



SISTEMA DE ATENDIMENTO AUTOMATIZADO USANDO CHATBOT

Aluna: Wandressa Reis

DESCRIÇÃO

O projeto consiste em um sistema de chatbot automatizado, desenvolvido com uma arquitetura de microsserviços, que opera em contêineres Docker.



FERRAMENTAS

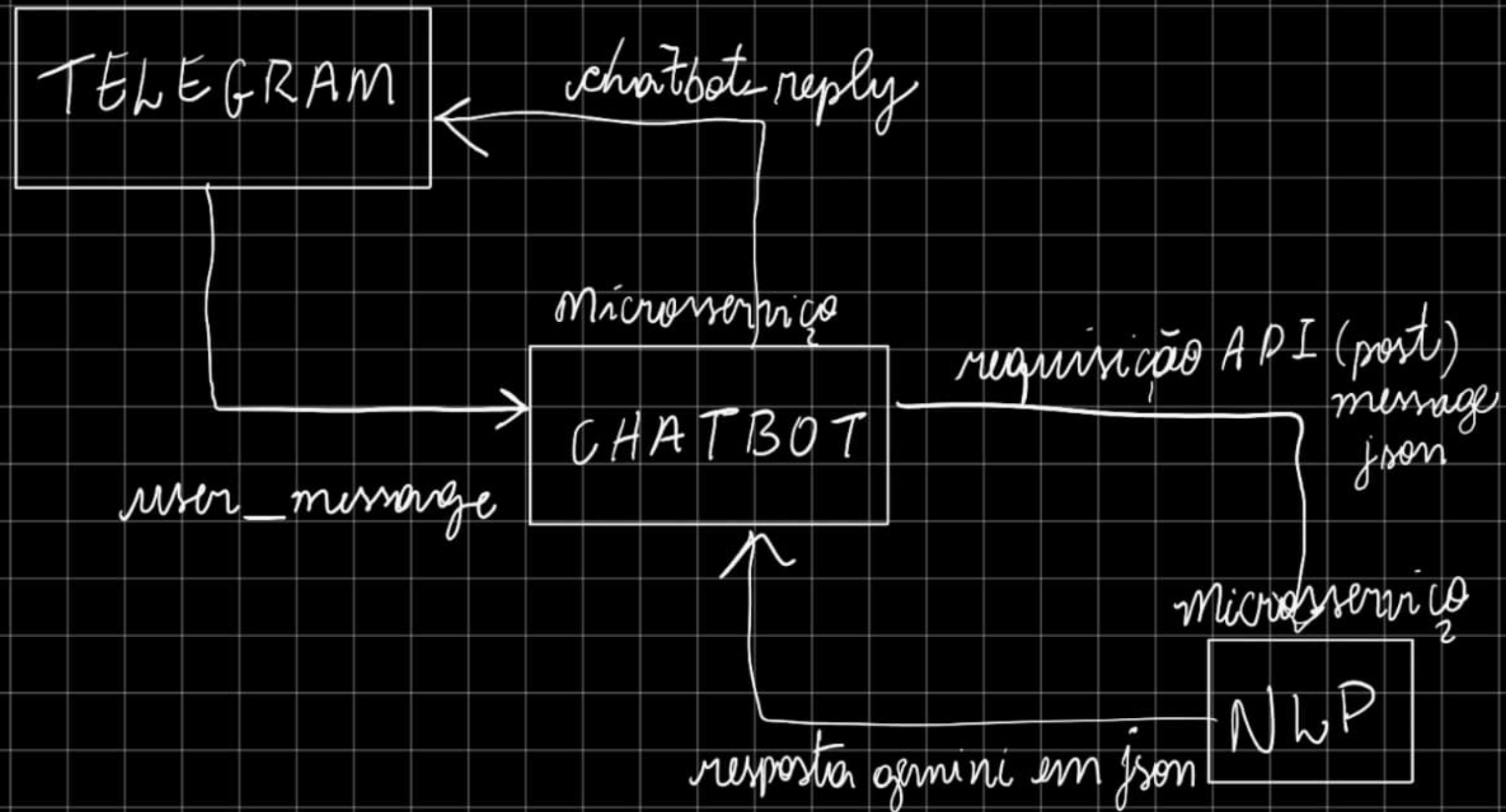
- Gemini
- FastAPI
- Docker
- Chatbot Telegram

MICROSSERVIÇOS

- Chatbot
- NLP

COMO FUNCIONAM?





