Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс
«**Парадигмы и конструкции языков программирования**»

Отчет по домашнему заданию
«Telegram bot для создания кулинарных рецептов с использованием AI»

Выполнил:
студент группы ИУ5-31Б
Санников Н.А.

Проверил:
преподаватель каф. ИУ5

Москва, 2024 г.

Описание задания

Разработать бота для Telegram. Бот должен использовать искусственный интеллект для генерации рецептов по заданным пользователем параметрам.

Текст программы

#cook_pcpl_bot.py

```
from aiogram import Bot, Dispatcher, types
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton, BotCommand
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from aiogram.fsm.storage.memory import MemoryStorage
from aiogram import Router
from aiogram.filters import Command
from openai import OpenAI
import asyncio
import json

# Токены телеграм бота, OpenAI, а так же имя json файла
API_TOKEN = ' сюда api телеграм бота '
OPENAI_API_KEY = ' сюда api от OpenAI ChatGPT '
CHAT_DATA_FILE = 'active_chats.json'

# Настройка бота, диспетчера, маршрутизатора и клиента OpenAI
bot = Bot(token=API_TOKEN)
storage = MemoryStorage()
dp = Dispatcher(storage=storage)
router = Router()
dp.include_router(router)
client = OpenAI(api_key=OPENAI_API_KEY)

# Загружаем список chat_id из файла
def load_active_chats():
    try:
        with open(CHAT_DATA_FILE, 'r') as f:
            return set(json.load(f))
    except (FileNotFoundError, json.JSONDecodeError):
        return set()

# Сохраняет список chat_id в файл
def save_active_chats():
    with open(CHAT_DATA_FILE, 'w') as f:
        json.dump(list(active_chats), f)

# Загружает chat_id при старте
active_chats = load_active_chats()
```

```
# Отправляет запрос к OpenAI с настройками, которые помогают получить текст рецепта,
и возвращает его
async def generate_recipe(prompt, model="gpt-3.5-turbo", message=None):
    print("[INFO] Начинается генерация рецепта через OpenAI API.\n")
    try:
        response = client.chat.completions.create(
            model=model,
            messages=[{"role": "user", "content": prompt}],
            max_tokens=1500,
            temperature=0.7,
        )
        recipe = response.choices[0].message.content.strip()
        print("[SUCCESS] Рецепт успешно сгенерирован с помощью OpenAI API.\n")
        return recipe
    except Exception as e:
        print("[ERROR] Ошибка при генерации рецепта:", e, "\n")
        return "Произошла ошибка при генерации рецепта. Попробуйте снова."

# Группа состояний
class RecipeStates(StatesGroup):
    ModelChoice = State()
    Category = State()
    Taste = State()
    Texture = State()
    Aroma = State()
    Additional = State()

# Установка команд в меню бота при запуске
async def set_default_commands():
    await bot.set_my_commands([
        BotCommand(command="start", description="Начать работу с ботом"),
        BotCommand(command="help", description="Получить помощь"),
        BotCommand(command="info", description="Информация о боте")
    ])

# Обработчик нажатия кнопки "Начать"
@router.message(lambda message: message.text == "Начать")
async def handle_start_button(message: types.Message):
    # Симулируем отправку команды /start от имени пользователя
    await send_welcome(message)

# Создаем стартовую клавиатуру с кнопкой "Начать"
def get_start_keyboard():
    return ReplyKeyboardMarkup(
        keyboard=[[KeyboardButton(text="Начать")]],
