from aiogram import Bot, Dispatcher, types

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

from aiogram.dispatcher import FSMContext

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

from aiogram.utils import executor

from sqlalchemy import create\_engine, Column, Integer, String, ForeignKey, Text, DateTime

from sqlalchemy.orm import declarative\_base, relationship, sessionmaker

from sqlalchemy.orm import Session

from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

from aiogram.types import ParseMode

from datetime import datetime

import logging

import os

import random

import string

import re

logging.getLogger("sqlalchemy.engine").setLevel(logging.WARNING)

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

# Настройка базы данных SQLite

DATABASE\_URL = "sqlite:///database.db"

Base = declarative\_base()

engine = create\_engine(DATABASE\_URL, echo=True)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

session = SessionLocal()

# Определение таблиц базы данных

class EducationalInstitution(Base):

\_\_tablename\_\_ = "educational\_institutions"

id = Column(Integer, primary\_key=True, index=True)

name = Column(String, unique=True)

class User(Base):

\_\_tablename\_\_ = "users"

id = Column(Integer, primary\_key=True, index=True)

telegram\_id = Column(String, unique=True, index=True)

full\_name = Column(String)

role = Column(String) # "student" или "teacher"

group\_id = Column(Integer)

institution\_id = Column(Integer, ForeignKey('educational\_institutions.id')) # Внешний ключ

institution = relationship("EducationalInstitution", back\_populates="users")

EducationalInstitution.users = relationship("User", back\_populates="institution")

class Group(Base):

\_\_tablename\_\_ = "groups"

id = Column(Integer, primary\_key=True, index=True)

name = Column(String, unique=True)

password = Column(String)

teacher\_id = Column(Integer, ForeignKey("users.id"), nullable=True) # Сделано nullable=True

class Task(Base):

\_\_tablename\_\_ = "tasks"

id = Column(Integer, primary\_key=True, index=True)

group\_id = Column(Integer, ForeignKey("groups.id"))

name = Column(String)

description = Column(Text)

deadline = Column(DateTime)

input\_data = Column(Text, nullable=True)

expected\_result = Column(String, nullable=True)

# Состояние для выбора группы

class WatchGroup(StatesGroup):

waiting\_for\_group\_id = State()

# Создание таблиц базы данных

Base.metadata.create\_all(bind=engine)

# Настройка Telegram-бота

API\_TOKEN = "7202246841:AAHFaSOPlpIJTRjzSKx-HEVXt65axfJPz\_Q"

bot = Bot(token=API\_TOKEN)

storage = MemoryStorage()

dp = Dispatcher(bot, storage=storage)

class Form(StatesGroup):

waiting\_for\_group\_id = State()

class CreateGroup(StatesGroup):

waiting\_for\_group\_name = State()

""""

class CreateTask(StatesGroup):

waiting\_for\_group\_id = State()

waiting\_for\_task\_name = State()

waiting\_for\_task\_description = State()

waiting\_for\_task\_deadline = State()

waiting\_for\_input\_data = State() # Новый шаг для ввода входных данных

waiting\_for\_expected\_result = State() # Новый шаг для ввода ожидаемого результата

"""

class CreateTask(StatesGroup):

waiting\_for\_group\_id = State()

waiting\_for\_task\_name = State()

waiting\_for\_task\_description = State()

waiting\_for\_task\_deadline = State() # Добавляем состояние для дедлайна

waiting\_for\_input\_data = State() # Исправлено имя состояния

waiting\_for\_expected\_result = State() # Исправлено имя состояния

def get\_back\_button():

return ReplyKeyboardMarkup(resize\_keyboard=True).add(KeyboardButton("Назад"))

def get\_input\_data\_keyboard():

return ReplyKeyboardMarkup(resize\_keyboard=True).add(

KeyboardButton("Назад"),

KeyboardButton("Входных данных нет")

)

def get\_deadline\_keyboard():

return ReplyKeyboardMarkup(resize\_keyboard=True).add(

KeyboardButton("Назад"),

KeyboardButton("НЕТ ДЕДЛАЙНА")

