

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

КОНСТРУКТОР ЧАТ-БОТОВ

Студент

Дор - Дорофеев Алексей Алексеевич
(подпись, дата) 23.06.23 (ФИО полностью)

Руководитель

Карамышева Н.С.
(подпись, дата) 23.06.23 (фамилия, инициалы)

Нормоконтролёр

Бычков А.С.
(подпись, дата) 23.06.23 (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 02.06.2023 г. № 14)

Заведующий кафедрой

М.А. Митрохин
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от

27.06.2023г. № 27)

Секретарь ГЭК

Е

Е.И. Гурин
(подпись)

(фамилия, инициалы)

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«26» апреля 2023 г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Дорофеев Алексей Алексеевич гр. 19ВВП1 факультета ФВТ

2. Тема работы Конструктор чат-ботов

Тема утверждена приказом ПГУ № 458/о от "26" апреля 2023 г.

3. Руководитель бакалаврской работы Карамышева Н.С.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать платформу, обладающую функционалом конструктора чат-ботов. Язык программирования Python, среда разработки PyCharm. Программа должна обладать интуитивно понятным интерфейсом, порождать уникальных чат-ботов в сети Telegram и использовать следующие технологии

1. TelegramAPI

2. python-telegram-bot

3. Асинхронное программирование

4. Сетевое программирование

5. PyQt6

5. Перечень подлежащих разработке вопросов

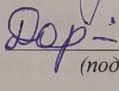
Требуется произвести анализ предметных областей: изучить существующие информационные системы, выделить их достоинства и недостатки. Выбрать язык программирования, среду разработки, СУБД, которые подойдут для создания конструктора чат-ботов. Определить архитектуру приложения и вспомогательные средства разработки. Изучить функционал библиотек для разработки чат-ботов и графических приложений.

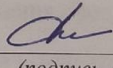
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	26.04.2023 – 28.05.2023	
2	Анализ требований	32	29.04.2023 – 30.05.2023	
3	Создание макета графического интерфейса	30	31.04.2023 – 3.05.2023	
4	Выбор технологий, применяемых при проектировании и разработке	40	4.05.2023 – 8.05.2023	
5	Реализация графического интерфейса	66	9.05.2023 – 15.05.2023	
6	Разработка основной части программы	70	16.05.2023 – 23.05.2023	
7	Оформление пояснительной записки	54	24.05.2023 – 31.05.2023	

Дата выдачи задания "26" апреля 2023 г.

Руководитель бакалаврской работы  Карамышева Н.С.
(подпись, дата)

Задание к исполнению принял студент  Дорофеев А.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)
26.06.2023

Реферат

Бакалаврская работа содержит 150 листов, 34 рисунка, 8 таблиц, 8 источников, 3 приложения

ПРИЛОЖЕНИЕ, ОБРАБОТКА СОСТОЯНИЙ, СОЗДАНИЕ ЧАТ-БОТОВ, ТЕСТИРОВАНИЕ, БАЗА ДАННЫХ, PYTHON-TELEGRAM-BOT, PSYCOPG2, PYQT6, THREADING, PYTHON, PYCHARM

Цель работы – разработать инструмент, позволяющий создавать чат-ботов для социальной сети *Telegram* без применения навыков программирования.

Одним из путей достижения поставленной задачи является программа с графическим интерфейсом, обеспечивающая выполнение следующих функций: регистрация и авторизация пользователей, порождение исходных файлов чат-ботов, редактирование функционала чат-ботов, путем добавления текстовых команд, профилей *Inline*-клавиатур, формирования задаваемых чат-ботом вопросов и ответов чат-бота в зависимости от комбинации полученных от пользователя ответов на вопросы, возможность отредактировать занесенные данные, управление чат-ботом, запуск и остановка его работы, удаление чат-бота из личного кабинета.

В конечном итоге был разработан инструмент на основе приложения с графическим интерфейсом, позволяющий пользователям создавать уникальных чат-ботов для сети *Telegram*, который полностью соответствует цели выпускной квалификационной работы. В достижении результата были использованы следующие технологии: *Python*, *python-telegram-bot*, *psycopg2*, *PyQt6*, *PostgreSQL*, *Telegram API*.

					ПГУ1.090301.07.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Дорофеев А.А.			Конструктор чат-ботов Пояснительная записка	Лит.	Лист	Листов
Провер.		Карамышева Н.С.					2	50
Н. Контр		Бычков А.С.				ФВТ, каф. ВТ		
Утверд.		Митрохин М.А.				гр. 19ВВП1		

Содержание

Перечень сокращений и обозначений.....	4
Введение.....	5
1 Конструктор чат-ботов.....	7
1.1 Анализ предметной области.....	7
1.2 Решение поставленной задачи	8
1.3 Анализ системных требований проекта.....	10
1.4 Анализ аналогов	12
1.5 Анализ и структура конструктора чат-ботов.....	21
1.6 Выбор инструментов разработки.....	23
1.6.1 Выбор мессенджера	23
1.6.2 Выбор языка программирования.....	24
1.6.3 Выбор СУБД.....	27
1.6.4 Выбранные технологии	29
1.7 Реализация.....	30
1.7.1 Регистрация чат-бота	30
1.7.2 Работа с БД	33
1.7.3 Описание доступных функций	39
1.7.4 Описание программы.....	52
1.7.5 Тестирование работы системы	66
2 Безопасность жизнедеятельности	69
2.1 Правила безопасности при использовании ПК	69
2.2 Правила безопасности при использовании смартфона	71
Заключение	74
Список использованных источников	76
Приложение А — UML-диаграммы	77
Приложение Б — Листинг программы	82
Приложение В — Презентация.....	144

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД — база данных, структурированная коллекция данных, организованных и хранимых в электронном виде

ОС — операционная система, программное обеспечение, выполняющее управление ресурсами устройства и обеспечивающее интерфейс между пользователем и устройством

ПК — персональный компьютер, электронное устройство, способное выполнять различные операции и задачи по обработке информации и хранению данных

ПО — программное обеспечение, совокупность программ, процессов и данных, которые используются устройстве для выполнения различных задач

СУБД — система управления базами данных, программное обеспечение, предназначенное для создания, управления и обработки баз данных

ЯП — язык программирования, формальный набор правил, позволяющих программисту создавать компьютерные программы

API (Application Programming Interface) — описание способов, при помощи которых одна компьютерная программа может взаимодействовать с другой программой

IDE (Integrated development environment) — интегрированная среда разработки

SQL (Structured Query Language) — язык структурированных запросов

UML (Unified Modeling Language) — унифицированный язык моделирования

Введение

В наше время чат-боты становятся все более популярными, поскольку являются эффективным способом автоматизации и оптимизации коммуникации между пользователями и компаниями. Боты предоставляют пользователю возможность взаимодействовать с брендом и получать ответы на вопросы или помощь в решении проблем. Использование чат-ботов для компаний может сократить расходы времени и денежных ресурсов на обслуживание клиентов.

Чат-боты также могут служить отличным решением для различных сфер общественной жизни, таких как медицина, образование и культура. Активное появление чат-ботов может улучшить качество жизни людей. Чат-боты могут быть использованы в любом месте и в любое время и помочь людям решать различные проблемы в кратчайшее время.

Telegram — один из самых популярных мессенджеров в мире, который предоставляет возможность создания чат-ботов для различных целей. Создание чат-ботов компаниями самостоятельно может сократить расходы на приобретение разработки чат-бота другими ИТ-компаниями. Однако создание чат-бота — непростая задача для людей, которые не обладают навыками программирования.

Для решения этой проблемы можно использовать конструктор чат-ботов. Конструктор чат-ботов — это инструмент, который позволяет пользователям создавать собственных чат-ботов без необходимости программировать их с нуля. Вместо этого пользователи могут использовать готовые шаблоны и настройки, чтобы быстро создать и настроить своих ботов.

Создание конструктора чат-ботов — интересная и актуальная тема, которая потенциально может заинтересовать многих разработчиков и предпринимателей.

Цели выпускной квалификационной работы:

- анализ актуальных средств программной разработки;
- изучение нюансов разработки чат-диалоговых систем на базе мессенджеров;

– реализация инструмента, способного создавать уникальных чат-ботов для ведения диалога с пользователями.

Для достижения поставленных целей, необходимо обозначить следующие задачи, решаемые в ходе выполнения выпускной квалификационной работы:

– провести анализ протекания действующих процессов в бизнесе и других общественных сферах;

– провести анализ существующих конструкторов чат-ботов и их квалификации;

– провести анализ архитектуры конструктора;

– реализовать личный кабинет пользователей конструктора;

– реализовать инструмент, позволяющий редактировать функционал чат-ботов без написания программного кода;

– реализовать возможность запуска чат-ботов из рабочей среды конструктора;

– протестировать конструктор чат-ботов.

1 Конструктор чат-ботов

1.1 Анализ предметной области

Telegram-бот — это программная сущность, которая может выполнять различные функции внутри мессенджера *Telegram*. Она может отвечать на сообщения от пользователей, отправлять сообщения, извлекать данные из других сервисов, и многое другое [1].

Пользователи могут взаимодействовать с ботами, задавая им вопросы или отправляя им команды, на что бот отвечает автоматически или выполняет запрошенные команды или задачи.

Telegram-боты используют множество *API* и могут быть созданы для различных целей, таких как рассылка новостей, управление бизнес-процессами, игры, сбор данных, и многие другие. Создание конструктора чат-ботов в *Telegram* является актуальной темой в сфере *digital*-технологий. Данная социальная сеть в последнее время стала особенно популярной у бизнеса и организаций, которые ищут новые способы взаимодействия с клиентами.

Конструкторы чат-ботов на базе *Telegram* дают возможность создавать и настраивать ботов без необходимости точных знаний программирования. Они позволяют пользователям создавать и настраивать чат-ботов с помощью графического интерфейса, что делает процесс более доступным и простым.

Появление подобных конструкторов помогает бизнесу упростить и автоматизировать коммуникацию с клиентами, предоставляя совершенно новые возможности. За счет применения машинного обучения, интеграции с различными сервисами и системами внутри компаний, чат-боты становятся все более "умными" и полезными для конечных пользователей.

Таким образом, создание конструктора чат-ботов в *Telegram* — это тема, которая находится в центре внимания многих бизнесов и организаций, которые хотят улучшить качество обслуживания клиентов, а также автоматизировать многие повторяющиеся действия с помощью чат-ботов.

					ПГУ1.090301.07.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

1.2 Решение поставленной задачи

Telegram-бот представляет из себя уникальную программную сущность, которая была создана для мессенджера *Telegram* и полностью интегрирована с его функционалом чата. Реализация ботов позволяет осуществить широкий спектр функций: от управления информацией до размещения заказов, а также проведения онлайн-оплат. Современные технологии ускоряют и упрощают процессы управления бизнесом [2]. Сегодня *Telegram*-боты успешно применяются в бизнесе различных отраслей, таких как финансовый сектор, общественное питание, медицина, авиация, онлайн-торговля и многие другие. Они играют важную роль в автоматизации клиентского сервиса и упрощении внутренних бизнес-процессов, усиливая эффективность множества компаний. Чат-боты могут быть настроены для рассылки рекламы его пользователям, что также может повысить доход компаний.

Пользователи конструкторов чат-ботов — это люди, которые создают ботов для своих нужд или для продвижения бизнеса. Пользователи конструктора могут быть отличными представителями различных сфер деятельности: от маркетологов и менеджеров по продажам до работников образовательных учреждений и блогеров. Конструктор чат-ботов предоставляют им возможность создавать ботов для повышения продаж, улучшения обслуживания клиентов, автоматизации процессов и упрощения взаимодействия с аудиторией.

Специфическими вопросами, с которыми сталкиваются пользователи конструкторов чат-ботов, могут являться технические аспекты создания и настройки бота, вопросы безопасности и защиты данных, а также выбор правильной сферы использования бота для достижения максимальной эффективности.

Благодаря возможностям конструкторов чат-ботов, пользователи могут создавать ботов без знания программирования, использовать готовые шаблоны, а также получить доступ к аналитике, которая поможет им улучшить пользовательский опыт и увеличить эффективность бота.

					ПГУ1.090301.07.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

Telegram-боты стали важным элементом в бизнес-стратегии предпринимателей, потому что они автоматизируют различные функции поддержки клиентов и улучшают коммуникацию между клиентами и фирмой. Перечислим главные преимущества создания *Telegram*-ботов для бизнеса.

Боты могут обеспечить круглосуточную поддержку клиентов, отвечают на часто задаваемые вопросы, обрабатывают заказы и запросы в режиме реального времени, что не только экономит время, но и улучшает качество обслуживания клиентов.

Чат-боты могут улучшить привлекательность бренда, помогают продвигать бренд через интерактивные опросы, конкурсы и розыгрыши.

Боты повышают рентабельность бизнеса, они способны автоматически обрабатывать заказы, принимать платежи и управлять состоянием склада, что способствует снижению затрат и повышению эффективности.

В целом, создание *Telegram*-бота является полезным и актуальным инструментом для предпринимателей, которые стремятся повысить качество обслуживания клиентов и улучшить свое бизнес-предложение.

Чат-боты также стали незаменимым инструментом для автоматизации и оптимизации процессов в общественных сферах жизни. Некоторые из применений чат-ботов включают:

– медицина. Чат-боты могут помочь пациентам записаться на прием к врачу, получить ответы на часто задаваемые вопросы по необходимым лекарствам и лечению, а также предложить консультацию по программам здорового образа жизни. Использование чат-ботов в данной сфере не заменяет полноценного специалиста, но вполне способно улучшить работу медицинских учреждений, а также обезопасить пациентов от контакта с людьми в поликлиниках;

– транспорт. Чат-боты могут уведомлять о расписании общественного транспорта и его текущем передвижении по городу, помогать пассажирам

добраться до нужной остановки, а также отслеживать пробки, наличие дорожных работ и задержек транспорта.

– государственная и муниципальная служба. Чат-боты могут помочь населению получать информацию о предоставляемых услугах, сроках их оказания, а также отвечать на юридические вопросы, например, информацию об обновлении паспорта в многофункциональных центрах;

– образование. Чат-боты могут помочь студентам получить информацию о предметах, преподавателях и расписании занятий, а также создавать удобные условия для обмена информацией между учителями и учащимися;

– культура. Чат-боты могут помочь посетителям музеев и культурных объектов получить информацию о достопримечательностях, экспонатах и их истории, а также заказать билеты на спектакли, концерты и другие мероприятия;

– торговля и рестораны. Чат-боты могут помочь клиентам сделать заказ и ознакомиться с меню в заведении, а также получить ответы на любые вопросы касательно качества и состава продуктов. Чат-боты могут являться платформой для реализации онлайн заказа продукции на дом.

1.3 Анализ системных требований проекта

Системные требования проекта определяют минимальные и рекомендуемые требования к технической конфигурации ПО и оборудования, для того чтобы проект мог работать корректно и эффективно. Можно сказать, что системные требования проекта составляют комплекс логически связанных параметров, которые определяют возможности среды исполнения кода и качество пользователя системы.

Это может включать в себя такие элементы, как ОС, версия браузера, объём оперативной памяти, жесткий диск, скорость интернет-соединения, процессор и другие аппаратные или программные компоненты необходимые для оптимальной работы проекта.

Системные требования могут различаться в зависимости от типа проекта, его цели и особенностей. Например, программа для компьютерных игр может требовать более высоких системных требований, чем веб-приложение для работы с электронной почтой.

Определение системных требований проекта очень важно для того, чтобы убедиться, что проект будет работать корректно на устройствах конечных пользователей и не вызовет проблем со совместимостью или продуктивностью. Все системные требования были учтены на этапе проектирования и разработки проекта.

Список минимальных системных требований для использования *Telegram* на ПК или мобильном устройстве в качестве клиента:

- для использования *Telegram* на ПК необходимо наличие ОС *Windows 7* и выше, *macOS X 10.10* и выше, *Linux Ubuntu 12.04* и выше или других совместимых ОС;

- для мобильных устройств на базе *Android* необходима версия ОС 4.1 и выше. Для устройств на базе ОС *iOS* потребуется: *iOS 9.0* и выше;

- для использования *Telegram* требуется наличие интернет-соединения;

- для просмотра и отправки медиа-файлов также необходима достаточная скорость интернет-соединения;

- для обеспечения безопасности и конфиденциальности сообщений, необходимо использовать актуальную версию *Telegram*, обновить которую возможно прямо в приложении;

- внутренняя память устройства должна иметь достаточное место для хранения приложения *Telegram* и ее данных, таких как фотографии, видео, документы и т.д;

- для более комфортного использования *Telegram* на ПК, рекомендуется наличие клавиатуры и мыши, а для персонального пользования на мобильных устройствах рекомендуется наличие сенсорного экрана и поддержка одновременного определения двух или более точек касания.

Также рассмотрим серверные требования. Для запуска конструктора чат-ботов, разработанного на *Python* в среде *PyCharm*, потребуются следующие серверные требования:

- ОС: возможно запуск на разных ОС, но чаще всего используют *Linux* (*Ubuntu*, *CentOS* и т.д.), *Windows Server* (*Windows Server 2012 R2* или более поздняя версия) или *macOS*;

- процессор: рекомендуется использовать мощный многоядерный процессор (например, *Intel Core i7* или более высокой версии), чтобы обеспечить высокую производительность и ускорить время обработки запросов;

- оперативная память: рекомендуемая конфигурация — не менее 8 ГБ;

- жесткий диск: обычно достаточно 100 ГБ свободного места на жестком диске для хранения проектов, кода и зависимостей;

- веб-сервер: конструктор чат-ботов может быть развернут на любом веб-сервере, таком как *Apache*, *Nginx* или *IIS*;

- *Python*: необходимо установить *Python* на сервере (*Python 3.x* рекомендован). Кроме того, на устройстве должны быть установлены все нужные библиотеки, с которыми работает конструктор, такие как *python-telegram-bot v. 13.14*, *PyQt6* и т.д.;

- *PyCharm*: среда разработки *PyCharm* не обязательна для запуска проекта, рекомендуется использовать ее при разработке и отладке;

- доступ к интернету: для обновления и установки зависимостей, а также для взаимодействия с пользователем через сеть, сервер должен иметь доступ к высокоскоростному интернет-соединению.

1.4 Анализ аналогов

Существует множество конструкторов чат-ботов для *Telegram*, рассмотрим наиболее популярные и часто используемые из них.

Dialogflow — это один из наиболее популярных конструкторов чат-ботов среди разработчиков. Он доступен для бесплатного использования и

предоставляет набор инструментов машинного обучения, которые могут использоваться для создания интеллектуальных ботов.

При анализе работы платформы можно отметить следующие особенности:

- интеграция с другими платформами. *Dialogflow* предоставляет широкий спектр интеграций, которые позволяют использовать его для создания ботов для *Facebook Messenger*, *Skype*, *Telegram* и других популярных платформ;

- обучение бота. *Dialogflow* использует инструменты машинного обучения, которые можно использовать для улучшения работы бота. Эти инструменты могут быть настроены на основе набора данных, что позволяет боту становиться все более интеллектуальным;

- аналитика. Конструктор предоставляет информацию о том, как пользователи взаимодействуют с ботом. Она может использоваться для улучшения работы бота и оптимизации его для улучшения пользовательского опыта;

- безопасность. Платформа обеспечивает высокий уровень безопасности и конфиденциальности данных пользователей.

Недостатком данной платформы является сложность в освоении работы конструктора, создание бота может занять больше времени и требуются знания настройки его платформы и умение работать с инструментами машинного обучения. Обычным пользователям данный конструктор может быть неудобен в использовании.

В целом, *Dialogflow* является мощной платформой, которая предоставляет набор инструментов, необходимых для разработки интеллектуальных чат-ботов. Его доступность и интеграционные возможности также делают его привлекательным для разработчиков, которые хотят создать ботов для различных платформ. Обзор на данный конструктор чат-ботов продемонстрирован на рисунках 1.1, 1.2.

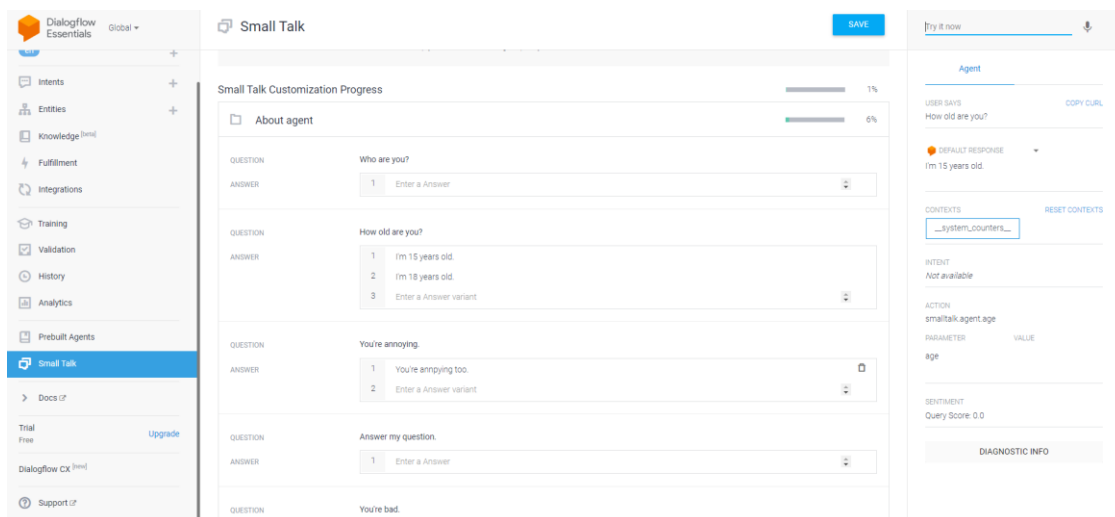


Рисунок 1.1 — Обзор *Dialogflow*

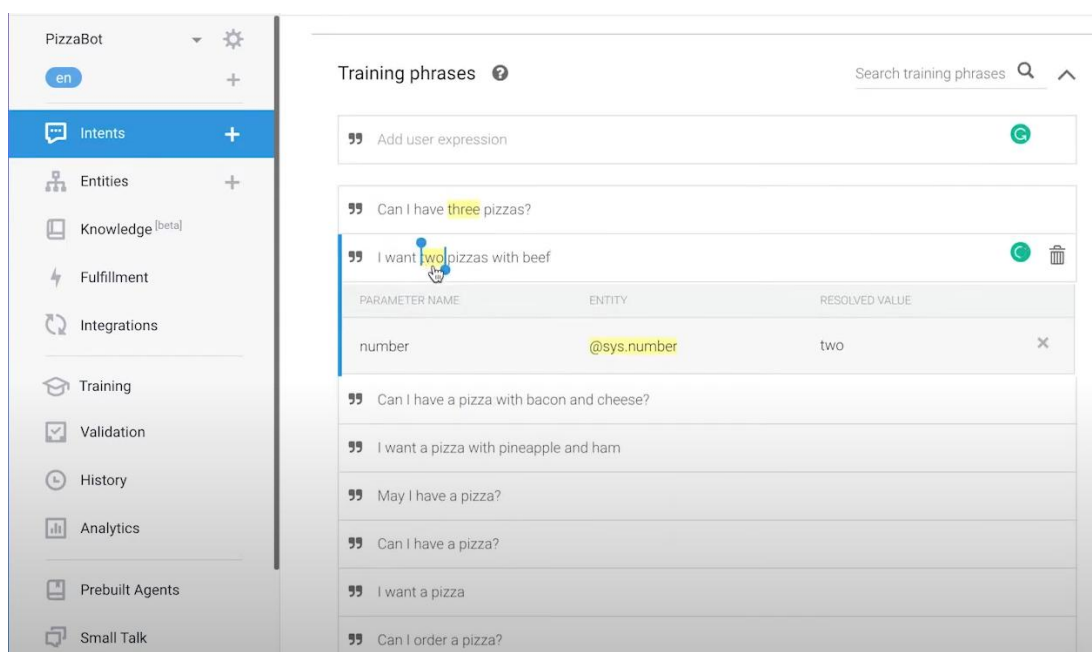


Рисунок 1.2 — Обзор *Dialogflow*

Chatfuel — один из популярных конструкторов чат-ботов, который позволяет пользователю создавать их без необходимости написания программного кода. Рассмотрим основные преимущества и недостатки данной платформы:

Преимущества:

— простота использования — для создания нужен только аккаунт в *Facebook Messenger* и основная настройка происходит через *drag-and-drop*

интерфейс;

- большое количество готовых шаблонов ботов, которые можно настроить под свои нужды;

- наличие платформы *Growth Tools* для продвижения чат-бота через различные социальные сети и рекламные кампании;

- *Chatfuel* обладает большими возможностями в настройке взаимодействия с пользователем, такими как: пользовательские переменные, использование блоков сценариев и многое другое;

- регулярные апдейты и улучшения интерфейса.

Недостатки:

- ограниченные возможности при создании сложных скриптов и дополнительных функций;

- необходимо хорошо понимать основные принципы построения бота, так как *Chatfuel* не предоставляет подробные инструкции по каждой функции;

- не всегда возможно точно настроить шаблон бота под свои нужды;

- платформа платная, существуют ограничения на бесплатный тариф, что может не позволить пользователю реализовать все задумки.

Основываясь на этих факторах, можно сделать вывод, что *Chatfuel* — это простой и удобный инструмент для создания базовых ботов без наличия глубоких знаний в программировании. Он позволяет создать чат-бота для социальной сети *Facebook* быстро и с минимумом затрат. Но, если вы планируете создать более сложный и уникальный чат-бот, возможно, вам стоит обратиться к другой платформе или рассмотреть создание бота с помощью разработки на заказ. Обзор данного конструктора чат-ботов можно увидеть на рисунках 1.3, 1.4.

Другой популярный конструктор чат-ботов — *ManyBot*. Конструктор также позволяет пользователям создавать и настраивать чат-ботов для социальной сети *Telegram* без написания программного кода.

Главным преимуществом этого конструктора является возможность создания и настройки чат-ботов внутри социальной сети *Telegram*, без необходимости переходить на сторонние веб-сайты или приложения.

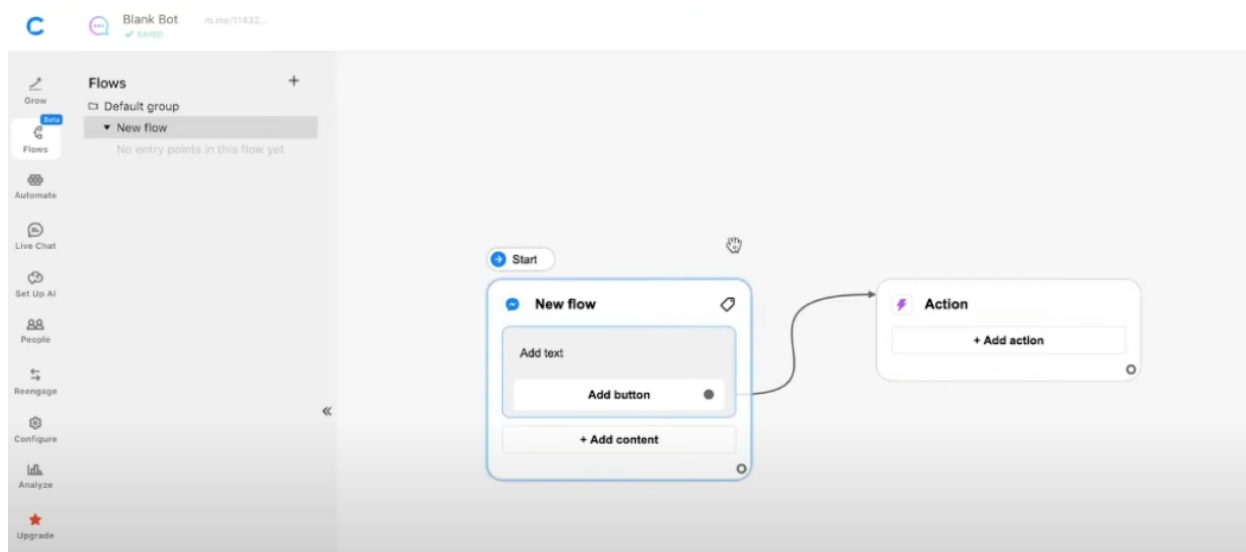


Рисунок 1.3 — Обзор *Chatfuel*

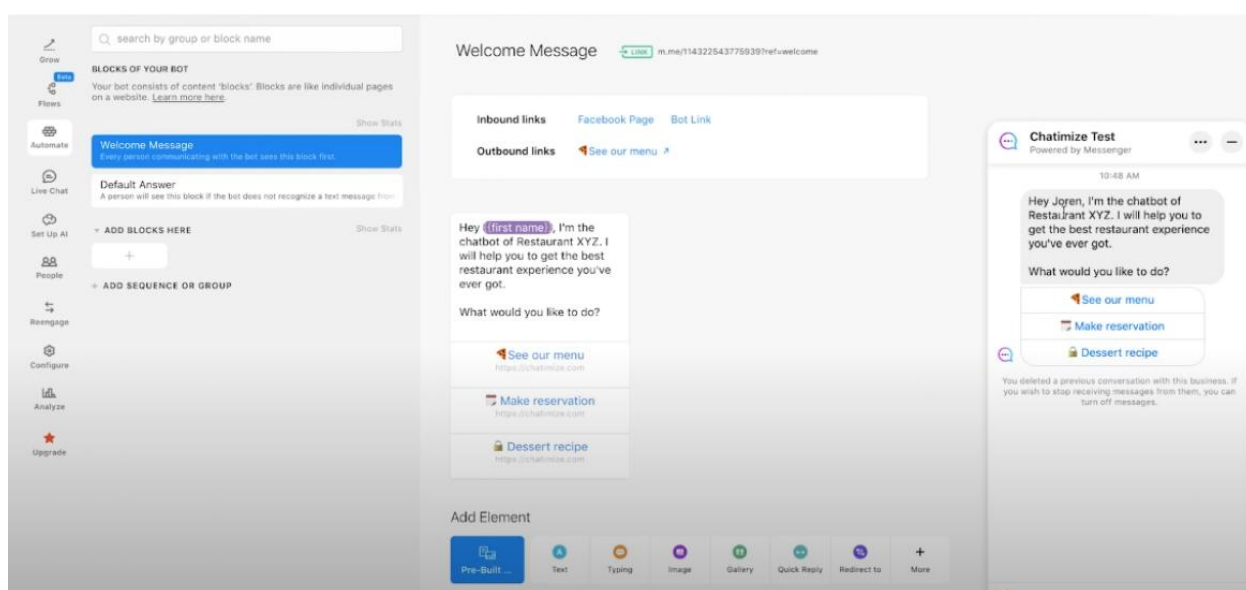


Рисунок 1.4 — Обзор *Chatfuel*

Пользователь может создать нового чат-бота, используя элементы встроенного визуального интерфейса социальной сети, такие, как, *Inline*-кнопки, текстовые сообщения и вложения, а также разослать сообщение всем пользователям бота всего в пару кликов.

Недостатки этой платформы включают ограниченный функционал создаваемых чат-ботов, а также недостаточно понятный интерфейс для некоторых пользователей. Этот конструктор также не предоставляет полноценного личного кабинета, что может быть неудобным, если вы хотите настроить дополнительные опции вашего чат-бота. Кроме того, создание чат-бота на этой платформе может занять больше времени, чем в других конструкторах.

В целом, *ManyBot* — это простой конструктор чат-ботов для *Telegram*, который мы можем использовать для создания новых чат-ботов без написания кода. Однако, его ограниченный функционал и не совсем интуитивный интерфейс могут быть недостатками для некоторых пользователей. Обзор на данный конструктор чат-ботов продемонстрирован на рисунке 1.5.

Botsify — это еще один конструктор чат-ботов, у которого есть ряд особенностей, перечислим их далее:

- простота использования. *Botsify* предлагает простой подход к созданию чат-ботов. Он предоставляет пользовательский интерфейс, который облегчает процесс настройки бота;

- интеграция со сторонними приложениями и платформами. *Botsify* обладает большим количеством интеграций и может использоваться для создания ботов на различных платформах;

- аналитика. *Botsify* предоставляет аналитические данные, которые позволяют пользователям оценить эффективность и производительность их чат-бота;

- наличие различных типов сообщений. *Botsify* поддерживает разные типы сообщений, такие как текстовые, изображения, видео и аудио, что делает чат-бота более привлекательным для пользователей;

- использование обработки естественного языка. *Botsify* использует данную технологию для обработки запросов пользователей и предоставления точного ответа на сообщение.

Недостатком *Botsify* является то, что он может быть ограничен в создании более сложных чат-ботов. Он также не обладает инструментами машинного обучения, которые могут быть полезны при создании более "умных" ботов.

					<i>ПГУ1.090301.07.001 ПЗ</i>	Лист
						18
Изм.	Лист	№ докум.	Подпись	Дата		



Рисунок 1.5 — Обзор ManyBot

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.07.001 ПЗ

Лист

19

В целом, *Botsify* — это хороший выбор для тех, кто хочет создать простого чат-бота. Он легок в использовании и предоставляет множество интеграций, а также может использоваться для создания чат-ботов на различных платформах. Он также обладает возможностью обработки естественного языка, что делает его привлекательным для пользователей, которые ищут точную и быструю обработку своего запроса. Однако, если вы хотите создать более сложного бота, *Botsify* может быть недостаточным в этом случае. Обзор данного конструктора чат-ботов можно на рисунках 1.6, 1.7.

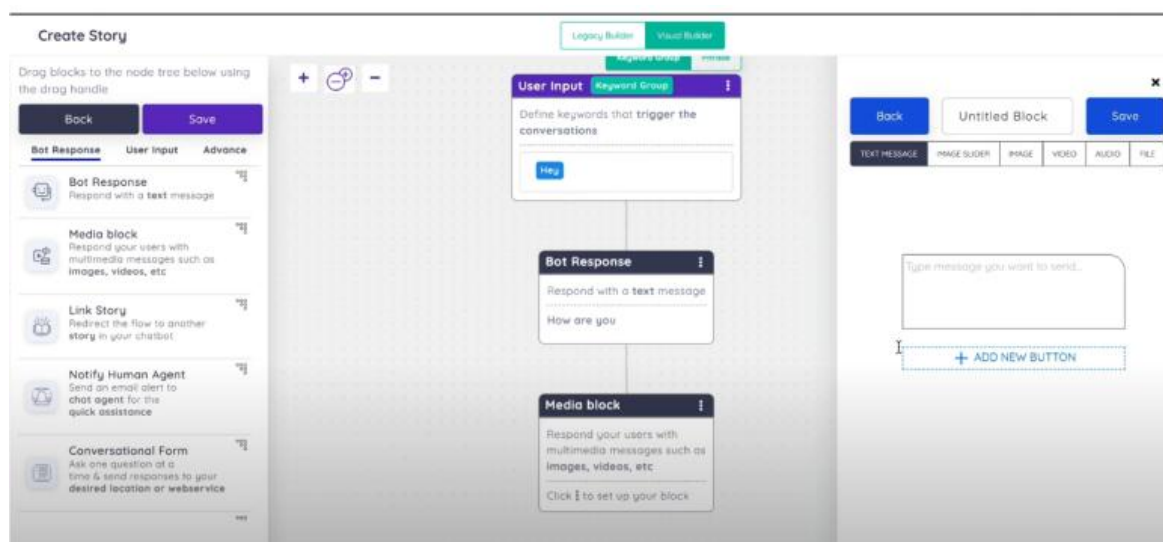


Рисунок 1.6 — Обзор *Botsify*

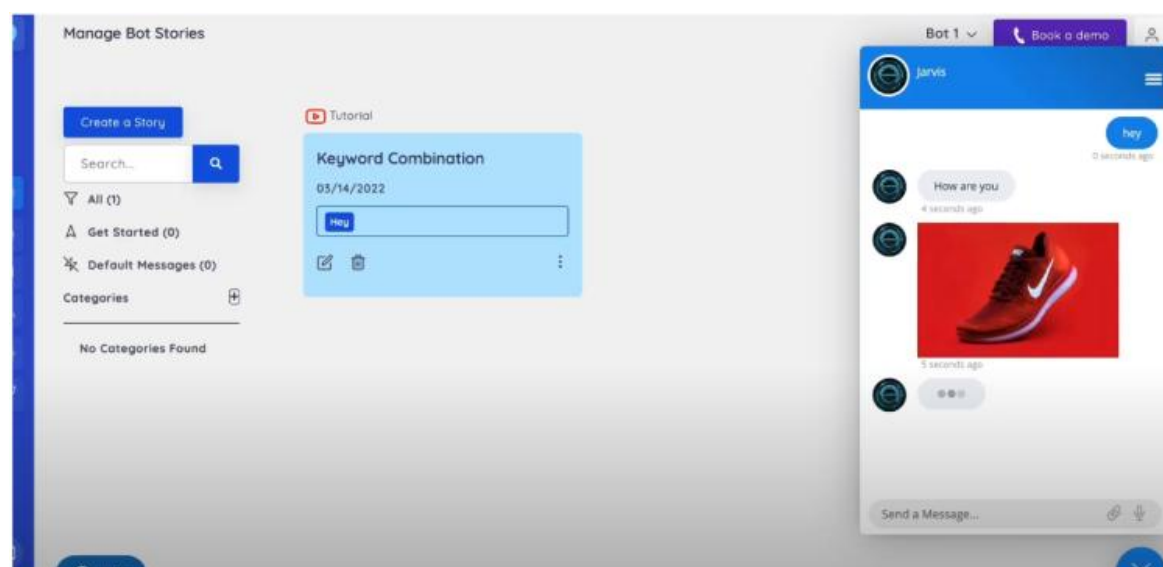


Рисунок 1.7 — Обзор *Botsify*

Проанализировав аналоги, можно выделить два основных недостатка. Во-первых, все представленные конструкторы — зарубежные, и не соблюдают требования 152-го Федерального закона «О персональных данных», данные о пользователях не хранятся на серверах Российской Федерации. Во-вторых, все рассмотренные аналоги для пользования ими требуют оплату, либо предоставляют пользователю пробную версию на несколько дней.

