Автономная некоммерческая организация Дополнительного профессионального образования

Компьютерная Академия «ТОП» филиал «Академия ТОП Уфа»

# Дипломная работа

по курсу «Веб-разработка на Python»

«Разработка Telegram-бота для автоматизации системы заказа еды»

Выполнил студент Компьютерной Академии ТОП
Мухамадеев Айдар Илдарович /_____

Дипломная работа допущена к защите и проверена на объем заимствования:

Директор филиала
АНО ДПО «Академия ТОП»
Игнатьева Азалия Фаритовна\ _____

Рук. учебной части филиала
АНО ДПО «Академия
Фатхинурова Светлана Форагатовна_____

Уфа, 2024г.

**ОГЛАВЛЕНИЕ**

ВВЕДЕНИЕ
ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ..... 4
1.1. Обзор рынка Telegram-ботов ..... …. 4
1.2. Функциональные требования к чат-боту ..... 5
ГЛАВА 2. ОБЗОР ТЕХНОЛОГИЙ..... 6
2.1. Архитектура разрабатываемого программного обеспечения ..... 6
2.2. Функциональные требования ..... 14
ГЛАВА 3. РЕАЛИЗАЦИЯ..... 17
3.1. Структура проекта ..... 17
3.2. Реализация функционала..... 20
3.3. Тестирование и отладка..... 23
ЗАКЛЮЧЕНИЕ

# ВВЕДЕНИЕ

В условиях стремительного развития технологий и растущей популярности онлайн-сервисов, автоматизация процессов становится всё более актуальной. Настоящая дипломная работа посвящена разработке чат-бота для автоматизации системы заказа еды, призванного повысить эффективность и удобство взаимодействия между клиентами и заведениями общественного питания.

В работе рассматриваются вопросы проектирования, разработки и внедрения интеллектуальной системы на основе современных технологий обработки естественного языка и машинного обучения, позволяющей автоматизировать процесс приема заказов, обработки платежей и предоставления информации о меню и статусе заказа. Актуальность выбранной темы обусловлена потребностью в оптимизации работы служб доставки и ресторанов, а также повышении удовлетворенности клиентов за счет быстрого и удобного интерфейса. В рамках исследования будут проанализированы существующие решения в данной области, описаны этапы разработки собственного чат-бота, представлены результаты тестирования и оценки его эффективности.

Бот должен обеспечивать простой интерфейс взаимодействия с пользователями, предоставляя информацию о меню, а также возможность выбора блюд и оформления заказа в режиме реального времени. Данная система позволит повысить эффективность процесса заказа, уменьшить время ожидания ответов и улучшить общее качество обслуживания клиентов. Бот реализован на языке программирования Python с использованием фреймворка Django и СУБД PostgreSQL.

# ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1. Обзор рынка Telegram-ботов

Анализ рынка сервисов доставки еды и онлайн-платформ для заказа выявил широкое распространение систем, частично или полностью использующих чат-ботов для автоматизации взаимодействия с клиентами. Эти системы демонстрируют различные подходы к реализации, отличающиеся по уровню сложности, функциональности и используемым технологиям. Можно выделить несколько категорий существующих решений:

- Многие приложения для заказа еды используют чат-ботов для упрощения процесса заказа, предоставления информации о статусе доставки и решения простых вопросов. Эти боты, как правило, обладают ограниченным функционалом и опираются на заранее запрограммированные сценарии. Недостатком таких решений часто является неспособность обрабатывать нестандартные запросы или сложные ситуации, что требует вмешательства операторов.
- Чат-боты, работающие в мессенджерах: Некоторые сервисы предлагают взаимодействие с клиентами через популярные мессенджеры, такие как Telegram, WhatsApp или Facebook Messenger. Это позволяет пользователям заказывать еду в привычной для них среде.
- Гибридные системы: Некоторые сервисы используют комбинацию автоматизированных и ручных процессов. Чат-бот отвечает на вопросы и обрабатывает стандартные заказы, а операторы принимают участие в решении сложных проблем или обработке нестандартных запросов. Такой подход позволяет обеспечить баланс между автоматизацией и качеством обслуживания.

Анализ существующих решений показал, что, несмотря на значительный прогресс в области разработки чат-ботов, многие системы страдают от ограниченного функционала, проблем с распознаванием естественного языка, недостаточно интуитивного интерфейса, длительного времени ожидания

ответа и отсутствия возможности обработки сложных или нестандартных запросов. Это указывает на необходимость разработки более совершенных и гибких систем, которые бы обеспечивали высокое качество обслуживания и удовлетворяли потребности как пользователей, так и заведений общественного питания.

## 1.2. Функциональные требования к чат-боту.

Разрабатываемый чат-бот должен обеспечивать следующие основные функции:

- Прием заказов: автоматическое распознавание блюд и напитков из меню, обработка количества и модификаций заказов.
- Обработка платежей: интеграция с платежными системами для безопасной и удобной оплаты.
- Управление доставкой: интеграция с сервисами доставки, отслеживание статуса заказа, уведомления о времени доставки.
- Предоставление информации: доступ к меню, информации о ценах, акциях и специальных предложениях.
- Обработка запросов: ответы на часто задаваемые вопросы, решение проблемных ситуаций (например, изменение или отмена заказа).
- Удобство использования: интуитивный и понятный интерфейс, быстрая скорость ответа.
- Совместимость: работа с различными платформами и устройствами (мобильные приложения, веб-сайты, мессенджеры).

# ГЛАВА 2. ОБЗОР ТЕХНОЛОГИЙ

## 2.1. Архитектура разрабатываемого программного обеспечения

Архитектура разрабатываемого Telegram-бота для автоматизации системы заказа еды будет основана на клиент-серверной модели с использованием многоуровневого подхода и фреймворка Django. Это обеспечит масштабируемость, поддерживаемость и гибкость системы. Основные компоненты архитектуры:

1. Telegram Bot API: Интерфейс программирования приложений (API) Telegram, предоставляющий доступ к функциональности Telegram для создания ботов. Бот будет использовать этот API для взаимодействия с пользователями, приема сообщений и отправки ответов, включая кнопки и инлайн-меню для улучшения пользовательского опыта (UX).

2. Серверная часть (Django): Серверная часть будет реализована с использованием фреймворка Django на языке Python и будет отвечать за обработку запросов от бота, взаимодействие с базой данных PostgreSQL и внешними сервисами. Она будет состоять из следующих модулей:

Модуль обработки сообщений (views.py): Этот модуль, реализованный с использованием Django views, будет получать сообщения от Telegram Bot API (через библиотеки python-telegram-bot или aiogram), распознавать намерения пользователя (с использованием NLP-модуля, если таковой применяется) и определять дальнейшие действия, обрабатывая навигацию по инлайн-меню и кнопкам.

Модуль управления заказами (models.py, forms.py): Этот модуль, используя Django models и forms, будет отвечать за создание, обработку и отслеживание заказов. Он будет взаимодействовать с базой данных PostgreSQL для хранения информации о заказах и меню.

Модуль интеграции с платежными системами: Этот модуль обеспечит безопасную обработку платежей с использованием API выбранной платежной системы. Интеграция будет осуществлена через соответствующие библиотеки Python.

Модуль интеграции с сервисами доставки: Этот модуль будет взаимодействовать с API выбранного сервиса доставки (например, Delivery Club API) для организации доставки заказов. Интеграция будет осуществлена через соответствующие библиотеки Python.

Модуль базы данных (PostgreSQL): База данных PostgreSQL будет хранить информацию о пользователях, меню, заказах, и других необходимых данных. Django ORM обеспечит удобное взаимодействие с базой данных.

Модуль логирования: Этот модуль будет регистрировать все действия бота и важные события для отладки и мониторинга.

3. Клиентская часть (Telegram-клиент): Это приложение пользователя, через которое он взаимодействует с ботом. В данном случае, это сам Telegram-клиент, взаимодействующий с ботом через удобный интерфейс, реализованный с помощью кнопок и инлайн-меню.

4. Взаимодействие компонентов:

Пользователь отправляет сообщение или взаимодействует с кнопками/инлайн-меню в Telegram.

Telegram Bot API передает информацию на сервер Django.

Django обрабатывает запрос, используя соответствующие модули и взаимодействуя с базой данных PostgreSQL.

Сервер отправляет ответ (включая кнопки и инлайн-меню) через Telegram Bot API обратно пользователю.

Данная архитектура, основанная на Django, позволяет эффективно использовать возможности этого фреймворка для разработки надежного, масштабируемого и поддерживаемого Telegram-бота. Использование PostgreSQL в качестве базы данных обеспечивает высокую производительность и надежность хранения данных. Выбор библиотек python-telegram-bot или aiogram обеспечивает удобное взаимодействие с Telegram Bot API. Применение кнопок и инлайн-меню значительно улучшает UX, делая взаимодействие с ботом интуитивно понятным.

Таким образом, выбранные технологии создают надёжную и эффективную основу для разработки бота, обеспечивая его функциональность и высокое качество пользовательского опыта.

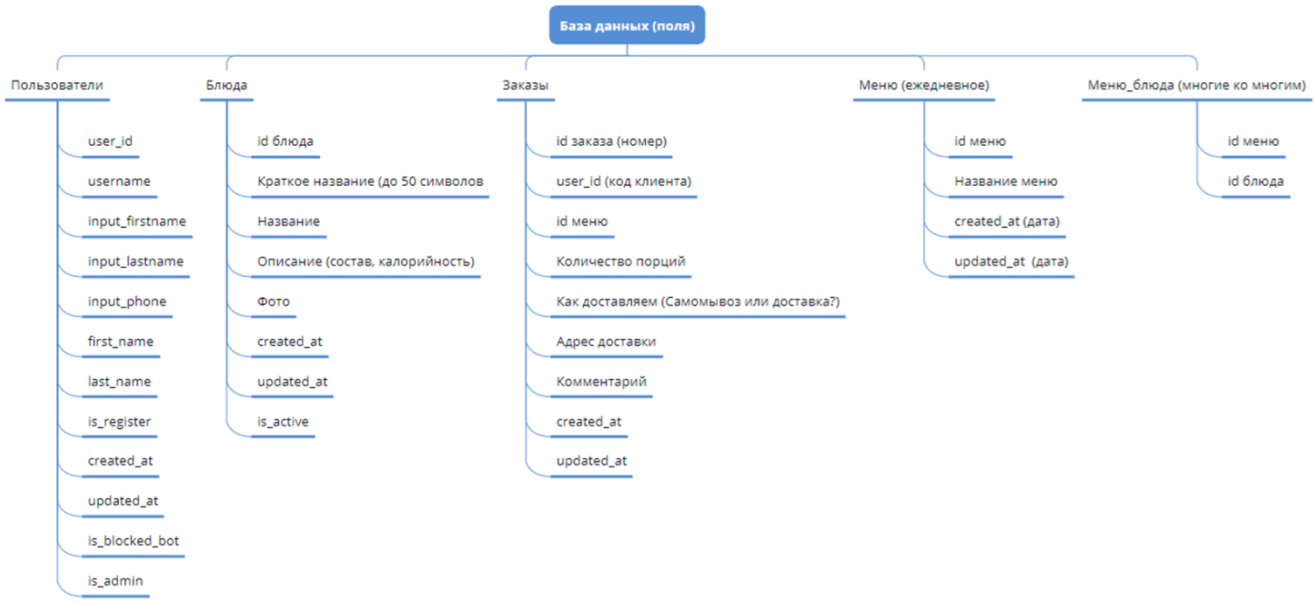



Рисунок 1. Схема базы данных

Схема на изображении описывает структуру базы данных, состоящей из нескольких таблиц:

Пользователи: Таблица содержит информацию о пользователях приложения, включая их идентификатор (user_id), имя пользователя

(username), имя и фамилию ( input_firstname, input_lastname, first_name, last_name), номер телефона (input_phone), флаги регистрации (is_register), активности (is_active), блокировки ботом (is_blocked_bot) и администраторских прав (is_admin), а также метки времени создания и обновления записи (created_at, updated_at).

