

# Desenvolvimento de um Chatbot para Auxílio em Python

Angel Flavius Alves Negri  
Matrícula: 24/1038110

Matheus De Melo Fellet  
Matrícula: 22/2015201

Matheus Duarte Da Silva  
Matrícula: 21/1062277

**Index Terms**—chatbot educacional, Flask, redes de computadores, comunicação cliente-servidor.

**Resumo**—Este relatório apresenta o desenvolvimento de um chatbot para auxiliar no ensino da linguagem Python. A aplicação foi construída com o framework Flask e integrada à API da OpenAI, possibilitando interação em linguagem natural. A análise de pacotes com o Wireshark demonstrou a comunicação correta entre cliente e servidor. O sistema mostrou-se funcional, promovendo um ambiente interativo de aprendizado.

## I. INTRODUÇÃO

As redes de computadores constituem a base da comunicação digital, viabilizando a troca de dados entre dispositivos e aplicações distribuídas. Dentro dessa área, compreender os mecanismos de comunicação entre cliente e servidor, bem como os protocolos que os sustentam, é essencial para o desenvolvimento de sistemas interativos e eficientes. Este trabalho foi desenvolvido no contexto da disciplina de Redes de Computadores, com o objetivo de aplicar na prática os conceitos estudados em sala, como encapsulamento, camadas do modelo TCP/IP e análise de tráfego.

O projeto consistiu na criação de um chatbot educacional para auxiliar o aprendizado da linguagem Python. A aplicação foi construída com o microframework Flask, integrada à API da OpenAI para interpretação de linguagem natural, e executada em um ambiente de rede local. Para verificar o funcionamento da comunicação entre os componentes do sistema, foram capturados e analisados pacotes utilizando a ferramenta Wireshark.

Este relatório está estruturado da seguinte forma: a Seção II apresenta os fundamentos teóricos que sustentam o experimento; a Seção III descreve o ambiente experimental e as análises dos resultados, com foco na comunicação cliente-servidor e na interpretação dos pacotes capturados; por fim, a Seção IV apresenta as conclusões.

## II. FUNDAMENTAÇÃO TEÓRICA

### A. Camada de Transporte

A camada de transporte, conforme [1], provê serviços de comunicação fim-a-fim entre aplicações em dispositivos distintos. No presente experimento, a escolha da tecnologia de comunicação foi abstruída pela utilização do framework *Flask*, que encapsula internamente a implementação dos protocolos de transporte, gerenciando automaticamente a criação de sockets, conexões TCP, bem como o envio e recebimento

de dados entre cliente e servidor — além de outros aspectos do ciclo de requisição HTTP.

### B. Linguagem, Bibliotecas e Interface de Usuário

A linguagem escolhida foi *Python* por sua simplicidade sintática, ampla comunidade de suporte e grande variedade de bibliotecas voltadas para desenvolvimento web e aprendizado de máquina [2]. Dentre essas bibliotecas, destaca-se o *Flask*, um microframework leve e eficiente, que abstrai detalhes da camada de transporte e facilita a criação de rotas HTTP [3]. Essa escolha permitiu foco na lógica da aplicação.

Para o armazenamento local de dados, como histórico de interações e configurações, utilizou-se o *SQLite* como banco de dados, com o *SQLAlchemy* como interface ORM devido à sua simplicidade e integração nativa com Python. A comunicação com o modelo de linguagem foi realizada por meio da API da *OpenAI*, utilizando requisições HTTP assíncronas para garantir respostas ágeis, e a renderização de conteúdo dinâmico foi aprimorada com a biblioteca *Markdown*, que transforma textos brutos em HTML formatado, garantindo uma apresentação clara e organizada para o usuário.

A interface gráfica da aplicação foi desenvolvida com foco na simplicidade. O layout possui fundo preto, com o título centralizado na parte superior e uma imagem representando o projeto. Abaixo, encontra-se o campo de *login* — que serve para o usuário inserir seu nome e senha — depois segue um campo de *notas*, onde é possível escrever dúvidas ou comandos. Essas mensagens são então processadas e respondidas pelo chatbot especializado na linguagem *Python* conforme mostra a Figura:

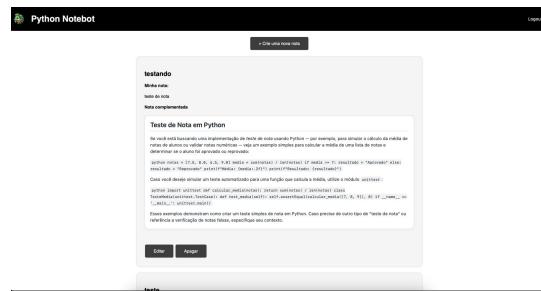


Figura 1: interpretação do prompt.

