

Разработка чат-бота для автоматизации процесса подачи заявок в отдел Сервиса компьютерных средств и связи

Языков Кирилл Александрович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Научный руководитель:

Лучанинов Дмитрий Васильевич

Приамурский государственный университет имени Шолом-Алейхема

Старший преподаватель

Аннотация

Целью данной статьи является разработка чат-бота телеграмм для отдела сервиса компьютерных средств и связи (СКСиС) в ПГУ им. Шолом-Алейхема. Статья может быть использована в качестве основы для дальнейших исследований и разработок в области информационно-коммуникационных систем. В дальнейшем в работе рассмотрены основные этапы разработки чат-бота в мессенджере Telegram, включая создание игрового процесса, реализацию сетевого взаимодействия и тестирование игры.

Ключевые слова: чат-бот, Telegram, сетевое взаимодействие.

Development of a chatbot to automate the application process to the Computer and Communication Services Department

Yazykov Kirill Aleksandrovich

Sholom-Aleichem Priamursky State University

Student

Scientific supervisor:

Luchaninov Dmitry Vasilyevich

Sholom-Aleichem Priamursky State University

Senior Lecturer

Abstract

The purpose of this article is to develop a telegram chatbot for the Computer Tools and communications service department (CSIS) at the Sholom-Aleichem Priamursky State University. The article can be used as a basis for further research and development in the field of information and communication systems. In the future, we will consider the main stages of developing a chatbot in the Telegram messenger, including creating a gameplay, implementing network interaction, and testing the game.

Keywords: chatbot, Telegram, networking.

1 Введение

1.1 Актуальность

В современном мире информационно-коммуникационные системы играют важную роль во многих сферах деятельности. Они позволяют людям обмениваться информацией, взаимодействовать друг с другом и создавать новые возможности для развития. Поэтому в условиях стремительного развития цифровых технологий и повсеместной цифровизации государственных услуг, разработка чат-бота для автоматизации процесса подачи заявок в отдел сервиса компьютерных средств и связи (СКСиС) является не просто актуальной, а критически важной задачей.

1.2 Обзор исследований

И. В. Бахтин в своей статье «Разработка чат-ботов для автоматизации бизнес-процессов» рассмотрел существующие типы чат-ботов, которые могут быть разработаны компаниями для увеличения количества проданных товаров, а также определил функции чат-ботов для удобного взаимодействия между клиентом и чат-ботом. [1]. Также в своей работе Н. Ю. Матвеева и А.В. Золотарюк представили результаты анализа возможностей создания чат-ботов в современных мессенджерах, их актуальность в современном мире и определили сферы их применения. [2]. Методологию разработки чат-ботов, их классификацию и процесс работы бота в своей статье осветили Б. С. Горячкин, К. С. Розен и Д. В. Бобров. Также ими были рассмотрены технические метрики оценки эффективности чат-бота и возможность применения ботов в специальном прогнозировании потребностей населения в жилье [3]. В книге «Разработка чат-ботов и разговорных интерфейсов», автором которой является С. Джанарса рассказывается о работе диалоговых интерфейсов как способе взаимодействия машины с людьми на естественном языке, а также описывается процесс разработки чат-ботов для выполнения различных прикладных задач с помощью различных мессенджеров [4].

1.3 Цель исследования

Целью данной работы является разработка чат-бота телеграмм для отдела сервиса компьютерных средств и связи (СКСиС) в ПГУ им. Шолом-Алейхема.

2 Материалы и методы

Для реализации чат-бота телеграмм будут использованы следующие программные средства и среда разработки: Python, PyCharm, библиотека python-telegram-bot. Также создание чат-бота будет проходить в мессенджере Telegram.

3 Результаты исследования

Данный чат-бот разработан для автоматизации процесса подачи заявок в отдел сервиса компьютерных средств и связи (СКСиС).

Для создания бота в программе PyCharm создадим новый проект в файле «ConsoleApp.py» (Рис. 1).