)

def generate\_password(length=8):

characters = string.ascii\_letters + string.digits + string.punctuation

return ''.join(random.choice(characters) for \_ in range(length))

def escape\_markdown(text: str) -> str:

# Экранирование символов, которые могут нарушить разметку Markdown

return re.sub(r'([\\\_\*[\]()>#+-.!|])', r'\\\1', text)

def is\_full\_name\_unique\_in\_institution(db: Session, institution\_id: int, full\_name: str) -> bool:

# Проверка, есть ли уже такой пользователь с данным ФИО в организации

existing\_user = db.query(User).filter\_by(institution\_id=institution\_id, full\_name=full\_name).first()

return existing\_user is None # Если пользователь с таким ФИО не найден, возвращаем True

# Команда /start

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

async def start\_command(message: types.Message):

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if user:

await message.reply(f"Вы уже зарегистрированы как {user.role}.")

else:

markup = types.ReplyKeyboardMarkup(resize\_keyboard=True)

markup.add("Студент", "Преподаватель")

await message.reply("Добро пожаловать! Выберите вашу роль:", reply\_markup=markup)

# Регистрация пользователя

# Обработчик для регистрации пользователя (студент или преподаватель)

@dp.message\_handler(lambda message: message.text in ["Студент", "Преподаватель"])

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

role = "student" if message.text == "Студент" else "teacher"

# Сохраняем роль в состоянии

await state.update\_data(role=role)

# Запрашиваем Ф.И.О. для пользователя

await state.set\_state("waiting\_for\_full\_name") # Переход к состоянию для ввода Ф.И.О.

await message.reply("Введите вашу фамилию и имя:" if role == "student" else "Введите вашу фамилию, имя и отчество:")

# Обработчик для ввода Ф.И.О. для студентов и преподавателей

@dp.message\_handler(state="waiting\_for\_full\_name", content\_types=types.ContentTypes.TEXT)

async def save\_full\_name(message: types.Message, state: FSMContext):

full\_name = message.text.strip()

# Получаем данные о пользователе из состояния

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

role = user\_data.get('role')

# Проверка для преподавателя, что введено полное Ф.И.О.

if role == "teacher" and len(full\_name.split()) < 3:

await message.reply("Для преподавателей нужно ввести фамилию, имя и отчество.")

return

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

# Запрашиваем регион

await state.set\_state("waiting\_for\_region")

await message.reply("Введите ваш регион:")

# Обработчик для ввода региона

@dp.message\_handler(state="waiting\_for\_region", content\_types=types.ContentTypes.TEXT)

async def save\_region(message: types.Message, state: FSMContext):

region = message.text.strip()

# Если регион — Москва, Санкт-Петербург или Севастополь, пропускаем ввод города и образовательной организации для студента

if region in ["Москва", "Санкт-Петербург", "Севастополь"]:

await state.update\_data(region=region)

await state.set\_state("waiting\_for\_educational\_institution") # Переходим к следующему шагу

await message.reply("Введите образовательную организацию:")

else:

# Сохраняем регион в базе

await state.update\_data(region=region)

# Запрашиваем город для студента

await state.set\_state("waiting\_for\_city")

await message.reply("Введите ваш город:")

# Обработчик для ввода образовательной организации (для преподавателя или студента)

# Обработчик для ввода образовательной организации (для преподавателя или студента)

@dp.message\_handler(state="waiting\_for\_educational\_institution", content\_types=types.ContentTypes.TEXT)

async def save\_educational\_institution(message: types.Message, state: FSMContext):

educational\_institution\_name = message.text.strip()

# Получаем данные о пользователе из состояния

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

# Проверяем, существует ли уже образовательная организация в базе данных

institution = session.query(EducationalInstitution).filter\_by(name=educational\_institution\_name).first()

# Если образовательная организация не найдена, создаем новую

if not institution:

institution = EducationalInstitution(name=educational\_institution\_name)

session.add(institution)

session.commit() # Сохраняем образовательную организацию в базе данных

# Сохраняем ID образовательной организации в данных

await state.update\_data(educational\_institution\_id=institution.id)

# Логика для студента

role = user\_data.get('role')

if role == "student":