1.5 Анализ и структура конструктора чат-ботов

Конструктор чат-ботов для *Telegram* — это инструмент, который позволяет создавать ботов для *Telegram* без написания кода. Это визуальный конструктор, который позволяет пользователям создавать собственные чат-боты в *Telegram*, используя различные функции, такие как ввод текста, кнопки, меню и т.д.

Архитектура диалоговой системы состоит из трех основных составляющих частей:

- клиентская часть;
- серверная часть;
- БД.

Клиентская часть — графическая часть приложения, предназначенная для взаимодействия пользователя с программным продуктом. Клиентская часть состоит из форм регистрации и авторизации, личного кабинета и панелей редактирования функционала чат-бота, которые состоят из различных функциональных кнопок, а также текстовых полей ввода, в эти поля пользователь может занести какую-либо информацию, которая различным образом будет обрабатываться или выводиться в будущем чат-боте. Кроме того, клиентская часть конструктора отвечает за проверку данных, вводимых пользователем. Необходимо это для того, чтобы серверная часть смогла получить и корректно обработать данные для регистрации пользователей и создания чат-ботов. Также клиентской частью можно назвать полученного в результате чат-бота в *Telegram*.

Серверная часть работает на стороне сервера и обрабатывает информацию, отправленную пользователем на сервер и выполняет требующиеся алгоритмы для выполнения определенной задачи.

База данных — это структурированный набор данных, который хранится в электронном виде и может быть использован для различных целей в проекте. БД хранятся и обрабатываются в соответствии с определенными правилами и структурами, так что данными легко управлять и использовать.

Клиентское приложение представлено в виде приложения, которое является интерфейсом пользователя конструктора. Конструктор позволяет пользователю добавлять или удалять собственных чат-ботов для мессенджера и изменять их функционал. Была разработана диаграмма прецедентов, которая изображена на рисунке 1.8 [3].

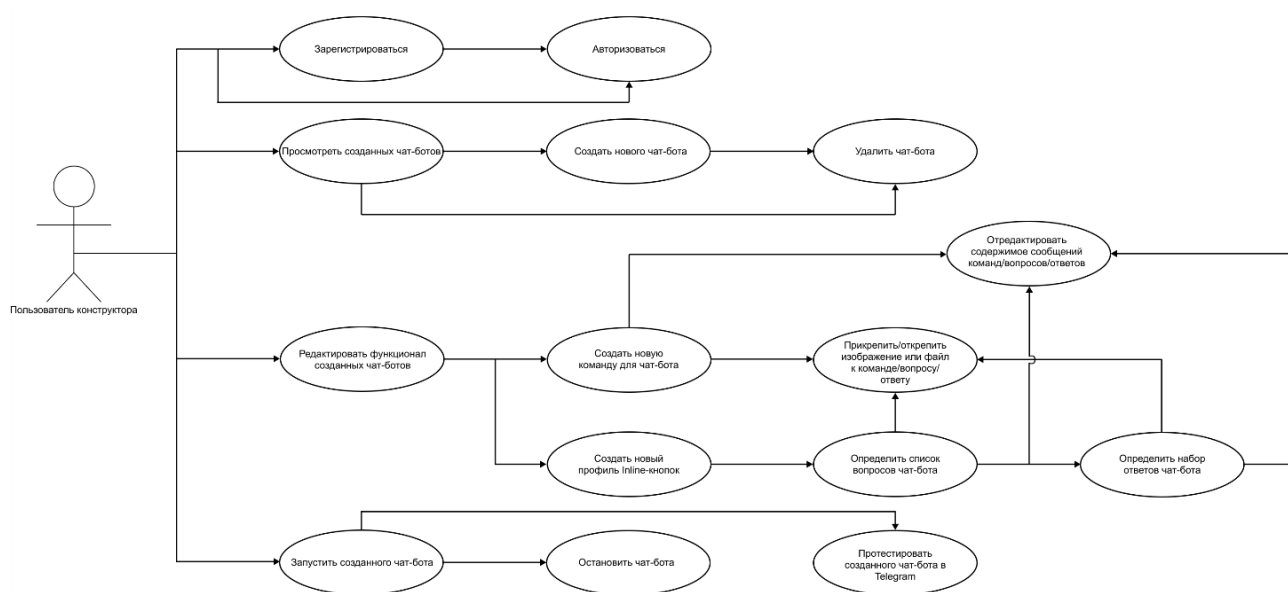


Рисунок 1.8 — UML-диаграмма прецедентов

Для наглядного восприятия взаимодействия компонентов конструктора, была разработана UML-диаграмма последовательности, которая продемонстрирована на рисунке 1.9 [3].

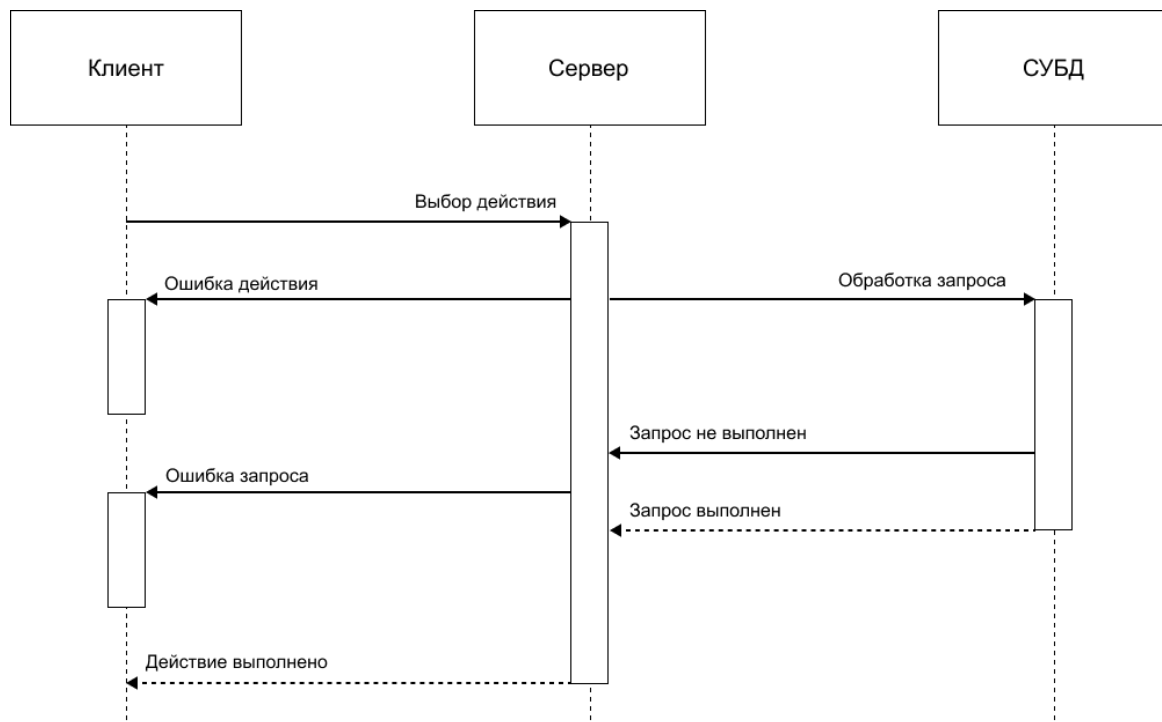


Рисунок 2.9 — UML-диаграмма последовательности

1.6 Выбор инструментов разработки

1.6.1 Выбор мессенджера

Изучая мессенджеры, на которых можно базировать создаваемых в конструкторе чат-ботов, были обнаружены следующие популярные мессенджеры на территории СНГ приложения: *WhatsApp*, *Viber*, *Telegram*.

Telegram — это мессенджер, который также может использоваться для создания чат-ботов. Некоторые из основных преимуществ *Telegram* для создания чат-ботов включают в себя:

- поддержка различных типов сообщений. *Telegram* поддерживает различные типы сообщений, такие как текст, изображения, видео и аудио, что позволяет боту создавать более интересный и удобный пользовательский интерфейс;

- большая аудитория. *Telegram* имеет большую аудиторию пользователей, что означает, что созданный чат-бот может быть доступен для широкого круга пользователей;

– защищенность. *Telegram* предлагает высокий уровень защиты для обмена сообщениями между ботом и пользователями. Вся информация, передаваемая через чат-ботов, защищена с помощью протокола шифрования *TLS*.

– *API* и *SDK*. *Telegram* предоставляет *API* и *SDK* для разработчиков для создания более интеллектуальных и сложных чат-ботов;

– инструменты разработки. *Telegram* обеспечивает пользовательские инструменты разработки и документацию, которые облегчают и ускоряют процесс создания бота;

– умный командный интерфейс. *Telegram* предлагает умный командный интерфейс, который позволяет пользователям легко управлять ботом через ключевые слова и команды;

– возможность интеграции с другими сервисами. *Telegram* легко интегрируется с другими сервисами, что делает его более гибким и функциональным для создания чат-бота;

– различные опции для хранения данных. *Telegram* предлагает различные опции для хранения данных, включая БД на стороне клиента, использование серверов *Telegram*, а также возможность работы с собственными серверами.

В целом, *Telegram* — это популярный выбор для создания чат-ботов благодаря своим различным функциональным возможностям, удобству в использовании и защищенности. Он также предлагает *API* и *SDK* для создания уникальных и более сложных чат-ботов с использованием различных технологий и алгоритмов.

1.6.2 Выбор языка программирования

Рассматривая процесс разработки программного продукта, такого как конструктор чат-ботов, необходимо с особой ответственностью подойти к выбору инструментов разработки, таких как мессенджер, язык программирования, и СУБД. Критически важно тщательно выбрать язык программирования, который соответствует требованиям проекта и обеспечит успешную реализацию требуемой платформы.

C# — это язык программирования с Си-подобным синтаксисом. Он уже продолжительное время успешно удерживает одно из лидирующих мест в рейтинге наиболее востребованных языков программирования на рынке. Для работы с *Telegram API* на языке *C#* существует несколько библиотек, таких как *Telegram.Bot*, которые облегчают процесс написания и развертывания кода. Эти библиотеки предоставляют доступ к основным функциям *Telegram API*, таким как отправка и получение сообщений, управление контактами, файлами и обновлениями. Одним из основных достоинств *C#* является наличие мощных функций, и предоставление прямого доступа к оперативной памяти. Недостатком языка является приоритетная ориентированность на платформу *Windows*.

JavaScript — это язык программирования, который широко используется веб-разработке, т.к. его можно использовать в браузере для создания интерактивных элементов на веб-страницах или в серверной среде для обработки запросов.

Основные преимущества языка *JavaScript*:

- простой и доступный язык программирования, который может использоваться для создания сложных приложений;
- высокая скорость выполнения кода благодаря многим *JavaScript*-библиотекам и фреймворкам;
- возможность использования *JavaScript* как на клиентской, так и на серверной стороне приложения (например, с помощью *Node.js*).

Основные недостатки *JavaScript*:

- иногда считается, что *JavaScript* слабо типизированный язык, что может приводить к сложностям при разработке больших приложений;
- поскольку это клиентский язык, код *JavaScript* может быть легко изменен и используется вредоносными программами;
- некоторые программисты считают, что *JavaScript* менее производительный, чем другие языки, такие как *Java* и *C++*.

При помощи языка *JavaScript* возможно создавать чат-ботов в *Telegram*, используя фреймворки и библиотеки, такие как *Telegraf.js*, который предоставляет конструктор ботов, и *Tgbot.js*, который является модулем для *Telegram* бота на асинхронном *JavaScript*-коде. Однако, при выборе *JavaScript* для создания конструктора чат-ботов в *Telegram*, следует учитывать его особенности, включая слабую типизацию и уязвимости программы, а также выбрать наиболее подходящие фреймворки и библиотеки для поддержки разработки.

Java — это объектно-ориентированный язык программирования, широко используемый для разработки приложений, включая веб-приложения, мобильные приложения, настольные приложения и т.д. Важными особенностями языка являются: кроссплатформенная поддержка, обширная и постоянно растущая коллекция библиотек и фреймворков, что упрощает разработку и позволяет использовать готовые решения для быстрой разработки чат-ботов и высокий уровень безопасности и надежности, что особенно важно при работе с конфиденциальной информацией, такой как данные клиентов [4]. В настоящее время доступны различные библиотеки и фреймворки для работы с *Telegram*, такие как *Java Telegram Bot API* или *aTelegramBot*, которые упрощают разработку собственных чат-ботов в *Telegram*. Однако, при выборе *Java* для создания конструктора чат-ботов в *Telegram* следует учитывать его особенности, включая требовательность к ресурсам и увеличенное время запуска программы.

Python — это высокоуровневый язык программирования, который широко используется в различных программах, таких как веб-приложения, научные вычисления и искусственный интеллект. Он относится к языкам с динамической типизацией, что означает, что тип переменной определяется автоматически во время выполнения программы [5].

Я выбрал *Python* для разработки моего проекта, так как этот язык имеет ряд преимуществ над другими языками:

- простой язык с понятным синтаксисом, что упрощает изучение и разработку кода;
- наличие большого количества библиотек и фреймворков, это упрощает и ускоряет разработку различных приложений, включая создание чат-ботов;
- *Python* является кроссплатформенным языком программирования, что позволяет запускать приложения, написанные на *Python*, на разных ОС;
- высокая производительность. *Python* отличается высокой эффективностью и может использоваться для разработки быстрых и эффективных приложений;
- популярность языка программирования. Существует огромное сообщество разработчиков, которые создают новые библиотеки и инструменты, что позволяет разработчикам быстро находить решения для проблем и делает *Python* одним из наиболее удобных языков для работы.

Python также является отличным вариантом для создания конструктора чат-ботов. Например, для этой задачи могут использоваться библиотеки, такие как *PyTelegramBotAPI*, *Telebot*, *BotBuilder*, *Botogram* и другие, которые будут упрощать разработку чат-ботов в *Telegram*.

Именно поэтому я выбрал данный язык для разработки пользовательской платформы для создания чат-ботов в социальной сети *Telegram*.

1.6.3 Выбор СУБД

Выбор СУБД является важным моментом при разработке ПО. СУБД позволяют надежно и удобно хранить, организовывать и обрабатывать большие объемы данных, используемых при работе программы.

Одним из главных факторов при выборе СУБД является ее совместимость с требованиями приложения. Также необходимо учесть такие факторы, как безопасность, доступность и надежность. При работе с большим объемом данных необходимо учесть масштабируемость и высокую производительность выбираемой СУБД.

Oracle Database — это объектно-реляционная модель СУБД, разработанная компанией *Oracle Corporation*. Она поддерживается на многих ОС, таких как *Linux*, *Windows*, *Unix* и другие. *Oracle* является одной из самых мощных и распространенных СУБД сегодня и используется для решения широкого спектра задач и разработки приложений.

Oracle является надежным, масштабируемым и безопасным решением для хранения, запроса и манипулирования данными в бизнес-процессах. Она поддерживает транзакционную обработку и управление данными, и обеспечивает множество возможностей для управления данными и настройки БД. Она также обладает рядом инструментов и функций, которые позволяют пользователю анализировать данные, создавать запросы и отчеты, а также создавать резервные копии для обеспечения сохранности данных.

Oracle Database также имеет широкое сообщество пользователей и разработчиков, которые предлагают множество дополнительных инструментов и ресурсов, чтобы помочь пользователям оптимизировать работу с БД. Она также является одной из наиболее популярных СУБД для разработки приложений на языке программирования *Java*.

SQLite — это легкий и мобильный встраиваемый движок СУБД, который имеет небольшой размер и не требует сложной настройки для работы. Она является самым распространенным движком СУБД, широко используется в мобильных приложениях, настольных приложениях, веб-приложениях и даже встроена во множество приложений для работы без подключения к интернету.

SQLite хранит данные в сжатой форме в одном файле, что упрощает архитектуру приложения и ускоряет работу с данными. Она поддерживает большинство стандартных операторов *SQL* и множество функций, включая транзакционную обработку и конкурентный доступ.

SQLite может работать на многих ОС, таких как *Windows*, *Unix*, *MacOS* и *Android*. *SQLite* обычно используется в малых проектах, например, в локальных хранилищах данных или в мини системах без возможности использования сети.

PostgreSQL — это мощная реляционная СУБД, которая является одной из наиболее популярных систем управления базами данных в мире. Ее особенностью является открытый исходный код и отсутствие расходов на лицензионные сборы, что делает ее доступной и привлекательной для малых, средних и крупных организаций.

PostgreSQL обладает множеством преимуществ, которые делают ее отличным выбором для создания приложений. Одно из главных преимуществ данной СУБД — это высокий уровень надежности и устойчивости. СУБД *PostgreSQL* разработана таким образом, что она может правильно обрабатывать сбои в работе оборудования или прерывания связи, что обеспечивает высокий уровень защиты ваших данных [6].

Кроме того, данная СУБД обладает широким набором функций, которые позволяют легко создавать и управлять базами данных любого размера и сложности. Она поддерживает большинство стандартов *SQL* и предоставляет мощные возможности по совместному использованию данных и управлению транзакциями.

Еще одним преимуществом является то, что *PostgreSQL* легко масштабируется, что позволяет без проблем изменять емкость БД в зависимости от потребностей вашего приложения. Также *PostgreSQL* прекрасно работает на различных платформах, что дает возможность использовать ее в широком ряде приложений.

1.6.4 Выбранные технологии

В ходе проведенных анализов, были приняты следующие решения:

- выбрать *Telegram* как мессенджер для базирования создаваемых чат-ботов;
- выбрать среду разработки *Pycharm IDE* и язык программирования *Python* с последующим применением библиотек *python-telegram-bot* и *PyQt6*;
- выбрать СУБД *PostgreSQL* для управления базами данных.

1.7 Реализация

1.7.1 Регистрация чат-бота

Регистрация *Telegram*-бота является первым шагом для создания и интеграции бота в информационную систему. В результате создания бота мы получим его *API*-токен, который в последствии нужно будет занести в конструктор. В этом разделе будет описана подробная процедура регистрации *Telegram*-бота [1].

Для начала нужно открыть официальное приложение *Telegram* на мобильном устройстве или компьютере и создать новый аккаунт, если его еще нет. Если уже есть аккаунт, нужно войти в него. Главное меню *Telegram* показано на рисунке 1.10.

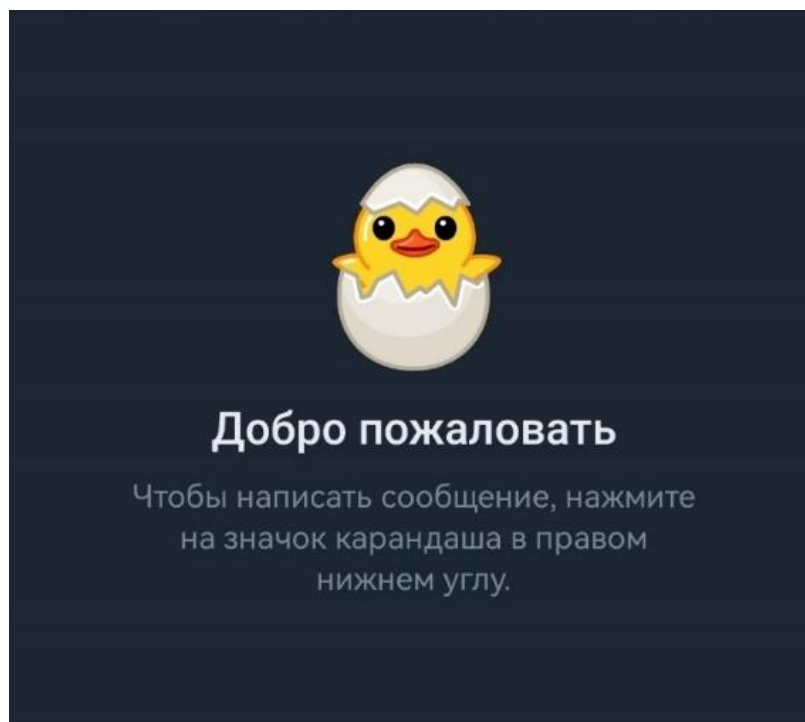


Рисунок 1.10 — Главное меню *Telegram*

Далее необходимо найти в поиске пользователей *Telegram* бота с именем "*BotFather*" — это официальный бот *Telegram* для создания новых ботов. Пример поиска показан на рисунке 1.11.



Рисунок 1.11 — Поиск *BotFather*

Далее необходимо начать работу бота путем нажатия кнопки «/start» или написанием аналогичной команды в чат. Ответ чат-бота показан рисунке 1.12.

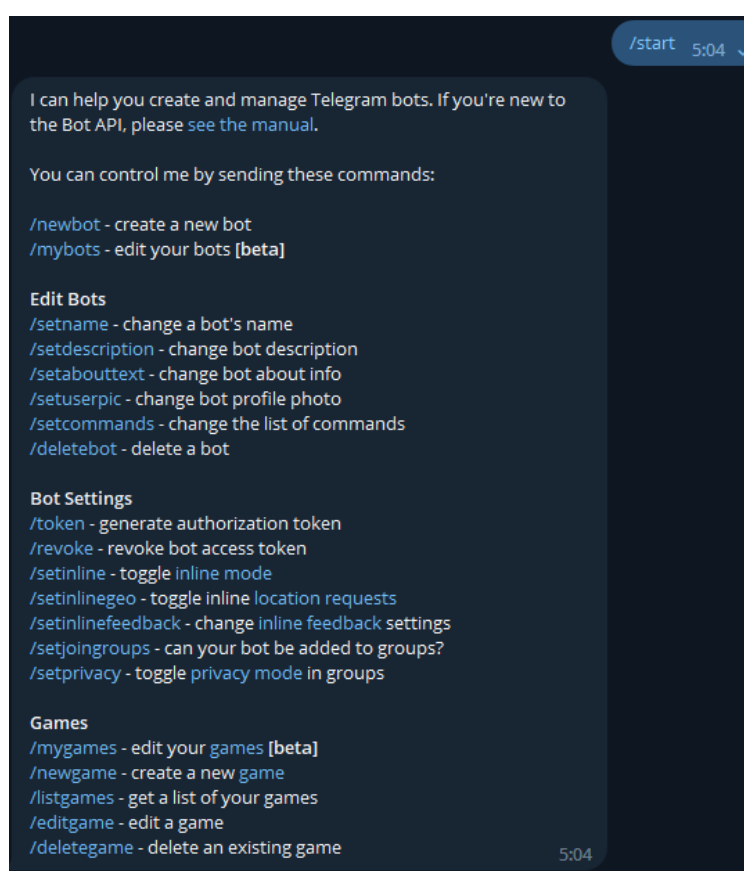


Рисунок 1.12 — Начало диалога с *BotFather*

Затем необходимо ввести команду */newbot* и задать боту имя, которое будет отображаться в поиске и логин, который будет отображаться в ссылке. Логин бота обязательно должен заканчиваться на *-bot*. Процесс создания нового чат-бота показан на рисунке 1.13.

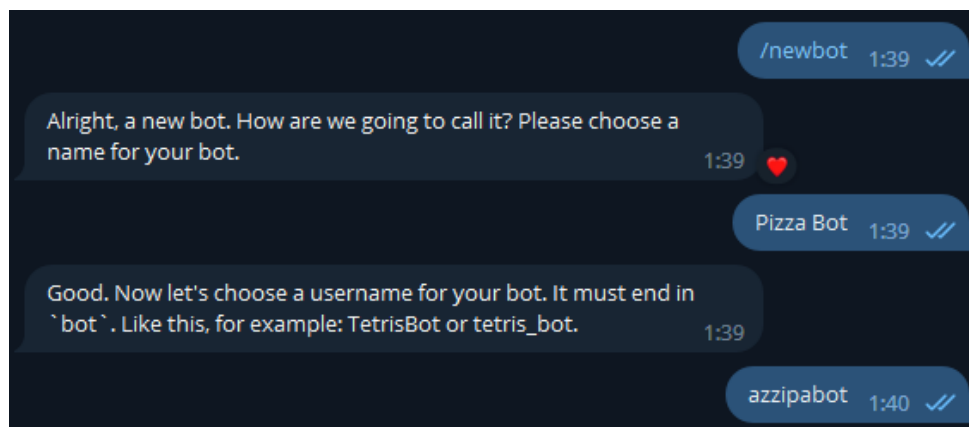


Рисунок 1.13 — Процесс создания бота

Далее *BotFather* предоставляет *API*-токен, который является ключом доступа к вашему *Telegram*-боту. При нажатии на токен он скопируется в буфер обмена. Полученный от *BotFather* токен должен храниться в надежном и безопасном месте, например, в блокноте или специальной программе для хранения личных данных. Данный токен будет использоваться при добавления нового бота в конструкторе для аутентификации бота при взаимодействии с *Telegram API*. Информация, полученная от бота показана на рисунке 1.14.

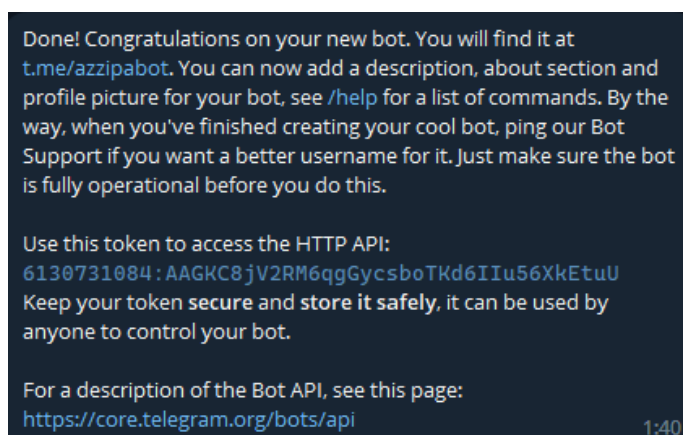


Рисунок 1.14 — Получение токена бота

После завершения всех этих шагов *Telegram*-бот успешно зарегистрирован. Теперь можно использовать токен бота для создания его функционала в конструкторе.

1.7.2 Работа с БД

Для хранения информации, используемой при создании чат-ботов и регистрации пользователей, требуется реализовать базу данных со следующими таблицами:

- *answers* — таблица, содержащая финальные ответы чат-бота на полученные от пользователя ответы на заданные чат-ботом вопросы;
- *bots* — таблица, содержащая информацию о созданных пользователями конструктора чат-ботов;
- *bots_users* — таблица, содержащая информацию о пользователях, активировавших в *Telegram* созданных в конструкторе чат-ботов;
- *buttons* — таблица, содержащая информацию о созданных пользователями конструктора *Inline*-клавиатур;
- *commands* — таблица, содержащая информацию о добавленных пользователями командах;
- *questions* — таблица, содержащая информацию о созданных пользователями вопросов для чат-ботов;
- *users* — таблица, содержащая информацию об учетных записях пользователей конструктора.

Для наглядного понимания работы БД, на рисунке 1.15 представлена схема физической модели базы данных [3].

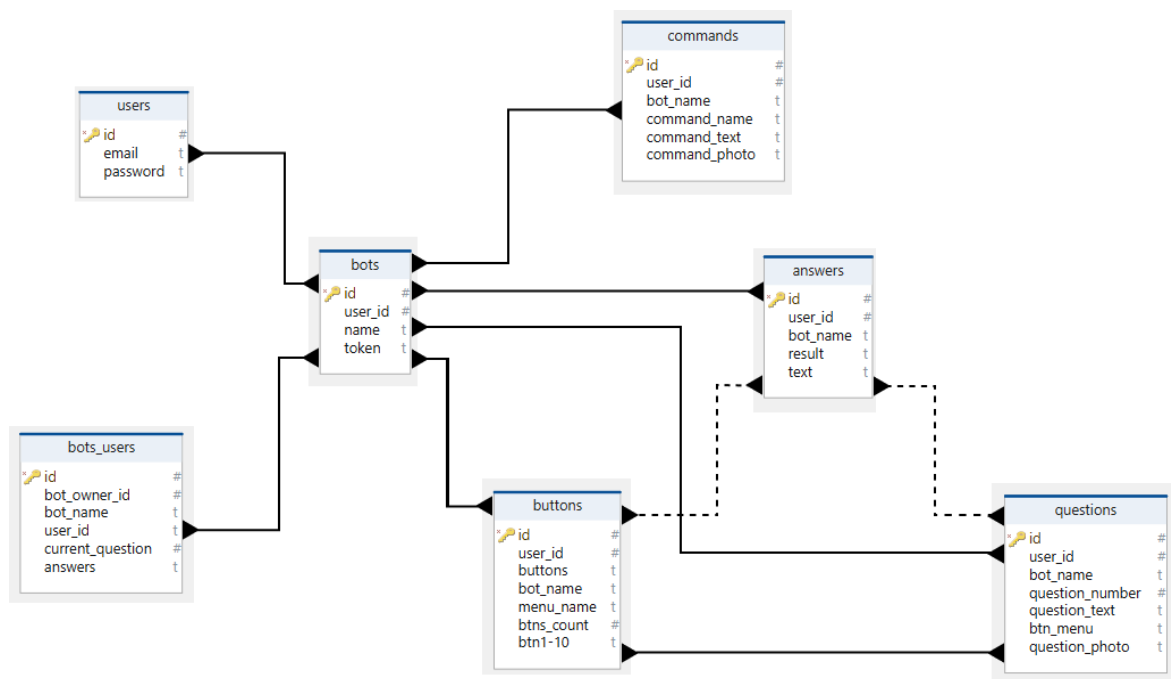


Рисунок 1.15 — Физическая модель БД

Таблица 1 — Атрибуты таблицы *answers*

Имя столбца	Тип данных	Описание	Пример
<i>id</i>	<i>[PK]integer</i>	Идентификатор чат-бота	1
<i>user_id</i>	<i>integer</i>	Идентификатор пользователя, создавшего чат-бота	2
<i>bot_name</i>	<i>character varying</i>	Название чат-бота	<i>BotCarConsultant</i>
<i>result</i>	<i>character varying</i>	Ответы пользователей	Седан Бензин Красный Большой
<i>text</i>	<i>character varying</i>	Текст сообщения для ответа чат-бота	Мы подобрали для вас следующую модель! <i>Tesla Model 3</i> . Стоимость от 4 млн рублей.

Имя столбца	Тип данных	Описание	Пример
<i>photo</i>	<i>character varying</i>	Путь к файлу для ответа чат-бота	<i>A:\javaprojects\ChatbotBuilder\2_BotCarConsultant\photos\2021-Tesla-Model-S-P.jpg</i>

Таблица 2 — Атрибуты таблицы *bots*

Имя столбца	Тип данных	Описание	Пример
<i>id</i>	<i>[PK]integer</i>	Идентификатор чат-бота	3
<i>user_id</i>	<i>integer</i>	Идентификатор пользователя, создавшего чат-бота	4
<i>name</i>	<i>character varying</i>	Название чат-бота	<i>BotOleg</i>
<i>token</i>	<i>character varying</i>	API-токен чат-бота	<i>5988496662:AAFDAhFRX29uWtJn2EyWlTOxLkqByf2bOXQ</i>

Таблица 3 — Атрибуты таблицы *bots_users*

Имя столбца	Тип данных	Описание	Пример
<i>id</i>	<i>[PK]integer</i>	Идентификатор пользователя чат-бота	1
<i>bot_owner_id</i>	<i>character varying</i>	Идентификатор создателя чат-бота	2

<i>bot_name</i>	<i>character varying</i>	Название чат-бота	<i>BotCarConsultant</i>
Имя столбца	Тип данных	Описание	Пример
<i>user_id</i>	<i>integer</i>	Идентификатор чата с пользователем чат-бота	494379289
<i>current_ques tion</i>	<i>integer</i>	Номер вопроса, на котором остановился пользователь	5
<i>answers</i>	<i>character varying</i>	Ответы пользователя	Хэтчбек Бензин Зеленый Малый

Таблица 4 — Атрибуты таблицы *buttons*

Имя столбца	Тип данных	Описание	Пример
<i>id</i>	<i>[PK]integer</i>	Идентификатор профиля <i>Inline</i> -клавиатур	1
<i>user_id</i>	<i>integer</i>	Идентификатор пользователя, которому принадлежит данный профиль <i>Inline</i> - клавиатуры	2
<i>bot_name</i>	<i>character varying</i>	Название чат-бота, для которого создан профиль	<i>BotCarConsultant</i>
<i>menu_name</i>	<i>character varying</i>	Название профиля <i>Inline</i> - клавиатуры	Цвет автомобиля
<i>btns_count</i>	<i>integer</i>	Количество кнопок в клавиатуре	4

<i>btn1-10</i>	<i>character varying</i>	Текст кнопки	Хэтчбек
----------------	------------------------------	--------------	---------

Таблица 5 — Атрибуты таблицы *commands*

Имя столбца	Тип данных	Описание	Пример
<i>id</i>	<i>[PK]integer</i>	Идентификатор команды	1
<i>user_id</i>	<i>integer</i>	Идентификатор пользователя, который добавил команду	6
<i>bot_name</i>	<i>character varying</i>	Название чат-бота, для которого создана команда	<i>ChainBot1</i>
<i>command_name</i>	<i>character varying</i>	Название команды	<i>/help</i>
<i>command_text</i>	<i>character varying</i>	Текст сообщения-ответа на команду	Здравствуйте, я готов проконсультировать вас по выбору автомобиля.
<i>command_photo</i>	<i>character varying</i>	Путь к файлу для сообщения-ответа на команду	<i>[null]</i>

Таблица 6 — Атрибуты таблицы *questions*

Имя столбца	Тип данных	Описание	Пример
<i>id</i>	<i>[PK]integer</i>	Идентификатор вопроса	65
<i>user_id</i>	<i>integer</i>	Идентификатор пользователя, который добавил вопрос	143
Имя столбца	Тип данных	Описание	Пример
<i>bot_name</i>	<i>character varying</i>	Название чат-бота, для которого добавлен вопрос	<i>BotQuestion</i>
<i>question_number</i>	<i>integer</i>	Номер вопроса в порядке отправления чат-ботом	2
<i>question_text</i>	<i>character varying</i>	Текстовое содержание сообщения-вопроса	Какой цвет вам нравится?
<i>btn_menu</i>	<i>character varying</i>	Название профиля <i>Inline</i> -клавиатуры для вопроса	Цвет автомобиля
<i>question_photo</i>	<i>character varying</i>	Путь к файлу для сообщения-вопроса	<i>[null]</i>

Таблица 7 — Атрибуты таблицы *users*

Имя столбца	Тип данных	Описание	Пример
<i>id</i>	<i>integer</i>	Идентификатор пользователя конструктора	1

<i>email</i>	<i>character varying</i>	Электронная почта пользователя	<i>alekseidorofeev1353@mail.ru</i>
<i>password</i>	<i>character varying</i>	Зашифрованный пароль пользователя	<i>ed734d8fee3ca52b94c4dcf 067188ab268450334418d b4c1c4 8f89c271bc810d</i>

1.7.3 Описание доступных функций

Созданная платформа предлагает ряд функций:

- регистрация и авторизация пользователей;
- добавление и удаление новых чат-ботов;
- редактирование функционала созданных ботов;
- запуск/приостановка работы чат-ботов.

Для начала работы с конструктором необходимо запустить исходный файл *bots_constructor.py*. Пользователя встретит окно авторизации, которое можно увидеть на рисунке 1.16.