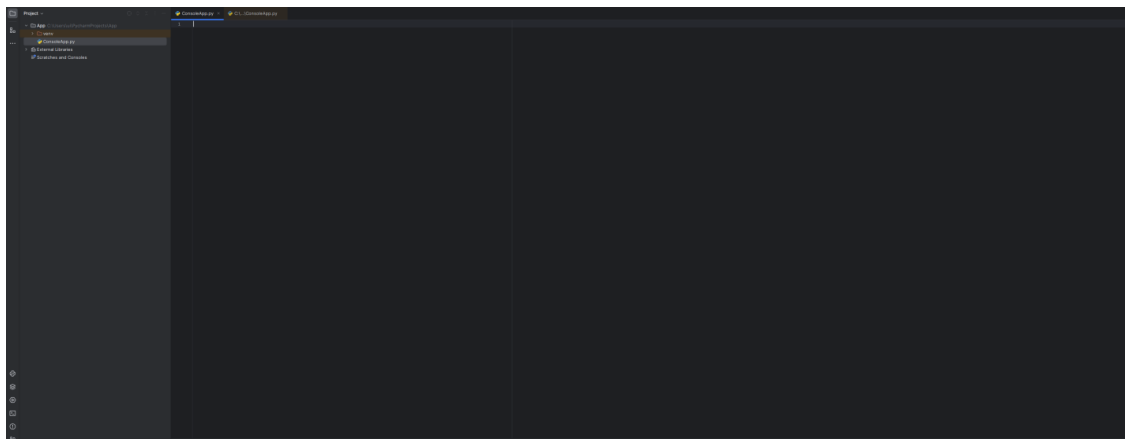


Рисунок 1 – Создание проекта в программе PyCharm

Для начала следует импортировать нужные нам библиотеки, чтобы работать с чат-ботом

Для установки библиотек рекомендуется ставить PIP, Установка PIP необходима для того, чтобы упростить процесс работы с библиотеками Python. Без PIP пользователям пришлось бы вручную загружать и устанавливать каждую библиотеку, что может быть трудоемким и подверженным ошибкам. PIP автоматизирует этот процесс, делая его более быстрым и надежным.

Таким образом, PIP является важным инструментом для разработчиков Python, позволяя им легко управлять библиотеками и сосредоточиться на разработке приложений.

Чтобы продолжить написать код, нужно получить токен для телеграмм-бота, а также узнать свой ID, чтобы указать его для административной части.

Токен - это идентификатор, который используется для представления доступа к ресурсам или выполнения определенных операций. В сущности, он является символом, который подтверждает вашу личность или разрешает вам осуществить определенное действие. Токен может быть использован в различных сферах, таких как криптовалюты, безопасность информации, доступ к интернет-сервисам и многое другое. Для того, чтобы получить токен, нужно написать боту: <https://web.telegram.org/k/#@BotFather>, отправить сообщение /start, а потом, /newbot, далее следуем указанием, после чего, он отправит токен для нашего чат-бота.

ID — это идентификатор, уникальный признак объекта, позволяющий отличать его от других объектов, то есть идентифицировать. Для того, чтобы получить ID, нужно перейти по следующей ссылке: https://web.telegram.org/k/#@getmyid_bot, написать /start, после чего он отправляет ID.

Возвращаемся к PyCharm, чтобы продолжить писать код с теми данными, которые получились.

Напишем код, в котором подключим нашего телеграмм-бота по токену, а также укажем ID, в качестве администратора (Рис. 2).

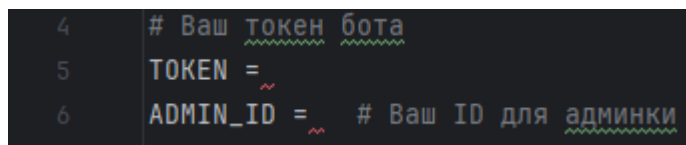
A screenshot of a code editor with a dark background. It shows three lines of Python code. Line 4: `# Ваш токен бота`. Line 5: `TOKEN =`. Line 6: `ADMIN_ID = # Ваш ID для админки`. The words 'токен бота' and 'админки' are underlined in green, and 'ADMIN_ID' is underlined in red.

Рисунок 2 – Добавление токена и ID для получения прав администратора

Будем создавать систему, которая будет хранить в себе заявки, которые будет отправлять пользователи, для этого используем `requests = []`, также нам нужно проверять статусы заявок при помощи `request_status = {}`.

`Asyncdefstart(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:`

Эта функция является обработчиком команды, которая запускается, когда пользователь начинает взаимодействие с ботом. Она принимает два параметра: `update`, содержащий информацию о сообщении от пользователя, и `context`, который предоставляет контекст выполнения.