# Сохраняем студента в базе данных

user = User(

telegram\_id=str(message.from\_user.id),

full\_name=user\_data.get('full\_name'),

role="student", # Студент

region=user\_data.get('region'),

city=user\_data.get('city'),

educational\_institution\_id=institution.id, # Ссылка на образовательную организацию

)

session.add(user)

session.commit() # Сохраняем студента в базе данных

# Завершаем регистрацию

await message.reply(f"Вы выбрали образовательную организацию: {institution.name}. Регистрация завершена! Пожалуйста, используйте команду /help для получения доступных вам команд.")

await state.finish()

else:

# Логика для преподавателя

user = User(

telegram\_id=str(message.from\_user.id),

full\_name=user\_data.get('full\_name'),

role="teacher", # Преподаватель

region=user\_data.get('region'),

city=user\_data.get('city'),

educational\_institution\_id=institution.id, # Ссылка на образовательную организацию

)

session.add(user)

session.commit() # Сохраняем преподавателя в базе данных

await message.reply(f"Вы выбрали образовательную организацию: {institution.name}. Регистрация завершена! Пожалуйста, используйте команду /help для получения доступных вам команд.")

await state.finish()

# Шаг 3: Обработка выбора преподавателя (для студента)

@dp.callback\_query\_handler(lambda c: c.data.startswith("teacher\_"), state="waiting\_for\_teacher")

async def handle\_teacher(callback\_query: types.CallbackQuery, state: FSMContext):

teacher\_id = int(callback\_query.data.split("\_")[1])

teacher = session.query(User).filter\_by(id=teacher\_id).first()

if not teacher:

await callback\_query.message.reply("Ошибка. Преподаватель с таким ID не найден.")

return

# Сохраняем выбор преподавателя в состоянии

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

# Шаг 4: Вывод групп, которые ведет выбранный преподаватель

groups = session.query(Group).filter\_by(teacher\_id=teacher\_id).all()

if not groups:

await callback\_query.message.reply(f"Этот преподаватель {teacher.full\_name} не ведет ни одной группы.")

return

group\_list = "\n".join([f"{group.id}: {group.name}" for group in groups])

await callback\_query.message.reply(f"Вы выбрали преподавателя: {teacher.full\_name}. Теперь выберите группу по ID:\n{group\_list}")

await state.set\_state("waiting\_for\_group")

# Шаг 5: Обработка выбора группы по ID (для студента)

@dp.message\_handler(lambda message: message.text.isdigit(), state="waiting\_for\_group")

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

group\_id = int(message.text.strip())

group = session.query(Group).filter\_by(id=group\_id).first()

if not group:

await message.reply("Ошибка. Группа с таким ID не найдена.")

return

# Сохраняем выбор группы в состоянии

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

# Шаг 6: Завершаем регистрацию студента в группе

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

user = User(

telegram\_id=str(message.from\_user.id),

full\_name=user\_data.get('full\_name'),

role="student", # Студент

region=user\_data.get('region'),

city=user\_data.get('city'),

educational\_institution\_id=user\_data.get('educational\_institution\_id'), # Ссылка на образовательную организацию

group\_id=group.id # Привязка к группе

)

session.add(user)

session.commit() # Сохраняем студента в базе данных

# Завершаем процесс регистрации

await state.finish()

await message.reply(f"Вы успешно зарегистрированы как студент в группе {group.name}. Добро пожаловать!")

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

async def cmd\_create\_group(message: types.Message, state: FSMContext):

# Получаем пользователя из базы данных по telegram\_id

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

# Проверка прав доступа (только администратор или преподаватель)

if not user or user.role not in ["admin", "teacher"]:

await message.reply("У вас нет прав для создания группы.")

return

# Переход к состоянию для ввода названия группы

await state.set\_state("waiting\_for\_group\_name")

await message.reply("Введите название новой группы:")

# Обработчик для ввода названия группы

@dp.message\_handler(state="waiting\_for\_group\_name", content\_types=types.ContentTypes.TEXT)

async def process\_group\_name(message: types.Message, state: FSMContext):

group\_name = message.text.strip()

# Проверка, что название группы не пустое

if not group\_name:

await message.reply("Название группы не может быть пустым. Попробуйте снова.")

return

# Получаем данные о преподавателе (например, текущий пользователь)

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

# Проверка, что пользователь является преподавателем

if not user or user.role != "teacher":

await message.reply("Только преподаватели могут создавать группы.")

return

# Сохраняем название группы в состоянии

await state.update\_data(group\_name=group\_name) # Сохраняем название группы

# Создаем группу с привязкой к преподавателю

new\_group = Group(name=group\_name, password=None, teacher\_id=user.id)

session.add(new\_group)

session.commit()

await message.reply(f"Группа '{group\_name}' успешно создана без пароля.")

