOT NULL,

"booking\_id" integer,

"customer\_id" integer NOT NULL,

"room\_id" integer NOT NULL,

"arrival\_date" DATE NOT NULL,

"departure\_date" DATE NOT NULL,

CONSTRAINT "check\_in\_pk" PRIMARY KEY ("check\_in\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "reviews" (

"check\_in\_id" integer NOT NULL,

"price\_evaluation" integer NOT NULL,

"service\_evaluation" integer NOT NULL,

"content" TEXT NOT NULL,

CONSTRAINT "reviews\_pk" PRIMARY KEY ("check\_in\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "complaint" (

"complaint\_id" serial NOT NULL,

"check\_in\_id" integer NOT NULL,

"type\_of\_complaint" varchar(32) NOT NULL,

"content" TEXT NOT NULL,

"date\_of\_complaint" DATE NOT NULL,

CONSTRAINT "complaint\_pk" PRIMARY KEY ("complaint\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "booking\_composition" (

"booking\_id" integer NOT NULL,

"room\_id" integer NOT NULL,

"start\_of\_booking" DATE NOT NULL,

"finish\_of\_booking" DATE NOT NULL,

CONSTRAINT "booking\_composition\_pk" PRIMARY KEY ("booking\_id","room\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "orders" (

"order\_id" serial NOT NULL,

"check\_in\_id" integer NOT NULL,

"service\_id" integer NOT NULL,

"payment\_status" BOOLEAN NOT NULL,

"quantity" integer NOT NULL,

CONSTRAINT "orders\_pk" PRIMARY KEY ("order\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "types\_of\_services" (

"service\_id" serial NOT NULL,

"name" TEXT NOT NULL,

"price" FLOAT NOT NULL,

CONSTRAINT "types\_of\_services\_pk" PRIMARY KEY ("service\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "availability\_of\_service" (

"housing\_id" integer NOT NULL,

"service\_id" integer NOT NULL,

CONSTRAINT "availability\_of\_service\_pk" PRIMARY KEY ("housing\_id","service\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "housing" (

"housing\_id" serial NOT NULL,

"name" TEXT NOT NULL,

"class" integer NOT NULL,

"numb\_of\_floor" integer NOT NULL,

"total\_number\_of\_rooms" integer NOT NULL,

CONSTRAINT "housing\_pk" PRIMARY KEY ("housing\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "rooms" (

"room\_id" serial NOT NULL,

"housing\_id" integer NOT NULL,

"floor" integer NOT NULL,

"room\_number" integer NOT NULL,

"number\_of\_seats" integer NOT NULL,

"price" FLOAT NOT NULL,

CONSTRAINT "rooms\_pk" PRIMARY KEY ("room\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "types\_of\_maintenance" (

"type\_id" serial NOT NULL,

"name" TEXT NOT NULL,

"price" FLOAT NOT NULL,

CONSTRAINT "types\_of\_maintenance\_pk" PRIMARY KEY ("type\_id")

) WITH (

OIDS=FALSE

);

CREATE TABLE "rooms\_maintenance" (

"maintenance\_id" integer NOT NULL,

"room\_id" integer NOT NULL,

"type\_id" integer NOT NULL,

"date\_time" DATE NOT NULL,

CONSTRAINT "rooms\_maintenance\_pk" PRIMARY KEY ("maintenance\_id")

) WITH (

OIDS=FALSE

);

ALTER TABLE "company" ADD CONSTRAINT "company\_fk0" FOREIGN KEY ("customer\_id") REFERENCES "customer\_profile"("customer\_id");

ALTER TABLE "person" ADD CONSTRAINT "person\_fk0" FOREIGN KEY ("customer\_id") REFERENCES "customer\_profile"("customer\_id");

ALTER TABLE "booking" ADD CONSTRAINT "booking\_fk0" FOREIGN KEY ("customer\_id") REFERENCES "customer\_profile"("customer\_id");

ALTER TABLE "check\_in" ADD CONSTRAINT "check\_in\_fk0" FOREIGN KEY ("booking\_id") REFERENCES "booking"("booking\_id");

ALTER TABLE "check\_in" ADD CONSTRAINT "check\_in\_fk1" FOREIGN KEY ("customer\_id") REFERENCES "person"("customer\_id");

ALTER TABLE "check\_in" ADD CONSTRAINT "check\_in\_fk2" FOREIGN KEY ("room\_id") REFERENCES "rooms"("room\_id");

ALTER TABLE "reviews" ADD CONSTRAINT "reviews\_fk0" FOREIGN KEY ("check\_in\_id") REFERENCES "check\_in"("check\_in\_id");

ALTER TABLE "complaint" ADD CONSTRAINT "complaint\_fk0" FOREIGN KEY ("check\_in\_id") REFERENCES "check\_in"("check\_in\_id");

ALTER TABLE "booking\_composition" ADD CONSTRAINT "booking\_composition\_fk0" FOREIGN KEY ("booking\_id") REFERENCES "booking"("booking\_id");

ALTER TABLE "booking\_composition" ADD CONSTRAINT "booking\_composition\_fk1" FOREIGN KEY ("room\_id") REFERENCES "rooms"("room\_id");

ALTER TABLE "orders" ADD CONSTRAINT "orders\_fk0" FOREIGN KEY ("check\_in\_id") REFERENCES "check\_in"("check\_in\_id");

ALTER TABLE "orders" ADD CONSTRAINT "orders\_fk1" FOREIGN KEY ("service\_id") REFERENCES "types\_of\_services"("service\_id");

ALTER TABLE "availability\_of\_service" ADD CONSTRAINT "availability\_of\_service\_fk0" FOREIGN KEY ("housing\_id") REFERENCES "housing"("housing\_id");

ALTER TABLE "availability\_of\_service" ADD CONSTRAINT "availability\_of\_service\_fk1" FOREIGN KEY ("service\_id") REFERENCES "types\_of\_services"("service\_id");

