

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 фундаментального образования

Кафедра «Научно-учебный центр информационной безопасности»

Оценка

Руководитель курсового

проектирования Мирвода С.Г.

Члены комиссии

Дата защиты

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе**

**по теме: Информационная система для просмотра расписания студентами и преподавателями университета**

**по дисциплине: Языки и методы программирования**

Студент: Латников Вячеслав Николаевич

Малявко Евгения Сергеевна

Потапова Марина Витальевна

(Подпись)

Группа: РИ-311055

Екатеринбург

2024

Министерство образования и науки Российской Федерации

ФГАОУ ВО «УрФУ имени первого Президента России Б.Н.Ельцина»

**РЕЦЕНЗИЯ**

на курсовую работу (проект)

Студента Латникова Вячеслава Николаевича, Малявко Евгении Сергеевны, Потаповой Марины Витальевны группы РИ-311055

(фамилия имя отчество)

Тема курсовой работы: Информационная система для просмотра расписания студентами и преподавателями университета

Модуль/дисциплина: Языки и методы программирования

1 Соответствие результатов выполнения работы целям и задачам курсового проектирования, результатам обучения по дисциплине/модулю\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 Оригинальность и самостоятельность выполнения работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3 Полнота и глубина проработки разделов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4 Общая грамотность и качество оформления текстового документа и графических материалов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5 Вопросы и замечания \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6 Общая оценка работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Сведения о рецензенте:

Ф.И.О. \_\_\_\_\_\_

Должность \_\_\_\_

Место работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Уч. звание \_\_\_\_\_ Уч. степень \_\_\_\_\_

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата

СОДЕРЖАНИЕ

[ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 4](#_Toc155905868)

[ВВЕДЕНИЕ 5](#_Toc155905869)

[1 ПРОЕКТИРОВАНИЕ 6](#_Toc155905870)

[1.1 Описание предметной области 6](#_Toc155905871)

[1.2 Создание функциональной диаграммы 7](#_Toc155905872)

[1.3 Описание архитектуры продукта 8](#_Toc155905873)

[1.4 Функциональная диаграмма UML 9](#_Toc155905874)

[1.5 Диаграмма последовательности UML 10](#_Toc155905875)

[1.6 Концептуальная модель БД 11](#_Toc155905876)

[1.7 Требования к системному ПО и СУБД 12](#_Toc155905877)

[2 РАЗРАБОТКА 13](#_Toc155905878)

[2.1 Диаграмма пакетов UML 14](#_Toc155905879)

[2.2 Диаграмма физической модели (ER-диаграмма) 15](#_Toc155905880)

[ЗАКЛЮЧЕНИЕ 16](#_Toc155905881)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 17](#_Toc155905882)

[ПРИЛОЖЕНИЕ А 18](#_Toc155905883)

# ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

База данных (БД) – некая именованная совокупность данных, отражающая состояние объектов или явлений в некоторой выбранной предметной области и связи между ними. Она создаётся для хранения данных и доступа к ним.

Система управления базами данных (СУБД) – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

ER-модель (англ. entity-relationship model) – физическая модель данных, позволяющая описывать концептуальные схемы предметной области. С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями [3].

Унифицированный язык моделирования (UML) – это семейство графических нотаций, в основе которого лежит единая метамодель, помогающих в описании и проектировании программных систем, в особенности систем, построенных с использованием объектно–ориентированных (ОО) технологий.

# ВВЕДЕНИЕ

Одним из важных аспектов процесса обучения является создание удобных и эффективных инструментов для взаимодействия между его участниками и оптимизации учебного процесса. Просмотр занятий студентами и преподавателями является одним из таких инструментов.

**Цель проекта:** Целью данного курсового проекта является разработка и создание телеграмм-бота для студентов и преподавателей университета для комфортного просмотра расписания.

**Видение проекта:** Данный инструмент обеспечит быстрый доступ к информации. С помощью бота пользователи смогут видеть кол-во пар в определенный день, преподавателя/группу, аудиторию, время проведения пары и домашнее задание со сроком его сдачи.

Быстрое получение информации о расписании занятий является залогом успешной учебы и работы преподавателей. Это позволяет студентам и преподавателям лучше планировать свое время, избегать перегрузок и эффективно использовать свои ресурсы.

## Создание функциональной диаграммы

Функциональная диаграмма показывает, какие функции выполняет система или проект. Она служит средством связи между различными заинтересованными группами, занимающимися разработкой продукта. Для построения схемы мы использовали графический сервис Draw.io. Она представлена на рисунке 1.

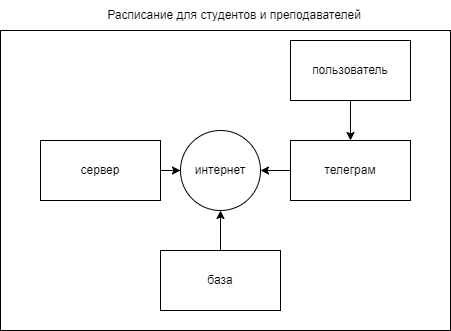


Рисунок 1 – Функциональная диаграмма

# ПРОЕКТИРОВАНИЕ

## Описание предметной области

Создание базы данных начинается с составления описания предметной области. В данном случае создается база данных для университета.

Студенты и преподаватели высшего учебного заведения могут просматривать свое расписание на определенный день.

Все пользователи имеют уникальные идентификаторы: номер пользователя, пароль, логин, роль.

Все студенты состоят в определенной академической группе. В ней содержится: номер самой группы, название специальности, курс и семестр.