        resize_keyboard=True
    )

# Функция для обновления клавиатуры при запуске бота
async def reset_keyboards_on_startup():
    for chat_id in active_chats:
        try:
```

```
            await bot.send_message(
                chat_id,
                text="Бот был перезапущен. Нажмите 'Начать', чтобы начать заново.",
                reply_markup=get_start_keyboard()
            )
        except Exception as e:
            print(f"[ERROR] Не удалось обновить клавиатуру для чата {chat_id}: {e}\n")

# Обработчик для команды /start в боте
@router.message(Command(commands=['start']))
async def send_welcome(message: types.Message):
    # Добавляем chat_id в список активных чатов и сохраняем
    active_chats.add(message.chat.id)
    save_active_chats()
    print("[START] Получена команда /start от пользователя. Отправляем приветственное сообщение.\n")
    keyboard = ReplyKeyboardMarkup(
        keyboard=[[KeyboardButton(text="Хорошо, задавай")]],
        resize_keyboard=True
    )
    await message.answer(
        "□□ Я помогу подобрать рецепт, учитывая все твои пожелания.\n"
        "□ Ответь на мои вопросы, и я подберу для тебя новое блюдо.\n"
        "□ Учти, что ты можешь не ограничиваться предложенными ответами — просто отправь свой вариант сообщением.",
        reply_markup=keyboard
        )
    print("[INFO] Стартовое сообщение отправлено.\n")

# Обработчик команды /help
@router.message(Command(commands=['help']))
async def send_help(message: types.Message, state: FSMContext):
    print("[HELP] Команда /help получена от пользователя.\n")

    # Сохраняем текущее состояние
    current_state = await state.get_state()
    await state.update_data(previous_state=current_state)  # Сохраняем предыдущее состояние в данные

    help_text = (
        "Команды бота:\n"
        "/start — Начать работу с ботом\n"
        "/help — Получить список команд и помощь\n"
        "/info — Узнать информацию о возможностях бота\n\n"
        "Этот бот помогает создавать персонализированные рецепты. "
        "Следуйте подсказкам, чтобы выбрать нужные параметры для рецепта."
    )

    # Клавиатура с кнопкой "Назад"
    keyboard = ReplyKeyboardMarkup(
        keyboard=[[KeyboardButton(text="Назад")]], resize_keyboard=True
    )
```

```
    # Отправляем сообщение с помощью и клавиатурой
    await message.answer(help_text, reply_markup=keyboard)
    print("[INFO] Сообщение помощи отправлено пользователю.\n")

# Обработчик команды /info
@router.message(Command(commands=['info']))
async def send_info(message: types.Message, state: FSMContext):
    print("[INFO] Команда /info получена от пользователя.\n")

    # Сохраняем текущее состояние
    current_state = await state.get_state()
    await state.update_data(previous_state=current_state)  # Сохраняем предыдущее состояние
в данные

    info_text = (
        "Этот бот позволяет создать рецепт, подходящий для ваших предпочтений. "
        "Вы можете указать категорию блюда, вкус, текстуру, аромат и дополнительные
пожелания. "
        "Бот использует модель GPT для генерации рецептов на основе ваших ответов. "
        "Просто начните с команды /start и следуйте инструкциям."
    )

    # Клавиатура с кнопкой "Назад"
    keyboard = ReplyKeyboardMarkup(
        keyboard=[[KeyboardButton(text="Назад")]], resize_keyboard=True
    )

    # Отправляем сообщение с информацией и клавиатурой