Для того, чтобы бот выводил две кнопки пользователю, создадим их при помощи `keyboard []`

`reply_markup = InlineKeyboardMarkup(keyboard)`

`awaitupdate.message.reply_text('Здравствуйте, выберите пожалуйста из списка проблему:', reply_markup=reply_markup)`, данная команда позволит выводить сообщение боту, которое появляется при нажатии на кнопку «Старт».

Допишем еще небольшую часть кода (Рис. 4).

```

20 async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
21     user_id = update.message.from_user.id
22
23     if user_id == ADMIN_ID:
24         keyboard = [
25             [InlineKeyboardButton(текст="Посмотреть все заявки", callback_data="view_requests"),
26              InlineKeyboardButton(текст="Удалить все заявки", callback_data="delete_requests")],
27             [InlineKeyboardButton(текст="Замена картриджа", callback_data="cartridge"),
28              InlineKeyboardButton(текст="Проблема", callback_data="other")]
29         ]
30         reply_markup = InlineKeyboardMarkup(keyboard)
31         await update.message.reply_text(текст="Выберите действие:", reply_markup=reply_markup)
32     else:
33         keyboard = [
34             [InlineKeyboardButton(текст="Замена картриджа", callback_data='cartridge')],
35             [InlineKeyboardButton(текст="Проблема", callback_data='other')]
36         ]
37         reply_markup = InlineKeyboardMarkup(keyboard)
38         await update.message.reply_text(текст="Запросите, выберите пожалуйста из списка проблему:", reply_markup=reply_markup)
39
40 async def button(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
41     query = update.callback_query
42     await query.answer()
43
44     user_id = query.from_user.id
45
46     if user_id == ADMIN_ID:
47         if query.data == 'cartridge':
48             await query.edit_message_text(текст="Напишите пожалуйста название и модель принтера")
49             context.user_data['state'] = 'awaiting_printer_model'
50         elif query.data == 'other':
51             await query.edit_message_text(текст="Напишите пожалуйста проблему, которая возникла у вас")
52             context.user_data['state'] = 'awaiting_other_problem'
53         elif query.data.startswith('toggle_request'):
54             request_index = int(query.data.split('-')[-1])
55             await toggle_request_status(update, request_index)
56         elif query.data == 'view_requests':
57             await view_requests(update, context)
58         elif query.data == 'delete_requests':
59             await confirm_delete_requests(update)
60         elif query.data == 'confirm_delete':
61             await delete_requests(update)
62         elif query.data == 'cancel_delete':
63             await update.callback_query.edit_message_text(текст="Удаление отменено.")
64         elif query.data == 'start': # Обработка нажатия на кнопку "Начать заново"
65             await start(update, context) # Повторный вызов функции start
66         elif query.data == 'restart':
67             await restart(update, context)
68         elif query.data == 'cancel_request': # Отмена заявки
69             await cancel_request(update, context)
70         elif query.data.startswith('delete_request'): # Обработка нажатия на кнопку "Удалить заявку"
71             request_index = int(query.data.split('-')[-1])
72             await delete_request(update, request_index)
73     else:
74         if query.data == 'cartridge':
75             await query.edit_message_text(текст="Напишите пожалуйста название и модель принтера",

```

Рисунок 4 – Добавление функционала для ответов бота и администрирования

Данный фрагмент кода реализует функциональность для обработки заявок на техническую помощь через Telegram-бота, который позволяет пользователям отправлять заявки, а также администратору управлять этими заявками, что делает процесс более организованным и эффективным.