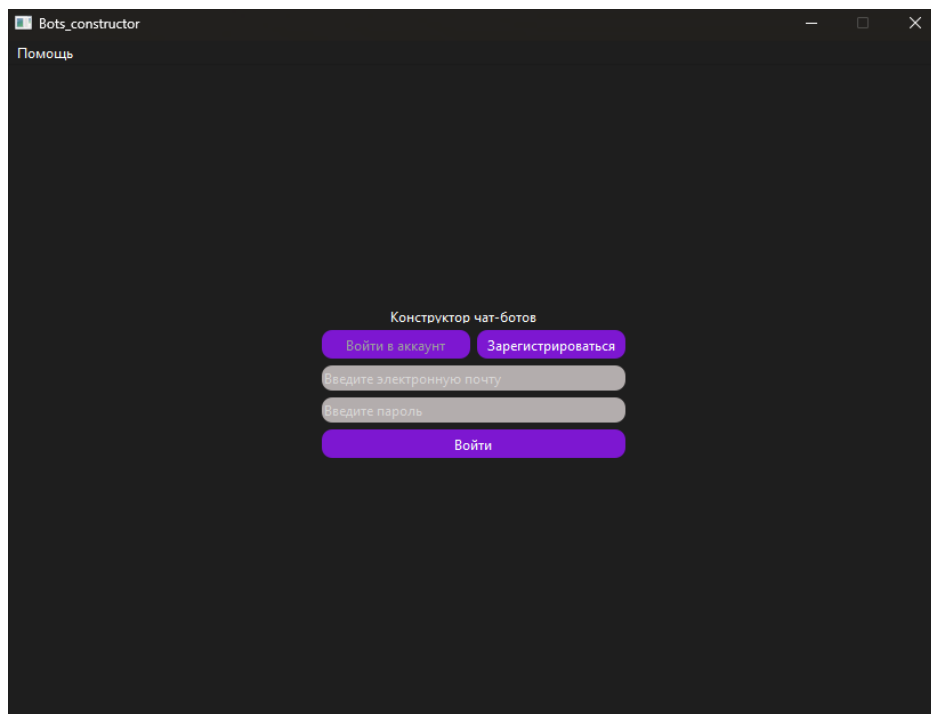


Рисунок 1.16 — Окно авторизации

Далее пользователю необходимо пройти этап регистрации, если он еще не зарегистрирован в системе, для этого необходимо нажать на кнопку «Зарегистрироваться». Пользователя перенесет на окно регистрации, в полях которого необходимо ввести электронную почту, которая будет служить логином пользователя и пароль. После внесения данных, необходимо нажать на кнопку «Зарегистрироваться», пользователь должен получить сообщение об успешной регистрации. Процесс регистрации показан на рисунке 1.17.

Рисунок 1.17 — Регистрация пользователя

Пользователь может ввести некорректную электронную почту, несовпадающие пароли или пароль, который будет не соответствовать политике безопасности, то есть меньше 6 символов, в таком случае пользователь получит предупреждение о соответствующей ошибках, которые изображены на рисунке 1.18.

Рисунок 1.18 — Ошибки при регистрации

При успешной регистрации пользователю необходимо перейти обратно на страницу авторизации, нажав на кнопку «Войти в аккаунт», в полях которых уже будут занесены данные, введенные ранее при регистрации. Для того, чтобы авторизоваться пользователю необходимо ввести логин и пароль в соответствующие поля, в нашем случае, они уже внесены, и нажать на клавишу «Войти», после чего пользователя перенесет на страницу личного кабинета, на которой отображены все созданные пользователем чат-боты. Личный кабинет пользователя изображен на рисунке 1.19.

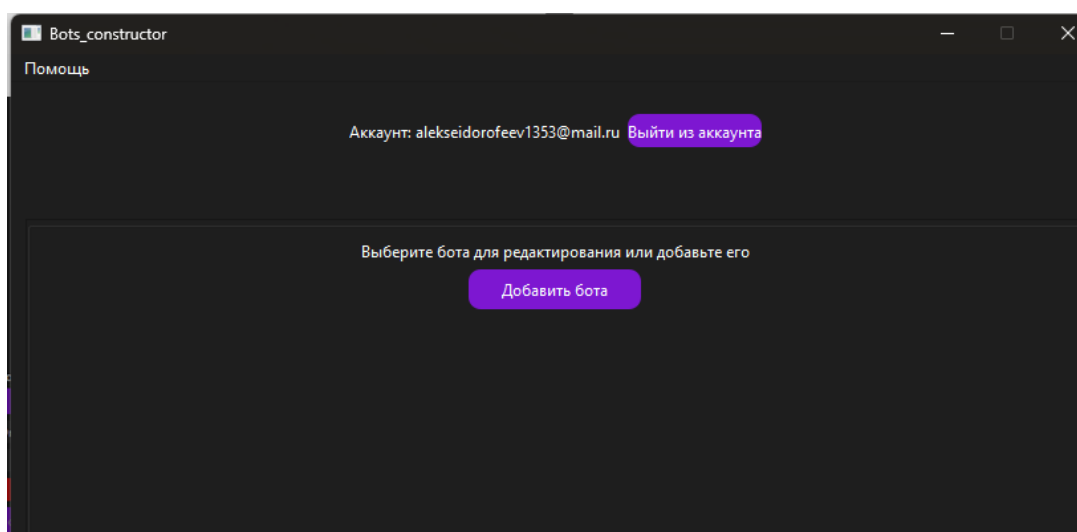


Рисунок 1.19 — Личный кабинет пользователя

У пользователей, не имеющих опыта работы с конструктором могут возникнуть некоторые трудности при создании чат-ботов с помощью инструмента. Для того, чтобы получить справку о создании чат-ботов в *BotFather* и общей информации о работе конструктора необходимо нажать на кнопки «Помощь»-«Справка», после чего пользователь увидит модальное окно с текстом о помощи при использовании, это окно продемонстрировано на рисунке 1.20. Пользователь может закрыть данное окно, нажав на «Ок», после чего его перенесет обратно в личный кабинет.

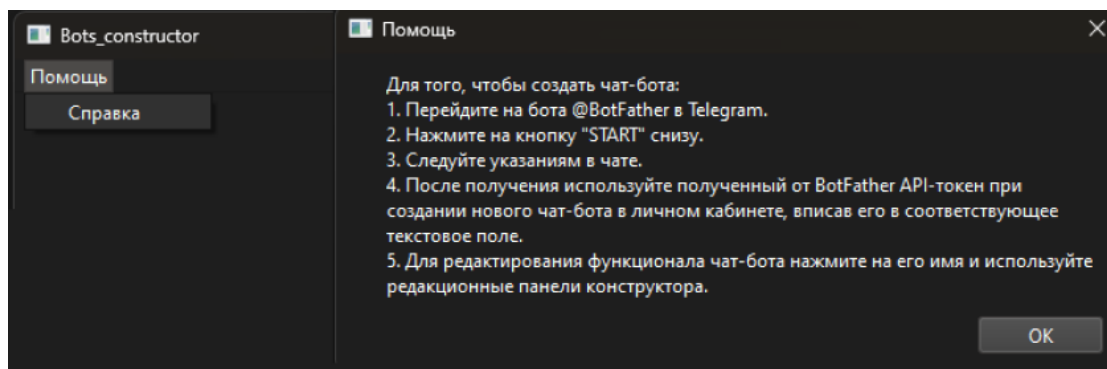


Рисунок 1.20 — Справочная информация об использовании конструктора

Для создания нового чат-бота необходимо нажать на кнопку «Добавить бота». В появившихся ниже текстовых полях необходимо ввести название чат-бота и *API*-токен созданного ранее в *BotFather* чат-бота и нажать кнопку «Добавить». Процесс добавления чат-бота в конструктор продемонстрирован на рисунке 1.21.

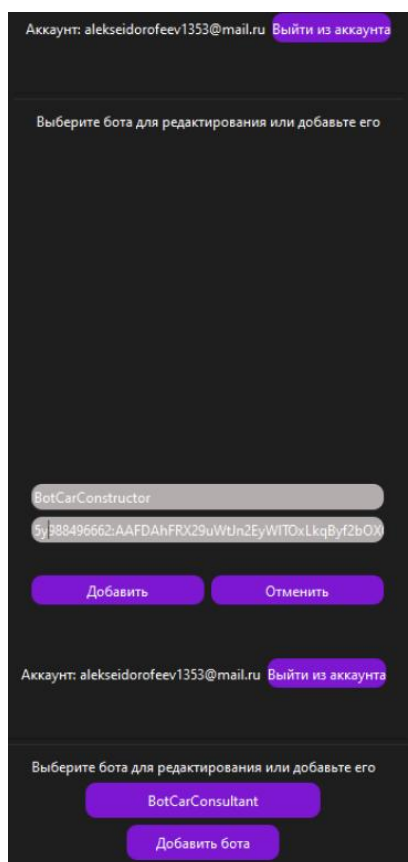


Рисунок 1.21 — Добавление чат-бота

Для начала редактирования чат-бота, необходимо нажать на кнопку с названием ранее добавленного в конструктор чат-бота. Пользователя перенесет в меню редактирования функционала чат-бота и управления его запуском, которое показано на рисунке 1.22.

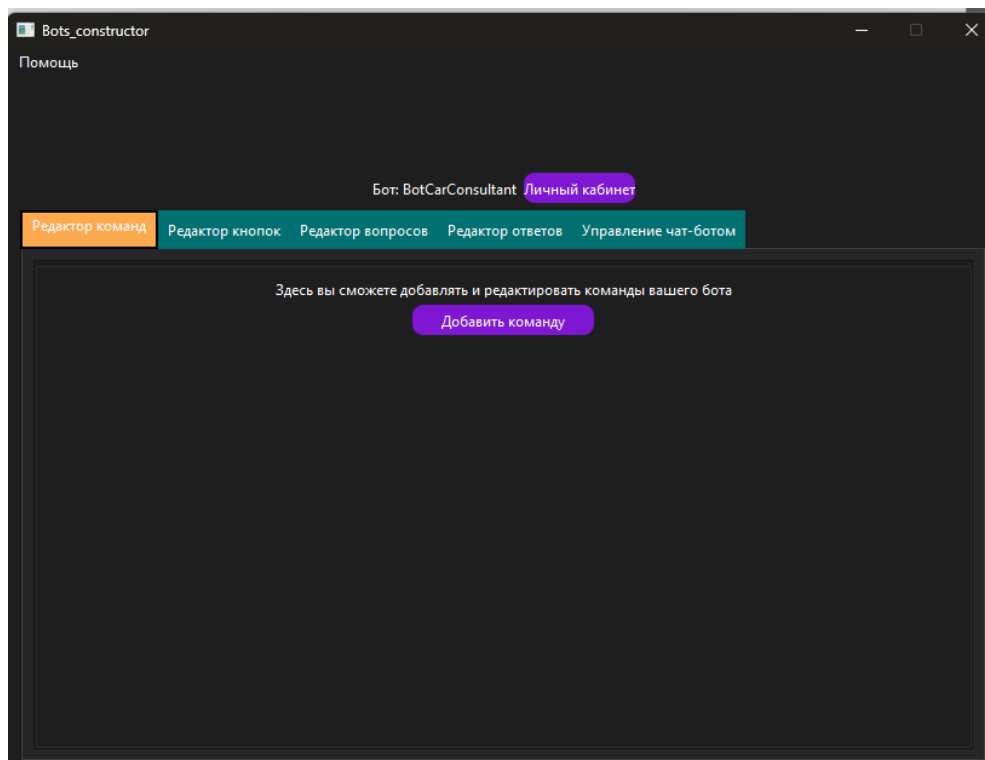


Рисунок 1.22 — Меню редактирования функционала чат-бота

Пользователь может изменять функционал чат-бота, используя 4 панели: «Редактор команд», «Редактор кнопок», «Редактор вопросов» и «Редактор ответов», а также панель «Управление чат-ботом» для запуска/остановки чат-бота и его удаления. Попробуем создать тематического чат-бота «Консультант автомобильного салона». Для начала в панели «Конструктор команд» необходимо добавить описание начальной команды для чат-бота в *Telegram* – команды «/start». На данном этапе для команды можно добавить текст сообщения, изображение или файл, которые будет отправляться в ответ на команду. Для добавления текста сообщения необходимо вписать его в соответствующее текстовое поле. Для добавления изображения или файла необходимо включить чекбокс «Добавить файл» и нажать на кнопку с тремя

точками правее, после чего добавить требуемое изображение или файл. Процесс добавления новой команды показан на рисунке 1.23.

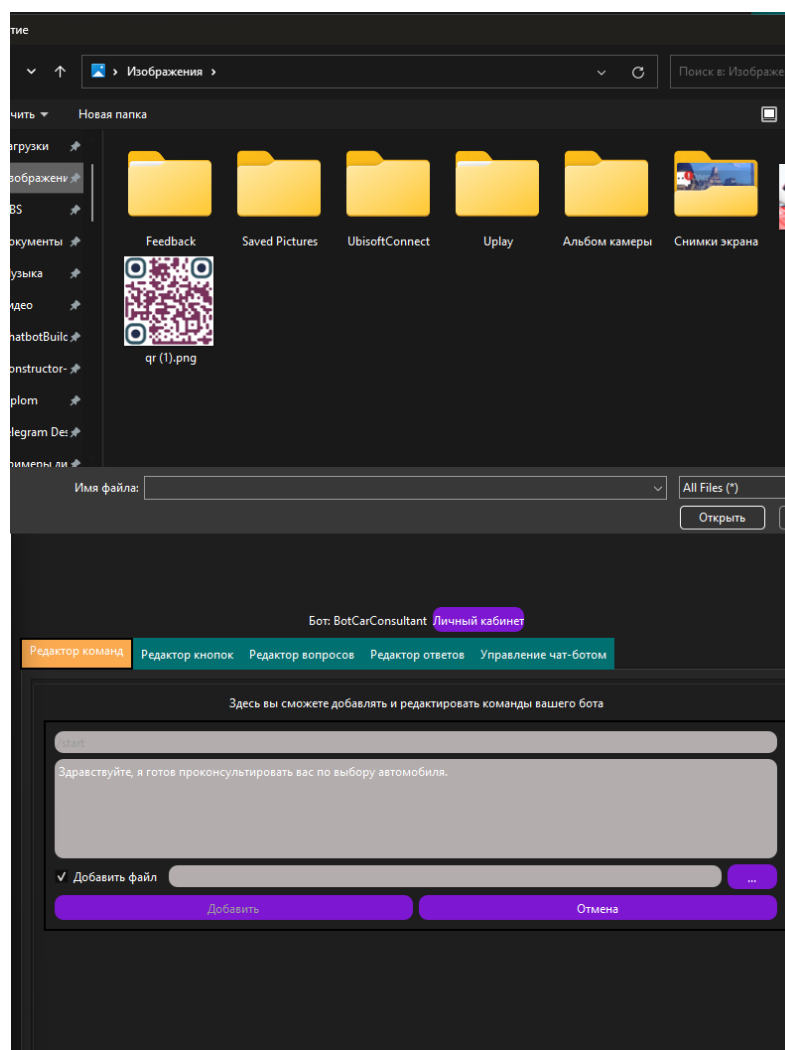


Рисунок 1.23 — Добавление команды

В следующей вкладке «Редактор кнопок» пользователь может создавать собственные профили *Inline*-клавиатур с их текстовым содержанием для возможного их использования в дальнейшей настройке ответов на сообщения. Для настройки необходимо создать новый профиль, нажав на кнопку «Добавить меню кнопок» выбрать количество кнопок для данной клавиатуры, используя счетчик, затем ввести название меню, текстовое содержание кнопок в соответствующие текстовые поля и нажать на кнопку «Добавить». Процесс создания *Inline*-клавиатур можно увидеть на рисунке 1.24.

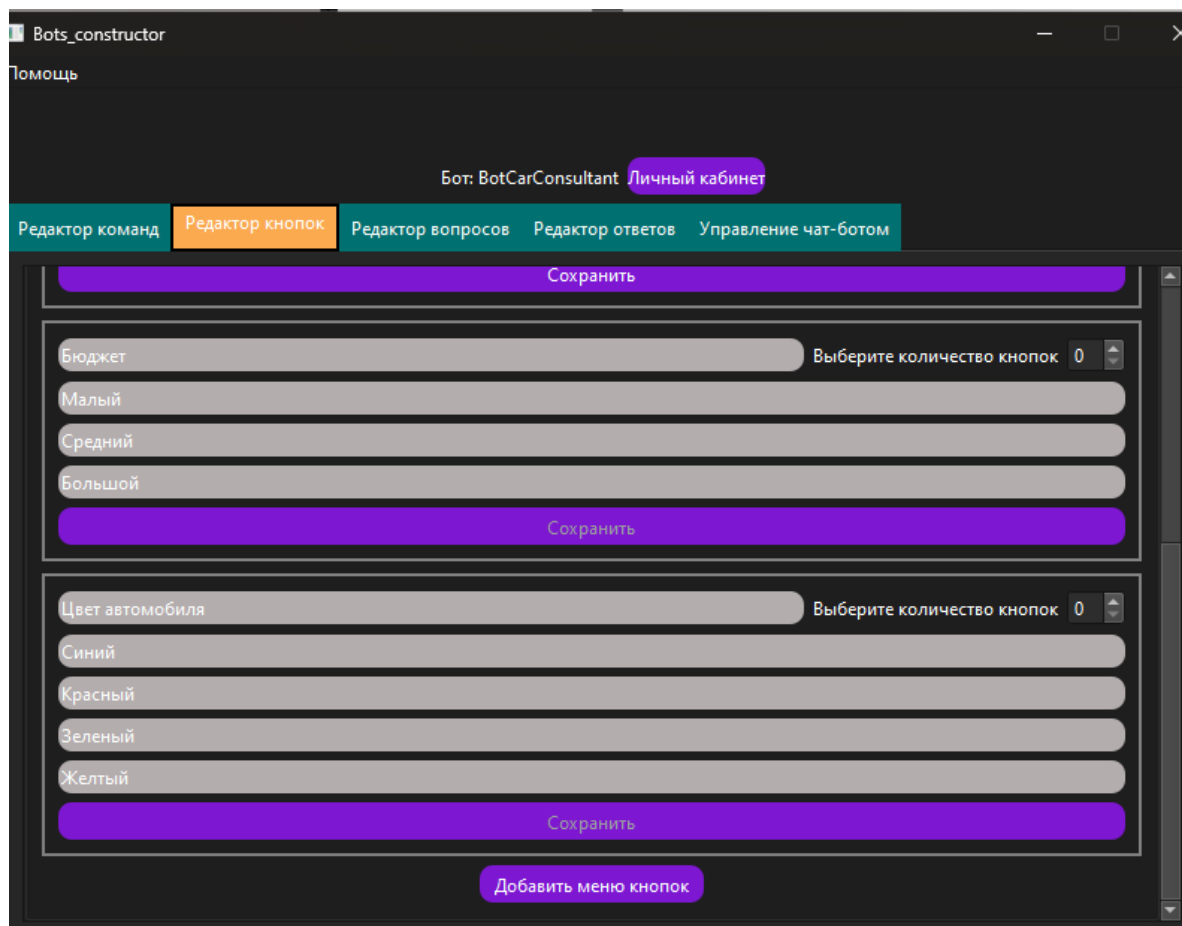


Рисунок 1.24 — Добавление профилей *Inline*-клавиатур

Во вкладке «Редактор вопросов» пользователю необходимо создать ряд вопросов, которые последовательно будут задаваться пользователю чат-бота. Для этого нужно нажать на кнопку «Добавить команду», затем вписать текст сообщения в соответствующее текстовое поле, добавить изображение или файл, нажав на кнопку с тремя точками, в выпадающем меню выбрать один из ранее созданных профилей кнопок и нажать на кнопку «Добавить». Процесс добавления списка вопросов показан на рисунке 1.25.

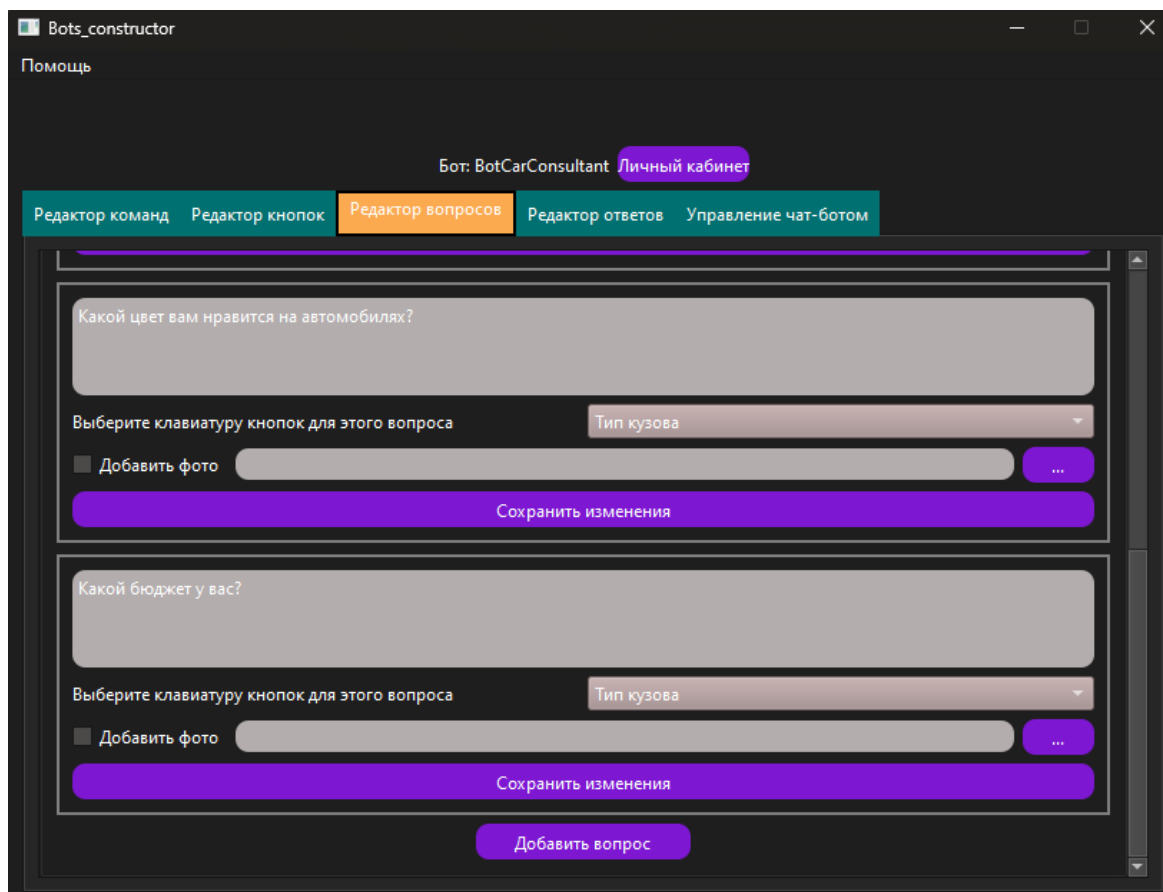


Рисунок 1.25 — Добавление списка вопросов

После создания команд, профилей *Inline*-клавиатур и списка вопросов необходимо создать конечные ответы бота, для этого воспользуемся панелью «Редактор ответов». Здесь пользователю необходимо задать ответы на раннее заданные вопросы. Для каждого вопроса необходимо выбрать в выпадающем меню предполагаемые ответы пользователя, добавить текстовое сообщение и файл или изображение, как это было описано ранее. Если пользователь не добавит ответ для одной из комбинаций вопросов чат-бот выведет сообщение о том, что не знает ответа. Процесс добавления конечных ответов чат-бота показан на рисунке 1.26.

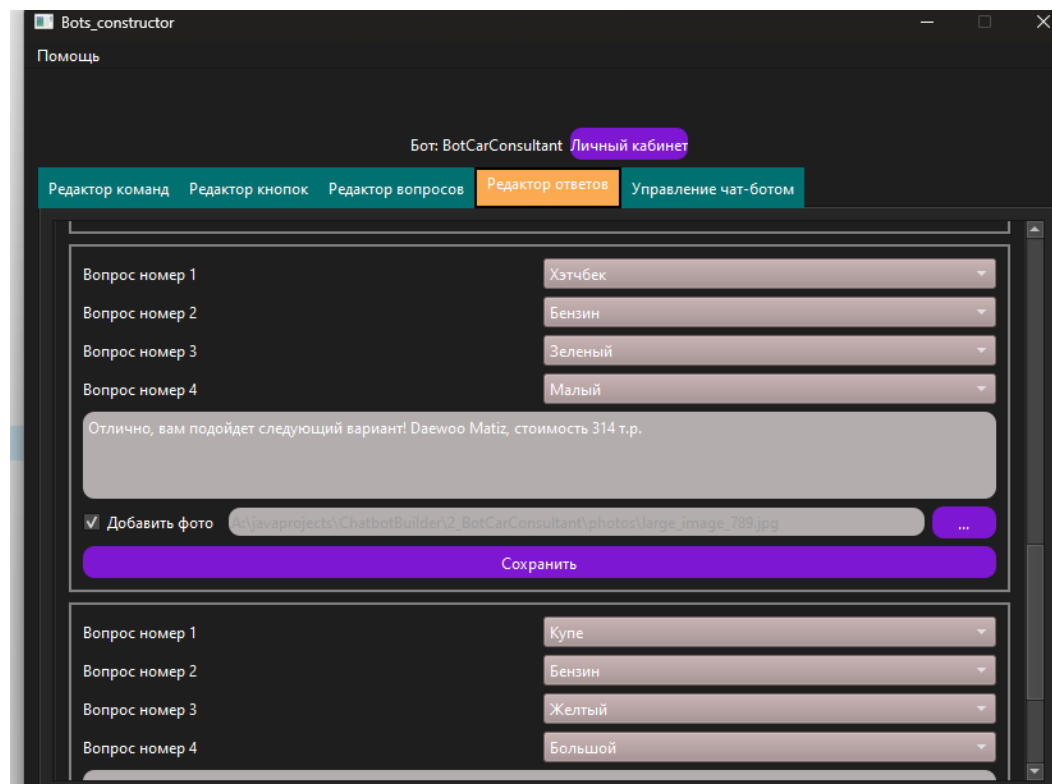


Рисунок 1.26 — Добавление ответов

Для запуска или остановки работы чат-бота необходимо перейти в панель «Управление чат-ботом» и нажать на соответствующую кнопку. Панель управления чат-ботом показана на рисунке 1.27.

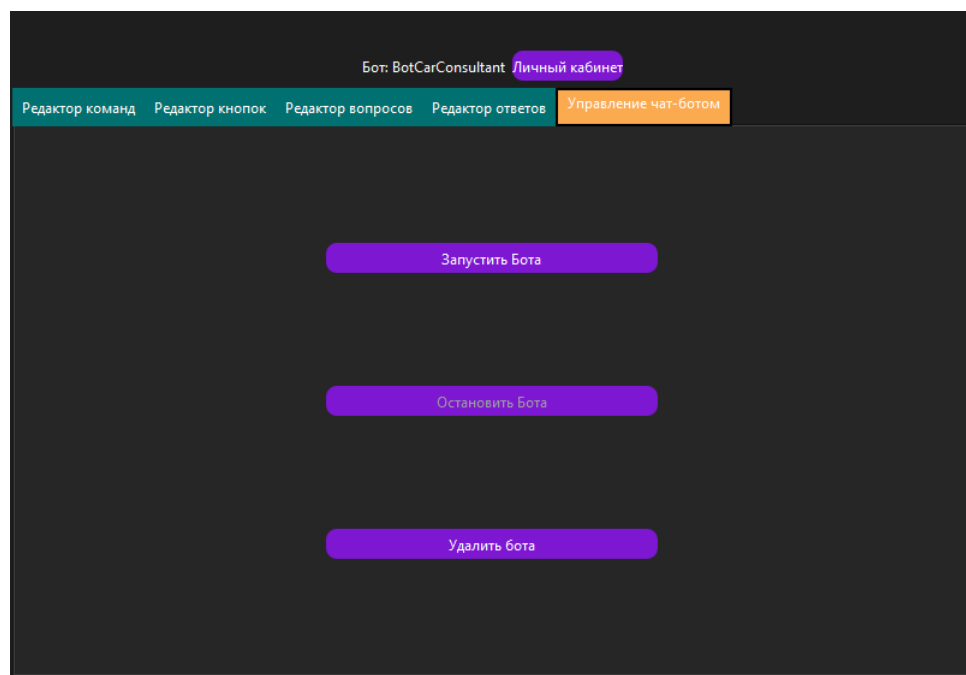


Рисунок 1.27 — Панель управления чат-ботом

Пользователь может удалять добавленных чат-ботов. Для этого необходимо в панели «Управление чат-ботом» нажать на кнопку «Удалить бота», затем подтвердить удаление, нажав на кнопку «Удалить». Процесс удаления чат-бота продемонстрирован на рисунке 1.28.

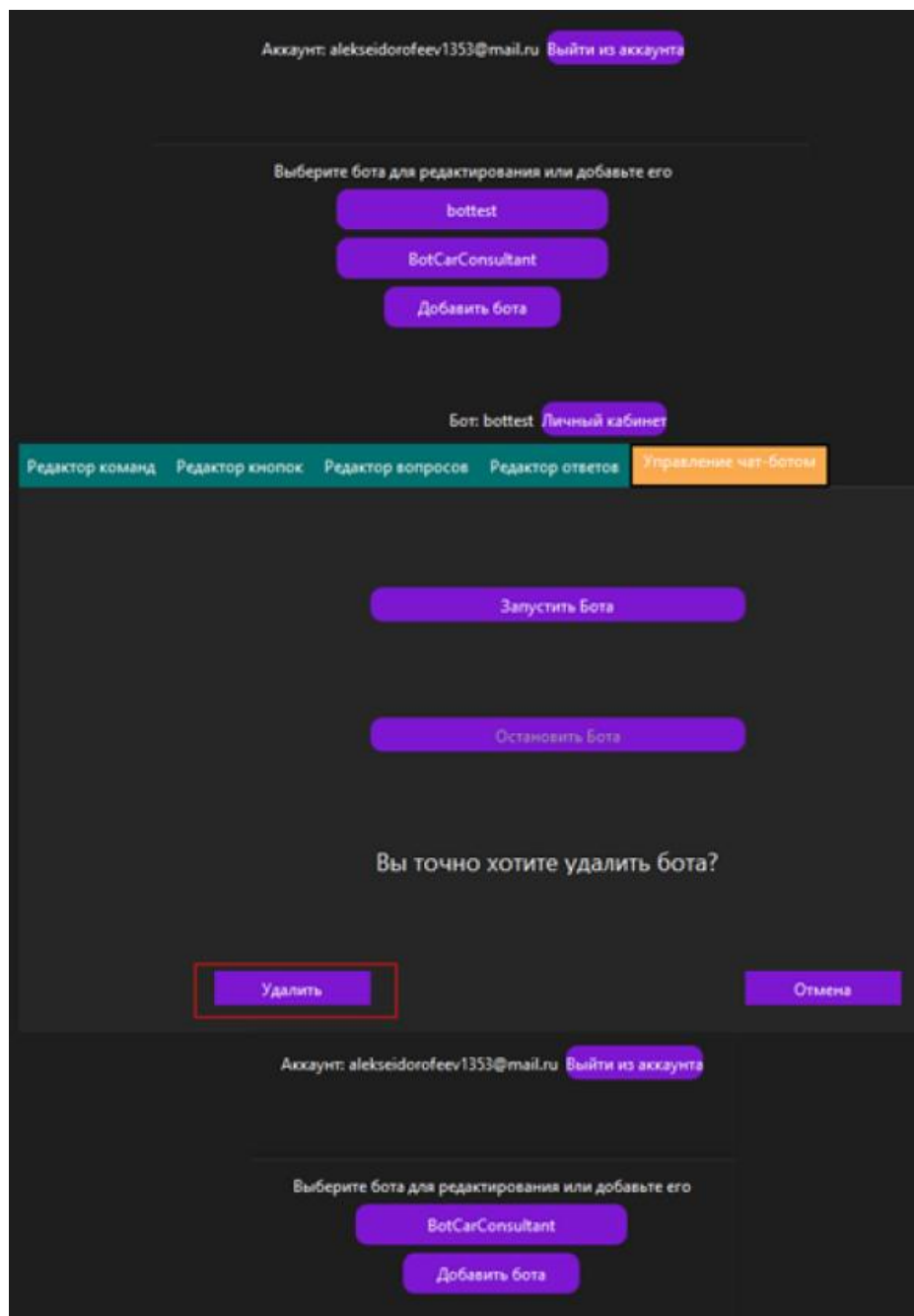


Рисунок 1.28 — Удаление чат-бота

Каждый пользователь может создать не более 5 чат-ботов. При преодолении данных ограничений, пользователя оповестит ошибка, которую можно увидеть на рисунке 1.29.

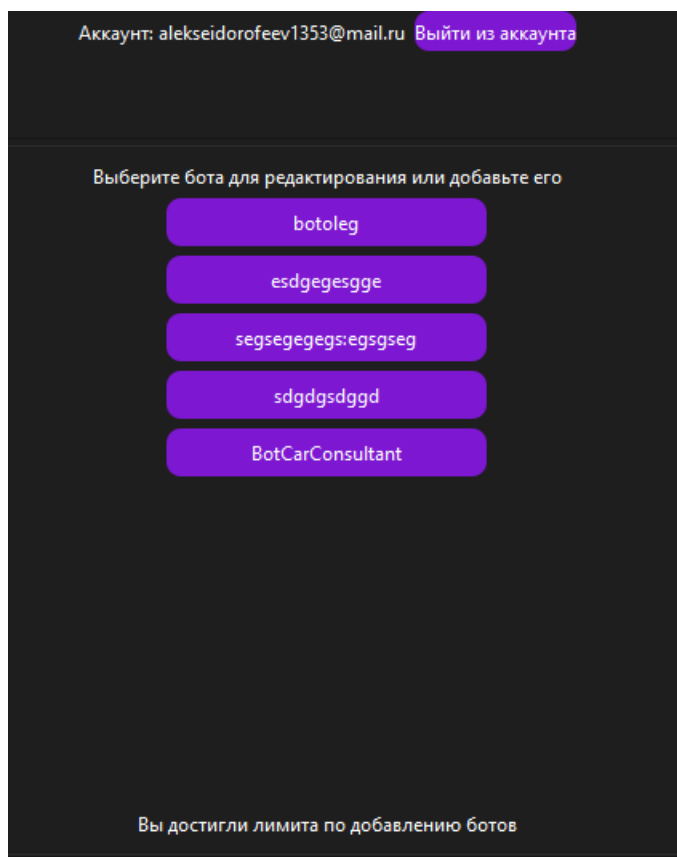


Рисунок 1.29 — Сообщение об ограничении по созданию чат-ботов

Проверим правильность работы полученного чат-бота, попробовав пообщаться с ботом в мессенджере. При прохождении опроса можно увидеть ранее заданные в конструкторе *Inline*-кнопки, команды, вопросы и ответы чат-бота. Результат показан на рисунке 1.30.

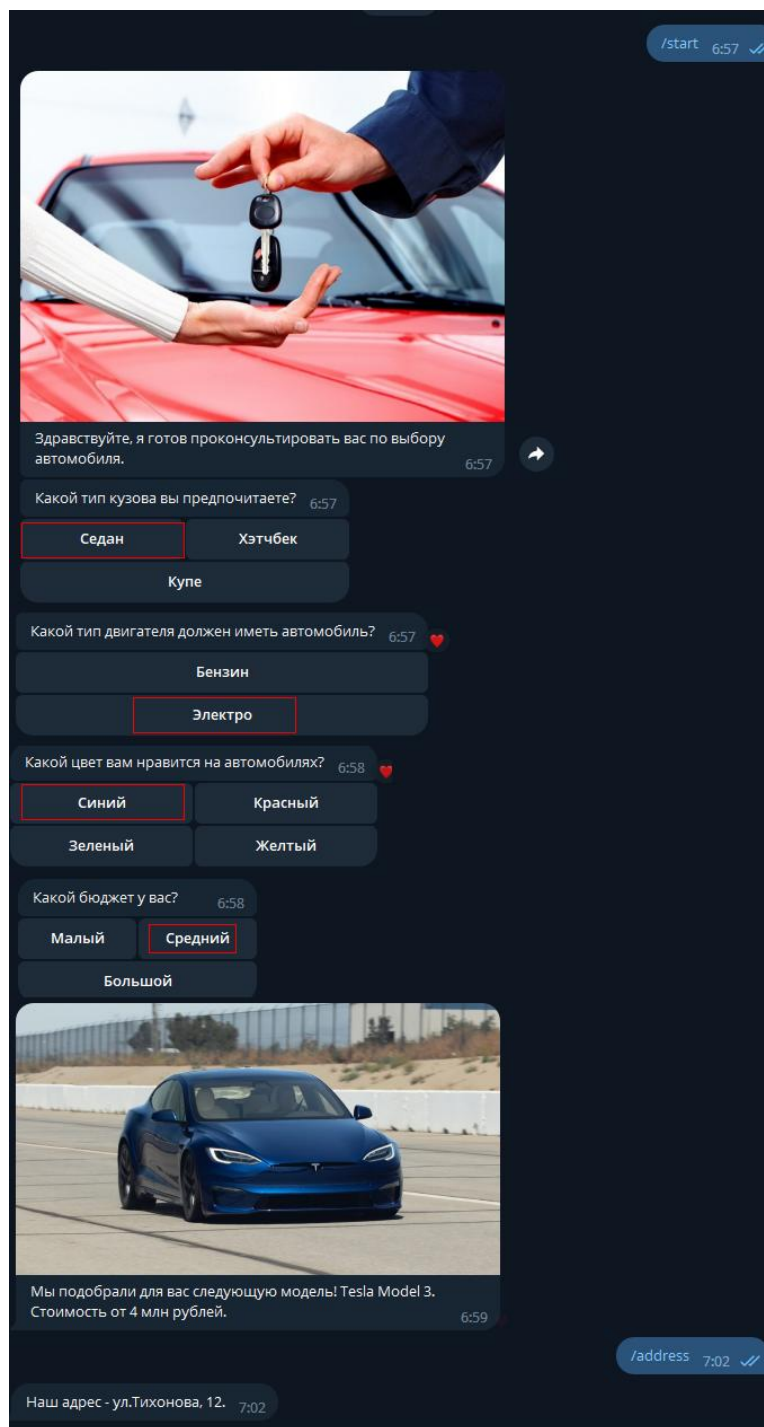


Рисунок 1.30 — Ответ созданного чат-бота

После создания команд или клавиатур пользователь может изменить их содержимое, вписав актуальный текст в поля или изменить изображение и нажав на кнопку «Сохранить». Процесс изменения изображения для стартовой команды и текстового содержания одного из вопросов и конечный ответ чат-бота показан на рисунках 1.31, 1.32.

