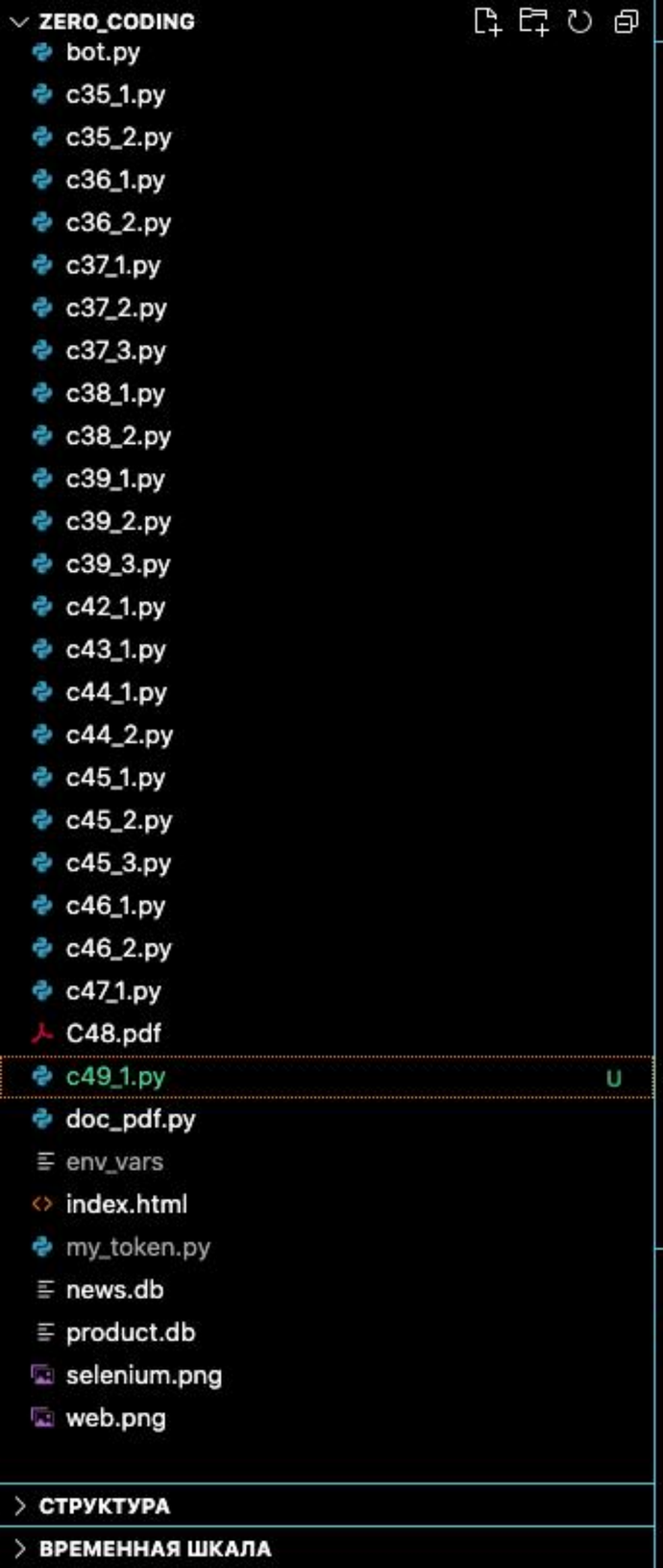

c49_1.py > ai_assistant

```python
import requests  # Для отправки HTTP-запросов к внешним API
import logging  # Для настройки системы логирования
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup, ReplyKeyboardMarkup, KeyboardButton, ReplyKey
from telegram.ext import Updater, CommandHandler, CallbackQueryHandler, CallbackContext, MessageHandler, Filters, Conv
import os  # Для работы с операционной системой и переменными окружения
import time  # Для замера времени выполнения операций
from dotenv import load_dotenv  # Для загрузки переменных окружения из .env файла

# Настройка системы логирования
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',  # Формат записи логов
    level=logging.INFO  # Уровень логирования (INFO и выше)
)
# Создание логгера для текущего модуля
logger = logging.getLogger(__name__)

load_dotenv(dotenv_path='env_vars')

class RussianAI:
    # Конструктор класса
    def __init__(self):
        # Получение провайдера по умолчанию из переменных окружения
        self.provider = os.getenv("DEFAULT_PROVIDER", "yandexgpt")
        # Инициализация истории диалога как пустого списка
        self.conversation_history = []
        # Настройка выбранного провайдера
        self.set_provider(self.provider)

    def set_provider(self, provider: str):
        self.provider = provider.lower() # Название провайдера переводим в нижний регистр
        # Сброс истории диалога при смене провайдера
        self.conversation_history = []

        # Настройка параметров для YandexGPT
        if self.provider == "yandexgpt":
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ

```
(.env) slava@172-10-22-236 zero_coding %  /usr/bin/env /Users/slava/Documents/python\ developer/zero_codin
g/.env/bin/python /Users/slava/.vscode/extensions/ms-python.debugpy-2024.8.0-darwin-x64/bundled/libs/debug
py/adapter/../../debugpy/launcher 49851 -- /Users/slava/Documents/python\ developer/zero_coding/c49_1.py
/Users/slava/Documents/python developer/zero_coding/.env/lib/python3.8/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning:
 urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com
/urllib3/urllib3/issues/3020
  warnings.warn(
2025-06-20 22:10:54,110 - __main__ - INFO - Используется провайдер: YANDEXGPT (yandexgpt-lite)
(.env) slava@172-10-22-236 zero_coding %
```

```
19  class RussianAI:
29      def set_provider(self, provider: str):
35          if self.provider == "yandexgpt":