Блюда: Таблица описывает каждое блюдо в меню, с полями для идентификатора (id блюда), краткого и полного названия (Краткое название, Название), описания ( Описание), пути к изображению (Фото) и метками времени создания и обновления записи (created_at, updated_at). Поле is_active вероятно указывает на доступность блюда в меню.

Меню (ежедневное): Таблица содержит информацию о ежедневном меню, включая идентификатор (id меню), название (Название меню) и метки времени создания и обновления (created_at, updated_at).

Заказы: Таблица хранит информацию о заказах, включая идентификатор (id заказа), идентификатор пользователя (user_id), идентификатор меню (id меню), количество порций (Количество порций), способ доставки (Как доставляем), адрес доставки (Адрес доставки), комментарии (Комментарий) и метки времени создания и обновления (created_at, updated_at).

Меню_блюда: Это связующая таблица, реализующая отношение "многие ко многим" между таблицами "Меню (ежедневное)" и "Блюда". Она содержит идентификаторы меню (id меню) и блюда (id блюда), позволяя одному меню включать в себя множество блюд.

1. handlers

Приветствует пользователя в клиентском разделе и показывает правила.

Отправляет текстовые сообщения и клавиатуру. Устанавливает состояние ожидания обработки правил.

**client_handling_rules:**

Обрабатывает разрешение пользователя на получение сообщений. Если пользователь согласен, вызывается функция user_registration для регистрации, и состояние изменяется на ожидание информации о заказе. Если пользователь отказывается, состояние очищается, и отправляется сообщение об отказе.

**client_msg_before_start:**

Отправляет приветственное сообщение перед началом взаимодействия с ботом.

**client_if_order:**

Запрашивает у клиента, хочет ли он заказывать еду на завтра.

Обрабатывает ответ: если «Да», то переходит к началу заказа; если «Нет», обрабатывает отказ.

**client_cancel_order:**

Обрабатывает отказ от заказа, отправляя соответствующее сообщение.

**client_start_order:**

Если пользователь согласен заказывать, бот запрашивает количество порций.

**client_count_of_serv:**

Обрабатывает количество порций и проверяет тип оплаты.

Вызывает проверку количества порций, и, если оно больше 100, повторно запрашивает количество.

**check_count_of_servings:**

Валидация введенного количества порций с использованием регулярного выражения.

Если количество неправильно, отправляет уведомление и возвращается к началу заказа.

**client_sel_payment:**

Обрабатывает выбор способа оплаты (картой или наличными).

Обновляет соответствующее состояние и включает клавиатуру для выбора доставки.

**client_payment_cash:**

Производит валидацию введенной суммы при оплате наличными.

Обрабатывает ситуации, если введенное значение не соответствует ожиданиям (используя регулярные выражения).

**client_check_order:**

Описание: Сохранение заказа в базе данных, если все введенные данные верны. Если пользователь подтверждает правильность данных, они собираются в словарь order_data и отправляются в функцию add_order для добавления заказа в базу данных.

Если добавление прошло успешно (код статуса 201), отправляется подтверждение заказа.

Если возникла ошибка при добавлении, отправляется сообщение об ошибке, с возможностью попробовать снова.

Если пользователь хочет исправить данные, возвращается к началу процесса создания заказа.

Эти функции взаимодействуют с пользователем, позволяя ему сделать полноценный заказ, проверить введенные данные и обработать финальные шаги, связанные с оформлением заказа. Они обеспечивают последовательность и контроль за процессом, а также поддержку состояния пользователя в процессе взаимодействия с ботом.

2. keyboards

Каждая функция создает кнопки для различных операций администратора, что упрощает управление ботом и взаимодействие с пользователями. Клавиатуры обеспечивают удобный интерфейс для выполнения административных задач в рамках бота. Клавиатуры, возвращающие ReplyKeyboardMarkup, автоматически настраиваются под размеры экрана устройства, так как задан параметр resize_keyboard=True, что улучшает пользовательский опыт.

Клавиатуры Администратора

**admin_check_accept_kb():**

Создаёт клавиатуру с одной кнопкой (без текста).

**admin_main_menu_kb():**

Возвращает разметку клавиатуры с одной кнопкой. Создаёт главное меню с кнопками для работы с меню, блюдами и заказами. Возвращает клавиатуру с тремя кнопками.

**admin_work_with_orders_kb():**

Работа с заказами. Создаёт клавиатуру с кнопками для отображения списка заказов и для возврата в главное меню.

**admin_choose_orders_kb():**

Создаёт клавиатуру с кнопками для выбора заказа по дате (сегодня, завтра, выбор по дате) и возвратом в главное меню.

**admin_work_with_dishes_kb():**

Работа с блюдами.

**admin_add_dish_noimage_kb():**

Создаёт клавиатуру для управления блюдами: просмотра списка, добавления, изменения и удаления блюд.

**admin_add_dish_check_kb():**

Клавиатура для выбора блюда без фото.

**admin_start_repair_dish_kb():**

Подтверждение корректности введённых данных с выбором между подтверждением или исправлением.

**admin_remove_dish_choose_kb():**

Клавиатура для выбора параметров, которые нужно изменить (название, краткое название, описание, фотография).

**admin_work_with_menus_kb():**

Работа с меню. Подтверждение удаления блюда с выбором между подтверждением и отменой.

**admin_add_menus_confirm_kb():**

Создаёт клавиатуру для управления меню: создание нового меню, удаление, просмотр созданных меню и массовая рассылка.

**admin_remove_menu_choose_kb():**

Подтверждение правильности данных о меню.

**admin_confirm_send_kb():**

Подтверждение удаления меню: подтверждение или отмена.

3. phrases

Данный раздел определяет р фразы, используемые в взаимодействии с пользователем для Telegram-бота, который может принимать заказы на еду. Эти фразы представляют собой тексты, которые бот отправляет пользователю на различных этапах процесса взаимодействия, от регистрации до оформления заказа. Рассмотрим на каждую секцию и её назначение.

Регистрация Пользователя

**check_registration:**

CHECK_REG_1: Сообщение для зарегистрированных пользователей с инструкциями по переходу к заказам.

CHECK_REG_2: Уведомление для незарегистрированных пользователей с просьбой зарегистрироваться.

**start_registration:**

START_REG_1: Направляющее сообщение для начала процесса регистрации.

START_REG_2: Запрос имени пользователя.

**input_firstname:**

INPUT_NAME_1: Ошибка, если имя введено неверно (допускаются только буквы и тире).

INPUT_NAME_2: Запрос фамилии после введения имени.

**input_lastname:**

INPUT_LASTNAME_1: Ошибка, если фамилия введена неверно.

INPUT_LASTNAME_2: Запрос номера телефона.

**input_phone:**

INPUT_PHONE_1: Ошибка, если телефон введен в неверном формате.

INPUT_PHONE_2: Подтверждение правильности ввода номера телефона.

**confirm_registration:**

CONFIRM_REG_1: Сообщение об успешной регистрации и шаге оформления заказа.

CONFIRM_REG_2: Предложение повторить процесс, если возникла ошибка.

CONFIRM_REG_ERR: Сообщение об ошибке в процессе регистрации.

Основной Раздел Пользователя

**enter_client_section:**

ENTER_CLIENT_1: Приветственное сообщение с указанием времени для раздумий о заказе.

ENTER_CLIENT_2: Запрос согласия с правилами.

**client_handling_rules:**

CLIENT_ACCEPT: Сообщение о том, что пользователь исключён из рассылки.

**client_msg_before_start:**

CLIENT_START_MSG: Сообщение о том, что меню будет отправлено пользователю.

**client_cancel_order:**

CL_CANCEL: Информация о том, что заказ не будет оформлен.

Заказ

**client_start_order:**

CL_START_ORDER: Запрос на указание количества порций.

**client_count_of_serv:**

CL_COUNT: Сообщение об ограничении на количество порций.

**check_count_of_servings:**

CL_COUNT_2: Подтверждение о количестве порций и запрос на выбор способа оплаты.

Этот набор фраз предоставляет четкий и последовательный пользовательский интерфейс в боте. Они помогают пользователю на каждом этапе, от регистрации до оформления заказа, предоставляя нужную информацию и поддержку. Правильная организация текстов способствует лучшему взаимодействию между ботом и пользователем, облегчая процесс выполнения заказа.

## 2.2. Функциональные требования

Функциональные требования для чат-бота доставки еды могут быть разделены на несколько ключевых категорий, включая регистрацию пользователей, процесс заказа, управление меню, обработку платежей и взаимодействие с пользователями. Ниже приводится детализированный список функциональных требований.

1. Регистрация и аутентификация пользователей

Регистрация пользователя:

- Пользователь должен иметь возможность зарегистрироваться через бота, вводя свое имя, фамилию и номер телефона.
- Бот должен проверять корректность введённых данных (имя и фамилия должны состоять только из букв и тире, номер телефона должен соответствовать формату).
- Если пользователь уже зарегистрирован, бот должен уведомить его об этом.

Аутентификация пользователя:

- Пользователь должен иметь возможность войти в систему (если требуется) и получить доступ к своим данным и истории заказов.

2. Процесс заказа

- Создание заказа:
- Пользователь может в любой момент инициировать заказ, запросив меню на ближайшую дату.
- Бот должен спрашивать количество порций, которые пользователь хочет заказать.
- Пользователь должен иметь возможность указать тип получения заказа (доставка или самовывоз).

Выбор блюд:

- Бот должен предоставлять пользователю меню с доступными блюдами, включая описание и цену.
- Пользователь должен иметь возможность выбрать одно или несколько блюд для заказа.

3. Оплата

Выбор способа оплаты:

- Пользователь должен иметь возможность выбрать метод оплаты: наличными или картой.

Обработка платежей:

- При выборе оплаты по карте бот должен обеспечить безопасность и конфиденциальность операций.
- Если пользователь выбирает оплату наличными, бот должен запросить сумму, с которой необходима сдача.

4. Доставка

Адрес доставки:

- При выборе доставки бот должен запросить и подтвердить адрес доставки.
- Бот должен поступить с ограничениями на места доставки (например, только в пределах определённого жилого комплекса).

Статус доставки:

- Пользователь должен иметь возможность отслеживать статус своего заказа: ожидается ли доставка, в процессе или завершена.

5. Управление меню

- Получение актуального меню:
- Бот должен гарантировать, что пользователи получают актуальное, обновлённое меню.
- Обновление меню:
- Если администратор или менеджер добавляет или изменяет меню, бот должен сообщить об изменениях всем пользователям.

- Поддержка пользователей

6. Административные функции

Управление заказами:

- Администраторы должны иметь возможность просматривать список всех заказов, управлять ими и изменять статус.
- Управление блюдами:
- Администраторы должны иметь возможность добавлять, изменять и удалять блюда из меню.
- Статистика и отчёты:
- Бот должен предоставлять статистику по заказам, отзывам и продажам, доступную для администраторов.

