Universidade de Brasília
Faculdade do Gama - FGA

FUNDAMENTOS DE REDES DE COMPUTADORES
Docente: Fernando William Cruz

Projeto de Pesquisa - Servidor de Diálogos Usando Camada de Aplicação

Gabrielly Assunção Rodrigues - 200018442
João Gabriel Antunes S. Medeiros e Silva - 170013651
Fernando Vargas Teotonio de Oliveira - 180016491
Luiz Henrique Fernandes Zamprogno - 190033681
Miguel Matos Costa de Frias Barbosa - 211039636

20/12/2023

1. Introdução

No âmbito do aprendizado sobre os protocolos de camada de aplicação mais recentes, como HTTP e WebSockets, este projeto teve como objetivo não apenas compreender a arquitetura subjacente a esses protocolos, mas também aplicá-la na construção de uma aplicação prática. A proposta consistiu em desenvolver um servidor de diálogos que oferecesse um ambiente interativo para interações em tempo real entre clientes, utilizando o modelo cliente/servidor.

A infraestrutura da aplicação foi planejada, considerando que os clientes eram navegadores/browsers e o servidor seria implementado utilizando o web server Apache. A configuração do DNS seguiu detalhes específicos disponíveis no Moodle da disciplina. O servidor Apache foi configurado para receber conexões tanto via HTTP quanto via HTTPS, promovendo segurança nas comunicações.

Quanto à aplicação em si, ela foi estruturada de maneira a contemplar os requisitos do projeto, contando com tela de login, cadastro e chats por tema, onde é possível realizar comunicação via mensagens e vídeo. Ao ingressar no servidor de diálogos, os usuários escolhiam temas de seu interesse para participar de discussões. O processo de cadastro envolvia a solicitação de um nome, e-mail e senha.

2. Metodologia

A abordagem metodológica adotada para a realização deste projeto foi essencial para garantir a eficiência, colaboração e sucesso na implementação do servidor de diálogos. As atividades foram estruturadas considerando as diretrizes definidas nas especificações do trabalho, contemplando desde a organização das reuniões até a entrega dos artefatos finais.

2.1 Reuniões

Reunião 1 - Distribuição do Trabalho

- Data: 15/12/2023
- Início: 21:37
- Fim: 22:10
- Assunto: A primeira reunião focou na distribuição de tarefas entre os membros da equipe, considerando habilidades individuais e áreas de interesse. Este passo inicial foi crucial para estabelecer responsabilidades claras e garantir uma colaboração equitativa.

Reunião 2 - Alinhamento e Integração Front/Back

- Data: 17/12/2023
- Início: 21:20
- Fim: 23:00
- Assunto: Realizamos a integração entre as partes front-end e back-end, discutindo estratégias para evitar retrabalhos e atrasos

Reunião 3 - Correção de Erros e Validação de Usuário

- Data: 19/12/2023
- Início: 21:02
- Fim: 3:32
- Assunto: Essa reunião concentrou-se na identificação e correção de erros na implementação, além de implementar a validação do usuário para garantir a segurança e integridade dos dados.

Reunião 4 - Implementação de Chat, Estruturação do Relatório e Gravação de Vídeo

- Data: 20/12/2023
- Início: 16:50
- Fim: 23:00
- Assunto: Esta reunião abrangeu a implementação específica da funcionalidade de chat de vídeo, a estruturação do relatório final e o início da gravação do vídeo explicativo. A gestão simultânea dessas atividades foi planejada para otimizar o tempo e garantir a conclusão bem-sucedida de cada componente.

2.2 Desenvolvimento e Documentação

Para garantir a clareza e compreensão do código e da solução implementada, adotamos as seguintes práticas:

- Identação e Comentários: Todo o código desenvolvido foi devidamente identado e comentado. Essa prática facilita a compreensão de cada linha de código e permite uma colaboração mais eficaz entre os membros da equipe.

- Documentação do Cliente: Scripts no cliente foram documentados de maneira abrangente, proporcionando instruções claras sobre a configuração e uso da aplicação. Essa documentação foi disponibilizados no GitHub para fácil acesso e referência.