    await message.answer(info_text, reply_markup=keyboard)
    print("[INFO] Сообщение с информацией отправлено пользователю.\n")

# Обработчик нажатия на кнопку "Назад"
@router.message(lambda message: message.text == "Назад")
async def go_back(message: types.Message, state: FSMContext):
    print("[BACK] Пользователь нажал кнопку 'Назад'. Возвращаем в предыдущее
состояние.")

    # Получаем сохраненное предыдущее состояние
    data = await state.get_data()
    previous_state = data.get("previous_state")

    # Восстанавливаем предыдущее состояние
    if previous_state:
        await state.set_state(previous_state)

        # Отправляем сообщение, связанное с предыдущим состоянием
        if previous_state == RecipeStates.ModelChoice.state:
            await choose_model(message, state)  # Переход к выбору модели
        elif previous_state == RecipeStates.Category.state:
            await process_category(message, state)  # Переход к выбору категории
        elif previous_state == RecipeStates.Taste.state:
```

```
            await process_taste(message, state)  # Переход к выбору вкуса
        elif previous_state == RecipeStates.Texture.state:
            await process_texture(message, state)  # Переход к выбору текстуры
        elif previous_state == RecipeStates.Aroma.state:
            await process_aroma(message, state)  # Переход к выбору аромата
        elif previous_state == RecipeStates.Additional.state:
            await process_additional(message, state)  # Переход к дополнительным пожеланиям

    else:
        await message.answer("Предыдущее состояние не найдено. Пожалуйста, начните с
/start.")

# Обработчик для сообщения "Хорошо, задавай" в боте
@router.message(lambda message: message.text == "Хорошо, задавай")
async def choose_model(message: types.Message, state: FSMContext):
    print("[STEP] Пользователь подтвердил готовность. Переход к выбору модели.\n")
    buttons = [
        [KeyboardButton(text="gpt-3.5-turbo")],
        [KeyboardButton(text="gpt-4")]
    ]
    keyboard = ReplyKeyboardMarkup(keyboard=buttons, resize_keyboard=True)
    await state.set_state(RecipeStates.ModelChoice)
    await message.answer("Выберите модель для генерации рецепта:",
reply_markup=keyboard)
    print("[INFO] Запрос на выбор модели отправлен пользователю.\n")

# Обрабатывает выбор модели для генерации рецепта и переводит пользователя на
следующий этап — выбор категории блюда
@router.message(RecipeStates.ModelChoice)
async def set_model_and_start(message: types.Message, state: FSMContext):
    chosen_model = message.text

    # Проверяем, выбрана ли корректная модель, или используем модель из прошлого
запроса
    if chosen_model not in ["gpt-3.5-turbo", "gpt-4"]:
        data = await state.get_data()
        chosen_model = data.get("model", "gpt-3.5-turbo")  #Используем модель из прошлого
запроса
        print(f"[INFO] Выбор модели пропущен. Используем модель из прошлого запроса:
{chosen_model}\n")
    else:
        print(f"[SELECTED] Пользователь выбрал модель: {chosen_model}\n")

    await state.update_data(model=chosen_model)

    buttons = [
        [KeyboardButton(text="Основное блюдо")],
        [KeyboardButton(text="Закуска")],
        [KeyboardButton(text="Десерт")],
        [KeyboardButton(text="Напиток")],
        [KeyboardButton(text="Салат")],
        [KeyboardButton(text="Суп")],
```