ALTER TABLE "rooms" ADD CONSTRAINT "rooms\_fk0" FOREIGN KEY ("housing\_id") REFERENCES "housing"("housing\_id");

ALTER TABLE "rooms\_maintenance" ADD CONSTRAINT "rooms\_maintenance\_fk0" FOREIGN KEY ("room\_id") REFERENCES "rooms"("room\_id");

ALTER TABLE "rooms\_maintenance" ADD CONSTRAINT "rooms\_maintenance\_fk1" FOREIGN KEY ("type\_id") REFERENCES "types\_of\_maintenance"("type\_id");

-- Creating procedures, functions and views

SET search\_path TO hotel;

CREATE OR REPLACE PROCEDURE hotel.insert\_person(

IN flname text,

IN pass text,

IN numb text,

IN mail text)

LANGUAGE 'plpgsql'

AS $BODY$

declare

per int;

begin

select max(customer\_id) from customer\_profile into per;

insert into customer\_profile(customer\_id, join\_date)

values (per + 1, current\_date);

insert into person(customer\_id, first\_last\_name, passport, phone\_number, mail)

values (per + 1,flname, pass, numb, mail);

end;

$BODY$;

ALTER PROCEDURE hotel.insert\_person(text, text, text, text)

OWNER TO postgres;

CREATE OR REPLACE PROCEDURE hotel.insert\_company(

IN comp\_name text,

IN contact\_name text,

IN itn text,

IN numb text,

IN mail text)

LANGUAGE 'plpgsql'

AS $BODY$

declare

per int;

begin

select max(customer\_id) from customer\_profile into per;

insert into customer\_profile(customer\_id, join\_date)

values (per + 1, current\_date);

insert into company(customer\_id, company\_name, contact\_person, "ITN", phone\_number, mail)

values (per + 1, comp\_name, contact\_name, itn, numb, mail);

end;

$BODY$;

ALTER PROCEDURE hotel.insert\_company(text, text, text, text, text)

OWNER TO postgres;

CREATE OR REPLACE VIEW hotel.active\_bookings

AS

SELECT per.first\_last\_name,

comp.company\_name,

b.prepayment,

bc.start\_of\_booking,

bc.finish\_of\_booking

FROM hotel.booking b

JOIN hotel.booking\_composition bc ON bc.booking\_id = b.booking\_id

FULL JOIN hotel.person per ON per.customer\_id = b.customer\_id

FULL JOIN hotel.company comp ON b.customer\_id = comp.customer\_id

WHERE bc.start\_of\_booking > CURRENT\_DATE

ORDER BY bc.start\_of\_booking, comp.company\_name, per.first\_last\_name;

ALTER TABLE hotel.active\_bookings

OWNER TO postgres;

CREATE OR REPLACE PROCEDURE hotel.change(

IN type\_of\_client text,

IN name\_ text,

IN date\_ date)

LANGUAGE 'plpgsql'

AS $BODY$

declare

per int;

begin

if type\_of\_client = 'Компания'

then

select boo.customer\_id into per from company comp

inner join booking boo

on comp.customer\_id = boo.customer\_id

inner join booking\_composition bc

on bc.booking\_id = boo.booking\_id

where company\_name = name\_ and start\_of\_booking = date\_;

elseif type\_of\_client = 'Клиент'

then

select boo.customer\_id into per from person pe

inner join booking boo

on boo.customer\_id = pe.customer\_id

inner join booking\_composition bc

on bc.booking\_id = boo.booking\_id

where first\_last\_name = name\_ and booking\_id is not NULL;

end if;

update booking

set rejection = 'True'

where customer\_id = per;

end;

$BODY$;

ALTER PROCEDURE hotel.change(text, text, date)

OWNER TO postgres;

CREATE OR REPLACE VIEW hotel.today\_check\_in

AS

SELECT p.first\_last\_name,

p.phone\_number,

r.room\_number,

h.name AS name\_of\_housing

FROM hotel.person p

JOIN hotel.booking b ON b.customer\_id = p.customer\_id

JOIN hotel.booking\_composition bc ON bc.booking\_id = b.booking\_id

JOIN hotel.rooms r ON r.room\_id = bc.room\_id

JOIN hotel.housing h ON h.housing\_id = r.housing\_id

WHERE bc.start\_of\_booking = CURRENT\_DATE;

ALTER TABLE hotel.today\_check\_in

OWNER TO postgres;

CREATE OR REPLACE VIEW hotel.free\_rooms

AS

SELECT DISTINCT r.room\_number,

h.name AS name\_of\_housing

FROM hotel.booking\_composition bc

LEFT JOIN hotel.check\_in ch ON ch.booking\_id = bc.booking\_id

JOIN hotel.rooms r ON r.room\_id = ch.room\_id

JOIN hotel.housing h ON h.housing\_id = r.housing\_id

WHERE NOT (EXISTS ( SELECT booking\_composition.room\_id

FROM hotel.booking\_composition

WHERE bc.room\_id = ch.room\_id AND (CURRENT\_DATE < bc.start\_of\_booking OR CURRENT\_DATE > bc.finish\_of\_booking))) AND NOT (EXISTS ( SELECT 1

FROM hotel.check\_in

WHERE ch.room\_id = bc.room\_id AND CURRENT\_DATE >= ch.arrival\_date AND CURRENT\_DATE <= ch.departure\_date))

ORDER BY h.name, r.room\_number;

ALTER TABLE hotel.free\_rooms

OWNER TO postgres;

CREATE OR REPLACE VIEW hotel.list\_of\_services

AS

SELECT ts.name AS service,

ts.price,

hou.name AS house

FROM hotel.types\_of\_services ts

JOIN hotel.availability\_of\_service ava ON ts.service\_id = ava.service\_id

JOIN hotel.housing hou ON hou.housing\_id = ava.housing\_id

WHERE ava.housing\_id = ANY (ARRAY[1, 2])

ORDER BY hou.housing\_id;

ALTER TABLE hotel.list\_of\_services

OWNER TO postgres;

