

KZ

Консультант Вячеслав (zerocode)


Премиум подписка на AI-ассистент

15:53

 YANDEXGPT отвечает:

✗ Доступ ограничен. Для использования бота необходимо оформить подписку.

[/buy](#) - купить подписку (200 руб/мес)

 Время генерации: 0.00 сек

15:53

Привет 15:53 ✓✓

[/buy](#) 15:53 ✓✓

Премиум подписка на AI-ассистент

Доступ ко всем функциям бота на 30 дней

200,00 RUB TEST INVOICE

15:53

Заплатить 200,00 RUB

Search

Консультант Вячеслав (zerocode)

Checkout (Test) ✕

Премиум подписка на AI-ассистент

Доступ ко всем функциям бота на 30 дней

Консультант Вячеслав (zerocode)

KZ

Консультант Вячеслав (zerocode)

Премиум подписка на AI-ассистент

Подписка

200,00 RUB

Total

200,00 RUB



Payment Method

Cancel

Pay 200,00 RUB

KZ

Консультант Вячеслав (zerocode)

17:02

YANDEXGPT отвечает: Здравствуйте! Чем я могу в...

YANDEXGPT отвечает:

❌ Доступ ограничен. Для использования бота необходимо оформить подписку.

[/buy](#) - купить подписку (200 руб/мес)

⌚ Время генерации: 0.00 сек

16:55

Привет 16:55 ✓✓

YANDEXGPT отвечает:

Здравствуйте! Чем я могу вам помочь?

⌚ Время генерации: 2.41 сек

17:02

Привет 17:02 ✓✓



Write a message...



[/yandex](#)

[/sber](#)

[/clear](#)

[/buy](#)



Update Telegram


```

9 import requests # HTTP-запросы к API
10 import datetime # Работа с датой и временем
11 from dotenv import load_dotenv # Для загрузки переменных окружения из .env файла
12
13
14 # Импорт компонентов Telegram API
15 from telegram import (
16     Update, # Объект обновления от Telegram
17     ReplyKeyboardMarkup, # Клавиатура с кнопками
18     KeyboardButton, # Кнопка клавиатуры
19     InlineKeyboardMarkup, # Инлайн-клавиатура
20     InlineKeyboardButton # Кнопка инлайн-клавиатуры
21 )
22 from telegram.ext import (
23     Updater, # Ядро для работы с Telegram API
24     CommandHandler, # Обработчик команд (начинающихся с /)
25     CallbackContext, # Контекст выполнения
26     MessageHandler, # Обработчик текстовых сообщений
27     Filters # Фильтры для обработчиков
28 )
29
30 from yookassa import Configuration, Payment
31 # Импорт компонентов Telegram API
32 from telegram import (
33     Update, # Объект обновления от Telegram
34     ReplyKeyboardMarkup, # Клавиатура с кнопками
35     KeyboardButton, # Кнопка клавиатуры
36     InlineKeyboardMarkup, # Инлайн-клавиатура
37     InlineKeyboardButton, # Кнопка инлайн-клавиатуры
38
39     LabeledPrice # Цена для платежей
40 )
41
42 # Создание логгера для текущего модуля
43 logger = logging.getLogger(__name__)
44
45 load_dotenv(dotenv_path='env_vars')
46 subscriptions = {}
47
48 def start(update: Update, context: CallbackContext) -> None:
49     logger.info(f'command /start gotten')
50
51     help_text = (
52         "Доступные команды:\n"
53         "/yandex - использовать YandexGPT\n"
54         "/sber - использовать SberAI (GigaChat)\n"
55         "/clear - очистить историю диалога\n\n"
56         "/buy - купить подписку (200 руб/мес)\n\n"
57         "Просто отправьте мне сообщение с вашим вопросом!"