Преподаватели ведут различные дисциплины. Это могут быть как лекции, так и семинары, практики.

По каждой дисциплине имеются домашние задания и соответствующие учебные материалы для успешного решения задач. В них указывается номер домашнего задания или учебного материала, название предмета, дедлайн, описание, т.е. тема или раздел для изучения.

Таким образом, были выделены следующие сущности:

* Расписание занятий;
* Пользователь;
* Преподаватель;
* Студент;
* Группа;
* Предмет;

С базой данных будет работать системный администратор и преподаватель.

## Требования к системному ПО и СУБД

Необходимо:

1. ПК с операционной системой windows 7 и выше или unix-подобная система, поддерживаемая для Telegram;
2. Токен телеграмм бота, полученный в боте BotFather;
3. Инструмент для работы с объектно-реляционной СУБД PostgreSQL 15. Она отвечает следующим требованиям, которые важны для нашего проекта:

* Скорость доступа;
* Регулярное резервное копирование данных;
* Целостность данных (непротиворечивость данных);
* Надёжность (целостность БД не должна нарушаться при технических сбоях);
* Возможность актуализации (изменение, корректировка данных);
* Совместимость (бд должна быть совместима с другими системами и стандартами).

1. Данные для подключения к серверу;
2. Язык программирования Python 3.7 и выше;
3. Необходимые библиотеки для создания бота;
4. Среда разработки Visual Studio Code/PyCharm или любая другая, поддерживающая Python 3.7 и выше.

## Описание архитектуры продукта

Архитектура проекта представляет собой Telegram-бота, который обеспечивает быстрый и удобный просмотр расписания занятий студентами и преподавателями.

Бот работает по принципу запроса информации пользователем и отправки ему ответа.

Telegram-бот состоит из следующих компонентов:

1. Серверная часть: отвечает за обработку запросов пользователей и отправку ответов.
2. Клиентская часть: представляет собой Telegram-бот, через который пользователи отправляют запросы боту и получают ответы. Пользователям должно быть удобно и понятно, как взаимодействовать с ботом через Telegram.
3. База данных: используется для хранения информации о расписании занятий. В качестве базы данных используем СУБД PostgreSQL, поскольку данная БД проста в использовании и является бесплатной.

Данная архитектура позволяет боту обрабатывать запросы пользователей, получать актуальную информацию о расписании и отправлять ее обратно пользователям. Благодаря описанной архитектуре можно составить архитектурную диаграмму (Рисунок 2).

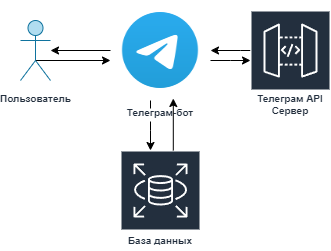


Рисунок 2 – Архитектурная диаграмма

## Функциональная диаграмма UML

Функциональная диаграмма представляет собой схематическое представление взаимосвязей нескольких функций. На рисунке 3 представлена UML диаграмма нашего продукта.

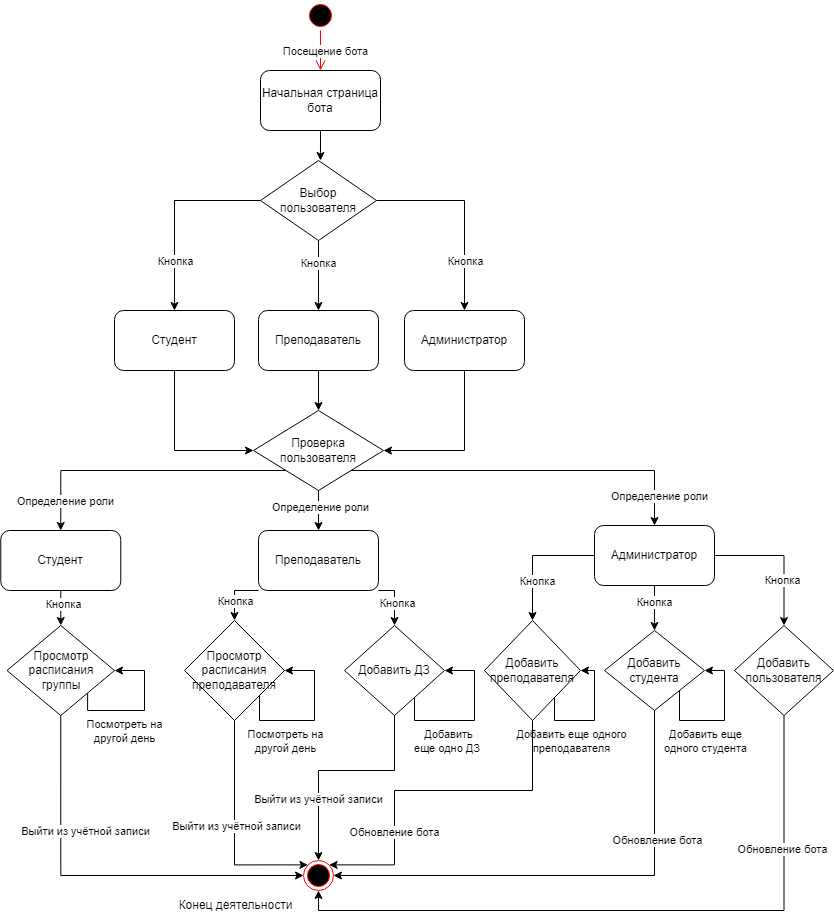


Рисунок 3 – Функциональная диаграмма UML

## Диаграмма последовательности UML

Обычно диаграмма последовательности описывает один сценарий. На диаграмме показаны экземпляры объектов и сообщения, которыми обмениваются объекты в рамках одного прецедента. Данная схема представлена на рисунке 4.