CREATE OR REPLACE FUNCTION hotel.zakazy(

ch\_in\_id integer)

RETURNS TABLE(order\_id integer, housing\_name text, num\_of\_room integer, flname text, name\_service text, serv\_quantity integer, payment\_stat boolean)

LANGUAGE 'plpgsql'

COST 100

VOLATILE PARALLEL UNSAFE

ROWS 1000

AS $BODY$

begin

return query

select ord.order\_id, h.name, r.room\_number, first\_last\_name, ts.name, ord.quantity, payment\_status

from types\_of\_services ts

inner join orders ord on ts.service\_id = ord.service\_id

inner join check\_in ch on ch.check\_in\_id = ord.check\_in\_id

inner join person pe on ch.customer\_id = pe.customer\_id

inner join rooms r on ch.room\_id = r.room\_id

inner join housing h on r.housing\_id = h.housing\_id

where ch\_in\_id = ch.check\_in\_id

order by payment\_status;

end;

$BODY$;

ALTER FUNCTION hotel.zakazy(integer)

OWNER TO postgres;

CREATE OR REPLACE PROCEDURE hotel.dolg(

IN fio text,

OUT phone text,

OUT summa integer)

LANGUAGE 'plpgsql'

AS $BODY$

declare

begin

select per.phone\_number, sum(price \* quantity) into phone, summa

from person per

inner join check\_in ch

on ch.customer\_id = per.customer\_id

inner join orders ord

on ch.check\_in\_id = ord.check\_in\_id

inner join types\_of\_services tp

on tp.service\_id = ord.service\_id

where (current\_date between arrival\_date and departure\_date) and payment\_status = 'False'

and ch.customer\_id = (select customer\_id from person where first\_last\_name = fio)

group by per.phone\_number, ch.customer\_id;

end;

$BODY$;

ALTER PROCEDURE hotel.dolg(text)

OWNER TO postgres;

CREATE OR REPLACE PROCEDURE hotel.ord(

IN flname text,

IN service integer,

IN status boolean,

IN amount integer)

LANGUAGE 'plpgsql'

AS $BODY$

declare

per int;

ret int;

serv int;

begin

select max(order\_id) from orders into per;

select service\_id from orders ord into serv

inner join types\_of\_of\_services tos

on ord.service\_id = tos.service\_id

where tos.name = service;

select check\_in\_id from check\_in ch into ret

inner join person pe on ch.customer\_id = pe.customer\_id

where flname = first\_last\_name

and

current\_date between arrival\_date and departure\_date;

insert into orders(order\_id, check\_in\_id, service\_id, payment\_status, quantity)

values(per+1, ret , serv, status, amount);

end;

$BODY$;

ALTER PROCEDURE hotel.ord(text, integer, boolean, integer)

OWNER TO postgres;

CREATE OR REPLACE PROCEDURE hotel.dele(

IN flname text,

IN service\_name text)

LANGUAGE 'plpgsql'

AS $BODY$

declare

fo int;

up int;

begin

select ch.customer\_id into fo from person per

inner join check\_in ch

on per.customer\_id = ch.customer\_id

where per.first\_last\_name = flname and current\_date between ch.arrival\_date and ch.departure\_date;

select order\_id into up from orders ord

inner join types\_of\_services ts

on ts.service\_id = ord.service\_id

inner join check\_in ch

on ch.check\_in\_id = ord.check\_in\_id

where ch.customer\_id = fo and ts.name = service\_name

order by order\_id desc

limit 1;

delete from orders

where order\_id = up;

end;

$BODY$;

ALTER PROCEDURE hotel.dele(text, text)

OWNER TO postgres;

# ПРИЛОЖЕНИЕ Б

**Код ТГ-бота**

import telebot

from telebot import types

import sql\_query

from re import split

from datetime import datetime

sq = sql\_query.QueryTool()

auto\_log = 'admin'

auto\_pas = '0000'

logged\_in = False

to\_switch = []

to\_del = []

bot = telebot.TeleBot('6031519919:AAFO0mD8GDuD3L8i9FFa1n9qJaNWMkfXP2E')

# ========================================= Start =========================================

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

def start\_bot(message):

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} запустил бота')

global to\_switch, to\_del

to\_switch.clear()

to\_del.clear()

rem\_key = types.ReplyKeyboardRemove()

bot.send\_message(message.from\_user.id,

"Приветствую!\nЯ – Ваш персональный помощник для работы с базой данных отеля \_'Jewelry Nature'\_",

reply\_markup=rem\_key,

parse\_mode='Markdown')

markup = types.InlineKeyboardMarkup()

begin\_btn = types.InlineKeyboardButton(text='Авторизация', callback\_data='begin')

markup.add(begin\_btn)

mes = bot.send\_message(message.from\_user.id,

'Для начала работы с ботом необходимо авторизироваться.',

reply\_markup=markup)

to\_switch = [mes]

# ========================================= Connecting =========================================

@bot.callback\_query\_handler(func=lambda call: call.data == 'begin')

def begin\_callback(call):

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} подключается к базе данных...')

if not sq.open\_connection():

markup = types.InlineKeyboardMarkup()

begin\_btn = types.InlineKeyboardButton(text='Повторить попытку', callback\_data='begin')

markup.add(begin\_btn)

bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=call.message.message\_id,

text='Не удалось подключиться к базе данных.',

reply\_markup=markup,

parse\_mode='Markdown')

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {call.from\_user.id} не смог подключиться к базе данных')

else:

begin\_callback(call)

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} успешно подключился к базе данных')

