МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____ 2025 г.

Разработка Telegram-бота для автомоечного комплекса

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2025.406 ПЗ ВКР

Руководитель работы,
к.пед.н., доцент каф. ЭВМ
_____Ю.Г. Плаксина
«___»_____ 2025 г.

Автор работы,
студент группы КЭ-406
_____В.С. Скворцов
«___»_____ 2025 г.

Нормоконтролёр,
ст. преподаватель каф. ЭВМ
_____С.В. Сяськов
«___»_____ 2025 г.

Челябинск-2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Д.В. Топольский
«___»_____2025 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-406
Скворцову Владимиру Сергеевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Разработка Telegram-бота для автомоечного комплекса» утверждена приказом по университету от «21» апреля 2025 г. № 648-13/12.

2. **Срок сдачи студентом законченной работы:** «01» июня 2025 г.

3. **Исходные данные к работе:**

3.1. Нефункциональные требования для запуска Telegram на ПК:

3.1.1. Операционная система Microsoft Windows 7 и выше

3.1.2. Минимальная частота процессора 2 ГГц.

3.1.3. Не менее 2 ГБ оперативной памяти.

3.1.4. Видеокарта с не менее2 ГБ памяти.

3.2. Нефункциональные требования для запуска Telegram на телефоне:

3.2.1 Объем оперативной памяти от 2 ГБ и выше.

3.2.2 Скорость мобильного интернета от 100 Кбит/с.

3.3. Функциональные требования к Telegram-боту для автомоечного комплекса:

3.3.1. Запись в автомоечный комплекс с выбором даты и времени.

3.3.2. Возможность просмотра и удаления личной записи пользователя.

3.3.3. Построение маршрута до комплекса.

3.3.4. Просмотр цен и услуг комплекса, правила и схемы расположения.

3.3.5. Наличие панели на работу администратора.

4. Перечень подлежащих разработке вопросов:

4.1. Аналитический обзор научно-технической литературы.

4.2. Проектирование архитектуры Telegram-бота.

4.3. Разработка Telegram-бота.

4.4. Функциональное тестирование разработанного Telegram-бота.

5. Дата выдачи задания: «02» декабря 2024 г.

Руководитель работы _____/*Ю.Г. Плаксина*/

Студент _____/*В.С. Скворцов*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Аналитический обзор научно-технической литературы	03.03.2025	
Проектирование архитектуры Telegram-бота	22.03.2025	
Разработка Telegram-бота	12.04.2025	
Функциональное тестирование разработанного Telegram-бота	26.04.2025	
Компоновка текста работы и сдача на нормоконтроль	22.05.2025	
Подготовка презентации и доклада	30.05.2025	

Руководитель работы _____/ *Ю.Г. Плаксина* /

Студент _____/ *В.С. Скворцов* /

Аннотация

В.С. Скворцов. Разработка Telegram-бота для автомоечного комплекса. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШ ЭКН; 2025, 70 с., библиогр. список – 20 наим.

Выпускная квалификационная работа посвящена разработке Telegram-бота. В рамках работы рассмотрены основные аспекты проектирования и реализации бота, а также его интеграция с существующими бизнес-процессами автомоечного комплекса. В первой части выпускной квалификационной работе проведен анализ существующих решений в области автоматизации услуг автомоечного комплекса и рассмотрены преимущества использования Telegram-ботов. Во второй части описан процесс проектирования, включая выбор технологий, архитектуру приложения и реализацию функционала. Для реализации были выбраны технологии: язык программирования Python, СУБД – SQLite и среда разработки PyCharm. Архитектура Telegram-бота включает в себя следующие файлы: файл запуска, файл обработки запросов, файл клавиатуры, файл моделей таблиц СУБД и файл запросов в СУБД. В третьей части представлен подробный процесс разработки Telegram-бота. Заключительная часть посвящена оценке эффективности разработанного решения и его перспективам дальнейшего развития. В результате выполнения работы планируется создать инструмент, который позволит упростить взаимодействие с клиентами, и оптимизировать внутренние процессы управления.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 8
1. АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОЙ ЛИТЕРАТРЫ 9
1.1. БЕЗОПАСНОСТЬ TELEGRAM-БОТОВ 10
1.2. АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОЙ ЛИТЕРАТУРЫ 12
1.3. ОБЗОР АНАЛОГОВ TELEGRAM-БОТОВ ДЛЯ АВТОМОЕЧНОГО КОМПЛЕКСА 16
1.4. ВЫВОД ПО РАЗДЕЛУ ОДИН 20
2. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ TELEGRAM-БОТА 21
2.1. ФАЙЛОВАЯ АРХИТЕКТУРА 21
2.2. ЯЗЫК ПРОГРАММИРОВАНИЯ 22
2.3. БАЗА ДАННЫХ 31
2.4. СРЕДА РАЗРАБОТКИ 33
2.5. ВЫВОД ПО РАЗДЕЛУ ДВА 35
3. РАЗРАБОТКА TELEGRAM-БОТА 36
3.1. ФАЙЛ ЗАПУСКА TELEGRAM-БОТА 36
3.2. ФАЙЛ ОБРАБОТКИ ЗАПРОСОВ TELEGRAM-БОТУ 36
3.3. ФАЙЛ ДЛЯ КЛАВИАТУРЫ В TELEGRAM-БОТЕ. 37
3.4. ФАЙЛ РАЗРАБОТКИ МОДЕЛЕЙ ДЛЯ БАЗЫ ДАННЫХ 40
3.5. РАЗРАБОТКА ФАЙЛА ЗАПРОСОВ В БАЗУ ДАННЫХ. 41
3.6. РАЗРАБОТКА КНОПОК «МЕСТО НАХОЖДЕНИЕ КОМПЛЕКСА» И «СТАТИСТИКА» 42
3.7. ВЫВОД ПО РАЗДЕЛУ ТРИ 43
4. ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО TELEGRAM-БОТА 44
4.1. ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ 45
4.2. ВЫВОД ПО РАЗДЕЛУ ЧЕТЫРЕ 53
5. ЗАКЛЮЧЕНИЕ 54
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 55

ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ФАЙЛА ЗАПУСКА TELEGRAM-БОТА 58
ПРИЛОЖЕНИЕ Б ИСХОДНЫЙ КОД ФАЙЛА ОБРАБОТКИ ЗАПРОСОВ TELEGRAM-БОТУ 59
ПРИЛОЖЕНИЕ В ИСХОДНЫЙ КОД ФАЙЛА ДЛЯ СОЗДАНИЯ КНОПОК В TELEGRAM-БОТЕ 64
ПРИЛОЖЕНИЕ Г ИСХОДНЫЙ КОД ФАЙЛА РАЗРАБОТКИ МОДЕЛЕЙ ДЛЯ БАЗЫ ДАННЫХ 67
ПРИЛОЖЕНИЕ Д ИСХОДНЫЙ КОД ФАЙЛА ЗАПРОСОВ В БАЗУ ДАННЫХ 68

ВВЕДЕНИЕ

Современные технологии стремительно проникают во все сферы нашей жизни, включая сферу услуг. Одним из примеров такого взаимодействия является использование мессенджеров для автоматизации и оптимизации бизнес-процессов. В последние годы Telegram стал одним из самых популярных мессенджеров, предоставляющим широкие возможности для создания ботов, которые могут выполнять различные функции от обработки заказов до предоставления информации в режиме реального времени.

Автомоечные комплексы, как важный элемент инфраструктуры обслуживания автомобилей, получают прибыль за счет внедрения современных технологий. Эффективное управление клиентскими запросами, автоматизация записи на мойку, информирование о текущих акциях и услугах – все это может быть реализовано с помощью Telegram-бота. Такой подход не только повышает уровень сервиса, но и способствует увеличению клиентской базы и улучшению финансовых показателей бизнеса.

Целью выпускной квалификационной работы является разработка функционального и удобного Telegram-бота для автомоечного комплекса, который позволит автоматизировать процессы записи на услуги, управления заказами и взаимодействия с клиентами, а также повысить эффективность работы бизнеса и уровень удовлетворенности пользователей.

1. АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОЙ ЛИТЕРАТРЫ

Telegram-бот – это автоматизированный аккаунт в мессенджере Telegram, предназначенный для выполнения различных задач и обработки сообщений. Боты позволяют пользователям взаимодействовать с сервисами и получать информацию через текстовые команды и сообщения. Они работают на основе Telegram Bot API, что позволяет разработчикам создавать функциональные и удобные инструменты для автоматизации процессов.

Telegram-боты для автомоечного комплекса должно иметь следующий функционал:

1. Упрощение записи на услуги. Бот предоставляет пользователям возможность быстро записываться на мойку, выбирая удобное время и тип услуги. Это позволяет сократить время, затрачиваемое на организацию посещения.
2. Предоставление актуальной информации. Бот информирует пользователей о доступных услугах и их ценах. Это помогает клиентам принимать осознанные решения и выбирать необходимые услуги.
3. Напоминания о записях. Бот может отправлять напоминания о совершенных записях на мойку, что помогает пользователям не забывать о своих планах и избегать пропусков.
4. Улучшение клиентского опыта. Бот создает удобный и доступный интерфейс для взаимодействия с сервисом, что повышает общий уровень удовлетворенности клиентов и их лояльность.
5. Сбор статистики и аналитики. Бот может собирать данные о предпочтениях пользователей, что позволяет владельцам автомойки лучше понимать спрос и адаптировать свои услуги под потребности клиентов.

6. Автоматизация процессов. Бот облегчает работу сотрудников автомойки, снижая нагрузку на администрацию и позволяя им сосредоточиться на предоставлении качественного обслуживания.

1.1. БЕЗОПАСНОСТЬ TELEGRAM-БОТОВ

Безопасность при использовании Telegram-ботов – важнейший аспект, который обеспечивает защиту данных пользователей, предотвращает несанкционированный доступ и соблюдение законодательства о защите персональных данных. Ниже подробно рассмотрены ключевые особенности и меры безопасности, связанные с использованием Telegram-ботов.

1. Шифрование и протоколы передачи данных.

Защищённый канал связи. Сообщения между пользователем и ботом передаются через платформу Telegram, которая использует собственные протоколы шифрования для обеспечения конфиденциальности данных. В частности, Telegram применяет протокол MTProto, обеспечивающий шифрование сообщений на уровне транспортного слоя.

End-to-end шифрование. Важно отметить, что стандартные чаты в Telegram (обычные чаты с пользователями или ботами) не используют end-to-end шифрование по умолчанию. Однако секретные чаты в Telegram используют end-to-end шифрование, что исключает возможность перехвата данных третьими лицами. Для ботов, как правило, применяется стандартное шифрование канала передачи данных, что обеспечивает безопасность во время передачи.

Защита от перехвата и подделки. Использование HTTPS (SSL/TLS) при взаимодействии с API Telegram гарантирует безопасность данных на уровне транспортного протокола. Это предотвращает возможность перехвата или изменения сообщений злоумышленниками.

2. Ограничение доступа.

Аутентификация пользователей. Пользователи взаимодействуют с ботом через свои аккаунты в Telegram. Внутри платформы реализована

система аутентификации – пользователь должен авторизоваться через свой аккаунт, что обеспечивает контроль доступа.

Контроль доступа к функциям бота. Разработчики могут реализовать дополнительные уровни авторизации внутри бота – например, проверку ID пользователя или групповых прав для ограничения доступа к определённым функциям или информации.

Ограничение по IP и API-токенам. Доступ к API Telegram осуществляется через уникальные токены бота. Их хранение и использование должны быть защищены от утечки, чтобы предотвратить несанкционированный доступ к управлению ботом.

3. Хранение данных.

Место хранения. Данные, полученные от пользователей (например, личная информация, история взаимодействий), могут храниться на серверах разработчика или сторонних системах для дальнейшей обработки.

Меры защиты хранения. Для обеспечения безопасности необходимо использовать защищённые серверы с актуальными средствами защиты (шифрование данных в базе данных, ограничение доступа по ролям, регулярные обновления системы).

Уязвимости хранения. Некорректная настройка серверов или слабая защита баз данных могут стать уязвимостью – злоумышленники могут получить доступ к конфиденциальной информации.

4. Обработка персональных данных.

Законодательство и нормативы. Обработка персональных данных должна соответствовать требованиям законодательства (например, GDPR в ЕС или локальным законам РФ). Это включает получение согласия пользователя на обработку его данных, информирование о целях их использования и обеспечение прав на доступ и удаление информации.

Безопасность хранения и передачи. Все данные должны передаваться по защищённым каналам (например, через HTTPS), а также храниться в зашифрованном виде при необходимости.

Документирование процессов. Необходимо вести документацию о сборе, обработке и хранении персональных данных для соблюдения требований законодательства и возможности аудита [2].

Безопасность Telegram-ботов – комплексная задача, включающая использование современных протоколов шифрования для защиты передаваемых данных, контроль доступа через платформу Telegram и внутренние механизмы авторизации, а также надежное хранение информации с учетом требований законодательства о защите персональных данных. Для обеспечения высокого уровня безопасности важно регулярно обновлять программное обеспечение, следить за уязвимостями систем и соблюдать лучшие практики по защите информации. Только так можно минимизировать риски утечки или неправомерного использования данных и обеспечить доверие пользователей к автоматизированным сервисам.

1.2. АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОЙ ЛИТЕРАТУРЫ

При разработке Telegram-бота можно столкнуться с рядом сложностей, которые могут усложнить процесс. Рассмотрим одни из самых распространённых проблем.

Одна из первых проблем – ограничения Telegram API. Бот не может отправлять сообщения слишком часто, иначе Telegram может временно заблокировать его за спам. Также есть лимит на длину сообщений (не более 4096 символов), а при использовании callback-запросов могут возникать задержки, особенно если бот работает под высокой нагрузкой.

Ещё одна частая сложность связана с настройкой вебхуков. Telegram требует, чтобы сервер бота работал по HTTPS, а значит, разработчику нужно SSL-сертификат. Это усложняет локальное тестирование – приходится использовать прокси-сервисы вроде ngrok, чтобы временно открыть доступ к боту из интернета. Если URL вебхука изменится, например, при переезде на другой сервер, его нужно правильно обновить, иначе бот перестанет отвечать.