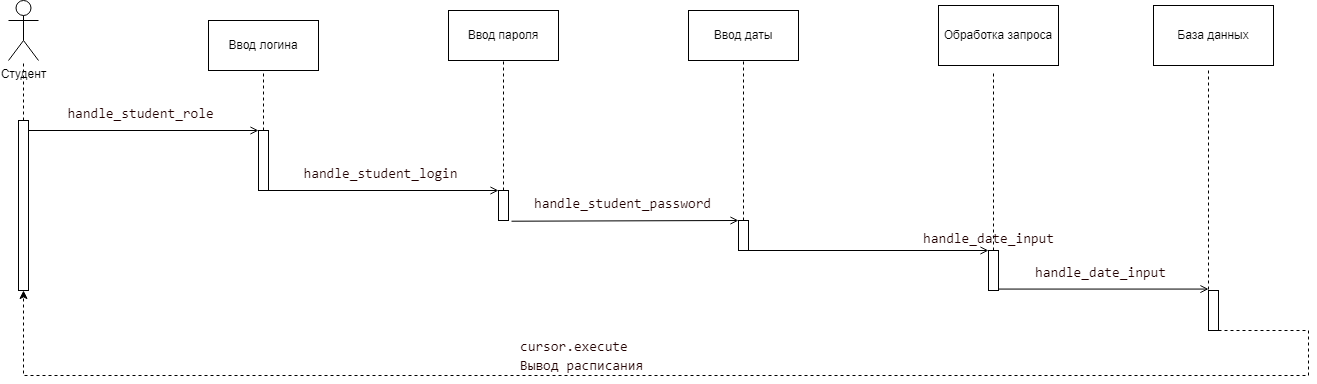


Рисунок 4 – Диаграмма последовательности

## Концептуальная модель БД

Концептуальная модель – модель предметной области, состоящей из перечня связанных понятий, используемых для описания области, вместе со свойствами и характеристиками, классификацией этих понятий, по видам, ситуациям, признакам в данной области и алгоритмов протекания процессов в ней. Представлена на рисунке 5.



Рисунок 5 – Концептуальная модель БД

# РАЗРАБОТКА

В соответствии с поставленными задачами, в данном проекте был использован следующий стек технологий и библиотек:

* Visual Studio Code – среда разработки, поддерживаемая интеграцию с GitHub;
* PostgreSQL – реляционная система управления базами данных. Использовалась для хранения и управления данными в проекте;
* Python – язык программирования, применяемый для реализации бизнес-логики и взаимодействия с базой данных;
* aiogram – библиотека для создания телеграм-ботов на языке Python. Использовалась для реализации чат-бота с функционалом, включающим обработку команд, работу с различными состояниями чат-бота и другие возможности;
* logging – библиотека Python для ведения логирования. Применялась для отслеживания различных событий и ошибок в приложении;
* psycopg2 – библиотека для взаимодействия с PostgreSQL из приложения на Python. Была использована для подключения к базе данных и выполнения SQL-запросов;
* MemoryStorage из aiogram.contrib.fsm\_storage – использовалось для сохранения состояний чат-бота в оперативной памяти.

## 2.1 Диаграмма пакетов UML

Пакет (package) – это инструмент группирования, который позволяет взять любую конструкцию UML и объединить ее элементы в единицы высокого уровня. В основном пакеты служат для объединения классов в группы. Диаграмма пакетов представлена на рисунке 6.

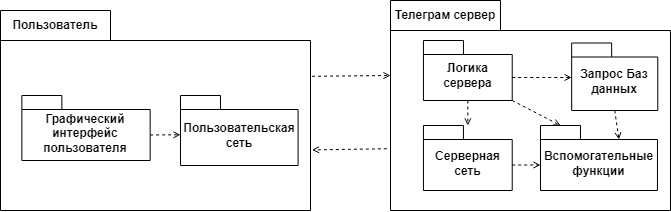


Рисунок 6 – Диаграмма пакетов UML

## 2.2 Диаграмма физической модели (ER-диаграмма)

С помощью бесплатного конструктора баз данных DbDesigner.net построили физическую модель проектируемой базы данных (Рисунок 7).