2.3 Controle de Versão e Colaboração

GitHub: Utilizamos a plataforma GitHub para controle de versão do código. Cada membro da equipe contribuiu para o repositório compartilhado, permitindo o rastreamento de alterações, resolução de conflitos e colaboração efetiva.

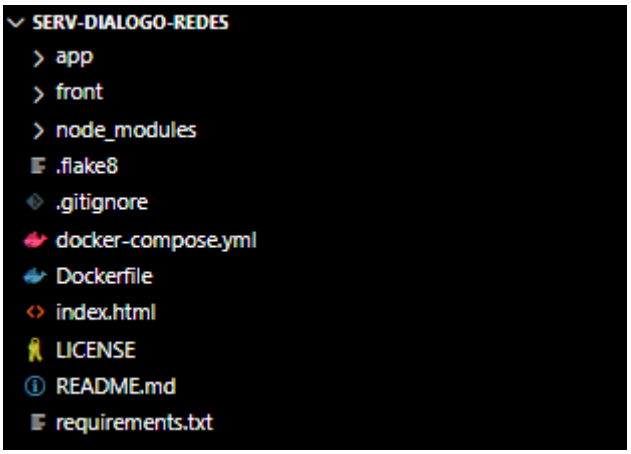
Balanceamento de Atividades: A distribuição de tarefas levou em consideração as habilidades individuais de cada membro, promovendo um equilíbrio na carga de trabalho. O acompanhamento contínuo durante as reuniões assegurou que todas as atividades estivessem alinhadas com o cronograma estabelecido.

Essa metodologia integrada, abordando desde a distribuição inicial de tarefas até práticas de desenvolvimento e documentação, contribuiu para o progresso consistente do projeto e para a entrega de um produto final de qualidade.

3. Arquitetura do Projeto

O projeto "SERV-DIALOGO-REDES" foi desenvolvido utilizando uma arquitetura moderna e escalável, com a aplicação frontend construída em React.js e o backend implementado em Python. A seguir, apresentamos uma visão geral da organização e componentes essenciais do projeto.

3.1 Estrutura de Diretórios



app:
- routes:
  - `crud_topics.py`: Implementa operações CRUD para gerenciamento de tópicos.
  - `crud_user.py`: Responsável pelas operações CRUD relacionadas aos usuários.
  - `topic_router.py`: Roteamento específico para tópicos.
  - `user_router.py`: Roteamento relacionado aos usuários.
- schemas: Contém os esquemas de dados utilizados na aplicação.
- `auth.py`: Implementa funcionalidades relacionadas à autenticação.
- `database.py`: Configurações e conexão com o banco de dados.
- `main.py`: Ponto de entrada principal da aplicação.
- `websocket.py`: Lida com a implementação do WebSocket.

front:
- build: Contém os artefatos gerados após o processo de construção do frontend.
- node_modules: Dependências do Node.js para o desenvolvimento frontend.
- public: Arquivos públicos acessíveis pelo frontend.
- src:
  - `components:` Componentes React reutilizáveis.
  - `pages:` Páginas principais da aplicação.
  - `App.css`: Estilos globais da aplicação React.
  - `App.js`: Componente principal da aplicação React.
  - Outros arquivos relacionados ao frontend.

- node_modules: Dependências do Node.js para o desenvolvimento frontend.
- .flake8: Configurações para o linter Flake8.
- .gitignore: Lista de arquivos e diretórios ignorados pelo controle de versão Git.
- docker-compose.yml: Configurações para orquestração de containers Docker.
- Dockerfile: Instruções para a construção da imagem Docker da aplicação.
- index.html: Página HTML principal da aplicação frontend.
- LICENSE: Documento de licença do projeto.
- README.md: Documentação principal do projeto.
- requirements.txt: Lista de dependências Python da aplicação backend.