Ошибки в логике бота – ещё один источник проблем. Если не продумать систему состояний, пользователь может "застрять" в каком-то шаге диалога и не суметь продолжить. А если callback-данные, например, нажатия кнопок обрабатываются некорректно, бот может терять важную информацию между запросами. Не менее опасны проблемы с валидацией ввода – если бот не проверяет, что именно отправил пользователь, это может привести к ошибкам или даже уязвимостям.

С ростом популярности бота могут проявиться проблемы с производительностью. Например, если бот делает запросы к внешним API, например, для проверки погоды или курса валют, задержки этих запросов будут влиять на скорость ответа. А если бот внезапно станет популярным и число пользователей резко вырастет, сервер может не справиться с нагрузкой – особенно если изначально не закладывалось масштабирование.

Безопасность – ещё один критичный аспект. Токен бота, если он попадёт в открытый доступ, позволит злоумышленникам управлять ботом. Если бот работает с базой данных, отсутствие проверки ввода может привести к SQL-инъекциям. Кроме того, сам бот может стать инструментом для фишинга или рассылки спама, если не фильтровать пользовательские данные.

Наконец, даже если технически всё работает, бот может оказаться неудобным для пользователей. Например, если интерфейс перегружен кнопками или требует слишком много шагов для выполнения простых действий, клиенты быстро потеряют интерес. А если бот не сообщает понятных ошибок, пользователи просто не поймут, что пошло не так [3].

Разработка стабильного и эффективного Telegram-бота требует комплексного подхода, учитывающего множество аспектов – от архитектуры до безопасности. В отличие от простых скриптов, Telegram-боты сталкиваются с реальными пользователями, чьи действия не всегда предсказуемы, а нагрузка может резко возрастать. Рассмотрим ключевые моменты, которые помогут создать надежного бота.

Одной из основных проблем в разработке ботов является управление диалогами. Когда бот ведет сложный многоэтапный сценарий, крайне важно отслеживать текущее состояние каждого пользователя. Здесь на помощь приходит Finite State Machine (FSM) – подход, при котором бот явно управляет состояниями диалога. Вместо хаотичной обработки сообщений FSM позволяет четко определить: если пользователь находится на этапе ввода имени, то следующее его сообщение должно обрабатываться как имя, а не как случайная команда. Современные библиотеки, такие как aiogram для Python или telegraf для Node.js, предоставляют удобные инструменты для работы с состояниями, что значительно упрощает реализацию сложных диалоговых сценариев [4].

Не менее важным аспектом является тщательное тестирование. В реальной жизни пользователи могут совершать самые неожиданные действия: отправлять фотографии вместо текста, нажимать кнопки несколько раз подряд или вводить специальные символы. Чтобы бот не ломался в таких ситуациях, необходимо покрыть код различными типами тестов. Юнит-тесты проверяют отдельные функции, например, корректность обработки команды /start, интеграционные тесты имитируют полные диалоговые сценарии, а нагрузочные тесты помогают понять, как бот поведет себя при резком увеличении числа пользователей. Особое внимание стоит уделить пограничным случаям. Проработав эти сценарии заранее, можно избежать многих проблем [5].

Архитектурные решения, вышеописанных проблем, играют ключевую роль в стабильности бота. По мере роста популярности бота нагрузка на него может увеличиться в сотни раз, и, если изначально не заложить возможность масштабирования, это приведет к постоянным отказам. Разделение кода на модули: отдельно обработчики команд, отдельно работа с внешними API, отдельно взаимодействие с базой данных. Асинхронная обработка запросов (async/await) позволяет боту эффективно работать даже при выполнении долгих операций, таких как запросы к сторонним сервисам. Для часто

запрашиваемых ботов стоит реализовать кеширование – это снизит нагрузку как на бота, так и на внешние API. Если бот использует вебхуки, стоит заранее продумать возможность горизонтального масштабирования – запуска нескольких экземпляров сервера с балансировкой нагрузки.

Особое внимание стоит уделить ограничениям Telegram API. Бот не может отправлять более 30 сообщений в секунду – превышение этого лимита приведет к временной блокировке. Длина сообщения ограничена 4096 символами, а частые callback-запросы могут вызывать задержки. Чтобы избежать этих проблем, необходимо реализовать систему очередей сообщений и добавлять искусственные задержки между отправками. Также важно корректно обрабатывать ошибки API.

Безопасность должна быть неотъемлемой частью процесса разработки. Утечка токена бота приведет к тому, что злоумышленники получат полный контроль над ним. Поэтому токен должен храниться в защищенном месте – переменных окружения или специальных сервисах для хранения секретов. При работе с базой данных необходимо использовать параметризованные запросы или ORM, чтобы исключить возможность SQL-инъекций. Все входящие данные от пользователей должны тщательно проверяться: даже безобидное на первый взгляд поле для ввода имени может содержать вредоносный код. Дополнительно стоит реализовать защиту от флуда – ограничение количества запросов от одного пользователя в единицу времени [2].

Реализация этих принципов требует дополнительных усилий на этапе разработки, но окупается в долгосрочной перспективе. Бот, построенный с учетом этих рекомендаций, сможет стабильно работать при любом количестве пользователей, корректно обрабатывать нестандартные ситуации и противостоять потенциальным угрозам. Главное – не рассматривать бота как простой скрипт, а проектировать его как полноценное приложение со своей архитектурой, логикой работы и системами защиты. Такой подход позволит

создать действительно надежного и удобного Telegram-бота, который будет радовать пользователей, а не доставлять проблемы разработчикам.

1.3. ОБЗОР АНАЛОГОВ TELEGRAM-БОТОВ ДЛЯ АВТОМОЕЧНОГО КОМПЛЕКСА

В открытых источниках было найдено несколько аналогово Telegram-бота. Из которых можно получить информацию только о функционале и интерфейсе найденных аналогов.

Аналоги, представленные к обзору, были найдены через поиск Telegram.

«JACUZZI автомойка» – Telegram-бот, который дает возможность записаться на автомоечный комплекс. Данный бот позволяет выбрать дату, время и услуги.

Рассмотрим более подробно данного Telegram-бота:

Позволяет подобрать дату записи на ближайшие 5 дней (рисунок 1).

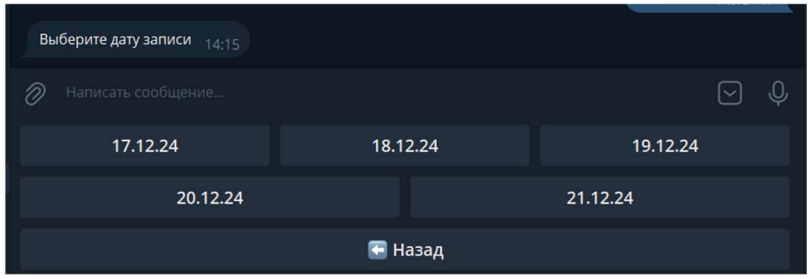

Рисунок 1 – Даты для записи

После выбора даты для записи, клиенту предоставляется подобрать удобное время в промежуток с 9:00 до 20:00 (рисунок 2).

Рисунок 2 – Свободное время для записи

Далее предоставляется выбор услуг (рисунок 3):

1. Комплексная мойка (кузов салон) (~ 45 минут).
2. 2-х фазная мойка кузова (~ 20 минут).
3. Уборка салона (~ 20 минут).
4. Экспресс мойка (~ 5 минут).

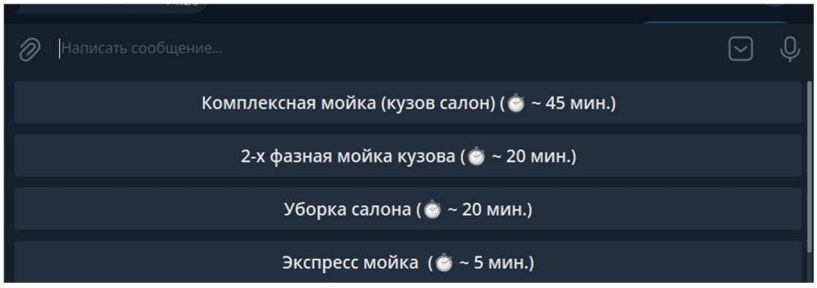

Рисунок 3 – Услуги

После выбора услуг необходимо ввести номер автомобиля для индикации, и номер телефона клиента.

Индикация необходима для проверки номера приехавшего автомобиля со списком записанных автомобилей.

«Avtomoyka_Yuzhnaya8_bot» с помощью этого Telegram-бота так же можно записаться на услуги автомоечного комплекса. Рассмотрим более подробно функции Telegram-бота:

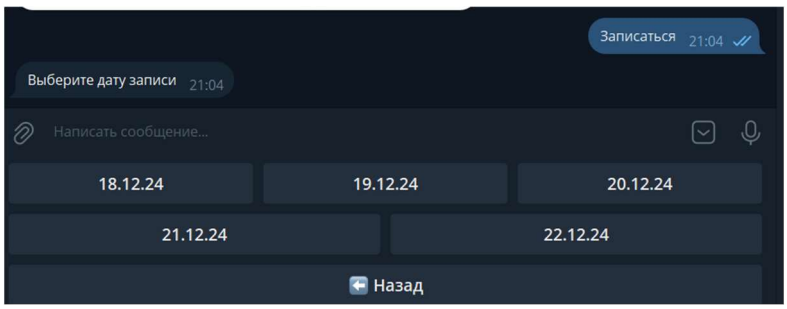

Рисунок 4 – Даты для записи

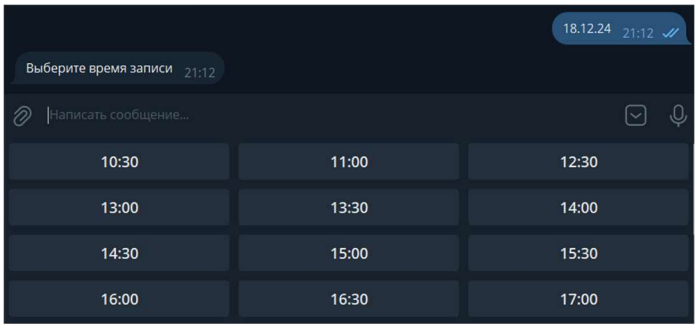

Рисунок 5 – Свободное время для записи

Данный Telegram-бот также как и предыдущий аналог позволяет записаться на определенную дату (рисунок 4) и выбрать подходящее время, с 9:00 до 22:30 (рисунок 5). Далее предлагаются услуги (рисунок 6):

1. Кузов + коврики.
2. Комплекс мини.
3. Комплекс люкс (рисунок 6).

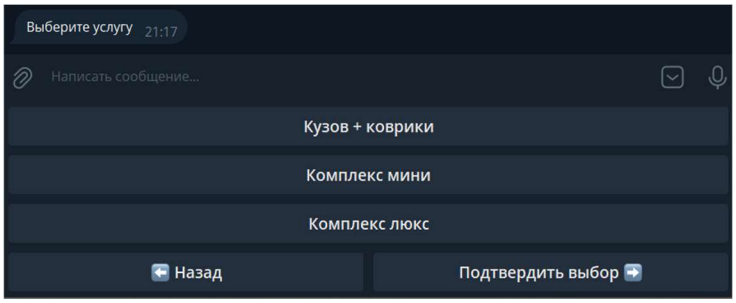

Рисунок 6 – Услуги

После выбора услуг необходимо ввести номер автомобиля для индикации, и номер телефона клиента.

Как можно заметить, рассмотренные два аналога практически полностью идентичны по функционалу, и имеют незначительные различия в интерфейсе.

Выделим несколько критериев и оценим рассмотренные аналоги.

Таблица 1 – Критерии анализа аналогов

Критерии анализа	JACUZZI автомойка	Avtomoyka_ Yuzhnaya8_ bot
Интерфейс	+	+
Просмотр записи	–	–
Визуализация	–	–
Прайс	–	–
Система оповещения	+	+
Место нахождения комплекса	–	–

Описание критериев оценивания:

1. Интерфейс включает в себя: удобство навигации, читаемость текста, логичность структуры (пошаговые действия).
2. Просмотр записи позволяет посмотреть активную запись, увидеть дату и время услуги. Без этой функции клиент не сможет убедиться, что запись создана, что приводит к ошибкам и недовольству.
3. Удаление записи. Данная функция позволяет удалить запись, своевременное освобождение слота. Если клиент не сможет самостоятельно удалять запись, то это приведет к «потери» слота и возможным штрафом к клиенту.
4. Визуализация упрощает новым клиентам ориентироваться в автомоечном комплексе. Без визуализации у клиентов могут возникнуть трудности.
5. Прайс. Список услуг с ценами. Без прайса пользователь не понимает стоимость услуг и может отказаться от записи.
6. Система оповещения присылает напоминает о бронировании за час до записи, снижает количество клиентов, которые не явились.
7. Место нахождение комплекса – одна из основных функций бота, которая поможет новым пользователям без сторонних сайтов или приложений узнать место положение комплекса и проложить до него путь.

Telegram-боты для автомоек JACUZZI и Avtomoyka_Yuzhnaya8_ были проанализированы по семи ключевым критериям. Оба бота получили положительную оценку за удобный интерфейс, что означает наличие понятного меню, кнопок и логичной структуры взаимодействия. Однако они не поддерживают функцию отображения местоположения комплекса, что вынуждает пользователей искать эту информацию на сторонних сайтах или картографических сервисах. Также оба бота не предоставляют возможности просмотра существующих записей, что создает неудобства для клиентов и может приводить к ошибкам в бронировании. Отсутствие функции отмены

записи является еще одним существенным недостатком, ограничивающим возможности пользователей.

Дополнительной проблемой, вышерассмотренных Telegram-ботов, является отсутствие встроенного прайс-листа, из-за чего клиенты не могут оперативно ознакомиться со стоимостью услуг. Единственный положительный аспект, помимо удобного интерфейса – это базовая система оповещений, отправляющая подтверждение записи за час до визита.

Таким образом, несмотря на простой и понятный интерфейс, оба бота обладают серьезными функциональными ограничениями. Для улучшения сервиса рекомендуется реализовать: отображение местоположения с возможностью построения маршрута, функции просмотра и отмены записей, а также встроенный прайс-лист. Эти изменения значительно повысят удобство использования сервиса и снизят количество ошибок при взаимодействии с ботом.