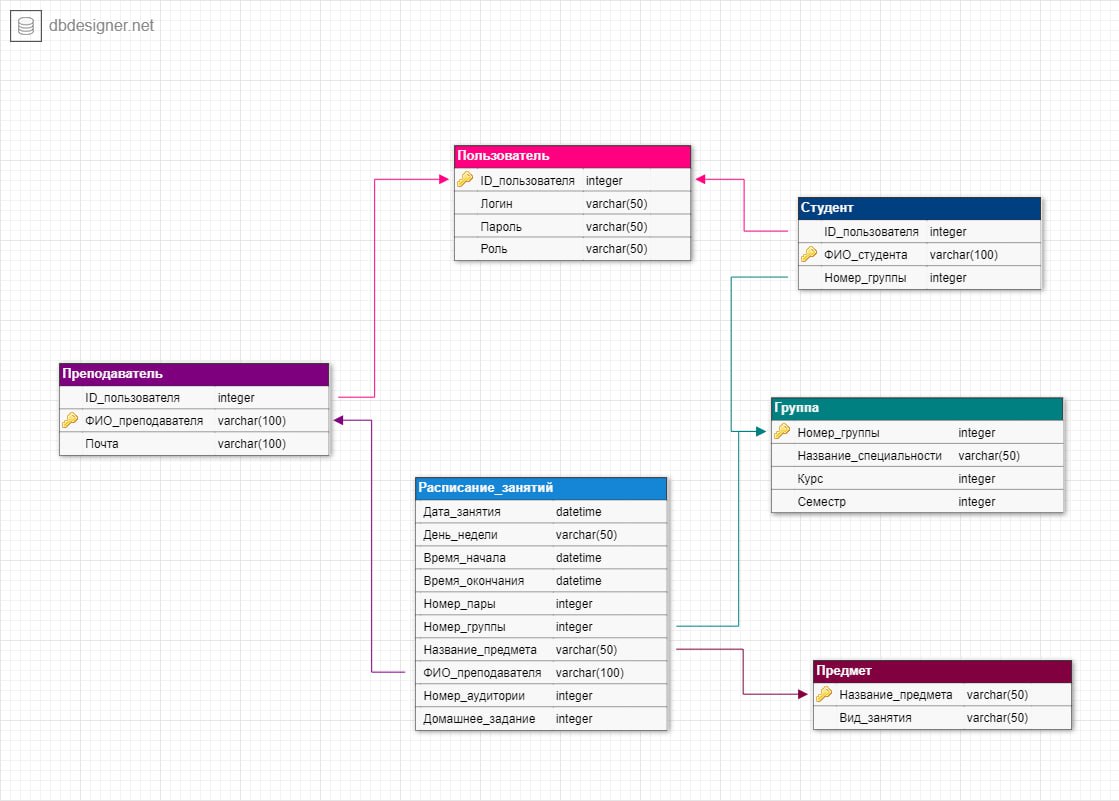


Рисунок 7 – Физическая модель БД

Для создания пользовательского интерфейса был выбран язык программирования Python, так как в нем собрано множество библиотек, удобных для написания телеграм-бота. К сожалению, не удалось согласно диаграммам реализовать добавление домашнего задания от роли преподавателя таким образом, чтобы после просмотра расписания можно было сразу его добавить. Для этого необходимо выйти, снова зайти под ролью преподавателя и написать домашнее задание.

Код для программы представлен в Приложении А.

# ЗАКЛЮЧЕНИЕ

В результате выполнения работы был реализован Telegram-бот, в котором студенты и преподаватели университета могут смотреть свое расписание занятий на день, не скачивая для этого дополнительное приложение.

Для создания нашей информационной системы мы описали предметную область, разработали архитектуру продукта, составили требования к СУБД и системному ПО. Кроме этого, были созданы несколько видов диаграмм для общего понимания работы нашего продукта.

Для реализации проекта мы создали базу данных с необходимыми исходными данными и три роли: студент, преподаватель и администратор. После, на основании предоставленных диаграмм, разработчик приступил к реализации пользовательского интерфейса.

Таким образом, можно сделать вывод о том, что задание курсового проекта выполнено.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Бесплатный конструктор баз данных DbDesigner.net [Электронный ресурс]. - URL: <https://app.dbdesigner.net/designer> (дата обращения 10.12.2023).
2. Крёнке Д. Теория и практика построения баз данных / Д.Крёнке. — СПб.: Питер, 2003. — 800 с.: ил. — Библиогр. рус. — Библиогр.: с. 198-199.
3. UML. Основы, 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2004. – 192 с., ил. ISBN 5-93286-060-Х. – Текст: непосредственный.
4. Бесплатный онлайн-сервис для создания блок-схем и диаграмм drawio.com [Электронный ресурс]. - URL: <https://www.drawio.com/> (дата обращения 18.12.2023).
5. Документация по работе с СУБД PostgreSQL. [Электронный ресурс]. - URL: <https://www.postgresql.org/docs/> (дата обращения 14.12.2023).

# ПРИЛОЖЕНИЕ А

Код программы на Python:

import logging

import psycopg2

import datetime

import os

import json

from aiogram import Bot, Dispatcher, types

from aiogram import executor

from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

from aiogram.dispatcher import FSMContext

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

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

from aiogram.dispatcher.filters import Command

host = 'localhost'

port = '5432'

user = 'postgres'

password = '12345'

database = 'schedule'

API\_TOKEN = '6780626018:AAGvOFKG4DmK8aIVckUGd-bC77zYv5uz\_70'

logging.basicConfig(level=logging.INFO)

bot = Bot(token=API\_TOKEN)

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

role\_keyboard = ReplyKeyboardMarkup(resize\_keyboard=True)

role\_keyboard.add(KeyboardButton('Студент'))

role\_keyboard.add(KeyboardButton('Преподаватель'))

role\_keyboard.add(KeyboardButton('Администратор'))

logout\_keyboard = ReplyKeyboardMarkup(resize\_keyboard=True)

logout\_keyboard.add(KeyboardButton('Выйти из учётной записи'))

class teacherAuthState(StatesGroup):

logteacher\_auth = State()

passteacher\_auth = State()

class studentState(StatesGroup):

logstudent = State()

passstudent = State()

date\_input = State()

logout = State()

class teacherState(StatesGroup):

logteacher = State()

passteacher = State()

date\_input\_teacher = State()

add\_homework\_date = State()

add\_homework\_group = State()

add\_homework\_task = State()

logout\_teacher = State()

authentication = teacherAuthState

class teacherAuthenticatedState(StatesGroup):

view\_schedule = State()

add\_homework = State()

enter\_date = State()

class AddHomeworkState(StatesGroup):

enter\_date = State()

enter\_group = State()

enter\_task = State()