7. Пользовательский интерфейс

Удобный интерфейс:

- Чат-бот должен иметь интуитивно понятный интерфейс с кнопками для выбора определённых опций.

Информация и уведомления:

- Бот должен отправлять уведомления пользователю о статусе заказа, специальных предложениях и акциях.

Эти функциональные требования обеспечивают создание эффективного, удобного и безопасного чат-бота для доставки еды. Они помогают пользователям легко заказывать еду и контролировать свои заказы, а также позволяют администраторам эффективно управлять процессом.

# ГЛАВА 3. РЕАЛИЗАЦИЯ

## 3.1. Структура проекта

Проект был организован с учётом принципов, заложенных в Django, что обеспечило структурированность и удобство разработки. Основная структура проекта выглядит следующим образом:

```
food_delivery_bot/
│
├── bot_app/
│   ├── __init__.py
│   ├── bot_create.py          # Инициализация бота и установка соединения с API Telegram
│   ├── data_exchanger.py      # Функции для работы с базой данных: добавление заказов, регистрация пользователей и т.д.
│   ├── handlers/
│   │   ├── __init__.py
│   │   ├── client_handlers.py    # Обработчики для пользовательских команд (регистрация, заказ и т.д.)
│   │   ├── admin_handlers.py     # Обработчики для административных команд
│   │   ├── utils.py           # Утилиты: функции проверки, обработки ошибок и т.д.
│   │   └── middlewares.py           # Middleware для обработки сообщений (например, логирование)
│   ├── keyboards/
│   │   ├── __init__.py
│   │   ├── user_kb.py         # Клавиатуры для пользователей
│   │   └── admin_kb.py        # Клавиатуры для администраторов
│   ├── phrases/
│   │   ├── __init__.py
│   │   └── user_phrases.py     # Фразы, используемые в общении с пользователями
│   ├── states.py              # Определение состояний для FSM (Finite State Machine)
│   └── models.py              # Определения моделей данных (если используется ORM)
│
├── config.py                  # Конфигурационный файл с настройками бота (токены, базы данных и т.д.)
├── requirements.txt           # Список зависимостей Python
└── main.py                    # Основной файл для запуска бота
```

Описание компонентов

**bot_app/:** Основная папка приложения, содержащая все компоненты чат-бота.

bot_create.py: Содержит код для инициализации бота и настройки соединения с API Telegram с использованием aiogram.

data_exchanger.py: Содержит функции для работы с данными, такие как регистрация пользователей, добавление заказов в базу данных и их получение.

**handlers/:**

client_handlers.py: Содержит обработчики команд и сообщений пользователей, такие как процесс регистрации и оформление заказа.

admin_handlers.py: Содержит обработчики команд для администраторов (например, управление меню, просмотр заказов).

utils.py: Вспомогательные функции, проверки и обработка различных данных.

middlewares.py: Опционально содержит обработчики промежуточных слоёв, такие как логирование запросов.

**keyboards/:**

user_kb.py: Определяет клавиатуры для взаимодействия с пользователями, включая кнопки для выбора меню, подтверждений и т.д.

admin_kb.py: Определяет клавиатуры для администраторов, чтобы они могли управлять заказами и блюдами.

**phrases/:**

user_phrases.py: Содержит все фразы, которые будут отправляться пользователям на разных этапах взаимодействия.

states.py: Определение различных состояний машины состояний (FSM), что помогает контролировать поток сообщений и взаимодействия с пользователями.

models.py: Если используется ORM (например, SQLAlchemy или Django ORM), здесь будут определены модели данных для хранения информации о пользователях и заказах.

config.py: Файл с конфигурацией проекта, включая параметры подключения к базе данных и токены API.

requirements.txt: Содержит список всех зависимостей для проекта (например, aiogram, psycopg2 для работы с PostgreSQL, и другие необходимые библиотеки).

main.py: Основной файл, в котором происходит запуск бота, инициализируются обработчики и запускается цикл обработки обновлений..

### 3.2. Реализация функционала

Чат-боты стали неотъемлемой частью современного бизнеса, предоставляя удобные и эффективные способы взаимодействия с клиентами.

Регистрация пользователей — это ключевой шаг в любом приложении, основанном на взаимодействии с пользователем. В контексте чат-бота для доставки еды регистрация позволяет закрепить за пользователем информацию, необходимую для выполнения его заказов.

Сбор информации: Бот собирает имя, фамилию и номер телефона через последовательные запросы к пользователю. Этот процесс подразумевает использование состояний, чтобы контролировать, на каком этапе регистрации находится пользователь.

Валидация данных: Важно убедиться, что пользователи вводят данные в правильном формате (например, имя и фамилия должны содержать только буквы, а номер телефона — соответствовать определенному шаблону).

Валидация позволяет предотвратить ошибки и обеспечивает корректное функционирование последующих этапов.

Хранение данных: После успешной регистрации данные пользователя сохраняются в базе данных, что позволяет системе идентифицировать его на следующих этапах взаимодействия.

2. Создание заказа

Создание заказа является центральным элементом функционала чат-бота. Этот процесс должен быть интуитивным и минимально затрудняющим пользователя. Запрос меню: Бот должен предоставлять пользователю актуальное меню с возможностью выбора блюд. Эта информация может быть динамически загружена из базы данных.

Определение количества порций: После выбора блюда бот запрашивает, сколько порций пользователь хочет заказать. Важно установить пределы, чтобы предотвратить избыточные заказы.

Выбор способа получения заказа: Пользователь может выбрать удобный способ получения заказа — самовывоз или доставка. Выбор должен поощрять пользователя, и бот должен объяснять, что именно включает каждый из вариантов.

3. Оплата заказа

Обработка оплаты — это важный аспект работы чат-бота, так как она напрямую связана с удовлетворённостью пользователей и безопасностью транзакций. Выбор метода оплаты: Бот должен предложить несколько способов оплаты (наличные, карта) и четко информировать пользователя о каждом процессе.

Валидация платежных данных: При выборе наличного расчета бот запрашивает сумму, с которой требуется сдача. Важно обеспечить, чтобы сумма была введена корректно, так как это может повлиять на процесс выполнения заказа.

4. Организация доставки

Организация доставки — ключевой шаг в завершении процесса заказа, который непосредственно влияет на удовлетворенность клиента.

Запрос адреса доставки: Бот должен запросить у пользователя адрес, по которому будет произведена доставка. Этот процесс может включать проверку на корректность введенного адреса. Подтверждение адреса: Перед окончательной отправкой заказа важно подтвердить адрес, чтобы избежать ошибок и недоразумений.

5. Подтверждение и управление заказом

Подтверждение заказа включает в себя предоставление пользователю всей информации о заказе для окончательной проверки. Проверка данных: Бот должен сформировать итоговое сообщение с деталями заказа и запросить подтверждение от пользователя. Это помогает убедиться, что информация верна до того, как заказ будет окончательно оформлен и передан в систему обработки.

6. Обработка ошибок и исключений

Эффективная обработка ошибок является важной частью любого программного обеспечения. Отправка сообщений об ошибках: Если бот сталкивается с проблемами (например, неверный ввод данных), он должен сообщать пользователю о проблеме и предлагать шаги для её устранения.

Реализация функционала чат-бота доставки еды требует тщательного проектирования и интеграции различных компонентов, начиная от регистрации пользователей и заканчивая обработкой заказов и платежей. Четкая структура интерфейса и продуманное взаимодействие с пользователями позволяют повысить общий уровень удовлетворённости и улучшить пользовательский опыт. Эти элементы являются основой для успешного функционирования чат-бота в сфере доставки еды и позволяют дифференцировать сервис на фоне конкурентов.

## 3.3. Тестирование и отладка

1. Подходы к тестированию чат-бота

1.1. Функциональное тестирование

Функциональное тестирование направлено на проверку того, выполняет ли бот все предусмотренные функции. Ключевые аспекты функционального тестирования включают:

- Регистрация пользователей: Тестирование всех этапов регистрации, включая ввод имени, фамилии и телефона, чтобы убедиться, что данные валидируются правильно.
- Создание заказа: Проверка возможности создания заказа от выбора блюда до выбора способа получения.
- Оплата: Тестирование различных методов оплаты (наличные, карта), включая валидацию данных о платежах и подсчет сдачи.
- Доставка: Проверка процесса запроса адреса доставки и его валидации.

1.2. Нефункциональное тестирование

- Нефункциональное тестирование фокусируется на производительности, безопасности и удобстве использования.

- Производительность: Оценка времени отклика бота на запросы пользователей в условиях повышенной нагрузки.
- Безопасность: Проверка обработки данных пользователей, правильности хранения конфиденциальной информации (например, номеров телефонов).
- Юзабилити: Тестирование интерфейса и логики взаимодействия, чтобы убедиться, что пользователи могут легко использовать бота.

1.3. Тестирование сценариев

- Тестирование сценариев включает в себя проверку предопределенных сценариев взаимодействия пользователя с ботом.
- Позитивные сценарии: Проверка всех ожидаемых действий, например, успешная регистрация, оформление заказа и оплата.
- Негативные сценарии: Тестирование неправильно введенных данных, несуществующих команд и других возможных ошибок, чтобы убедиться, что бот обрабатывает их корректно.

2. Автоматизированное тестирование

Автоматизированное тестирование позволяет значительно ускорить процесс проверки кода и его эффективности.

API-тестирование: бот взаимодействует с веб-сервисами для обработки оплаты или получения меню, протестированы API с помощью инструментов, таких как Postman или Insomnia.

3. Виды тестов

В ходе тестирования чат-бота выполнены следующие виды тестов:

- Модульные тесты: Проверка отдельных функциональных блоков бота, чтобы убедиться в их корректности.
- Интеграционные тесты: Тестирование взаимодействия между различными компонентами системы (например, приложением и базой данных).
- Системные тесты: Оценка всего чат-бота как единого целого, чтобы удостовериться, что он работает в соответствии с требованиями.
- Регрессионные тесты: Проверка функциональности после внесения изменений в код, чтобы убедиться, что ранее работающие функции по-прежнему функционируют корректно.

4. Процесс отладки

Отладка — это процесс обнаружения и исправления ошибок в программном обеспечении.

4.1. Использование логирования

- Логи предоставляют информацию о том, что происходит в боте в процессе его работы, и позволяют разработчикам выявлять ошибки.
- Настройка логирования: Необходимо включить логирование всех действий бота, включая входящие сообщения, обработанные команды и ошибки.

4.2. Интерактивная отладка

Использование отладчиков позволяет в реальном времени просматривать состояние программы.

- Отладчики: Инструменты, такие как pdb (Python Debugger), могут использоваться для пошагового выполнения кода и отслеживания переменных в момент возникновения ошибки.
- Тестовые фреймворки: Многие тестовые фреймворки также имеют встроенные средства для отладки.

5. Подготовка к тестированию

- Создан план тестирования, включающий все ключевые сценарии и методы, которые будут использоваться.
- Подготовлены тестовые данные, включая как валидные, так и невалидные варианты данных, с которыми будет работать бот.

# ЗАКЛЮЧЕНИЕ

В результате выполнения дипломной работы был успешно разработан чат-бот доставки еды, который представляет собой современное решение для автоматизации процесса заказа и доставки пищи. Проект включал в себя полный цикл разработки, начиная с анализа требований и проектирования архитектуры и заканчивая реализацией функционала и проведением тестирования. Бот был реализован с использованием Python и библиотеки aiogram.