1.4. ВЫВОД ПО РАЗДЕЛУ ОДИН

Telegram-боты для автомоечных комплексов представляют собой эффективный инструмент автоматизации, однако имеют ограничения. Ключевые функциональные возможности таких ботов включают: онлайн-запись с выбором даты и времени, информирование о услугах и ценах, напоминания о записях и сбор статистики. Анализ существующих решений JACUZZI автомойка и Avtomoyka_Yuzhnaya8_bot выявил типовые недостатки: отсутствие функций просмотра и отмены записей, встроенного прайс-листа и местоположения до комплекса.

Основные проблемы разработки Telegram-ботов связаны с ограничениями Telegram API – лимиты на частоту сообщений и длину текста, сложностями настройки вебхуков и обеспечением безопасности данных. Особое внимание требует защита персональных данных пользователей – необходимо использовать шифрование, строгую аутентификацию и безопасное хранение информации.

2. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ TELEGRAM-БОТА

Архитектура Telegram-бота будет включать в себя несколько фалов с расширением .py, в которых будет описан функционал проектируемого бота, а также выбор языка программирования, СУБД и среды разработки.

2.1. ФАЙЛОВАЯ АРХИТЕКТУРА

При разработке программного обеспечения, в том числе Telegram-ботов, крайне важно соблюдать принципы модульности и чистоты кода. Хотя технически возможно реализовать весь функционал в одном файле, такой подход создает существенные проблемы при разработке и сопровождении проекта. Рассмотрим детализированную структуру нашего бота и преимущества разделения кода на задачи.

К таким задачам отнесем:

1. Файл запуска Telegram-бота. Файл, который запускает нашего бота, используя его токен, и даёт возможность к программированию файла обработки запросов, а также создает базу данных и таблицы в ней.
2. Файл обработчик запросов. Это основной файл после файла запуска, так как он обрабатывает запросы пользователя, и именно этот файл отвечает на запросы пользователя, такие как «записаться», «прайс на услуги» и другие функциональные возможности.
3. Файл создания клавиатуры для пользователя. Это файл создает клавиатуру для пользователя, которая помогает реализовать «общение» пользователя и бота. Иными словами, выбор какого-либо элемента клавиатуры будет обрабатываться файлом обработчика запросов и выполнять действия, которые прописаны в коде программы.
4. Файл разработки моделей для базы данных. Это файл, в котором прописана структура для каждой таблицы в базе данных, то есть название таблицы, тип переменных таблицы и «ключ» значение.

5. Файл запросов в базу данных. Этот файл отвечает за заполнение таблицы «Пользователи» и «Запись», которые помогут в дальнейшем отслеживать какой пользователь на какое время был записан. А также в базе данных имеется таблица «Администраторы», которая хранит в себе информацию о конкретных пользователях, и открывает им возможность к кнопке «Админ панель».

Разделение кода на логические модули обеспечивает:

1. Упрощение отладки (ошибки локализуются в конкретном модуле).
2. Повышение читаемости кода.
3. Возможность параллельной разработки разными программистами.
4. Упрощение тестирования отдельных компонентов.
5. Гибкость при внесении изменений (модификация одного модуля не затрагивает другие).

2.2. ЯЗЫК ПРОГРАММИРОВАНИЯ

Telegram-бота можно написать на любом высокоуровневом языке программирования начиная от редко встречаемых такими как PHP и заканчивая популярными, такими как Python или Java. Рассмотри наиболее популярные языки программирования для написания Telegram-бота:

1. Python.

Python – это язык программирования, применяемый в разработке программного обеспечения, в интернет-приложениях, в машинном обучении и науке о данных.

Python является одним из самых востребованных языков программирования для создания ботов в Telegram. Он имеет низкий порог вхождения и предлагает широкий спектр библиотек, таких как python-Telegram-bot, упрощающих процесс разработки. Также Python предоставляет фреймворки и инструменты, которые могут быть полезны при создании ботов. Например, для разработки веб-интерфейсов управления ботами можно

использовать Django и Flask, а библиотеки для машинного обучения, такие как TensorFlow и scikit-learn, помогут в создании умных ботов [1].

Рассмотрим плюсы и минусы Python:

Плюсы:

1. Многофункциональность. Python можно использовать в различных областях, например, научные вычисления, веб-разработка, анализ данных, машинное обучение, искусственный интеллект, автоматизация и многое другое.
2. Простота и читаемость. Простой и понятный синтаксис делает язык Python легким и понятный для изучения, особенно для начинающих программистов.
3. Большое количество библиотек. Существует множество библиотек для работы с Telegram API, таких как python-Telegram-bot, telepot и aiogram.
4. Кроссплатформенность. Python поддерживается на различных операционных системах, таких как macOS, Windows и Linux, это делает его универсальным инструментом для разработки.

Минусы:

1. Производительность. Python является интерпретируемым языком, что делает его медленнее по сравнению с компилируемыми языками, такими как C или Go. Это может быть проблемой для высокопроизводительных приложений.
2. Отсутствие статической типизации. Python динамически типизирован, что может привести к ошибкам, которые выявляются только во время выполнения программы. Это требует более тщательного тестирования.
3. Проблемы с многопоточностью. Из-за GIL (Global Interpreter Lock) Python не может эффективно использовать многопоточность для CPU-bound задач, что может стать ограничением для некоторых приложений.

2. JavaScript (Node.js).

JavaScript – это язык программирования, который используют разработчики для создания интерактивных веб-страниц.

JavaScript (Node.js) применяется для разработки Telegram ботов. Node.js обеспечивает создание высокопроизводительных и масштабируемых приложений, что делает его отличным выбором для ботов с высокой нагрузкой. Так как JavaScript это основной язык веб-разработки, его использование облегчает интеграцию с различными веб-сервисами и API.

Асинхронная модель программирования в Node.js позволяет эффективно обрабатывать множество запросов одновременно, что важно для ботов, которые должны взаимодействовать с большим числом пользователей. Библиотеки, такие как telegraf, предоставляют удобные инструменты для работы с Telegram API [6].

Рассмотри плюсы и минусы JavaScript (Node.js):

Плюсы:

1. Асинхронная природа. Node.js использует неблокирующую, асинхронную модель, что позволяет обрабатывать множество запросов одновременно, обеспечивая высокую производительность и отзывчивость бота.
2. Быстрая разработка. JavaScript – динамический язык, который позволяет быстро писать и тестировать код. Это особенно полезно для создания прототипов и разработки ботов.
3. Библиотеки для создания Telegrem-ботов. Существует множество библиотек для работы с Telegram API, таких как node-Telegram-bot-api и telegraf, которые упрощают процесс разработки и предоставляют множество готовых решений.
4. Универсальность. JavaScript является основным языком для веб-разработки, что позволяет легко интегрировать бота с веб-приложениями и API, а также применять одни и те же навыки и инструменты для фронтенд и бэкенд разработки.

Минусы:

1. Проблемы с производительностью. Хотя Node.js хорошо справляется с I/O задачами, для вычислительно интенсивных задач он может быть менее эффективным по сравнению с языками, такими как Java или C++.
2. Обработка ошибок. Асинхронный код может усложнить обработку ошибок, что может привести к трудностям в отладке и повышенному риску пропуска ошибок.
3. Отсутствие статической типизации. JavaScript – язык с динамической типизацией, что может привести к ошибкам, которые будут обнаружены только во время выполнения, а не на этапе компиляции.
4. Меньшая поддержка для многопоточности. Node.js работает в однопоточном режиме, что может быть ограничением для приложений, требующих высокой параллелизации.
5. Зависимости от внешних библиотек. Использование множества сторонних библиотек может привести к проблемам с совместимостью и сложностям в управлении зависимостями [8, 10].

3. PHP.

PHP – популярный язык программирования с открытым исходным кодом, созданный специально для веб-разработки. Его код можно встраивать напрямую в HTML.

PHP – это язык, используемый в веб-разработке, но и подходящий для создания Telegram-ботов. Для упрощения взаимодействия с Telegram API используются библиотеки, такие как MadelineProto. PHP доступен для многих разработчиков, так как прост и широко распространён.

PHP подходят для разработчиков, которые имеют опыт в веб-разработке и хотят его расширить для разработки Telegram-ботов [6].

Рассмотрим плюсы и минусы язык PHP для Telegram-бота.

Плюсы:

1. Простота и доступность. PHP – это язык, который легко изучить и использовать, особенно для разработчиков, уже знакомых с веб-разработкой. Он широко распространён и поддерживается.
2. Широкая экосистема. Существует множество библиотек и фреймворков для работы с Telegram API, таких как php-Telegram-bot и Telegram-bot-sdk, которые упрощают процесс разработки.
3. Поддержка веб-хуков. PHP хорошо подходит для работы с веб-хуками, что позволяет ботам быстро реагировать на события и сообщения без постоянного опроса API.
4. Интеграция с веб-приложениями. PHP часто используется для создания веб-приложений, что позволяет легко интегрировать бота с существующими сайтами и сервисами.

Минусы:

1. Производительность. PHP может быть менее производительным по сравнению с другими языками, такими как Node.js или Go, особенно в высоконагруженных приложениях или при обработке большого объёма данных.
2. Сложности с асинхронностью. PHP изначально не предназначен для асинхронного программирования, что может усложнить обработку параллельных запросов и задач.
3. Отладка и обработка ошибок. Отладка кода на PHP может быть сложной задачей, особенно в больших проектах, где могут возникать проблемы с обработкой ошибок.
4. Динамическая типизация. PHP имеет динамическую типизацию, что может привести к ошибкам, которые будут обнаружены только во время выполнения, а не на этапе разработки.
5. Зависимость от веб-сервера. PHP требует веб-сервера (например, Apache или Nginx) для работы, что может добавить дополнительный уровень

сложности для развертывания бота, особенно если вы хотите использовать его в режиме реального времени [9].

4. Go.

Go – это компилируемый многопоточный язык с открытым исходным кодом. В основном его применяют в веб-сервисах и клиент-серверных приложениях.

Go – это язык программирования, разработанный компанией Google. Имеет высокую производительность, а библиотеки, например, таких как tgbotapi, дают возможность создавать высокопроизводительных Telegram-ботов. Параллельное программирование – одна из возможностей языка Go, позволяющая как обрабатывать большое количество запросов, так и эффективно использовать ресурсы [6].

Рассмотрим минусы и плюсы языка программирования Go.

Плюсы:

1. Высокая производительность. Go компилируется в машинный код, что обеспечивает отличную производительность. Это делает его подходящим для создания высоконагруженных приложений и ботов, которые должны обрабатывать множество запросов одновременно.
2. Простота синтаксиса. Go имеет простой и понятный синтаксис, что облегчает обучение и разработку. Он сочетает в себе элементы, знакомые разработчикам из других языков, таких как C и Python.
3. Поддержка параллелизма. Go имеет встроенные механизмы для работы с параллелизмом, такие как горутины и каналы. Это позволяет легко обрабатывать несколько задач одновременно, что особенно полезно для ботов, работающих с большим количеством пользователей.
4. Статическая типизация. Go является статически типизированным языком, что позволяет обнаруживать ошибки на этапе компиляции, улучшая надежность кода и упрощая отладку.

5. Легкая компиляция и развертывание. Программы языка Go превращаются в самодостаточный исполняемый файл, избавляя от необходимости устанавливать зависимости при развертывании.
6. Стандартная библиотека. Go имеет мощную стандартную библиотеку, включающую в себя множество функций для работы с HTTP, сетью, JSON и другими форматами данных, что упрощает разработку ботов.

Минусы:

1. Меньшая библиотек. Хотя количество библиотек Go растет, она все еще меньше по сравнению с такими как Python или JavaScript. Это может ограничить доступность библиотек и фреймворков для работы с Telegram API.
2. Сложности с отладкой. Хотя Go имеет инструменты для отладки, некоторые разработчики могут найти их менее удобными по сравнению с инструментами, доступными в других языках.
3. Меньше учебных материалов. По сравнению с более популярными языками, такими как Python или JavaScript, количество учебных материалов и ресурсов для изучения Go может быть ограничено.
4. Узкая специализация. Go часто используется для разработки серверных приложений и микросервисов, что может ограничить его применение в других областях [10].

По данным сайта [11] можно сделать вывод, что для создания телеграмм ботов, используют язык Python.

Для разработки Telegram-бота для автомоечного комплекса наиболее подходящими будут такие языки как: Python и JavaScript. Рассмотрим какой из языков подходит для данной работы.

Таблица 2 – Критерия выбора языка программирования

Критерии	**Python**	**JavaScript**
Использование	+	–
Библиотеки	+	+

Продолжение таблицы 2

Производительность	+	+
Кроссплатформенность	+	+
Асинхронное программирование	+	+

Объяснение критериев оценивания:

2. Использование – Python подходит для разработки бэкенда ботов благодаря специализированным библиотекам, таким как aiogram и python-telegram-bot. JavaScript с фреймворком Telegraf.js можно использовать, но он подходит под веб-интеграцию — например, если боту необходимо взаимодействовать с веб-интерфейсом.
3. Оба языка имеют мощные библиотеки для работы с Telegram API. У языка программирования Python – aiogram, python-telegram-bot, а у JavaScript – Telegraf.js, node-telegram-bot-api.
4. Производительность. Для нашего Telegram-бота, не высоконагруженного, оба языка показывают сопоставимую производительность. JavaScript (Node.js) может быть немного быстрее в I/O-операциях, но Python превосходит в вычислительных задачах, например, при работе с ML.
5. Кроссплатформенность. Python и JavaScript (Node.js) работают на Windows/Linux/macOS без изменений кода. Оба языка позволяют развернуть бота на любом сервере или даже serverless-платформах.
6. Асинхронное программирование. Python имеет async/await, asyncio, а JavaScript: изначально асинхронная модель. Оба отлично подходят для ботов, где много I/O-операций, такие как запросы к API, СУБД и другие.