3.2 Dependências

O projeto utiliza uma variedade de bibliotecas e frameworks essenciais para garantir a eficiência e segurança da aplicação. Algumas das principais dependências incluem:

- React.js: Biblioteca JavaScript para construção de interfaces de usuário interativas.
- FastAPI (0.105.0): Framework assíncrono para construção de APIs em Python.
- WebSockets (12.0): Biblioteca para implementação de comunicação WebSocket.
- SQLAlchemy (2.0.23): Ferramenta de mapeamento objeto-relacional para interação com o banco de dados.
- Pydantic (2.5.2): Biblioteca para validação e serialização de dados.
- Uvicorn (0.24.0.post1): Servidor ASGI para execução da aplicação.

3.3 Backend e Frontend

O backend da aplicação, localizado na pasta "app," é responsável pela lógica de negócios, manipulação de dados e comunicação WebSocket. O frontend, encontrado em "front," é construído em React.js e concentra-se na interface do usuário, oferecendo uma experiência amigável e responsiva.

3.4 Tecnologias Utilizadas

O projeto incorpora tecnologias modernas como React.js para o desenvolvimento frontend, proporcionando uma interface dinâmica e interativa, enquanto o backend utiliza FastAPI para uma construção eficiente de APIs assíncronas em Python. Essa combinação garante uma arquitetura robusta, escalável e orientada às melhores práticas de desenvolvimento de software.

4. Implementação projeto

O desenvolvimento da solução com WebSockets é central para a eficácia da comunicação em tempo real na aplicação "SERV-DIALOGO-REDES". Toda a implementação, tanto no lado do cliente quanto no servidor, está contida no repositório ***https://github.com/SFernandoS/serv-dialogo-redes*** . Abaixo, detalhamos a estrutura e os aspectos-chave dessa implementação, atendendo às diretrizes estabelecidas.

4.1 Repositório

Todo o código relacionado à implementação de WebSockets pode ser encontrado no repositório ***https://github.com/SFernandoS/serv-dialogo-redes*** . A organização do repositório é estruturada para garantir clareza, facilitando a navegação e compreensão do código.

4.2 Estrutura Geral

```
SERV-DIALOGO-REDES
│
├── app
│   ├── routes
│   │   ├── crud_topics.py
│   │   ├── crud_user.py
│   │   ├── topic_router.py
│   │   └── user_router.py
│   │
│   ├── schemas
│   ├── auth.py
│   ├── database.py
│   ├── main.py
│   └── websocket.py
│
├── front
│   ├── build
│   ├── node_modules
│   ├── public
│   └── src
│       ├── App.css
│       ├── App.js
│       └── ...
│
├── node_modules
├── .flake8
```

```
├── .gitignore
├── docker-compose.yml
├── Dockerfile
├── index.html
├── LICENSE
├── README.md
├── requirements.txt
└── ...
```

4.3 Implementação do Websockts

A lógica da implementação do lado do servidor, representada pelo código a seguir, é crucial para o correto funcionamento da comunicação via WebSockets na aplicação.

```python
from typing import Dict, List
from fastapi import WebSocket


class ConnectionManager:
    def __init__(self):
        self.active_connections: Dict[str, WebSocket] = {}
        self.topic_rooms: Dict[int, List[str]] = {}

    async def connect(self, websocket: WebSocket, user_id: str):
        await websocket.accept()
        self.active_connections[user_id] = websocket

    def disconnect(self, user_id: str):
        if user_id in self.active_connections:
            del self.active_connections[user_id]
            for room in self.topic_rooms.values():
                if user_id in room:
                    room.remove(user_id)

    async def send_private_message(self, message: str, receiver_id: str):
        receiver = self.active_connections.get(receiver_id)
        if receiver:
            await receiver.send_text(message)

    async def join_topic(self, user_id: str, topic_id: int):
        if topic_id not in self.topic_rooms:
            self.topic_rooms[topic_id] = []
```

```
        self.topic_rooms[topic_id].append(user_id)

    async def send_topic_message(self, message: str, topic_id: int):
        if topic_id in self.topic_rooms:
            for user_id in self.topic_rooms[topic_id]:
                user_socket = self.active_connections.get(user_id)
                if user_socket:
                    await user_socket.send_text(message)
```

- `ConnectionManager` Class:
  - `__init__` Method: Inicializa a instância do gerenciador de conexões com dois dicionários, um para conexões ativas (`active_connections`) e outro para salas de tópicos (`topic_rooms`).