# ========================================= Log in =========================================

@bot.callback\_query\_handler(func=lambda call: call.data == 'begin')

def begin\_callback(call):

mes = bot.send\_message(call.from\_user.id,

'Введите свой логин:',

parse\_mode='html')

to\_del.append(mes.message\_id)

bot.register\_next\_step\_handler(mes, get\_login)

def get\_login(message):

user\_login = message.text

to\_del.append(message.message\_id)

del\_mes(message)

mes = bot.send\_message(message.from\_user.id,

'Введите свой пароль:')

to\_del.append(mes.message\_id)

bot.register\_next\_step\_handler(mes, get\_pass, user\_login)

def get\_pass(message, user\_login):

global logged\_in

user\_password = message.text

to\_del.append(message.message\_id)

del\_mes(message)

if auto\_log == user\_login and auto\_pas == user\_password:

logged\_in = True

markup = types.InlineKeyboardMarkup()

btn1 = types.InlineKeyboardButton('Начать работу', callback\_data='main\_menu')

markup.row(btn1)

bot.edit\_message\_text(

chat\_id=message.from\_user.id,

message\_id=to\_switch[0].message\_id,

text='Вы были успешно авторизованы!\nДавайте приступим к работе.',

reply\_markup=markup)

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} успешно авторизовался.\nВведенный логин: {user\_login}\nВведенный пароль: {user\_password}')

else:

markup = types.InlineKeyboardMarkup()

begin\_btn = types.InlineKeyboardButton(text='Повторить попытку', callback\_data='begin')

markup.row(begin\_btn)

mes = bot.send\_message(message.from\_user.id,

'Ошибка авторизации. Неверный логин или пароль.',

reply\_markup=markup)

to\_del.append(mes.message\_id)

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {message.chat.id} не смог авторизироваться.\nВведенный логин: {user\_login}\nВведенный пароль: {user\_password}')

# ========================================= Main Menu =========================================

@bot.message\_handler(commands=['menu'], func=lambda call: logged\_in)

@bot.callback\_query\_handler(func=lambda callback: callback.data == 'main\_menu' and logged\_in)

def main\_menu(call):

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} перешел в главное меню')

markup = types.InlineKeyboardMarkup()

btn1 = types.InlineKeyboardButton('Регистрация', callback\_data='reg\_mm')

btn2 = types.InlineKeyboardButton('Бронирование', callback\_data='bk\_mm')

btn3 = types.InlineKeyboardButton('Заселение', callback\_data='ch\_mm')

btn4 = types.InlineKeyboardButton('Заказы', callback\_data='ord\_mm')

btn5 = types.InlineKeyboardButton('Завершить работу', callback\_data='end\_mm')

markup.row(btn1, btn2).row(btn3, btn4).row(btn5)

mm\_mes = '\*Главное меню\*\n\nВыберите необходимый раздел.'

if type(call) is types.CallbackQuery:

bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=call.message.message\_id,

text=mm\_mes,

reply\_markup=markup,

parse\_mode='Markdown')

else:

bot.send\_message(call.from\_user.id,

mm\_mes,

reply\_markup=markup,

parse\_mode='Markdown')

# ========================================= Func4All =========================================

def gen\_markup(back):

markup = types.InlineKeyboardMarkup()

btn1 = types.InlineKeyboardButton('В главное меню', callback\_data='main\_menu')

btn2 = types.InlineKeyboardButton('<< Назад', callback\_data=back)

markup.row(btn2, btn1)

return markup

def del\_mes(message):

for i in range(len(to\_del) - 1, -1, -1):

bot.delete\_message(message.from\_user.id, to\_del[i])

to\_del.clear()

def divide\_str(text):

div = split(", |\n|,", text)

if len(div) > 1:

return div

else:

return None

# ========================================= Register =========================================

@bot.callback\_query\_handler(func=lambda callback: callback.data.startswith('reg') and logged\_in)

def reg\_handler(call):

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} перешел в меню Регстрация')

global to\_switch

if call.data == 'reg\_mm':

markup = types.InlineKeyboardMarkup()

btn1 = types.InlineKeyboardButton('Добавить профиль клиента', callback\_data='reg\_add\_per')

btn2 = types.InlineKeyboardButton('Добавить профиль компании', callback\_data='reg\_add\_comp')

btn3 = types.InlineKeyboardButton('<< Назад', callback\_data='main\_menu')

markup.row(btn1).row(btn2).row(btn3)

mes = bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=call.message.message\_id,

text='\*Регистрация\*\n\nВыберите необходимое действие.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch = [mes]

elif call.data == 'reg\_add\_per':

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} инициировал регистрацию клиента')

mes = bot.send\_message(

call.from\_user.id,

'Введите данные через запятую или каждое с новой строки\nДанные для ввода:\n\nИмя и фамилия\nНомер паспорта\nТелефонный номер\nЭлектранная почта')

to\_switch.append(mes)

bot.register\_next\_step\_handler(mes, reg\_per\_in)

elif call.data == 'reg\_add\_comp':

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} инициировал регистрацию компании')

mes = bot.send\_message(

call.from\_user.id,

'Введите данные через запятую или каждое с новой строки\nДанные для ввода:\n\nНазвание компании\nФИО контактного лица\nИНН\nТелефонный номер\nЭлектранная почта')

to\_switch.append(mes)

bot.register\_next\_step\_handler(mes, reg\_comp\_in)

def reg\_per\_in(message):

user\_in = message.text

to\_del.append(message.message\_id)

markup = gen\_markup('reg\_mm')

divided = divide\_str(user\_in)

if divided is not None and len(divided) == 4:

flname, pas, phone, mail = (divided[i] for i in range(0, len(divided)))

if sq.reg\_per(flname, pas, phone, mail):

del\_mes(message)

bot.edit\_message\_text(

chat\_id=message.from\_user.id,

message\_id=to\_switch[1].message\_id,

text=f'\*Внесенные данные\*\nИмя и фамилия: {flname}\nПаспорт: {pas}\nТелефонный номер: {phone}\nПочта: {mail}',

parse\_mode='Markdown')

bot.send\_message(

message.from\_user.id,

'Данные были успешно внесены в базу данных!',

reply\_markup=markup,

parse\_mode='Markdown'

)

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} успешно добавил нового клиента\nВнесенные данные\nИмя и фамилия: {flname}\nПаспорт: {pas}\nТелефонный номер: {phone}\nПочта: {mail}')

else:

bot.send\_message(

message.from\_user.id,

'При внесении данных возникла ошибка.',

reply\_markup=markup,

parse\_mode='Markdown')

