RELATÓRIO - LABORATÓRIO DE WEB SERVICES COM C++

Equipe:

Mônica Maria Rodrigues de Castro

Maryana Moraes Sousa

Disciplina: Sistemas Distribuídos

Professor: Rafael Braga

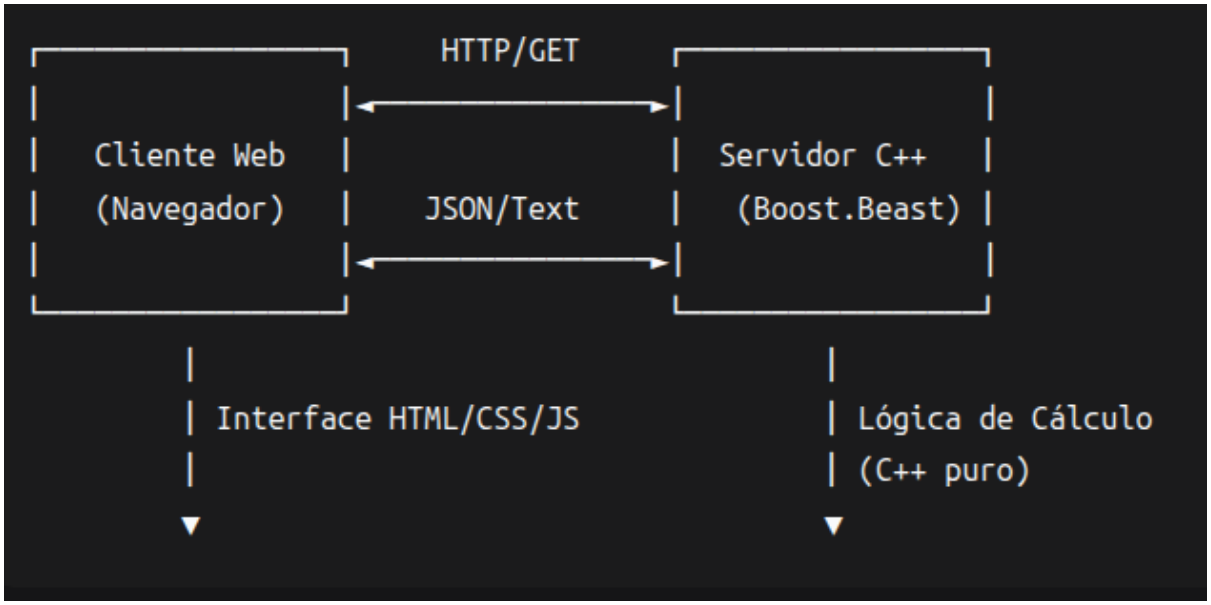
Universidade Federal do Ceará - Campus de Quixadá

OBJETIVOS DO LABORATÓRIO:

1. Implementar um servidor Web Services em C++ usando a biblioteca Boost.Beast
2. Criar uma API REST com endpoints para operações matemáticas básicas
3. Desenvolver uma interface gráfica web para interação com o serviço
4. Compreender os princípios de comunicação cliente-servidor em sistemas distribuídos
5. Analisar as diferenças entre implementações em Java (Spring Boot) e C++

ARQUITETURA DO SISTEMA:

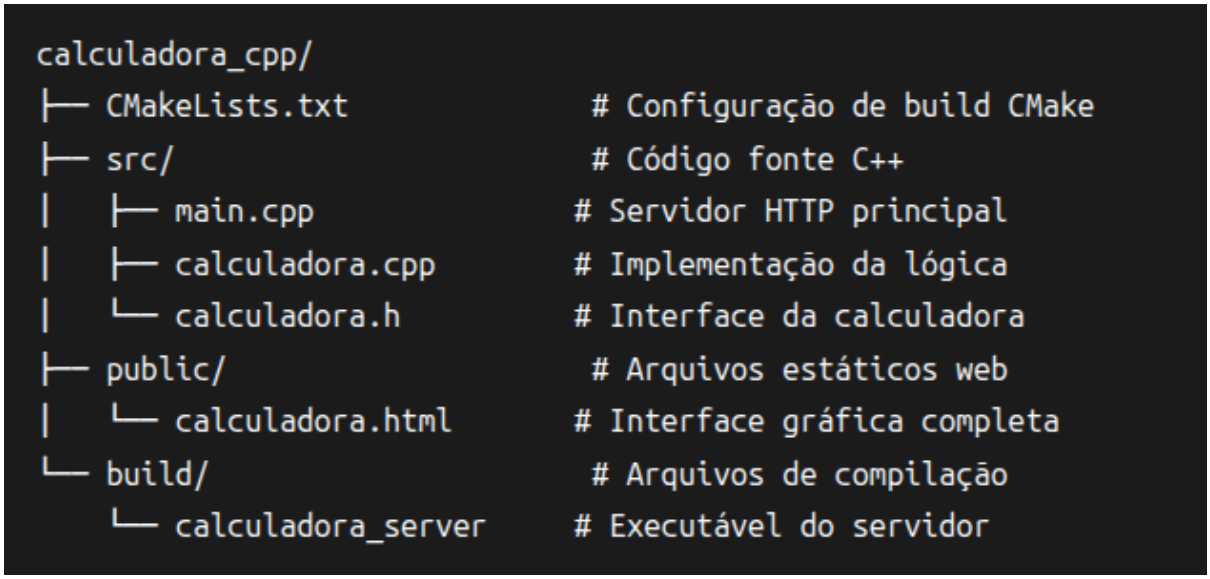
Diagrama da Arquitetura:



Componentes do Sistema:

1. Servidor HTTP (C++/Boost.Beast)
   - Escuta na porta 8080
   - Processa requisições HTTP GET
   - Implementa roteamento de URLs
   - Gerencia conexões de rede assíncronas
2. API REST
   - 4 endpoints bem definidos
   - Parâmetros via path (/operacao/num1/num2)
   - Respostas em formato texto simples
3. Interface Web (HTML/CSS/JavaScript)
   - Página responsiva e moderna
   - Comunicação via Fetch API
   - Validação de entrada do usuário
4. Lógica de Negócio (C++)
   - Operações matemáticas básicas
   - Tratamento de erros (divisão por zero)
   - Formatação de resultados

ESTRUTURA DO PROJETO:



API REST - ENDPOINTS IMPLEMENTADOS:

1. Endpoint de Soma

GET /somar/{numero1}/{numero2}

Exemplo:

curl http://localhost:8080/somar/15/3

Resposta:

Resultado: 18.00

2. Endpoint de Subtração

GET /subtrair/{numero1}/{numero2}

Exemplo:

curl http://localhost:8080/subtrair/20/7

Resposta:

Resultado: 13.00

3. Endpoint de Multiplicação

GET /multiplicar/{numero1}/{numero2}

Exemplo:

curl http://localhost:8080/multiplicar/6/8

Resultado:

Resultado: 48.00

4. Endpoint de Divisão

GET /dividir/{numero1}/{numero2}

Exemplo:

curl http://localhost:8080/dividir/100/4

Resultado:

Resultado: 25.00

CÓDIGO PRINCIPAL - SERVIDOR HTTP

Estrutura do Código C++:

```cpp
// 1. Includes e namespaces
#include <boost/beast/core.hpp>
#include <boost/beast/http.hpp>
#include <boost/asio.hpp>

// 2. Lógica da calculadora
std::string calcular(const std::string& operacao, double a, double b) {
    // Implementação das 4 operações
}

// 3. Roteamento de requisições
void handle_request(http::request<http::string_body>& req,
                    http::response<http::string_body>& res) {
    // Análise da URL e direcionamento
}

// 4. Loop principal do servidor
int main() {
    // Inicialização, bind, listen, accept loop
}
```

Principais Características da Implementação:

1. Conexões Assíncronas: Usa Boost.Asio para I/O não-bloqueante
2. Thread-Safe: Processa múltiplas conexões sequencialmente
3. Parsing de URLs: Extrai parâmetros do caminho da requisição
4. Headers CORS: Permite acesso cross-origin para a interface web
5. Tratamento de Erros: Exceções para entradas inválidas

INTERFACE WEB - DETALHES DE IMPLEMENTAÇÃO

```
<div class="container">
    <!-- Inputs para números -->
    <input type="number" id="numero1">
    <input type="number" id="numero2">

    <!-- Botões de operações -->
    <button onclick="calcular('somar')">+ Somar</button>

    <!-- Área de resultado -->
    <div id="resultado"></div>
</div>
```

CSS - Design Responsivo

1. Layout flexbox para adaptação a diferentes telas
2. Cores diferenciadas para cada operação
3. Animações e transições suaves
4. Design moderno com gradientes e sombras

JavaScript - Comunicação com API

```
async function calcular(operacao) {
    // 1. Validação de entrada
    // 2. Requisição Fetch para o servidor
    // 3. Atualização da interface com resultado
    // 4. Tratamento de erros
}
```

Execução do Servidor:

./calculadora_server

CONCEITOS DE SISTEMAS DISTRIBUÍDOS APLICADOS

1. Arquitetura Cliente-Servidor

- Separação clara de responsabilidades
- Comunicação via protocolo padronizado (HTTP)
- Servidor stateless (cada requisição independente)

2. Comunicação via Protocolos de Rede

- HTTP/1.1 como protocolo de aplicação
- TCP/IP como protocolo de transporte/rede
- Porta 8080 como endpoint de serviço

3. Interface de Serviço Bem Definida

- Endpoints RESTful
- Métodos HTTP semânticos (GET)
- Parâmetros no caminho da URL
- Respostas padronizadas

4. Concorrência e Escalabilidade

- Servidor single-threaded mas assíncrono
- Possibilidade de expansão para thread pool
- Modelo de I/O não-bloqueante

5. Interoperabilidade

- Interface web (qualquer navegador)
- API consumível por qualquer cliente HTTP
- Formato de dados independente de plataforma