Немного допишем код для администрирования бота, а также сделаем, чтобы при нажатии на определенную кнопку, бот мог спрашивать у пользователя, что необходимо указать в сообщении (Рис. 5).

```

80     else:
81         if query.data == 'cartridge':
82             await query.edit_message_text(text="Напишите пожалуйста название и модель принтера",
83                                           reply_markup=InlineKeyboardMarkup([
84                                             [InlineKeyboardButton(text="Отменить заявку", callback_data='cancel_request')]
85                                           ]))
86             context.user_data['state'] = 'awaiting_printer_model'
87         elif query.data == 'other':
88             await query.edit_message_text(text="Напишите пожалуйста проблему, которая возникла у вас",
89                                           reply_markup=InlineKeyboardMarkup([
90                                             [InlineKeyboardButton(text="Отменить заявку", callback_data='cancel_request')]
91                                           ]))
92             context.user_data['state'] = 'awaiting_other_problem'
93         elif query.data == 'start': # Обработка нажатия на кнопку "Начать заново"
94             await start(update, context) # Повторный вызов функции start
95         elif query.data == 'restart':
96             await restart(update, context)
97         elif query.data == 'cancel_request': # Отмена заявки
98             await cancel_request(update, context)
99
100     async def restart(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
101         user_id = update.callback_query.from_user.id
102         keyboard_user = [
103             [InlineKeyboardButton(text="Замена картриджа", callback_data='cartridge')],
104             [InlineKeyboardButton(text="Другое", callback_data='other')]
105         ]
106
107         keyboard_admin = [
108             [InlineKeyboardButton(text="Посмотреть все заявки", callback_data='view_requests')],
109             [InlineKeyboardButton(text="Удалить все заявки", callback_data='delete_requests')],
110             [InlineKeyboardButton(text="Замена картриджа", callback_data='cartridge')],
111             [InlineKeyboardButton(text="Другое", callback_data='other')]
112         ]
113
114         reply_markup_user = InlineKeyboardMarkup(keyboard_user)
115         reply_markup_admin = InlineKeyboardMarkup(keyboard_admin)
116
117         # Проверяем, является ли пользователь администратором
118         if user_id == ADMIN_ID:
119             await context.bot.send_message(chat_id=ADMIN_ID, text="Выберите действие:", reply_markup=reply_markup_admin)
120         else:
121             await update.callback_query.edit_message_text(text="Здравствуйте, выберите пожалуйста из списка проблему:",
122                                                           reply_markup=reply_markup_user)

```

Рисунок 5 – Использование библиотеки python-telegram-bot

Объяснение кода: Если пользователь нажимает кнопку "Замена картриджа", бот запрашивает название и модель принтера, добавляя кнопку для отмены заявки. Состояние пользователя устанавливается в `awaiting_printer_model`. Если нажата кнопка "Другое", бот просит описать возникшую проблему и также добавляет кнопку для отмены заявки, устанавливая состояние в `awaiting_other_problem`.

Если пользователь нажимает кнопку "Начать заново", вызывается функция `start`, что позволяет пользователю вернуться к началу взаимодействия с ботом.

Если нажата кнопка "restart", вызывается функция `restart`, которая также перезапускает процесс. Если нажата кнопка "Отменить заявку", вызывается функция `cancel_request`, которая обрабатывает отмену текущей заявки.

Данный код представляет собой основной контроллер игры. Игра представляет собой гонку между двумя игроками на игровом поле, состоящем из пути с определенными индексами.

Также написав проверку прав администратора, можно убедиться, что пользователь является администратором. В таком случае у него появляются две дополнительные кнопки «Посмотреть все заявки» и «Удалить все заявки».

На следующем этапе допишем функциональность для административной части, чтобы администратору приходили уведомления о новых заявках в телеграмм, через уведомления на телефоне, также напишем `id` пользователя, который отправляет заявку, его фамилию и имя.

Также сделаем так, чтобы новая заявка, которая приходит администратору была со значком **✖** (Рис. 6).

```

115 async def handle_message(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
116     user_id = update.message.from_user.id
117     text = update.message.text
118     user_name = update.message.from_user.first_name
119     user_surname = update.message.from_user.last_name
120
121     if context.user_data.get('state') == 'awaiting_printer_model':
122         context.user_data['printer_model'] = text
123         await update.message.reply_text("Напишите пожалуйста название корпуса и номер кабинета")
124         context.user_data['state'] = 'awaiting_cabinet_info'
125     elif context.user_data.get('state') == 'awaiting_cabinet_info':
126         cabinet_info = text
127         printer_model = context.user_data.get('printer_model')
128         request_text = f"Замена картриджа: {printer_model}, Корпус: {cabinet_info} от пользователя {user_id} ({user_name} {user_surname}) ✖"
129         requests.append((request_text, datetime.now()))
130         request_status[len(requests) - 1] = '✖'
131         context.user_data['state'] = None
132         await update.message.reply_text("Спасибо за заявку, ожидайте техника!")
133         await notify_admin(context, request_text)
134         await send_start_button(update, context)
135
136     elif context.user_data.get('state') == 'awaiting_other_problem':
137         context.user_data['problem_description'] = text
138         await update.message.reply_text("Укажите пожалуйста ваш корпус и кабинет")
139         context.user_data['state'] = 'awaiting_other_cabinet_info'
140     elif context.user_data.get('state') == 'awaiting_other_cabinet_info':
141         cabinet_info = text
142         problem_description = context.user_data.get('problem_description')
143         request_text = f"Заявка из {cabinet_info}, Проблема: {problem_description} от пользователя {user_id} ({user_name} {user_surname}) ✖"
144         requests.append((request_text, datetime.now()))
145         request_status[len(requests) - 1] = '✖'
146         context.user_data['state'] = None
147         await update.message.reply_text("Спасибо за заявку, ожидайте техника!")
148         await notify_admin(context, request_text)
149         await send_start_button(update, context)