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {message.chat.id} не смог зарегистрировать пользователя\nВнесенные данные\nИмя и фамилия: {flname}\nПаспорт: {pas}\nТелефонный номер: {phone}\nПочта: {mail}')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} не смог зарегистрировать пользователя\nНеверный ввод данных: {user\_in}')

bot.send\_message(

message.from\_user.id,

'Неверный ввод данных, повторите попытку.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch.clear()

to\_del.clear()

def reg\_comp\_in(message):

user\_in = message.text

to\_del.append(message.message\_id)

markup = gen\_markup('reg\_mm')

divided = divide\_str(user\_in)

if divided is not None and len(divided) == 5:

comp\_name, contact\_name, itn, phone, mail = (divided[i] for i in range(0, len(divided)))

if sq.reg\_comp(comp\_name, contact\_name, itn, phone, mail):

del\_mes(message)

bot.edit\_message\_text(

chat\_id=message.from\_user.id,

message\_id=to\_switch[1].message\_id,

text=f'\*Внесенные данные\*\nНазвание компании: {comp\_name}\nФИО контактного лица: {contact\_name}\nИНН: {itn}\nТелефонный номер: {phone}\nЭлектранная почта: {mail}',

parse\_mode='Markdown')

bot.send\_message(

message.from\_user.id,

'Данные были успешно внесены в базу данных!',

reply\_markup=markup,

parse\_mode='Markdown'

)

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} успешно добавил новую компанию\nВнесенные данные\nНазвание компании: {comp\_name}\nФИО контактного лица: {contact\_name}\nИНН: {itn}\nТелефонный номер: {phone}\nЭлектранная почта: {mail}')

else:

bot.send\_message(

message.from\_user.id,

'При внесении данных возникла ошибка.',

reply\_markup=markup,

parse\_mode='Markdown')

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {message.chat.id} не смог зарегистрировать компанию\nВнесенные данные\nНазвание компании: {comp\_name}\nФИО контактного лица: {contact\_name}\nИНН: {itn}\nТелефонный номер: {phone}\nЭлектранная почта: {mail}')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} не смог зарегистрировать компанию\nНеверный ввод данных: {user\_in}')

bot.send\_message(

message.from\_user.id,

'Неверный ввод данных, повторите попытку.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch.clear()

to\_del.clear()

# ========================================= Booking =========================================

@bot.callback\_query\_handler(func=lambda callback: callback.data.startswith('bk') and logged\_in)

def bk\_handler(call):

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} перешел в меню Бронирование')

global to\_switch

if call.data == 'bk\_mm':

markup = types.InlineKeyboardMarkup()

btn1 = types.InlineKeyboardButton('Список активной брони', callback\_data='bk\_act')

btn2 = types.InlineKeyboardButton('Отменить бронь', callback\_data='bk\_reject')

btn3 = types.InlineKeyboardButton('<< Назад', callback\_data='main\_menu')

markup.row(btn1).row(btn2).row(btn3)

mes = bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=call.message.message\_id,

text='\*Бронирование\*\n\nВыберите необходимое действие.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch = [mes]

elif call.data == 'bk\_act':

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} вывел список активной брони')

bk\_active = sq.bk\_active()

markup = gen\_markup('bk\_mm')

if bk\_active is not None:

out = ''

corp = None

for i in range(0, len(bk\_active)):

if bk\_active[i][0] == '':

if corp is None:

corp = bk\_active[i][1]

out += (f'\nКомпания: {corp}\nДаты брони: %s – %s\nПредоплата за комнаты: ${bk\_active[i][2]}',

bk\_active[i][3].replace('-', '.'), bk\_active[i][4].replace('-', '.'))

continue

if corp == bk\_active[i][1]:

out += f', ${bk\_active[i][2]}'

else:

corp = bk\_active[i][1]

out += (f'\nКомпания: {corp}\nДаты брони: %s – %s\nПредоплата за комнаты: ${bk\_active[i][2]}',

bk\_active[i][3].replace('-', '.'), bk\_active[i][4].replace('-', '.'))

else:

out += (f'\nИмя и фамилия: {bk\_active[i][0]}\nДата брони: %s – %s\nПредоплата за комнату: ${bk\_active[i][2]}',

bk\_active[i][3].replace('-', '.'), bk\_active[i][4].replace('-', '.'))

bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=to\_switch[0].message\_id,

text=f'\*Список активной брони\*\n{out}',

reply\_markup=markup,

parse\_mode='Markdown')

else:

print(f'ВНИМАНИЕ! \nПользователь {call.from\_user.id} не смог вывести список активной брони')

bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=to\_switch[0].message\_id,

text='При выводе данных произошла ошибка.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch.clear()

elif call.data == 'bk\_reject':

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} инициировал отмену брони')

mes = bot.send\_message(

call.from\_user.id,

'Введите данные через запятую или каждое с новой строки\nДанные для ввода:\n\nТип пользователя (Компания / Клиент)\nИНН / Имя и фамилия\nДата начала брони (В формате год-месяц-день)')

to\_switch.append(mes)

bot.register\_next\_step\_handler(mes, bk\_rej\_in)

def bk\_rej\_in(message):

user\_in = message.text

to\_del.append(message.message\_id)

markup = gen\_markup('bk\_mm')

divided = divide\_str(user\_in)

if divided is not None and len(divided) == 3:

user\_type, name, date = (divided[i] for i in range(0, len(divided)))

if sq.bk\_change(user\_type, name, date):

del\_mes(message)

if name.isdigit():

info\_type = 'ИНН'

else:

info\_type = 'Имя и фамилия'

bot.edit\_message\_text(

chat\_id=message.from\_user.id,

message\_id=to\_switch[1].message\_id,

text=f'\*Удаление брони\*\nТип пользователя: {user\_type}\n{info\_type}: {name}\nДата начала брони: {date}',

parse\_mode='Markdown')

bot.send\_message(

message.from\_user.id,

'Бронь была успешно удалена!',

reply\_markup=markup,

parse\_mode='Markdown'