Разработанный чат-бот обладает рядом ключевых функциональных возможностей, включая:

Регистрация и аутентификация пользователей: Пользователи могут легко зарегистрироваться и войти в систему, что позволяет им сохранять свои данные и историю заказов.

Интерактивный процесс оформления заказов: Бот предоставляет пользователям актуальное меню, позволяет легко выбирать блюда и указывать количество порций, а также предлагает варианты доставки и самовывоза.

Обработка платежей: Реализованы различные способы оплаты, включая наличные и безналичные расчеты, что дает пользователям свободу выбора.

Поддержка и взаимодействие с пользователями: Чат-бот включает функционал для сбора обратной связи и комментариев, что помогает улучшать качество услуг.

Высокий уровень безопасности: Обеспечена защита данных пользователей и конфиденциальная обработка платежной информации.

В процессе разработки был уделен особый внимание тестированию и отладке системы, что позволило избежать критических ошибок и обеспечить

надежность работы чат-бота. Использование различных методов тестирования позволило выявить и исправить потенциальные уязвимости и улучшить пользовательский интерфейс.

Кроме того, реализация проекта изучила и применяла лучшие практики разработки чат-ботов, такие как применение архитектуры с разделением на модули, использование состояний для управления логикой взаимодействия с пользователем и использование фреймворков для автоматизации тестирования.

Перспективы развития

Разработка чат-бота доставки еды открывает возможности для будущих улучшений и расширения функционала. В дальнейшем можно рассмотреть внедрение следующих возможностей:

Расширение ассортимента меню: Добавление новых блюд и категорий продуктов для увеличения выбора для пользователей.

Интеграция с системами лояльности: Реализация программ лояльности и акций для постоянных клиентов.

В заключение, разработка чат-бота доставки еды является успешным примером применения современных технологий для улучшения пользовательского опыта и оптимизации процессов в сфере общественного питания. Этот проект может служить основой для дальнейших исследований и разработок в области автоматизации сервисов и улучшения взаимодействия между бизнесом и клиентами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Официальная документация Django: docs.djangoproject.com

Официальная документация PostgreSQL.: postgresql.org/docs

Иванов, В. И. Python для разработчиков / В. И. Иванов. — Москва: Диалектика, 2020. — 350 с.

Документация WireGuard. https://www.wireguard.com. [Дата обращения: 19.12.2024].

Python Software Foundation. Python 3 Documentation.

Морозов, И. В. Основы работы с базами данных MySQL / И. В. Морозов. Москва: Бином, 2020. — 220 с.

Включены визуальные примеры пользовательского интерфейса для наглядного представления функциональности

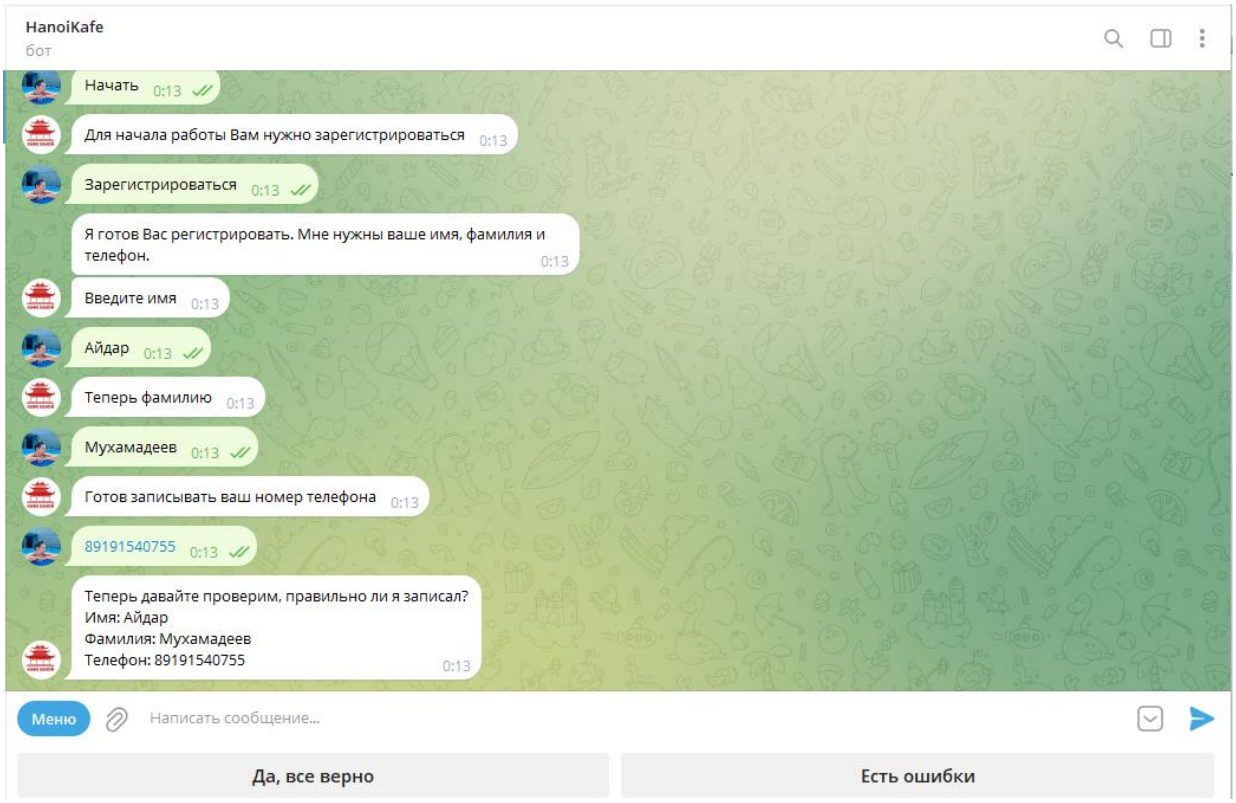



Рис.2 Меню регистрации пользователей Бота

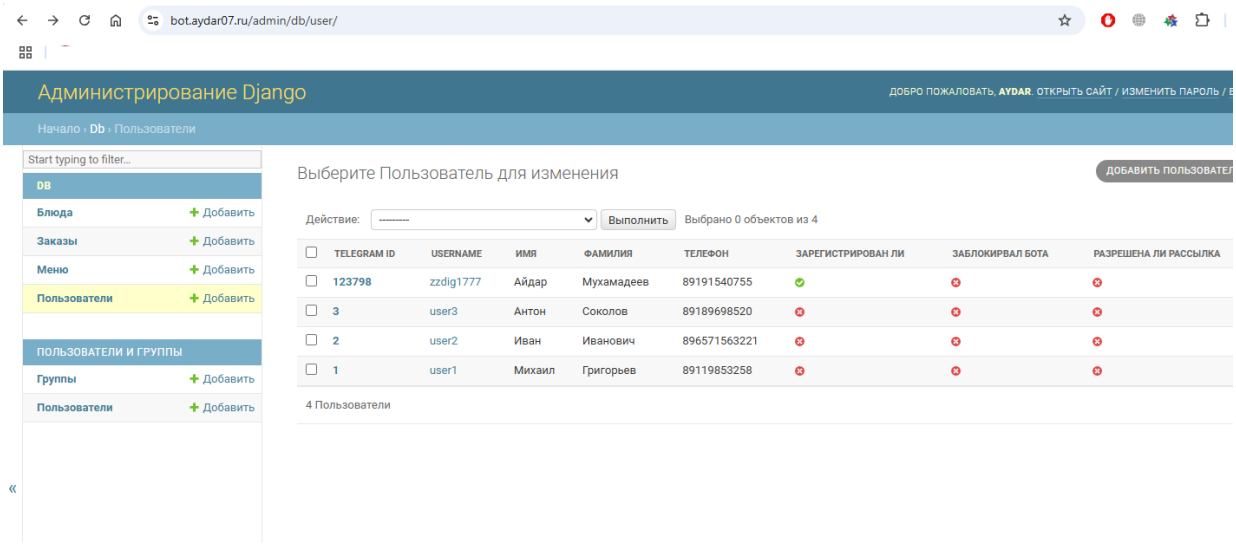



Рис.3 Меню регистрации пользователей

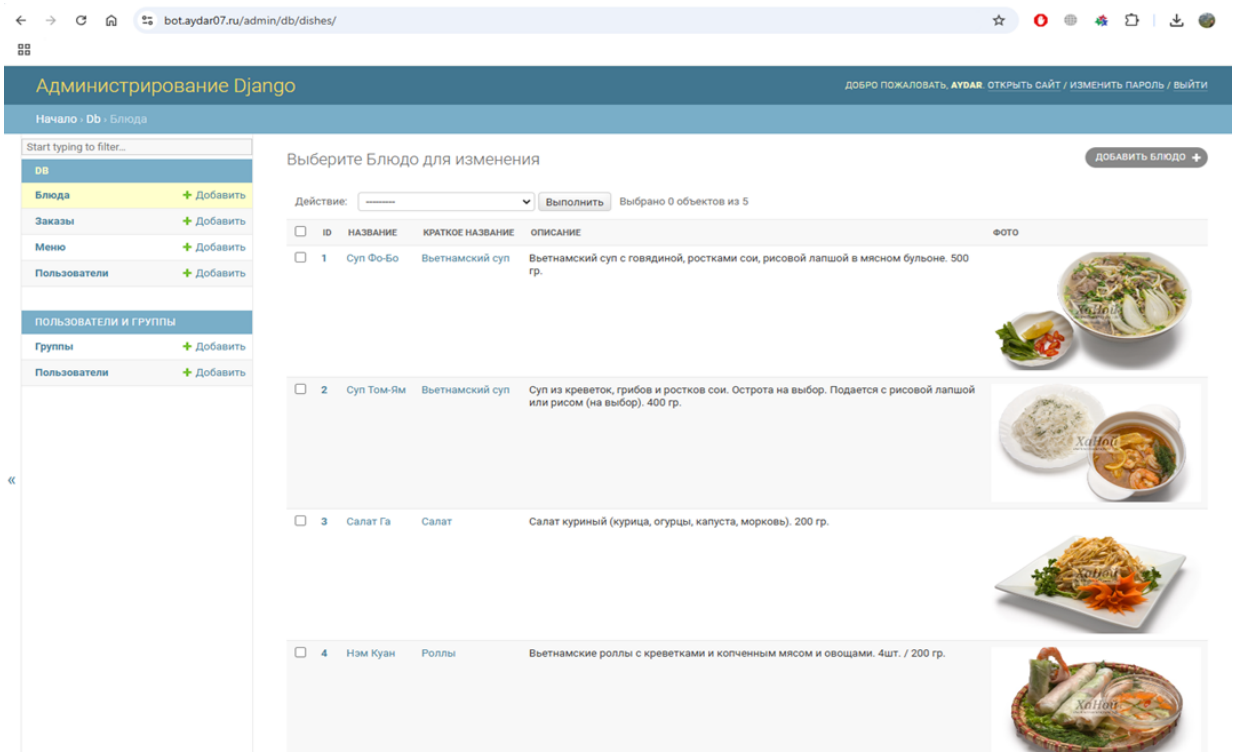
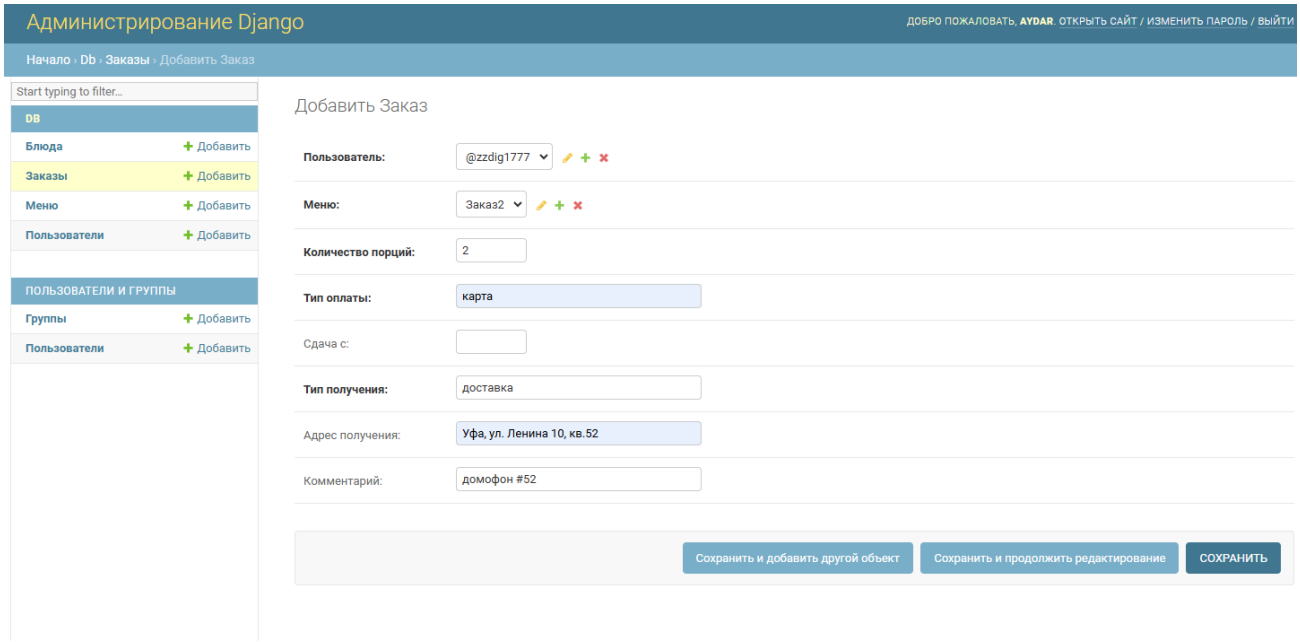
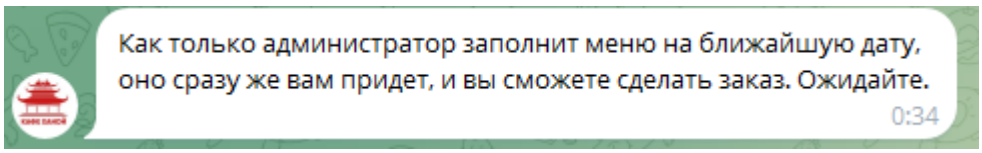