Для разработки Telegram-бота для автомоечного комплекса был выбран язык программирования Python как наиболее оптимальное решение. Этот выбор обусловлен рядом ключевых преимуществ:

1. Специализированные библиотеки. Python предлагает мощные, хорошо документированные библиотеки для работы с Telegram API, такие как aiogram и python-telegram-bot, которые значительно ускоряют процесс разработки и позволяют реализовать весь необходимый функционал.
2. Асинхронная модель. Встроенная поддержка асинхронного программирования через asyncio идеально подходит для Telegram-бота, где важно эффективно обрабатывать множество одновременных запросов без блокировки основного потока.
3. Интеграция с базами данных. Python предоставляет широкий выбор драйверов и ORM для работы с различными СУБД (PostgreSQL, MySQL, SQLite), что упрощает хранение и обработку данных клиентов, записей и прайс-листов.
4. Производительность. Для задач автомоечного бота (обработка запросов, работа с базой данных, отправка уведомлений) производительности Python более чем достаточно, при этом он менее ресурсоемок по сравнению с JavaScript-решениями.

Хотя JavaScript (Node.js) также обладает возможностями для создания ботов (например, через Telegraf.js), он больше ориентирован на веб-разработку и требует дополнительных усилий для реализации аналогичного функционала. Python же предоставляет все необходимое "из коробки", позволяя сосредоточиться на бизнес-логике проекта, а не на технических сложностях реализации.

Дополнительным преимуществом Python является его популярность в сфере автоматизации и обработки данных, что может пригодиться при дальнейшем расширении функционала бота (например, для аналитики посещаемости или системы лояльности). Большое сообщество разработчиков и обширная база готовых решений гарантируют, что для любой возникающей задачи уже существует проверенное решение.

2.3. БАЗА ДАННЫХ

Современные базы данных различаются по своей архитектуре и могут быть разделены на категории в зависимости от модели данных, системы хранения и методов обработки информации. Рассмотрим несколько основных баз данных:

1. Реляционные базы данных (RDBMS) – это база данных, которая позволяет хранить данные в виде таблиц, которые могут связаны друг с другом. Используют SQL для выполнения запросов [12].
2. Нереляционные базы данных (NoSQL) – база данных, которая не использует традиционные таблицы и реляционные методы. Данная база данных предназначена для работы с большими объемами неструктурированных или полуструктурированных данных. Выделим несколько подкатегорий:

2.1. Документные базы данных – базы данных хранящие данные в формате JSON или BSON (например, MongoDB).

2.2. Ключ-значение – хранят данные в виде пар ключ-значение (например, Redis).

2.3. Графовые базы данных, такие базы данных оптимизированы для хранения и обработки графов и связей между данными (например, Neo4j).

2.4. Столбцовые базы данных – хранят данные в виде колонок, что позволяет эффективно обрабатывать большие объемы данных (например, Cassandra) [13].

3. Объектно-ориентированные базы данных (OODBMS) – базы данные, которые хранят данные в виде объектов, как в объектно-ориентированном программирование. Подходит для приложения со сложными данными [14].

Для разработки Telegram-ботов можно использовать различные базы данных в зависимости от требований проекта. Вот несколько популярных вариантов:

1. SQLite: легковесная реляционная база данных, встроенная в Python через библиотеку sqlite3. Встраиваемая база данных, не требующая отдельного сервера. Подходит для небольших проектов или при разработке на локальной машине. Не подходит для высоких нагрузок и многопользовательских приложений.
2. PostgreSQL: мощная реляционная база данных с поддержкой сложных запросов и транзакций. Хорошо подходит для масштабируемых приложений. Требуется настройка и управление сервером. Используется для больших приложений с большим объемом данных.
3. MySQL: популярная реляционная база данных, часто используемая в веб-приложениях. Подходит для проектов, где требуется высокая производительность и масштабируемость. Имеет некоторые ограничения по типам данных и функциональности по сравнению с PostgreSQL.
4. MongoDB: документо-ориентированная NoSQL база данных, которая хорошо подходит для хранения неструктурированных данных. Подходит для проектов с гибкой структурой данных. Не поддерживает сложные транзакции, так как реляционная база данных.
5. Redis: быстрая NoSQL база данных, работающая в памяти, часто используемая для кэширования и хранения временных данных. Поддерживает структуры данных, такие как списки и множества. Данные хранятся в памяти, что может быть проблемой для больших объемов данных [15].

Проанализировав различные типы баз данных, мы пришли к выводу, что SQLite - оптимальное реляционное решение для нашего проекта. Основные преимущества этого выбора заключаются в том, что она является наиболее

подходящей для разработки Telegram-бота, так как в нашем проекте не будет высокой нагрузки и хранением больших объемов данных.

Рассмотрим более подробно базу данных SQLite и ее основные характеристики:

1. Легковесность. SQLite не требует настройки сервера и может работать прямо из файла, что упрощает развертывание.
2. Простота использования. SQLite имеет простой интерфейс и хорошо интегрируется с Python через стандартную библиотеку sqlite3.
3. Поддержка SQL. SQLite поддерживает большинство стандартных SQL-запросов, что позволяет использовать привычные методы работы с реляционными базами данных.
4. Производительность. Для небольших и средних приложений SQLite предлагает высокую производительность и низкие задержки.
5. Кроссплатформенность. SQLite работает на различных операционных системах, что делает его универсальным выбором [16].

Так же важный плюс использования базы данных - возможность использования в коммерческих продуктах без каких-либо исключений.

2.4. СРЕДА РАЗРАБОТКИ

Современная IDE (Integrated Development Environment) представляет собой универсальную платформу, объединяющую все необходимые инструменты для эффективной разработки ПО: от написания до отладки кода. Благодаря возможностям, интеллектуального автодополнения, подсветки синтаксиса, встроенного отладчик и поддержке систем контроля версий, эти среды существенно повышают продуктивность программистов. [17].

Для языка программирования Python существует несколько популярных сред разработки:

1. PyCharm Community – это профессиональная IDE, разработанная компанией JetBrains специально для Python-разработки. Ключевые преимущества включают:

1.1. Умное автодополнение кода (IntelliSense).

1.2. Встроенный отладчик с визуализацией выполнения.

1.3. Полноценную интеграцию с Git и GitHub.

1.4. Поддержку виртуальных окружений.

1.5. Инструменты для рефакторинга кода.

1.6. Шаблоны для быстрого создания проектов [18].

2. Visual Studio Code (VS Code) – это мощный, но при этом легковесный редактор кода от Microsoft с расширенной поддержкой Python через плагины. Основные возможности:

2.1. Модульная архитектура с тысячами расширений.

2.2. Встроенный терминал и отладчик.

2.3. Интеграция с Docker и удаленными серверами.

2.4. Настраиваемый интерфейс [19].

3. Jupyter Notebook – интерактивная среда, особенно популярная в области анализа данных и научных вычислений. Позволяет:

3.1. Комбинировать исполняемый код с текстом и визуализациями.

3.2. Проводить интерактивные вычисления.

3.3. Создавать документы с живыми примерами.

3.4. Визуализировать данные непосредственно в интерфейсе [20].

Для разработки Telegram-бота автомоечного комплекса была выбрана среда PyCharm Community по следующим причинам:

1. Специализированная поддержка Python без необходимости дополнительных настроек.
2. Встроенные инструменты для работы с базами данных.
3. Удобная интеграция с системами контроля версий.
4. Полнофункциональный отладчик для поиска и исправления ошибок.
5. Возможности рефакторинга для поддержания чистоты кода.
6. Готовые шаблоны для быстрого старта проекта.

Хотя VS Code и Jupyter Notebook также могут использоваться для разработки, они требуют дополнительной настройки и уступают PyCharm в

удобстве работы именно с Python-проектами. Выбор PyCharm Community Edition позволяет сосредоточиться на реализации бизнес-логики бота, а не на настройке рабочего окружения.

2.5. ВЫВОД ПО РАЗДЕЛУ ДВА

В данной главе рассмотрена архитектура Telegram-бота для автомоечного комплекса. Проект реализован на Python с использованием модульной структуры: файл запуска, файл обработки запросов, клавиатуры, модели данных и файл запросов в базу данных. Такой подход обеспечивает четкое разделение логики и упрощает поддержку кода.

Выбор Python обусловлен наличием специализированных библиотек и поддержкой асинхронности. Для хранения данных используется SQLite – легковесная реляционная СУБД, не требующая отдельного сервера и идеально подходящая для проектов с умеренной нагрузкой.

Разработка ведется в PyCharm Community Edition, которая предоставляет полный набор инструментов для Python-разработки: автодополнение кода, встроенный отладчик и интеграцию с системами контроля версий. Выбранный стек технологий обеспечивает баланс между производительностью, простотой разработки и легкостью развертывания.

3. РАЗРАБОТКА TELEGRAM-БОТА

Telegram-бот для автомоечного комплекса состоит из следующих файлов:

1. Файл запуска Telegram-бота.
2. Файл обработки запросов Telegram-боту.
3. Файл для клавиатуры в Telegram-боте.
4. Файл разработки моделей для базы данных.
5. Разработка файла запросов в базу данных.

Также в этой главе рассмотрим разработку кнопок «Место нахождение комплекса» и «Статистика».

3.1. ФАЙЛ ЗАПУСКА TELEGRAM-БОТА

Этот код создает Telegram-бота, который использует асинхронное программирование для обработки сообщений и команд. Подключение к боту происходит за счет токена, который мы получили, когда создавали бота. Он подключается к базе данных, инициализирует обработчики и запускает процесс получения обновлений от Telegram. Это позволяет боту работать эффективно и быстро реагировать на запросы пользователей (рисунок 7).

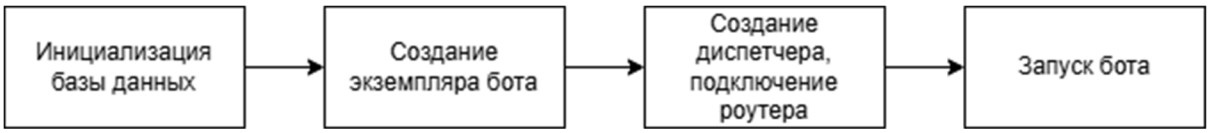

Рисунок 7 – Логика файла запуска

3.2. ФАЙЛ ОБРАБОТКИ ЗАПРОСОВ TELEGRAM-БОТУ

Обработка запросов – следующий по важности файл, который позволяет боту «отвечать» на сообщения пользователя. Для этого боту необходим роутер. Роутер необходим для обработки запросов пользователя в бот. Он позволяет направить данные от Телеграма в нужный метод сценария. (рисунок 8).

Рисунок 8 – Блок-схема алгоритма обработки запросов

3.3. ФАЙЛ ДЛЯ КЛАВИАТУРЫ В TELEGRAM-БОТЕ.

Данный файл создает клавиатуру для пользователя, бот реагирует на сообщение отправленный с нажатия кнопки и выполняет сценарий, который был прописан во втором файле.

Рассмотрим клавиатуру, которую создает бот (рисунок 9).

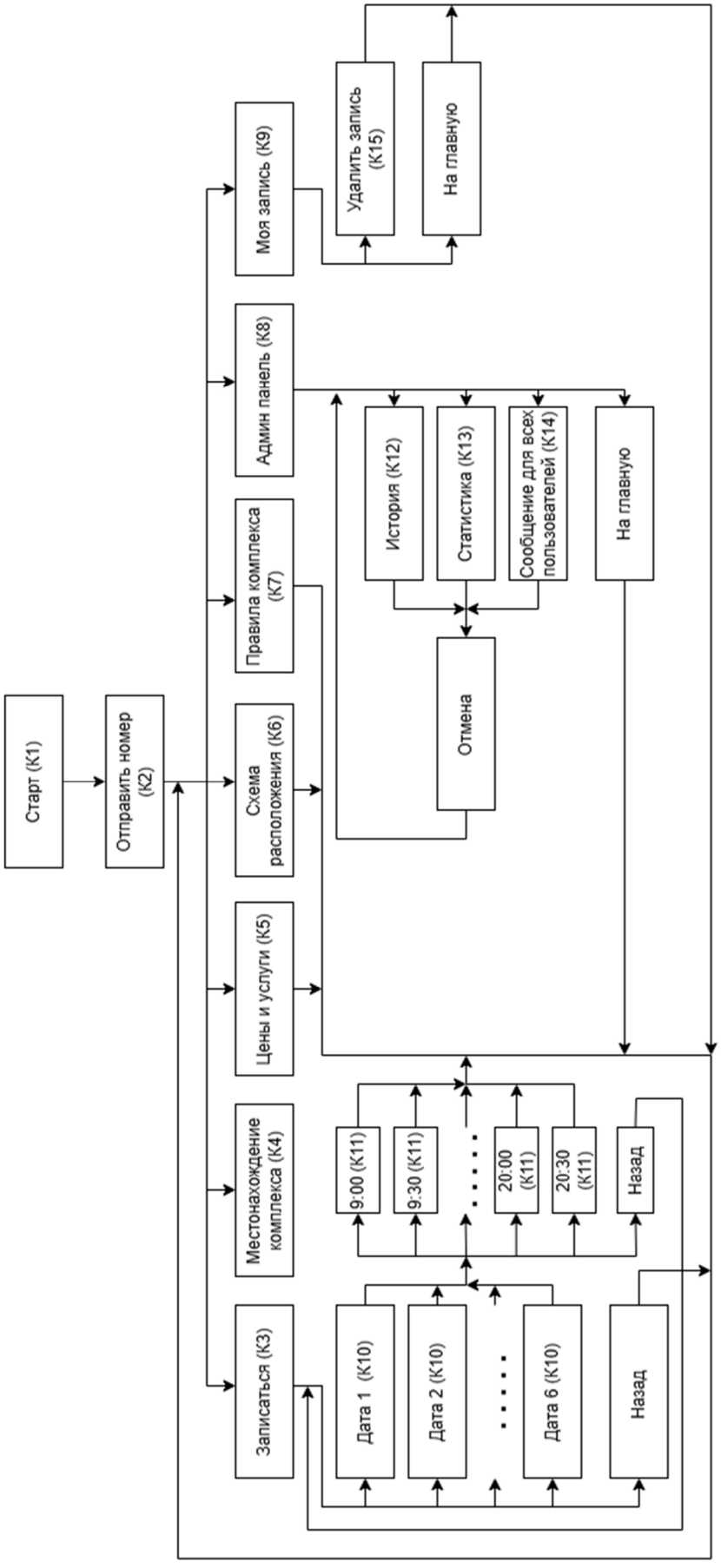

Рисунок 9 – Клавиатура Telegram-бота

Описание клавиатуры, отображаемой в Telegram-боте:

1. Кнопки «Назад», «Отмена» и «На главную» возвращают пользователя, соответственно, на клавиатура созданную ранее или на главное меню, в котором находятся кнопки К1-К9.
2. «К1» – кнопка, созданная Telegram для запуска бота у пользователя.
3. «К2» – кнопка, которая появляется только при регистрации, и автоматически отправляет номер пользователя.
4. «К3» – кнопка, которая запускает процесс записи пользователя на мойку. Нажав на нее у пользователя появиться кнопки с датами «К10».
5. «К10» – даты, начиная от нынешнего дня или со следящего, если время больше 20:30 и на шесть дней вперед. Выбрав дату у пользователя, появятся кнопки «К11».
6. «К11» – свободное время с 9:00 до 20:30, если был выбран нынешний день, то будут кнопки начиная от нынешнего времени.
7. «К4» – кнопка, отправляющая местоположение комплекса, и предоставляет выбор навигатора, который есть на телефоне или навигатора Telegram.
8. «К5» – кнопка, отправляющая текстовое сообщение пользователю, с ценами и услугами автомоечного комплекса.
9. «К6» – кнопка, отправляющая фотографии, на которых показано, направление движение.
10. «К7» – кнопка, отправляющая текстовое сообщение пользователю, с правилами автомоечного комплекса.
11. «К8» – кнопка, которая появляется только администраторам комплекса, данные о которых есть в таблице «Администраторы» базы данных. Данная кнопка открывает доступ к кнопкам «К12», «К13» и «К14».
12. «К12» – просит у пользователя ввести дату, и выдает список записей в этот день или сообщение о том, что записей не было.
13. «К13» – выводит статистику по времени, то есть в какое время дольше всего пользователей записывается в автомоечный комплекс.