36              # Получение API-ключа из переменных окружения
37              self.api_key = os.getenv("YANDEX_API_KEY")
38              # Получение идентификатора каталога Yandex Cloud
39              self.folder_id = os.getenv("YANDEX_FOLDER_ID")
40              # Получение модели (по умолчанию "yandexgpt-lite")
41              self.model = os.getenv("YANDEX_MODEL", "yandexgpt-lite")
42              # URL API YandexGPT
43              self.base_url = "https://llm.api.cloud.yandex.net/foundationModels/v1/completion"
44              # Проверка наличия обязательных ключей
45              if not self.api_key or not self.folder_id:
46                  # Запись ошибки в лог
47                  logger.error("Не заданы YANDEX_API_KEY и YANDEX_FOLDER_ID")
48                  return False # Возврат статуса ошибки
49              # Настройка параметров для SberAI (GigaChat)
50          elif self.provider == "sberai":
51              # Получение API-ключа SberAI
52              self.api_key = os.getenv("SBER_API_KEY")
53                  # Получение модели (по умолчанию "GigaChat:latest")
54              self.model = os.getenv("SBER_MODEL", "GigaChat:latest")
55              # URL API SberAI
56              self.base_url = "https://api.gigachat.dev/v1/chat/completions"
57              # Проверка наличия API-ключа
58              if not self.api_key:
59                  logger.error("Не задан SBER_API_KEY")
60                  return False
61          # Обработка неизвестного провайдера
62          else:
63              logger.error(f"Неизвестный провайдер: {provider}")
64              return False
65
```

```python
class RussianAI:
    def set_provider(self, provider: str):
        # Запись информации о выбранном провайдере в лог
        logger.info(f"Используется провайдер: {self.provider.upper()} ({self.model})")
        return True  # Успешное завершение настройки

    # Метод для добавления сообщения в историю диалога
    def add_message(self, role: str, content: str):…

    # Основной метод для генерации ответа на пользовательский ввод
    def generate_response(self, user_input: str):
        self.add_message("user", user_input)
        try:
            # Выбор соответствующего метода API в зависимости от провайдера
            if self.provider == "yandexgpt":
                return self._yandex_request()
            elif self.provider == "sberai":
                return self._sber_request()

        except Exception as e:
            return f"❌ Ошибка API ({self.provider}): {str(e)}"