Esse comportamento é viabilizado pela biblioteca da *OpenAI* e seu modelo de linguagem, que interpreta o prompt

enviado pelo usuário e retorna uma resposta contextualizada [4].

### C. Padrões de Projeto e Técnicas Utilizadas

O desenvolvimento da aplicação incorporou padrões de projeto e técnicas específicas para garantir organização do código, facilitar manutenções futuras e permitir escalabilidade. Os principais adotados foram:

- **Singleton:** Implementado nas classes DB e AssistantBot, esse padrão garante que apenas uma instância de cada classe seja criada ao longo da execução. Isso é feito pela sobreescrita do método `__new__`, assegurando o compartilhamento da mesma instância em diferentes partes do sistema, o que é útil para o controle centralizado de recursos como o banco de dados e o assistente de respostas.
- **Facade:** A classe DB também exerce o papel de fachada para as operações com o banco de dados. Ela encapsula métodos como `create_all`, `commit_session`, `add_data` e `delete_data`, simplificando o uso do SQLAlchemy para o restante da aplicação.
- **Aggregation (Composição Fraca):** A classe Note mantém uma referência a uma instância da classe AssistantBot, recebida via construtor. Essa relação caracteriza uma agregação, pois embora a Note utilize o AssistantBot para gerar respostas automáticas, ambas as classes podem existir de forma independente.
- **Factory Method (uso indireto):** Métodos estáticos como `User.get_user(name: str)` funcionam como *Factory Method*, pois encapsula a criação e a recuperação de objeto User a partir do banco de dados.

## III. AMBIENTE EXPERIMENTAL E ANÁLISE DE RESULTADOS

### A. Descrição do Cenário

O experimento foi executado em uma rede local com topologia em *estrela*, onde um *switch* central conecta os dispositivos cliente e servidor. Essa configuração permite uma comunicação eficiente entre os nós.

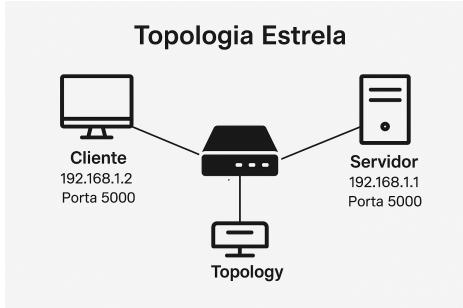


Figura 2: Topologia de rede experimental.

Do ponto de vista de hardware, os testes foram realizados em computadores pessoais com suporte a ambiente Linux e Windows. As ferramentas utilizadas incluíram navegadores

como Google Chrome e Brave para acesso à interface da aplicação; o editor VSCode para desenvolvimento; e utilitários como Wireshark, CMD (no Windows) e Terminal (no Linux) para análise de pacotes e execução de comandos no sistema.

O servidor da aplicação foi implementado utilizando o microframework Flask, configurado para escutar requisições em todas as interfaces de rede da máquina local, por meio do parâmetro `host='0.0.0.0'` e da porta 5000. Essa configuração permite que qualquer dispositivo conectado à mesma rede consiga acessar o sistema, simulando um cenário realista de comunicação entre cliente e servidor em uma rede local.

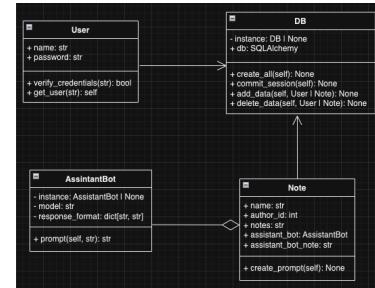


Figura 3: Esboço da arquitetura da aplicação, destacando a interação entre cliente e servidor.

Durante os testes, o usuário acessava a interface via navegador e interagia com o chatbot por meio de um campo de entrada de texto. As mensagens eram enviadas ao servidor Flask, que processava o conteúdo com o auxílio da API da OpenAI e retornava uma resposta apropriada. O monitoramento do tráfego de rede foi realizado com o Wireshark, possibilitando observar os pacotes trafegando via protocolo HTTP.