# Создаем словарь для хранения данных пользователей

user\_data = {}

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

async def send\_welcome(message: types.Message):

await message.answer("Привет! Я бот расписания университета. Кто вы?", reply\_markup=role\_keyboard)

####################################################СТУДЕНТ########################################

@dp.message\_handler(lambda message: message.text == "Студент")

async def handle\_student\_role(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

await student(message, chat\_id, state)

async def student(message: types.Message, chat\_id: int, state: FSMContext):

await bot.send\_message(chat\_id, "Введите свой логин:")

await studentState.logstudent.set()

await state.update\_data(chat\_id=chat\_id)

@dp.message\_handler(state=studentState.logstudent)

async def handle\_student\_login(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

p\_logstudent = message.text

# Сохраняем логин в словаре

user\_data[chat\_id] = {'p\_logstudent': p\_logstudent}

await bot.send\_message(chat\_id, "Введите свой пароль:")

await studentState.passstudent.set()

@dp.message\_handler(state=studentState.passstudent)

async def handle\_student\_password(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

p\_password = message.text

# Сохраняем пароль в словаре

user\_data[chat\_id]['p\_password'] = p\_password

await bot.send\_message(chat\_id, "Введите дату в формате ГГГГ-ММ-ДД:")

await studentState.date\_input.set()

@dp.message\_handler(state=studentState.date\_input)

async def handle\_date\_input(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

schedule\_date\_param = message.text

if schedule\_date\_param.lower() == 'выйти из учётной записи':

await handle\_logout(message, state)

return

await state.update\_data(schedule\_date\_param=schedule\_date\_param)

data = await state.get\_data()

try:

# Получаем логин и пароль из словаря

p\_logstudent = user\_data[chat\_id]['p\_logstudent']

p\_password = user\_data[chat\_id]['p\_password']

conn = psycopg2.connect(database=database, user=user, password=password, host=host, port=port)

cursor = conn.cursor()

# Вызов функции с логином, паролем и датой

cursor.execute('SELECT \* FROM get\_student\_schedule(%s, %s, %s)', (p\_logstudent, p\_password, data['schedule\_date\_param']))

# Получаем все результаты функции

results = cursor.fetchall()

cursor.close()

conn.close()

if results:

# Группируем результаты по дням

grouped\_results = group\_by\_date(results)

# Получаем номер группы из первой строки расписания (если есть)

group\_number = results[0][5] if len(results[0]) > 5 else None

# Получаем день недели из первой строки расписания (если есть)

day\_of\_week = results[0][1] if len(results[0]) > 1 else None

# Формируем одно сообщение с номером группы и днем недели

message\_text = ""

if group\_number is not None:

message\_text += f"Номер группы: {group\_number}\n"

if day\_of\_week is not None:

message\_text += f"День недели: {day\_of\_week}\n"

# Форматируем расписание и добавляем к сообщению

for date, data\_for\_date in grouped\_results.items():

message\_text += format\_date\_schedule(date, data\_for\_date)

await bot.send\_message(chat\_id, message\_text, reply\_markup=logout\_keyboard)

else:

await bot.send\_message(chat\_id, f"На {schedule\_date\_param} пар нет 😺")

except Exception as e:

await bot.send\_message(chat\_id, f"Неправильный логин, пароль или дата. Попробуйте ещё раз")

await bot.send\_message(chat\_id, f"Нажмите /start")

await state.reset\_state()

@dp.message\_handler(lambda message: message.text == "Выйти из учётной записи", state=studentState.date\_input)

async def handle\_logout(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

# Remove user data for the current chat\_id

user\_data.pop(chat\_id, None)

await state.finish() # Finish the current state

await message.answer("Вы успешно вышли из учётной записи. Начнем заново.", reply\_markup=role\_keyboard)

def group\_by\_date(results):

"""Группирует результаты по дням."""

grouped\_results = {}

for row in results:

date = row[0]

if date not in grouped\_results:

grouped\_results[date] = []

grouped\_results[date].append(row[1:])

return grouped\_results

def format\_date\_schedule(date, schedule):

"""Форматирует расписание для одного дня."""

if not schedule:

return f"На {date} расписания нет."

date\_schedule = f"📆Дата: {date}\n"

for row in schedule:

date\_schedule += "\n"

date\_schedule += "Номер пары: {}\n".format(row[3]) if len(row) > 3 else ""

date\_schedule += "🙀Время начала пары: {}\n".format(row[1]) if len(row) > 1 else ""

date\_schedule += "😸Время окончания пары: {}\n".format(row[2]) if len(row) > 2 else ""

date\_schedule += "Название предмета: {}\n".format(row[5]) if len(row) > 4 else ""

date\_schedule += "😻ФИО преподавателя: {}\n".format(row[6]) if len(row) > 5 else ""

date\_schedule += "Аудитория: {}\n".format(row[7]) if len(row) > 6 else ""

date\_schedule += "😿Домашнее задание: {}\n".format(row[8]) if len(row) > 7 else ""

date\_schedule += "\n"

return date\_schedule

#######################################ПРЕПОДАВАТЕЛЬ#################################################################

@dp.message\_handler(lambda message: message.text == "Преподаватель")

async def handle\_teacher\_role(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

await bot.send\_message(chat\_id, "Введите свой логин:")

await teacherAuthState.logteacher\_auth.set()

await state.update\_data(chat\_id=chat\_id)