```



```

48 def start(update: Update, context: CallbackContext) -> None:
49     # Просто отправьте мне сообщение ☞ вашим вопросом!
50
51
52
53
54
55
56
57
58
59 keyboard = [
60     [KeyboardButton("/yandex"), KeyboardButton("/sber")],
61     [KeyboardButton("/clear"), KeyboardButton("/buy")]
62 ]
63
64 update.message.reply_text(
65     help_text,
66     reply_markup=ReplyKeyboardMarkup(
67         keyboard,
68         resize_keyboard=True,
69         one_time_keyboard=False
70     )
71 )
72
73 class RussianAI:
74     # Конструктор класса
75     def __init__(self):
76         # Получение провайдера по умолчанию из переменных окружения
77         self.provider = os.getenv("DEFAULT_PROVIDER", "yandexgpt")
78         # Инициализация истории диалога как пустого списка
79         self.conversation_history = []
80         # Настройка выбранного провайдера
81         self.set_provider(self.provider)
82
83     def set_provider(self, provider: str):
84         self.provider = provider.lower() # Название провайдера переводим в нижний регистр
85         # Сброс истории диалога при смене провайдера
86         self.conversation_history = []
87
88         # Настройка параметров для YandexGPT
89         if self.provider == "yandexgpt":
90             # Получение API-ключа из переменных окружения
91             self.api_key = os.getenv("YANDEX_API_KEY")
92             # Получение идентификатора каталога Yandex Cloud
93             self.folder_id = os.getenv("YANDEX_FOLDER_ID")
94             # Получение модели (по умолчанию "yandexgpt-lite")
95             self.model = os.getenv("YANDEX_MODEL", "yandexgpt-lite")
96             # URL API YandexGPT
97             self.base_url = "https://llm.api.cloud.yandex.net/foundationModels/v1/completion"
98             # Проверка наличия обязательных ключей
99             if not self.api_key or not self.folder_id:
100                 # Запись ошибки в лог
101                 logger.error("Не заданы YANDEX_API_KEY и YANDEX_FOLDER_ID")
102                 return False # Возврат статуса ошибки
103             # Настройка параметров для SberAI (GigaChat)
104         elif self.provider == "sberai":

```



```

73 class RussianAI:
83     def set_provider(self, provider: str):
100         # Запись ошибки в лог
101         logger.error("Не заданы YANDEX_API_KEY и YANDEX_FOLDER_ID")
102         return False # Возврат статуса ошибки
103         # Настройка параметров для SberAI (GigaChat)
104     elif self.provider == "sberai":
105         # Получение API-ключа SberAI
106         self.api_key = os.getenv("SBER_API_KEY")
107         # Получение модели (по умолчанию "GigaChat:latest")
108         self.model = os.getenv("SBER_MODEL", "GigaChat:latest")
109         # URL API SberAI
110         self.base_url = "https://api.gigachat.dev/v1/chat/completions"
111         # Проверка наличия API-ключа
112         if not self.api_key:
113             logger.error("Не задан SBER_API_KEY")
114             return False
115         # Обработка неизвестного провайдера
116     else:
117         logger.error(f"Неизвестный провайдер: {provider}")
118         return False
119
120     # Запись информации о выбранном провайдере в лог
121     logger.info(f"Используется провайдер: {self.provider.upper()} ({self.model})")
122     return True # Успешное завершение настройки
123
124     # Метод для добавления сообщения в историю диалога
125     def add_message(self, role: str, content: str) -> None:
126         self.conversation_history.append({"role": role, "content": content})
127
128     # Основной метод для генерации ответа на пользовательский ввод
129     def generate_response(self, user_input: str, user_id: int):
130         # :param user_input: Входное сообщение пользователя
131         # :param user_id: ID пользователя для проверки подписки
132         # :return: Ответ AI или сообщение о необходимости подписки
133         if not self.check_subscription(user_id):
134             return ("X Доступ ограничен. Для использования бота необходимо оформить подписку.\n\n"
135                     "/buy - купить подписку (200 руб/мес)")
136
137         self.add_message("user", user_input)
138         try:
139             # Выбор соответствующего метода API в зависимости от провайдера
140             if self.provider == "yandexgpt":
141                 return self._yandex_request()
142             elif self.provider == "sberai":
143                 return self._sber_request()
144
145         except Exception as e:
146             return f"Ошибка API ({self.provider}): {str(e)}"