1

BOT


```
1  import os
2  from telegram import Update
3  from telegram.ext import ApplicationBuilder, CommandHandler, MessageHandler, filters, CallbackContext
4  from dotenv import load_dotenv
5  import json
6  import requests
7
8  # Carregar variáveis de ambiente
9  load_dotenv()
10
11 def requisicao_api(message):
12     url = 'http://nlp_service:8002/response'
13     data = {"message": message }
14     data_json = json.dumps(data)
15     headers = {'content-type': 'application/json'}
16     response = requests.post(url, json=data, headers=headers)
17     print(response.json())
18     return response.json()
19
20 # Função de start
21 async def start(update: Update, context: CallbackContext):
22     user_id = update.message.chat_id
23     nome = update.message.chat.first_name
24     # salvar_usuario(user_id, nome)
25     await update.message.reply_text(f'Bem-vindo, {nome}! Como posso ajudar?')
```

```
27 # Função para lidar com mensagens
28 async def handle_message(update: Update, context: CallbackContext):
29     user_message = update.message.text
30     try:
31         # Processa a mensagem usando a função de processamento com o Gemini
32         # resposta = process_message(user_message)
33         await update.message.reply_text(requisicao_api(user_message))
34     except Exception as e:
35         await update.message.reply_text(f"Desculpe, ocorreu um erro: {e}")
36
37 # Configuração do bot
38 def start_bot():
39     token = os.getenv("TELEGRAM_TOKEN")
40     application = ApplicationBuilder().token(token).build()
41
42     application.add_handler(CommandHandler("start", start))
43     application.add_handler(MessageHandler(filters.TEXT & (~filters.COMMAND), handle_message))
44
45     application.run_polling()
46
47 # Inicializa o bot do Telegram
48 start_bot()
```

2

NLP

```
1  import os
2  import google.generativeai as genai
3  from dotenv import load_dotenv
4  from fastapi import FastAPI
5  from pydantic import BaseModel, Field
6
7
8  app = FastAPI()
9
10
11  # Carregar variáveis de ambiente
12  load_dotenv()
13  # Configure a chave de API
14  genai.configure(api_key=os.getenv("GEMINI_API_KEY"))
15
16  class Message(BaseModel):
17      message: str = Field(..., example="Hello World")
18
19  @app.post("/response")
20  def process_message(message: Message):
21      print(message)
22      try:
23          # Criar um modelo generativo com o nome correto do modelo
24          model = genai.GenerativeModel("gemini-1.5-flash")
25
26          # Gerar conteúdo baseado na mensagem do usuário
27          response = model.generate_content(message.message)
28
29          # Retornar o texto gerado
30          return response.text
31      except Exception as e:
32          return f"Ocorreu um erro ao processar sua solicitação: {e}"
```


 docker-compose.yml

```
1  version: '3.8'
2
3  services:
4    bot_service:
5      build: ./src/services/bot_service
6      environment:
7        # - GEMINI_API_KEY=${GEMINI_API_KEY}
8        - TELEGRAM_TOKEN=${TELEGRAM_TOKEN}
9      ports:
10       - "8000:8000"
11      # depends_on:
12      #   - nlp_service
13      #   - db_service
14
15    nlp_service:
16      build: ./src/services/nlp_service
17      environment:
18        - GEMINI_API_KEY=${GEMINI_API_KEY} # Pa
19      ports:
20       - "8002:8002"
```