# Завершаем процесс создания группы

await state.finish()

# Обработчик для выбора пароля

"""

@dp.message\_handler(state="waiting\_for\_password\_choice", content\_types=types.ContentTypes.TEXT)

async def process\_password\_choice(message: types.Message, state: FSMContext):

password\_choice = message.text.strip().lower()

if password\_choice not in ["да", "нет"]:

await message.reply("Пожалуйста, ответьте 'да' или 'нет'.")

return

group\_name = (await state.get\_data())["group\_name"]

if password\_choice == "нет":

# Если без пароля, создаем группу без пароля

new\_group = Group(name=group\_name, password=None)

session.add(new\_group)

session.commit()

await message.reply(f"Группа '{group\_name}' успешно создана без пароля.")

else:

# Если с паролем, просим задать пароль или генерируем его

await state.set\_state("waiting\_for\_password")

await message.reply(f"Введите пароль для группы '{group\_name}' или отправьте 'генерировать' для автоматического пароля.")

"""

# Обработчик для ввода пароля

@dp.message\_handler(state="waiting\_for\_password", content\_types=types.ContentTypes.TEXT)

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

password = message.text.strip()

if password.lower() == 'генерировать':

# Генерация случайного пароля

password = generate\_password()

group\_name = (await state.get\_data())["group\_name"]

# Экранируем название группы и пароль для использования в Markdown

group\_name = escape\_markdown(group\_name)

password = escape\_markdown(password)

# Проверяем, существует ли группа с таким названием в базе данных

existing\_group = session.query(Group).filter\_by(name=group\_name).first()

if existing\_group:

await message.reply(f"Группа с названием '{group\_name}' уже существует. Пожалуйста, выберите другое название.")

return

# Сохраняем группу с паролем

new\_group = Group(name=group\_name, password=password)

session.add(new\_group)

try:

session.commit()

await message.reply(f"Группа '{group\_name}' успешно создана с паролем: {password}", parse\_mode=ParseMode.MARKDOWN)

except Exception as e:

# В случае ошибки с коммитом выводим сообщение

await message.reply("Произошла ошибка при создании группы. Пожалуйста, попробуйте снова.")

session.rollback()

# Завершаем процесс создания группы

await state.finish()

# Обработчик для изменения пароля

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

async def cmd\_change\_password(message: types.Message, state: FSMContext):

# Проверяем роль пользователя (должен быть преподавателем или администратором)

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user or user.role not in ["admin", "teacher"]:

await message.reply("У вас нет прав для изменения пароля группы.")

return

# Запросить название группы для изменения пароля

await state.set\_state("waiting\_for\_group\_name\_for\_password\_change")

await message.reply("Введите название группы, для которой хотите изменить пароль:")

# Обработчик для выбора группы для изменения пароля

@dp.message\_handler(state="waiting\_for\_group\_name\_for\_password\_change", content\_types=types.ContentTypes.TEXT)

async def process\_group\_name\_for\_password\_change(message: types.Message, state: FSMContext):

group\_name = message.text.strip()

# Проверяем, существует ли такая группа

group = session.query(Group).filter\_by(name=group\_name).first()

if not group:

await message.reply(f"Группа с названием '{group\_name}' не найдена.")

return

# Переход к состоянию для ввода нового пароля

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

await state.set\_state("waiting\_for\_new\_password")

await message.reply(f"Введите новый пароль для группы '{group\_name}'.")