```

Рисунок 6 – Написание функциональности для администрирования

Теперь добавим одну полезную функциональность, чтобы после отправки заявки, пользователь не прописывал постоянно /start, а была отдельно кнопка «Начать заново», при нажатии на которую пользователь перезапускает процесс взаимодействия с ботом. Также сделаем администратору кнопку для изменения статуса заявок и вывод всех заявок для администратора (Рис. 7).

```

153 async def send_start_button(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
154     keyboard = [
155         [InlineKeyboardButton(text="Начать заново", callback_data='restart')]
156     ]
157     reply_markup = InlineKeyboardMarkup(keyboard)
158     await update.message.reply_text(text="Вы можете отправить новую заявку:", reply_markup=reply_markup)
159
160 async def notify_admin(context: ContextTypes.DEFAULT_TYPE, message: str) -> None:
161     # Отправка уведомления администратору
162     await context.bot.send_message(chat_id=ADMIN_ID, text=f"Новая заявка: {message}\n" +
163                                     "Нажмите на кнопку, чтобы изменить статус:",
164                                     reply_markup=InlineKeyboardMarkup([
165                                         [InlineKeyboardButton(text="Изменить статус",
166                                                                 callback_data=f'toggle_request_{len(requests) - 1}']],
167                                         [InlineKeyboardButton(text="Удалить заявку",
168                                                                 callback_data=f'delete_request_{len(requests) - 1}')] # Кнопка удаления заявки
169                                     ]))
170
171
172 async def view_requests(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
173     if requests:
174         for i, (request, timestamp) in enumerate(requests):
175             formatted_time = timestamp.strftime('%Y-%m-%d %H:%M:%S')
176             # Отправка сообщения с заявкой и кнопками изменения статуса и удаления
177             status = request_status.get(i, 'X') # Получаем статус из request_status
178             await context.bot.send_message(chat_id=ADMIN_ID,
179                                           text=f"{request} {status} (время: {formatted_time})",
180                                           reply_markup=InlineKeyboardMarkup([
181                                               [InlineKeyboardButton(text="Изменить статус", callback_data=f'toggle_request_{i}']],
182                                               [InlineKeyboardButton(text="Удалить заявку", callback_data=f'delete_request_{i}')]
183                                           ]))
184             # Обновление предыдущего сообщения с кнопкой "Начать заново"
185             await update.callback_query.edit_message_reply_markup(
186                 reply_markup=InlineKeyboardMarkup([
187                     [InlineKeyboardButton(text="Начать заново", callback_data='restart')]
188                 ])
189             )
190     else:
191         await update.callback_query.edit_message_text(text="Нет новых заявок.",
192                                                       reply_markup=InlineKeyboardMarkup([
193                                                         [InlineKeyboardButton(text="Начать заново", callback_data='restart')]
194                                                       ]))
195

```

Рисунок 7 – Создание кнопки «Начать заново» и «Изменить статус»