    # Приватный метод для работы с API YandexGPT
    def _yandex_request(self):
        headers = {
            "Authorization": f"Api-Key {self.api_key}",  # API-ключ для аутентификации
            "Content-Type": "application/json",  # Тип содержимого
            "x-folder-id": self.folder_id  # Идентификатор каталога
        }
        yandex_messages = []

        for msg in self.conversation_history:
            yandex_messages.append({
                "role": msg["role"],
                "text": msg["content"]  # Yandex использует "text" вместо "content"
            })
```

```
19  class RussianAI:
88      def _yandex_request(self):
```

```
102         # Формирование тела запроса (payload)
103         payload = {
104             "modelUri": f"gpt://{self.folder_id}/{self.model}",  # URI модели
105             "completionOptions": {
106                 "stream": False,  # Режим без потоковой передачи
107                 "temperature": 0.7,  # Креативность ответов
108                 "maxTokens": 2000  # Максимальное количество токенов в ответе
109             },
110             "messages": yandex_messages  # История диалога
111         }
112         try:
113             response = requests.post(
114                 self.base_url,  # URL API
115                 headers=headers,  # Заголовки
116                 json=payload,  # Тело запроса в формате JSON
117                 timeout=30  # Таймаут запроса (30 секунд)
118             )
119             # Проверка статуса ответа
120             if response.status_code != 200:
121                 # Логирование ошибки при ненормальном статусе
122                 logger.error(f"Ошибка {response.status_code}: {response.text}")
123                 return f"Ошибка API: {response.text}"
124 
125             data = response.json()
126             ai_reply = data["result"]["alternatives"][0]["message"]["text"]
127             self.add_message("assistant", ai_reply)
128             return ai_reply  # Возврат сгенерированного ответа
129         except Exception as e:
130             return f"Ошибка соединения: {str(e)}"
131 
132     def _sber_request(self):
133         auth_response = requests.post(
134             "https://api.gigachat.dev/v1/oauth/token",  # URL для аутентификации
135             headers={"Content-Type": "application/x-www-form-urlencoded"},
136             data={
137                 "grant_type": "urn:ietf:params:oauth:grant-type:jwt-bearer",
138                 "assertion": self.api_key,  # Использование API-ключа в качестве JWT
139                 "scope": "GIGACHAT_API_PERS"  # Область доступа
140             }
141         )
```

```
19  class RussianAI:
132     def _sber_request(self):
142         if auth_response.status_code != 200:
143             logger.error(f"Ошибка аутентификации SberAI: {auth_response.text}")
144             return "Ошибка аутентификации SberAI"
145         auth_data = auth_response.json()
146         # Получение токена доступа
147         access_token = auth_data["access_token"]
148         headers = {
149             "Authorization": f"Bearer {access_token}",  # Использование токена доступа
150             "Content-Type": "application/json"
151         }
152     # }
153         payload = {
154             "model": self.model,  # Идентификатор модели
155             "messages": self.conversation_history,  # История диалога
156             "temperature": 0.7,  # Креативность ответов
157             "max_tokens": 2000  # Максимальное количество токенов в ответе
158         }
159         response = requests.post(
160             self.base_url,
161             headers=headers,
162             json=payload,
163             timeout=30
164         )
165
166         if response.status_code != 200:
167             logger.error(f"Ошибка SberAI: {response.status_code} - {response.text}")
168             return f"Ошибка SberAI: {response.text}"
169         data = response.json()
170         ai_reply = data["choices"][0]["message"]["content"]
171         self.add_message("assistant", ai_reply)
172
173         return ai_reply
174
175     def clear_history(self):
176         self.conversation_history = []
177         logger.info("История диалога очищена")  # Логирование события
178         return True  # Подтверждение успешного выполнения
179
180 ai_assistant = RussianAI()
```