)

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} успешно удалил бронь.\nУдаленные данные\nТип пользователя: {user\_type}\n{info\_type}: {name}\nДата начала брони: {date}')

else:

bot.send\_message(

message.from\_user.id,

'При удалении возникла ошибка.',

reply\_markup=markup,

parse\_mode='Markdown')

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {message.chat.id} не смог удалить бронь\nВведенные данные\nТип пользователя: {user\_type}\nИдентификатор: {name}\nДата начала брони: {date}')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {message.chat.id} неверно ввел данные')

bot.send\_message(

message.from\_user.id,

'Неверный ввод данных, повторите попытку.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch.clear()

to\_del.clear()

# ========================================= Check in =========================================

@bot.callback\_query\_handler(func=lambda callback: callback.data.startswith('ch') and logged\_in)

def ch\_handler(call):

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} перешел в меню Заселение')

global to\_switch

if call.data == 'ch\_mm':

markup = types.InlineKeyboardMarkup()

btn1 = types.InlineKeyboardButton('Сегодняшнее заселение', callback\_data='ch\_today')

btn2 = types.InlineKeyboardButton('Свободные комнаты', callback\_data='ch\_free')

btn3 = types.InlineKeyboardButton('Список услуг', callback\_data='ch\_serv')

btn4 = types.InlineKeyboardButton('<< Назад', callback\_data='main\_menu')

markup.row(btn1).row(btn2, btn3).row(btn4)

mes = bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=call.message.message\_id,

text='\*Заселение\*\n\nВыберите необходимое действие.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch = [mes]

elif call.data == 'ch\_today':

ch\_today = sq.ch\_today()

markup = gen\_markup('ch\_mm')

if ch\_today is not None:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} вывел список сегодняшего заселения')

out = ''

for i in range(0, len(ch\_today)):

out += f'Имя и фамилия: {ch\_today[i][0]}\nТелефон: {ch\_today[i][1]}\nКорпус и комната: {ch\_today[i][3]}, {ch\_today[i][2]}\n\n'

bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=to\_switch[0].message\_id,

text=f'\*Список сегодняшнего заселения\*\n\n{out}',

reply\_markup=markup,

parse\_mode='Markdown')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {call.from\_user.id} не смог вывести список сегодняшнего заселения')

bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=to\_switch[0].message\_id,

text='При выводе данных произошла ошибка.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch.clear()

elif call.data == 'ch\_free':

ch\_today = sq.ch\_free()

markup = gen\_markup('ch\_mm')

if ch\_today is not None:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} вывел список свободных комнат')

housing = ch\_today[0][1]

out = f'Корпус: {ch\_today[0][1]}\nКомнаты: {ch\_today[0][0]}'

for i in range(1, len(ch\_today)):

if ch\_today[i][1] == housing:

out += f', {ch\_today[i][0]}'

else:

housing = ch\_today[i][1]

out += f'\n\nКорпус: {ch\_today[i][1]}\nКомнаты: {ch\_today[i][0]}'

bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=to\_switch[0].message\_id,

text=f'\*Свободные для заселения комнаты\*\n\n{out}',

reply\_markup=markup,

parse\_mode='Markdown')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {call.from\_user.id} не смог вывести список свободных комнат')

bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=to\_switch[0].message\_id,

text='При выводе данных произошла ошибка.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch.clear()

elif call.data == 'ch\_serv':

ch\_serv = sq.ch\_serv()

markup = gen\_markup('ch\_mm')

if ch\_serv is not None:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} вывел список услуг')

housing = ch\_serv[0][2]

out = f'Корпус: {ch\_serv[0][2]}\nДоступный сервис, цена:\n'

for i in range(0, len(ch\_serv)):

if ch\_serv[i][2] == housing:

out += f'{ch\_serv[i][0]}, ${ch\_serv[i][1]}\n'

else:

housing = ch\_serv[i][2]

out += f'\nКорпус: {ch\_serv[i][2]}\nДоступный сервис, цена:\n'

out += f'{ch\_serv[i][0]}, ${ch\_serv[i][1]}\n'

bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=to\_switch[0].message\_id,

text=f'\*Список доступных услуг\*\n\n{out}',

reply\_markup=markup,

parse\_mode='Markdown')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {call.from\_user.id} не смог вывести список услуг')

bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=to\_switch[0].message\_id,

text='При выводе данных произошла ошибка.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch.clear()

# ========================================= Orders =========================================

@bot.callback\_query\_handler(func=lambda callback: callback.data.startswith('ord') and logged\_in)

def ord\_handler(call):

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} перешел в меню Заказы')

global to\_switch

if call.data == 'ord\_mm':

markup = types.InlineKeyboardMarkup()

btn1 = types.InlineKeyboardButton('Просмотр заказов', callback\_data='ord\_zak')

btn2 = types.InlineKeyboardButton('Задолженность', callback\_data='ord\_duty')

btn3 = types.InlineKeyboardButton('Добавить заказ', callback\_data='ord\_add')

btn4 = types.InlineKeyboardButton('Удалить заказ', callback\_data='ord\_del')

btn5 = types.InlineKeyboardButton('<< Назад', callback\_data='main\_menu')

markup.row(btn1, btn2).row(btn3, btn4).row(btn5)

mes = bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=call.message.message\_id,

text='\*Заказы\*\n\nВыберите необходимое действие.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch = [mes]

elif call.data == 'ord\_zak':

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} инициировал просмотр заказов')

mes = bot.send\_message(

call.from\_user.id,

'Введите ID заселения, по которому необходимо просмотреть заказы:')

to\_switch.append(mes)

bot.register\_next\_step\_handler(mes, ord\_zak\_in)

elif call.data == 'ord\_duty':