### B. Análise com Wireshark (Quadro 1)

**A. Identificação do tipo e versão de software:** A versão do framework no servidor é **Flask 3.1.0**, conforme mostrado no campo `Server`



Figura 4: User-Agent e Server

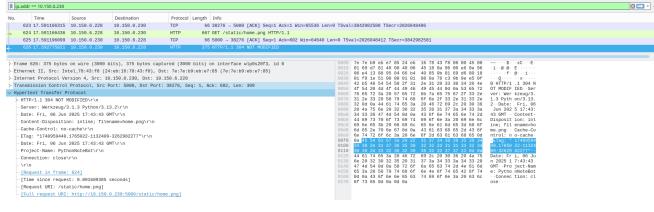
O cliente utiliza um navegador em um dispositivo Android, identificado pelo campo `User-Agent`, o que confirma o uso de celular.

**B. Endereços IP dos clientes e servidor:** O IP do servidor é 10.150.0.230, conforme mostrado na (Figura 5a). A conexão inicial do celular é 10.150.6.228, conforme a (Figura 5b). A diante o endereço IP sofrerá alteração devido

à reinicialização do servidor. O IP do cliente, observado nos pacotes, será 10.150.11.206, Figura 13.

```
~ ifconfig | grep inet
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    inet 10.150.0.230 netmask 255.255.240.0 broadcast 10.150.15.255
    inet6 fe80::aa87:e5fa:dc07:b59e prefixlen 64 scopeid 0x20<link>
```

(a) Endereços de IP do servidor



(b) Conexão inicial do cliente móvel

Figura 5: Análise dos endereços IP e conexão do cliente celular com o servidor.

### Análise Visual Detalhada dos Pacotes

Capturas selecionadas dos pacotes analisados entre servidor, celular e cliente desktop.

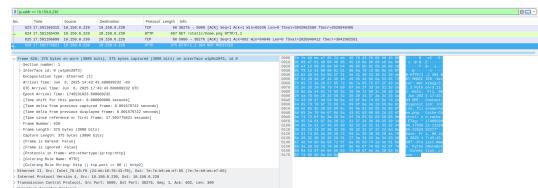


Figura 6: 1. Frame (S→C)

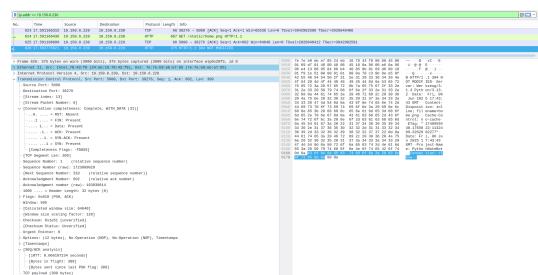


Figura 7: 2. TCP (S→C)

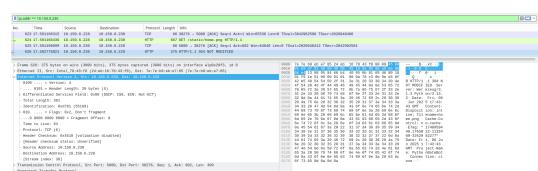


Figura 8: 3. IPv4 (S→C)

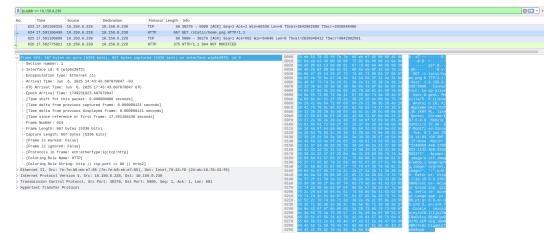


Figura 9: 4. GET Celular

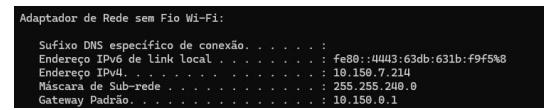


Figura 10: 5. IP máquina externa

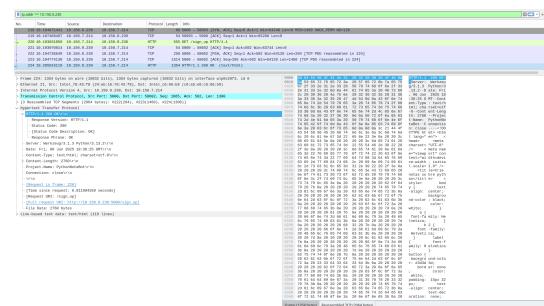


Figura 11: 6. Cliente PC

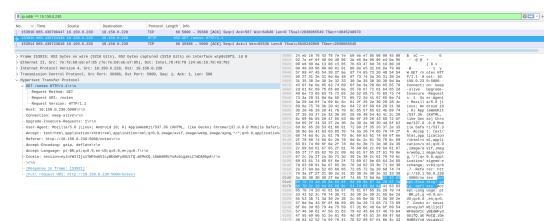


Figura 12: 7. Nota via Cel

*Observação:* Mesmo com filtros de protocolo e porta aplicados no Wireshark, não foi possível remover totalmente pacotes como ACK, TCP ou QUIC. Eles foram ignorados durante a análise das mensagens relevantes.