```



```

73 class RussianAI:
147
148     # Приватный метод для работы с API YandexGPT
149     def _yandex_request(self):
150         headers = {
151             "Authorization": f"Api-Key {self.api_key}", # API-ключ для аутентификации
152             "Content-Type": "application/json", # Тип содержимого
153             "x-folder-id": self.folder_id # Идентификатор каталога
154         }
155         yandex_messages = []
156
157         for msg in self.conversation_history:
158             yandex_messages.append({
159                 "role": msg["role"],
160                 "text": msg["content"] # Yandex использует "text" вместо "content"
161             })
162
163         # Формирование тела запроса (payload)
164         payload = {
165             "modelUri": f"gpt://{self.folder_id}/{self.model}", # URI модели
166             "completionOptions": {
167                 "stream": False, # Режим без потоковой передачи
168                 "temperature": 0.7, # Креативность ответов
169                 "maxTokens": 2000 # Максимальное количество токенов в ответе
170             },
171             "messages": yandex_messages # История диалога
172         }
173         try:
174             response = requests.post(
175                 self.base_url, # URL API
176                 headers=headers, # Заголовки
177                 json=payload, # Тело запроса в формате JSON
178                 timeout=30 # Таймаут запроса (30 секунд)
179             )
180             # Проверка статуса ответа
181             if response.status_code != 200:
182                 # Логирование ошибки при ненормальном статусе
183                 logger.error(f"Ошибка {response.status_code}: {response.text}")
184                 return f"Ошибка API: {response.text}"
185
186             data = response.json()
187             ai_reply = data["result"]["alternatives"][0]["message"]["text"]
188             self.add_message("assistant", ai_reply)
189             return ai_reply # Возврат сгенерированного ответа
190         except Exception as e:
191             return f"Ошибка соединения: {str(e)}"
192

```



```

73     class RussianAI:
193         def _sber_request(self):
195             "https://api.gigachat.dev/v1/oauth/token", # URL для аутентификации
196             headers={"Content-Type": "application/x-www-form-urlencoded"},
197             data={
198                 "grant_type": "urn:ietf:params:oauth:grant-type:jwt-bearer",
199                 "assertion": self.api_key, # Использование API-ключа в качестве JWT
200                 "scope": "GIGACHAT_API_PERS" # Область доступа
201             }
202         )
203         if auth_response.status_code != 200:
204             logger.error(f"Ошибка аутентификации SberAI: {auth_response.text}")
205             return "Ошибка аутентификации SberAI"
206         auth_data = auth_response.json()
207         # Получение токена доступа
208         access_token = auth_data["access_token"]
209         headers = {
210             "Authorization": f"Bearer {access_token}", # Использование токена доступа
211             "Content-Type": "application/json"
212         }
213     # }
214     payload = {
215         "model": self.model, # Идентификатор модели
216         "messages": self.conversation_history, # История диалога
217         "temperature": 0.7, # Креативность ответов
218         "max_tokens": 2000 # Максимальное количество токенов в ответе
219     }
220     response = requests.post(
221         self.base_url,
222         headers=headers,
223         json=payload,
224         timeout=30
225     )
226
227     if response.status_code != 200:
228         logger.error(f"Ошибка SberAI: {response.status_code} - {response.text}")
229         return f"Ошибка SberAI: {response.text}"
230     data = response.json()
231     ai_reply = data["choices"][0]["message"]["content"]
232     self.add_message("assistant", ai_reply)
233
234     return ai_reply
235
236     def clear_history(self):
237         self.conversation_history = []
238         logger.info("История диалога очищена") # Логирование события
239         return True # Подтверждение успешного выполнения