@dp.message\_handler(state=teacherAuthState.logteacher\_auth)

async def handle\_teacher\_login\_auth(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

p\_logteacher\_auth = message.text

# Сохраняем логин в словаре user\_data

user\_data[chat\_id] = {'p\_logteacher\_auth': p\_logteacher\_auth}

await bot.send\_message(chat\_id, "Введите свой пароль:")

await teacherAuthState.passteacher\_auth.set()

async def authenticate\_teacher(chat\_id):

try:

p\_logteacher\_auth = user\_data[chat\_id]['p\_logteacher\_auth']

p\_password\_auth = user\_data[chat\_id]['p\_password\_auth']

conn = psycopg2.connect(database=database, user=user, password=password, host=host, port=port)

cursor = conn.cursor()

# Проверяем, существует ли пользователь с указанными учетными данными и ролью 'Преподаватель'

cursor.execute('SELECT \* FROM Пользователь WHERE Логин = %s AND Пароль = %s AND Роль = %s', (p\_logteacher\_auth, p\_password\_auth, 'преподаватель'))

result = cursor.fetchone()

cursor.close()

conn.close()

return result is not None

except Exception as e:

print(e)

return False

@dp.message\_handler(state=teacherAuthState.passteacher\_auth)

async def handle\_teacher\_password\_auth(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

p\_password\_auth = message.text

# Сохраняем пароль в словаре user\_data

user\_data[chat\_id]['p\_password\_auth'] = p\_password\_auth

# Аутентификация преподавателя

if await authenticate\_teacher(chat\_id):

await bot.send\_message(chat\_id, "Авторизация успешна!")

# Добавляем кнопки после успешной аутентификации

keyboard = ReplyKeyboardMarkup(resize\_keyboard=True)

keyboard.add(KeyboardButton('Посмотреть расписание'))

keyboard.add(KeyboardButton('Добавить домашнее задание'))

keyboard.add(KeyboardButton('Выйти из учётной записи'))

await bot.send\_message(chat\_id, "Выберите действие:", reply\_markup=keyboard)

await teacherAuthenticatedState.view\_schedule.set()

else:

await bot.send\_message(chat\_id, "Неверный логин или пароль. Попробуйте ещё раз.")

await state.finish()

@dp.message\_handler(lambda message: message.text == "Посмотреть расписание", state=teacherAuthenticatedState.view\_schedule)

async def handle\_view\_schedule(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

await bot.send\_message(chat\_id, "Введите дату в формате ГГГГ-ММ-ДД:")

await teacherAuthenticatedState.enter\_date.set()

@dp.message\_handler(state=teacherAuthenticatedState.enter\_date)

async def handle\_teacher\_enter\_date(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

schedule\_date\_param = message.text

try:

# Получаем логин и пароль из словаря user\_data

p\_logteacher\_auth = user\_data[chat\_id]['p\_logteacher\_auth']

p\_password\_auth = user\_data[chat\_id]['p\_password\_auth']

conn = psycopg2.connect(database=database, user=user, password=password, host=host, port=port)

cursor = conn.cursor()

# Вызов функции с логином, паролем и введенной датой

cursor.execute('SELECT \* FROM get\_teacher\_schedule(%s, %s, %s)', (p\_logteacher\_auth, p\_password\_auth, schedule\_date\_param))

# Получаем все результаты функции

results = cursor.fetchall()

cursor.close()

conn.close()

if results:

grouped\_results = group\_by\_date(results)

# Выводим дату и день недели

message\_text = f"Ваше расписание на {schedule\_date\_param}:\n\n"

day\_of\_week = results[0][1] if len(results[0]) > 1 else None

if day\_of\_week is not None:

message\_text += f"День недели: {day\_of\_week}\n"

# Форматируем расписание и добавляем к сообщению

for date, data\_for\_date in grouped\_results.items():

message\_text += f"\nДата: {date}\n"

for row in data\_for\_date:

message\_text += "Время начала пары: {}\n".format(row[1]) if len(row) > 1 else ""

message\_text += "Время окончания: {}\n".format(row[2]) if len(row) > 2 else ""

message\_text += "Номер пары: {}\n".format(row[3]) if len(row) > 2 else ""

message\_text += "Номер группы: {}\n".format(row[4]) if len(row) > 3 else ""

message\_text += "Название предмета: {}\n".format(row[5]) if len(row) > 4 else ""

message\_text += "ФИО преподавателя: {}\n".format(row[6]) if len(row) > 5 else ""

message\_text += "Номер аудитории: {}\n".format(row[7]) if len(row) > 6 else ""

message\_text += "Домашнее задание: {}\n".format(row[8]) if len(row) > 7 else ""

message\_text += "\n"

await bot.send\_message(chat\_id, message\_text, reply\_markup=logout\_keyboard)

else:

await bot.send\_message(chat\_id, f"На {schedule\_date\_param} у вас нет занятий 😺")

except Exception as e:

await bot.send\_message(chat\_id, "Произошла ошибка при получении расписания. Попробуйте ещё раз. Нажмите /start")

await state.finish()

@dp.message\_handler(lambda message: message.text == "Добавить домашнее задание", state=teacherAuthenticatedState.view\_schedule)

async def handle\_add\_homework(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

await bot.send\_message(chat\_id, "Введите дату в формате ГГГГ-ММ-ДД:")

await AddHomeworkState.enter\_date.set()

@dp.message\_handler(state=AddHomeworkState.enter\_date)

async def handle\_enter\_date\_for\_homework(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

date\_input = message.text

await state.update\_data(date\_input=date\_input)

await bot.send\_message(chat\_id, "Введите номер группы:")

await AddHomeworkState.enter\_group.set()

@dp.message\_handler(state=AddHomeworkState.enter\_group)

async def handle\_enter\_group\_for\_homework(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

group\_number = message.text

await state.update\_data(group\_number=group\_number)

await bot.send\_message(chat\_id, "Введите текст домашнего задания:")

await AddHomeworkState.enter\_task.set()