Рис.4 Каталог блюд в панели администратора Django



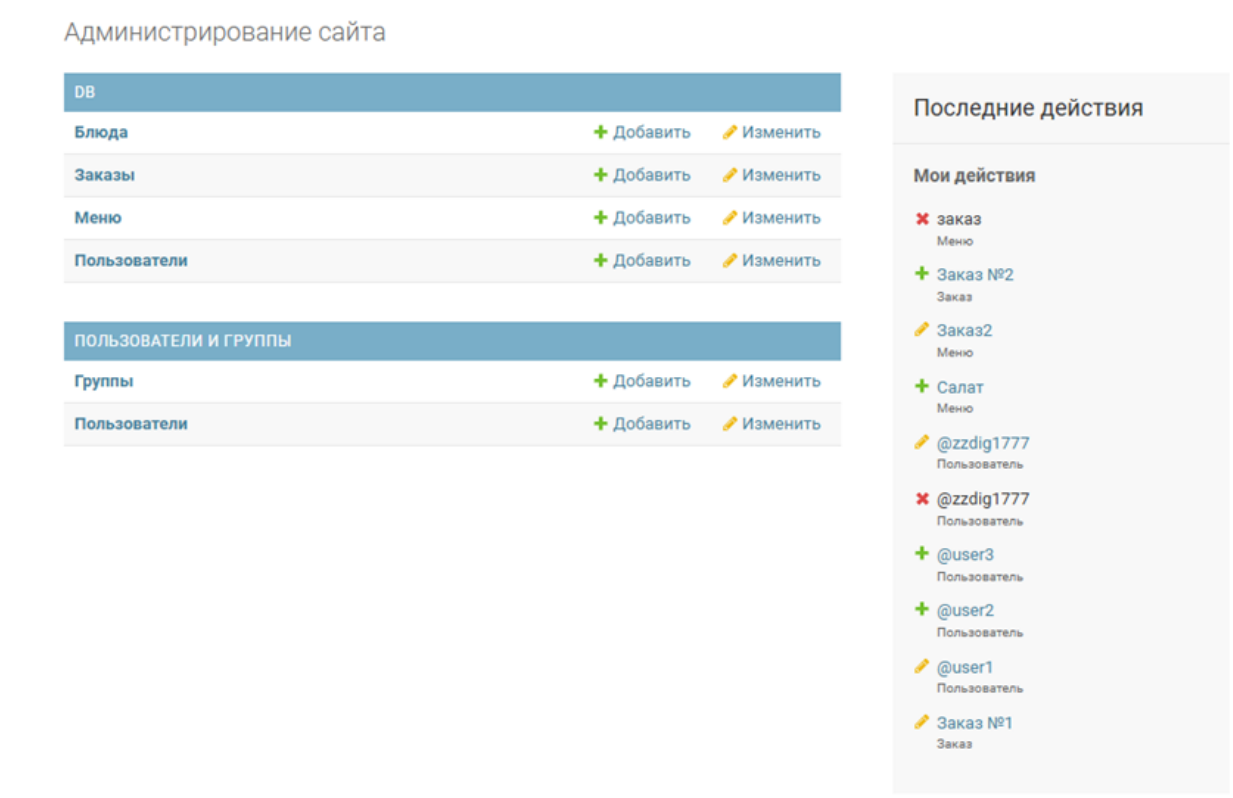
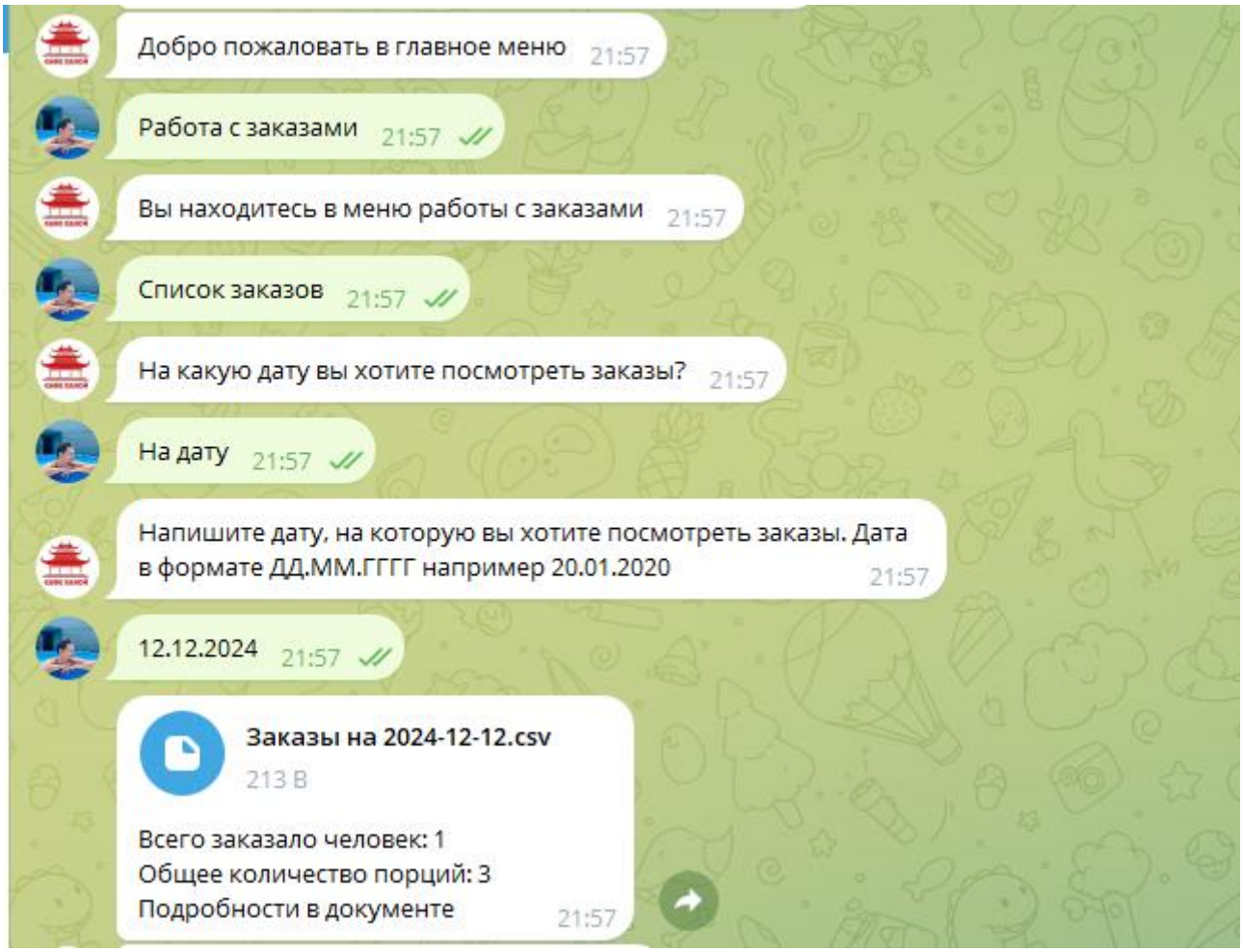