# Обработчик для ввода нового пароля

@dp.message\_handler(state="waiting\_for\_new\_password", content\_types=types.ContentTypes.TEXT)

async def process\_new\_password(message: types.Message, state: FSMContext):

new\_password = message.text.strip()

group\_name = (await state.get\_data())["group\_name"]

# Обновляем пароль группы

group = session.query(Group).filter\_by(name=group\_name).first()

group.password = new\_password

session.commit()

await message.reply(f"Пароль для группы '{group\_name}' успешно изменен.", parse\_mode=ParseMode.MARKDOWN)

# Завершаем процесс изменения пароля

await state.finish()

# Обработчик для просмотра паролей (только для преподавателя или администратора)

"""

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

async def cmd\_view\_passwords(message: types.Message, state: FSMContext):

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user or user.role not in ["admin", "teacher"]:

await message.reply("У вас нет прав для просмотра паролей групп.")

return

# Получаем список групп и их паролей

groups = session.query(Group).all()

response = "Пароли групп:\n"

for group in groups:

response += f"Группа: {group.name}, Пароль: {group.password}\n"

await message.reply(response)

"""

# Присоединение ученика к группе

@dp.message\_handler(lambda message: not message.text.startswith("Группа: ") and len(message.text) == 8)

async def join\_group(message: types.Message):

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user or user.role != "student":

await message.reply("Эта команда доступна только для студентов.")

return

group = session.query(Group).filter\_by(password=None).first() # Ищем группу без пароля

if group:

user.group\_id = group.id

session.commit()

await message.reply(f"Вы присоединились к группе '{group.name}'.")

else:

await message.reply("Группа с таким паролем не найдена.")

# Создание задания

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

async def create\_task(message: types.Message):

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user or user.role != "teacher":

await message.reply("Эта команда доступна только для преподавателей.")

return

groups = session.query(Group).all()

if not groups:

await message.reply("Сначала создайте группу с помощью /create\_group.")

return

group\_list = "\n".join([f"{g.id}: {g.name}" for g in groups])

await message.reply(f"Выберите ID группы для задания:\n{group\_list}")

await CreateTask.waiting\_for\_group\_id.set()

@dp.message\_handler(state=CreateTask.waiting\_for\_group\_id, content\_types=types.ContentTypes.TEXT)

async def set\_task\_group(message: types.Message, state: FSMContext):

if message.text == "Назад":

await message.reply("Вы вернулись на начало.", reply\_markup=ReplyKeyboardMarkup(resize\_keyboard=True).add(KeyboardButton("Назад")))

await CreateTask.waiting\_for\_group\_id.set()

return

group\_id = int(message.text)

group = session.query(Group).filter\_by(id=group\_id).first()

if not group:

await message.reply("Группа с таким ID не найдена. Попробуйте ещё раз.")

return

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

await message.reply("Введите название задания:", reply\_markup=get\_back\_button())

await CreateTask.waiting\_for\_task\_name.set()

@dp.message\_handler(state=CreateTask.waiting\_for\_task\_name, content\_types=types.ContentTypes.TEXT)

async def set\_task\_name(message: types.Message, state: FSMContext):

if message.text == "Назад":

await message.reply("Вы вернулись на шаг выбора группы.", reply\_markup=get\_back\_button())

await CreateTask.waiting\_for\_group\_id.set()

return

await state.update\_data(task\_name=message.text.strip())

await message.reply("Введите описание задания:", reply\_markup=get\_back\_button())

await CreateTask.waiting\_for\_task\_description.set()

@dp.message\_handler(state=CreateTask.waiting\_for\_task\_description, content\_types=types.ContentTypes.TEXT)

async def set\_task\_description(message: types.Message, state: FSMContext):

if message.text == "Назад":

await message.reply("Вы вернулись на шаг ввода названия задания.", reply\_markup=get\_back\_button())

await CreateTask.waiting\_for\_task\_name.set()

return

await state.update\_data(task\_description=message.text.strip())

await message.reply("Введите дедлайн задания в формате YYYY-MM-DD HH:MM или выберите 'НЕТ ДЕДЛАЙНА'.", reply\_markup=get\_deadline\_keyboard())

await CreateTask.waiting\_for\_task\_deadline.set() # Переходим к вводу дедлайна

@dp.message\_handler(state=CreateTask.waiting\_for\_task\_deadline, content\_types=types.ContentTypes.TEXT)

async def set\_task\_deadline(message: types.Message, state: FSMContext):

if message.text == "Назад":

await message.reply("Вы вернулись на шаг ввода описания задания.", reply\_markup=get\_back\_button())

await CreateTask.waiting\_for\_task\_description.set()

return

elif message.text == "НЕТ ДЕДЛАЙНА":