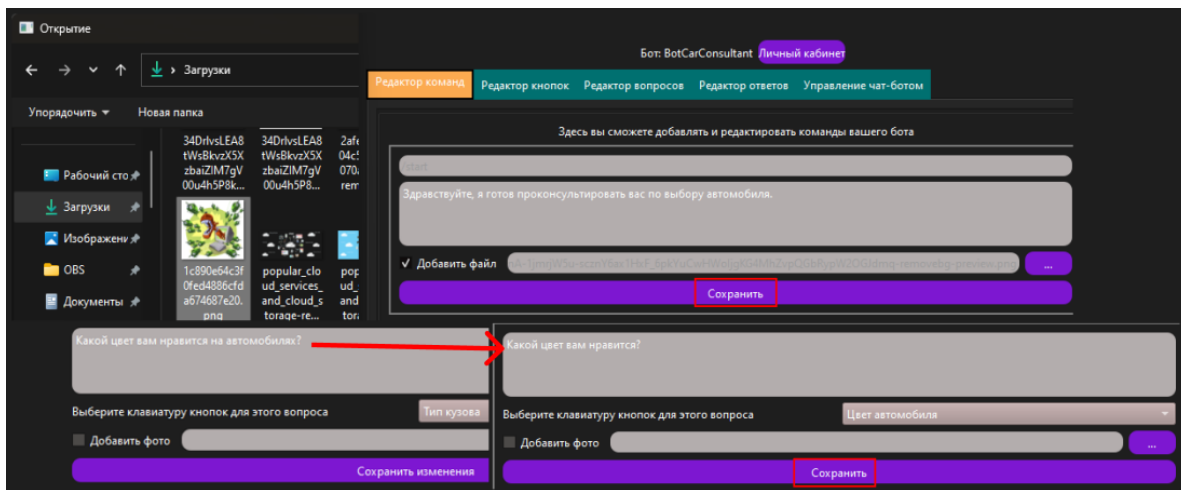


Рисунок 1.31 — Изменение содержимого команд и вопросов

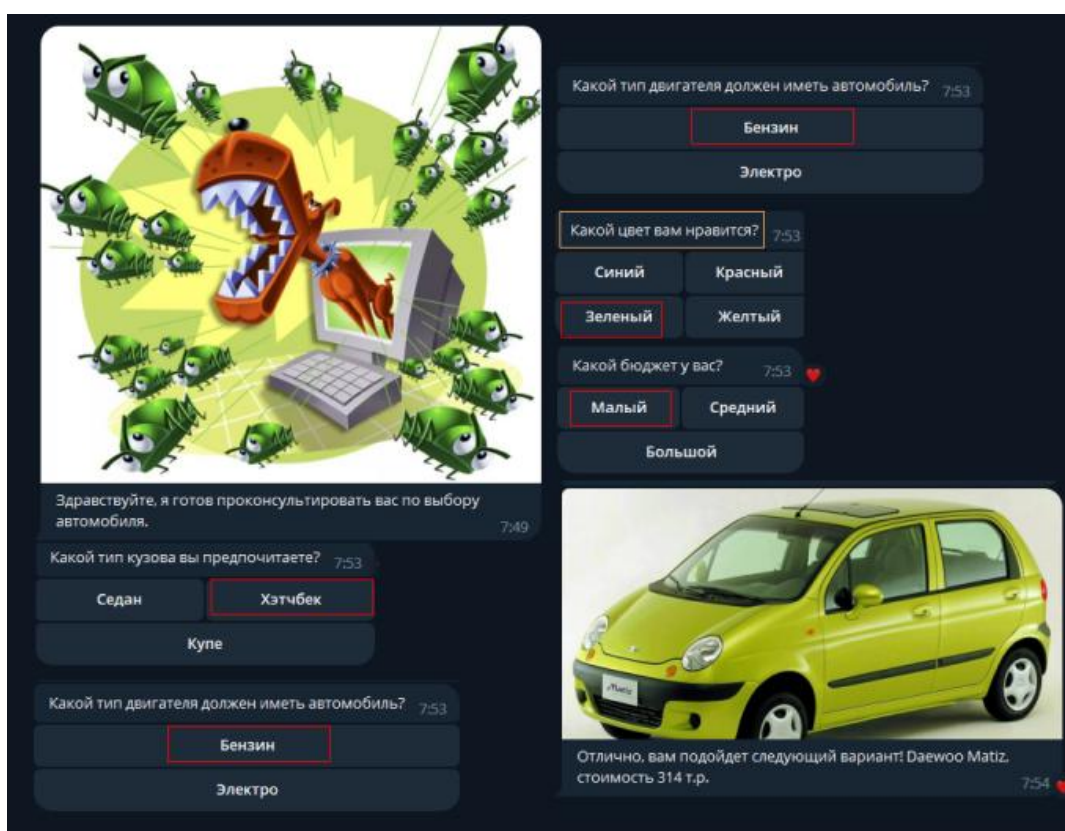


Рисунок 1.32 — Ответ чат-бота после внесенных изменений

При завершении работы по созданию, редактированию и управлению чат-ботами пользователь может выйти из учетной записи, в личном кабинете необходимо нажать на кнопку «Выйти из аккаунта». Пользователя перенесет на окно авторизации.

Изм.	Лист	№ докум.	Подпись	Дата

1.7.4 Описание программы

В процессе разработки программы, происходило изучение материала, указанного в [7], [8].

На рисунке 1.33 продемонстрирована диаграмма классов проекта.

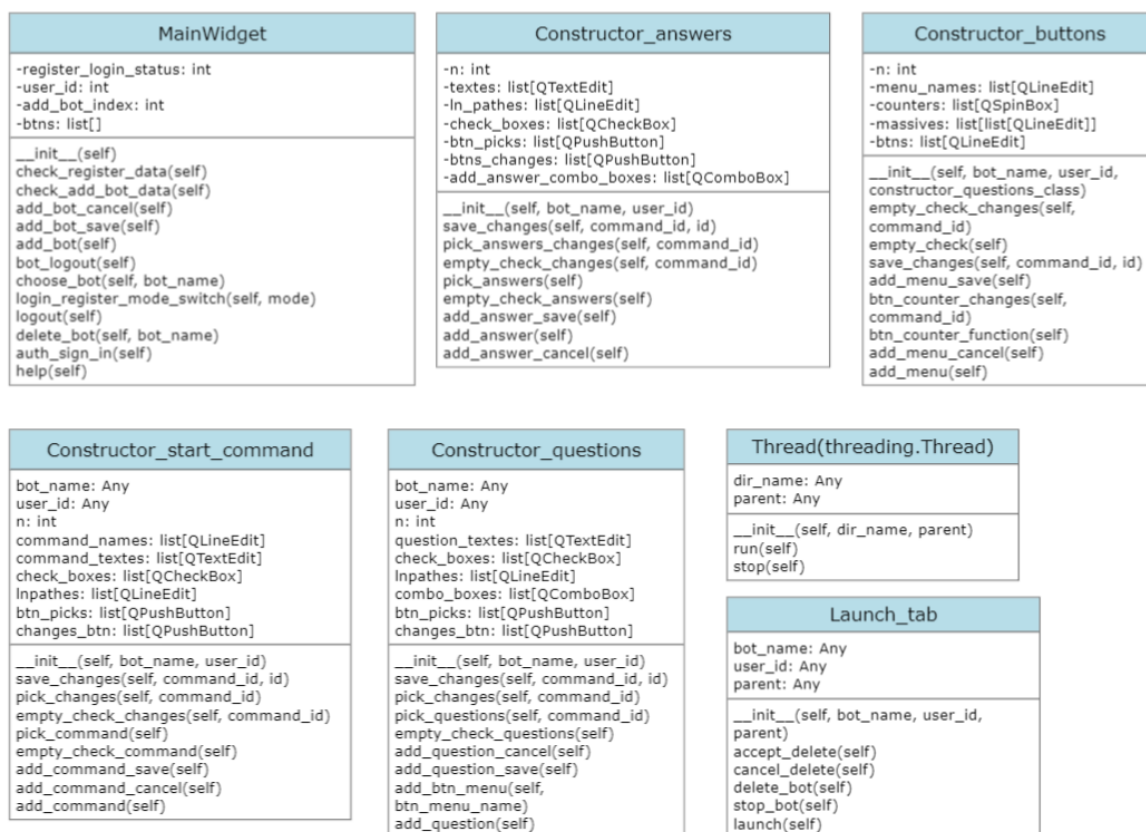


Рисунок 1.33 — Диаграмма классов

Для разработки графического интерфейса приложения применялась библиотека *PyQt6*, использующая инструментарий фреймворка *Qt*. В работе были задействованы следующие элементы данной библиотеки:

- *QLabel*, текстовая метка, которая может быть использована для отображения текста или изображений;
- *QPushButton*, кнопка, которая может быть использована для запуска определенного действия, вызова другого окна или перехода по ссылке;
- *QLineEdit*, однострочное поле ввода текста, позволяющее пользователю вводить текстовые данные;

– *QComboBox*, выпадающее меню, которое позволяет пользователю выбрать один вариант из нескольких предложенных, таких, как профили *Inline*-клавиатур или вариантов ответов пользователя;

– *QCheckBox*, флажок, который позволяет пользователю включить или выключить определенную функцию или настройку, в нашем случае используется для инициации добавления файла к сообщению или команде;

– *QSpinBox*, поле ввода целых чисел, которое позволяет пользователю вводить количество кнопок для *Inline*-клавиатуры;

– *QMenu*, класс, используемый для работы с контекстным меню, в нашем случае для вызова меню «Справка»;

– *QMessageBox*, предоставляет модальное окно для вывода справочной информации на экран.

Далее перечислены модули, которые разделены для каждого подменю интерфейса и включают в себя необходимые функции.

Модуль *dbacc.py* — содержит данные, необходимые для подключения к СУБД.

Модуль *bots_constructor.py* — исполнительный файл интерфейса, который написанный с помощью библиотеки *PyQt6*. Рассмотрим функции, содержащиеся в модуле.

Функция *login_register_mode_switch* — отображает окно авторизации или регистрации.

Функция *check_register_data* — отвечает за проверку содержимого текста полей для регистрации на соответствие требованиям:

- валидность адреса электронной почты;
- длина пароля не меньше 6 символов;
- пароли в основном поле и поле для подтверждения должны совпадать.

Функция *auth_sign_in* — в зависимости от режима авторизует в личный кабинет или заносит учетные данные пользователя в БД при регистрации.

Функция *logout* — обеспечивает выход пользователя из личного кабинета и переносит в режим входа/регистрации.

Функция *help* — выполняет отображение всплывающего окна с информацией о том, как создать *Telegram*-бота с помощью *BotFather*.

Функция *add_bot* — отображает группу виджетов (полей и кнопок) для заполнения данных бота и добавления его в БД.

Функция *add_bot_cancel* — скрывает группу виджетов для добавления бота.

Функция *check_add_bot_data* — проверяет данные полей для добавления бота:

- имена создаваемых чат-ботов должны быть уникальными;
- *API*-токен бота должен соответствовать требованиям.

Функция *add_bot_save* — сохраняет данные бота в БД, сериализует его данные в отдельную папку и добавляет кнопку с именем бота в личный кабинет, для дальнейшего выбора бота.

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

with conn.cursor() as cur:

    try:
        dirname = os.path.dirname(os.path.abspath(__file__))
        dir_name = os.path.join(dirname,
f'{self.user_id}_{self.add_bot_name_ln.text()}')
        os.mkdir(dir_name)
        os.mkdir(os.path.join(dir_name, 'photos'))

        shutil.copy(os.path.join(dirname, 'main.py'), dir_name)
        shutil.copy(os.path.join(dirname, 'buttons.py'), dir_name)
    except:
        pass

    cur.execute("""INSERT INTO bots (user_id, name, token) VALUES
(%s,%s,%s)""",
                (self.user_id, self.add_bot_name_ln.text(),
self.add_bot_token_ln.text()))

    conn.commit()

conn.close()
```

```

self.add_bot_btn.show()

self.add_bot_save_btn.hide()
self.add_bot_cancel_btn.hide()
self.add_bot_name_ln.hide()
self.add_bot_token_ln.hide()
self.add_bot_status_lb.hide()

hl = QtWidgets.QHBoxLayout()
btn = QtWidgets.QPushButton(self.add_bot_name_ln.text())
self.bots_btn.append(btn)
btn.clicked.connect(lambda checked,
bot_name=self.add_bot_name_ln.text(): self.choose_bot(bot_name))
btn.setMaximumWidth(200)
btn.setMinimumHeight(30)
hl.addWidget(btn)
self.bots.append(btn)
self.bots_list.insertLayout(self.bots_list.count() -
self.add_bot_index, hl)

self.add_bot_index += 1

```

Функция *choose_bot* — переносит пользователя в режим редактирования бота, который состоит из 5 вкладок, за каждую из которых отвечает отдельный класс.

Функция *bot_logout* — выходит из режима редактирования бота в режим личного кабинета.

Модуль *constructor_comands.py* — хранит в себе класс *Constructor_comands*, отвечающий за создание и редактирование команд бота. Рассмотрим функции, содержащиеся в данном классе.

Функция *add_comand* — отображает группу виджетов, которые отображают новые команды. Для добавления команды необходимо внести следующие данные:

- название команды;
- текст, который будет отправляться по команде;
- фото или файл, который будет отправляться по команде.

Функция *add_comand_cancel* — скрывает группу виджетов, предназначенных для добавления команды.

Функция *empty_check* — проверяет текстовые поля для добавления команды на валидность:

- название команды начинается с символа «/» и должна состоять только из английских букв;
- поле, содержащее за текст команды не должно быть пустым;
- если стоит галочка в чекбоксе на добавление фото, то поле, содержащее путь к фото не должно быть пустым.

Функция *pick* — вызывает всплывающее окно для выбора файла и записывает путь к нему.

Функция *add_comand_save* — добавляет команду в БД и выводит на главный экран с возможностью редактирования.

Функция *empty_check_changes* — выполняет те же функции, что и команда *empty_check*, только для уже добавленных элементов.

Функция *pick_changes* — выполняет тот же функционал, что и команда *pick*, только для уже добавленных элементов.

Функция *save_changes* — сохраняет и перезаписывает новые значение для команд.

Функция *delete_bot* — удаляет чат-бота из личного кабинета.

Модуль *constructor_buttons.py* — хранит в себе класс *Constructor_buttons*, отвечающий за создание и редактирование меню кнопок для вопросов.

Функция *add_menu* — отображает группу виджетов, используемых для добавления новых меню кнопок. Для добавления меню кнопок необходимо внести следующие данные:

- название меню;
- количество кнопок в меню;
- значение количества кнопок для данного профиля клавиатуры.

Функция *add_menu_cancel* — скрывает группу виджетов для добавления меню кнопок.

Функция *empty_check* — проверяет текстовое содержание и счетчик меню кнопок на корректность:

- название меню кнопок не должно быть пустым;
- значение кнопок не должны быть пустыми.

Функция *btn_counter_function* — при изменении значения количества кнопок меню добавляет/убирает поля необходимые для заполнения значений кнопок.

Функция *add_menu_save* — добавляет меню кнопок в БД и выводит на главный экран с возможностью редактирования.

Функция *empty_check_changes* — выполняет те же действия, что и функция *empty_check*, только для уже добавленных элементов.

Функция *btn_counter_changes* — отвечает за то же самое что и функция *btn_counter_function*, только для уже добавленных элементов.

Команда *save_changes* — сохраняет и перезаписывает новые значение для меню кнопок.

Модуль *constructor_questions.py* — хранит в себе класс *Constructor_questions* отвечающий за создание и редактирование вопросов, на которые будет отвечать пользователь в боте. Далее описаны функции, входящие в этот класс.

Функция *add_question* — отображает группу виджетов для добавление новых вопросов. Для добавления вопроса необходимо внести следующие данные:

- текст вопроса;
- клавиатура кнопок, которая будет отправляться с вопросом;
- фото или файл, которое будет отправляться с вопросом.

Функция *add_question_cancel* — скрывает группу виджетов для добавления вопроса.

Функция *empty_check* — проверяет поля для добавления вопроса на правильность:

- поле отвечающие за текст вопроса не должно быть пустым;
- если стоит галочка в чекбоксе на добавление фото, то поле, содержащее путь к фото не должно быть пустым.

Функция *pick* — вызывает всплывающее окно для выбора файла и записывает путь к нему.

Функция *add_question_save* — добавляет вопрос в БД и выводит на главный экран с возможностью редактирования.

```
def add_question_save(self):
    conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
    with conn.cursor() as cur:
        if self.cb.isChecked():
            dirname = os.path.dirname(os.path.abspath(__file__))
            dir_name = os.path.join(dirname,
f'{self.user_id}_{self.bot_name}')
            dir_name = os.path.join(dir_name, 'photos')

            shutil.copy(self.lnm.text(), dir_name)
            save_path = os.path.join(dir_name,
self.lnm.text().split('/')[ -1])
            cur.execute("""SELECT MAX(question_number) FROM
questions WHERE user_id=%s AND bot_name=%s""",
                        (self.user_id, self.bot_name))
            max_number = cur.fetchone()
            max_number = max_number[0]
            if max_number is None:
                max_number = 1
            else:
                max_number += 1

            cur.execute(
                f"""INSERT INTO questions (user_id, bot_name,
question_number, question_text, btn_menu, question_photo) VALUES
(%s,%s,%s,%s,%s,%s)""",
                (self.user_id, self.bot_name, max_number,
self.tem.toPlainText(), self.combo_box.currentText(), save_path))
            conn.commit()
        else:
            cur.execute("""SELECT MAX(question_number) FROM
questions WHERE user_id=%s AND bot_name=%s""",
                        (self.user_id, self.bot_name))
            max_number = cur.fetchone()
            max_number = max_number[0]
            if max_number is None:
                max_number = 1
            else:
```

```

        max_number += 1

        cur.execute(
            f"""INSERT INTO questions (user_id, bot_name,
question_number, question_text, btn_menu) VALUES
(%s,%s,%s,%s,%s)""",
            (self.user_id, self.bot_name, max_number,
self.tem.toPlainText(), self.combo_box.currentText()))
        conn.commit()

    conn.close()
    groupBox = QtWidgets.QGroupBox()
    groupBox.setMaximumWidth(940)
    groupBox.setStyleSheet(''...'')

    cb = QtWidgets.QCheckBox('Добавить фото')
    self.check_boxes.append(cb)
    lnm = QtWidgets.QLineEdit()
    self.lnpathes.append(lnm)
    lnm.setEnabled(False)
    btnm1 = QtWidgets.QPushButton('...')
    btnm1.setMinimumWidth(50)
    self.btn_picks.append(btnm1)

    if self.cb.isChecked():
        cb.setChecked(True)
        lnm.setText(self.lnm.text())

    cb.stateChanged.connect(lambda checked, command_id=self.n:
self.empty_check_cganges(command_id))
    lnm.textChanged.connect(lambda checked, command_id=self.n:
self.empty_check_cganges(command_id))
    btnm1.clicked.connect(lambda checked, command_id=self.n:
self.pick_changes(command_id))

    tem = QtWidgets.QTextEdit(self.tem.toPlainText())
    tem.textChanged.connect(lambda command_id=self.n:
self.empty_check_cganges(command_id))
    tem.setPlaceholderText('Введите текст вопроса')
    tem.setMaximumHeight(90)
    self.question_textes.append(tem)

    hlm = QtWidgets.QHBoxLayout()
    hlm.addWidget(cb)
    hlm.addWidget(lnm)
    hlm.addWidget(btnm1)

    combo_box_lb = QtWidgets.QLabel('Выберите клавиатуру кнопок для
этого вопроса')
    combo_box = QtWidgets.QComboBox()
    self.combo_boxes.append(combo_box)
    conn = psycopg2.connect(database=data_base, user=user,

```

```

password=password, host=host, port=port)

    with conn.cursor() as cur:
        cur.execute("""SELECT menu_name FROM buttons WHERE
user_id=%s AND bot_name=%s""",
                    (self.user_id, self.bot_name))
        btn_menus = cur.fetchall()

        for btn_menu2 in btn_menus:
            combo_box.addItem(btn_menu2[0])
    conn.close()

    combo_box_hl = QtWidgets.QHBoxLayout()
    combo_box_hl.addWidget(combo_box_lb)
    combo_box_hl.addWidget(combo_box)
    conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)

    with conn.cursor() as cur:

        cur.execute("""SELECT MAX(id) FROM questions WHERE
user_id=%s AND bot_name=%s""",
                    (self.user_id, self.bot_name))
        max_number = cur.fetchone()
        max_number = max_number[0]
    conn.close()

    btnm3 = QtWidgets.QPushButton('Сохранить изменения')
    btnm3.clicked.connect(lambda checked, command_id=self.n, id=id:
self.save_changes(command_id, max_number))
    self.changes_btn.append(btnm3)
    add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
    add_comand_btn_hl2.addWidget(btnm3)

    add_menu_vl = QtWidgets.QVBoxLayout(groupBox)
    add_menu_vl.addWidget(tem)
    add_menu_vl.addLayout(combo_box_hl)
    add_menu_vl.addLayout(hlm)
    add_menu_vl.addLayout(add_comand_btn_hl2)

    self.bots_list.insertWidget(self.bots_list.count()-3, groupBox)

    self.groupBox.hide()
    self.add_menu_btn.show()

```

Функция *empty_check_changes* — работает также, как и *empty_check*, только для уже добавленных элементов.

Функция *pick_changes* — выполняет тот же функционал, что и *pick*, только для уже добавленных элементов.

Функция *save_changes* — сохраняет и перезаписывает новые значения для команд.

Модуль *constructor_answers.py* — хранит в себе класс *Constructor_answers*, отвечающий за создание и редактирование того, какой ответ получит пользователь при выборе определенной комбинации ответов.

Функция *add_answer* — отображает группу виджетов для добавления новых вариантов ответа. Для добавления варианта ответа необходимо внести следующие данные:

- текст ответа;
- комбинацию ответов на вопросы;
- фото или файл, который будет отправляться с ответом.

Функция *add_answer_cancel* — скрывает группу виджетов, отвечающих за добавление варианта ответа.

Функция *empty_check* — проверяет поля для добавления варианта ответа на корректность:

- поле отвечающие за текст вопроса не должно быть пустым;
- если стоит галочка в чекбоксе на добавление фото, то поле, содержащее путь к фото не должно быть пустым.

Функция *pick* — вызывает всплывающее окно для выбора файла и записывает путь к нему.

Функция *add_answer_save* — добавляет ответ в БД и выводит на главный экран с возможностью редактирования.

Функция *empty_check_changes* — выполняет те же действия, что и функция *empty_check*, только для уже добавленных элементов.

Функция *pick_changes* — отвечает за то же самое что и функция *pick*, только для уже добавленных элементов.

Функция *save_changes* — сохраняет и перезаписывает новые значение для варианта ответа.

Модуль *launch_tab.py* — хранит в себе класс *Launch_tub*, отвечающий за запуск, остановку и удаления бота, а также класс *Thread* – являющийся потоком для запуска бота.

Функция *launch* — запускает поток *Thread* внутри которого запускается чат-бот.

Функция *stop_bot* — останавливает бота.

Функция *delete_bot* — выводит кнопки для подтверждения удаления бота из базы данных.

Функция *accept_delete* — вызывает функцию *delete_bot* класса *Bots_constructor* и удаляет бота из БД.

Функция *cancel_delete* — скрывает кнопки для подтверждения удаления.

Файл *main.py* — данный файл, файл *buttons.py* и папка для хранения файлов/фото, которые прикрепит пользователь к сообщениям или командам сериализуются в новую создаваемую для каждого созданного чат-бота папку. *main.py* является исполнительным файлом, который запускает созданного бота.

Функция *query_handler* — отвечает за обработку событий нажатия *Inline*-кнопок, фиксирует какой вариант ответа выбрал пользователь в ответ на вопрос и отправляет следующий вопрос.

```
def queryHandler(update, context: CallbackContext):
    query = update.callback_query
    data = query.data
    text = query.message.text
    id = query.message.chat_id
    m_id = query.message.message_id
    str_id = str(id)
    context.bot.delete_message(chat_id=id, message_id=m_id)

    conn = psycopg2.connect(database=data_base, user=user,
        password=password, host=host, port=port)
    with conn.cursor() as cur:
        cur.execute("""SELECT answears FROM bots_users WHERE
        bot_owner_id=%s AND bot_name=%s AND user_id=%s""", (bot_owner_id,
        bot_name, str(id)))
        answears = cur.fetchone()
        answears = answears[0]
        if answears is None:
            cur.execute("""UPDATE bots_users SET answears=%s WHERE
        bot_owner_id=%s AND bot_name=%s AND user_id=%s""", (data+'|',
```

```

bot_owner_id, bot_name, str(id))
    else:
        cur.execute("""UPDATE bots_users SET answers=%s WHERE
bot_owner_id=%s AND bot_name=%s AND user_id=%s""",
                    (answers+data + '|', bot_owner_id,
bot_name, str(id)))

        cur.execute("""UPDATE bots_users SET
current_question=current_question+1 WHERE bot_owner_id=%s AND
bot_name=%s AND user_id=%s""", (bot_owner_id, bot_name, str(id)))
        conn.commit()
        cur.execute("""SELECT current_question FROM bots_users WHERE
bot_owner_id=%s AND bot_name=%s AND user_id=%s""",
                    (bot_owner_id, bot_name, str(id)))
        curent_question = cur.fetchone()
        curent_question = curent_question[0]

    try:
        cur.execute("""SELECT question_text FROM questions WHERE
user_id=%s AND bot_name=%s AND question_number=%s""",
                    (bot_owner_id, bot_name, curent_question))
        question_text = cur.fetchone()
        question_text = question_text[0]
        cur.execute(
            """SELECT btn_menu FROM questions WHERE user_id=%s
AND bot_name=%s AND question_number=%s""",
            (bot_owner_id, bot_name, curent_question))
        btn_menu = cur.fetchone()
        btn_menu = btn_menu[0]
        cur.execute(
            """SELECT question_photo FROM questions WHERE
user_id=%s AND bot_name=%s AND question_number=%s""",
            (bot_owner_id, bot_name, curent_question))
        question_photo = cur.fetchone()
        question_photo = question_photo[0]
        if question_photo is not None:
            context.bot.send_photo(chat_id=id,
caption=question_text, photo=open(question_photo, 'rb'),
reply_markup=get_keyboard(btn_menu))
        else:
            context.bot.send_message(chat_id=id,
text=question_text, reply_markup=get_keyboard(btn_menu))
    except:
        cur.execute("""SELECT answers FROM bots_users WHERE
bot_owner_id=%s AND bot_name=%s AND user_id=%s""",
                    (bot_owner_id, bot_name, str_id))
        answers = cur.fetchone()
        answers = answers[0]
        cur.execute("""SELECT result, text, photo FROM answers
WHERE user_id=%s AND bot_name=%s""", (bot_owner_id, bot_name))
        for result, text, photo in cur.fetchall():
            if answers == result:

```

```

        if photo is not None:
            context.bot.send_photo(chat_id=id,
caption=text, photo=open(photo, 'rb'))
            return False
        else:
            context.bot.send_message(chat_id=id,
text=text)
            return False
        context.bot.send_message(chat_id=id, text="Я не знаю
ответа на вопрос :(")
        cur.execute(
            """UPDATE bots_users SET current_question=1,
answears=NULL WHERE bot_owner_id=%s AND bot_name=%s AND
user_id=%s""",
            (bot_owner_id, bot_name, str_id))
        conn.commit()
        conn.close()

conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
with conn.cursor() as cur:
    cur.execute("""SELECT comand_name, comand_text, comand_photo
FROM comands WHERE user_id=%s AND bot_name=%s""",
            (bot_owner_id, bot_name))
    comands = cur.fetchall()
    for comand_name, comand_text, comand_photo in comands:
        def function(update, bot, comand_text=comand_text,
comand_name=comand_name, comand_photo=comand_photo):
            id = update.message.chat_id
            if comand_photo is not None:
                bot.bot.send_photo(chat_id=id, caption=comand_text,
photo=open(comand_photo, 'rb'))
            else:
                bot.bot.send_message(chat_id=id, text=comand_text)

        if comand_name.replace('/', '') == 'start':
            conn = psycopg2.connect(database=data_base,
user=user, password=password, host=host, port=port)
            with conn.cursor() as cur:
                cur.execute("""SELECT user_id FROM bots_users
WHERE bot_owner_id=%s AND bot_name=%s""", (bot_owner_id, bot_name))
                all_ids = []
                for i in cur.fetchall():
                    all_ids.append(i[0])

                if str(id) in all_ids:
                    pass
                else:
                    cur.execute("""INSERT INTO bots_users
(bot_owner_id, bot_name, user_id, current_question) VALUES
(%s,%s,%s,%s)""", (bot_owner_id, bot_name, str(id), 1))
                    conn.commit()

```

```

cur.execute("""SELECT current_question FROM
bots_users WHERE bot_owner_id=%s AND bot_name=%s AND user_id=%s""",
(bot_owner_id, bot_name, str(id)))
current_question = cur.fetchone()
current_question = current_question[0]
try:
cur.execute(
"""SELECT question_text FROM questions
WHERE user_id=%s AND bot_name=%s AND question_number=%s""",
(bot_owner_id, bot_name,
current_question))
question_text = cur.fetchone()
question_text = question_text[0]
cur.execute(
"""SELECT btn_menu FROM questions WHERE
user_id=%s AND bot_name=%s AND question_number=%s""",
(bot_owner_id, bot_name,
current_question))
btn_menu = cur.fetchone()
btn_menu = btn_menu[0]
cur.execute(
"""SELECT question_photo FROM questions
WHERE user_id=%s AND bot_name=%s AND question_number=%s""",
(bot_owner_id, bot_name,
current_question))
question_photo = cur.fetchone()
question_photo = question_photo[0]
if question_photo is not None:
bot.bot.send_photo(chat_id=id,
caption=question_text, photo=open(question_photo, 'rb'),
reply_markup=get_keyboard(btn_menu))
else:
bot.bot.send_message(chat_id=id,
text=question_text,
reply_markup=get_keyboard(btn_menu))
except Exception as e:
bot.bot.send_message(chat_id=id, text="Я не
знаю такой команды :(")
cur.execute("""UPDATE bots_users SET
current_question=1, answears=NULL WHERE bot_owner_id=%s AND
bot_name=%s AND user_id=%s""",
(bot_owner_id, bot_name,
str(id)))
conn.commit()
conn.close()
functions.append(function)

dispatcher.add_handler(CommandHandler(comand_name.replace('/', ''),
functions[functions_counter], run_async=True))

```