Рис.4 Добавление заказа



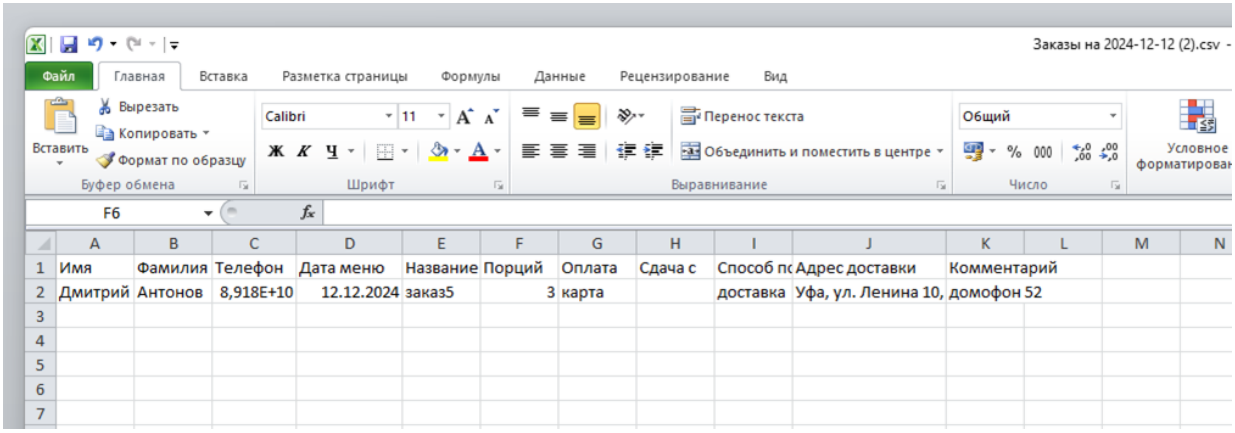


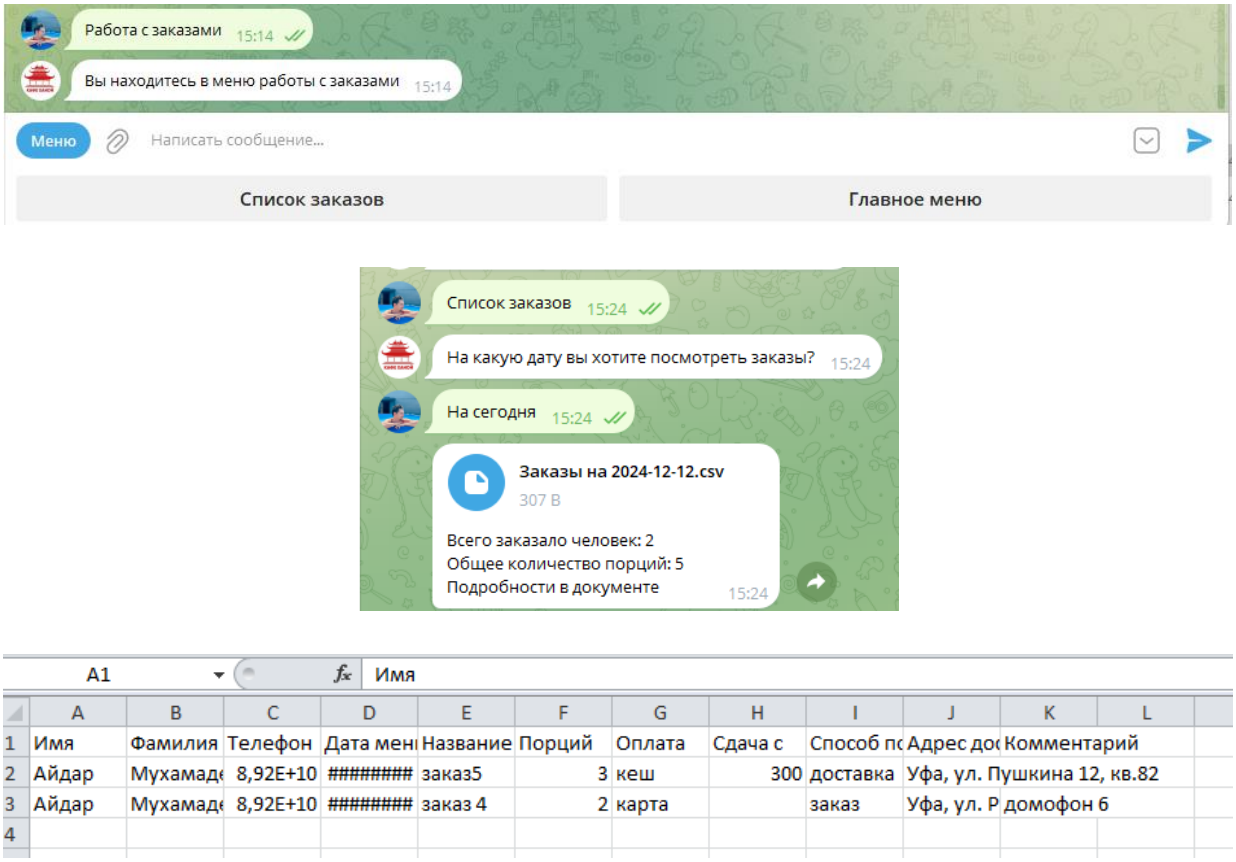

Рис.5 Обработка заказа





Рис.6 Панель администратора чат-бота (меню)


Рис.7 Панель администратора чат-бота (блюда)




| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Имя | Фамилия | Телефон | Дата мен | Название | Порций | Оплата | Сдача с | Способ по | Адрес дос | Комментарий | |
| 2 | Айдар | Мухамад | 8,92E+10 | ######## | заказ5 | 3 | кеш | 300 | доставка | Уфа, ул. Пушкина 12, кв.82 | | |
| 3 | Айдар | Мухамад | 8,92E+10 | ######## | заказ 4 | 2 | карта | | заказ | Уфа, ул. Р | домофон 6 | |
| 4 | | | | | | | | | | | | |

Рис.8 Панель администратора чат-бота (заказы)

**/env.bot**

```
TELEGRAM_TOKEN=**********
USERS_URL=http://web:8000/api/users/
DISHES_URL=http:/web:8000/api/dishes/
MENUS_URL=http:/web:8000/api/menus/
ORDERS_URL=http:/web:8000/api/orders/
ADMIN_IDS=**********
IS_REDIS_STORAGE=True
IS_WEBHOOK=True
REDIS_DSN=redis://redis:6379/0
BASE_URL=https://bot.aydar07.ru
WEB_SERVER_HOST = "0.0.0.0"
WEB_SERVER_PORT = 7771
WEBHOOK_PATH = "/webhook/main/"
```

**handlers/admin_handlers.py**

```
# ==================администнский доступ=========================
from datetime import date, timedelta

from aiogram import types
from aiogram.dispatcher.fsm.context import FSMContext

from bot_app.bot_create import bot
from bot_app.data_exchanger import get_user_info, list_menus
from bot_app.keyboards.admin_kb import admin_main_menu_kb
from bot_app.phrases.admin_phrases import ADM_ENTER_ERR, ADM_MENU
from bot_app.states import AdminState


async def enter_admin_section(message: types.Message, state: FSMContext):
    """Вход в раздел админа."""
    user_id = message.from_user.id
    response = await get_user_info(user_id)
    if not response['json']['is_admin']:
        await bot.send_message(text=ADM_ENTER_ERR,
                               chat_id=message.from_user.id)
    else:
        await state.set_state(AdminState.wait_admin_check_today_menu)
        return await admin_check_today_menu(message, state)


async def admin_check_today_menu(message: types.Message, state: FSMContext):
    """Проверка на заполненность меню на завтра. Происходит каждый раз при
    входе в раздел админа."""
    menus = await list_menus()
    tomorrow = date.today() + timedelta(days=1)
    text = 'Произвожу проверку наполненности меню на завтра...'
    if not any(menu['date_of_menu'] == tomorrow for menu in menus):
        text += '\nМеню на завтра НЕ ЗАПОЛНЕНО!!!'
    else:
        text += '\nВы уже наполнили меню на завтра, все ОК'
```

```
    await bot.send_message(text=text,
                           chat_id=message.from_user.id)
    await state.set_state(AdminState.wait_admin_main_menu)
    return await admin_main_menu(message, state)


async def admin_main_menu(message: types.Message, state: FSMContext):
    """Главное меню администратора."""
    await bot.send_message(text=ADM_MENU,
                           chat_id=message.from_user.id,
                           reply_markup=admin_main_menu_kb())
    await state.set_state(AdminState.wait_admin_choose_main_menu)
```