### C. Carga útil dos pacotes entre cliente e servidor:

Na Figura 14, é apresentada a carga útil da requisição HTTP do tipo POST enviada pelo cliente para o endpoint /new. Essa requisição utiliza o cabeçalho Content-Type: application/x-www-form-urlencoded e carrega dois campos principais: name=Listas e notes=Quero aprender mais sobre listas. Esses dados representam, respectivamente, o título e o conteúdo de uma nova nota a ser armazenada na aplicação, o que está de acordo com o funcionamento esperado do sistema desenvolvido [5].

Já a Figura 15 mostra a resposta HTTP enviada pelo servidor após o recebimento da requisição. Nela, observa-se uma resposta HTTP/1.1 200 OK, com conteúdo HTML renderizando a nota recém-criada. O conteúdo da resposta inclui os dados fornecidos pelo cliente, evidenciando que a informação foi recebida, interpretada e armazenada corretamente pelo backend.

Portanto, conclui-se que a comunicação entre cliente e servidor ocorre conforme o funcionamento esperado da aplicação. Os dados transmitidos seguem o formato estipulado e a resposta do servidor valida a integridade da operação, comprovando que a nota foi corretamente processada e exibida na interface web.

```
→ ~ ifconfig | grep inet
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    inet 10.150.11.206 netmask 255.255.240.0 broadcast 10.150.15.255
    inet6 fe80::fe07:b59e prefixlen 64 scopeid 0x20<link>
```

Figura 13: Endereço de IP alterado

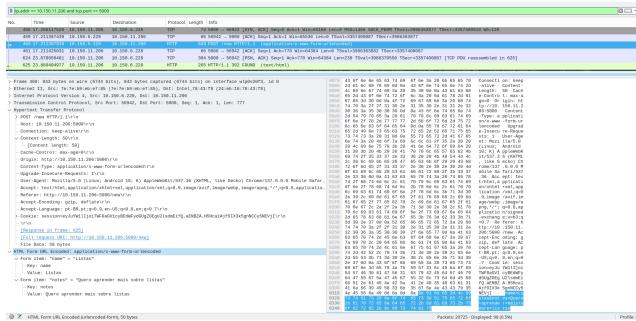


Figura 14: Carga útil da requisição POST enviada pelo cliente.

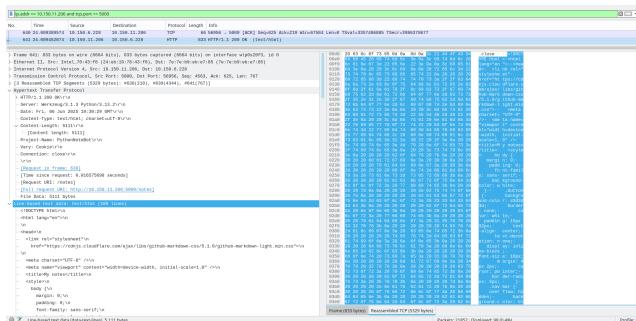


Figura 15: Resposta do servidor com o conteúdo renderizado da nota.

#### D. Análise dos cabeçalhos e encapsulamento

As Figuras 16 a 19 mostram a sequência de cabeçalhos das camadas Ethernet, IP, TCP e HTTP. Isso evidencia o processo de encapsulamento: a aplicação gera os dados (HTTP), que são encapsulados pelas camadas de transporte, rede e enlace, como discutido em sala de aula.

Na inspeção dos dados brutos de um pacote HTTP no Wireshark, observam-se claramente os cabeçalhos de todas as

camadas envolvidas no processo de comunicação. O cabeçalho Ethernet, localizado no início do pacote, contém os endereços MAC de origem e destino, além do campo de tipo que indica o protocolo da camada superior (IPv4, neste caso).

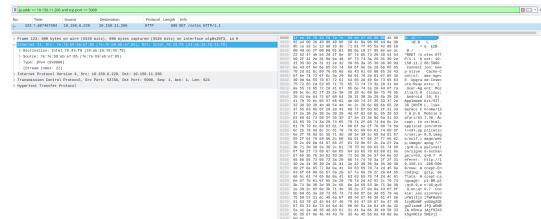


Figura 16: Cabeçalho da camada de enlace (Ethernet)

Em seguida, o cabeçalho IP encapsula o segmento TCP. Ele traz informações como os endereços IP de origem e destino, o protocolo da camada superior (código 6 para TCP), TTL e outros campos que auxiliam no roteamento do pacote pela rede.