```



```

73     class RussianAI:
236         def clear_history(self):
237             self.conversation_history = []
238             logger.info("История диалога очищена") # Логирование события
239             return True # Подтверждение успешного выполнения
240
241         def check_subscription(self, user_id: int) -> bool:
242             sub = subscriptions.get(user_id)
243             if sub and sub['end_date'] > datetime.datetime.now():
244                 return True
245             return False
246
247         def switch_to_yandex(update: Update, context: CallbackContext) -> None:
248             if ai_assistant.set_provider("yandexgpt"):
249                 Click to add a breakpoint message.reply_text(
250                     + ☒ Переключено на YandexGPT ({ai_assistant.model}),
251                     reply_markup=create_keyboard() # Обновление клавиатуры
252                 )
253             else:
254                 update.message.reply_text("❌ Не удалось переключиться на YandexGPT")
255
256         def switch_to_sber(update: Update, context: CallbackContext) -> None:
257             if ai_assistant.set_provider("sberai"):
258                 update.message.reply_text(
259                     f"☒ Переключено на SberAI ({ai_assistant.model})",
260                     reply_markup=create_keyboard() # Обновление клавиатуры
261                 )
262             else:
263                 update.message.reply_text("❌ Не удалось переключиться на SberAI")
264
265         def clear_history(update: Update, context: CallbackContext) -> None:
266             if ai_assistant.clear_history():
267                 update.message.reply_text(
268                     "🗑 История диалога очищена!",
269                     reply_markup=create_keyboard() # Обновление клавиатуры
270                 )
271             else:
272                 update.message.reply_text("❌ Не удалось очистить историю")
273
274         def handle_message(update: Update, context: CallbackContext) -> None:
275             user_input = update.message.text
276             if user_input.startswith('/'):
277                 return
278             user_id = update.message.from_user.id
279             context.bot.send_chat_action(
280                 chat_id=update.effective_chat.id, # ID текущего чата
281                 action="typing"
282             )
283