**handlers /dishes_handlers.py**

```
# ==================работа с блюдами==================
from aiogram import types
from aiogram.dispatcher.fsm.context import FSMContext
from aiogram.exceptions import TelegramBadRequest

from bot_app.bot_create import BASE_URL
from bot_app.bot_create import bot
from bot_app.data_exchanger import (list_dishes, add_dish, get_dish,
                                    edit_dish, remove_dish)
from bot_app.handlers.admin.admin_handlers import admin_main_menu
from bot_app.keyboards.admin_kb import (admin_work_with_dishes_kb,
                                        admin_add_dish_noimage_kb,
                                        admin_add_dish_check_kb,
                                        admin_start_repair_dish_kb,
                                        admin_remove_dish_choose_kb)
from bot_app.keyboards.common_kb import remove_kb
from bot_app.phrases.admin_phrases import (DISH_MENU_TEXT, ADM_CHOOSE_DISH,
                                           ADM_CHOOSE_DISH_2, LIST_DISH,
                                           ADM_ADD_TITLE, ADM_ADD_DESCR,
                                           ADM_ADD_PHOTO, ADD_TO_DB,
                                           DISH_REPAIR, DISH_REPAIR_2,
                                           DISHES_TO_REPAIR,
                                           CHECK_ID_ERR, ID_DISH_404,
                                           ID_DISH_OTHER_ERR,
                                           START_REPAIR_DISH,
                                           START_REPAIR_DISH_2, DISH_EDIT_DATA,
                                           DISH_EDIT_DATA_2, DISH_EDIT_DATA_3,
                                           DISH_EDIT_DATA_4, DISH_EDIT_SUCCESS,
                                           DISH_EDIT_ERR, DISH_RM_ENTER_ID,
                                           DISH_RM_ACCEPT, DISH_RM_CONFIRM,
                                           DISH_RM_CONFIRM_2,
                                           DISH_RM_CONFIRM_3,
                                           DISH_RM_CANCEL)
from bot_app.states import AdminState


async def admin_work_with_dishes(message: types.Message, state: FSMContext):
    """Меню работы с блюдами."""
    await bot.send_message(text=DISH_MENU_TEXT,
                           chat_id=message.from_user.id,
                           reply_markup=admin_work_with_dishes_kb())
```

```
    await state.set_state(AdminState.wait_admin_choose_dishes)


async def admin_choose_dishes(message: types.Message, state: FSMContext):
    """Обработка кнопок из меню работы с блюдами."""
    if message.text == 'Список блюд':
        text = await admin_list_dishes()
        await bot.send_message(text=text,
                               chat_id=message.from_user.id)
        await state.set_state(AdminState.wait_admin_work_with_dishes)
        return await admin_work_with_dishes(message, state)
    elif message.text == 'Добавить блюдо':
        await bot.send_message(text=ADM_CHOOSE_DISH,
                               chat_id=message.from_user.id)
        await bot.send_message(text=ADM_CHOOSE_DISH_2,
                               chat_id=message.from_user.id,
                               reply_markup=remove_kb())
        await state.set_state(AdminState.wait_admin_add_dish_title)
    elif message.text == 'Изменить блюдо':
        await state.set_state(AdminState.wait_admin_repair_dish)
        return await admin_repair_dish(message, state)
    elif message.text == 'Удалить блюдо':
        await state.set_state(AdminState.wait_admin_remove_dish)
        return await admin_remove_dish(message, state)
    elif message.text == 'Главное меню':
        await state.set_state(AdminState.wait_admin_main_menu)
        return await admin_main_menu(message, state)


# Список доступных блюд
async def admin_list_dishes():
    """Представление списка доступных блюд в базе данных"""
    response = await list_dishes()
    text = LIST_DISH
    for item in response['json']:
        text += f'{item["id"]}: {item["shortname"]}\n'
    return text


# Добавление блюда
async def admin_add_dish_title(message: types.Message, state: FSMContext):
    """Добавление нового блюда. Краткое название блюда."""
    await state.update_data(title=message.text)
    await bot.send_message(text=ADM_ADD_TITLE,
                           chat_id=message.from_user.id)
    await state.set_state(AdminState.wait_admin_add_dish_shortname)


async def admin_add_dish_shortname(message: types.Message, state: FSMContext):
    """Добавление нового блюда. Описание блюда."""
    await state.update_data(shortname=message.text)
    await bot.send_message(text=ADM_ADD_DESCR,
                           chat_id=message.from_user.id)
    await state.set_state(AdminState.wait_admin_add_dish_descr)


async def admin_add_dish_descr(message: types.Message, state: FSMContext):
    """Добавление нового блюда. Фотография блюда, либо без фотографии."""
    await state.update_data(descr=message.text)
```

```
    await bot.send_message(text=ADM_ADD_PHOTO,
                           chat_id=message.from_user.id,
                           reply_markup=admin_add_dish_noimage_kb())
    await state.set_state(AdminState.wait_admin_add_dish_photo)


async def admin_add_dish_photo(message: types.Message, state: FSMContext):
    """Добавление нового блюда. Обработка информации, фотографии и вывод
    пользователю на подтверждение."""
    data = await state.get_data()
    text = ''
    if message.photo:
        mes_img = message.photo[-1]
        file_id = mes_img.file_id
        file = await bot.get_file(file_id)
        filename = mes_img.file_id
        await state.update_data(image=file.file_path,
                                filename=filename,
                                image_id=file_id)
        text = (f'Теперь проверим как это выглядит:\n'
                f'Название: {data["title"]}\n'
                f'Короткое название: {data["shortname"]}\n'
                f'Описание: {data["descr"]}\n'
                f'Фото: на которое отвечаю')
    if message.text:
        await state.update_data(image=None,
                                filename=None,
                                image_id=None)
        text = (f'Теперь проверим как это выглядит:\n'
                f'Название: {data["title"]}\n'
                f'Короткое название: {data["shortname"]}\n'
                f'Описание: {data["descr"]}\n'
                f'Фото: Без фото')
    await message.reply(text=text,
                        reply_markup=admin_add_dish_check_kb())
    await state.set_state(AdminState.wait_admin_confirm_add_dish_data)


async def admin_confirm_add_dish_data(message: types.Message,
                                      state: FSMContext):
    """Сохранение блюда в базу данных в случае подтверждения, возврат в
    начало добавления блюда в случае отказа."""
    data = await state.get_data()
    if message.text == 'Да, все верно':
        image = None
        if data["image"]:
            image_path = data["image"]
            image = (await bot.download_file(image_path)).read()
        dish_data = {
            'title': data["title"],
            'shortname': data["shortname"],
            'description': data["descr"],
            'image': image,
            'image_id': data["image_id"],
            'filename': data["filename"],
        }
        await add_dish(dish_data)
        await bot.send_message(text=ADD_TO_DB,
                               chat_id=message.from_user.id)
```

```
        await state.set_state(AdminState.wait_admin_work_with_dishes)
        return await admin_work_with_dishes(message, state)

    elif message.text == 'Нужно исправить':
        await bot.send_message(text=DISH_REPAIR,
                               chat_id=message.from_user.id)
        await bot.send_message(text=DISH_REPAIR_2,
                               chat_id=message.from_user.id,
                               reply_markup=remove_kb())
        await state.set_state(AdminState.wait_admin_add_dish_title)


# исправление блюда
async def admin_repair_dish(message: types.Message, state: FSMContext):
    """Исправление блюда. Ввод ID блюда из базы."""
    dishes = await admin_list_dishes()
    text = (f'{dishes}\n'
            f'{DISHES_TO_REPAIR}')
    await bot.send_message(text=text,
                           chat_id=message.from_user.id,
                           reply_markup=remove_kb())
    await state.set_state(AdminState.wait_admin_start_repair_dish)


async def admin_check_id_dish_value(message: types.Message):
    """Проверка введенного ID блюда на валидность"""
    dish_id = message.text
    response = await get_dish(dish_id)
    try:
        int(dish_id)
    except ValueError:
        await bot.send_message(text=CHECK_ID_ERR,
                               chat_id=message.from_user.id)
        return response['json'], False
    if response['status'] == 404:
        await bot.send_message(text=ID_DISH_404,
                               chat_id=message.from_user.id)
        return response['json'], False
    elif response['status'] == 200:
        return response['json'], True
    else:
        await bot.send_message(text=ID_DISH_OTHER_ERR,
                               chat_id=message.from_user.id)
        return response['json'], False


async def admin_get_formalized_dish(message: types.Message, dish_instance):
    """Стилизованный вывод блюда."""
    dish = dish_instance
    dish_title = dish['title']
    dish_shortname = dish['shortname']
    dish_descr = dish['description']
    dish_image = dish['image']
    dish_filename = f"{BASE_URL}/{dish['filename']}"
    dish_image_id = dish['image_id']
    formalized_dish = (f'<b>Название</b>\n{dish_title}\n'
                       f'<b>Краткое название</b>\n{dish_shortname}\n'
                       f'<b>Описание</b>\n{dish_descr}')
    if dish_image is None:
```

```
        formalized_dish += '\n<b>Фото</b>\nБез фото'
        await bot.send_message(text=formalized_dish,
                               chat_id=message.from_user.id)
    else:
        if dish_image_id is None:
            await bot.send_photo(message.from_user.id, dish_filename,
                                 formalized_dish)
        else:
            try:
                await bot.send_photo(message.from_user.id, dish_image_id,
                                     formalized_dish)
            except TelegramBadRequest:
                await bot.send_photo(message.from_user.id,
                                     dish_filename,
                                     formalized_dish)


async def admin_start_repair_dish(message: types.Message, state: FSMContext):
    """Вывод стилизованного меню админу и выбор параметра блюда, который
    нужно изменить."""
    dish_instance, is_correct_digit = await admin_check_id_dish_value(
        message)
    if not is_correct_digit:
        await state.set_state(AdminState.wait_admin_repair_dish)
        return await admin_repair_dish(message, state)
    await state.update_data(dish_instance=dish_instance)
    await bot.send_message(text=START_REPAIR_DISH,
                           chat_id=message.from_user.id)
    await admin_get_formalized_dish(message, dish_instance)
    await bot.send_message(text=START_REPAIR_DISH_2,
                           chat_id=message.from_user.id,
                           reply_markup=admin_start_repair_dish_kb())
    await state.set_state(AdminState.wait_admin_repair_dish_choose_field)


async def admin_repair_dish_choose_field(message: types.Message,
                                         state: FSMContext):
    """Запрос нового значения для выбранного параметра блюда."""
    db_key_to_change = None
    if message.text == 'Название':
        db_key_to_change = 'title'
        await bot.send_message(text=DISH_EDIT_DATA,
                               chat_id=message.from_user.id,
                               reply_markup=remove_kb())
    elif message.text == 'Краткое название':
        db_key_to_change = 'shortname'
        await bot.send_message(text=DISH_EDIT_DATA_2,
                               chat_id=message.from_user.id,
                               reply_markup=remove_kb())
    elif message.text == 'Описание':
        db_key_to_change = 'description'
        await bot.send_message(text=DISH_EDIT_DATA_3,
                               chat_id=message.from_user.id,
                               reply_markup=remove_kb())
    elif message.text == 'Фотография':
        db_key_to_change = 'image'
        await bot.send_message(text=DISH_EDIT_DATA_4,
                               chat_id=message.from_user.id,
                               reply_markup=admin_add_dish_noimage_kb())
```

```
    await state.update_data(db_key_to_change=db_key_to_change)
    await state.set_state(AdminState.wait_admin_repair_dish_fields)


async def admin_repair_dish_fields(message: types.Message, state: FSMContext):
    """Внесение изменений в указанный параметр блюда, его сохранение в базе
    данных."""
    new_dish_data = {}
    data = await state.get_data()
    dish_id = data['dish_instance']['id']
    key = data['db_key_to_change']
    if message.photo:
        mes_img = message.photo[-1]
        file_id = mes_img.file_id
        file = await bot.get_file(file_id)
        filename = mes_img.file_id
        image = (await bot.download_file(file.file_path)).read()
        new_dish_data['image'] = image
        new_dish_data['filename'] = filename
        new_dish_data['image_id'] = file_id
    if message.text:
        if message.text == 'Без фото':
            new_dish_data['image'] = None
            new_dish_data['filename'] = None
            new_dish_data['image_id'] = None
        else:
            new_dish_data[key] = message.text
    response = await edit_dish(new_dish_data, dish_id)
    if response['status'] == 200:
        await bot.send_message(text=DISH_EDIT_SUCCESS,
                               chat_id=message.from_user.id,
                               reply_markup=admin_work_with_dishes_kb())
        await state.set_state(AdminState.wait_admin_work_with_dishes)
        return await admin_work_with_dishes(message, state)
    else:
        await bot.send_message(text=DISH_EDIT_ERR,
                               chat_id=message.from_user.id,
                               reply_markup=remove_kb())
        await state.set_state(AdminState.wait_admin_work_with_dishes)
        return await admin_work_with_dishes(message, state)


# удаление блюда
async def admin_remove_dish(message: types.Message, state: FSMContext):
    """Удаление блюда. Запрос ID блюда для удаления."""
    await bot.send_message(text=DISH_RM_ENTER_ID,
                           chat_id=message.from_user.id,
                           reply_markup=remove_kb())
    await state.set_state(AdminState.wait_admin_remove_dish_choose)


async def admin_remove_dish_choose(message: types.Message, state: FSMContext):
    """Проверка введенного ID блюда на валидность, вывод админу блюда,
    запрос подтверждения удаления этого блюда."""
    dish_instance, is_correct_digit = await admin_check_id_dish_value(
        message)
    if not is_correct_digit:
        await state.set_state(AdminState.wait_admin_repair_dish)
        return await admin_repair_dish(message, state)
```

```
    await state.update_data(dish_id=message.text)
    await admin_get_formalized_dish(message, dish_instance)
    await bot.send_message(text=DISH_RM_ACCEPT,
                           chat_id=message.from_user.id,
                           reply_markup=admin_remove_dish_choose_kb())
    await state.set_state(AdminState.wait_admin_remove_dish_confirm)


async def admin_remove_dish_confirm(message: types.Message, state: FSMContext):
    """Обработка подтверждения / отмены для удаления блюда."""
    if message.text == 'Да, уверен':
        data = await state.get_data()
        dish_id = data['dish_id']
        response = await remove_dish(dish_id)
        if response['status'] == 204:
            text = DISH_RM_CONFIRM
        elif response['status'] == 404:
            text = DISH_RM_CONFIRM_2.format(response)
        else:
            text = DISH_RM_CONFIRM_3.format(response)
        await bot.send_message(text=text,
                               chat_id=message.from_user.id,
                               reply_markup=admin_work_with_dishes_kb())
        await state.set_state(AdminState.wait_admin_work_with_dishes)
        return await admin_work_with_dishes(message, state)

    elif message.text == 'Отмена!!':
        await bot.send_message(text=DISH_RM_CANCEL,
                               chat_id=message.from_user.id,
                               reply_markup=admin_work_with_dishes_kb())
        await state.set_state(AdminState.wait_admin_work_with_dishes)
        return await admin_work_with_dishes(message, state)


async def client_get_formalized_dish(dish_instance, client_id):
    """Красивое представление меню для пользователя."""
    dish = dish_instance
    dish_title = dish['title']
    dish_descr = dish['description']
    dish_image = dish['image']
    dish_filename = f"{BASE_URL}/{dish['filename']}"
    dish_image_id = dish['image_id']
    formalized_dish = (f'<b>Блюдо:</b>\n{dish_title}\n'
                       f'<b>Описание: </b>\n{dish_descr}')
    if dish_image is None:
        formalized_dish += '\n<b>Фото</b>\nБез фото'
        await bot.send_message(text=formalized_dish,
                               chat_id=client_id)
    else:
        if dish_image_id is None:
            await bot.send_photo(client_id, dish_filename,
                                 formalized_dish)
        else:
            try:
                await bot.send_photo(client_id, dish_image_id,
                                     formalized_dish)
            except TelegramBadRequest:
                print(dish_filename)
                await bot.send_photo(client_id,
```

```
                        dish_filename,
                        formalized_dish)
```