Теперь сделаем функциональность, чтобы статус кода менялся при нажатии на кнопку с X на ✓, а также напишем, чтобы администратор мог удалять неправильные заявки или заявки, которые долго лежат в статусе ✓ (Рис. 8).

```

197 async def toggle_request_status(update: Update, request_index: int) -> None:
198     if request_index in request_status:
199         if request_status[request_index] == 'X':
200             request_status[request_index] = '✓'
201             # Обновляем сообщение о заявке в админке
202             request, timestamp = requests[request_index]
203             formatted_time = timestamp.strftime('%Y-%m-%d %H:%M:%S')
204             await update.callback_query.edit_message_text(
205                 text=f"{request.replace(' X', '')} {request_status[request_index]} (время: {formatted_time})",
206                 reply_markup=InlineKeyboardMarkup([
207                     [InlineKeyboardButton(text="Изменить статус", callback_data=f'toggle_request_{request_index}']],
208                     [InlineKeyboardButton(text="Удалить заявку", callback_data=f'delete_request_{request_index}')]
209                 ])
210             )
211         else:
212             request_status[request_index] = 'X'
213             # Обновляем сообщение о заявке в админке
214             request, timestamp = requests[request_index]
215             formatted_time = timestamp.strftime('%Y-%m-%d %H:%M:%S')
216             await update.callback_query.edit_message_text(
217                 text=f"{request.replace(' ✓', '')} {request_status[request_index]} (время: {formatted_time})",
218                 reply_markup=InlineKeyboardMarkup([
219                     [InlineKeyboardButton(text="Изменить статус", callback_data=f'toggle_request_{request_index}']],
220                     [InlineKeyboardButton(text="Удалить заявку", callback_data=f'delete_request_{request_index}')]
221                 ])
222             )
223
224
225 async def confirm_delete_requests(update: Update) -> None:
226     keyboard = [
227         [InlineKeyboardButton(text="Удалить", callback_data='confirm_delete')],
228         [InlineKeyboardButton(text="Отмена", callback_data='cancel_delete')]
229     ]
230     reply_markup = InlineKeyboardMarkup(keyboard)
231     await update.callback_query.edit_message_text(text="Вы действительно хотите удалить все заявки?",
232                                                   reply_markup=reply_markup)
233

```

Рисунок 8 – Добавление статуса изменения заявок и их удаление

На последнем этапе допишем функционал кнопки «Удалить все заявки», чтобы администратор удалял все заявки и их статусы, а также кнопка «Удалить заявку» удаляла конкретную заявку и ее статус, и доделаем функционал кнопки «Отменить заявку», чтобы при ее нажатии появлялась кнопка «Начать заново». В конце пропишем блок `main` для запуска функции (Рис. 9).

```
235 async def delete_requests(update: Update) -> None:
236     global requests, request_status
237     requests.clear()
238     request_status.clear()
239     await update.callback_query.edit_message_text("Все заявки успешно удалены.")
240
241
242 async def delete_request(update: Update, request_index: int) -> None:
243     global requests, request_status
244     if 0 <= request_index < len(requests):
245         del requests[request_index]
246         del request_status[request_index]
247         # Переиндексация ключей request_status после удаления заявки
248         request_status = {i: status for i, status in enumerate(request_status.values())}
249         await update.callback_query.edit_message_text("Заявка успешно удалена.")
250     else:
251         await update.callback_query.edit_message_text("Ошибка удаления заявки.")
252
253
254 async def cancel_request(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
255     await update.callback_query.edit_message_text(text="Заявка отменена.",
256                                                    reply_markup=InlineKeyboardMarkup([
257                                                        [InlineKeyboardButton(text="Начать заново", callback_data='start')]
258                                                    ]))
259     context.user_data['state'] = None
260
261
262 def main() -> None:
263     application = ApplicationBuilder().token(TOKEN).build()
264
265     application.add_handler(CommandHandler(command="start", start))
266     application.add_handler(CallbackQueryHandler(button))
267     application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, handle_message))
268
269     application.run_polling()
270
271
272 if __name__ == '__main__':
273     main()
```

Рисунок 9 – Дописывание функционала кнопки и создание блока для запуска

Теперь запускаем бота и проверяем функциональность, чтобы запустить, достаточно перейти по ссылке от бота, в котором создавали токен.

Нажимаем на кнопки и проверяем. Дальнейшее описание работы бота будет с аккаунта администратора (Рис. 10).