  - `connect` Method: Estabelece uma nova conexão WebSocket, aceitando a conexão e registrando o WebSocket ativo associado ao ID do usuário.

  - `disconnect` Method: Desconecta um usuário identificado por `user_id`. Remove a conexão ativa e o usuário das salas de tópicos.

  - `send_private_message` Method: Envia uma mensagem privada de um usuário para outro com base nos IDs fornecidos.

  - `join_topic` Method: Permite que um usuário entre em uma sala de tópicos específica, associando o ID do usuário à sala de tópicos correspondente.

  - `send_topic_message` Method: Envia uma mensagem para todos os usuários na sala de tópicos especificada.

Observações:
- A utilização de dicionários (`active_connections` e `topic_rooms`) facilita o acesso eficiente às informações relevantes.

- Os métodos `send_private_message`, `join_topic`, e `send_topic_message` operam com base nas conexões ativas e nas salas de tópicos gerenciadas pela classe.

- A implementação fornece uma estrutura sólida para o gerenciamento de conexões e comunicação em tempo real entre usuários na aplicação, cumprindo os requisitos estabelecidos.

4.4 Boas Práticas na Implementação

Todo o código presente no repositório segue boas práticas de codificação:

- Identação e Comentários: O código está devidamente identado e comentado, tornando-o compreensível para outros desenvolvedores.

- Origem de Funções: Indicação clara de funções que podem ter sido derivadas de outros projetos, promovendo transparência no desenvolvimento.

A consulta ao repositório proporciona uma visão abrangente e detalhada da implementação de WebSockets na aplicação, facilitando a análise e compreensão. .

5. Conclusão

Ao longo deste projeto, a equipe SERV-DIALOGO-REDES empreendeu esforços significativos na implementação de um ambiente de diálogos em tempo real usando WebSockets, culminando em resultados valiosos e aprendizados significativos.

5.1 Resultados Alcançados:

Organização da infraestrutura:

Status: Concluído

Observações: As configurações do servidor Apache e do DNS foram realizadas com sucesso, estabelecendo uma base sólida para a aplicação.

Configuração do DNS para acesso via URL:

Status: Concluído

Observações: O DNS foi configurado, possibilitando o acesso à aplicação por meio de uma URL específica.

Configuração do servidor WEB para HTTPS:

Status: Concluído

Observações: O servidor Apache foi configurado para suportar conexões tanto HTTP quanto HTTPS, garantindo uma camada adicional de segurança.

Funcionalidades básicas da aplicação:

Status: Concluído

Observações: As funcionalidades essenciais, como o cadastro de usuários, apresentação do catálogo e o estabelecimento/encerramento de diálogos, foram implementadas com sucesso.

Diálogos simultâneos:

Status: Concluído

Observações: A aplicação suporta a realização de dois ou mais diálogos simultâneos, proporcionando uma experiência dinâmica aos usuários.

Diálogos considerando os três tipos citados (chat, video-call e ambos) usando websockets:

Status: Concluído

Observações: A implementação bem-sucedida dos três tipos de diálogos (chat, video-call e ambos) utilizando WebSockets contribui para uma comunicação versátil entre os usuários.

5.2 Lições aprendidas:

5.2.1 Individuais:

Gabrielly Assunção Rodrigues:

Aprendi a importância vital dos WebSockets na implementação de comunicação em tempo real, percebendo como essa tecnologia é fundamental para estabelecer diálogos eficientes em aplicações web. Essa experiência também ampliou meu entendimento sobre o papel crucial das redes de computadores na troca de informações entre clientes e servidores.

Fernando Vargas Teotonio de Oliveira:

Ganhei uma compreensão profunda sobre WebSockets e sua integração com redes de computadores. Essa aprendizagem fortaleceu meu conhecimento sobre a importância desses protocolos na transmissão eficaz de dados e na comunicação entre sistemas distribuídos.