14. «К14» – позволяет администраторам написать сообщение, которое будет отображаться у всех текущих пользователей Telegram-бота
15. «К9» – кнопка, появляющаяся только после того, как пользователь записался на мойку и пропадающая после того, как у пользователя пройдет время записи. Дает доступ к кнопке «К15».
16. «К15» – позволяет удалить запись у пользователя.

3.4. ФАЙЛ РАЗРАБОТКИ МОДЕЛЕЙ ДЛЯ БАЗЫ ДАННЫХ

Данный файл создает модели таблиц для базы данных (рисунок 10).

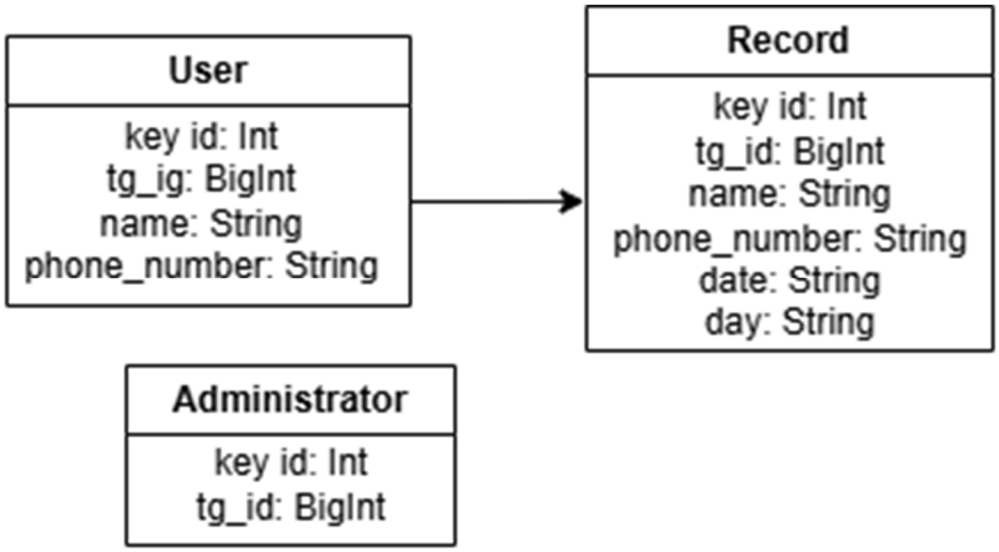

Рисунок 10 – Модель таблиц

Каждая модель отражает конкретную сущность системы и включает в себя поля, типы данных, а также связи между таблицами, что обеспечивает целостность и логическую организацию данных.

Создание таких моделей позволяет автоматизировать процесс генерации таблиц в базе данных, облегчает управление схемой данных при дальнейшем развитии проекта. В дальнейшем эти модели могут быть использованы для выполнения миграций, обновления структуры базы и обеспечения согласованности данных.

3.5. РАЗРАБОТКА ФАЙЛА ЗАПРОСОВ В БАЗУ ДАННЫХ.

Данный файл напрямую взаимодействует с базой данных, заполняя такие таблицы как «Пользователи» и «Запись». Этот файл взаимодействует с таблицами «Администраторы» и «Запись». С помощью таблицы «Администраторы» проверяет пользователей на администратора и обычного клиента, а таблица «Запись» необходима для формирования свободного времени для пользователя ботом. Блок-схема файла запросов в базу данных представлена на рисунке 11.

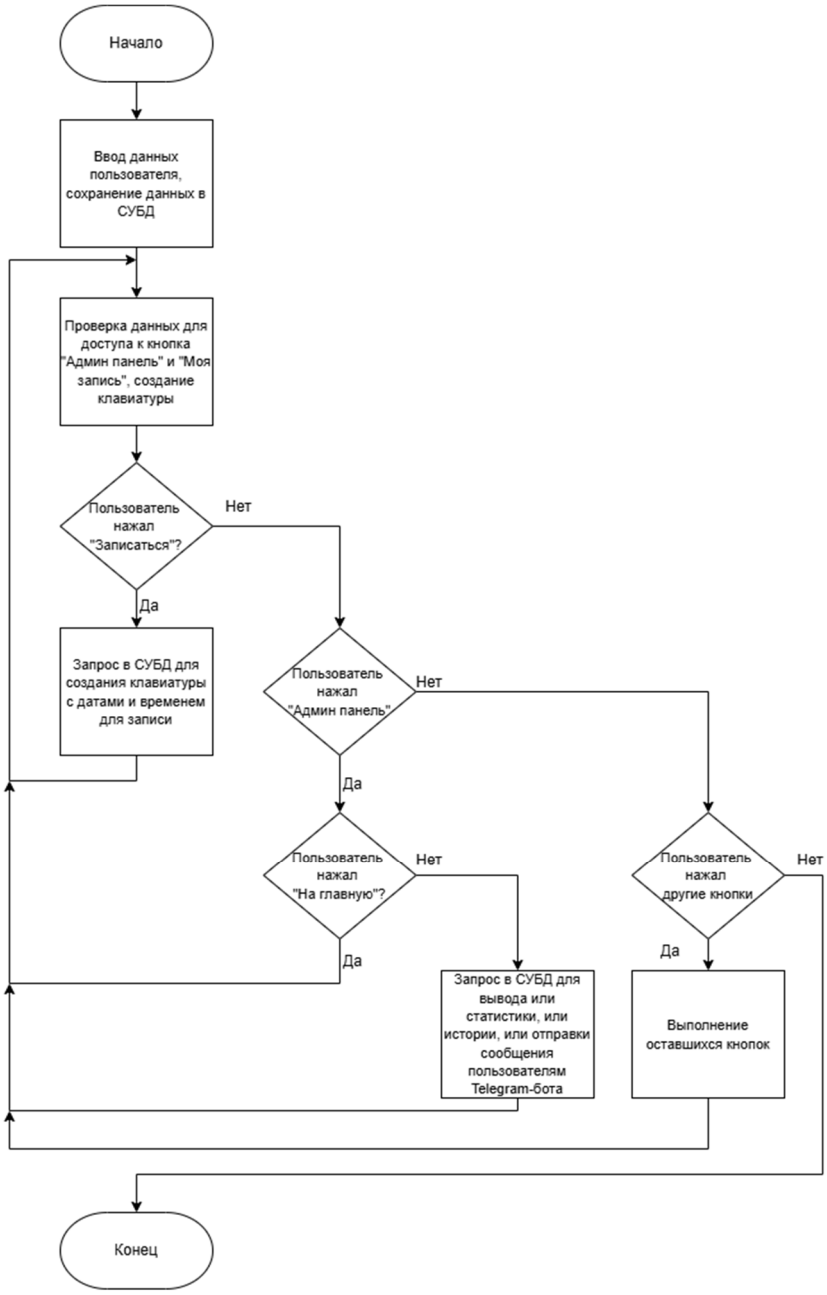

Рисунок 11 – Блок-схема алгоритма запросов в базу данных

3.6. РАЗРАБОТКА КНОПОК «МЕСТО НАХОЖДЕНИЕ КОМПЛЕКСА» И «СТАТИСТИКА»

«Место нахождение комплекса» – это кнопка отправляет пользователю сообщение, с точкой на карте и позволяет построить маршрут в самом Telegram или воспользоваться сторонними приложениями, скаченными у пользователя на устройстве. Исходный код описанной функции представлен в листинге Б.1 приложения Б.

Если пользователь нажимает на кнопку «Место нахождение комплекса», то бот воспроизводит сценарий из листинга Б.1 приложения Б. Задает широту и долготу комплекса, и вызывает функцию «answer_location», которая в свою очередь отправляет сообщение с геолокацией автомоечного комплекса пользователю.

«Статистика» – кнопка, доступная только для администраторов комплекса, которая выводит график по времени записей пользователей Telegram-ботом. Рассмотрим код функции представлен в листинге Б.1 приложения Б.

Если пользователь нажимает на кнопку «Статистика», то бот воспроизводит сценарий из листинга Б.1 приложения Б. Telegram-бот отправляет сообщение пользователю «Предоставляю статистику по времени» и выполнят функцию «rq.statistics». Рассмотрим функцию «rq.statistics» представленную в листинге Д.1 приложения Д.

Функция statistics создает словарь, ключом которым является время записей и значение – количество записей пользователей на это время. Для заполнения значений словаря создаем запрос к таблице «Record» и сохраняем в переменную times. С помощью цикла проходим по созданному массиву со временем. Если время с массива times соответствует ключу в словаре «time_dict», то увеличиваем количество значения у советующего ключа на единицу. Для создания графики используем функцию «plot_statistics», в

которую передаем заполненный словарь. Рассмотрим код функции plot_statistics представленной в листинге Д.1 приложения Д.

Для создания графика необходимо импортировать библиотеку, позволяющую строить графики. Для постройки графика необходимо задать размер графика, поля и переменные, по которым стоится график. Создав график по данным из переданного словаря бот сохраняет картинку на компьютер, отправляет ее пользователю и сразу же удаляет.

3.7. ВЫВОД ПО РАЗДЕЛУ ТРИ

Разработанный Telegram-бот для автомоечного комплекса представляет решение на базе aiogram. В основе системы лежит модульная архитектура разделением функционала.

Бот предлагает интуитивно понятный интерфейс с пошаговой записью на услуги, отображением цен и местоположения комплекса. Для администраторов реализованы специальные функции: просмотр статистики посещаемости в виде автоматически генерируемых графиков и управление записями.

Особенностью решения является гибкая система управления доступом, разделяющая функционал для клиентов и персонала. Асинхронная архитектура обеспечивает работу при любом количестве одновременных запросов. Все данные хранятся в оптимизированной SQLite-базе, включающей три взаимосвязанные таблицы: пользователи, записи, администраторы.

Разработанная структура проекта позволяет масштабировать функционал и адаптировать Telegram-бота под конкретные бизнес-процессы автомоечного комплекса, сохраняя при этом производительность и удобство использования.

4. ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО TELEGRAM-БОТА

Проведем функциональное тестирование разработанного Telegram-бота для автомоечного комплекса. Функционал, подлежащей проверке:

1. При вводе команды /start или первым запуском бота начинается процесс регистрации нового клиента. Кнопка «Отправить номер» должна работать исправно. Данные о пользователе сохраняться в таблицу «Пользователи».
2. Все кнопки должны быть нажиматься и работать исправно.
3. Дата записи на мойку должна иметь хотя бы один пустой слот времени или не превышать время 20:30, иначе создавать кнопку со следующим днем.
4. Слоты с занятым временем не должны появляться у пользователей.
5. После создания записи в таблице «Запись» появляться данные о записи пользователя, также появляется кнопка «Моя запись», нажав на которую появляется время и дата записи клиента, и другая кнопка с возможностью удалить запись.
6. Кнопки «Место нахождения комплекса», «Цены и услуги», «Схема расположения» и «Правила комплекса» должны выводить информацию о автомоечном комплексе.
7. Проверка работоспособности кнопки «Админ панель» и появление ее на главном экране, только если данные о пользователи занесены в таблицу «Администраторы».
8. Кнопки «История», «Статистика» и «Сообщения для всех пользователей» должны работать исправно и выводить историю по указанной дате, статистику по времени посещения и отправки сообщения для всех пользователей.

4.1. ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ

При первом использовании бота, у пользователя появиться кнопка «Старт», после нажатия на которую бот попросит ввести имя (рисунок 12).

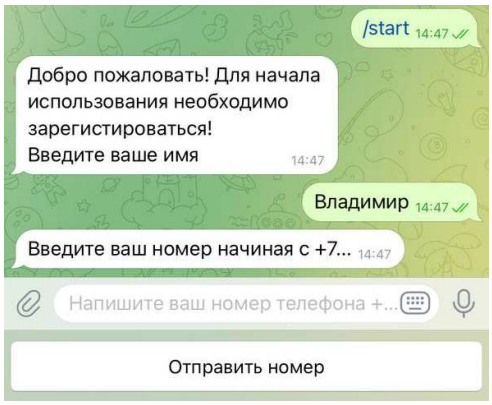

Рисунок 12 – Запуск бота

Введя имя пользователю представиться возможно ввести свой номер вручную или использовать кнопку, которая сделает все автоматически. Нажав на нее, пользователь получит следующее сообщение (рисунок 13).