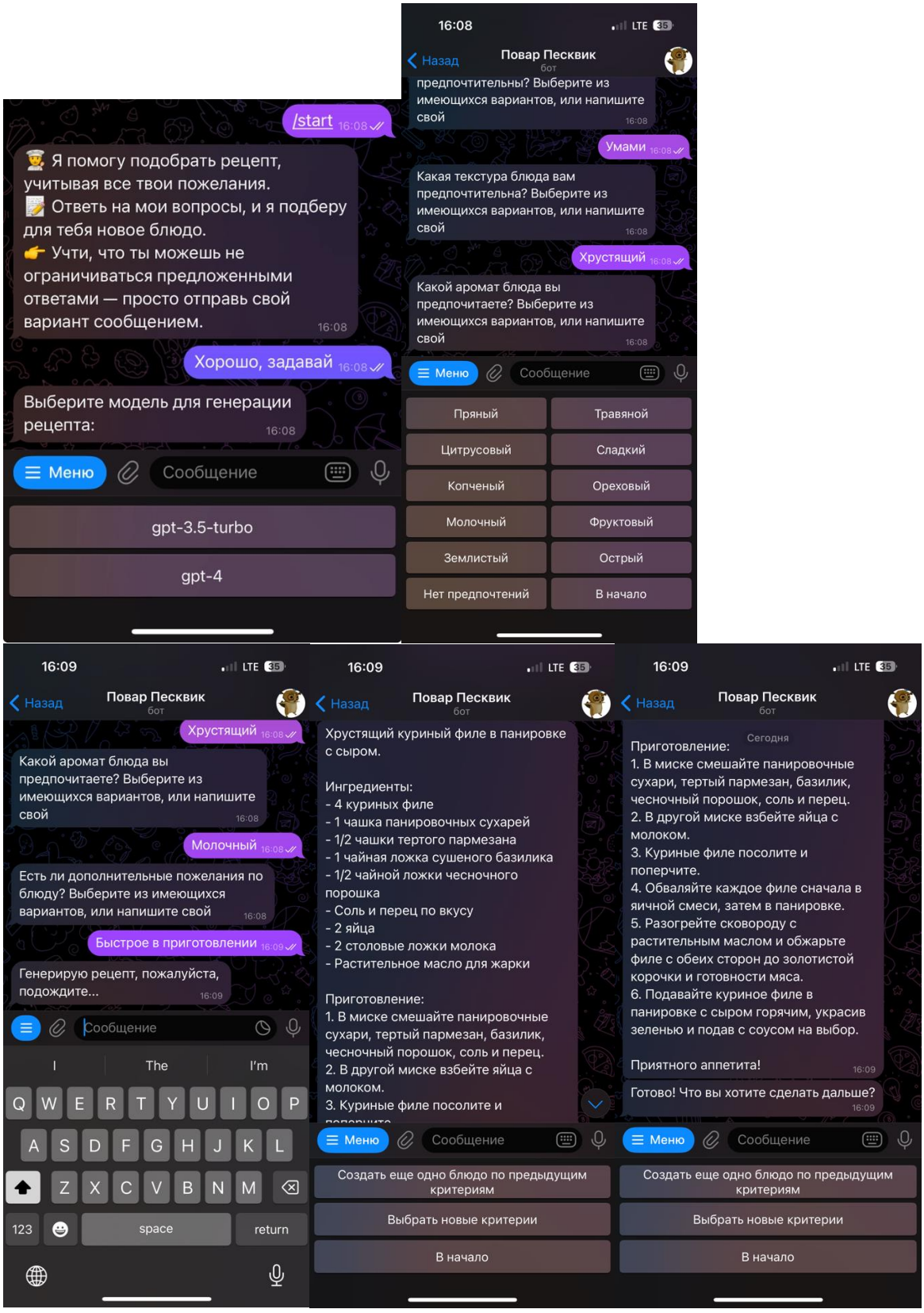
```
        [KeyboardButton(text="Соус")],
        [KeyboardButton(text="Гарнир")],
        [KeyboardButton(text="Выпечка")],
        [KeyboardButton(text="Нет предпочтений")],
        [KeyboardButton(text="В начало")]
    ]
    keyboard = ReplyKeyboardMarkup(keyboard=buttons, resize_keyboard=True)
    await state.set_state(RecipeStates.Category)
    await message.answer("Категория блюда? Выберите из имеющихся вариантов, или
напишите свой", reply_markup=keyboard)
    print("[INFO] Запрос на выбор категории блюда отправлен пользователю.\n")

# Обрабатывает выбор категории для генерации рецепта и переводит пользователя на
следующий этап — выбор вкуса блюда
@router.message(RecipeStates.Category)
async def process_category(message: types.Message, state: FSMContext):
    if message.text == "В начало":
        print("[ACTION] Пользователь решил вернуться к началу.\n")
        await send_welcome(message)
        return

    category = message.text
    await state.update_data(category=category)
    print(f"[CATEGORY SELECTED] Пользователь выбрал категорию блюда: {category}\n")

    if category == "Напиток":
        print("[INFO] Категория 'Напиток' выбрана - Пропускаем выбор текстуры, переходим
к выбору аромата.\n")
        await state.set_state(RecipeStates.Aroma)
        await ask_aroma(message)
    else:
        buttons = [
            [KeyboardButton(text="Сладкий"), KeyboardButton(text="Солёный")],
            [KeyboardButton(text="Кислый"), KeyboardButton(text="Горький")],
            [KeyboardButton(text="Умами"), KeyboardButton(text="Острый")],
            [KeyboardButton(text="Пряный"), KeyboardButton(text="Терпкий")],
            [KeyboardButton(text="Свежий"), KeyboardButton(text="Нет предпочтений")],
            [KeyboardButton(text="В начало")]
        ]
        keyboard = ReplyKeyboardMarkup(keyboard=buttons, resize_keyboard=True)
        await state.set_state(RecipeStates.Taste)
        await message.answer("Какие вкусовые характеристики вам предпочтительны?
Выберите из имеющихся вариантов, или напишите свой", reply_markup=keyboard)
        print("[INFO] Запрос на выбор вкусовых характеристик отправлен пользователю.\n")

# Обрабатывает выбор вкуса блюда для генерации рецепта и переводит пользователя на
следующий этап — выбор текстуры
@router.message(RecipeStates.Taste)
async def process_taste(message: types.Message, state: FSMContext):
    if message.text == "В начало":
        print("[ACTION] Пользователь решил вернуться к началу.\n")
        await send_welcome(message)
```

```
        return

    await state.update_data(taste=message.text)
    print(f"[TASTE SELECTED] Пользователь выбрал вкус: {message.text}\n")
    category = (await state.get_data()).get("category")

    if category == "Десерт" or category == "Основное блюдо":
        buttons = [
            [KeyboardButton(text="Хрустящий"), KeyboardButton(text="Нежный")],
            [KeyboardButton(text="Мягкий"), KeyboardButton(text="Плотный")],
            [KeyboardButton(text="Воздушный"), KeyboardButton(text="Жевательный")],
            [KeyboardButton(text="Кремовый"), KeyboardButton(text="Сочный")],
            [KeyboardButton(text="Тягучий"), KeyboardButton(text="Хлопьевидный")],