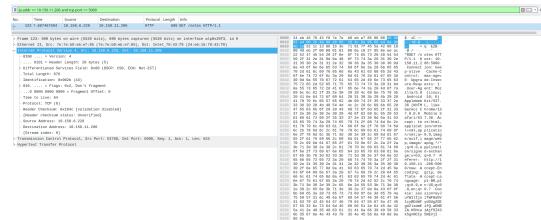


Figura 17: Cabeçalho da camada de rede (IP)

Na camada de transporte, o cabeçalho TCP apresenta as portas de origem e destino (por exemplo, 443 para HTTPS), os números de sequência e reconhecimento, bem como as flags de controle, como SYN, ACK e FIN. Esses campos são fundamentais para garantir a confiabilidade da transmissão e o controle do fluxo de dados entre cliente e servidor.

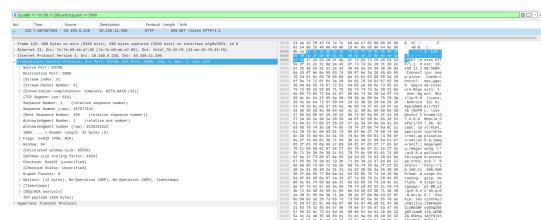


Figura 18: Cabeçalho da camada de transporte (TCP)

Por fim, localiza-se o cabeçalho HTTP, que pertence à camada de aplicação. Ele especifica o método da requisição (como GET ou POST), a URL do recurso solicitado e diversos cabeçalhos adicionais como Host, User-Agent e Content-Type. Esses dados representam efetivamente a informação gerada pela aplicação e são o núcleo da comunicação de interesse.

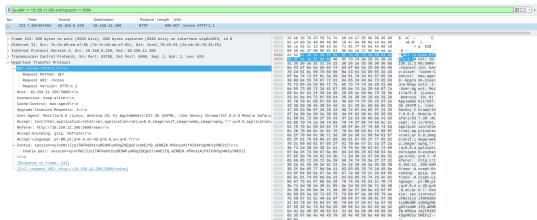


Figura 19: Cabeçalho da camada de aplicação (HTTP)

Esse empilhamento de informações — onde cada camada encapsula a anterior — reflete o modelo teórico de encapsulamento do modelo TCP/IP estudado em sala de aula. Cada camada adiciona seu próprio cabeçalho, formando uma estrutura hierárquica que permite o correto transporte, roteamento, entrega e interpretação dos dados. Na recepção, o processo inverso (desencapsulamento) é realizado, permitindo que a aplicação destino interprete os dados corretamente [1].

#### IV. CONCLUSÃO

O desenvolvimento da aplicação permitiu aplicar, na prática, os principais conceitos da disciplina de Redes de Computadores, como comunicação cliente-servidor, camadas de protocolo e encapsulamento. Através da análise com o Wireshark, foi possível visualizar a estrutura completa dos pacotes transmitidos, identificando os cabeçalhos das camadas Ethernet, IP, TCP e HTTP. A aplicação demonstrou funcionamento correto, com integridade na troca de dados e respostas coerentes do chatbot. O ambiente experimental foi bem-sucedido e refletiu um cenário realista de rede local.

#### REFERÊNCIAS

- [1] J. F. Kurose and K. W. Ross, *Redes de Computadores e a Internet: Uma Abordagem Top-Down*, 8th ed. São Paulo: Pearson, 2020.
- [2] M. Lutz, *Learning Python*, 5th ed. Sebastopol, CA: O'Reilly Media, 2013.
- [3] F. Project, “Flask documentation,” Disponível em: <https://flask.palletsprojects.com/>, 2025, acesso em: 5 jun. 2025.
- [4] OpenAI, “Openai API documentation,” Disponível em: <https://platform.openai.com/docs/introduction>, 2024, acesso em: 5 jun. 2025.
- [5] R. T. Fielding and J. Reschke, “Hypertext transfer protocol – HTTP/1.1,” Disponível em: <https://tools.ietf.org/html/rfc7231>, 2014, acesso em: 6 jun. 2025.

#### APÊNDICE

##### A. Vídeo pitch

```
1 https://youtu.be/H08VOfHgzXc
```

Listing 1: pitch técnico da aplicação