```


c53_1.py > ...

```
275 def handle_message(update: Update, context: CallbackContext) -> None:
276     ,
284     start_time = time.time()
285
286     try:
287         response = ai_assistant.generate_response(user_input, user_id)
288         elapsed_time = time.time() - start_time
289         formatted_response = (
290             f"🤖 {ai_assistant.provider.upper()} отвечает:\n\n"
291             f"{response}\n\n"
292             f"⌚ Время генерации: {elapsed_time:.2f} сек"
293         )
294         update.message.reply_text(
295             formatted_response,
296             reply_markup=create_keyboard() # Отправка с клавиатуры команд
297         )
298     except Exception as e:
299         logger.error(f"Ошибка генерации ответа: {str(e)}")
300         update.message.reply_text(
301             "🚨 Произошла ошибка при генерации ответа. Попробуйте позже.",
302             reply_markup=create_keyboard()
303         )
304
305 def create_keyboard():
306     keyboard = [
307         [KeyboardButton("/yandex"), KeyboardButton("/sber")],
308         [KeyboardButton("/clear"), KeyboardButton("/buy")]
309     ]
310     return ReplyKeyboardMarkup(
311         keyboard,
312         resize_keyboard=True,
313         one_time_keyboard=False
314     )
315
316 # subsription&buying
317 def buy_subscription(update: Update, context: CallbackContext) -> None:
318     try:
319         user_id = update.message.from_user.id
320         if ai_assistant.check_subscription(user_id):
321             update.message.reply_text("✅ 📌 вас уже есть активная подписка!")
322             return
323         price = int(os.getenv("SUBSCRIPTION_PRICE", 20000)) # Сумма в копейках (200 руб)
324         provider_token = os.getenv("TELEGRAM_PROVIDER_TOKEN") # Токен платежного провайдера
325
326         logger.info(f"Creating payment for user {user_id}, price: {price}, provider: {provider_token}")
327         context.bot.send_invoice(
328             chat_id=update.effective_chat.id,
329             title="Премиум подписка на AI-ассистент",
330             description="Доступ ко всем функциям бота на 30 дней",
```



```

317 def buy_subscription(update: Update, context: CallbackContext) -> None:
331     payload=f"subscription_{user_id}",
332     provider_token=provider_token,
333     currency="RUB",
334     prices=[LabeledPrice("Подписка", price)],
335     start_parameter="subscription"
336 )
337 except Exception as e:
338     logger.exception("Ошибка в buy_subscription")
339     update.message.reply_text("❌ Ошибка при создании платежа. Попробуйте позже.")
340
341 def successful_payment(update: Update, context: CallbackContext) -> None:
342     try:
343         user_id = update.message.from_user.id
344         payment_info = update.message.successful_payment
345
346         logger.info(f"Успешный платеж получен: {payment_info}")
347         end_date = datetime.datetime.now() + datetime.timedelta(
348             days=int(os.getenv("SUBSCRIPTION_DAYS", 30)))
349
350         subscriptions[user_id] = {
351             'start_date': datetime.datetime.now(),
352             'end_date': end_date,
353             'status': 'active'
354         }
355         update.message.reply_text(
356             f"👉 Подписка успешно активирована до {end_date.strftime('%d.%m.%Y')}!\n\n"
357             "Теперь вы можете использовать все возможности бота!"
358         )
359     except Exception as e:
360         logger.exception("Ошибка в successful_payment")
361         update.message.reply_text(
362             "❌ Ошибка активации подписки. Пожалуйста, свяжитесь с поддержкой."
363         )
364
365 def precheckout_handler(update: Update, context: CallbackContext) -> None:
366     query = update.pre_checkout_query
367     try:
368         context.bot.answer_pre_checkout_query(
369             pre_checkout_query_id=query.id,
370             ok=True
371         )
372         logger.info(f"PreCheckout подтвержден для платежа: {query.invoice_payload}")
373     except Exception as e:
374         logger.error(f"Ошибка в precheckout_handler: {str(e)}")
375         context.bot.answer_pre_checkout_query(
376             pre_checkout_query_id=query.id,
377             ok=False,
378             error_message="Произошла ошибка при обработке платежа"

```



```
365 def precheckout_handler(update: Update, context: CallbackContext) -> None:
378     error_message= произошла ошибка при обработке платежа
379 )
380
381 def error_handler(update: Update, context: CallbackContext) -> None:
382     logger.error(msg="Глобальная ошибка:", exc_info=context.error)
383     if update and update.message:
384         update.message.reply_text(
385             "⚠ Произошла системная ошибка. Разработчики уже уведомлены. Попробуйте позже."
386         )
387
388 def main():
389     updater = Updater(TOKEN)
390     dispatcher = updater.dispatcher
391     dispatcher.add_handler(CommandHandler("start", start))
392     dispatcher.add_handler(CommandHandler("yandex", switch_to_yandex))
393     dispatcher.add_handler(CommandHandler("sber", switch_to_sber))
394     dispatcher.add_handler(CommandHandler("clear", clear_history))
395     dispatcher.add_handler(CommandHandler("buy", buy_subscription)) # Добавляем обработчик команды buy
396
397     dispatcher.add_handler(MessageHandler(Filters.text & ~Filters.command, handle_message))
398     dispatcher.add_handler(PreCheckoutQueryHandler(precheckout_handler)) # ОБЯЗАТЕЛЬНО для платежей
399     dispatcher.add_handler(MessageHandler(Filters.successful_payment, successful_payment))
400     dispatcher.add_error_handler(error_handler)
401
402     updater.start_polling()
403
404     print("Бот успешно запущен. Используйте /start в Telegram для начала работы.")
405     updater.idle()
406
407 ai_assistant = RussianAI()
408
409 if __name__ == '__main__':
410     main()
```