```

        functions_counter+=1
conn.close()

```

Диаграмма развертывания созданного чат-бота, продемонстрированная на рисунке 1.34, используется для визуализации местоположения и взаимодействия компонентов системы на уровне аппаратного и ПО. Диаграмма предоставляет возможность представить архитектуру системы, показав, на каких устройствах или серверах расположены компоненты приложения и как они связаны друг с другом. Диаграмма развертывания чат-бота представляет компоненты и связи между ними на одном графическом изображении, что упрощает понимание структуры системы [3].

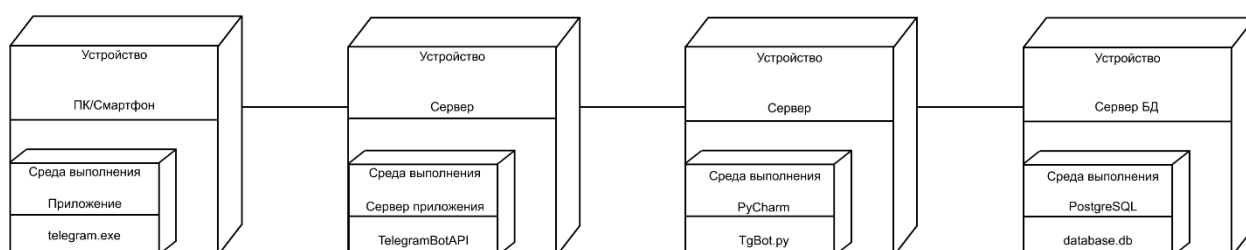


Рисунок 1.34 — Диаграмма развертывания

Файл *buttons.py* — содержит в себе функцию, которая отвечает за создание объекта меню кнопок *InlineKeyboardMarkup*.

Функция *get_menu* — возвращает объект *InlineKeyboardMarkup* в зависимости от того, какое меню клавиатур было указано пользователем в панели редактирования.

1.7.5 Тестирование работы системы

При использовании конструктора пользователи могут вносить в текстовые поля некорректные данные или данные, которые могут нарушить работу программы. Чтобы избежать нарушений в работе программы, необходимо составить набор тест-кейсов по тестированию обработчиков ошибок и корректной работе чат-ботов в асинхронном режиме. Данные тест-кейсы можно увидеть в таблице 8.

Таблица 8 — Тест-кейсы

					ПГУ1.090301.07.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		66

Цель тестирования	Данные	Шаги	Ожидаемый результат
1. Проверка корректности вводимых данных при регистрации.	Логин — <i>login</i> . Пароль — <i>pass</i> .	1. Запустить программу. 2. Нажать на кнопку «Зарегистрироваться». 3. Ввести данные в соответствующие текстовые поля.	Высвечиваются ошибки, текстовые поля становятся красными, кнопка регистрации неактивна.
Цель тестирования	Данные	Шаги	Ожидаемый результат
2. Проверка корректной авторизации.	Логин — <i>test@mail.ru</i> . Пароль — <i>Password</i> .	1. Запустить программу. 2. Нажать на кнопку «Зарегистрироваться». 3. Ввести данные в соответствующие текстовые поля. 4. Нажать на кнопку «Зарегистрироваться». 5. Нажать на кнопку «Войти в аккаунт». 6. Ввести логин или пароль, не соответствующий исходным данным и нажать на кнопку «Войти».	Регистрация нового пользователя прошла успешно. При попытке войти в учетную запись, используя неверные данные, высвечивается ошибка.
3. Проверка невозможности создания двух чат-ботов с одинаковыми именами. Проверка введенного API-токена.	Имя чат-бота – <i>BotName</i> . API-токен, полученный от <i>BotFather</i> .	1. Запустить программу. 2. Войти в систему с учетными данными из пункта 2. 3. Нажать кнопку «Добавить бота». 4. Ввести имя из исходных данных, 5. В поле токена ввести случайный набор символов. 6. Ввести токен из исходных данных. 7. Нажать кнопку добавить бота. 8. Повторить действия подпунктов 4 и 6.	Высветились ошибки о некорректном токене и о создании бота с уже имеющимся названием.

4. Проверка корректности вводимых данных при создании новой команды.	—	1. Запустить программу. 2. Войти в систему с учетными данными из пункта 2. 3. Ввести в текстовое поле названия команды символы, не относящиеся к символам латинского алфавита, не арабские цифры или попытаться начать команду не с символа «/».	Данные символы не проходят проверку и не заносятся в текстовое поле.
Цель тестирования	Данные	Шаги	Ожидаемый результат
5. Проверка работы программы в случае удаления созданного чат-бота в <i>BotFather</i> .	Чат-бот, созданный в пункте 3.	1. Перейти в <i>BotFather</i> и удалить ранее созданного чат-бота. 2. Запустить этого чат-бота в конструкторе.	При запуске высвечивается ошибка некорректного <i>API</i> -токена.

6. Проверка асинхронной работы чат-ботов.	Учётная запись с чат-ботом, созданная в пункте 1.7.3.	1. Создать подобного п. 1.7.3 чат-бота 2. Запустить чат-ботов в конструкторе 2. Зайти в приложение <i>Telegram</i> с разных аккаунтов. 3. Найти чат-ботов и одновременно начать общение.	Корректная работа чат-бота, согласно заданным параметрам в пункте 1.7.3.
---	---	---	--

2 Безопасность жизнедеятельности

2.1 Правила безопасности при использовании ПК

В данном разделе необходимо учесть риски, описанные в ГОСТ 12.0.003-2015 "Система стандартов безопасности труда. Электронно-вычислительная техника. Общие требования безопасности". Далее перечислены риски, которым необходимо уделить особое внимание:

– риск повреждения зрения и усталости глаз:

1) продолжительное использование компьютера без перерыва и отдыха для глаз;

2) неправильная расстановка монитора, отражение света на экране и недостаточное освещение, что может вызывать неприятные чувства в глазах;

– риск получения травм и заболеваний опорно-двигательной системы:

1) неправильное положение тела при работе с ПК: неправильная осанка, поза или расположение рук и ног;

2) отсутствие эргономической мебели, такой как мягкие стулья с подлокотниками и подставки для рук, которые способствуют достигнуть оптимального положения тела при работе;

– риск нарушения информационной безопасности:

1) использование зараженных программ и шпионского ПО, которые могут использоваться для некоторых целей, например, для утечки конфиденциальной информации или для нанесения вреда ПК;

2) неправильное использование периферийных устройств, таких как флешки или жесткие диски, что может привести к утечке данных;

3) небезопасное использование паролей и учетных записей, использование слабых паролей или одинаковых паролей для разных платформ или совместное использование учетных записей;

– риск возникновения пожара и получения электрического удара:

1) неправильное подключение и использование электрооборудования, к примеру, перегрузка розеток и использование поврежденных кабелей;

2) нарушение правил пожарной безопасности, использование запрещенных приборов рядом с ПК и наличие горючих материалов рядом с рабочим оборудованием, все это может угрожать жизни и здоровью людей;

– риск быть подверженным воздействию электромагнитного поля:

1) работа с устаревшим или утратившим пригодность оборудованием, такое оборудование может создавать электромагнитные поля;

2) неправильное расположение оборудования, например, близость к рабочему месту или другим электронным устройствам, что может воздействовать на работу оборудования или здоровье человека;

– риск возникновения пожара и получения электрического удара:

1) неправильное подключение и использование электрооборудования, к примеру, перегрузка розеток и использование поврежденных кабелей;

2) нарушение правил пожарной безопасности, использование запрещенных приборов рядом с ПК и наличие горючих материалов рядом с рабочим оборудованием, все это может угрожать жизни и здоровью людей;

– риск быть подверженным воздействию шума:

1) работа в шумных помещениях или использование шумных устройств.

2.2 Правила безопасности при использовании смартфона

В данном разделе требуется уделить внимание основным рискам, которые могут возникнуть при использовании смартфона. В результате был разработан свод правил, гарантирующий предотвращение рисков, перечисленных далее:

– риск потери или кражи смартфона:

1) потеря личной информации, хранящейся на смартфоне, такой как фотографии, доступ к социальным сетям, банковским данным и другим личным файлам;

2) несанкционированный доступ к личной информации, если у устройства отключена функция блокировки или пароля;

3) риск финансовых потерь, так как смартфоны могут хранить финансовые данные, такие как информация о банковских картах, платежных приложениях и других финансовых данных;

4) риск злоупотреблений в сети, так как неизвестные лица могут получить доступ к социальным сетям, мессенджерам или электронной почте, которые могут быть доступны на смартфоне;

– риск негативного воздействия на психическое и физическое здоровье человека:

1) появление зависимости от использования смартфона. Человек может постоянно использовать смартфон на протяжении всего дня, с трудом оторваться от него, заходить в социальные сети или чаты, даже находясь в неудобной физической позиции, что ведет к психическим страданиям и физическим проблемам;

2) снижение физической активности. Многие люди проводят много времени за смартфоном, это отнимает время на физическую активность, что может привести к ожирению, недостаточной физической форме и различным заболеваниям;

3) негативное воздействие на психическое состояние. Чрезмерное использование смартфона может привести к тревожности, депрессивным состояниям, снижению концентрации и ухудшению памяти;

4) влияние на сон. Использование смартфона перед сном может вызвать бессонницу и беспокойный сон, что может приводить к усталости и различным заболеваниям;

5) риск переживания стресса из-за онлайн-буллинга. Социальные сети и мессенджеры могут стать местом возникновения стресса и онлайн-буллинга, что может негативно сказаться на психическом состоянии и поведении;

6) риск несчастных случаев. Использование смартфона на улице или в общественном транспорте может привести к непредвиденным ситуациям, опасным для жизни. Также запрещается использовать смартфон без включенной функции авиарежим, так как даже заблокированный телефон, который ищет сеть, способен перекрыть частоту оборудования самолета.

Учитывая вышеперассмотренные риски, были разработаны следующие правила при использовании смартфона:

- не использовать смартфон во время вождения автомобиля или других транспортных средств;
- регулярно бэкапировать важные данные на другие устройства или в облачное хранилище, чтобы сохранить информацию в случае потери или кражи;
- ограничить время использования смартфона, особенно перед сном, чтобы избежать проблем со сном;
- использовать пароль, блокировку экрана и другие мера безопасности, чтобы предотвратить незаконный доступ к вашему смартфону;
- обновлять ПО смартфона регулярно, чтобы иметь наиболее защищенный и актуальный софт;
- стараться не посещать ненадежные веб-сервисы и не устанавливать на устройство небезопасные приложения, которые могли бы способствовать краже персональных данных;
- выключать или переводить в авиарежим смартфон при нахождении в самолете.

Соблюдение этих правил поможет вам бережно относиться к своему смартфону и защитить свои данные, а также снизить влияние использования смартфона на ваше здоровье и безопасность.

Заключение

Результатом выполнения данной выпускной квалификационной работы можно считать успешно реализованный в соответствии с заданием конструктор чат-ботов для социальной сети *Telegram*, для которого была разработана структура, написана программная часть и графический пользовательский интерфейс на языке программирования *Python*.

В процессе написания работы были выполнены ряд задач:

- анализ актуальности создания подобной платформы. Были изучены возможности и преимущества, которые могут дать создаваемые конструктором чат-боты для компаний и организаций из различных сфер жизни общества;

- выбор мессенджера для создаваемых чат-ботов. После анализа многих мессенджеров был выбран *Telegram*. Данное решение обусловлено удобством использования, популярностью, безопасностью и многофункциональностью мессенджера;

- выбор ЯП. *Python* — простой язык с понятным синтаксисом и с наличием большого количества библиотек и фреймворков, что упрощает изучение и разработку кода, приложений, в том числе, чат-ботов и их конструкторов;

- выбор СУБД. Была выбрана *PostgreSQL* в качестве СУБД для хранения и управления данными. Данная СУБД обладает рядом преимуществ, среди которых надежность и устойчивость к сбоям, масштабируемость и многофункциональность.

На основе приведенных выше технологий, удалось спроектировать, разработать и протестировать приложение, со следующим функционалом:

- регистрация новых пользователей и вход в систему;
- создание уникальных функциональных чат-ботов для социальной сети *Telegram* при помощи графического интерфейса, без применения навыков программирования;
- запуск созданных чат-ботов.

В результате работы была создана платформа, предоставляющая возможность создавать чат-ботов представителям различных сфер деятельности: от маркетологов и менеджеров по продажам до образовательных учреждений и блогеров. Использование данной платформы способно повысить уровень продаж, улучшить обслуживание клиентов, автоматизировать процессы и упростить взаимодействие с аудиторией.

В перспективе планируется развитие платформы: расширение функционала добавляемых чат-ботов, улучшение пользовательского интерфейса, оптимизация программного кода и добавление интеграции с другими платформами.

					ПГУ1.090301.07.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		75

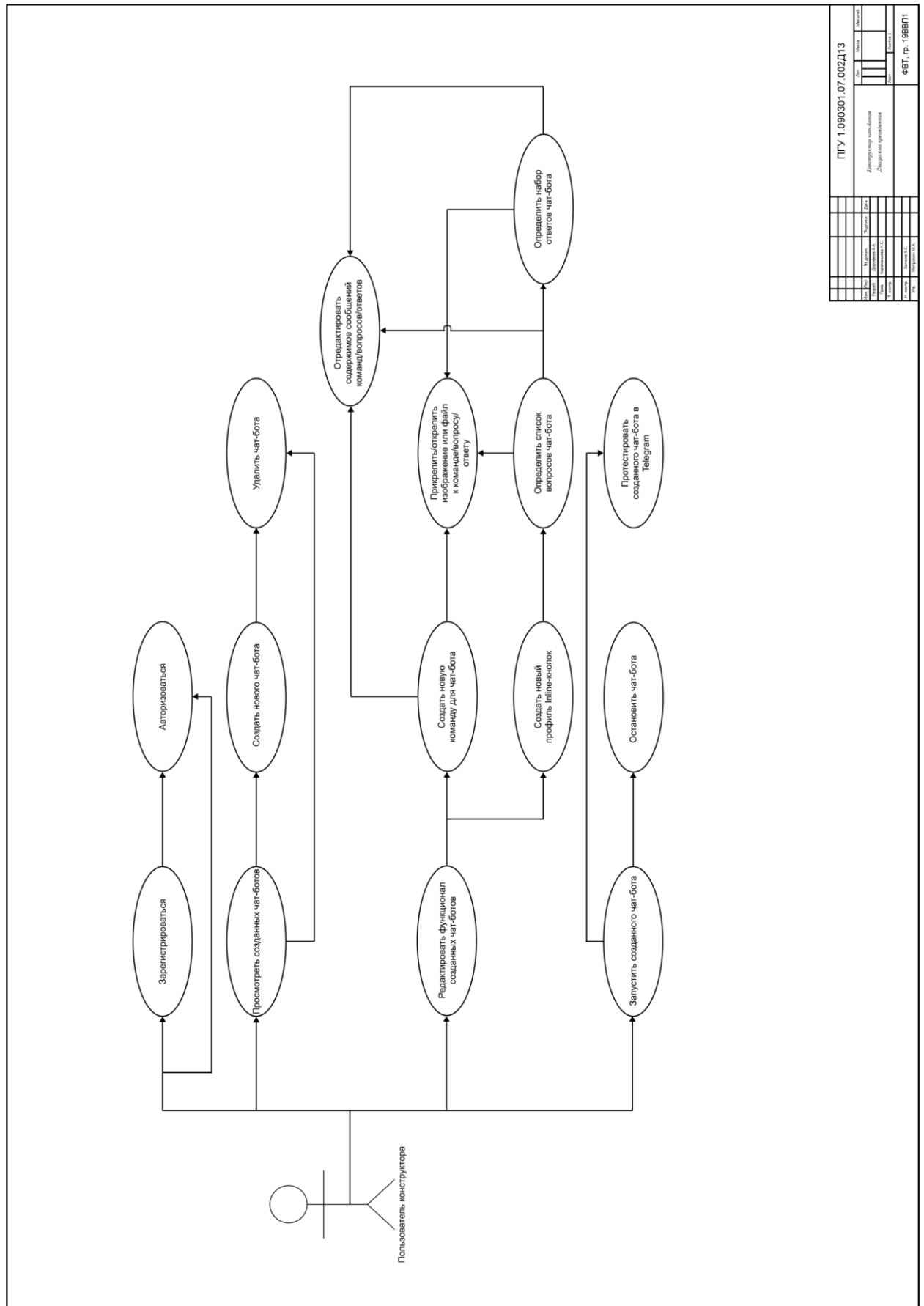
Список использованных источников

1. Артем Демиденко. "Telegram Bot. Руководство по созданию бота в мессенджере Телеграм", 2023. — 23с.
2. Сальникова Л.С. "Современные коммуникационные технологии в бизнесе", 2015. — 296с.
3. Лекции по UML. [Электронный ресурс] UML: https://vk.com/doc4588049_663086875, https://vk.com/doc4588049_663086859 (дата обращения 27.04.2023).
4. Цилюрик Олег Иванович "Сравнение языков программирования", 2018. — 206с.
5. Марк Лутц. "Изучаем Python", 2009. — 832с.
6. Моргунов Е. П. "PostgreSQL. Основы языка SQL: учебное пособие / Е. П. Моргунов." — СПб. БХВ-Петербург, 2018. — 336 с.
7. Mastergroosha.github.io. [Электронный ресурс]. URL: <https://mastergroosha.github.io/telegram-tutorial/> (дата обращения 25.04.2023).
8. Агус Курниаван. Python and PostgreSQL Development, 2019. — 60с.

Приложение А
(обязательное)
Конструктор чат-ботов
UML-диаграммы

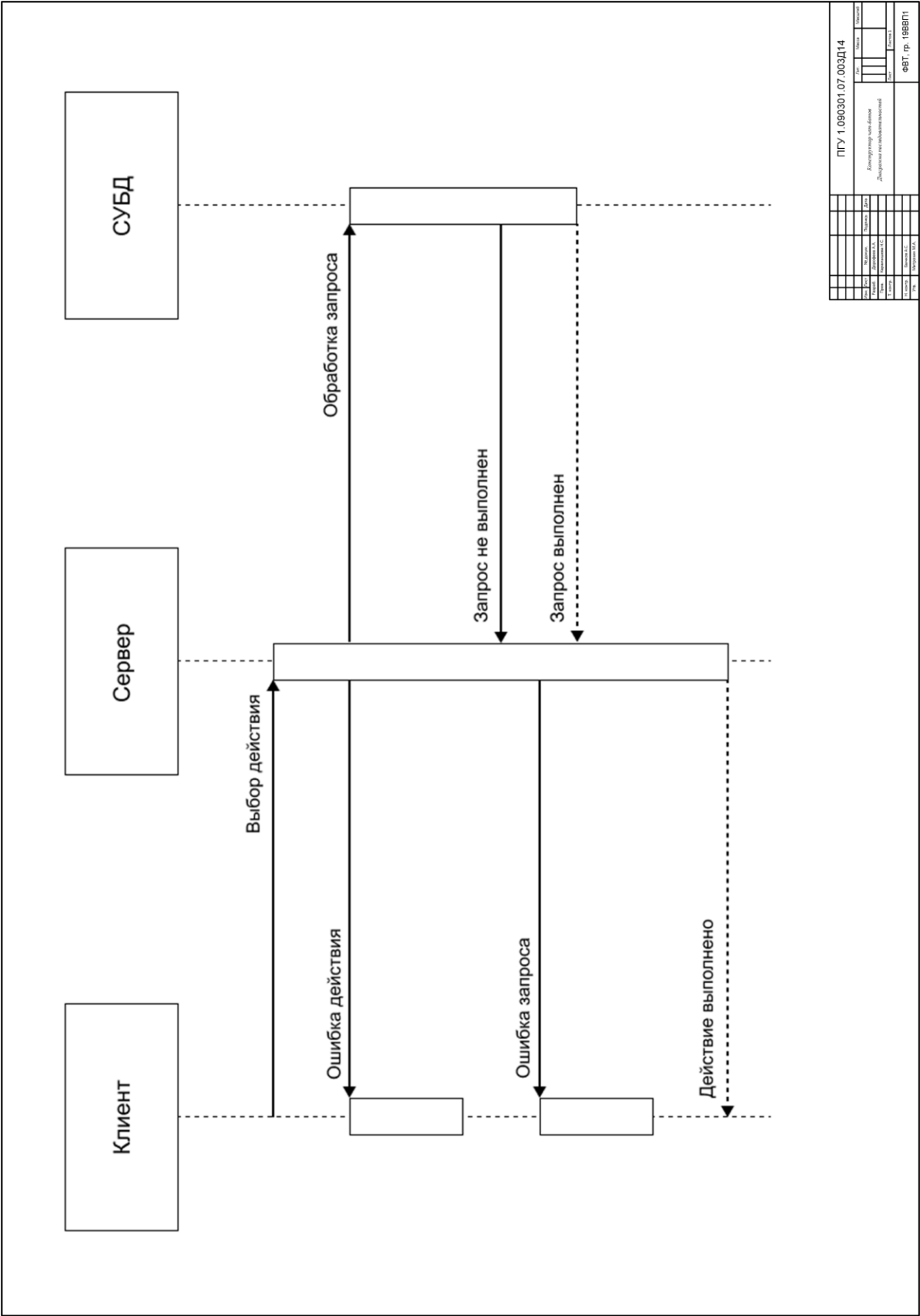
					ПГУ1.090301.07.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – *UML*-диаграмма прецедентов пользователя

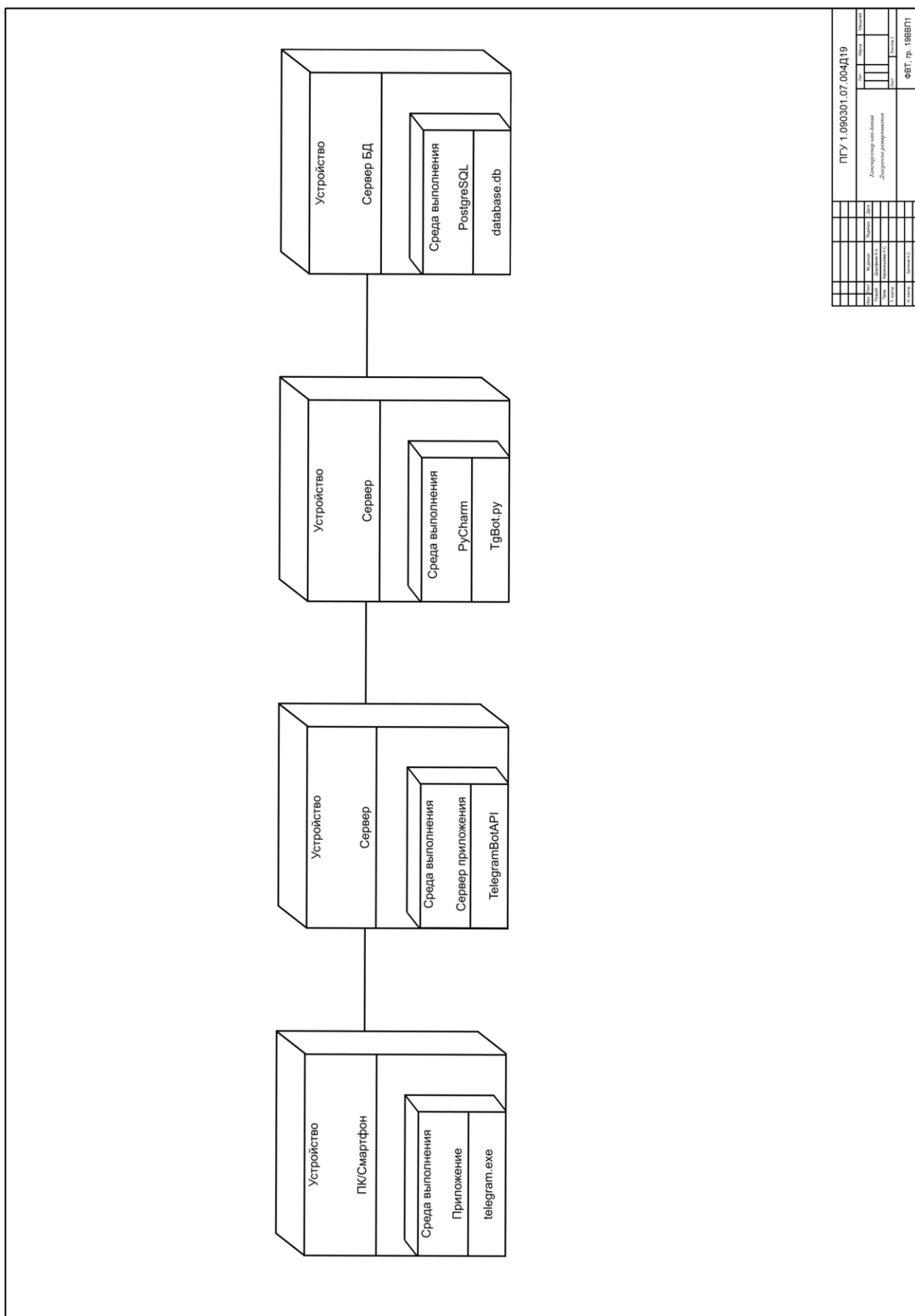


Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.2 – UML-диаграмма последовательностей



Приложение А.3 – UML-диаграмма развертывания чат-ботов



ПГУ 1.090301.07.004Д19									
Изм.	Лист	№ докум.	Подпись	Дата	Изм.	Лист	№ докум.	Подпись	Дата
Утверждено: _____									
Должность: _____									
Подпись: _____									
Дата: _____									
Лист 1									
ФСТ, пр. 168ВП1									

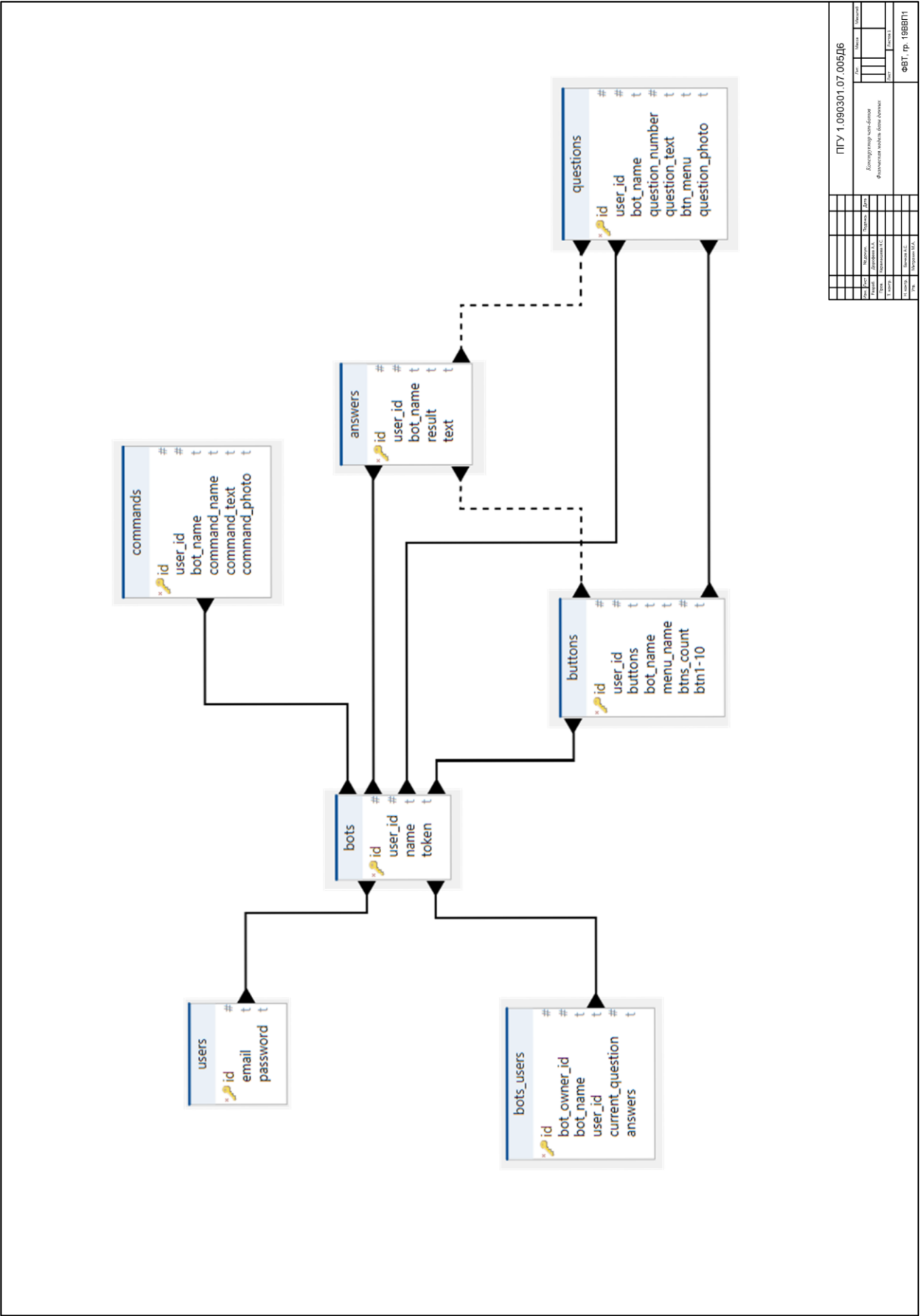
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.07.001 ПЗ

Лист

80

Приложение А.4 – Физическая модель базы данных



Приложение Б
(обязательное)
Конструктор чат-ботов
Листинг программы

					<i>ПГУ1.090301.07.001 ПЗ</i>	Лист
						82
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «bots_constructor.py»

```

from PyQt6 import QtWidgets, QtCore, QtGui
import psycopg2
import sys
import re
import os
import hashlib
import shutil

from constructor_questions import Constructor_questions
from constructor_comands import Constructor_start_command
from constructor_buttons import Constructor_buttons
from constructor_answears import Constructor_answers
from launch_tab import Launch_tab
from dbacc import *

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

with conn.cursor() as cur:
    try:
        cur.execute(f"""CREATE TABLE answears(id serial PRIMARY
KEY, user_id int, bot_name varchar, result varchar, text
varchar, photo varchar)""")
        conn.commit()
    except:
        pass
    try:
        cur.execute(f"""CREATE TABLE bots_users(id serial
PRIMARY KEY, bot_owner_id int, bot_name varchar, user_id
varchar, current_question int, answears varchar)""")
        conn.commit()

    except:
        pass
    try:
        cur.execute(f"""CREATE TABLE comands(id serial PRIMARY
KEY, user_id int, bot_name varchar, comand_name varchar,
comand_text varchar,comand_photo varchar)""")
        conn.commit()
    except:
        pass

    try:
        cur.execute(f"""CREATE TABLE buttons(id serial PRIMARY
KEY,user_id int,bot_name varchar,menu_name varchar,btns_count
int, btn1 varchar, btn2 varchar,
                btn3 varchar, btn4 varchar, btn5
varchar,btn6 varchar,btn7 varchar,btn8 varchar,btn9
varchar,btn10 varchar)""")

```

```

        conn.commit()
    except:
        pass

    try:

        cur.execute(f"""CREATE TABLE questions(id serial PRIMARY
KEY,user_id int,bot_name varchar, question_number int,
question_text varchar, btn_menu varchar,
question_photo varchar)""")
        conn.commit()
    except:
        pass

    try:
        cur.execute("""CREATE TABLE bots(id serial PRIMARY KEY,
user_id int, name varchar, token varchar)""")
        conn.commit()
    except:
        pass

    try:
        cur.execute("""CREATE TABLE users(id serial PRIMARY KEY,
email varchar, password varchar)""")
        conn.commit()
    except:
        pass
conn.close()

class MainWidget(QtWidgets.QWidget):
    def __init__(self):
        QtWidgets.QWidget.__init__(self)

        self.setStyleSheet("""QPushButton{border-radius:9px;
height:25px; background-color:#7d17d1;}

                                QPushButton:hover{background-
color:#9127e8;}

                                QLineEdit{background-color:#b3adad;
border-radius:9px; height:22px;}
                                QTextEdit{background-color:#b3adad;
border-radius:9px;}

                                QTabWidget{background-
color:#1E1E1E;}

                                QTabBar::tab:selected {background-
color:#FBAA50; padding-bottom:10px; padding-top:2px; border: 2px
solid black; padding-left:8px; padding-right:8px;}
                                QTabBar::tab:!selected {background-
color:#007070; padding-left:8px; padding-right:8px;}""")

        self.help_menu = QtWidgets.QMenu("Помощь")
        self.help_menu.addAction('Справка', self.help)

```

```

self.menu_bar = QtWidgets.QMenuBar()
self.menu_bar.addMenu(self.help_menu)

self.setFixedSize(820, 600)
self.setWindowTitle('Bots_constructor')

self.register_login_status = 0
self.user_id = 0
self.add_bot_index = 3
self.bots_btn = []
self.btns = []

self.register_login_label = QtWidgets.QLabel('
Выберите что хотите сделать')
self.register_login_label.setMaximumHeight(12)

self.login_btn = QtWidgets.QPushButton('Войти в
аккаунт')
self.login_btn.setEnabled(False)
self.login_btn.clicked.connect(lambda checked, mode=0:
self.login_register_mode_switch(mode))
self.register_btn =
QtWidgets.QPushButton('Зарегистрироваться')
self.register_btn.clicked.connect(lambda checked,
mode=1: self.login_register_mode_switch(mode))

self.register_login_hl = QtWidgets.QHBoxLayout()
self.register_login_hl.addWidget(self.login_btn)
self.register_login_hl.addWidget(self.register_btn)

self.email_ln = QtWidgets.QLineEdit()
self.email_ln.setPlaceholderText('Введите электронную
почту')

self.password_ln = QtWidgets.QLineEdit()
self.password_ln.setPlaceholderText('Введите пароль')

self.password_ln.setEchoMode(QtWidgets.QLineEdit.EchoMode.Passwo
rd)

self.confirm_password_ln = QtWidgets.QLineEdit()
self.confirm_password_ln.setPlaceholderText('Подтвердите
пароль')

self.confirm_password_ln.setEchoMode(QtWidgets.QLineEdit.EchoMod
e.Password)
self.confirm_password_ln.hide()

self.email_ln.textChanged.connect(self.chack_register_data)

```

```

self.password_ln.textChanged.connect(self.chack_register_data)

self.confirm_password_ln.textChanged.connect(self.chack_register_data)

        self.enter_register_btn = QtWidgets.QPushButton('Войти')

self.enter_register_btn.clicked.connect(self.auth_sign_in)

        self.status_label = QtWidgets.QLabel('')

        self.register_login_vl = QtWidgets.QVBoxLayout()
        self.register_login_vl.addStretch()

self.register_login_vl.addWidget(self.register_login_label)
        self.register_login_vl.addLayout(self.register_login_hl)
        self.register_login_vl.addWidget(self.email_ln)
        self.register_login_vl.addWidget(self.password_ln)

self.register_login_vl.addWidget(self.confirm_password_ln)

self.register_login_vl.addWidget(self.enter_register_btn)
        self.register_login_vl.addWidget(self.status_label)
        self.register_login_vl.addStretch()

        self.register_login_main_hl = QtWidgets.QHBoxLayout()
        self.register_login_main_hl.addStretch()

self.register_login_main_hl.addLayout(self.register_login_vl)
        self.register_login_main_hl.addStretch()

        self.main_vl = QtWidgets.QVBoxLayout()
        self.main_vl.setMenuBar(self.menu_bar)
        self.main_vl.addLayout(self.register_login_main_hl)

        self.setLayout(self.main_vl)

def chack_register_data(self):
    if self.register_login_status == 1:
        pattern = r"^[a-zA-Z0-9_+-.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+\.$"

        if re.match(pattern, self.email_ln.text()) is None:
            self.status_label.setText('Адрес электронной почты некорректный')
            self.email_ln.setStyleSheet("""background-color:#d91111;""")
            self.enter_register_btn.setEnabled(False)
        elif len(self.password_ln.text()) < 6:
            self.status_label.setText('Пароль слишком короткий')

```

```

        self.password_ln.setStyleSheet("""background-
color:#d91111;""")
        self.email_ln.setStyleSheet("""background-
color:None""")
        self.enter_register_btn.setEnabled(False)

        elif self.confirm_password_ln.text() !=
self.password_ln.text():
            self.status_label.setText('Пароли не совпадают')

self.confirm_password_ln.setStyleSheet("""background-
color:#d91111;""")
        self.password_ln.setStyleSheet("""background-
color:None;""")
        self.enter_register_btn.setEnabled(False)
    else:
        self.status_label.setText('Все данные прошли
проверку')
        self.email_ln.setStyleSheet("""background-
color:None""")
        self.password_ln.setStyleSheet("""background-
color:None""")

self.confirm_password_ln.setStyleSheet("""background-
color:None;""")
        self.enter_register_btn.setEnabled(True)

    def chack_add_bot_data(self):
        conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)

        with conn.cursor() as cur:

            cur.execute("""SELECT name FROM bots WHERE
user_id=%s""", (self.user_id,))
            all_bots_names = cur.fetchall()
            names = []
            for bot_name in all_bots_names:
                names.append(bot_name[0])

            if self.add_bot_name_ln.text() == '':
                self.add_bot_status_lb.setText('Имя бота не
может быть пустым')
            elif self.add_bot_name_ln.text() in names:
                self.add_bot_status_lb.setText('Такое имя уже
занято, придумайте другое')
            elif self.add_bot_token_ln.text() == '':
                self.add_bot_status_lb.setText('Токен не может
быть пустым')
            elif len(self.add_bot_token_ln.text()) < 20:

```

```

        self.add_bot_status_lb.setText('Значение токена
некорректно')
        elif ':' not in self.add_bot_token_ln.text():
            self.add_bot_status_lb.setText('Значение токена
некорректно')
        else:
            self.add_bot_status_lb.setText('')
            self.add_bot_save_btn.setEnabled(True)
    conn.close()

def add_bot_cancel(self):
    self.add_bot_btn.show()

    self.add_bot_save_btn.hide()
    self.add_bot_cancel_btn.hide()
    self.add_bot_name_ln.hide()
    self.add_bot_token_ln.hide()
    self.add_bot_status_lb.hide()

def add_bot_save(self):
    conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)

    with conn.cursor() as cur:

        try:
            dirname =
os.path.dirname(os.path.abspath(__file__))
            dir_name = os.path.join(dirname,
f'{self.user_id}_{self.add_bot_name_ln.text()}')
            os.mkdir(dir_name)
            os.mkdir(os.path.join(dir_name, 'photos'))

            shutil.copy(os.path.join(dirname, 'main.py'),
dir_name)
            shutil.copy(os.path.join(dirname, 'buttons.py'),
dir_name)
        except:
            pass

        cur.execute("""INSERT INTO bots (user_id, name,
token) VALUES (%s,%s,%s)""",
                    (self.user_id,
self.add_bot_name_ln.text(), self.add_bot_token_ln.text()))

        conn.commit()

    conn.close()

    self.add_bot_btn.show()
    self.add_bot_save_btn.hide()
    self.add_bot_cancel_btn.hide()

```

```

self.add_bot_name_ln.hide()
self.add_bot_token_ln.hide()
self.add_bot_status_lb.hide()

hl = QtWidgets.QHBoxLayout()
btn = QtWidgets.QPushButton(self.add_bot_name_ln.text())
self.bots_btn.append(btn)
btn.clicked.connect(lambda checked,
bot_name=self.add_bot_name_ln.text(): self.choose_bot(bot_name))
btn.setMaximumWidth(200)
btn.setMinimumHeight(30)
hl.addWidget(btn)
self.bots_btn.append(btn)
self.bots_list.insertLayout(self.bots_list.count() -
self.add_bot_index, hl)

self.add_bot_index += 1

def add_bot(self):
    self.add_bot_btn.hide()

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

    with conn.cursor() as cur:
        cur.execute("""SELECT * FROM bots WHERE
user_id=%s""", (self.user_id,))
        bots_count = cur.fetchall()
        bots_count = len(bots_count)

        if bots_count == 5:
            lb = QtWidgets.QLabel('Вы достигли лимита по
добавлению ботов')
            hl = QtWidgets.QHBoxLayout()
            hl.addStretch()
            hl.addWidget(lb)
            hl.addStretch()
            self.bots_list.addLayout(hl)
            return False
        conn.close()

    self.add_bot_vl = QtWidgets.QVBoxLayout()

    self.add_bot_name_ln = QtWidgets.QLineEdit()
    self.add_bot_name_ln.setPlaceholderText('Введите имя
Бота')
    self.add_bot_name_ln.setMaximumWidth(300)

    self.add_bot_name_ln.textChanged.connect(self.chack_add_bot_data
)

    self.add_bot_token_ln = QtWidgets.QLineEdit()

```

```

        self.add_bot_token_ln.setPlaceholderText('Введите токен
Бота')
        self.add_bot_token_ln.setMaximumWidth(300)

self.add_bot_token_ln.textChanged.connect(self.chack_add_bot_dat
a)

        self.add_bot_btn_hl = QtWidgets.QHBoxLayout()

        self.add_bot_save_btn =
QtWidgets.QPushButton('Добавить')
        self.add_bot_save_btn.setMaximumWidth(150)
        self.add_bot_save_btn.clicked.connect(self.add_bot_save)
        self.add_bot_save_btn.setEnabled(False)

        self.add_bot_cancel_btn =
QtWidgets.QPushButton('Отменить')

self.add_bot_cancel_btn.clicked.connect(self.add_bot_cancel)
        self.add_bot_cancel_btn.setMaximumWidth(150)

        self.add_bot_btn_hl.addWidget(self.add_bot_save_btn)
        self.add_bot_btn_hl.addWidget(self.add_bot_cancel_btn)

        self.add_bot_status_lb = QtWidgets.QLabel('Заполните
данные')

        self.add_bot_vl.addWidget(self.add_bot_name_ln)
        self.add_bot_vl.addWidget(self.add_bot_token_ln)
        self.add_bot_vl.addWidget(self.add_bot_status_lb)
        self.add_bot_vl.addLayout(self.add_bot_btn_hl)

        self.add_bot_hl2 = QtWidgets.QHBoxLayout()
        self.add_bot_hl2.addLayout(self.add_bot_vl)

        self.bots_list.addLayout(self.add_bot_hl2)

def bot_logout(self):
    self.scroll_area.show()
    self.account_label.show()
    self.log_out_btn.show()

    self.bot_label.hide()
    self.bot_log_out_btn.hide()
    self.tw.hide()
    self.tw.deleteLater()
    self.bot_label.deleteLater()
    self.bot_log_out_btn.deleteLater()

def choose_bot(self, bot_name):

    self.scroll_area.hide()

```

```

self.account_label.hide()
self.log_out_btn.hide()

self.bot_label = QtWidgets.QLabel(f'Бот: {bot_name}')
self.bot_log_out_btn = QtWidgets.QPushButton('Вернуться
ко всем ботам')
self.bot_log_out_btn.clicked.connect(self.bot_logout)

self.bot_log_out_hl = QtWidgets.QHBoxLayout()
self.bot_log_out_hl.addStretch()
self.bot_log_out_hl.addWidget(self.bot_label)
self.bot_log_out_hl.addWidget(self.bot_log_out_btn)
self.bot_log_out_hl.addStretch()

self.tw = QtWidgets.QTabWidget()

constructor_questions_class =
Constructor_questions(bot_name, self.user_id)

self.tw.addTab(Constructor_start_command(bot_name,
self.user_id), 'Редактор команд')
self.tw.addTab(Constructor_buttons(bot_name,
self.user_id, constructor_questions_class), 'Редактор кнопок')
self.tw.addTab(constructor_questions_class, 'Редактор
вопросов')
self.tw.addTab(Constructor_answers(bot_name,
self.user_id), 'Редактор ответов')
self.tw.addTab(Launch_tab(bot_name, self.user_id, self),
'Управление чат-ботом')

self.main_vl.addLayout(self.bot_log_out_hl)
self.main_vl.addWidget(self.tw)
self.main_vl.addWidget(self.tw)

def login_register_mode_switch(self, mode):
    if mode == 1:
        self.login_btn.setEnabled(True)
        self.register_btn.setEnabled(False)
        self.confirm_password_ln.show()

self.enter_register_btn.setText('Зарегистрироваться')
self.register_login_status = 1
self.enter_register_btn.setEnabled(False)
elif mode == 0:
    self.login_btn.setEnabled(False)
    self.register_btn.setEnabled(True)
    self.confirm_password_ln.hide()
    self.enter_register_btn.setText('Войти')
    self.register_login_status = 0
    self.enter_register_btn.setEnabled(True)

```