            [KeyboardButton(text="Нет предпочтений"), KeyboardButton(text="В начало")]
        ]
        keyboard = ReplyKeyboardMarkup(keyboard=buttons, resize_keyboard=True)
        await state.set_state(RecipeStates.Texture)
        await message.answer("Какая текстура блюда вам предпочтительна? Выберите из
имеющихся вариантов, или напишите свой", reply_markup=keyboard)
        print("[INFO] Запрос на выбор текстуры блюда отправлен пользователю.\n")
    else:
        print("[INFO] Категория не требует выбора текстуры. Переход к выбору аромата.\n")
        await state.set_state(RecipeStates.Aroma)
        await ask_aroma(message)

# Обрабатывает выбор текстуры для генерации рецепта и переводит пользователя на
следующий этап — выбор аромата
@router.message(RecipeStates.Texture)
async def process_texture(message: types.Message, state: FSMContext):
    if message.text == "В начало":
        print("[ACTION] Пользователь решил вернуться к началу.\n")
        await send_welcome(message)
        return

    await state.update_data(texture=message.text)
    print(f"[TEXTURE SELECTED] Пользователь выбрал текстуру: {message.text}\n")
    await state.set_state(RecipeStates.Aroma)
    await ask_aroma(message)

async def ask_aroma(message: types.Message):
    buttons = [
        [KeyboardButton(text="Пряный"), KeyboardButton(text="Травяной")],
        [KeyboardButton(text="Цитрусовый"), KeyboardButton(text="Сладкий")],
        [KeyboardButton(text="Копченый"), KeyboardButton(text="Ореховый")],
        [KeyboardButton(text="Молочный"), KeyboardButton(text="Фруктовый")],
        [KeyboardButton(text="Землистый"), KeyboardButton(text="Острый")],
        [KeyboardButton(text="Нет предпочтений"), KeyboardButton(text="В начало")]
    ]
    keyboard = ReplyKeyboardMarkup(keyboard=buttons, resize_keyboard=True)
    await message.answer("Какой аромат блюда вы предпочитаете? Выберите из имеющихся
вариантов, или напишите свой", reply_markup=keyboard)
    print("[INFO] Запрос на выбор аромата отправлен пользователю.\n")
```

```
# Обрабатывает выбор аромата для генерации рецепта и переводит пользователя на следующий этап — выбор доп. пожеланий
@router.message(RecipeStates.Aroma)
async def process_aroma(message: types.Message, state: FSMContext):
    if message.text == "В начало":
        print("[ACTION] Пользователь решил вернуться к началу.\n")
        await send_welcome(message)
        return

    await state.update_data(aroma=message.text)
    print(f"[AROMA SELECTED] Пользователь выбрал аромат: {message.text}\n")
    buttons = [
        [KeyboardButton(text="Лёгкое"), KeyboardButton(text="Сытное")],
        [KeyboardButton(text="Необычное"), KeyboardButton(text="Классическое")],
        [KeyboardButton(text="Полезное"), KeyboardButton(text="Быстрое в приготовлении")],
        [KeyboardButton(text="Дополнительных пожеланий нет"), KeyboardButton(text="В начало")]
    ]
    keyboard = ReplyKeyboardMarkup(keyboard=buttons, resize_keyboard=True)
    await state.set_state(RecipeStates.Additional)
    await message.answer("Есть ли дополнительные пожелания по блюду? Выберите из имеющихся вариантов, или напишите свой", reply_markup=keyboard)
    print("[INFO] Запрос на дополнительные пожелания отправлен пользователю.\n")