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} инициировал просмотр задолженностей клиента')

mes = bot.send\_message(

call.from\_user.id,

'Введите имя и фамилию человека, задолженность которого необходимо узнать:')

to\_switch.append(mes)

bot.register\_next\_step\_handler(mes, ord\_duty\_in)

elif call.data == 'ord\_add':

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} инициировал добавление заказа')

mes = bot.send\_message(

call.from\_user.id,

'Введите данные через запятую или каждое с новой строки\nДанные для ввода:\n\nИмя и фамилия\nНазвание услуги\nКоличетсво\nСтатус оплаты (Оплачено / Не оплачено)')

to\_switch.append(mes)

bot.register\_next\_step\_handler(mes, ord\_add\_in)

elif call.data == 'ord\_del':

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.from\_user.id} инициировал удаление заказа')

mes = bot.send\_message(

call.from\_user.id,

'Введите данные через запятую или каждое с новой строки\nДанные для ввода:\n\nИмя и фамилия\nНазвание услуги')

to\_switch.append(mes)

bot.register\_next\_step\_handler(mes, ord\_add\_in)

def ord\_zak\_in(message):

user\_in = message.text

to\_del.append(message.message\_id)

markup = gen\_markup('ord\_mm')

if user\_in.isdigit():

ord\_zak = sq.ord\_zak(user\_in)

if ord\_zak is not None:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} вывел список заказов по ID заселения: {user\_in}')

del\_mes(message)

if ord\_zak == []:

out = 'Клиент не совершал заказов'

else:

out = f'ID заселения: {user\_in}\nИмя клиента: {ord\_zak[0][3]}\nКорпус, комната: {ord\_zak[0][1]}, {ord\_zak[0][2]}\nЗаказы, количество и статус оплаты: '

for i in range(0, len(ord\_zak)):

p\_stat = ('Оплачено' if ord\_zak[i][6] else 'Не оплачено')

out += f'{ord\_zak[i][4]} {ord\_zak[i][5]}, {p\_stat}\n'

bot.edit\_message\_text(

chat\_id=message.from\_user.id,

message\_id=to\_switch[1].message\_id,

text=f'\*Список заказов\*\n{out}',

parse\_mode='Markdown')

bot.send\_message(

message.from\_user.id,

'Список заказов был успешно выведен!',

reply\_markup=markup,

parse\_mode='Markdown')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {message.chat.id} не смог вывести список заказов по ID заселения: {user\_in}')

bot.send\_message(

message.from\_user.id,

'При выводе данных возникла ошибка.',

reply\_markup=markup,

parse\_mode='Markdown')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} неверно ввел данные: {user\_in}')

bot.send\_message(

message.from\_user.id,

'Неверный ввод данных, повторите попытку.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch.clear()

to\_del.clear()

def ord\_duty\_in(message):

user\_in = message.text

to\_del.append(message.message\_id)

markup = gen\_markup('ord\_mm')

if len(user\_in.split(' ')) == 2:

ord\_duty = sq.ord\_duty(user\_in)

if ord\_duty is not None:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} успешно вывел задолженности клиента: {user\_in}')

del\_mes(message)

out = f'Имя клиента: {user\_in}\nТелефон клиента: {ord\_duty[0][0]}\nСумма долга: ${ord\_duty[0][1]}'

bot.edit\_message\_text(

chat\_id=message.from\_user.id,

message\_id=to\_switch[1].message\_id,

text=f'\*Задолженность клиента\*\n{out}',

parse\_mode='Markdown')

bot.send\_message(

message.from\_user.id,

'Задолженность клиента была успешно выведена!',

reply\_markup=markup,

parse\_mode='Markdown')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {message.chat.id} не смог вывести задолженность клиента: {user\_in}')

bot.send\_message(

message.from\_user.id,

'При выводе данных возникла ошибка.',

reply\_markup=markup,

parse\_mode='Markdown')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} неверно ввел данные: {user\_in}')

bot.send\_message(

message.from\_user.id,

'Неверный ввод данных, повторите попытку.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch.clear()

to\_del.clear()

def ord\_add\_in(message):

user\_in = message.text

to\_del.append(message.message\_id)

markup = gen\_markup('ord\_mm')

divided = divide\_str(user\_in)

if divided is not None and len(divided) == 4:

flname, serv, quantity, stat = (divided[i] for i in range(0, len(divided)))

if sq.ord\_add(flname, serv, (1 if stat == 'Оплачено' or stat == 'оплачено' else 0), quantity):

del\_mes(message)

out = f'Имя клиента: {flname}\nНазвание услуги, количество и статус оплаты: {serv}, {quantity}, {stat}'

bot.edit\_message\_text(

chat\_id=message.from\_user.id,

message\_id=to\_switch[1].message\_id,

text=f'\*Добавление заказа\*\n{out}',

parse\_mode='Markdown')

bot.send\_message(

message.from\_user.id,

'Заказ был успешно добавлен!',

reply\_markup=markup,

parse\_mode='Markdown')

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} успешно добавил заказ.\nВнесенные данные\nИмя клиента: {flname}\nНазвание услуги, количество и статус оплаты: {serv}, {quantity}, {stat}')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {message.chat.id} не смог добавить заказ. Введенные данные: {user\_in}')

bot.send\_message(

message.from\_user.id,

'При выводе данных возникла ошибка.',

reply\_markup=markup,

parse\_mode='Markdown')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} неверно ввел данные: {user\_in}')

bot.send\_message(

message.from\_user.id,

'Неверный ввод данных, повторите попытку.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch.clear()

to\_del.clear()

def ord\_del\_in(message):

user\_in = message.text

to\_del.append(message.message\_id)

markup = gen\_markup('ord\_mm')

divided = divide\_str(user\_in)

if divided is not None and len(divided) == 2:

flname, serv = (divided[i] for i in range(0, len(divided)))

if sq.ord\_del(flname, serv):

del\_mes(message)

out = f'Имя клиента: {flname}\nНазвание услуги {serv}'

bot.edit\_message\_text(

chat\_id=message.from\_user.id,

message\_id=to\_switch[1].message\_id,

text=f'\*Удаление последнего заказа\*\n{out}',

parse\_mode='Markdown')

bot.send\_message(

message.from\_user.id,

'Последний заказ пользователя был удален!',

reply\_markup=markup,

parse\_mode='Markdown')