# Устанавливаем дедлайн как None

await state.update\_data(task\_deadline=None)

await message.reply("Введите входные данные для теста:", reply\_markup=get\_input\_data\_keyboard())

await CreateTask.waiting\_for\_input\_data.set() # Переходим к вводу входных данных

return

try:

# Пытаемся преобразовать введённую строку в формат даты

deadline = datetime.strptime(message.text.strip(), "%Y-%m-%d %H:%M")

await state.update\_data(task\_deadline=deadline)

await message.reply("Введите входные данные для теста:", reply\_markup=get\_input\_data\_keyboard())

await CreateTask.waiting\_for\_input\_data.set() # Переходим к вводу входных данных

except ValueError:

# Если формат неправильный, отправляем ошибку

await message.reply("Неверный формат даты. Пожалуйста, введите дедлайн в формате YYYY-MM-DD HH:MM.")

@dp.message\_handler(state=CreateTask.waiting\_for\_expected\_result, content\_types=types.ContentTypes.TEXT)

async def save\_task(message: types.Message, state: FSMContext):

if message.text == "Назад":

await message.reply("Вы вернулись на шаг ввода входных данных.", reply\_markup=get\_back\_button())

await CreateTask.waiting\_for\_input\_data.set()

return

try:

# Получаем все данные из состояния

data = await state.get\_data()

# Проверяем, все ли необходимые данные присутствуют

if 'group\_id' not in data or 'task\_name' not in data or 'task\_description' not in data:

await message.reply("Не все данные были введены. Пожалуйста, вернитесь и заполните все поля.")

return

# Если входные данные нет, то присваиваем None

input\_data = data.get('input\_data', None)

expected\_result = message.text.strip() # Ожидаемый результат — это введённый текст

task = Task(

group\_id=data["group\_id"],

name=data["task\_name"],

description=data["task\_description"],

deadline=data["task\_deadline"], # Дедлайн из состояния

input\_data=input\_data, # Входные данные, если они есть

expected\_result=expected\_result # Ожидаемый результат

)

session.add(task)

session.commit()

await message.reply(f"Задание '{task.name}' успешно создано.", reply\_markup=ReplyKeyboardMarkup(resize\_keyboard=True).add(KeyboardButton("Назад")))

await state.finish()

except ValueError:

await message.reply("Произошла ошибка при создании задания.")

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

async def help\_command(message: types.Message):

help\_text = """

📚 \*\*Список доступных команд:\*\*

👨‍🏫 \*\*Для преподавателей:\*\*

- `/create\_group` — создать новую группу.

- `/create\_task` — создать задание для группы.

- `/watchgroup` — посмотреть участников группы.

- `/report` — получить отчёт о прогрессе учеников.

👨‍🎓 \*\*Для студентов:\*\*

- Введите пароль группы, чтобы присоединиться.

- `/submit` — отправить решение задания.

- `/view\_tasks` — просмотреть задания своей группы.

- `/task\_status` — узнать статус выполнения заданий.

- `/task\_deadline` — посмотреть дедлайн для заданий.

- `/task\_requirements` — получить требования для выполнения задания.

- `/my\_group` — информация о вашей группе.

ℹ️ \*\*Общие команды:\*\*

- `/help` — показать это сообщение.

- `/start` — начать работу с ботом.

💡 Если у вас возникли вопросы или проблемы, обратитесь к администратору.