João Gabriel Antunes S. Medeiros e Silva:

Minha participação no projeto proporcionou uma visão aprofundada sobre a gestão de conexões WebSocket e a importância da eficiência na arquitetura cliente-servidor. Além disso, aprendi sobre a dinâmica das redes de computadores, especialmente no contexto de diálogos em tempo real.

Luiz Henrique Fernandes Zamprogno:

Entendi de forma abrangente como WebSockets são fundamentais na criação de soluções interativas em redes de computadores. A experiência prática na configuração de infraestrutura também me proporcionou uma compreensão mais profunda sobre os desafios e considerações em projetos dessa natureza.

Miguel Matos Costa de Frias Barbosa:

Minha atuação no desenvolvimento do lado do cliente e integração me proporcionou uma compreensão prática da importância dos WebSockets nas interações em tempo real. Além disso, ganhei insights sobre como redes de computadores desempenham um papel crucial na transmissão eficiente de dados entre o front-end e o back-end.

5.2.2 Aprendidas pelo Grupo:

- Importância dos WebSockets: Todos compreendemos a importância crítica dos WebSockets na implementação de diálogos em tempo real. A aplicação prática desses protocolos contribuiu significativamente para nosso entendimento da dinâmica das comunicações em rede.

- Desafios e Considerações em Redes de Computadores: Ao enfrentarmos desafios na implementação do projeto, aprendemos sobre a complexidade das redes de computadores e como considerações adequadas são necessárias para garantir a eficiência da comunicação entre clientes e servidores.

- Troca Eficiente de Dados: Aprendemos sobre a necessidade de uma troca eficiente de dados entre sistemas distribuídos, reconhecendo como essa eficiência é crítica para proporcionar uma experiência de usuário sem interrupções.

Essas lições individuais e do grupo destacam o enriquecimento do conhecimento sobre WebSockets e redes de computadores, aspectos essenciais para o sucesso da aplicação desenvolvida.

5.3 Considerações Finais:

O projeto SERV-DIALOGO-REDES ofereceu uma experiência prática e enriquecedora, permitindo à equipe explorar a fundo os conceitos de WebSockets, configuração de servidores e integração front/back. As lições aprendidas refletem a dedicação de cada membro, contribuindo para um entendimento mais profundo das tecnologias envolvidas e suas aplicações práticas. A equipe está satisfeita com os resultados alcançados e otimista em relação à qualidade da solução entregue.

Referências Bibliográficas:

1. TANENBAUM, Andrew S., WETHERALL, David. *Redes de Computadores*. São Paulo: Pearson Prentice Hall, 2011.

2. GRIFFITHS, David. *Head First HTML and CSS*. O'Reilly Media, 2012.

3. ALLEN, Ethan, et al. Real-Time Communication with WebRTC: Peer-to-Peer in the Browser. O'Reilly Media, 2014.

4. LOWENTHAL, Doug. WebSockets: Lightweight Client-Server Communications. O'Reilly Media, 2015.

5. ALMES, Guy, et al. RFC 791: Internet Protocol. IETF, 1981.

6. GÖTZ, Sebastian, et al. Mastering Websockets: Build Modern Web Applications with the Real-time Power of WebSockets. Packt Publishing, 2015.

7. CHERKASOV, Artemij. Network Programming with Go: Essential Skills for Using and Securing Networks. Packt Publishing, 2017.

8. WANG, Wallace. Network Security Essentials: Applications and Standards. Cengage Learning, 2016.

9. RAGHAVAN, Sriram, et al. TCP/IP Sockets in C: Practical Guide for Programmers. Morgan Kaufmann, 2009.

10. FRANKEL, Ricardo. High Performance Browser Networking: What every web developer should know about networking and web performance. O'Reilly Media, 2013.

11. HALL, Mark L., NEIL, Greg. Core Web Programming. Prentice Hall, 2001.

12. ANDERSON, Rick. Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley, 2008.