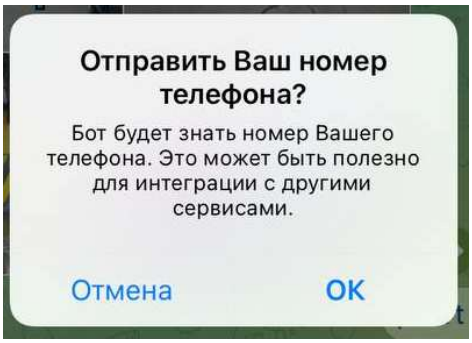

Рисунок 13 – Всплывающее окно

После нажатия на «ОК» пользователь получит сообщение от бота о успешной регистрации, а также появляется другие клавиши, такие как «Записаться», «Местонахождение комплекса», «Цены и Услуги», «Правила комплекса» и «Схема расположения» (рисунок 14).

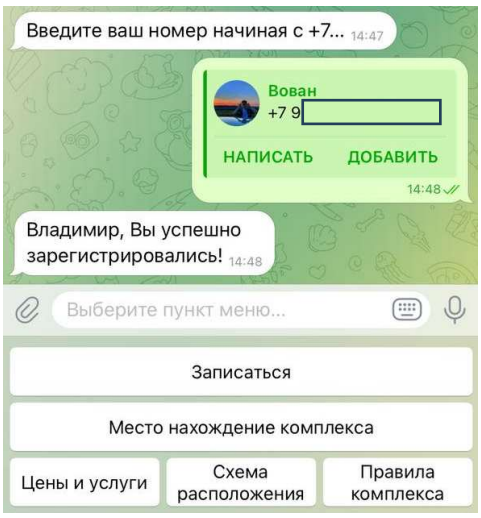

Рисунок 14 – Клавиатура с новыми возможностями

Рассмотрим кнопку «Записаться». При нажатии на нее у пользователя появиться выбор даты (рисунок 15), 6 кнопок в формате «ДД.ММ.ГГГГ» начиная от сегодняшнего дня и следующие 5 дней.

Рисунок 15 – Кнопки с датами

Выбрав дату, появляются кнопки со свободным временем с 9:00 до 20:30 (рисунок 16).

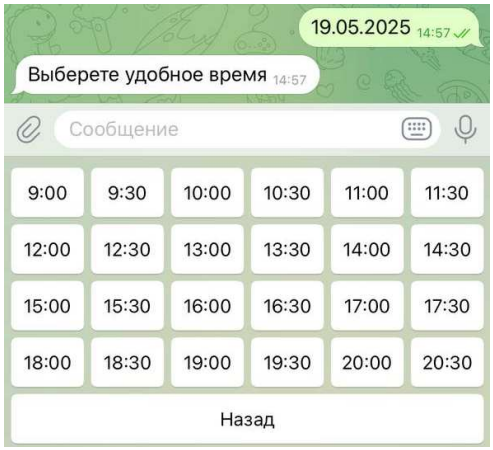

Рисунок 16 – Кнопки со свободным временем

Выбрав свободное время, у пользователя, появится новая кнопка «Моя запись» (рисунок 17).

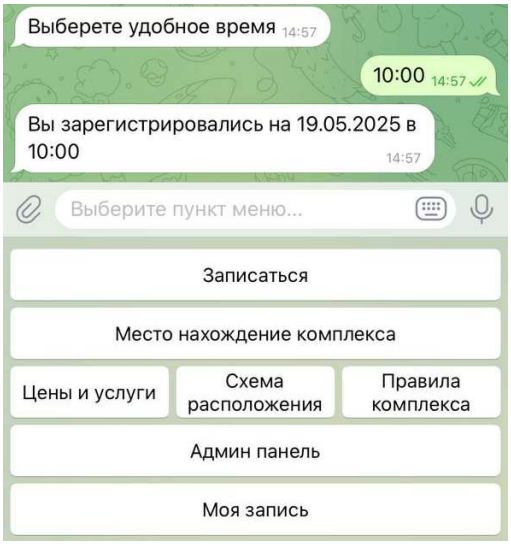

Рисунок 17 – кнопка "Моя запись"

Проверим, пропало ли время, которое только что зарегистрировали. Для этого нажмем кнопку «Записаться» и выберем дату 19.05.2025 (рисунок 18).

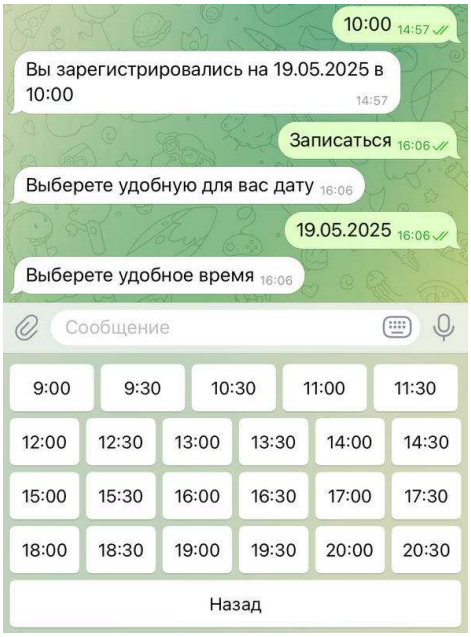

Рисунок 18 – Проверка свободного времени

Можно заметить, что на отправленных Telegram-ботом кнопок отсутствует слот со временем 10:00, то время, которое было зарезервировано, а значит другой пользователь не сможет забронировать время, занятое другим пользователем.

Дальше проверим функционал кнопки «Моя запись», нажав на нее появляется возможность удалить запись (рисунок 19).

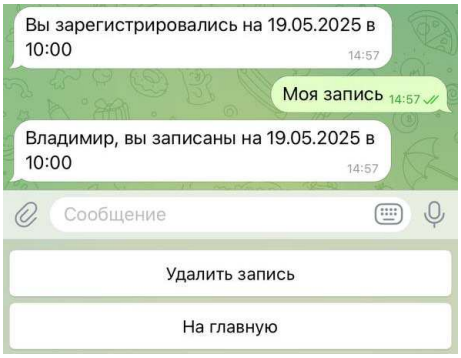

Рисунок 19 – Функционал кнопки "Моя запись"

Нажмем на кнопку «Удалить запись» для проверки работы функционала (рисунок 20). Проверим так же, появиться ли свободный слот со временем, если нажать на главном меню кнопку «Записаться» и выбрать дату 19.05.2025 (рисунок 21).

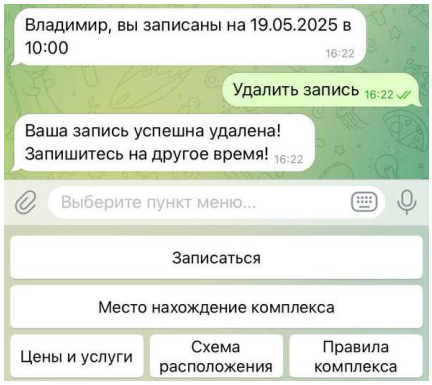

Рисунок 20 – Удаление записи

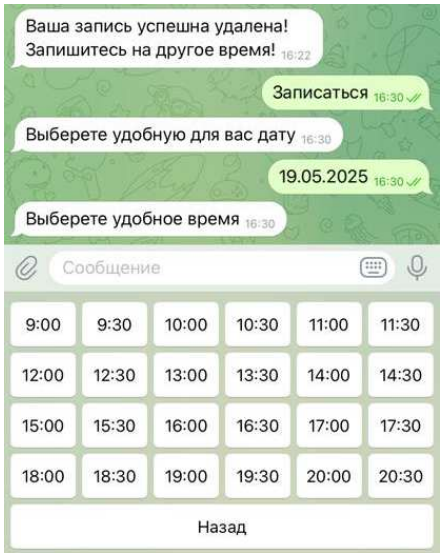

Рисунок 21 – Проверка свободного времени

Можно заметить, после удаления записи появился свободный слот со временем на 10:00.

Дальше проверим функционал кнопом «Место нахождение комплекса» и «Цены и услуги», «Схема расположения» и «Правила комплекса» (рисунок 22).

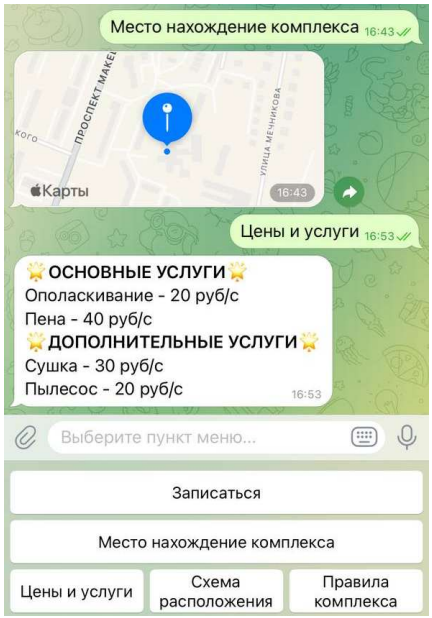

Рисунок 22 – "Местонахождение комплекса" и "Цены и услуги"

Рассмотрим функционал кнопок «Схема расположения» и «Правила комплекса» (рисунки 23, 24).

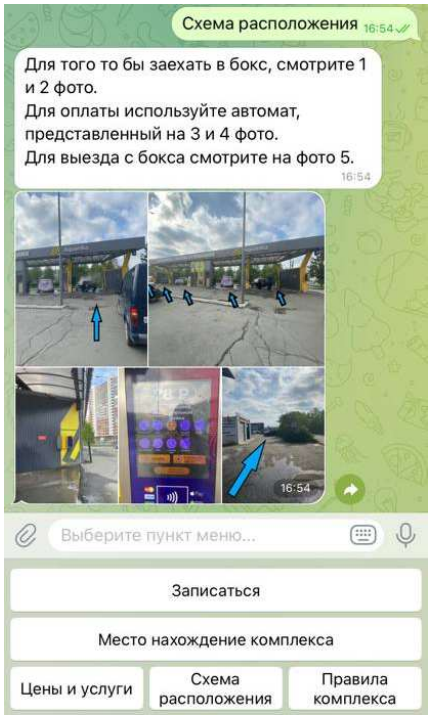

Рисунок 23 – функционал кнопки "Схема расположения"

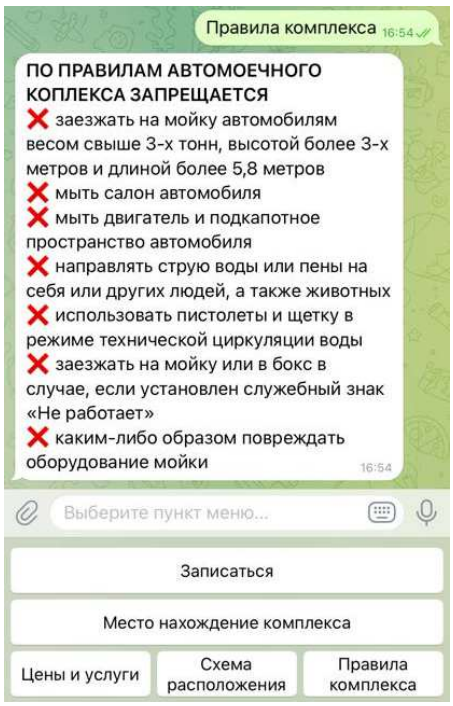

Рисунок 24 – функционал кнопки "Правила комплекса"

Рассмотрим кнопку «Админ панель», эта кнопка появляться появляется только у тех пользователей, данные о которые есть в таблице «Администраторы». Данная кнопка дает возможность к функциям «История», «Статистика» и «Сообщение для всех пользователей» (рисунок 25).

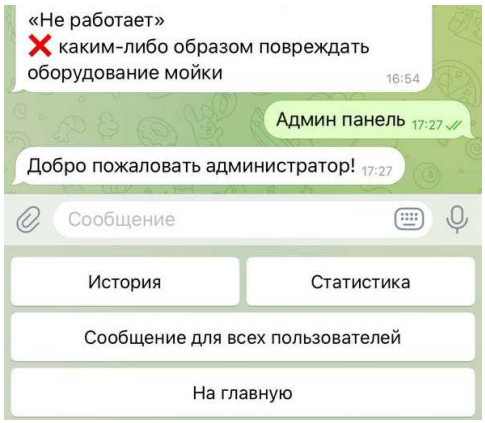

Рисунок 25 – функционал кнопки "Админ панель"

Рассмотрим кнопку «История». Telegram-бот запросит дату в формате «ДД.ММ.ГГГГ» и если в этот день были записи, то бот выведет все записи на этот день, а если записей нет, то выведет информацию о том, что записей нет (рисунок 26).

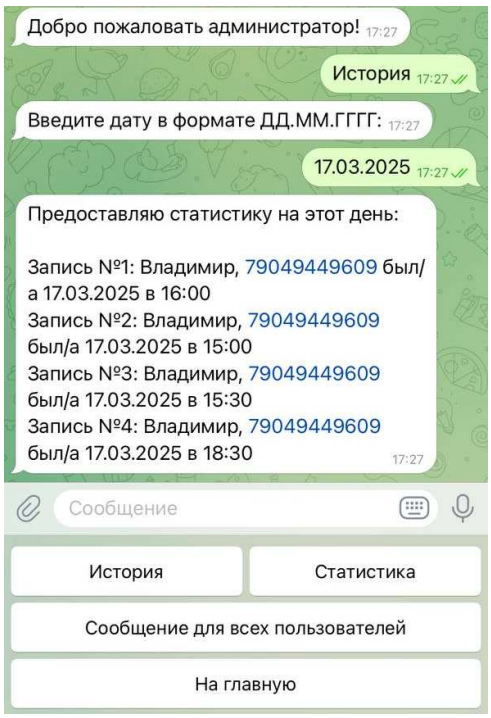

Рисунок 26 – кнопка "История"

Рассмотрим кнопку «Статистика» – она выводит график со статистикой за все время по времени бронирования записи (рисунок 27).

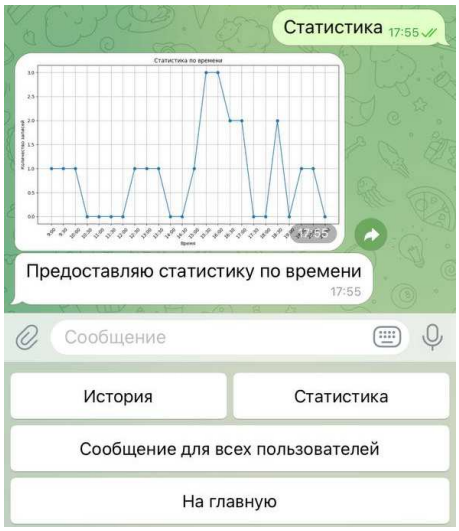

Рисунок 27 – кнопка "Статистика"

Рассмотрим кнопку «Сообщение для всех пользователей» – она позволяет администратору отправить сообщение для всех пользователей (рисунок 28). Проверим, получит ли сообщения другие пользователи Telegram-бота (рисунок 29).