"""

await message.reply(help\_text, parse\_mode="Markdown")

# Обработчик команды /watchgroup

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

async def watch\_group(message: types.Message):

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user or user.role != "teacher":

await message.reply("Эта команда доступна только для преподавателей.")

return

# Получение всех групп, созданных преподавателем

groups = session.query(Group).all()

if not groups:

await message.reply("Вы ещё не создали ни одной группы. Используйте /create\_group, чтобы создать группу.")

return

# Отправляем список групп преподавателю

group\_list = "\n".join([f"{group.id}: {group.name}" for group in groups])

await message.reply(f"Выберите ID группы, чтобы посмотреть её участников:\n{group\_list}")

await WatchGroup.waiting\_for\_group\_id.set()

# Обработчик выбора группы

@dp.message\_handler(state=WatchGroup.waiting\_for\_group\_id, content\_types=types.ContentTypes.TEXT)

async def show\_group\_members(message: types.Message, state: FSMContext):

try:

group\_id = int(message.text.strip())

group = session.query(Group).filter\_by(id=group\_id).first()

if not group:

await message.reply("Группа с таким ID не найдена. Попробуйте ещё раз.")

return

# Получение участников группы

members = session.query(User).filter\_by(group\_id=group.id).all()

if not members:

await message.reply(f"В группе '{group.name}' пока нет участников.")

else:

member\_list = "\n".join([f"- {member.full\_name}" for member in members])

await message.reply(f"Участники группы '{group.name}':\n{member\_list}")

except ValueError:

await message.reply("Пожалуйста, введите корректный ID группы (число).")

finally:

# Завершаем состояние

await state.finish()

@dp.message\_handler(lambda message: message.text.isdigit())

async def set\_task\_group(message: types.Message):

group\_id = int(message.text)

group = session.query(Group).filter\_by(id=group\_id).first()

if group:

await message.reply("Введите название задания:")

else:

await message.reply("Группа с таким ID не найдена.")

@dp.message\_handler(state=CreateTask.waiting\_for\_input\_data, content\_types=types.ContentTypes.TEXT)

async def set\_task\_input\_data(message: types.Message, state: FSMContext):

if message.text == "Назад":

await message.reply("Вы вернулись на шаг ввода дедлайна.", reply\_markup=get\_back\_button())

await CreateTask.waiting\_for\_task\_deadline.set() # Возвращаемся на шаг ввода дедлайна

return

elif message.text == "Входных данных нет":

await state.update\_data(input\_data=None) # Если входных данных нет, сохраняем None

await message.reply("Введите ожидаемый результат:", reply\_markup=get\_back\_button())

await CreateTask.waiting\_for\_expected\_result.set() # Переходим к вводу ожидаемого результата

return

# Если введены входные данные

await state.update\_data(input\_data=message.text.strip())

await message.reply("Введите ожидаемый результат:", reply\_markup=get\_back\_button())

await CreateTask.waiting\_for\_expected\_result.set() # Переходим к вводу ожидаемого результата

# Студентские команды

# Просмотр всех заданий для группы

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

async def view\_tasks(message: types.Message):

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user or user.role != "student":

await message.reply("Эта команда доступна только для студентов.")

return

tasks = session.query(Task).filter\_by(group\_id=user.group\_id).all()

if not tasks:

await message.reply("У вашей группы нет заданий.")

return

task\_list = "\n".join([f"{task.id}: {task.name} (Дедлайн: {task.deadline})" for task in tasks])

await message.reply(f"Доступные задания для вашей группы:\n{task\_list}")

# Информация о группе

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

async def my\_group(message: types.Message):

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user or user.role != "student":

await message.reply("Эта команда доступна только для студентов.")

return

if not user.group\_id:

await message.reply("Вы ещё не присоединились к группе.")

return

group = session.query(Group).filter\_by(id=user.group\_id).first()

members = session.query(User).filter\_by(group\_id=user.group\_id).all()

member\_list = "\n".join([f"- {member.full\_name}" for member in members])

await message.reply(f"Вы в группе '{group.name}'.\nУчастники группы:\n{member\_list}")

# Отправить решение задания

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

async def submit\_solution(message: types.Message):

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user or user.role != "student":

await message.reply("Эта команда доступна только для студентов.")

return

await message.reply("Отправьте ваше решение задания (например, файл или текст).")