# Обрабатывает выбор доп. пожеланий для генерации рецепта и переводит пользователя на следующий этап — генерацию
@router.message(RecipeStates.Additional)
async def process_additional(message: types.Message, state: FSMContext):
    if message.text == "В начало":
        print("[ACTION] Пользователь решил вернуться к началу.\n")
        await send_welcome(message)
        return

    if message.text == "Выбрать новые критерии":
        print("[ACTION] Пользователь выбрал: Выбрать новые критерии.\n")
        data = await state.get_data()

        # Проверка и сохранение корректного значения модели
        chosen_model = data.get("model", "gpt-3.5-turbo")
        if chosen_model not in ["gpt-3.5-turbo", "gpt-4"]:
            chosen_model = "gpt-3.5-turbo"  # Назначаем значение по умолчанию, если модель недействительна
        await state.update_data(model=chosen_model)

        # Переход к выбору категории блюда, модель остается неизменной
        await set_model_and_start(message, state)
        return

    update_additional = message.text != "Создать еще одно блюдо по предыдущим критериям"
    if update_additional:
        await state.update_data(additional=message.text)
```

```
    data = await state.get_data()
    print("[INFO] Все данные собраны, подготовка к генерации рецепта...\n")

    prompt = (
        f"Составь рецепт. "
        f"Категория блюда: {data['category']}. "
        f"Желаемый вкус: {data.get('taste', 'не указан')}. "
        f"Предпочтительная текстура: {data.get('texture', 'не указано')}. "
        f"Аромат, который хотелось бы получить: {data['aroma']}. "
        f"Дополнительные пожелания по блюду: {data['additional']}. "
        f"В одном своем ответе составляй только один рецепт! "
    )

    await message.answer("Генерирую рецепт, пожалуйста, подождите...",
reply_markup=types.ReplyKeyboardRemove())
    model = data.get('model', "gpt-3.5-turbo")  # Использовать выбранную модель
    try:
        recipe = await generate_recipe(prompt, model=model, message=message)
    except Exception as e:
        print("[ERROR] Ошибка при генерации рецепта:", e, "\n")
        recipe = "Произошла ошибка при генерации рецепта. Попробуйте снова."

    print(f"[INFO] Рецепт сгенерирован с использованием модели {model}. \n\n[PROMPT]
Промпт:\n\n{prompt} \n\n[RESULT] Результат:\n\n{recipe}\n\n")

    await message.answer(recipe)
    print("[INFO] Рецепт отправлен пользователю.\n")

    keyboard = ReplyKeyboardMarkup(
        keyboard=[
            [KeyboardButton(text="Создать еще одно блюдо по предыдущим критериям")],
            [KeyboardButton(text="Выбрать новые критерии")],
            [KeyboardButton(text="В начало")]
        ],
        resize_keyboard=True
    )
    await message.answer("Готово! Что вы хотите сделать дальше?", reply_markup=keyboard)

async def main():
    await set_default_commands()
    print("[BOOT] Команды по умолчанию установлены\n")
    await reset_keyboards_on_startup()
    print("[BOOT] Клавиатуры пользователей обновлены\n")
    print("[BOOT] Бот готов к работе\n")
    await dp.start_polling(bot)

if __name__ == '__main__':
    print("\n[BOOT] Запуск бота...\n")
    asyncio.run(main())
```

Экранные формы
с примерами выполнения программы



**Программа так же уведомляет о действиях пользователя в консоль