@dp.message\_handler(state=AddHomeworkState.enter\_task)

async def handle\_enter\_task\_for\_homework(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

task\_text = message.text

data = await state.get\_data()

date\_input = data['date\_input']

group\_number = data['group\_number']

# Вызов процедуры в базе данных для добавления домашнего задания

try:

conn = psycopg2.connect(database=database, user=user, password=password, host=host, port=port)

cursor = conn.cursor()

cursor.execute('CALL add\_homework(%s, %s, %s, %s, %s)', (user\_data[chat\_id]['p\_logteacher\_auth'], user\_data[chat\_id]['p\_password\_auth'], data['date\_input'], data['group\_number'], task\_text))

conn.commit()

cursor.close()

conn.close()

await bot.send\_message(chat\_id, "Домашнее задание успешно добавлено! Нажмите /start")

except Exception as e:

await bot.send\_message(chat\_id, "Произошла ошибка при добавлении домашнего задания. Попробуйте ещё раз.")

print(e)

finally:

await state.finish()

@dp.message\_handler(lambda message: message.text == "Выйти из учётной записи", state=teacherAuthenticatedState.view\_schedule)

async def handle\_logout\_teacher(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

# Remove user data for the current chat\_id

user\_data.pop(chat\_id, None)

await state.finish() # Finish the current state

await message.answer("Вы успешно вышли из учётной записи. Начнем заново.", reply\_markup=role\_keyboard)

###################АДМИНИСТРАТОР###############################################################################

# Add a new state for handling the "Добавить пользователя" action

class AddUserState(StatesGroup):

enter\_login = State()

enter\_password = State()

enter\_role = State()

# Modify the existing AdminState to include the new state

class AdminState(StatesGroup):

logadmin = State()

passadmin = State()

date\_input\_admin = State()

start\_admin = State()

add\_user = State()

add\_student = State()

add\_teacher = State()

class AddStudentState(StatesGroup):

enter\_user\_id = State()

enter\_student\_name = State()

enter\_group\_number = State()

class AddTeacherState(StatesGroup):

enter\_user\_id = State()

enter\_teacher\_name = State()

enter\_teacher\_email = State()

@dp.message\_handler(lambda message: message.text == "Администратор")

async def handle\_admin\_role(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

await admin\_authentication(message, chat\_id, state)

async def admin\_authentication(message: types.Message, chat\_id: int, state: FSMContext):

await bot.send\_message(chat\_id, "Введите свой логин:")

await AdminState.logadmin.set()

await state.update\_data(chat\_id=chat\_id)

@dp.message\_handler(state=AdminState.logadmin)

async def handle\_admin\_login(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

p\_logadmin = message.text

# Сохраняем логин в словаре

user\_data[chat\_id] = {'p\_logadmin': p\_logadmin}

await bot.send\_message(chat\_id, "Введите свой пароль:")

await AdminState.passadmin.set()

@dp.message\_handler(state=AdminState.passadmin)

async def handle\_admin\_password(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

p\_passadmin = message.text

# Сохраняем пароль в словаре

user\_data[chat\_id]['p\_passadmin'] = p\_passadmin

try:

# Получаем логин и пароль из словаря

p\_logadmin = user\_data[chat\_id]['p\_logadmin']

p\_passadmin = user\_data[chat\_id]['p\_passadmin']

conn = psycopg2.connect(database=database, user=user, password=password, host=host, port=port)

cursor = conn.cursor()

# Проверяем логин, пароль и роль админа в таблице "Пользователь"

cursor.execute('SELECT Роль FROM "Пользователь" WHERE "Логин" = %s AND "Пароль" = %s', (p\_logadmin, p\_passadmin))

role = cursor.fetchone()

if role and role[0] == 'админ':

await AdminState.start\_admin.set() # Перемещаем эту строку выше

keyboard = ReplyKeyboardMarkup(resize\_keyboard=True)

keyboard.add(KeyboardButton("Добавить пользователя"))

keyboard.add(KeyboardButton("Добавить студента"))

keyboard.add(KeyboardButton("Добавить преподавателя"))

await bot.send\_message(chat\_id, "Вы успешно вошли как администратор.", reply\_markup=keyboard)

else:

await bot.send\_message(chat\_id, "Неправильный логин, пароль или вы не администратор.")

await state.reset\_state()

except Exception as e:

await bot.send\_message(chat\_id, f"Произошла ошибка при аутентификации администратора. Попробуйте еще раз.")

await state.reset\_state()

finally:

cursor.close()

conn.close()

# Add a message handler for the "Добавить пользователя" button

@dp.message\_handler(lambda message: message.text == "Добавить пользователя", state=AdminState.start\_admin)

async def add\_user\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

await bot.send\_message(chat\_id, "Введите логин нового пользователя:")

await AddUserState.enter\_login.set()

# Add message handlers for entering login, password, and role for the new user

@dp.message\_handler(state=AddUserState.enter\_login)

async def enter\_login\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

new\_user\_login = message.text

await state.update\_data(new\_user\_login=new\_user\_login)

await bot.send\_message(chat\_id, "Введите пароль нового пользователя:")

await AddUserState.enter\_password.set()

@dp.message\_handler(state=AddUserState.enter\_password)

async def enter\_password\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

new\_user\_password = message.text

await state.update\_data(new\_user\_password=new\_user\_password)

await bot.send\_message(chat\_id, "Введите роль нового пользователя:")

await AddUserState.enter\_role.set()