# Статус выполнения задания

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

async def task\_status(message: types.Message):

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user or user.role != "student":

await message.reply("Эта команда доступна только для студентов.")

return

tasks = session.query(Task).filter\_by(group\_id=user.group\_id).all()

if not tasks:

await message.reply("У вашей группы нет заданий.")

return

status\_list = ""

for task in tasks:

status\_list += f"{task.name}: Статус выполнения - Не сдано\n" # Пример статуса

await message.reply(f"Статусы заданий:\n{status\_list}")

# Дедлайны для заданий

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

async def task\_deadline(message: types.Message):

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user or user.role != "student":

await message.reply("Эта команда доступна только для студентов.")

return

tasks = session.query(Task).filter\_by(group\_id=user.group\_id).all()

if not tasks:

await message.reply("У вашей группы нет заданий.")

return

deadline\_list = "\n".join([f"{task.name}: {task.deadline}" for task in tasks])

await message.reply(f"Дедлайны для заданий:\n{deadline\_list}")

# Требования к заданиям

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

async def task\_requirements(message: types.Message):

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user or user.role != "student":

await message.reply("Эта команда доступна только для студентов.")

return

tasks = session.query(Task).filter\_by(group\_id=user.group\_id).all()

if not tasks:

await message.reply("У вашей группы нет заданий.")

return

requirements\_list = "\n".join([f"{task.name}: {task.description}" for task in tasks])

await message.reply(f"Требования к заданиям:\n{requirements\_list}")

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

async def link\_group\_command(message: types.Message):

# Проверяем, есть ли пользователь в базе данных

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if not user:

await message.reply("Вы не зарегистрированы в системе. Пожалуйста, сначала завершите регистрацию.")

return

# Запрашиваем ID группы

await message.reply("Введите ID группы, к которой вы хотите привязаться:")

# Переходим к следующему состоянию для ввода ID группы

await Form.waiting\_for\_group\_id.set()

@dp.message\_handler(state=Form.waiting\_for\_group\_id)

async def process\_group\_id(message: types.Message, state: FSMContext):

group\_id = message.text.strip()

# Проверка на корректность введенного ID

if not group\_id.isdigit():

await message.reply("Пожалуйста, введите корректный ID группы.")

return

# Запись в базу данных или другие действия с group\_id

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

user.group\_id = group\_id

session.commit()

await message.reply(f"Вы успешно привязались к группе с ID: {group\_id}")

await state.finish()

# Обработчик для ввода ID группы

@dp.message\_handler(state="waiting\_for\_group\_id", content\_types=types.ContentTypes.TEXT)

async def save\_group\_id(message: types.Message, state: FSMContext):

group\_id = message.text.strip()

# Проверяем, существует ли группа в базе данных

group = session.query(Group).filter\_by(id=group\_id).first()

if not group:

await message.reply("Группа с таким ID не найдена. Пожалуйста, убедитесь, что ID введен корректно.")

return

# Привязываем пользователя к группе без запроса пароля

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

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

# Привязываем пользователя к группе

user.group\_id = group.id

session.commit()

# Завершаем процесс

await message.reply(f"Вы успешно привязаны к группе {group.name}. Пожалуйста, используйте команду /help для получения доступных вам команд.")

await state.finish()

# Обработчик для ввода пароля группы

@dp.message\_handler(state="waiting\_for\_group\_password", content\_types=types.ContentTypes.TEXT)

async def verify\_group\_password(message: types.Message, state: FSMContext):

# Получаем данные о группе из состояния

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

group\_id = user\_data.get('group\_id')

# Ищем группу в базе данных по ID

group = session.query(Group).filter\_by(id=group\_id).first()

if not group:

await message.reply("Ошибка! Группа не найдена.")

await state.finish()

return

# Привязываем пользователя к группе

user = session.query(User).filter\_by(telegram\_id=str(message.from\_user.id)).first()

if user:

user.group\_id = group.id

session.commit()

# Завершаем процесс

await message.reply(f"Вы успешно привязаны к группе {group.name}. Пожалуйста, используйте команду /help для получения доступных вам команд.")

await state.finish()

else:

await message.reply("Ошибка: Не удалось найти пользователя. Попробуйте снова.")

await state.finish()

# Запуск бота

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

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