**handlers /registration_handlers.py**

```
import json

from aiogram import types
from aiogram.dispatcher.fsm.context import FSMContext

from bot_app.bot_create import bot
from bot_app.data_exchanger import get_user_info, user_registration
from bot_app.handlers.user_handlers import client_msg_before_start
from bot_app.handlers.utils import (validate_phone, get_data_to_validate,
                        validate_name)
from bot_app.keyboards.common_kb import (start_using, reg_kb, remove_kb,
                            ready_repair_reg_kb,
                            confirm_reg_kb)
from bot_app.phrases.user_phrases import (CHECK_REG_1, CHECK_REG_2,
                            START_REG_1,
                            START_REG_2, INPUT_NAME_1,
                            INPUT_NAME_2, INPUT_LASTNAME_1,
                            INPUT_LASTNAME_2, INPUT_PHONE_1,
                            INPUT_PHONE_2, CONFIRM_REG_1,
                            CONFIRM_REG_ERR, CONFIRM_REG_2)
from bot_app.states import UserState, RegState


async def check_registration(message: types.Message, state: FSMContext):
    """Проверка на регистрацию."""
    user_id = message.from_user.id
    response = await get_user_info(user_id)
    if response['json']['is_register'] and response['json']['is_allow_mail']:
        await state.set_state(UserState.wait_to_waiting_order_info)
        return await client_msg_before_start(message)
    elif (response['json']['is_register']
          and not response['json']['is_allow_mail']):
        await bot.send_message(text=CHECK_REG_1,
                               chat_id=message.from_user.id,
                               reply_markup=start_using())
        await state.set_state(UserState.wait_to_enter_client_section)
    else:
        await bot.send_message(text=CHECK_REG_2,
                               chat_id=message.from_user.id,
                               reply_markup=reg_kb())
        await state.set_state(RegState.wait_start_registration)


async def start_registration(message: types.Message, state: FSMContext):
    """Начало регистрации."""
    await bot.send_message(text=START_REG_1,
                           chat_id=message.from_user.id,
                           reply_markup=remove_kb())
    await bot.send_message(text=START_REG_2,
                           chat_id=message.from_user.id)
    await state.set_state(RegState.wait_input_firstname)
```

```
async def input_firstname(message: types.Message, state: FSMContext):
    """Ввод имени."""
    name = validate_name(message)
    if not name:
        await bot.send_message(text=INPUT_NAME_1,
                               chat_id=message.from_user.id,
                               reply_markup=remove_kb())
        return await get_data_to_validate(state)
    await state.update_data(input_firstname=name)
    await bot.send_message(text=INPUT_NAME_2,
                           chat_id=message.from_user.id,
                           reply_markup=remove_kb())
    await state.set_state(RegState.wait_input_lastname)


async def input_lastname(message: types.Message, state: FSMContext):
    """Ввод фамилии."""
    surname = validate_name(message)
    if not surname:
        await bot.send_message(text=INPUT_LASTNAME_1,
                               chat_id=message.from_user.id,
                               reply_markup=remove_kb())
        return await get_data_to_validate(state)
    await state.update_data(input_lastname=surname)
    await bot.send_message(text=INPUT_LASTNAME_2,
                           chat_id=message.from_user.id,
                           reply_markup=remove_kb())
    await state.set_state(RegState.wait_input_phone)


async def input_phone(message: types.Message, state: FSMContext):
    """Ввод номера телефона."""
    phone = validate_phone(message)
    if not phone:
        await bot.send_message(text=INPUT_PHONE_1,
                               chat_id=message.from_user.id,
                               reply_markup=remove_kb())
        return await get_data_to_validate(state)
    await state.update_data(input_phone=phone)
    data = await state.get_data()
    text = (f'{INPUT_PHONE_2}\n'
            f'Имя: {data["input_firstname"]}\n'
            f'Фамилия: {data["input_lastname"]}\n'
            f'Телефон: {data["input_phone"]}\n')
    await bot.send_message(text=text,
                           chat_id=message.from_user.id,
                           reply_markup=confirm_reg_kb())
    await state.set_state(RegState.wait_confirm_registration)


async def confirm_registration(message: types.Message, state: FSMContext):
    """Подтверждение регистрации."""
    if message.text == 'Да, все верно':
        data = await state.get_data()
        user_id = str(message.from_user.id)
        user_data = {
            'input_firstname': data["input_firstname"],
            'input_lastname': data["input_lastname"],
```

```
        'input_phone': data["input_phone"],
        'is_register': True,
      }
      json_data = json.loads(json.dumps(user_data))
      response = await user_registration(user_id, json_data)
      data.clear()
      if response['status'] == 200:
        await bot.send_message(text=CONFIRM_REG_1,
                               chat_id=message.from_user.id,
                               reply_markup=start_using())
        await state.set_state(UserState.wait_to_enter_client_section)
      else:
        await bot.send_message(text=CONFIRM_REG_ERR,
                               chat_id=message.from_user.id)
        await bot.send_message(text=str(json_data),
                               chat_id=message.from_user.id)
    if message.text == 'Есть ошибки':
      await bot.send_message(text=CONFIRM_REG_2,
                             chat_id=message.from_user.id,
                             reply_markup=ready_repair_reg_kb())
      await state.set_state(RegState.wait_start_registration)
```

**handlers /utils.py**

```
import csv
import re
from datetime import datetime

from aiogram import types
from aiogram.dispatcher.fsm.context import FSMContext
from aiogram.types import FSInputFile

from bot_app.bot_create import bot
from bot_app.keyboards.common_kb import remove_kb
from bot_app.phrases.common_phrases import VALIDATE_DATE_TEXT


async def get_data_to_validate(state: FSMContext):
    """Запрос данных для валидации"""


def validate_phone(message: types.Message):
    """Проверка введенного номера телефона"""
    phone = message.text
    pattern = re.compile(
        r'^((8|\+7)[\- ]?)?(\(?\d{3}\)?[\- ]?)?[\d\- ]{7,10}$')
    if not re.match(pattern, phone):
        return False
    else:
        return phone


def validate_name(message: types.Message):
    """Проверка введенного имени или фамилии"""
    name = message.text
    pattern = re.compile(
        r'^[-a-zA-Za-яA-Яё]+$')
```

```
    if not re.match(pattern, name):
        return False
    else:
        return name


async def validate_date(message: types.Message, my_date):
    """Проверка на валидность введенной даты.  Принимается дата в формате
    ДД.ММ.ГГГГ ."""
    pattern = re.compile('(^(0[1-9]|[12][0-9]|3[01])[.](0[1-9]|1[012])[.]('
                         '19|20)[0-9]{2}$)')
    if not re.match(pattern, my_date):
        await bot.send_message(text=VALIDATE_DATE_TEXT,
                               chat_id=message.from_user.id,
                               reply_markup=remove_kb())
        formalized_date = None
        return formalized_date
    else:
        formalized_date = datetime.strptime(my_date, "%d.%m.%Y").date()
        return formalized_date


def create_csv_file(orders):

    fieldnames = ['Имя', 'Фамилия', 'Телефон', 'Дата меню',
                  'Название меню', 'Порций', 'Оплата', 'Сдача с',
                  'Способ получения', 'Адрес доставки', 'Комментарий']
    cnt_users = 0
    cnt_servings = 0
    with open('orders.csv', mode='w', encoding='windows-1251',
              newline='') as f:
        writer = csv.writer(f, delimiter=';')
        writer.writerow(fieldnames)
        for order in orders['json']:
            csvdata = [
                order['user']['input_firstname'],
                order['user']['input_lastname'],
                order['user']['input_phone'],
                order['menu']['date_of_menu'],
                order['menu']['title'],
                order['num_of_servings'],
                order['payment_type'],
                order['cash_change'],
                order['delivery'],
                order['delivery_address'],
                order['comment']
            ]
            writer.writerow(csvdata)
            cnt_users += 1
            cnt_servings += int(order['num_of_servings'])
    caption = (f'Всего заказало человек: {cnt_users} \n'
               f'Общее количество порций: {cnt_servings}\n'
               f'Подробности в документе')
    f.close()
    file = FSInputFile("orders.csv",
                       filename=f"Заказы на"
                                f" {order['menu']['date_of_menu']}.csv")
    return file, caption
```

```
def check_case_serving(count):
    """Подбор правильного окончания для слово "порция"."""
    mes1 = 'порция'
    mes2 = 'порции'
    mes3 = 'порций'
    mes_count = ''
    try:
        count = int(count)
        if count % 10 == 1:
            mes_count = mes1
        elif 1 <= count % 10 <= 4:
            mes_count = mes2
        elif 5 <= count % 10 <= 9 or count % 10 == 0:
            mes_count = mes3
        return mes_count
    except TypeError:
        pass
```

**bot_app/bot_create.py**

```
import os

from aiogram import Bot, Dispatcher
from aiogram.client.session.aiohttp import AiohttpSession
from aiogram.dispatcher.fsm.storage.memory import MemoryStorage
from aiogram.dispatcher.fsm.storage.redis import RedisStorage
from aiogram.dispatcher.webhook.aiohttp_server import (SimpleRequestHandler,
                                    setup_application)
from aiohttp import web
from dotenv import load_dotenv

load_dotenv()

TELEGRAM_TOKEN = os.getenv('TELEGRAM_TOKEN')
REDIS_DSN = os.getenv('REDIS_DSN')

BASE_URL = os.getenv('BASE_URL')
WEB_SERVER_HOST = os.getenv('WEB_SERVER_HOST')
WEB_SERVER_PORT = os.getenv('WEB_SERVER_PORT')
WEBHOOK_PATH = os.getenv('WEBHOOK_PATH')
WEBHOOK_URL = f'{BASE_URL}{WEBHOOK_PATH}'
IS_REDIS_STORAGE = os.getenv('IS_REDIS_STORAGE')

session = AiohttpSession()
bot_settings = {
    "session": session,
    "parse_mode": "HTML"
}
bot = Bot(token=TELEGRAM_TOKEN, **bot_settings)

if IS_REDIS_STORAGE.lower() in ['true', '1', 'yes']:
    storage = RedisStorage.from_url(REDIS_DSN)
else:
    storage = MemoryStorage()
dp = Dispatcher(storage=storage)
```

```
def start_webhook():
    """ Запускает бота в режиме webhook"""
    app = web.Application()
    handler = SimpleRequestHandler(dispatcher=dp, bot=bot)
    handler.register(app, path=WEBHOOK_PATH)
    setup_application(app, dp, bot=bot)
    web.run_app(app, host=WEB_SERVER_HOST, port=WEB_SERVER_PORT)
```