@dp.message\_handler(state=AddUserState.enter\_role)

async def enter\_role\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

new\_user\_role = message.text

await state.update\_data(new\_user\_role=new\_user\_role)

# Access the entered data

user\_data = await state.get\_data()

new\_user\_login = user\_data.get("new\_user\_login")

new\_user\_password = user\_data.get("new\_user\_password")

new\_user\_role = user\_data.get("new\_user\_role")

try:

# Establish a database connection

conn = psycopg2.connect(database=database, user=user, password=password, host=host, port=port)

cursor = conn.cursor()

# Insert new user data into the "Пользователь" table

cursor.execute(f'INSERT INTO "Пользователь" ("Логин", "Пароль", "Роль") VALUES (%s, %s, %s) RETURNING "ID\_пользователя"', (new\_user\_login, new\_user\_password, new\_user\_role))

new\_user\_id = cursor.fetchone()[0]

conn.commit()

await bot.send\_message(chat\_id, f"Пользователь с ID {new\_user\_id}, логином {new\_user\_login} успешно добавлен с ролью {new\_user\_role}.")

await state.reset\_state()

finally:

cursor.close()

conn.close()

# Добавьте обработчик для кнопки "Добавить студента"

@dp.message\_handler(lambda message: message.text == "Добавить студента")

async def add\_student\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

await bot.send\_message(chat\_id, "Введите ID пользователя студента:")

await AddStudentState.enter\_user\_id.set()

@dp.message\_handler(state=AddStudentState.enter\_user\_id)

async def enter\_user\_id\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

user\_id = message.text

await state.update\_data(user\_id=user\_id)

await bot.send\_message(chat\_id, "Введите ФИО студента:")

await AddStudentState.enter\_student\_name.set()

# Обработчик для ввода ФИО студента

@dp.message\_handler(state=AddStudentState.enter\_student\_name)

async def enter\_student\_name\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

student\_name = message.text

await state.update\_data(student\_name=student\_name)

await bot.send\_message(chat\_id, "Введите номер группы:")

await AddStudentState.enter\_group\_number.set()

# Обработчик для ввода номера группы студента

@dp.message\_handler(state=AddStudentState.enter\_group\_number)

async def enter\_group\_number\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

group\_number = message.text

# Достаем данные из состояния

user\_data = await state.get\_data()

user\_id = user\_data.get("user\_id")

student\_name = user\_data.get("student\_name")

try:

# Создаем подключение к базе данных

conn = psycopg2.connect(database=database, user=user, password=password, host=host, port=port)

cursor = conn.cursor()

# Вставляем данные студента в таблицу "Студент"

insert\_student\_query = '''

INSERT INTO "Студент" ("ID\_пользователя", "ФИО\_студента", "Номер\_группы")

VALUES (%s, %s, %s);

'''

cursor.execute(insert\_student\_query, (user\_id, student\_name, group\_number))

conn.commit()

await bot.send\_message(chat\_id, f"Студент {student\_name} успешно добавлен в группу {group\_number}.")

await state.reset\_state()

except Exception as e:

await bot.send\_message(chat\_id, f"Произошла ошибка при добавлении студента: {str(e)}. Попробуйте еще раз.")

await state.reset\_state()

finally:

cursor.close()

conn.close()

@dp.message\_handler(lambda message: message.text == "Добавить преподавателя")

async def add\_teacher\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

await bot.send\_message(chat\_id, "Введите ID пользователя:")

await AddTeacherState.enter\_user\_id.set()

# Обработчик для ввода ID пользователя

@dp.message\_handler(state=AddTeacherState.enter\_user\_id)

async def enter\_user\_id\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

user\_id = message.text

await state.update\_data(user\_id=user\_id)

await bot.send\_message(chat\_id, "Введите ФИО преподавателя:")

await AddTeacherState.enter\_teacher\_name.set()

# Обработчик для ввода ФИО преподавателя

@dp.message\_handler(state=AddTeacherState.enter\_teacher\_name)

async def enter\_teacher\_name\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

teacher\_name = message.text

await state.update\_data(teacher\_name=teacher\_name)

await bot.send\_message(chat\_id, "Введите почту преподавателя:")

await AddTeacherState.enter\_teacher\_email.set()

# Обработчик для ввода почты преподавателя

@dp.message\_handler(state=AddTeacherState.enter\_teacher\_email)

async def enter\_teacher\_email\_handler(message: types.Message, state: FSMContext):

chat\_id = message.chat.id

teacher\_email = message.text

# Достаем данные из состояния

user\_data = await state.get\_data()

user\_id = user\_data.get("user\_id")

teacher\_name = user\_data.get("teacher\_name")

try:

# Создаем подключение к базе данных

conn = psycopg2.connect(database=database, user=user, password=password, host=host, port=port)

cursor = conn.cursor()

# Вставляем данные преподавателя в таблицу "Преподаватель"

cursor.execute(

'INSERT INTO "Преподаватель" ("ID\_пользователя", "ФИО\_преподавателя", "Почта") VALUES (%s, %s, %s)',

(user\_id, teacher\_name, teacher\_email)

)

conn.commit()

await bot.send\_message(chat\_id, f"Преподаватель {teacher\_name} успешно добавлен с почтой {teacher\_email}.")

await state.reset\_state()

except Exception as e:

await bot.send\_message(chat\_id, f"Произошла ошибка при добавлении преподавателя: {str(e)}. Попробуйте еще раз.")

await state.reset\_state()

finally:

cursor.close()

conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

logging.info("Бот запущен!")

executor.start\_polling(dp, skip\_updates=True)