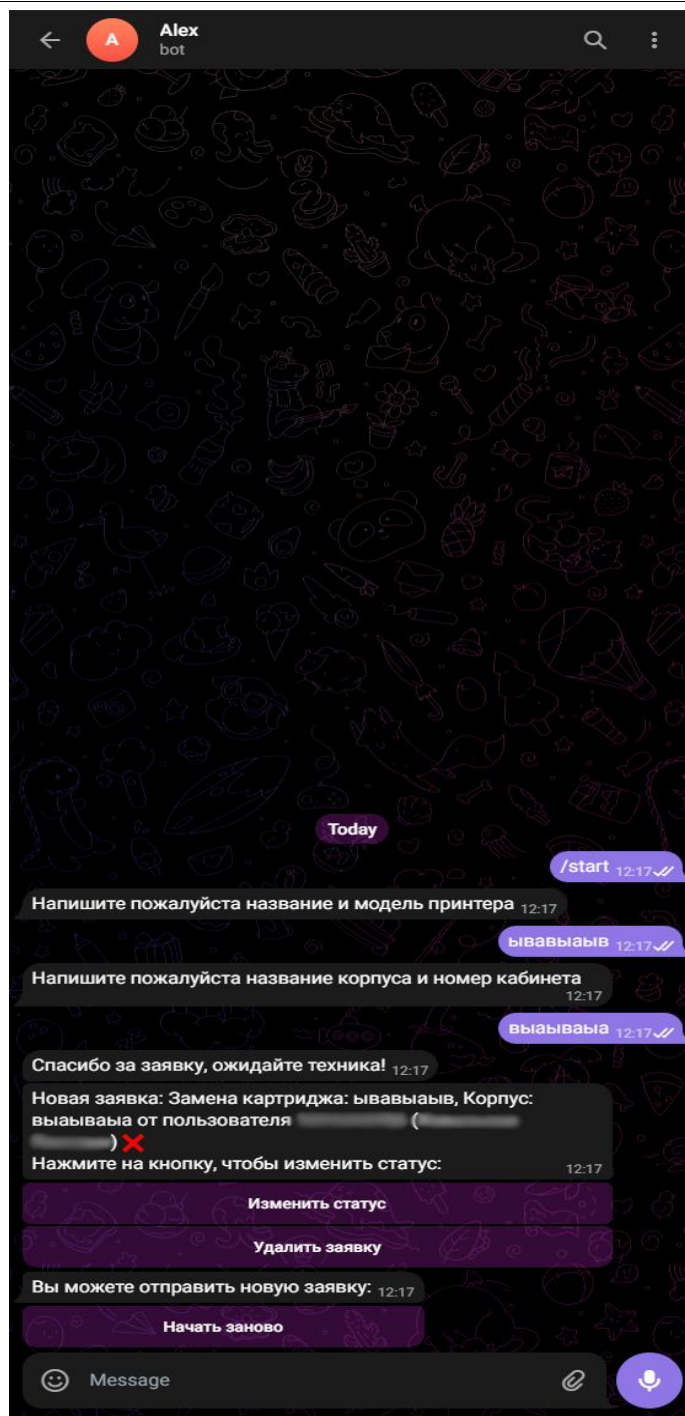


Рисунок 10 – проверка отправки заявки телеграмм-боту

В данном случае чат-бот работает верно, заявка сформирована, что и требовалось в работе чат-бота. В случае нехватки функционала работы бота, в любой момент его можно усовершенствовать.

4 Выводы

Подводя итог, можно сделать вывод, что использование чат-бота для автоматизации некоторых процессов это достаточно удобно, а разработка подобного чат-бота не потребует крайне много времени. В дальнейшем можно рассмотреть возможности расширения функциональности чат-бота,

добавление просмотра всех заявок от имени пользователя: сколько и какие заявки отправлял определенный пользователь за все время

Библиографический список

1. Бахтин И. В. Разработка чат-ботов для автоматизации бизнес-процессов // Форум молодых ученых. 2020. №. 2. С. 48-50.
2. Матвеева Н. Ю., Золотарюк А. В. Технологии создания и применения чат-ботов // Научные записки молодых исследователей. 2018. №. 1. С. 28-30.
3. Горячкин Б. С., Розен К. С., Бобров Д. В. Методология создания чат-ботов // Современная наука: актуальные проблемы теории и практики. 2023. 58-63 с.
4. Джанарсанам С. Разработка чат-ботов и разговорных интерфейсов. 2018. 340 с.