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} успешно удалил последний заказ: {user\_in}')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nВНИМАНИЕ! Пользователь {message.chat.id} не смог удалить последний заказ: {user\_in}')

bot.send\_message(

message.from\_user.id,

'При выводе данных возникла ошибка.',

reply\_markup=markup,

parse\_mode='Markdown')

else:

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} неверно ввел данные: {user\_in}')

bot.send\_message(

message.from\_user.id,

'Неверный ввод данных, повторите попытку.',

reply\_markup=markup,

parse\_mode='Markdown')

to\_switch.clear()

to\_del.clear()

# ========================================= End =========================================

@bot.message\_handler(commands=['exit'], func=lambda call: logged\_in)

@bot.callback\_query\_handler(func=lambda callback: callback.data == 'end\_mm' and logged\_in)

def end\_handler(call):

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {call.user\_in.id} закончил работу')

global logged\_in

end\_mes = '\*Завершение работы.\*\n\nХорошо поработали сегодня! Соединение с базой данных было успешно разорвано. До новых встреч.'

if type(call) is types.CallbackQuery:

bot.edit\_message\_text(

chat\_id=call.from\_user.id,

message\_id=call.message.message\_id,

text=end\_mes,

parse\_mode='Markdown')

else:

bot.send\_message(call.from\_user.id,

end\_mes,

parse\_mode='Markdown')

sq.close\_connection()

logged\_in = False

# ========================================= Not logged =========================================

@bot.message\_handler(commands=['menu', 'exit'], func=lambda call: not logged\_in)

def not\_logged(message):

print(f'\n[{datetime.now().replace(microsecond=0)}]\nПользователь {message.chat.id} попытался выполнить команду без авторизации')

bot.send\_message(message.from\_user.id,

'Невозможно выполнить команду. Вы еще не авторизованы. Для авторизации воспользуйтесь командой /start',

parse\_mode='Markdown')

# ========================================= Start up bot =========================================

if \_\_name\_\_ == "\_\_main\_\_":

bot.infinity\_polling()

# ПРИЛОЖЕНИЕ В

**Код для взаимодействия бота с БД**

import psycopg2

class QueryTool:

def \_\_init\_\_(self):

self.\_conn = None

self.\_cursor = None

def open\_connection(self):

try:

self.\_conn = psycopg2.connect(dbname='hotel', user='postgres', password='123234', host='localhost')

self.\_cursor = self.\_conn.cursor()

except Exception:

return False

else:

return True

def reg\_per(self, flname, pas, phone, mail):

try:

self.\_cursor.execute('SET search\_path TO hotel')

self.\_cursor.execute('CALL insert\_person(\'%s\', \'%s\', \'%s\', \'%s\')' % (flname, pas, phone, mail))

self.\_conn.commit()

except Exception:

return False

else:

return True

def reg\_comp(self, comp\_name, contact\_name, itn, phone, mail):

try:

self.\_cursor.execute('SET search\_path TO hotel')

self.\_cursor.execute('CALL insert\_company(\'%s\', \'%s\', \'%s\', \'%s\', \'%s\')' % (comp\_name, contact\_name, itn, phone, mail))

self.\_conn.commit()

except Exception:

return False

else:

return True

def bk\_active(self):

try:

self.\_cursor.execute('SET search\_path TO hotel')

self.\_cursor.execute('SELECT \* FROM active\_bookings')

except Exception:

return None

else:

return self.\_cursor.fetchall()

def bk\_change(self, type\_of\_client, name, date):

try:

self.\_cursor.execute('SET search\_path TO hotel')

self.\_cursor.execute('CALL change(\'%s\', \'%s\', date \'%s\')' % (type\_of\_client, name, date))

self.\_conn.commit()

except Exception:

return False

else:

return True

def ch\_today(self):

try:

self.\_cursor.execute('SET search\_path TO hotel')

self.\_cursor.execute('SELECT \* FROM today\_check\_in')

except Exception:

return None

else:

return self.\_cursor.fetchall()

def ch\_free(self):

try:

self.\_cursor.execute('SET search\_path TO hotel')

self.\_cursor.execute('SELECT \* FROM free\_rooms')

except Exception:

return None

else:

return self.\_cursor.fetchall()

def ch\_serv(self):

try:

self.\_cursor.execute('SET search\_path TO hotel')

self.\_cursor.execute('SELECT \* FROM list\_of\_services')

except Exception:

return None

else:

return self.\_cursor.fetchall()

def ord\_zak(self, ch\_id):

try:

self.\_cursor.execute('SET search\_path TO hotel')

self.\_cursor.execute('SELECT \* FROM zakazy(\'%s\')' % (ch\_id))

except Exception:

return None

else:

return self.\_cursor.fetchall()

def ord\_duty(self, ch\_id):

try:

self.\_cursor.execute('SET search\_path TO hotel')

self.\_cursor.execute('CALL dolg(\'%s\', NULL, NULL)' % (ch\_id))

except Exception:

return None

else:

return self.\_cursor.fetchall()

def ord\_add(self, flname, serv, stat, quantity):

try:

self.\_cursor.execute('SET search\_path TO hotel')

self.\_cursor.execute('CALL ord(\'%s\', \'%s\', \'%s\')' % (flname, serv, stat, quantity))

self.\_conn.commit()

except Exception:

return False

else:

return True

def ord\_del(self, flname, serv):

try:

self.\_cursor.execute('SET search\_path TO hotel')

self.\_cursor.execute('CALL dele(\'%s\', \'%s\')' % (flname, serv))

self.\_conn.commit()

except Exception:

return False

else:

return True

def close\_connection(self):

self.\_cursor.close()

self.\_conn.close()