```

def logout(self):
    self.scroll_area.hide()
    self.scroll_area.deleteLater()

    self.account_label.hide()
    self.log_out_btn.hide()
    self.account_label.deleteLater()
    self.log_out_btn.deleteLater()

    self.login_btn.show()
    self.register_btn.show()
    self.enter_register_btn.show()
    self.email_ln.show()
    self.password_ln.show()
    self.status_label.show()
    self.register_login_label.show()

    self.add_bot_index = 3

def delete_bot(self, bot_name):
    print(bot_name)
    for btn in self.bots_btn:
        print(btn.text())
        if btn.text() == bot_name:
            btn.hide()
def auth_sign_in(self):
    conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)

    with conn.cursor() as cur:
        if self.register_login_status == 0:
            str_to_hash = self.password_ln.text() +
'2aebyoJP4'

            hashed = hashlib.sha256(str_to_hash.encode())
            hash = hashed.hexdigest()
            email = self.email_ln.text()
            cur.execute("""SELECT id FROM users WHERE
email=%s AND password=%s""", (email, hash))
            id = cur.fetchone()
            if id is None:
                self.status_label.setText('Вы ввели
некорректный логин или пароль')
            else:
                self.user_id = id[0]

                self.login_btn.hide()
                self.register_btn.hide()
                self.enter_register_btn.hide()
                self.email_ln.hide()
                self.password_ln.hide()
                self.confirm_password_ln.hide()
                self.status_label.hide()

```

```

        self.register_login_label.hide()

        for i in range(self.main_vl.count()):
            layout_item = self.main_vl.itemAt(i)
            self.main_vl.removeItem(layout_item)

        self.account_label =
QtWidgets.QLabel(f'Аккаунт: {email}')
        self.log_out_btn =
QtWidgets.QPushButton('Выйти из аккаунта')

self.log_out_btn.clicked.connect(self.logout)

        self.log_out_hl = QtWidgets.QHBoxLayout()
        self.log_out_hl.addStretch()

self.log_out_hl.addWidget(self.account_label)
        self.log_out_hl.addWidget(self.log_out_btn)
        self.log_out_hl.addStretch()

        self.bots_list = QtWidgets.QVBoxLayout()

        self.gb = QtWidgets.QGroupBox()
        self.gb.setLayout(self.bots_list)

        self.scroll_area = QtWidgets.QScrollArea()
        self.scroll_area.setWidgetResizable(True)
        self.scroll_area.setWidget(self.gb)
        self.scroll_area.setMaximumHeight(450)

        self.main_vl.addLayout(self.log_out_hl)
        self.main_vl.addSpacing(35)
        self.main_vl.addWidget(self.scroll_area)

        self.bots_info = QtWidgets.QLabel('Выберите
бота для редактирования или добавьте его')
        self.bots_info_hl = QtWidgets.QHBoxLayout()
        self.bots_info_hl.addStretch()
        self.bots_info_hl.addWidget(self.bots_info)
        self.bots_info_hl.addStretch()

        self.add_bot_btn =
QtWidgets.QPushButton('Добавить бота')

self.add_bot_btn.clicked.connect(self.add_bot)
        self.add_bot_btn.setMinimumWidth(130)
        self.add_bot_btn.setMinimumHeight(30)
        self.add_bot_hl = QtWidgets.QHBoxLayout()
        self.add_bot_hl.addStretch()
        self.add_bot_hl.addWidget(self.add_bot_btn)
        self.add_bot_hl.addStretch()

```

```

        self.bots_list.addLayout(self.bots_info_hl)
        cur.execute("""SELECT name FROM bots WHERE
user_id=%s""", (self.user_id,))
        all_bots_names = cur.fetchall()

        for bot_name in all_bots_names:
            hl = QtWidgets.QHBoxLayout()
            btn = QtWidgets.QPushButton(bot_name[0])
            self.bots_btn.append(btn)
            btn.clicked.connect(lambda checked,
bot_name=bot_name[0]: self.choose_bot(bot_name))
            btn.setMaximumWidth(200)
            btn.setMinimumHeight(30)
            hl.addWidget(btn)
            self.btns.append(btn)
            self.bots_list.addLayout(hl)

            self.add_bot_index += 1
            self.bots_list.addLayout(self.add_bot_hl)
            self.bots_list.addStretch()

    elif self.register_login_status == 1:
        str_to_hash = self.password_ln.text() +
'2aebyoJP4'
        hashed = hashlib.sha256(str_to_hash.encode())
        hash = hashed.hexdigest()
        cur.execute("""INSERT INTO users (email,
password) VALUES (%s,%s)""", (self.email_ln.text(), hash))
        conn.commit()
        self.status_label.setText('Ваш аккаунт успешно
зарегистрирован')

    conn.close()

def help(self):
    msg = QtWidgets.QMessageBox()
    msg.setWindowTitle("Помощь")
    with open("info.txt", encoding='utf-8') as file:
        textInfo = file.read()
    file.close()
    msg.setText(textInfo)
    msg.exec()

app = QtWidgets.QApplication(sys.argv)
app.setStyle('Fusion')
wid = MainWidget()
wid.show()
sys.exit(app.exec())

```

Файл «*buttons.py*»

```
from telegram import *
```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.07.001 ПЗ

Лист

94

```

import psycopg2
import os
from dbacc import *

dirname = os.path.dirname(os.path.abspath(__file__))
bot_owner_id = int(dirname.split('\\')[ -1].split('_', 1)[0])
bot_name = dirname.split('\\')[ -1].split('_', 1)[1]

def get_keyboard(btn_menu):
    conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
    with conn.cursor() as cur:
        cur.execute("""SELECT btns_count, btn1,
btn2,btn3,btn4,btn5,btn6,btn7,btn8,btn9,btn10 FROM buttons WHERE
user_id=%s AND bot_name=%s AND menu_name=%s""",
                    (bot_owner_id, bot_name, btn_menu))

        menu = cur.fetchall()

        for btns_count, btn1,
btn2,btn3,btn4,btn5,btn6,btn7,btn8,btn9,btn10 in menu:
            if btns_count == 2:
                markup2 = InlineKeyboardMarkup(
                    [[(InlineKeyboardButton(btn1,
callback_data=btn1))], [(InlineKeyboardButton(btn2,
callback_data=btn2))]])
                return markup2
            elif btns_count == 3:
                markup2 = InlineKeyboardMarkup(
                    [[(InlineKeyboardButton(btn1,
callback_data=btn1)), (InlineKeyboardButton(btn2,
callback_data=btn2))],
                    [(InlineKeyboardButton(btn3,
callback_data=btn3))]])
                return markup2
            elif btns_count == 4:
                markup2 = InlineKeyboardMarkup(
                    [[(InlineKeyboardButton(btn1,
callback_data=btn1)), (InlineKeyboardButton(btn2,
callback_data=btn2))],
                    [(InlineKeyboardButton(btn3,
callback_data=btn3)), (InlineKeyboardButton(btn4,
callback_data=btn4))]])
                return markup2
            elif btns_count == 5:
                markup2 = InlineKeyboardMarkup(
                    [[(InlineKeyboardButton(btn1,
callback_data=btn1)), (InlineKeyboardButton(btn2,
callback_data=btn2))],

```

```

        [(InlineKeyboardButton(btn3,
callback_data=btn3)), (InlineKeyboardButton(btn4,
callback_data=btn4))],
        [(InlineKeyboardButton(btn5,
callback_data=btn5))]])
    return markup2
elif btns_count == 6:
    markup2 = InlineKeyboardMarkup(
        [(InlineKeyboardButton(btn1,
callback_data=btn1)), (InlineKeyboardButton(btn2,
callback_data=btn2))],
        [(InlineKeyboardButton(btn3,
callback_data=btn3)), (InlineKeyboardButton(btn4,
callback_data=btn4))],
        [(InlineKeyboardButton(btn5,
callback_data=btn5)), (InlineKeyboardButton(btn6,
callback_data=btn6))]])
    return markup2
elif btns_count == 7:
    markup2 = InlineKeyboardMarkup(
        [(InlineKeyboardButton(btn1,
callback_data=btn1)), (InlineKeyboardButton(btn2,
callback_data=btn2))],
        [(InlineKeyboardButton(btn3,
callback_data=btn3)), (InlineKeyboardButton(btn4,
callback_data=btn4))],
        [(InlineKeyboardButton(btn5,
callback_data=btn5)), (InlineKeyboardButton(btn6,
callback_data=btn6))],
        [(InlineKeyboardButton(btn7,
callback_data=btn7))]])
    return markup2
elif btns_count == 8:
    markup2 = InlineKeyboardMarkup(
        [(InlineKeyboardButton(btn1,
callback_data=btn1)), (InlineKeyboardButton(btn2,
callback_data=btn2))],
        [(InlineKeyboardButton(btn3,
callback_data=btn3)), (InlineKeyboardButton(btn4,
callback_data=btn4))],
        [(InlineKeyboardButton(btn5,
callback_data=btn5)), (InlineKeyboardButton(btn6,
callback_data=btn6))],
        [(InlineKeyboardButton(btn7,
callback_data=btn7)), (InlineKeyboardButton(btn8,
callback_data=btn8))]])
    return markup2
elif btns_count == 9:
    markup2 = InlineKeyboardMarkup(
        [(InlineKeyboardButton(btn1,
callback_data=btn1)), (InlineKeyboardButton(btn2,
callback_data=btn2))],

```

```

        [(InlineKeyboardButton(btn3,
callback_data=btn3)), (InlineKeyboardButton(btn4,
callback_data=btn4))],
        [(InlineKeyboardButton(btn5,
callback_data=btn5)), (InlineKeyboardButton(btn6,
callback_data=btn6))],
        [(InlineKeyboardButton(btn7,
callback_data=btn7)), (InlineKeyboardButton(btn8,
callback_data=btn8))],
        [(InlineKeyboardButton(btn9,
callback_data=btn9))]])
    return markup2
elif btns_count == 10:
    markup2 = InlineKeyboardMarkup(
        [(InlineKeyboardButton(btn1,
callback_data=btn1)), (InlineKeyboardButton(btn2,
callback_data=btn2))],
        [(InlineKeyboardButton(btn3,
callback_data=btn3)), (InlineKeyboardButton(btn4,
callback_data=btn4))],
        [(InlineKeyboardButton(btn5,
callback_data=btn5)), (InlineKeyboardButton(btn6,
callback_data=btn6))],
        [(InlineKeyboardButton(btn7,
callback_data=btn7)), (InlineKeyboardButton(btn8,
callback_data=btn8))],
        [(InlineKeyboardButton(btn9,
callback_data=btn9)), (InlineKeyboardButton(btn10,
callback_data=btn10))]])
    return markup2

conn.close()

```

Файл «*constructor_answers.py*»

```

from PyQt6 import QtWidgets, QtCore, QtGui
import psycpg2
import os
import shutil
from dbacc import *
class Constructor_answers(QtWidgets.QWidget):
    def __init__(self, bot_name, user_id):
        QtWidgets.QWidget.__init__(self)
        self.bot_name = bot_name
        self.user_id = user_id
        self.n = 0
        self.textes = []
        self.ln_pathes = []
        self.check_boxes = []
        self.btn_picks = []
        self.btns_changes = []

        self.setStyleSheet("""QPushButton{

```

```

border-radius:9px;
height:25px;
background-color:#7d17d1;
}

QPushButton:hover{
background-
color:#9127e8;
}

QLineEdit{
background-color:#b3adad;
border-radius:9px;
height:22px;
}

QTextEdit{
background-color:#b3adad;
border-radius:9px;
}

QComboBox{
background-color:#a38c8c;
}

""")

self.add_answer_combo_boxes = []
self.bots_list = QtWidgets.QVBoxLayout()
self.gb = QtWidgets.QGroupBox()
self.gb.setLayout(self.bots_list)

self.scroll_area = QtWidgets.QScrollArea()
self.scroll_area.setWidgetResizable(True)
self.scroll_area.setWidget(self.gb)
self.scroll_area.setMaximumHeight(450)

self.info_hl = QtWidgets.QHBoxLayout()
self.info_label = QtWidgets.QLabel('Здесь ты сможешь
настраивать разные варианты того, какой ответ получит
пользователь по окончании теста')
self.info_label.setMaximumHeight(15)

self.info_hl.addStretch()
self.info_hl.addWidget(self.info_label)
self.info_hl.addStretch()

self.add_menu_btn = QtWidgets.QPushButton('Добавить
ответ на опрос')
self.add_menu_btn.clicked.connect(self.add_answer)
self.add_menu_btn.setMinimumHeight(25)
self.add_menu_btn.setMinimumWidth(150)

self.add_menu_btn_hl = QtWidgets.QHBoxLayout()

```

```

self.add_menu_btn_hl.addStretch()
self.add_menu_btn_hl.addWidget(self.add_menu_btn)
self.add_menu_btn_hl.addStretch()

self.groupBox = QtWidgets.QGroupBox()
self.groupBox.setMaximumWidth(940)
self.groupBox.setStyleSheet(''QGroupBox {margin-top:
5px; }
QGroupBox:enabled {border: 2px solid black; }
QGroupBox::title {subcontrol-origin: margin; left:
3ex;}''')

self.cb = QtWidgets.QCheckBox('Добавить фото')
self.cb.stateChanged.connect(self.empty_check1)
self.lnm = QtWidgets.QLineEdit()
self.lnm.setEnabled(False)
self.lnm.textChanged.connect(self.empty_check1)
self.btnm1 = QtWidgets.QPushButton('...')
self.btnm1.setMinimumWidth(50)
self.btnm1.clicked.connect(self.pick1)

self.tem = QtWidgets.QTextEdit()
self.tem.setPlaceholderText('Введите текст ответа')
self.tem.setMaximumHeight(90)
self.tem.textChanged.connect(self.empty_check1)

self.hlm = QtWidgets.QHBoxLayout()
self.hlm.addWidget(self.cb)
self.hlm.addWidget(self.lnm)
self.hlm.addWidget(self.btnm1)

self.btnm3 = QtWidgets.QPushButton('Добавить')
self.btnm3.clicked.connect(self.add_answer_save)
self.btnm3.setEnabled(False)
self.btnm4 = QtWidgets.QPushButton('Отмена')
self.btnm4.clicked.connect(self.add_answer_cancel)
self.add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
self.add_comand_btn_hl2.addWidget(self.btnm3)
self.add_comand_btn_hl2.addWidget(self.btnm4)
self.add_menu_vl = QtWidgets.QVBoxLayout(self.groupBox)

conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
with conn.cursor() as cur:
    cur.execute("""SELECT question_number, btn_menu FROM
questions WHERE user_id=%s AND bot_name=%s""",
                (self.user_id, self.bot_name))
    all_questions = cur.fetchall()
    for question_number, btn_menu in all_questions:
        label=QtWidgets.QLabel(f'Вопрос номер
{question_number}')
        combo_box = QtWidgets.QComboBox()

```

```

        cur.execute("""SELECT btn1, btn2,
btn3,btn4,btn5,btn6,btn7,btn8,btn9,btn10 FROM buttons WHERE
menu_name=%s AND user_id=%s AND bot_name=%s""",
                    (btn_menu, self.user_id,
self.bot_name))

        for btn1, btn2,
btn3,btn4,btn5,btn6,btn7,btn8,btn9,btn10 in cur.fetchall():
            if btn1:
                combo_box.addItem(btn1)
            if btn2:
                combo_box.addItem(btn2)
            if btn3:
                combo_box.addItem(btn3)
            if btn4:
                combo_box.addItem(btn4)
            if btn5:
                combo_box.addItem(btn5)
            if btn6:
                combo_box.addItem(btn6)
            if btn7:
                combo_box.addItem(btn7)
            if btn8:
                combo_box.addItem(btn8)
            if btn9:
                combo_box.addItem(btn9)
            if btn10:
                combo_box.addItem(btn10)
        self.add_answear_combo_boxes.append(combo_box)
        hl = QtWidgets.QHBoxLayout()
        hl.addWidget(label)
        hl.addWidget(combo_box)

        self.add_menu_vl.addLayout(hl)
    conn.close()

    self.add_menu_vl.addWidget(self.tem)
    self.add_menu_vl.addLayout(self.hlm)
    self.add_menu_vl.addLayout(self.add_comand_btn_hl2)
    self.groupBox.hide()
    self.bots_list.addLayout(self.info_hl)

    conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
    with conn.cursor() as cur:
        cur.execute("""SELECT id, result, text, photo FROM
answers WHERE user_id=%s AND bot_name=%s""", (self.user_id,
self.bot_name))
        all_answers = cur.fetchall()
        for id, answer, text, photo in all_answers:
            groupBox = QtWidgets.QGroupBox()
            groupBox.setMaximumWidth(940)

```

```

        groupBox.setStyleSheet('''QGroupBox {margin-top:
2px; }
QGroupBox:enabled {border: 2px solid grey;}
QGroupBox::title {subcontrol-origin: margin; left: 3ex; }''')
        cb = QtWidgets.QCheckBox('Добавить фото')
        self.check_boxes.append(cb)
        lnm = QtWidgets.QLineEdit()
        self.ln_pathes.append(lnm)
        lnm.setEnabled(False)
        btnm1 = QtWidgets.QPushButton('...')
        btnm1.setMinimumWidth(50)
        self.btn_picks.append(btnm1)

        if photo is not None:
            cb.setChecked(True)
            lnm.setText(photo)

        cb.stateChanged.connect(lambda checked,
command_id=self.n: self.empty_check_cganges(command_id))
        lnm.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_cganges(command_id))
        btnm1.clicked.connect(lambda checked,
command_id=self.n: self.pick_changes(command_id))

        tem = QtWidgets.QTextEdit(text)
        self.textes.append(tem)
        tem.textChanged.connect(lambda
command_id=self.n: self.empty_check_cganges(command_id))
        tem.setPlaceholderText('Введите текст ответа')
        tem.setMaximumHeight(90)

        hlm = QtWidgets.QHBoxLayout()
        hlm.addWidget(cb)
        hlm.addWidget(lnm)
        hlm.addWidget(btnm1)

        btnm3 = QtWidgets.QPushButton('Сохранить')
        self.btns_changes.append(btnm3)
        btnm3.clicked.connect(lambda checked,
command_id=self.n, id=id: self.save_changes(command_id, id))

        self.n +=1
        add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
        add_comand_btn_hl2.addWidget(btnm3)

        add_menu_vl = QtWidgets.QVBoxLayout(groupBox)
        n = 1
        for comb in answer.split('|'):
            if comb != '':
                label = QtWidgets.QLabel(f'Вопрос номер
{n}')

                combo_box = QtWidgets.QComboBox()

```

```

        combo_box.addItem(comb)
        hl = QtWidgets.QHBoxLayout()
        hl.addWidget(label)
        hl.addWidget(combo_box)
        add_menu_vl.addLayout(hl)
        n += 1

    add_menu_vl.addWidget(tem)
    add_menu_vl.addLayout(hlm)
    add_menu_vl.addLayout(add_comand_btn_hl2)

    self.bots_list.addWidget(groupBox)
    conn.close()

    self.bots_list.addLayout(self.add_menu_btn_hl)
    self.bots_list.addWidget(self.groupBox)

    self.main_vl = QtWidgets.QVBoxLayout()
    self.main_vl.addWidget(self.scroll_area)
    self.bots_list.addStretch()

    self.setLayout(self.main_vl)
    def save_changes(self, command_id, id):
        conn = psycopg2.connect(database=data_base, user=user,
                                password=password, host=host, port=port)
        with conn.cursor() as cur:
            if self.check_boxes[command_id].isChecked():
                dirname =
os.path.dirname(os.path.abspath(__file__))
                dir_name = os.path.join(dirname,
f'{self.user_id}_{self.bot_name}')
                dir_name = os.path.join(dir_name, 'photos')

                shutil.copy(self.ln_paths[command_id].text(),
dir_name)

                save_path = os.path.join(dir_name,
self.ln_paths[command_id].text().split('/')[ -1])

                cur.execute("""UPDATE answers SET text=%s,
photo=%s, question_photo=%s WHERE
id=%s""", (self.textes[command_id].toPlainText(), save_path, id))
                conn.commit()
            else:
                cur.execute("""UPDATE answers SET text=%s WHERE
id=%s""", (self.textes[command_id].toPlainText(), id))
                conn.commit()
        conn.close()

    def pick_changes(self, comand_id):
        print(comand_id)
        self.filename = QtWidgets.QFileDialog.getOpenFileName()

```

```

        if self.filename:
            self.ln_pathes[comamd_id].setText(self.filename[0])

def empty_check_cganges(self, comand_id):
    print(comand_id)

    if self.textes[comand_id].toPlainText() == '':
        self.btns_changes[comand_id].setEnabled(False)
    else:
        if self.check_boxes[comand_id].isChecked():
            if self.ln_pathes[comand_id].text() == '':

self.btns_changes[comand_id].setEnabled(False)
        else:

self.btns_changes[comand_id].setEnabled(True)
        else:
            self.btns_changes[comand_id].setEnabled(True)
def pick1(self):
    self.filename = QtWidgets.QFileDialog.getOpenFileName()
    if self.filename:
        self.lnm.setText(self.filename[0])
def empty_check1(self):
    if self.tem.toPlainText() == '':
        self.btnm3.setEnabled(False)
    else:
        if self.cb.isChecked():
            if self.lnm.text() == '':
                self.btnm3.setEnabled(False)
            else:
                self.btnm3.setEnabled(True)
        else:
            self.btnm3.setEnabled(True)
def add_answer_save(self):
    string = ''
    for cb in self.add_answer_combo_boxes:
        string+=cb.currentText()+'|'

    conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
    with conn.cursor() as cur:

        if self.cb.isChecked():
            dirname =
os.path.dirname(os.path.abspath(__file__))
            dir_name = os.path.join(dirname,
f'{self.user_id}_{self.bot_name}')
            dir_name = os.path.join(dir_name, 'photos')

            shutil.copy(self.lnm.text(), dir_name)
            save_path = os.path.join(dir_name,
self.lnm.text().split('/')[0])

```

```

        cur.execute(
            f"""INSERT INTO answears (user_id, bot_name,
result, text, photo) VALUES (%s,%s,%s,%s,%s)""",
            (self.user_id, self.bot_name, string,
self.tem.toPlainText(), save_path))
        conn.commit()
    else:
        cur.execute(
            f"""INSERT INTO answears (user_id, bot_name,
result, text) VALUES (%s,%s,%s,%s)""",
            (self.user_id, self.bot_name, string,
self.tem.toPlainText()))
        conn.commit()

    conn.close()
    groupBox = QtWidgets.QGroupBox()
    groupBox.setMaximumWidth(940)
    groupBox.setStyleSheet('''QGroupBox {margin-top: 2px; }
QGroupBox:enabled {border: 2px solid grey;
}QGroupBox::title {subcontrol-origin: margin; left: 3ex; } ''')
    cb = QtWidgets.QCheckBox('Добавить файл')
    self.check_boxes.append(cb)
    lnm = QtWidgets.QLineEdit()
    self.ln_pathes.append(lnm)
    lnm.setEnabled(False)
    btnm1 = QtWidgets.QPushButton('...')
    btnm1.setMinimumWidth(50)
    self.btn_picks.append(btnm1)

    cb.stateChanged.connect(lambda checked,
command_id=self.n: self.empty_check_cganges(command_id))
    lnm.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_cganges(command_id))
    btnm1.clicked.connect(lambda checked, command_id=self.n:
self.pick_changes(command_id))

    tem = QtWidgets.QTextEdit(self.tem.toPlainText())
    tem.textChanged.connect(lambda command_id=self.n:
self.empty_check_cganges(command_id))
    tem.setPlaceholderText('Введите текст ответа')
    tem.setMaximumHeight(90)
    self.textes.append(tem)

    hlm = QtWidgets.QHBoxLayout()
    hlm.addWidget(cb)
    hlm.addWidget(lnm)
    hlm.addWidget(btnm1)

    btnm3 = QtWidgets.QPushButton('Сохранить')
    self.btns_changes.append(btnm3)

```

```

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

        with conn.cursor() as cur:

            cur.execute("""SELECT MAX(id) FROM answears WHERE
user_id=%s AND bot_name=%s""",
                        (self.user_id, self.bot_name))
            max_number = cur.fetchone()
            max_number = max_number[0]

        conn.close()

        btnm3.clicked.connect(lambda checked, command_id=self.n,
id=id: self.save_changes(command_id, max_number))
        add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
        add_comand_btn_hl2.addWidget(btnm3)
        self.n += 1

        add_menu_vl = QtWidgets.QVBoxLayout(groupBox)
        n = 1
        for cb in self.add_answer_combo_boxes:
            label = QtWidgets.QLabel(f'Вопрос номер {n}')
            combo_box = QtWidgets.QComboBox()
            combo_box.addItem(cb.currentText())
            hl = QtWidgets.QHBoxLayout()
            hl.addWidget(label)
            hl.addWidget(combo_box)
            add_menu_vl.addLayout(hl)
            n+=1

        add_menu_vl.addWidget(tem)
        add_menu_vl.addLayout(hlm)
        add_menu_vl.addLayout(add_comand_btn_hl2)

        self.bots_list.insertWidget(self.bots_list.count()-3,
groupBox)
        self.groupBox.hide()
        self.add_menu_btn.show()
        def add_answer(self):
            self.groupBox.show()
            self.add_menu_btn.hide()
        def add_answer_cancel(self):
            self.groupBox.hide()
            self.add_menu_btn.show()

Файл «constructor_buttons.py»
from PyQt6 import QtWidgets, QtCore, QtGui
import psycopg2

from dbacc import *
class Constructor_buttons(QtWidgets.QWidget):

```

```

def __init__(self, bot_name, user_id,
constructor_questions_class):
    QtWidgets.QWidget.__init__(self)
    self.bot_name = bot_name
    self.user_id = user_id
    self.constructor_questions_class =
constructor_questions_class
    self.n = 0
    self.menu_names = []
    self.counters = []
    self.massives = []
    self.btns = []
    self.setStyleSheet("""QPushButton{ border-radius:9px;
height:25px; background-color:#7d17d1; }
QPushButton:hover{
background-color:#9127e8; }
QLineEdit{ background-color:#b3adad; border-radius:9px;
height:22px; }
QTextEdit{ background-color:#b3adad; border-radius:9px; }""")

    self.bots_list = QtWidgets.QVBoxLayout()

    self.gb = QtWidgets.QGroupBox()
    self.gb.setLayout(self.bots_list)

    self.scroll_area = QtWidgets.QScrollArea()
    self.scroll_area.setWidgetResizable(True)
    self.scroll_area.setWidget(self.gb)
    self.scroll_area.setMaximumHeight(450)

    self.info_hl = QtWidgets.QHBoxLayout()
    self.info_label = QtWidgets.QLabel('Здесь ты сможешь
создавать и редактировать клавиатуры кнопок для вопросов')
    self.info_label.setMaximumHeight(15)

    self.info_hl.addStretch()
    self.info_hl.addWidget(self.info_label)
    self.info_hl.addStretch()

    self.bots_list.addLayout(self.info_hl)

    self.groupBox = QtWidgets.QGroupBox()
    self.groupBox.setMaximumWidth(940)
    self.groupBox.setStyleSheet('''
QGroupBox {margin-top: 5px; }
QGroupBox:enabled {border: 2px solid black; }
QGroupBox::title {subcontrol-origin: margin; left: 3ex; }''')
    self.btn_menu_name = QtWidgets.QLineEdit()
    self.btn_menu_name.textChanged.connect(self.empty_check)
    self.btn_menu_name.setPlaceholderText('Введите название
меню')

```

```

        self.btn_counter_lb = QtWidgets.QLabel('Выберите
количество кнопок')
        self.btn_counter = QtWidgets.QSpinBox()
        self.btn_counter.setValue(2)

self.btn_counter.valueChanged.connect(self.btn_counter_function)
        self.btn_counter.setMinimumWidth(35)
        self.btn_counter.setMinimum(2)
        self.btn_counter.setMaximum(10)
        self.btn_counter_hl = QtWidgets.QHBoxLayout()
        self.btn_counter_hl.addWidget(self.btn_menu_name)
        self.btn_counter_hl.addWidget(self.btn_counter_lb)
        self.btn_counter_hl.addWidget(self.btn_counter)

        self.ln_name1 = QtWidgets.QLineEdit()
        self.ln_name1.textChanged.connect(self.empty_check)
        self.ln_name1.setPlaceholderText('Введите название
кнопки')

        self.ln_name2 = QtWidgets.QLineEdit()
        self.ln_name2.textChanged.connect(self.empty_check)
        self.ln_name2.setPlaceholderText('Введите название
кнопки')

        self.ln_name3 = QtWidgets.QLineEdit()
        self.ln_name3.textChanged.connect(self.empty_check)
        self.ln_name3.setPlaceholderText('Введите название
кнопки')
        self.ln_name3.hide()

        self.ln_name4 = QtWidgets.QLineEdit()
        self.ln_name4.textChanged.connect(self.empty_check)
        self.ln_name4.setPlaceholderText('Введите название
кнопки')
        self.ln_name4.hide()

        self.ln_name5 = QtWidgets.QLineEdit()
        self.ln_name5.textChanged.connect(self.empty_check)
        self.ln_name5.setPlaceholderText('Введите название
кнопки')
        self.ln_name5.hide()

        self.ln_name6 = QtWidgets.QLineEdit()
        self.ln_name6.textChanged.connect(self.empty_check)
        self.ln_name6.setPlaceholderText('Введите название
кнопки')
        self.ln_name6.hide()

        self.ln_name7 = QtWidgets.QLineEdit()
        self.ln_name7.textChanged.connect(self.empty_check)
        self.ln_name7.setPlaceholderText('Введите название
кнопки')

```

```

self.ln_name7.hide()

self.ln_name8 = QtWidgets.QLineEdit()
self.ln_name8.textChanged.connect(self.empty_check)
self.ln_name8.setPlaceholderText('Введите название
кнопки')
self.ln_name8.hide()

self.ln_name9 = QtWidgets.QLineEdit()
self.ln_name9.textChanged.connect(self.empty_check)
self.ln_name9.setPlaceholderText('Введите название
кнопки')
self.ln_name9.hide()

self.ln_name10 = QtWidgets.QLineEdit()
self.ln_name10.textChanged.connect(self.empty_check)
self.ln_name10.setPlaceholderText('Введите название
кнопки')
self.ln_name10.hide()

self.btnm3 = QtWidgets.QPushButton('Добавить')
self.btnm3.clicked.connect(self.add_menu_save)
self.btnm3.setEnabled(False)
self.btnm4 = QtWidgets.QPushButton('Отмена')
self.btnm4.clicked.connect(self.add_menu_cancel)
self.add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
self.add_comand_btn_hl2.addWidget(self.btnm3)
self.add_comand_btn_hl2.addWidget(self.btnm4)

self.add_menu_vl = QtWidgets.QVBoxLayout(self.groupBox)
self.add_menu_vl.addLayout(self.btn_counter_hl)
self.add_menu_vl.addWidget(self.ln_name1)
self.add_menu_vl.addWidget(self.ln_name2)
self.add_menu_vl.addWidget(self.ln_name3)
self.add_menu_vl.addWidget(self.ln_name4)
self.add_menu_vl.addWidget(self.ln_name5)
self.add_menu_vl.addWidget(self.ln_name6)
self.add_menu_vl.addWidget(self.ln_name7)
self.add_menu_vl.addWidget(self.ln_name8)
self.add_menu_vl.addWidget(self.ln_name9)
self.add_menu_vl.addWidget(self.ln_name10)
self.add_menu_vl.addLayout(self.add_comand_btn_hl2)

self.add_menu_btn = QtWidgets.QPushButton('Добавить меню
кнопок')
self.add_menu_btn.clicked.connect(self.add_menu)
self.add_menu_btn.setMinimumHeight(25)
self.add_menu_btn.setMinimumWidth(150)

self.add_menu_btn_hl = QtWidgets.QHBoxLayout()
self.add_menu_btn_hl.addStretch()

```

```

self.add_menu_btn_hl.addWidget(self.add_menu_btn)
self.add_menu_btn_hl.addStretch()

conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
with conn.cursor() as cur:
    cur.execute("""SELECT id, menu_name, btns_count,
btn1, btn2,btn3,btn4,btn5,btn6,btn7,btn8,btn9,btn10 FROM buttons
WHERE user_id=%s AND bot_name=%s""",
                (self.user_id, self.bot_name))
    comands = cur.fetchall()

    for id, menu_name, btns_count, btn1,
btn2,btn3,btn4,btn5,btn6,btn7,btn8,btn9,btn10 in comands:
        groupBox = QtWidgets.QGroupBox()
        groupBox.setMaximumWidth(940)
        groupBox.setStyleSheet('''QGroupBox {margin-top:
2px; }
QGroupBox:enabled {border: 2px solid grey; }
QGroupBox::title {subcontrol-origin: margin; left: 3ex;}''')
        btn_menu_name = QtWidgets.QLineEdit(menu_name)
        self.menu_names.append(btn_menu_name)
        btn_menu_name.setPlaceholderText('Введите
название меню')
        btn_counter_lb = QtWidgets.QLabel('Выберите
количество кнопок')
        btn_counter = QtWidgets.QSpinBox()
        btn_counter.setValue(btns_count)
        btn_counter.setMinimum(2)
        btn_counter.setMaximum(10)
        self.counters.append(btn_counter)
        btn_counter.valueChanged.connect(lambda checked,
command_id=self.n: self.btn_counter_changes(command_id))
        btn_counter.setMinimumWidth(35)
        btn_counter_hl = QtWidgets.QHBoxLayout()
        btn_counter_hl.addWidget(btn_menu_name)
        btn_counter_hl.addWidget(btn_counter_lb)
        btn_counter_hl.addWidget(btn_counter)

        massive = []

        ln_name1 = QtWidgets.QLineEdit(btn1)
        massive.append(ln_name1)
        ln_name1.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name1.setPlaceholderText('Введите название
кнопки')

        ln_name2 = QtWidgets.QLineEdit(btn2)
        massive.append(ln_name2)
        ln_name2.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))

```

```

        ln_name2.setPlaceholderText('Введите название
кнопки')

        ln_name3 = QtWidgets.QLineEdit(btn3)
        massive.append(ln_name3)
        ln_name3.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name3.setPlaceholderText('Введите название
кнопки')

        ln_name3.hide()

        ln_name4 = QtWidgets.QLineEdit(btn4)
        massive.append(ln_name4)
        ln_name4.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name4.setPlaceholderText('Введите название
кнопки')

        ln_name4.hide()

        ln_name5 = QtWidgets.QLineEdit(btn5)
        massive.append(ln_name5)
        ln_name5.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name5.setPlaceholderText('Введите название
кнопки')

        ln_name5.hide()

        ln_name6 = QtWidgets.QLineEdit(btn6)
        massive.append(ln_name6)
        ln_name6.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name6.setPlaceholderText('Введите название
кнопки')

        ln_name6.hide()

        ln_name7 = QtWidgets.QLineEdit(btn7)
        massive.append(ln_name7)
        ln_name7.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name7.setPlaceholderText('Введите название
кнопки')

        ln_name7.hide()

        ln_name8 = QtWidgets.QLineEdit(btn8)
        massive.append(ln_name8)
        ln_name8.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name8.setPlaceholderText('Введите название
кнопки')

        ln_name8.hide()

        ln_name9 = QtWidgets.QLineEdit(btn9)

```

```

        massive.append(ln_name9)
        ln_name9.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name9.setPlaceholderText('Введите название
кнопки')

        ln_name9.hide()

        ln_name10 = QtWidgets.QLineEdit(btn10)
        massive.append(ln_name10)
        ln_name10.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name10.setPlaceholderText('Введите название
кнопки')

        ln_name10.hide()

        self.massives.append(massive)

        if btns_count > 2:
            ln_name3.show()
        if btns_count > 3:
            ln_name4.show()
        if btns_count > 4:
            ln_name5.show()
        if btns_count > 5:
            ln_name6.show()
        if btns_count > 6:
            ln_name7.show()
        if btns_count > 7:
            ln_name8.show()
        if btns_count > 8:
            ln_name9.show()
        if btns_count > 9:
            ln_name10.show()

        btnm3 = QtWidgets.QPushButton('Сохранить')
        btnm3.clicked.connect(lambda checked,
command_id=self.n, id=id: self.save_changes(command_id, id))
        self.btns.append(btnm3)
        add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
        add_comand_btn_hl2.addWidget(btnm3)
        self.n += 1

        add_menu_vl = QtWidgets.QVBoxLayout(groupBox)
        add_menu_vl.addLayout(btn_counter_hl)
        add_menu_vl.addWidget(ln_name1)
        add_menu_vl.addWidget(ln_name2)
        add_menu_vl.addWidget(ln_name3)
        add_menu_vl.addWidget(ln_name4)
        add_menu_vl.addWidget(ln_name5)
        add_menu_vl.addWidget(ln_name6)
        add_menu_vl.addWidget(ln_name7)
        add_menu_vl.addWidget(ln_name8)

```