Рисунок 28 – Отправка сообщений для пользователей

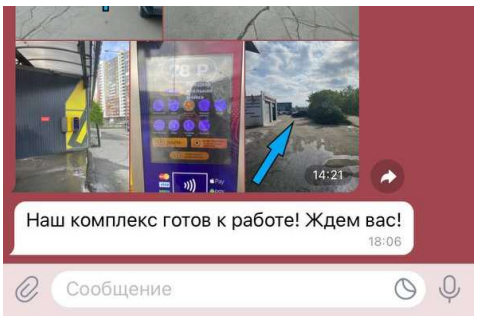

Рисунок 29 – Сообщение от бота у других пользователей

Таблица 3 –Результаты функционального тестирования

Команда/Кнопка	**Ожидаемый результат**	**Фактический результат**
/start	Начало работы, регистрация нового пользователя	+
«Отправить номер»	Отправляет номер пользователя	+
«Запись»	Дает возможность выбрать дату и время	+
Дата и время записи	Создает постой слот для записи пользователя	+
«Моя запись»	Появляется поле записи, есть кнопка удаляющая запись	+
«Место нахождение комплекса»	Отправляет геолокацию автомоечного комплекса	+
«Цены и услуги»	Предоставляет прайс и услуги автомоечного комплекса	+

Продолжение таблицы 3

«Схема расположения»	Отправляет фото с направлением движения в автомоечном комплексе	+
«Правила комплекса»	Выводит правила автомоечного комплекса	+
«Админ панель»	Дает доступ к кнопкам «Статистика», «История» и «Сообщение для всех пользователей»	+
«История»	Выводит историю посещения по дате	+
«Статистика»	Выводит информацию о посещении клиентов	+
«Сообщение для всех пользователей»	Администратор имеет право отправить сообщение для всех пользователей	+

4.2. ВЫВОД ПО РАЗДЕЛУ ЧЕТЫРЕ

Проведено комплексное тестирование функционала бота, подтвердившее работоспособность всех модулей. Основные проверки включали: процесс регистрации с сохранением данных в таблицу "Пользователи", работу клавиатуры и систему бронирования. Тестирование подтвердило, что занятые слоты автоматически исключаются из доступных вариантов, а временное окно ограничено периодом с 9:00 до 20:30.

Кнопки "Местонахождение комплекса", "Цены и услуги" и "Правила комплекса" предоставляя пользователям актуальные данные. Административные функции, включая просмотр статистики посещаемости рассылку сообщений, доступны только авторизованным сотрудникам и функционируют в соответствии с требованиями. Все операции записи и отмены бронирования корректно отражаются в базе данных.

5. ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы была проведена комплексная разработка Telegram-бота для автомоечного комплекса. В первой главе проведен сравнительный анализ существующих решений, который позволил сформулировать ключевые критерии оценки и выделить конкурентные преимущества разрабатываемого решения, и обзор научно-технической литературы.

Анализ архитектурных решений показал, что выбранные технологии: Python, PyCharm и SQLite оптимально соответствует поставленным задачам по производительности, масштабируемости и поддержки. Особое внимание было уделено проектированию модульной файловой структуры, включающей компоненты обработки запросов, управления базой данных и взаимодействия с пользователем.

Реализованный программный продукт обладает такими особенностями, как интуитивный интерфейс бронирования с визуализацией схемы расположения оборудования, система учета и анализа статистики посещений, идентификация клиентов для предотвращения вандализма, а также предоставление клиентом прайс-листом.

Тестирование функциональности подтвердило соответствие системы всем предъявленным требованиям. Основные преимущества разработанного решения это - сокращение времени ожидания за счет системы онлайн-бронирования, упрощение процесса первичного ознакомления с услугами для новых клиентов, возможность аналитики посещаемости для оптимизации работы комплекса, снижение рисков порчи имущества благодаря системе учета клиентов.

Перспективы развития проекта включают интеграцию с системами безналичной оплаты, внедрение push-уведомлений. Модульная архитектура позволяет легко расширять функциональность без необходимости переработки основной структуры приложения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Pros, Cons, and Use Cases of Telegram Chatbots – https://botpenguin.com/blogs/pros-cons-and-use-cases-of-telegram-chatbots. Дата обращения 18.12.2024.
2. Как обеспечить безопасность телеграм-бота. – https://tproger.ru/articles/obespechenie-bezopasnosti-telegram-botov. Дата обращения 18.12.2024.
3. 9 архитектурных антипаттернов при разработке телеграм-ботов на Python. – https://habr.com/ru/companies/otus/articles/763158/. Дата обращения 18.12.2024.
4. Telegram Боты на Aiogram 3.x: Все про FSM простыми словами. – https://habr.com/ru/articles/822061/. Дата обращения 18.12.2024.
5. Как реализовать end-to-end-тестирование telegram-бота. – https://habr.com/ru/companies/beeline_cloud/articles/734164/. Дата обращения 18.12.2024.
6. Языки программирования для Telegram ботов. – https://sky.pro/wiki/javascript/yazyki-programmirovaniya-dlya-Telegram-botov/. Дата обращения 18.12.2024.
7. Всё о языке программирования Python: растущая популярность, плюсы и минусы, сферы применения. – https://practicum.yandex.ru/blog/vsyo-o-yazyke-programmirovaniya-python/#preimuschestva-pythona. Дата обращения 18.12.2024.
8. Особенности, характеристики и области применения Node.js. – https://scand.com/ru/company/blog/node-js-features-uses-and-benefits-of-development/. Дата обращения 18.12.2024.
9. Язык PHP: особенности, актуальность, перспективы. – https://practicum.yandex.ru/blog/yazyk-programmirovaniya-php/#plyusy-i-minusy. Дата обращения 18.12.2024.

10. Python или Go: какой язык выбрать программисту. – https://blog.skillfactory.ru/python-ili-go-chto-vybrat/. Дата обращения 18.12.2024.
11. Как создать бота в Telegram: большая инструкция. – https://blog.skillfactory.ru/kak-sozdat-bota-v-telegram/. Дата обращения 18.12.2024.
12. Реляционная база данных: принцип работы, перспективы использования. – https://gb.ru/blog/relyatsionnaya-baza-dannykh/. Дата обращения 18.12.2024.
13. Нереляционные данные и базы данных NoSQL. – https://learn.microsoft.com/ru-ru/azure/architecture/data-guide/big-data/non-relational-data. Дата обращения 18.12.2024.
14. Введение в объектно-ориентированные базы данных. – https://habr.com/ru/articles/56399/. Дата обращения 18.12.2024.
15. Виды баз данных. Большой обзор типов СУБД. – https://habr.com/ru/companies/amvera/articles/754702/. Дата обращения 18.12.2024.
16. SQLite. – https://blog.skillfactory.ru/glossary/sqlite/. Дата обращения 18.12.2024.
17. Главный инструмент разработчика: что такое IDE, зачем она нужна и как её выбрать. – https://practicum.yandex.ru/blog/integrirovannaya-sreda-razrabotki-ide/. Дата обращения 18.12.2024.
18. PyCharm. – https://blog.skillfactory.ru/glossary/pycharm/. Дата обращения 18.12.2024.
19. Visual Studio Code: установка, настройка, русификация и список горячих клавиш. – https://skillbox.ru/media/code/visual-studio-code-ustanovka-nastroyka-rusifikatsiya-i-spisok-goryachikh-klavish/?ysclid=mayn53xd44135879545. Дата обращения 18.12.2024.

20. Питон, панды и анаконда: что внутри у Jupyter Notebook. – https://practicum.yandex.ru/blog/chto-takoe-jupyter-notebook/. Дата обращения 18.12.2024

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ФАЙЛА ЗАПУСКА TELEGRAM-БОТА

Листинг А.1 – Исходный код файла запуска Telegram-бота

```
import asyncio
from aiogram import Bot, Dispatcher

from app.database.models import async_main
bot = Bot(token='7727935073:AAF7Noz9gX7u_JSwAybUNbuu-G-Z-qTSo1c')

async def main():
    await async_main()

    from app.handlers import router
    dp = Dispatcher()
    dp.include_router(router)
    await dp.start_polling(bot)

if __name__ == '__main__':
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print('Бот выключен')
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ФАЙЛА ОБРАБОТКИ ЗАПРОСОВ TELEGRAM-БОТУ

Листинг Б.1 – Исходный код файла обработки запросов Telegram-боту

```
from aiogram import F, Router
from aiogram.exceptions import TelegramForbiddenError
from aiogram.types import Message, FSInputFile, InputMediaPhoto
from aiogram.filters import CommandStart
from aiogram.fsm.state import State, StatesGroup
from aiogram.fsm.context import FSMContext

import app.keyboards as kb
import app.database.requests as rq
import main

class RegisterStateGroup(StatesGroup):
    name = State()
    number = State()

class RecordStateGroup(StatesGroup):
    date = State()
    time = State()
    service = State()

class AdminStateGroup(StatesGroup):
    message = State()
    history_group = State()

class Form(StatesGroup):
    waiting_for_time = State()
    waiting_for_date = State()

router = Router()

#----Начало и регистрация и запись пользователя в бд---
@router.message(CommandStart())
async def cmd_start(message: Message, state: FSMContext):
    await message.answer('Добро пожаловать! Для начала использования
необходимо зарегистрироваться!\n'
                         'Введите ваше имя')
    await state.set_state(RegisterStateGroup.name)

@router.message(RegisterStateGroup.name)
async def register_name(message: Message, state: FSMContext):
    await state.update_data(name=message.text)
    await state.set_state(RegisterStateGroup.number)
    await message.answer('Введите ваш номер начиная с +7...',
                         input_field_placeholder='Введите ваш номер начиная с
+7...', reply_markup=kb.get_number)

@router.message(F.contact)
async def register_number(message: Message, state: FSMContext):
    await state.update_data(number=message.contact.phone_number)
    data = await state.get_data()
    await message.answer(f'{data["name"]}, Вы успешно зарегистрировались!',
reply_markup= await kb.main_kb(message.from_user.id))
```

Продолжение листинга Б.1

```
    await rq.set_user(tg_id=message.from_user.id, name=data["name"],
phone_number=data["number"])
    await state.clear()


@router.message(RegisterStateGroup.number)
async def register_number(message: Message, state: FSMContext):

    while True:
        if (len(message.text) < 11) or (len(message.text) > 12):
            await message.answer('Повторите ввод номера телефона',
reply_markup=kb.get_number)
            return

        await state.update_data(number=message.text)
        data = await state.get_data()
        await message.answer(f'{data["name"]}, Вы успешно
зарегистрировались!', reply_markup= await kb.main_kb(message.from_user.id))

        await rq.set_user(tg_id=message.from_user.id, name=data["name"],
phone_number=data["number"])
        await state.clear()
        break


#-------------------------Запись на мойку--------------------------
@router.message(F.text == 'Записаться')
async def record(message: Message, state: FSMContext):
    await state.set_state(RecordStateGroup.date)
    await message.answer('Выберете удобную для вас дату', reply_markup=await
kb.create_keyboard_date()) #

@router.message(F.text == 'Назад')
async def back(message: Message, state: FSMContext):
    st = await state.get_state()

    if st == RecordStateGroup.time:
        await state.set_state(RecordStateGroup.date)
        await message.answer('Выберите дату', reply_markup=await
kb.create_keyboard_date())
    elif st == RecordStateGroup.date:
        await message.answer('Вы на главном меню', reply_markup= await
kb.main_kb(message.from_user.id))
        await state.clear()
    elif st == Form.waiting_for_time or st == Form.waiting_for_date:
        await message.answer('Вы вернулись назад', reply_markup=kb.my_record)
        await state.clear()


@router.message(RecordStateGroup.date)
async def record_date(message: Message, state: FSMContext):
    await state.update_data(date=message.text)
    await state.set_state(RecordStateGroup.time)
    await message.answer('Выберете удобное время', reply_markup= await
kb.create_keyboard_time(date=message.text))

@router.message(RecordStateGroup.time)
async def record_time(message: Message, state: FSMContext):
    await state.update_data(time=message.text)
    data = await state.get_data()
```

Продолжение листинга Б.1

```
    await rq.set_record(date=data["date"], time=data["time"],
tg_id=message.from_user.id)

    await message.answer(f'Вы зарегистрировались на {data["date"]} в
{data["time"]}\n',
                         reply_markup=await kb.main_kb(message.from_user.id))
    await state.clear()

#-------------------------Admin panel-------------------------
@router.message(F.text == 'Admin panel')
async def admin(message: Message):
    await message.answer('Добро пожаловать администратор!',
reply_markup=kb.ad_kb)

@router.message(F.text == 'History')
async def history(message: Message, state: FSMContext):
    await message.answer('Введите дату в формате ДД.ММ.ГГГГ: ',
reply_markup=kb.cancellation)
    await state.set_state(AdminStateGroup.history_group)
    # await message.answer(f'Предоставляю за {message.text}:\n{await
rq.history_washing(message.text)}')

@router.message(AdminStateGroup.history_group)
async def history_g(message: Message, state: FSMContext):

    if message.text == 'Отмена':
        await message.answer('Предоставление истории отменено!',
reply_markup=kb.ad_kb)
        await state.clear()
        return

    await message.answer(await rq.history_washing(message.text),
reply_markup=kb.ad_kb)
    await state.clear()

@router.message(F.text == 'Статистика')
async def static(message: Message):
    await message.answer(f'Предоставляю статистику по времени', await
rq.statistics(message.from_user.id))

@router.message(F.text == 'Сообщение для всех пользователей')
async def history(message: Message, state: FSMContext):
    await message.answer('Введите сообщение, которое вы хотите отправить всем
пользователям:', reply_markup=kb.cancellation)
    await state.set_state(AdminStateGroup.message)