```

        add_menu_vl.addWidget(ln_name9)
        add_menu_vl.addWidget(ln_name10)

        add_menu_vl.addLayout(add_comand_btn_hl2)
        self.bots_list.addWidget(groupBox)
    conn.close()

    self.bots_list.addLayout(self.add_menu_btn_hl)
    self.bots_list.addWidget(self.groupBox)

    self.groupBox.hide()

    self.main_vl = QtWidgets.QVBoxLayout()
    self.main_vl.addWidget(self.scroll_area)
    self.bots_list.addStretch()

    self.setLayout(self.main_vl)
    def empty_check_changes(self, comand_id):
        count = self.counters[comand_id].value()
        if self.menu_names[comand_id].text() == '':
            self.btns[comand_id].setEnabled(False)
        else:
            if count == 2:
                if self.massives[comand_id][0].text() == '' or
self.massives[comand_id][1].text() == '':
                    self.btns[comand_id].setEnabled(False)
                else:
                    self.btns[comand_id].setEnabled(True)
            elif count == 3:
                if self.massives[comand_id][0].text() == '' or
self.massives[comand_id][1].text() == '' or
self.massives[comand_id][2].text() == '':
                    self.btns[comand_id].setEnabled(False)
                else:
                    self.btns[comand_id].setEnabled(True)
            elif count == 4:
                if self.massives[comand_id][0].text() == '' or
self.massives[comand_id][1].text() == '' or
self.massives[comand_id][2].text() == '' or
self.massives[comand_id][3].text() == '':
                    self.btns[comand_id].setEnabled(False)
                else:
                    self.btns[comand_id].setEnabled(True)
            elif count == 5:
                if self.massives[comand_id][0].text() == '' or
self.massives[comand_id][1].text() == '' or
self.massives[comand_id][2].text() == '' or
self.massives[comand_id][3].text() == ''\
                    or self.massives[comand_id][4].text() ==
'':
                    self.btns[comand_id].setEnabled(False)
                else:

```

```

        self.btns[comand_id].setEnabled(True)
    elif count == 6:
        if self.massives[comand_id][0].text() == '' or
self.massives[comand_id][1].text() == '' or
self.massives[comand_id][2].text() == '' or
self.massives[comand_id][3].text() == '' \
        or self.massives[comand_id][4].text() ==
'' or self.massives[comand_id][5].text() == '':
            self.btns[comand_id].setEnabled(False)
        else:
            self.btns[comand_id].setEnabled(True)
    elif count == 7:
        if self.massives[comand_id][0].text() == '' or
self.massives[comand_id][1].text() == '' or
self.massives[comand_id][2].text() == '' or
self.massives[comand_id][3].text() == '' \
        or self.massives[comand_id][4].text() ==
'' or self.massives[comand_id][5].text() == '' or
self.massives[comand_id][6].text() == '':
            self.btns[comand_id].setEnabled(False)
        else:
            self.btns[comand_id].setEnabled(True)
    elif count == 8:
        if self.massives[comand_id][0].text() == '' or
self.massives[comand_id][1].text() == '' or
self.massives[comand_id][2].text() == '' or
self.massives[comand_id][3].text() == '' \
        or self.massives[comand_id][4].text() ==
'' or self.massives[comand_id][5].text() == '' or
self.massives[comand_id][6].text() == '' \
        or self.massives[comand_id][7].text() ==
'':
            self.btns[comand_id].setEnabled(False)
        else:
            self.btns[comand_id].setEnabled(True)
    elif count == 9:
        if self.massives[comand_id][0].text() == '' or
self.massives[comand_id][1].text() == '' or
self.massives[comand_id][2].text() == '' or
self.massives[comand_id][3].text() == '' \
        or self.massives[comand_id][4].text() ==
'' or self.massives[comand_id][5].text() == '' or
self.massives[comand_id][6].text() == '' \
        or self.massives[comand_id][7].text() ==
'' or self.massives[comand_id][8].text() == '':
            self.btns[comand_id].setEnabled(False)
        else:
            self.btns[comand_id].setEnabled(True)
    elif count == 10:
        if self.massives[comand_id][0].text() == '' or
self.massives[comand_id][1].text() == '' or

```

```

self.massives[comand_id][2].text() == '' or
self.massives[comand_id][3].text() == '' \
    or self.massives[comand_id][4].text() ==
'' or self.massives[comand_id][5].text() == '' or
self.massives[comand_id][6].text() == '' \
    or self.massives[comand_id][7].text() ==
'' or self.massives[comand_id][8].text() == '' or
self.massives[comand_id][9].text() == '':
    self.btns[comand_id].setEnabled(False)
else:
    self.btns[comand_id].setEnabled(True)
def empty_check(self):
    count = self.btn_counter.value()
    if self.btn_menu_name.text() == '':
        self.btnm3.setEnabled(False)
    else:
        if count == 2:
            if self.ln_name1.text() == '' or
self.ln_name2.text() == '':
                self.btnm3.setEnabled(False)
            else:
                self.btnm3.setEnabled(True)
        elif count == 3:
            if self.ln_name1.text() == '' or
self.ln_name2.text() == '' or self.ln_name3.text() == '':
                self.btnm3.setEnabled(False)
            else:
                self.btnm3.setEnabled(True)
        elif count == 4:
            if self.ln_name1.text() == '' or
self.ln_name2.text() == '' or self.ln_name3.text() == '' or
self.ln_name4.text() == '':
                self.btnm3.setEnabled(False)
            else:
                self.btnm3.setEnabled(True)
        elif count == 5:
            if self.ln_name1.text() == '' or
self.ln_name2.text() == '' or self.ln_name3.text() == '' or
self.ln_name4.text() == '' \
                or self.ln_name5.text() == '':
                self.btnm3.setEnabled(False)
            else:
                self.btnm3.setEnabled(True)
        elif count == 6:
            if self.ln_name1.text() == '' or
self.ln_name2.text() == '' or self.ln_name3.text() == '' or
self.ln_name4.text() == '' \
                or self.ln_name5.text() == '' or
self.ln_name6.text() == '':
                self.btnm3.setEnabled(False)
            else:
                self.btnm3.setEnabled(True)

```

```

        elif count == 7:
            if self.ln_name1.text() == '' or
self.ln_name2.text() == '' or self.ln_name3.text() == '' or
self.ln_name4.text() == '' \
            or self.ln_name5.text() == '' or
self.ln_name6.text() == '' or self.ln_name7.text() == '':
                self.btnm3.setEnabled(False)
            else:
                self.btnm3.setEnabled(True)
        elif count == 8:
            if self.ln_name1.text() == '' or
self.ln_name2.text() == '' or self.ln_name3.text() == '' or
self.ln_name4.text() == '' \
            or self.ln_name5.text() == '' or
self.ln_name6.text() == '' or self.ln_name7.text() == '' \
            or self.ln_name8.text() == '':
                self.btnm3.setEnabled(False)
            else:
                self.btnm3.setEnabled(True)
        elif count == 9:
            if self.ln_name1.text() == '' or
self.ln_name2.text() == '' or self.ln_name3.text() == '' or
self.ln_name4.text() == '' \
            or self.ln_name5.text() == '' or
self.ln_name6.text() == '' or self.ln_name7.text() == '' \
            or self.ln_name8.text() == '' or
self.ln_name9.text() == '':
                self.btnm3.setEnabled(False)
            else:
                self.btnm3.setEnabled(True)
        elif count == 10:
            if self.ln_name1.text() == '' or
self.ln_name2.text() == '' or self.ln_name3.text() == '' or
self.ln_name4.text() == '' \
            or self.ln_name5.text() == '' or
self.ln_name6.text() == '' or self.ln_name7.text() == '' \
            or self.ln_name8.text() == '' or
self.ln_name9.text() == '' or self.ln_name10.text() == '':
                self.btnm3.setEnabled(False)
            else:
                self.btnm3.setEnabled(True)

    def save_changes(self, comand_id, id):

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

        with conn.cursor() as cur:
            cur.execute("""UPDATE buttons SET menu_name=%s,
btns_count=%s, btn1=%s, btn2=%s, btn3=%s, btn4=%s, btn5=%s,
btn6=%s, btn7=%s, btn8=%s, btn9=%s, btn10=%s WHERE id=%s""", \

```

```

        (self.menu_names[comand_id].text(),
self.counters[comand_id].value(),
self.massives[comand_id][0].text(),self.massives[comand_id][1].t
ext(),
        self.massives[comand_id][2].text(),
self.massives[comand_id][3].text(),self.massives[comand_id][4].t
ext(),self.massives[comand_id][5].text(),

self.massives[comand_id][6].text(),self.massives[comand_id][7].t
ext(),self.massives[comand_id][8].text(),
        self.massives[comand_id][9].text(),
id))

        conn.commit()
        conn.commit()

self.constructor_questions_class.add_btn_menu(self.btn_menu_name
.text())
        conn.close()
def add_menu_save(self):

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

        with conn.cursor() as cur:
            cur.execute(f"""INSERT INTO buttons (user_id,
bot_name, menu_name, btns_count, btn1, btn2, btn3, btn4, btn5,
            btn6, btn7, btn8, btn9, btn10) VALUES
            (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s) """,
            (self.user_id, self.bot_name,
self.btn_menu_name.text(), self.btn_counter.value(),
self.ln_name1.text()
            , self.ln_name2.text() , self.ln_name3.text() ,
self.ln_name4.text() , self.ln_name5.text() ,
self.ln_name6.text()
            , self.ln_name7.text() , self.ln_name8.text() ,
self.ln_name9.text() , self.ln_name10.text()))
            conn.commit()
self.constructor_questions_class.add_btn_menu(self.btn_menu_name
.text())
        conn.close()

        groupBox = QtWidgets.QGroupBox()
        groupBox.setMaximumWidth(940)
        groupBox.setStyleSheet('''
QGroupBox {margin-top: 2px;}
QGroupBox:enabled {
border: 2px solid grey;}
QGroupBox::title {
subcontrol-origin: margin;
left: 3ex;
}
''')

```

```

        btn_menu_name =
QtWidgets.QLineEdit(self.btn_menu_name.text())
        self.menu_names.append(btn_menu_name)
        btn_counter_lb = QtWidgets.QLabel('Выберите количество
кнопок')
        btn_counter = QtWidgets.QSpinBox()
        btn_counter.setValue(btn_counter.value())
        btn_counter.valueChanged.connect(lambda checked,
command_id=self.n: self.btn_counter_changes(command_id))
        self.counters.append(btn_counter)
        btn_counter.setMinimumWidth(35)
        btn_counter_hl = QtWidgets.QHBoxLayout()
        btn_counter_hl.addWidget(btn_menu_name)
        btn_counter_hl.addWidget(btn_counter_lb)
        btn_counter_hl.addWidget(btn_counter)

        count = self.btn_counter.value()

        massive = []

        ln_name1 = QtWidgets.QLineEdit(self.ln_name1.text())
        massive.append(ln_name1)
        ln_name1.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name1.setPlaceholderText('Введите название кнопки')

        ln_name2 = QtWidgets.QLineEdit(self.ln_name2.text())
        massive.append(ln_name2)
        self.ln_name2.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name2.setPlaceholderText('Введите название кнопки')

        ln_name3 = QtWidgets.QLineEdit(self.ln_name3.text())
        ln_name3.hide()
        massive.append(ln_name3)
        self.ln_name3.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name3.setPlaceholderText('Введите название кнопки')

        ln_name4 = QtWidgets.QLineEdit(self.ln_name4.text())
        massive.append(ln_name4)
        ln_name4.hide()
        self.ln_name4.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
        ln_name4.setPlaceholderText('Введите название кнопки')

        ln_name5 = QtWidgets.QLineEdit(self.ln_name5.text())
        massive.append(ln_name5)
        ln_name5.hide()
        self.ln_name5.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))

```

```

ln_name5.setPlaceholderText('Введите название кнопки')

ln_name6 = QtWidgets.QLineEdit(self.ln_name6.text())
massive.append(ln_name6)
ln_name6.hide()
self.ln_name6.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
ln_name6.setPlaceholderText('Введите название кнопки')

ln_name7 = QtWidgets.QLineEdit(self.ln_name7.text())
massive.append(ln_name7)
ln_name7.hide()
self.ln_name7.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
ln_name7.setPlaceholderText('Введите название кнопки')

ln_name8 = QtWidgets.QLineEdit(self.ln_name8.text())
massive.append(ln_name8)
ln_name8.hide()
self.ln_name8.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
ln_name8.setPlaceholderText('Введите название кнопки')

ln_name9 = QtWidgets.QLineEdit(self.ln_name9.text())
massive.append(ln_name9)
ln_name9.hide()
self.ln_name9.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
ln_name9.setPlaceholderText('Введите название кнопки')

ln_name10 = QtWidgets.QLineEdit(self.ln_name10.text())
massive.append(ln_name10)
ln_name10.hide()
self.ln_name10.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_changes(command_id))
ln_name10.setPlaceholderText('Введите название кнопки')

self.massives.append(massive)

add_menu_vl = QtWidgets.QVBoxLayout(groupBox)
add_menu_vl.addLayout(btn_counter_hl)

add_menu_vl.addWidget(ln_name1)
add_menu_vl.addWidget(ln_name2)
add_menu_vl.addWidget(ln_name3)
add_menu_vl.addWidget(ln_name4)
add_menu_vl.addWidget(ln_name5)
add_menu_vl.addWidget(ln_name6)
add_menu_vl.addWidget(ln_name7)
add_menu_vl.addWidget(ln_name8)
add_menu_vl.addWidget(ln_name9)
add_menu_vl.addWidget(ln_name10)

```

```

        if count > 2:
            ln_name3.show()
        if count > 3:
            ln_name4.show()
        if count > 4:
            ln_name5.show()
        if count > 5:
            ln_name6.show()
        if count > 6:
            ln_name7.show()
        if count > 7:
            ln_name8.show()
        if count > 8:
            ln_name9.show()
        if count > 9:
            ln_name10.show()

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

        with conn.cursor() as cur:

            cur.execute("""SELECT MAX(id) FROM buttons WHERE
user_id=%s AND bot_name=%s""",
                        (self.user_id, self.bot_name))
            max_number = cur.fetchone()
            max_number = max_number[0]

        conn.close()

        btnm3 = QtWidgets.QPushButton('Сохранить')
        btnm3.setEnabled(False)
        btnm3.clicked.connect(lambda checked, command_id=self.n,
id=id: self.save_changes(command_id, max_number))
        add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
        add_comand_btn_hl2.addWidget(btnm3)
        self.n += 1

        self.groupBox.hide()
        self.add_menu_btn.show()
        add_menu_vl.addLayout(add_comand_btn_hl2)

        self.bots_list.insertWidget(self.bots_list.count()-3,
groupBox)
        def btn_counter_changes(self, comand_id):
            count = self.counters[comand_id].value()
            try:
                if count == 2:
                    self.massives[comand_id][2].hide()
                elif count == 3 | 4 | 5 | 6 | 7 | 8 | 9:
                    self.massives[comand_id][count - 1].show()

```

```

        self.massives[comand_id][count].hide()
    elif count == 10:
        self.massives[comand_id][9].show()

        self.empty_check_changes(comand_id)
except:
    pass
def btn_counter_function(self):
    count = self.btn_counter.value()
    self.btm3.setEnabled(False)

    try:
        if count == 2:
            self.ln_name3.hide()
        elif count == 3:
            self.ln_name3.show()
            self.ln_name4.hide()
        elif count == 4:
            self.ln_name4.show()
            self.ln_name5.hide()
        elif count == 5:
            self.ln_name5.show()
            self.ln_name6.hide()
        elif count == 6:
            self.ln_name6.show()
            self.ln_name7.hide()
        elif count == 7:
            self.ln_name7.show()
            self.ln_name8.hide()
        elif count == 8:
            self.ln_name8.show()
            self.ln_name9.hide()
        elif count == 9:
            self.ln_name9.show()
            self.ln_name10.hide()
        elif count == 10:
            self.ln_name10.show()
    except:
        pass
def add_menu_cancel(self):
    self.groupBox.hide()
    self.add_menu_btn.show()
def add_menu(self):
    self.add_menu_btn.hide()
    self.groupBox.show()

```

Файл «*constructor_commands.py*»

```

import shutil

from PyQt6 import QtWidgets, QtCore, QtGui
import psycpg2

```

```

import os

from dbacc import *
class Constructor_start_command(QtWidgets.QWidget):
    def __init__(self, bot_name, user_id):
        QtWidgets.QWidget.__init__(self)
        self.bot_name = bot_name
        self.user_id = user_id

        self.setStyleSheet("""QPushButton{border-radius:9px;
height:25px;background-color:#7d17d1;}
                        QPushButton:hover{background-
color:#9127e8;}
                        QLineEdit{background-color:#b3adad; border-
radius:9px; height:22px; }
QTextEdit{background-color:#b3adad; border-radius:9px; }""")
        self.n = 0
        self.comand_names = []
        self.comand_textes = []
        self.check_boxes = []
        self.lnpathes = []
        self.btn_picks = []
        self.changes_btn = []
        self.bots_list = QtWidgets.QVBoxLayout()

        self.gb = QtWidgets.QGroupBox()
        self.gb.setLayout(self.bots_list)

        self.scroll_area = QtWidgets.QScrollArea()
        self.scroll_area.setWidgetResizable(True)
        self.scroll_area.setWidget(self.gb)
        self.scroll_area.setMaximumHeight(475)

        self.info_hl = QtWidgets.QHBoxLayout()
        self.info_label = QtWidgets.QLabel('Здесь вы сможете
добавлять и редактировать команды вашего бота')
        self.info_label.setMaximumHeight(15)

        self.info_hl.addStretch()
        self.info_hl.addWidget(self.info_label)
        self.info_hl.addStretch()

        self.groupBox2 = QtWidgets.QGroupBox()
        self.groupBox2.setMaximumWidth(940)
        self.groupBox2.setStyleSheet('''QGroupBox {margin-top:
5px;}
QGroupBox:enabled {border: 2px solid black;}
QGroupBox::title {subcontrol-origin: margin; left: 3ex; }''')

        self.comand_name_ln2 = QtWidgets.QLineEdit()

        self.comand_name_ln2.textChanged.connect(self.empty_check2)

```

```

        regex = QtCore.QRegularExpression("/+[a-zA-Z_]+")
        validator = QtGui.QRegularExpressionValidator(regex)
        self.comand_name_ln2.setValidator(validator)
        self.comand_name_ln2.setPlaceholderText('Введите
название команды')

        self.cb2 = QtWidgets.QCheckBox('Добавить файл')
        self.cb2.stateChanged.connect(self.empty_check2)
        self.lnm2 = QtWidgets.QLineEdit()
        self.lnm2.setEnabled(False)
        self.lnm2.textChanged.connect(self.empty_check2)
        self.btm2 = QtWidgets.QPushButton('...')
        self.btm2.setMinimumWidth(50)
        self.btm2.clicked.connect(self.pick2)

        self.tem2 = QtWidgets.QTextEdit()
        self.tem2.setPlaceholderText('Введите текст который
будет отправляться по команде')
        self.tem2.setMaximumHeight(100)
        self.tem2.textChanged.connect(self.empty_check2)

        self.hlm2 = QtWidgets.QHBoxLayout()
        self.hlm2.addWidget(self.cb2)
        self.hlm2.addWidget(self.lnm2)
        self.hlm2.addWidget(self.btm2)

        self.btm3 = QtWidgets.QPushButton('Добавить')
        self.btm3.clicked.connect(self.add_comand_save)
        self.btm3.setEnabled(False)
        self.btm4 = QtWidgets.QPushButton('Отмена')
        self.btm4.clicked.connect(self.add_comand_cancel)
        self.add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
        self.add_comand_btn_hl2.addWidget(self.btm3)
        self.add_comand_btn_hl2.addWidget(self.btm4)

        self.vlm2 = QtWidgets.QVBoxLayout(self.groupBox2)
        self.vlm2.addWidget(self.comand_name_ln2)
        self.vlm2.addWidget(self.tem2)
        self.vlm2.addLayout(self.hlm2)
        self.vlm2.addLayout(self.add_comand_btn_hl2)

        self.add_comand_btn = QtWidgets.QPushButton('Добавить
команду')
        self.add_comand_btn.clicked.connect(self.add_comand)
        self.add_comand_btn.setMinimumHeight(25)
        self.add_comand_btn.setMinimumWidth(150)
        self.add_comand_btn_hl = QtWidgets.QHBoxLayout()
        self.add_comand_btn_hl.addStretch()
        self.add_comand_btn_hl.addWidget(self.add_comand_btn)
        self.add_comand_btn_hl.addStretch()
        self.groupBox2.hide()

```

```

        self.bots_list.addLayout(self.info_h1)
        conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
        with conn.cursor() as cur:
            cur.execute("""SELECT id, comand_name,
comand_text,comand_photo FROM comands WHERE user_id=%s AND
bot_name=%s""",
                        (self.user_id, self.bot_name))
            comands = cur.fetchall()

        for id, comand_name, comand_text,comand_photo in
comands:
            groupBox = QtWidgets.QGroupBox()
            groupBox.setStyleSheet(''

QGroupBox {
margin-top: 2px;
}
QGroupBox:enabled {
border: 2px solid grey; }
QGroupBox::title {subcontrol-origin: margin; left: 3ex;
}')

            comand_name_ln2 =
QtWidgets.QLineEdit(comand_name)
            if comand_name == '/start':
                comand_name_ln2.setEnabled(False)

            self.comand_names.append(comand_name_ln2)
            comand_name_ln2.textChanged.connect(lambda
checked, command_id=self.n:
self.empty_check_cganges(command_id))
            regex = QtCore.QRegularExpression("/+[a-z-A-
Z_]+")
            validator =
QtGui.QRegularExpressionValidator(regex)
            comand_name_ln2.setValidator(validator)
            comand_name_ln2.setPlaceholderText('Введите
название команды')

            cb2 = QtWidgets.QCheckBox('Добавить файл')
            self.check_boxes.append(cb2)
            lnm2 = QtWidgets.QLineEdit()
            self.lnpathes.append(lnm2)
            lnm2.setEnabled(False)
            btnm2 = QtWidgets.QPushButton('...')
            btnm2.setMinimumWidth(50)
            self.btn_picks.append(btnm2)
            if comand_photo is not None:
                cb2.setChecked(True)
                lnm2.setText(comand_photo)

```

```

        cb2.stateChanged.connect(lambda checked,
command_id=self.n: self.empty_check_cganges(command_id))
        lnm2.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_cganges(command_id))
        btnm2.clicked.connect(lambda checked,
command_id=self.n: self.pick_changes(command_id))

        tem2 = QtWidgets.QTextEdit(comand_text)
        tem2.setPlaceholderText('Введите текст который
будет отправляться по команде')
        tem2.setMaximumHeight(100)
        tem2.textChanged.connect(lambda
command_id=self.n: self.empty_check_cganges(command_id))
        self.comand_textes.append(tem2)

        hlm2 = QtWidgets.QHBoxLayout()
        hlm2.addWidget(cb2)
        hlm2.addWidget(lnm2)
        hlm2.addWidget(btnm2)

        btnm3 = QtWidgets.QPushButton('Сохранить')
        btnm3.clicked.connect(lambda checked,
command_id=self.n, id=id: self.save_changes(command_id, id))
        btnm3.setEnabled(True)
        self.changes_btn.append(btnm3)
        self.n+=1
        add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
        add_comand_btn_hl2.addWidget(btnm3)

        vlm2 = QtWidgets.QVBoxLayout(groupBox)
        vlm2.addWidget(comand_name_ln2)
        vlm2.addWidget(tem2)
        vlm2.addLayout(hlm2)
        vlm2.addLayout(add_comand_btn_hl2)

        self.bots_list.addWidget(groupBox)

    conn.close()
    self.bots_list.addWidget(self.groupBox2)
    self.bots_list.addLayout(self.add_comand_btn_hl)

    self.main_vl = QtWidgets.QVBoxLayout()
    self.main_vl.addWidget(self.scroll_area)
    self.bots_list.addStretch()

    self.setLayout(self.main_vl)

    def save_changes(self, command_id, id):
        conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
        with conn.cursor() as cur:
            if self.check_boxes[command_id].isChecked():

```

```

        dirname =
os.path.dirname(os.path.abspath(__file__))
        dir_name = os.path.join(dirname,
f'{self.user_id}_{self.bot_name}')
        dir_name = os.path.join(dir_name, 'photos')
        shutil.copy(self.lnpathes[command_id].text(),
dir_name)

        save_path = os.path.join(dir_name,
self.lnpathes[command_id].text().split('/')[ -1])

        cur.execute("""UPDATE comands SET
comand_name=%s, comand_text=%s, comand_photo=%s WHERE
id=%s""", (self.comand_names[command_id].text(),
self.comand_textes[command_id].toPlainText(), save_path, id))
        conn.commit()
    else:
        cur.execute("""UPDATE comands SET
comand_name=%s, comand_text=%s WHERE
id=%s""", (self.comand_names[command_id].text(),
self.comand_textes[command_id].toPlainText(), id))
        conn.commit()
    def pick_changes(self, comand_id):
        print(comamd_id)
        self.filename = QtWidgets.QFileDialog.getOpenFileName()
        if self.filename:
            self.lnpathes[comamd_id].setText(self.filename[0])
    def empty_check_cganges(self, comand_id):
        if self.comand_names[comand_id].text() == '' or
self.comand_names[comand_id].text() == '/':
            self.changes_btn[comand_id].setEnabled(False)
        else:
            if self.comand_textes[comand_id].toPlainText() ==
':
                self.changes_btn[comand_id].setEnabled(False)
            else:
                if self.check_boxes[comand_id].isChecked():
                    if self.lnpathes[comand_id].text() == ':
self.changes_btn[comand_id].setEnabled(False)
                    else:
self.changes_btn[comand_id].setEnabled(True)
                else:
                    self.changes_btn[comand_id].setEnabled(True)
    def pick2(self):
        self.filename = QtWidgets.QFileDialog.getOpenFileName()
        if self.filename:
            self.lnm2.setText(self.filename[0])
    def empty_check2(self):
        if self.comand_name_ln2.text() == '' or
self.comand_name_ln2.text() == '/':

```

```

        self.btnm3.setEnabled(False)
    else:
        if self.tem2.toPlainText() == '':
            self.btnm3.setEnabled(False)
        else:
            if self.cb2.isChecked():
                if self.lnm2.text() == '':
                    self.btnm3.setEnabled(False)
                else:
                    self.btnm3.setEnabled(True)
            else:
                self.btnm3.setEnabled(True)
    def add_comand_save(self):
        conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
        with conn.cursor() as cur:
            if self.cb2.isChecked():
                dirname =
os.path.dirname(os.path.abspath(__file__))
                dir_name = os.path.join(dirname,
f'{self.user_id}_{self.bot_name}')
                dir_name = os.path.join(dir_name, 'photos')

                shutil.copy(self.lnm2.text(), dir_name)
                save_path = os.path.join(dir_name,
self.lnm2.text().split('/')[ -1])

                cur.execute(
                    f"""INSERT INTO comands (user_id, bot_name,
comand_name, comand_text, comand_photo) VALUES
(%s,%s,%s,%s,%s)""",
                    (self.user_id, self.bot_name,
self.comand_name_ln2.text(), self.tem2.toPlainText(),
save_path))
                conn.commit()
            else:
                cur.execute(f"""INSERT INTO comands (user_id,
bot_name, comand_name, comand_text) VALUES (%s,%s,%s,%s)""",
                    (self.user_id, self.bot_name,
self.comand_name_ln2.text(), self.tem2.toPlainText()))
                conn.commit()
        conn.close()

        groupBox = QtWidgets.QGroupBox()
        groupBox.setMaximumWidth(940)
        groupBox.setStyleSheet('''QGroupBox {margin-top: 2px;}
QGroupBox:enabled {
border: 2px solid grey;

}
QGroupBox::title {
subcontrol-origin: margin; left: 3ex;}

```

```

'''

comand_name_ln =
QtWidgets.QLineEdit(self.comand_name_ln2.text())
comand_name_ln.setPlaceholderText('Введите название
команды')
comand_name_ln.setEnabled(False)
self.comand_names.append(comand_name_ln)

cb = QtWidgets.QCheckBox('Добавить файл')
lnm = QtWidgets.QLineEdit()
lnm.setEnabled(False)
btnm1 = QtWidgets.QPushButton('...')
btnm1.setMinimumWidth(50)

if self.cb2.isChecked():
    cb.setChecked(True)
    lnm.setText(self.lnm2.text())

tem = QtWidgets.QTextEdit(self.tem2.toPlainText())
tem.setPlaceholderText('Введите текст который будет
отправляться по команде')
tem.setMaximumHeight(100)
self.comand_textes.append(tem)

hlm = QtWidgets.QHBoxLayout()
hlm.addWidget(cb)
hlm.addWidget(lnm)
hlm.addWidget(btnm1)

conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
with conn.cursor() as cur:
    cur.execute("""SELECT MAX(id) FROM comands WHERE
user_id=%s AND bot_name=%s""",
                (self.user_id, self.bot_name))
    max_number = cur.fetchone()
    max_number = max_number[0]
conn.close()

btnm1 = QtWidgets.QPushButton('Сохранить изменения')
btnm1.clicked.connect(lambda checked, command_id=self.n,
max_number=max_number: self.save_changes(command_id,
max_number))
self.n+=1

vlm = QtWidgets.QVBoxLayout(groupBox)
vlm.addWidget(comand_name_ln)
vlm.addWidget(tem)
vlm.addLayout(hlm)
vlm.addWidget(btnm1)

```

```

        self.groupBox2.hide()
        self.bots_list.insertWidget(self.bots_list.count()-3,
groupBox)
        self.add_comand_btn.show()

    try:
        self.comand_name_ln2.setEnabled(True)
    except:
        pass

    self.comand_name_ln2.setText('')
    self.tem2.setText('')
    self.cb2.setChecked(False)
    self.lnm2.setText('')
    def add_comand_cancel(self):
        self.groupBox2.hide()
        self.add_comand_btn.show()
    def add_comand(self):
        conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
        with conn.cursor() as cur:
            cur.execute("""SELECT * FROM comands WHERE
user_id=%s AND bot_name=%s""", (self.user_id, self.bot_name))
            all_comands = cur.fetchall()
            all_comands_count = len(all_comands)
            print(all_comands_count)
            if all_comands_count == 0:
                self.comand_name_ln2.setText('/start')
                self.comand_name_ln2.setEnabled(False)
        conn.close()
        self.add_comand_btn.hide()
        self.groupBox2.show()

```

Файл «*constructor_question.py*»

```

from PyQt6 import QtWidgets, QtCore, QtGui
import psycopg2
import os
import shutil

from dbacc import *
class Constructor_questions(QtWidgets.QWidget):
    def __init__(self, bot_name, user_id):
        QtWidgets.QWidget.__init__(self)
        self.bot_name = bot_name
        self.user_id = user_id
        self.n = 0

        self.question_textes = []
        self.check_boxes = []
        self.lnpathes = []
        self.combo_boxes = []

```

```

        self.btn_picks = []
        self.changes_btn = []

        self.setStyleSheet("""QPushButton{border-radius:9px;
height:25px; background-color:#7d17d1;}
                                QPushButton:hover{background-
color:#9127e8; }
QLineEdit{background-color:#b3adad; border-
radius:9px;height:22px;} QTextEdit{background-
color:#b3adad;border-radius:9px;}
QComboBox{background-color:#a38c8c;}""")
        self.bots_list = QtWidgets.QVBoxLayout()
        self.gb = QtWidgets.QGroupBox()
        self.gb.setLayout(self.bots_list)

        self.scroll_area = QtWidgets.QScrollArea()
        self.scroll_area.setWidgetResizable(True)
        self.scroll_area.setWidget(self.gb)
        self.scroll_area.setMaximumHeight(450)

        self.info_hl = QtWidgets.QHBoxLayout()
        self.info_label = QtWidgets.QLabel('Здесь ты сможешь
создавать и редактировать вопросы')
        self.info_label.setMaximumHeight(15)

        self.info_hl.addStretch()
        self.info_hl.addWidget(self.info_label)
        self.info_hl.addStretch()

        self.bots_list.addLayout(self.info_hl)
        self.groupBox = QtWidgets.QGroupBox()
        self.groupBox.setMaximumWidth(940)
        self.groupBox.setStyleSheet('''
QGroupBox {
margin-top: 5px;
}
QGroupBox:enabled {
border: 2px solid black;

}
QGroupBox::title {
subcontrol-origin: margin;
left: 3ex;
}''')

        self.cb = QtWidgets.QCheckBox('Добавить файл')
        self.cb.stateChanged.connect(self.empty_check1)
        self.lnm = QtWidgets.QLineEdit()
        self.lnm.setEnabled(False)
        self.lnm.textChanged.connect(self.empty_check1)
        self.btnm1 = QtWidgets.QPushButton('...')
        self.btnm1.setMinimumWidth(50)

```

```

self.btnm1.clicked.connect(self.pick1)

self.tem = QtWidgets.QTextEdit()
self.tem.setPlaceholderText('Введите текст вопроса')
self.tem.setMaximumHeight(90)
self.tem.textChanged.connect(self.empty_check1)

self.hlm = QtWidgets.QHBoxLayout()
self.hlm.addWidget(self.cb)
self.hlm.addWidget(self.lnm)
self.hlm.addWidget(self.btnm1)

self.combo_box_lb = QtWidgets.QLabel('Выберите
клавиатуру кнопок для этого вопроса')
self.combo_box = QtWidgets.QComboBox()

conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
with conn.cursor() as cur:
    cur.execute("""SELECT menu_name FROM buttons WHERE
user_id=%s AND bot_name=%s""",
                (self.user_id, self.bot_name))
    btn_menus = cur.fetchall()

    for btn_menu in btn_menus:
        self.combo_box.addItem(btn_menu[0])
conn.close()

self.combo_box_hl= QtWidgets.QHBoxLayout()
self.combo_box_hl.addWidget(self.combo_box_lb)
self.combo_box_hl.addWidget(self.combo_box)

self.btnm3 = QtWidgets.QPushButton('Добавить')
self.btnm3.clicked.connect(self.add_question_save)
self.btnm3.setEnabled(False)
self.btnm4 = QtWidgets.QPushButton('Отмена')
self.btnm4.clicked.connect(self.add_question_cancel)
self.add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
self.add_comand_btn_hl2.addWidget(self.btnm3)
self.add_comand_btn_hl2.addWidget(self.btnm4)

self.add_menu_vl = QtWidgets.QVBoxLayout(self.groupBox)
self.add_menu_vl.addWidget(self.tem)
self.add_menu_vl.addLayout(self.combo_box_hl)
self.add_menu_vl.addLayout(self.hlm)
self.add_menu_vl.addLayout(self.add_comand_btn_hl2)

self.add_menu_btn = QtWidgets.QPushButton('Добавить
вопрос')
self.add_menu_btn.clicked.connect(self.add_question)
self.add_menu_btn.setMinimumHeight(25)
self.add_menu_btn.setMinimumWidth(150)

```

```

self.add_menu_btn_hl = QtWidgets.QHBoxLayout()
self.add_menu_btn_hl.addStretch()
self.add_menu_btn_hl.addWidget(self.add_menu_btn)
self.add_menu_btn_hl.addStretch()

conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
with conn.cursor() as cur:
    cur.execute("""SELECT id, question_text, btn_menu,
question_photo FROM questions WHERE user_id=%s AND
bot_name=%s""",
                (self.user_id, self.bot_name))
    btn_menus = cur.fetchall()

for id, question_text, btn_menu, question_photo in
btn_menus:
    groupBox = QtWidgets.QGroupBox()
    groupBox.setMaximumWidth(940)
    groupBox.setStyleSheet('''
QGroupBox {
    margin-top: 2px;}
QGroupBox:enabled {
    border: 2px solid grey;}
QGroupBox::title {subcontrol-origin: margin; left: 3ex;}''')
    cb = QtWidgets.QCheckBox('Добавить фото')
    self.check_boxes.append(cb)
    lnm = QtWidgets.QLineEdit()
    self.lnpaths.append(lnm)
    lnm.setEnabled(False)
    btnm1 = QtWidgets.QPushButton('...')
    btnm1.setMinimumWidth(50)
    self.btn_picks.append(btnm1)

    if question_photo is not None:
        cb.setChecked(True)
        lnm.setText(question_photo)

    cb.stateChanged.connect(lambda checked,
command_id=self.n: self.empty_check_cganges(command_id))
    lnm.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_cganges(command_id))
    btnm1.clicked.connect(lambda checked,
command_id=self.n: self.pick_changes(command_id))

    tem = QtWidgets.QTextEdit(question_text)
    tem.textChanged.connect(lambda
command_id=self.n: self.empty_check_cganges(command_id))
    self.question_textes.append(tem)
    tem.setPlaceholderText('Введите текст вопроса')
    tem.setMaximumHeight(90)

```