@router.message(AdminStateGroup.message)
async def send_message_to_all(message: Message, state: FSMContext):
    mes = message.text

    if mes == 'Отмена':
        await message.answer('Отправка сообщения отменена.',
reply_markup=kb.ad_kb)
        await state.clear()
        return

    users = await rq.send()
    users.remove(message.from_user.id)

    for user in users:
        try:
```

Продолжение листинга Б.1

```
            await main.bot.send_message(user, mes)
        except TelegramForbiddenError:
            print(f"Бот был заблокирован пользователем с ID {user}.")
    await message.answer('Сообщение отправлено!', reply_markup=kb.ad_kb)
    await state.clear()

#-------------------------Запись пользователя-------------------------
@router.message(F.text == 'Моя запись')
async def record(message: Message):
    r = await rq.my_record(message.from_user.id)
    await message.answer(f'{r.name}, вы записаны на {r.date} в {r.time}',
reply_markup=kb.my_record)

@router.message(F.text == 'Удалить запись')
async def delete(message: Message):
    if await rq.delete_my_record(await rq.my_record(message.from_user.id)):
        await message.answer('Ваша запись успешна удалена! Запишитесь на
другое время!',
                             reply_markup=await
kb.main_kb(message.from_user.id))
    else:
        await message.answer('С удалением вашей записи произошла ошибка:(')

#-------------------------О автомойке-------------------------
@router.message(F.text == 'Место нахождение комплекса')
async def way_to_car_wash(message: Message):
    latitude = 55.16331569652977
    longitude = 60.16279950569519

    await message.answer_location(latitude=latitude, longitude=longitude)

@router.message(F.text == 'Цены и услуги')
async def prices_services(message: Message):
    await message.answer(
                    '    🌟*ОСНОВНЫЕ УСЛУГИ*🌟\n'
                         'Ополаскивание - 20 руб/с\n'
                         'Пена - 40 руб/с\n'
                         '🌟*ДОПОЛНИТЕЛЬНЫЕ УСЛУГИ*🌟\n'
                         'Сушка - 30 руб/с\n'
                         'Пылесос - 20 руб/с', parse_mode='Markdown',
reply_markup= await kb.main_kb(message.from_user.id))

@router.message(F.text == 'Правила комплекса')
async def rules(message: Message):
    await message.answer('*ПО ПРАВИЛАМ АВТОМОЕЧНОГО КОПЛЕКСА ЗАПРЕЩАЕТСЯ* '
                         '\n❌ заезжать на мойку автомобилям весом свыше 3-х
тонн, высотой более 3-х метров и длиной более 5,8 метров '
                         '\n❌ мыть салон автомобиля '
                         '\n❌ мыть двигатель и подкапотное пространство
автомобиля '
                         '\n❌ направлять струю воды или пены на себя или
других людей, а также животных '
                         '\n❌ использовать пистолеты и щетку в режиме
технической циркуляции воды '
                         '\n❌ заезжать на мойку или в бокс в случае, если
установлен служебный знак «Не работает» '
                         '\n❌ каким-либо образом повреждать оборудование
мойки',
                         parse_mode='Markdown', reply_markup= await
```

Окончание листинга Б.1

```
kb.main_kb(message.from_user.id))

@router.message(F.text == 'Схема расположения')
async def rules(message: Message):
    await message.answer('Для того то бы заехать в бокс, смотрите 1 и 2 фото.
\n'
                         'Для оплаты используйте автомат, представленный на 3
и 4 фото. \n'
                         'Для выезда с бокса смотрите на фото 5.',
                         reply_markup= await
kb.main_kb(message.from_user.id))

    file_paths = ["1 фото.jpg", "2 фото.jpg", "3 фото.jpg", "4 фото.jpg", "5
фото.jpg"]

    media_group = []
    for file_path in file_paths:
        media_group.append(InputMediaPhoto(media=FSInputFile(file_path)))

    await main.bot.send_media_group(chat_id=message.from_user.id,
media=media_group)

#-------------------------Действия-------------------------
@router.message(F.text == 'На главную')
async def at_home(message: Message):
    await message.answer('Вы на главном меню!', reply_markup= await
kb.main_kb(message.from_user.id))
```

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД ФАЙЛА ДЛЯ СОЗДАНИЯ КНОПОК В TELEGRAM-БОТЕ

Листинг В.1 – Исходный код файла для создания кнопок в Telegram-боте

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton,
InlineKeyboardButton, InlineKeyboardMarkup
from datetime import datetime, timedelta

import app.database.requests as rq

async def main_kb(tg_ig):
    kb_list = [[KeyboardButton(text='Записаться')],
               [KeyboardButton(text='Место нахождение комплекса')],
               [KeyboardButton(text='Цены и услуги'),
               KeyboardButton(text='Схема расположения'),
                KeyboardButton(text='Правила комплекса')]]

    if await rq.search_admins(tg_ig):
        kb_list.append([KeyboardButton(text='Админ панель')])

    if await rq.my_record(tg_ig):
        kb_list.append([KeyboardButton(text='Моя запись')])


    keyboard = ReplyKeyboardMarkup(keyboard=kb_list, resize_keyboard=True,
input_field_placeholder='Выберите пункт меню...')
    return keyboard

ad_kb = ReplyKeyboardMarkup(keyboard=[[KeyboardButton(text='История'),
KeyboardButton(text='Статистика')],
                                      [KeyboardButton(text='Сообщение для
всех пользователей')],
                                      [KeyboardButton(text='На главную')]],
resize_keyboard=True)

get_number = ReplyKeyboardMarkup(keyboard=[[KeyboardButton(text='Отправить
номер', request_contact=True)]],
                                 resize_keyboard=True,
input_field_placeholder='Напишите ваш номер телефона +7...')

my_record = ReplyKeyboardMarkup(keyboard=[[KeyboardButton(text='Удалить
запись')],
                                          [KeyboardButton(text='На
главную')]],
                                resize_keyboard=True)

cancellation =
ReplyKeyboardMarkup(keyboard=[[KeyboardButton(text='Отмена')]],
resize_keyboard=True)

async def create_keyboard_date():
    today = datetime.now()
    hour_start = 9
    hour_end = 21

    if int(today.strftime('%H')) >= hour_end:
```

Продолжение листинга В.1

```
        today += timedelta(1)
    busy_times = await rq.del_date_time(today.strftime('%d.%m.%Y'))

    if len(busy_times) == (hour_end - hour_start) * 2:

        today += timedelta(1)

    buttons = []
    for i in range(2):
        new = []
        for j in range(3):
            if i == 0:
                date = (today + timedelta(days=j)).strftime('%d.%m.%Y')
                new.append(KeyboardButton(text=date))
            else:
                date = (today + timedelta(days=j+3)).strftime('%d.%m.%Y')
                new.append(KeyboardButton(text=date))

        buttons.append(new)

    buttons.append([KeyboardButton(text='Назад')])

    keyboard = ReplyKeyboardMarkup(keyboard=buttons, resize_keyboard=True)
    return keyboard


async def create_keyboard_time(date):
    busy_times = await rq.del_date_time(date)
    buttons = []

    hour_start = 9
    hour_end = 21

    today = datetime.now()
    user_today = today.strftime('%d.%m.%Y')
    hour_today = int(today.strftime('%H'))
    min_today = int(today.strftime('%M'))

    a = 0

    if hour_today > hour_start and user_today == date:
        hour_start = hour_today
        if min_today < 30:
            a = 1
        elif min_today > 30:
            hour_start += 1

    if hour_today > hour_end:
        today += timedelta(1)
        user_today = today.strftime('%d.%m.%Y')

    for i in range(4):
        if hour_start >= hour_end:
            break
        else:
            new = []
            for j in range(3):

                if hour_start >= hour_end:
                    break
```

Окончание листинга В.1

```
                if user_today == date and a == 1:
                    if f'{hour_start}:30' not in busy_times:
                        new.append(KeyboardButton(text=f'{hour_start}:30'))
                    a = 0
                    hour_start += 1

                    if hour_start >= hour_end:
                        break

                if f'{hour_start}:00' not in busy_times:
                    new.append(KeyboardButton(text=f'{hour_start}:00'))

                if f'{hour_start}:30' not in busy_times:
                    new.append(KeyboardButton(text=f'{hour_start}:30'))
                hour_start += 1


            buttons.append(new)

    buttons.append([KeyboardButton(text='Назад')])
    keyboard = ReplyKeyboardMarkup(keyboard=buttons, resize_keyboard=True)
    return keyboard
```

ПРИЛОЖЕНИЕ Г
ИСХОДНЫЙ КОД ФАЙЛА РАЗРАБОТКИ МОДЕЛЕЙ ДЛЯ БАЗЫ ДАННЫХ

Листинг Г.1 – Исходный код файла разработки моделей для базы данных

```
# -------------Структуры базы данных-------------
from sqlalchemy import BigInteger, String
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
from sqlalchemy.ext.asyncio import AsyncAttrs, async_sessionmaker,
create_async_engine

engine = create_async_engine(url='sqlite+aiosqlite:///db.sqlite3') # создаем
бд
async_session = async_sessionmaker(engine) # подключаем бд

class Base(AsyncAttrs, DeclarativeBase):
    pass

class User(Base):
    __tablename__ = 'users'

    id: Mapped[int] = mapped_column(primary_key=True)
    tg_id = mapped_column(BigInteger)
    name: Mapped[str] = mapped_column(String(25))
    phone_number: Mapped[str] = mapped_column(String(15))

class Record(Base):
    __tablename__ = 'records'

    id: Mapped[int] = mapped_column(primary_key=True)
    tg_id = mapped_column(BigInteger)
    name: Mapped[str] = mapped_column(String(25))
    phone_number: Mapped[str] = mapped_column(String(15))
    date: Mapped[str] = mapped_column(String(10))
    time: Mapped[str] = mapped_column(String(5))

class Administrator(Base):
    __tablename__ = 'administrators'
    id: Mapped[int] = mapped_column(primary_key=True)
    tg_id = mapped_column(BigInteger)

async def async_main():
    async with engine.begin() as conn:
        await conn.run_sync(Base.metadata.create_all)
```

ПРИЛОЖЕНИЕ Д
ИСХОДНЫЙ КОД ФАЙЛА ЗАПРОСОВ В БАЗУ ДАННЫХ

Листинг Д.1 – Исходный код файла запросов в базу данных

```
import matplotlib.pyplot as plt
import main
import os

from app.database.models import async_session, Record, User, Administrator
from sqlalchemy import select, BigInteger, String, desc
from datetime import datetime
from aiogram.types import FSInputFile


class Records:
    tg_id = BigInteger
    name = String


async def set_user(tg_id, name, phone_number):  #
    async with async_session() as session:
        user = await session.scalar(select(User).where(User.tg_id == tg_id))

        if not user:
            session.add(User(tg_id=tg_id, name=name,
phone_number=phone_number))  #
            await session.commit()


async def set_record(date, time, tg_id):
    async with async_session() as session:

        u = await session.scalar(select(User).where(User.tg_id == tg_id))
        session.add(Record(date=date, time=time, tg_id=tg_id, name=u.name,
                           phone_number=u.phone_number))
        await session.commit()


async def del_date_time(date):
    async with async_session() as session:
        result = await session.execute(select(Record.time).where(Record.date
== date))
        times = result.scalars().all()

    await session.commit()
    return times


async def search_admins(tg_id):
    async with async_session() as session:
        admin = await
session.scalar(select(Administrator).where(Administrator.tg_id == tg_id))

        if not admin:
            return False
        else:
            return True
```

Продолжение листинга Д.1

```
async def statistics(user_id):
    time_dict = {
         '9:00': 0,  '9:30': 0, '10:00': 0, '10:30': 0, '11:00': 0, '11:30':
0,
        '12:00': 0, '12:30': 0, '13:00': 0, '13:30': 0, '14:00': 0, '14:30':
0,
        '15:00': 0, '15:30': 0, '16:00': 0, '16:30': 0, '17:00': 0, '17:30':
0,
        '18:00': 0, '18:30': 0, '19:00': 0, '19:30': 0, '20:00': 0, '20:30':
0
    }
    async with async_session() as session:
        times = (await session.scalars(select(Record.time))).all()

        for time in times:
            if time in time_dict:
                time_dict[time] += 1

        await plot_statistics(user_id, time_dict)


async def plot_statistics(user_id, time_dict):
    times = list(time_dict.keys())
    counts = list(time_dict.values())

    plt.figure(figsize=(10, 6))
    plt.plot(times, counts, marker='o')
    plt.title('Статистика по времени')
    plt.xlabel('Время')
    plt.ylabel('Количество записей')
    plt.xticks(rotation=45)
    plt.grid()
    plt.tight_layout()

    file_path = "test.png"
    plt.savefig(file_path)
    plt.close()

    photo = FSInputFile(file_path)

    await main.bot.send_photo(user_id, photo)

    os.remove(file_path)


async def history_washing(date):
    async with async_session() as session:

        history = (await session.scalars(select(Record).where(Record.date ==
date))).all()

        n = 0
        history_messages = ['Предоставляю статистику на этот день:\n']

        if len(history) == 0:
            return 'На этот день записей не было'

        for record in reversed(history):
            n += 1
            history_messages.append(
                f"Запись №{n}: {record.name}, {record.phone_number} был/а
```

Окончание листинга Д.1

```
{record.date} в {record.time}")

        messages = "\n".join(history_messages)

        return messages


async def my_record(tg_id):
    async with async_session() as session:
        full_record = await
session.execute(select(Record).filter(Record.tg_id ==
tg_id).order_by(desc(Record.id)))
        record = full_record.scalars().first()

        if not record:
            return False

        record_datetime = datetime.strptime(f"{record.date} {record.time}",
"%d.%m.%Y %H:%M")
        now = datetime.now()

        if record_datetime < now:
            return False

        return record


async def delete_my_record(record: Record):
    async with async_session() as session:
        if record:
            await session.delete(record)
            await session.commit()
            pass
            return True
        else:
            pass
            return False


async def send():
    async with async_session() as session:
        users_tg_id = (await session.scalars(select(User.tg_id))).all()
    pass
    return users_tg_id


async def bd_change_time(recoder: Record, time):
    async with async_session() as session:
        session.delete(recoder)
        recoder.time = time
        session.add(recoder)
        await session.commit()

async def bd_change_date(recoder: Record, date):
    async with async_session() as session:
        session.delete(recoder)
        recoder.date = date
        session.add(recoder)
        await session.commit()
```