```

        hlm = QtWidgets.QHBoxLayout()
        hlm.addWidget(cb)
        hlm.addWidget(lnm)
        hlm.addWidget(btnm1)

        combo_box_lb = QtWidgets.QLabel('Выберите
клавиатуру кнопок для этого вопроса')
        combo_box = QtWidgets.QComboBox()
        self.combo_boxes.append(combo_box)
        combo_box.addItem(btn_menu)

        conn = psycopg2.connect(database=data_base,
user=user, password=password, host=host, port=port)
        with conn.cursor() as cur:
            cur.execute("""SELECT menu_name FROM buttons
WHERE user_id=%s AND bot_name=%s""",
                        (self.user_id, self.bot_name))
            btn_menus = cur.fetchall()

            for btn_menu2 in btn_menus:
                if btn_menu2[0] != btn_menu:
                    combo_box.addItem(btn_menu2[0])
        conn.close()

        combo_box_hl = QtWidgets.QHBoxLayout()
        combo_box_hl.addWidget(combo_box_lb)
        combo_box_hl.addWidget(combo_box)

        btnm3 = QtWidgets.QPushButton('Сохранить')
        self.changes_btn.append(btnm3)
        btnm3.clicked.connect(lambda checked,
command_id=self.n, id=id: self.save_changes(command_id, id))
        add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
        add_comand_btn_hl2.addWidget(btnm3)
        self.n += 1

        add_menu_vl = QtWidgets.QVBoxLayout(groupBox)
        add_menu_vl.addWidget(tem)
        add_menu_vl.addLayout(combo_box_hl)
        add_menu_vl.addLayout(hlm)
        add_menu_vl.addLayout(add_comand_btn_hl2)

        self.bots_list.addWidget(groupBox)

    conn.close()

    self.bots_list.addLayout(self.add_menu_btn_hl)
    self.bots_list.addWidget(self.groupBox)

    self.groupBox.hide()

    self.main_vl = QtWidgets.QVBoxLayout()

```

```

        self.main_vl.addWidget(self.scroll_area)
        self.bots_list.addStretch()
        self.setLayout(self.main_vl)
    def save_changes(self, command_id, id):
        conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
        with conn.cursor() as cur:
            if self.check_boxes[command_id].isChecked():
                dirname =
os.path.dirname(os.path.abspath(__file__))
                dir_name = os.path.join(dirname,
f'{self.user_id}_{self.bot_name}')
                dir_name = os.path.join(dir_name, 'photos')

                shutil.copy(self.lnpathes[command_id].text(),
dir_name)

                save_path = os.path.join(dir_name,
self.lnpathes[command_id].text().split('/')[ -1])
                cur.execute("""UPDATE questions SET
question_text=%s, btn_menu=%s, question_photo=%s WHERE
id=%s""", (self.question_textes[command_id].toPlainText(),
self.combo_boxes[command_id].currentText(), save_path, id))
                conn.commit()
            else:

print(self.question_textes[command_id].toPlainText())

print(self.combo_boxes[command_id].currentText())
                cur.execute("""UPDATE questions SET
question_text=%s, btn_menu=%s WHERE
id=%s""", (self.question_textes[command_id].toPlainText(),
self.combo_boxes[command_id].currentText(), id))
                conn.commit()
        conn.close()
    def pick_changes(self, comand_id):
        self.filename = QtWidgets.QFileDialog.getOpenFileName()
        if self.filename:
            self.lnpathes[comamd_id].setText(self.filename[0])
    def empty_check_cganges(self, comand_id):
        if self.question_textes[comand_id].toPlainText() == '':
            self.changes_btn[comand_id].setEnabled(False)
        else:
            if self.check_boxes[comand_id].isChecked():
                if self.lnpathes[comand_id].text() == '':

self.changes_btn[comand_id].setEnabled(False)
            else:
                self.changes_btn[comand_id].setEnabled(True)
        else:
            self.changes_btn[comand_id].setEnabled(True)
    def pick1(self):
        self.filename = QtWidgets.QFileDialog.getOpenFileName()

```

```

        if self.filename:
            self.lnm.setText(self.filename[0])
def empty_check1(self):
    if self.tem.toPlainText() == '':
        self.btnm3.setEnabled(False)
    else:
        if self.cb.isChecked():
            if self.lnm.text() == '':
                self.btnm3.setEnabled(False)
            else:
                self.btnm3.setEnabled(True)
        else:
            self.btnm3.setEnabled(True)
def add_question_cancel(self):
    self.groupBox.hide()
    self.add_menu_btn.show()

def add_question_save(self):
    conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)
    with conn.cursor() as cur:
        if self.cb.isChecked():
            dirname =
os.path.dirname(os.path.abspath(__file__))
            dir_name = os.path.join(dirname,
f'{self.user_id}_{self.bot_name}')
            dir_name = os.path.join(dir_name, 'photos')

            shutil.copy(self.lnm.text(), dir_name)
            save_path = os.path.join(dir_name,
self.lnm.text().split('/')[-1])
            cur.execute("""SELECT MAX(question_number) FROM
questions WHERE user_id=%s AND bot_name=%s""",
                        (self.user_id, self.bot_name))
            max_namber = cur.fetchone()
            max_namber = max_namber[0]
            if max_namber is None:
                max_namber = 1
            else:
                max_namber += 1

            cur.execute(
                f"""INSERT INTO questions (user_id,
bot_name, question_number, question_text, btn_menu,
question_photo) VALUES (%s,%s,%s,%s,%s,%s)""",
                (self.user_id, self.bot_name, max_namber,
self.tem.toPlainText(), self.combo_box.currentText(),
save_path))
            conn.commit()
        else:
            cur.execute("""SELECT MAX(question_number) FROM
questions WHERE user_id=%s AND bot_name=%s""",

```

```

            (self.user_id, self.bot_name))
max_number = cur.fetchone()
max_number = max_number[0]
if max_number is None:
    max_number = 1
else:
    max_number += 1

    cur.execute(
        f"""INSERT INTO questions (user_id,
bot_name, question_number, question_text, btn_menu) VALUES
(%s,%s,%s,%s,%s) """,
        (self.user_id, self.bot_name, max_number,
self.tem.toPlainText(), self.combo_box.currentText()))
    conn.commit()

    conn.close()
    groupBox = QtWidgets.QGroupBox()
    groupBox.setMaximumWidth(940)
    groupBox.setStyleSheet('''QGroupBox {margin-top: 2px;}
QGroupBox:enabled {border: 2px solid grey;}
QGroupBox::title {
subcontrol-origin: margin;
left: 3ex; } ''')

    cb = QtWidgets.QCheckBox('Добавить фото')
    self.check_boxes.append(cb)
    lnm = QtWidgets.QLineEdit()
    self.lnpaths.append(lnm)
    lnm.setEnabled(False)
    btnm1 = QtWidgets.QPushButton('...')
    btnm1.setMinimumWidth(50)
    self.btn_picks.append(btnm1)

    if self.cb.isChecked():
        cb.setChecked(True)
        lnm.setText(self.lnm.text())

    cb.stateChanged.connect(lambda checked,
command_id=self.n: self.empty_check_cganges(command_id))
    lnm.textChanged.connect(lambda checked,
command_id=self.n: self.empty_check_cganges(command_id))
    btnm1.clicked.connect(lambda checked, command_id=self.n:
self.pick_changes(command_id))

    tem = QtWidgets.QTextEdit(self.tem.toPlainText())
    tem.textChanged.connect(lambda command_id=self.n:
self.empty_check_cganges(command_id))
    tem.setPlaceholderText('Введите текст вопроса')
    tem.setMaximumHeight(90)
    self.question_textes.append(tem)

```

```

        hlm = QtWidgets.QHBoxLayout()
        hlm.addWidget(cb)
        hlm.addWidget(lnm)
        hlm.addWidget(btnm1)

        combo_box_lb = QtWidgets.QLabel('Выберите клавиатуру
кнопок для этого вопроса')
        combo_box = QtWidgets.QComboBox()
        self.combo_boxes.append(combo_box)
        conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)

        with conn.cursor() as cur:
            cur.execute("""SELECT menu_name FROM buttons WHERE
user_id=%s AND bot_name=%s""",
                        (self.user_id, self.bot_name))
            btn_menus = cur.fetchall()

            for btn_menu2 in btn_menus:
                combo_box.addItem(btn_menu2[0])
        conn.close()

        combo_box_hl = QtWidgets.QHBoxLayout()
        combo_box_hl.addWidget(combo_box_lb)
        combo_box_hl.addWidget(combo_box)
        conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)

        with conn.cursor() as cur:

            cur.execute("""SELECT MAX(id) FROM questions WHERE
user_id=%s AND bot_name=%s""",
                        (self.user_id, self.bot_name))
            max_number = cur.fetchone()
            max_number = max_number[0]
            conn.close()

            btnm3 = QtWidgets.QPushButton('Сохранить изменения')
            btnm3.clicked.connect(lambda checked, command_id=self.n,
id=id: self.save_changes(command_id, max_number))
            self.changes_btn.append(btnm3)
            add_comand_btn_hl2 = QtWidgets.QHBoxLayout()
            add_comand_btn_hl2.addWidget(btnm3)

            add_menu_vl = QtWidgets.QVBoxLayout(groupBox)
            add_menu_vl.addWidget(tem)
            add_menu_vl.addLayout(combo_box_hl)
            add_menu_vl.addLayout(hlm)
            add_menu_vl.addLayout(add_comand_btn_hl2)

            self.bots_list.insertWidget(self.bots_list.count()-3,
groupBox)

```

```

        self.groupBox.hide()
        self.add_menu_btn.show()
    def add_btn_menu(self, btn_menu_name):
        self.combo_box.addItem(btn_menu_name)
    def add_question(self):
        self.groupBox.show()
        self.add_menu_btn.hide()

```

Файл «dbacc.py»

```

data_base = 'ChatbotBuilder'
user = 'postgres'
password = 'resist'
host = 'localhost'
port = '5434'

```

Файл «launch_tab.py»

```

from PyQt6 import QtWidgets, QtCore, QtGui
import psycpg2
import os
import threading
import subprocess

```

```

from dbacc import *

```

```

class Thread(threading.Thread):
    def __init__(self, dir_name, parent):
        threading.Thread.__init__(self)
        self.dir_name = dir_name
        self.parent = parent

    def run(self):
        command = f'python {self.dir_name}'
        self.sub = subprocess.Popen(command, shell=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        res = self.sub.communicate()
        if 'InvalidToken' in res[1].decode(encoding='utf-8'):
            self.parent.info_lb.show()
            self.parent.info_lb.setText('Токен этого бота
некорректный')

```

```

    def stop(self):
        self.parent.threads.pop(0)
        self.sub.terminate()
class Launch_tab(QtWidgets.QWidget):
    def __init__(self, bot_name, user_id, parent):
        QtWidgets.QWidget.__init__(self)
        self.bot_name = bot_name
        self.user_id = user_id
        self.parent = parent

```

```

        self.setStyleSheet("""QPushButton{border-radius:9px;
height:25px; width:250px; margin-left:250px; margin-right:250px;
background-color:#7d17d1;}

```

```

        QPushButton:hover{background-
color:#9127e8;}

        QLineEdit{background-color:#b3adad;
border-radius:9px; height:22px;}

        QTextEdit{background-
color:#b3adad;border-radius:9px;}"""
        self.threads = []
        self.launch_btn = QtWidgets.QPushButton('Запустить
Бота')
        self.launch_btn.clicked.connect(self.launch)
        self.stop_btn = QtWidgets.QPushButton('Остановить Бота')
        self.stop_btn.clicked.connect(self.stop_bot)
        self.stop_btn.setEnabled(False)
        self.delete_btn = QtWidgets.QPushButton('Удалить бота')
        self.delete_btn.clicked.connect(self.delete_bot)

        self.info_lb = QtWidgets.QLabel('')
        self.info_lb.setStyleSheet("""font-size:18px;
padding-
left:250px;""")
        self.info_lb.setMaximumHeight(20)
        self.info_lb.hide()

        self.ask_label = QtWidgets.QLabel('Вы точно хотите
удалить бота ?')
        self.ask_label.setStyleSheet("""font-size:18px;
padding-left:250px;""")
        self.ask_label.setMaximumHeight(20)
        self.ask_label.hide()

        self.delete_accept = QtWidgets.QPushButton('Да уверен')
        self.delete_accept.clicked.connect(self.accept_delete)
        self.delete_accept.hide()
        self.delete_cancel = QtWidgets.QPushButton('Отмена')
        self.delete_cancel.clicked.connect(self.cancel_delete)
        self.delete_cancel.hide()

        self.accept_cancel_hl = QtWidgets.QHBoxLayout()
        self.accept_cancel_hl.addWidget(self.delete_accept)
        self.accept_cancel_hl.addWidget(self.delete_cancel)

        self.main_vl = QtWidgets.QVBoxLayout()
        self.main_vl.addWidget(self.launch_btn)
        self.main_vl.addWidget(self.info_lb)
        self.main_vl.addWidget(self.stop_btn)
        self.main_vl.addWidget(self.delete_btn)
        self.main_vl.addWidget(self.ask_label)
        self.main_vl.addLayout(self.accept_cancel_hl)

```

```

        self.setLayout(self.main_vl)
    def accept_delete(self):
        self.parent.delete_bot(self.bot_name)
        conn = psycopg2.connect(database=data_base, user=user,
                                password=password, host=host, port=port)
        with conn.cursor() as cur:
            cur.execute("""DELETE FROM bots WHERE user_id=%s AND
name=%s""", (self.user_id, self.bot_name))
            conn.commit()
        conn.close()
        self.parent.bot_logout()
    def cancel_delete(self):
        self.delete_btn.show()
        self.ask_label.hide()
        self.delete_accept.hide()
        self.delete_cancel.hide()
    def delete_bot(self):
        self.delete_btn.hide()
        self.ask_label.show()
        self.delete_accept.show()
        self.delete_cancel.show()
    def stop_bot(self):
        self.threads[0].stop()
        self.launch_btn.setEnabled(True)
        self.stop_btn.setEnabled(False)
    def launch(self):
        self.launch_btn.setEnabled(False)
        self.stop_btn.setEnabled(True)
        dirname = os.path.dirname(os.path.abspath(__file__))
        dir_name = os.path.join(dirname,
f'{self.user_id}_{self.bot_name}')
        dir_name = os.path.join(dir_name, 'main.py')
        print(dir_name)

        thread = Thread(dir_name, self)
        self.threads.append(thread)
        thread.start()

```

Файл «main.py»

```

from telegram.ext import *
from buttons import *
from dbacc import *

```

```

dirname = os.path.dirname(os.path.abspath(__file__))
bot_owner_id = int(dirname.split('\\')[-1].split('_', 1)[0])
bot_name = dirname.split('\\')[-1].split('_', 1)[1]
conn = psycopg2.connect(database=data_base, user=user,
password=password, host=host, port=port)

```

```

with conn.cursor() as cur:
    cur.execute("""SELECT token FROM bots WHERE user_id=%s AND
name=%s""",
                (bot_owner_id, bot_name))

```

```

        TOKEN = cur.fetchone()
        TOKEN = TOKEN[0]
        print(TOKEN)
    conn.close()

    updater = Updater(TOKEN)
    dispatcher = updater.dispatcher
    functions= []
    functions_counter = 0
    def queryHandler(update, context: CallbackContext):
        query = update.callback_query
        data = query.data
        text = query.message.text
        id = query.message.chat_id
        m_id = query.message.message_id
        str_id = str(id)
        context.bot.delete_message(chat_id=id, message_id=m_id)

        conn = psycopg2.connect(database=data_base, user=user,
                                password=password, host=host, port=port)
        with conn.cursor() as cur:
            cur.execute("""SELECT answears FROM bots_users WHERE
bot_owner_id=%s AND bot_name=%s AND user_id=%s""",
            (bot_owner_id, bot_name, str(id)))
            answears = cur.fetchone()
            answears = answears[0]
            if answears is None:
                cur.execute("""UPDATE bots_users SET answears=%s
WHERE bot_owner_id=%s AND bot_name=%s AND user_id=%s""",
                (data+'|', bot_owner_id, bot_name, str(id)))
            else:
                cur.execute("""UPDATE bots_users SET answears=%s
WHERE bot_owner_id=%s AND bot_name=%s AND user_id=%s""",
                (answears+data + '|', bot_owner_id,
bot_name, str(id)))

            cur.execute("""UPDATE bots_users SET
current_question=current_question+1 WHERE bot_owner_id=%s AND
bot_name=%s AND user_id=%s""", (bot_owner_id, bot_name,
str(id)))
            conn.commit()
            cur.execute("""SELECT current_question FROM bots_users
WHERE bot_owner_id=%s AND bot_name=%s AND user_id=%s""",
            (bot_owner_id, bot_name, str(id)))
            curent_question = cur.fetchone()
            curent_question = curent_question[0]

            try:
                cur.execute("""SELECT question_text FROM questions
WHERE user_id=%s AND bot_name=%s AND question_number=%s""",
                (bot_owner_id, bot_name,
curent_question))

```

```

        question_text = cur.fetchone()
        question_text = question_text[0]
        cur.execute(
            """SELECT btn_menu FROM questions WHERE
user_id=%s AND bot_name=%s AND question_number=%s""",
            (bot_owner_id, bot_name, curent_question))
        btn_menu = cur.fetchone()
        btn_menu = btn_menu[0]
        cur.execute(
            """SELECT question_photo FROM questions WHERE
user_id=%s AND bot_name=%s AND question_number=%s""",
            (bot_owner_id, bot_name, curent_question))
        question_photo = cur.fetchone()
        question_photo = question_photo[0]
        if question_photo is not None:
            context.bot.send_photo(chat_id=id,
caption=question_text, photo=open(question_photo, 'rb'),
reply_markup=get_keyboard(btn_menu))
        else:
            context.bot.send_message(chat_id=id,
text=question_text, reply_markup=get_keyboard(btn_menu))
    except:
        cur.execute("""SELECT answears FROM bots_users WHERE
bot_owner_id=%s AND bot_name=%s AND user_id=%s""",
            (bot_owner_id, bot_name, str_id))
        answears = cur.fetchone()
        answears = answears[0]
        cur.execute("""SELECT result, text, photo FROM
answears WHERE user_id=%s AND bot_name=%s""", (bot_owner_id,
bot_name))
        for result, text, photo in cur.fetchall():
            if answears == result:
                if photo is not None:
                    context.bot.send_photo(chat_id=id,
caption=text, photo=open(photo, 'rb'))
                    return False
                else:
                    context.bot.send_message(chat_id=id,
text=text)
                    return False
            context.bot.send_message(chat_id=id, text="Вы прошли
тест")
        cur.execute(
            """UPDATE bots_users SET current_question=1,
answears=NULL WHERE bot_owner_id=%s AND bot_name=%s AND
user_id=%s""",
            (bot_owner_id, bot_name, str_id))
        conn.commit()
        conn.close()

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

```

```

with conn.cursor() as cur:
    cur.execute("""SELECT comand_name, comand_text, comand_photo
FROM comands WHERE user_id=%s AND bot_name=%s""",
                (bot_owner_id, bot_name))
    comands = cur.fetchall()
    for comand_name, comand_text, comand_photo in comands:
        def function(update, bot, comand_text=comand_text,
comand_name=comand_name, comand_photo=comand_photo):
            id = update.message.chat_id
            if comand_photo is not None:
                bot.bot.send_photo(chat_id=id,
caption=comand_text, photo=open(comand_photo, 'rb'))
            else:
                bot.bot.send_message(chat_id=id,
text=comand_text)

        if comand_name.replace('/', '') == 'start':
            conn = psycopg2.connect(database=data_base,
user=user, password=password, host=host, port=port)
            with conn.cursor() as cur:
                cur.execute("""SELECT user_id FROM
bots_users WHERE bot_owner_id=%s AND
bot_name=%s""", (bot_owner_id, bot_name))
                all_ids = []
                for i in cur.fetchall():
                    all_ids.append(i[0])

                if str(id) in all_ids:
                    pass
                else:
                    cur.execute("""INSERT INTO bots_users
(bot_owner_id, bot_name, user_id, current_question) VALUES
(%s,%s,%s,%s)""", (bot_owner_id, bot_name, str(id), 1))
                    conn.commit()
                    cur.execute("""SELECT current_question FROM
bots_users WHERE bot_owner_id=%s AND bot_name=%s AND
user_id=%s""", (bot_owner_id, bot_name, str(id)))
                    curent_question = cur.fetchone()
                    curent_question = curent_question[0]
                    try:
                        cur.execute(
                            """SELECT question_text FROM
questions WHERE user_id=%s AND bot_name=%s AND
question_number=%s""",
                            (bot_owner_id, bot_name,
curent_question))
                        question_text = cur.fetchone()
                        question_text = question_text[0]
                        cur.execute(
                            """SELECT btn_menu FROM questions
WHERE user_id=%s AND bot_name=%s AND question_number=%s""",

```

```

        (bot_owner_id, bot_name,
current_question))

        btn_menu = cur.fetchone()
        btn_menu = btn_menu[0]
        cur.execute(
            """SELECT question_photo FROM
questions WHERE user_id=%s AND bot_name=%s AND
question_number=%s""",
            (bot_owner_id, bot_name,
current_question))

        question_photo = cur.fetchone()
        question_photo = question_photo[0]
        if question_photo is not None:
            bot.bot.send_photo(chat_id=id,
caption=question_text, photo=open(question_photo, 'rb'),

reply_markup=get_keyboard(btn_menu))
        else:
            bot.bot.send_message(chat_id=id,
text=question_text,

reply_markup=get_keyboard(btn_menu))
        except Exception as e:
            bot.bot.send_message(chat_id=id,
text="Вы прошли тест")
            cur.execute("""UPDATE bots_users SET
current_question=1, answears=NULL WHERE bot_owner_id=%s AND
bot_name=%s AND user_id=%s""",
            (bot_owner_id, bot_name,
str(id)))
            conn.commit()
            conn.close()
            functions.append(function)

dispatcher.add_handler(CommandHandler(comand_name.replace('/', ''
), functions[functions_counter], run_async=True))
            functions_counter+=1
conn.close()

dispatcher.add_handler(CallbackQueryHandler(queryHandler,
run_async=True))
updater.start_polling(read_latency=4, timeout=1)

```

Приложение В
(обязательное)
Конструктор чат-ботов
Презентация

					<i>ПГУ1.090301.07.001 ПЗ</i>	Лист
						144
Изм.	Лист	№ докум.	Подпись	Дата		

Конструктор чат-ботов

Автор работы: студент группы 19ВВП1

Дорофеев А.А.

Научный руководитель: к.т.н., доцент кафедры ВТ

Карамышева Н.С.

АКТУАЛЬНОСТЬ РАБОТЫ

Пандемия коронавируса существенно повлияла на популярность использования чат-ботов, особенно в сфере предоставления услуг и торговли. Боты предоставляют пользователю возможность взаимодействовать с брендом и получать ответы на вопросы или помощь в решении проблем. Использование чат-ботов для бизнеса может сократить расходы времени и денежных ресурсов на обслуживание клиентов.

Конструктор чат-ботов — это инструмент, который позволяет пользователям создавать собственных чат-ботов без необходимости программировать их с нуля. Данное решение для бизнеса способно снизить расходы на реализацию требуемых чат-ботов, компании смогут создавать необходимых чат-ботов самостоятельно, без приобретения услуг разработки другими IT-компаниями.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.07.001 ПЗ

Лист

145

ЦЕЛИ И ЗАДАЧИ РАЗРАБОТКИ

Цели данного проекта:

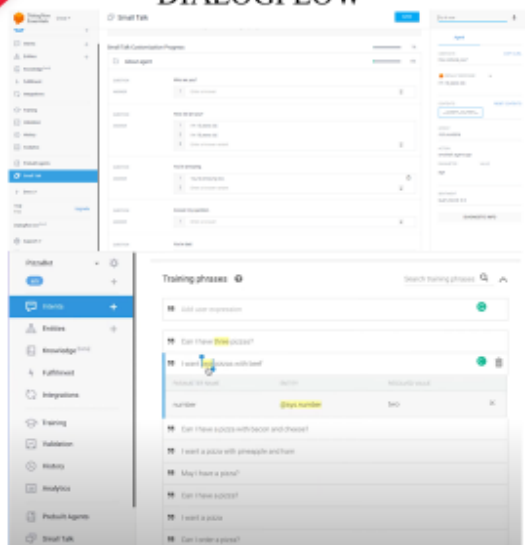
- анализ актуальных средств программной разработки;
- изучение нюансов разработки чат-диалоговых систем на базе мессенджеров;
- реализация приложения с графическим интерфейсом, способного создавать уникальных чат-ботов для ведения диалога с пользователями.

Задачи разработки:

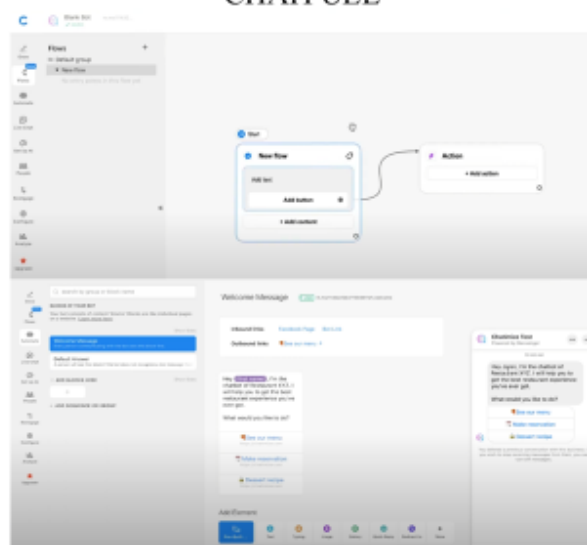
- анализ предметной области;
- анализ требований на разработку системы;
- проектирование приложения;
- разработка информационного и программного приложения;
- тестирование программного обеспечения.

АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

DIALOGFLOW



CHATFUEL

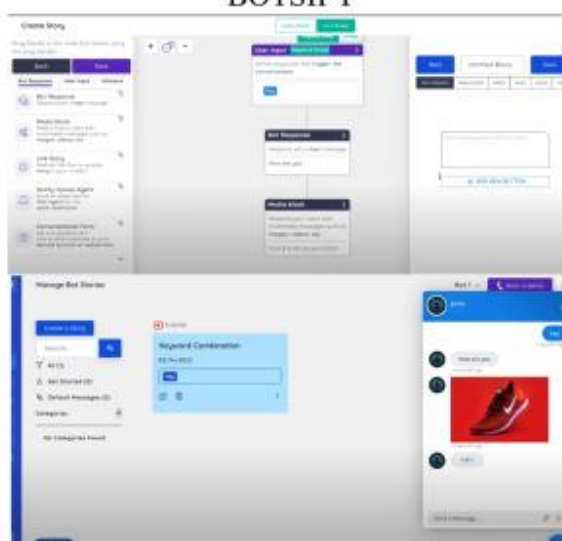


АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

MANYBOT



BOTSIFY



АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

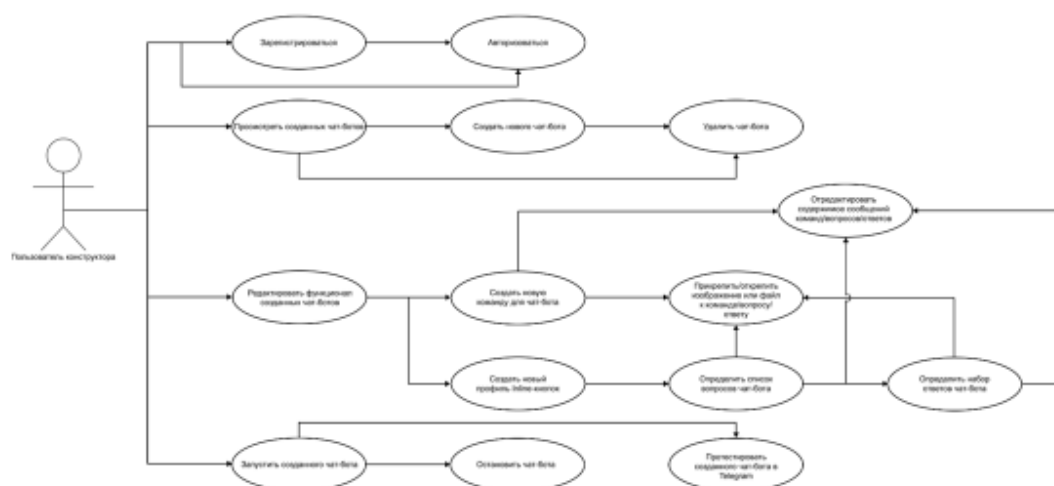
Платформа	Удобство интерфейса	Функциональность	Бесплатный план	Совместимость
Dialogflow	Неудобный	Высокая	Навсегда	Все мессенджеры
Chatfuel	Удобный	Высокая	Ограничен	Facebook
Manybot	Неудобный	Средняя	Навсегда	Telegram
Botsify	Удобный	Средняя	Отсутствует	Telegram, WhatsApp
Разрабатываемое приложение	Удобный	Средняя	Навсегда	Telegram

АНАЛИЗ ТРЕБОВАНИЙ

Функциональные требования:

- регистрация и авторизация пользователей;
- добавление и удаление новых чат-ботов;
- редактирование функционала созданных ботов;
- запуск/приостановка работы чат-ботов.

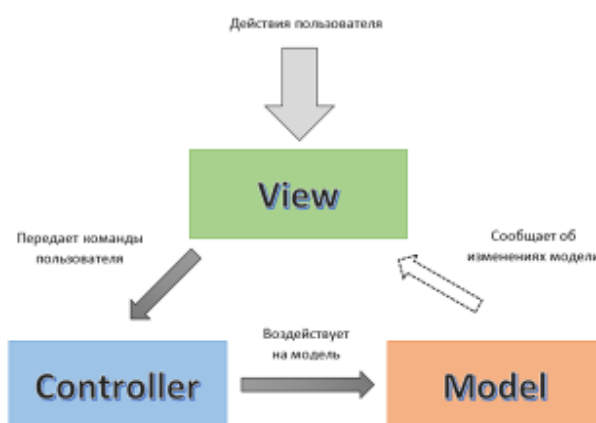
ДИАГРАММА ПРЕЦЕДЕНТОВ ПОЛЬЗОВАТЕЛЯ



СТЕК ТЕХНОЛОГИЙ



АРХИТЕКТУРА ПРИЛОЖЕНИЯ

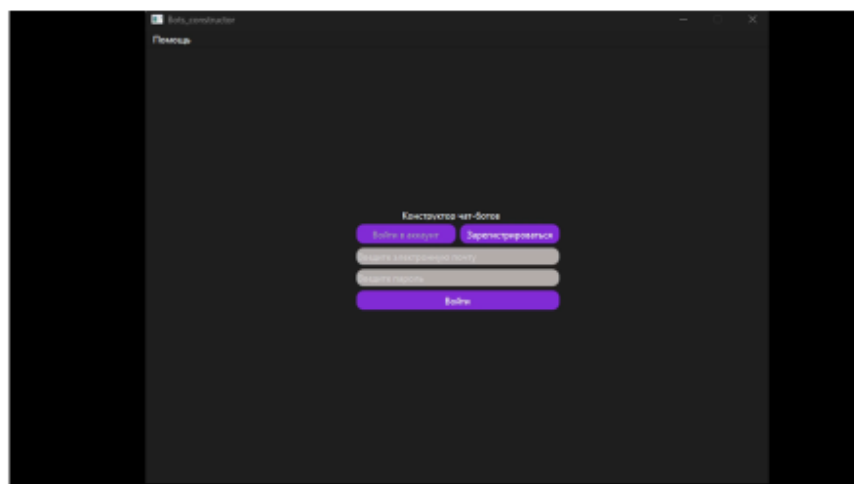


ЭЛЕМЕНТЫ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА

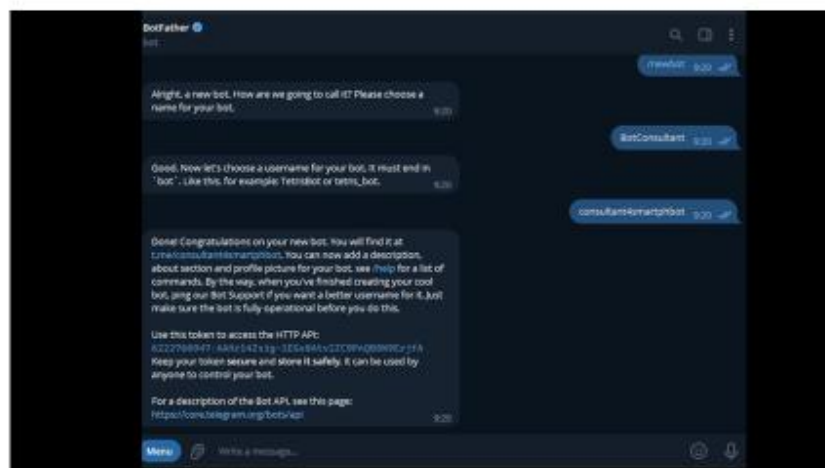
При разработке графического интерфейса программы были использованы следующие элементы:

- Текстовые метки QLabel
- Кнопки QPushButton
- Поля ввода QLineEdit
- Поля ввода QTextEdit
- Выпадающие меню QComboBox
- Флажки QCheckBox
- Счётчики QSpinBox
- Контекстное меню QMenu
- Модальное окно QMessageBox

ПРОЦЕСС СОЗДАНИЯ ЧАТ-БОТА



ТЕСТ РАБОТЫ СОЗДАННОГО ЧАТ-БОТА



ТЕСТИРОВАНИЕ

Цель тестирования	Данные	Шаги	Ожидаемый результат
1. Проверка корректности введенных данных при регистрации.	Логин — login. Пароль — pass.	1. Запустить программу. 2. Нажать на кнопку «Зарегистрироваться». 3. Ввести данные в соответствующие текстовые поля.	Высвечиваются соответствующие ошибки, текстовые поля становятся красными, кнопка регистрации неактивна.
2. Проверка корректной авторизации.	Логин — test@mail.ru. Пароль — Password.	1. Запустить программу. 2. Нажать на кнопку «Зарегистрироваться». 3. Ввести данные в соответствующие текстовые поля. 4. Нажать на кнопку «Зарегистрироваться». 5. Нажать на кнопку «Войти в аккаунт». 6. Ввести логин или пароль, не соответствующий исходным данным и нажать на кнопку «Войти».	Регистрация нового пользователя прошла успешно. При попытке войти в учетную запись, используются неверные данные, высвечивается ошибка.
3. Проверка возможности создания двух чат-ботов с одинаковыми именами. Проверка введенного API-токена.	Имя чат-бота — BotFather. API-токен, полученный от BotFather.	1. Запустить программу. 2. Войти в систему с учетными данными из пункта 2. 3. Нажать кнопку «Добавить бота». 4. Ввести имя и исходные данные. 5. В поле токена ввести случайный набор символов. 6. Ввести токен из исходных данных. 7. Нажать кнопку «Добавить бота». 8. Повторить действия подпунктов 4 и 6.	Высвечивается ошибка о некорректном токене и о создании бота с уже существующим названием.
4. Проверка корректности введенных данных при создании новой команды.	-	1. Запустить программу. 2. Войти в систему с учетными данными из пункта 2. 3. Ввести в текстовое поле название команды символы, не относящиеся к символу латинского алфавита, не арабские цифры или попытаться начать команду не с символа «/».	Данные символы не проходят проверку и не вводятся в текстовое поле.
5. Проверка работы программы в случае удаления созданного чат-бота в BotFather.	Чат-бот, созданный в пункте 3.	1. Перейти в BotFather и удалить ранее созданный чат-бота. 2. Запустить этого чат-бота в конструкторе.	При запуске высвечивается ошибка некорректного API-токена.
6. Проверка возможности работы чат-бота.	Учетная запись с двумя и более созданными чат-ботами.	1. Запустить чат-бота в конструкторе. 2. Войти в приложение Telegram с разных устройств. 3. Найти чат-бота и однократно нажать общение.	Корректная работа чат-бота, согласно заданным в конструкторе параметрам.

ЗАКЛЮЧЕНИЕ

В результате работы:

- проведен анализ предметной области и требований
- проведен анализ наиболее популярных аналогов
- разработано программное приложение
- проведено тестирование приложения

В будущем возможно усовершенствование приложения:

- расширение функционала добавляемых чат-ботов
- улучшение пользовательского интерфейса
- оптимизация программного кода
- интеграция с другими платформами

Спасибо за внимание!

					<i>ПГУ1.090301.07.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		152