TriProtocol: Implementação e Análise Comparativa de Clientes Multi-Protocolo

Disciplina: Sistemas Distribuídos

Trabalho Prático: Análise de Protocolos de Comunicação em Ambientes Distribuídos

1. VISÃO GERAL

1.1 Propósito do Trabalho

Este trabalho prático é um **estudo comparativo hands-on** dos principais paradigmas de comunicação em sistemas distribuídos modernos. Os alunos implementarão clientes para três protocolos distintos, conectando-se a servidores remotos na nuvem AWS para executar operações reais e analisar características de **desempenho**, **eficiência**, **complexidade de implementação** e **facilidade de debugging**.

1.2 Objetivos de Aprendizagem

Ao concluir este trabalho, os alunos serão capazes de:

- **Implementar clientes** para diferentes protocolos de comunicação
- **Comparar protocolos** baseados em critérios objetivos e subjetivos
- **Compreender trade-offs** entre simplicidade, eficiência e flexibilidade
- **Analisar características** de serialização, parsing e overhead de rede
- **Aplicar boas práticas** de programação para sistemas distribuídos
- **Documentar e apresentar** resultados técnicos de forma profissional

1.3 Contexto Tecnológico

O trabalho simula um **ambiente real de desenvolvimento** onde diferentes sistemas precisam se comunicar usando protocolos diversos. Esta é uma situação comum em arquiteturas de microsserviços, APIs empresariais e sistemas distribuídos em produção.

1.4 Protocolos Abordados

Os servidores estão implementados e serão executados pelo professor em **instâncias EC2 da Amazon Web Services** com IPs públicos. Os alunos devem implementar os clientes para os seguintes protocolos:

1.4.1 Protocolo de Strings (Porta 8080)

- **Paradigma**: Comunicação baseada em texto estruturado
- **Formato**: `COMANDO|param=valor|FIM`
- **Vantagens**: Simplicidade extrema, fácil debugging
- **Casos de uso**: Sistemas legados, protocolos proprietários simples

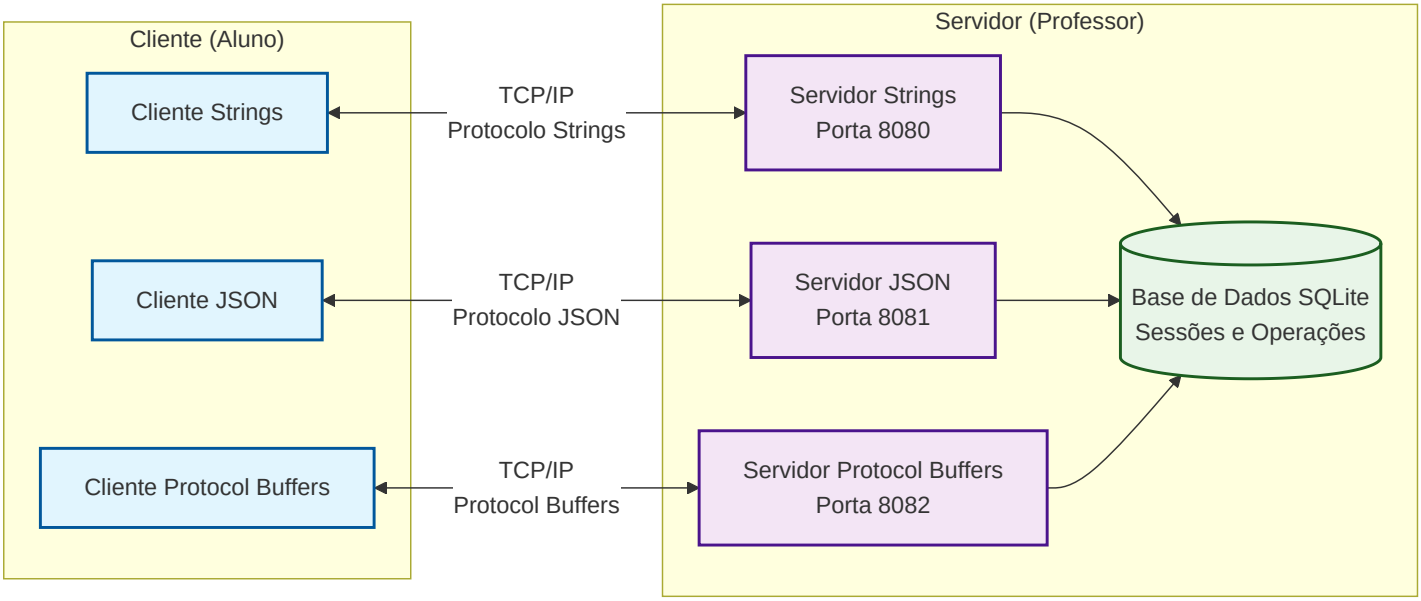
1.4.2 Protocolo JSON (Porta 8081)

- **Paradigma**: Comunicação estruturada legível
- **Formato**: Objetos JSON com schema definido
- **Vantagens**: Flexibilidade, compatibilidade universal, facilidade de debugging
- **Casos de uso**: APIs REST, aplicações web, microsserviços

1.4.3 Protocolo Protocol Buffers (Porta 8082)

- **Paradigma**: Comunicação binária eficiente
- **Formato**: Mensagens binárias serializadas com schema tipado
- **Vantagens**: Alta performance, economia de bandwidth, tipagem forte
- **Casos de uso**: Sistemas de alta performance, gRPC, comunicação interna

1.5 Arquitetura do Sistema



1.6 Metodologia de Trabalho

1. **Implementação Iterativa**: Começar com o protocolo mais simples (Strings) e progredir
2. **Testes Incrementais**: Validar cada operação antes de avançar
3. **Análise Comparativa**: Documentar diferenças observadas durante o desenvolvimento
4. **Medição de Performance**: Coletar métricas básicas de tempo e tamanho de mensagens
5. **Documentação Contínua**: Registrar decisões técnicas e desafios encontrados

2. SISTEMA DE AUTENTICAÇÃO E ACESSO

2.1 Funcionamento da Autenticação

O sistema implementa um **mecanismo de autenticação baseado em matrícula** seguido por **autorização via token de sessão**. Este processo garante que apenas alunos cadastrados possam acessar os serviços e que todas as operações sejam rastreadas adequadamente.

2.2 Processo de Autenticação

Passo 1: Autenticação Inicial

- **Chave de Acesso**: O número de matrícula do aluno é a **única credencial** necessária para autenticação
- **Validação**: O servidor consulta sua base de dados para verificar se a matrícula é válida e está cadastrada
- **Segurança**: Cada matrícula está associada aos dados completos do aluno (nome, turma, etc.)

Passo 2: Geração do Token de Sessão

Após a autenticação bem-sucedida:

- O servidor **gera automaticamente um token único** para a sessão
- Este token tem **validade limitada** (1 hora por sessão)
- O token é **vinculado especificamente** à matrícula do aluno autenticado
- **Todas as operações subsequentes** requerem este token

Passo 3: Autorização de Operações

- **Token Obrigatório**: Toda chamada para os métodos de negócio (echo, soma, timestamp, status, histórico) **deve incluir o token**
- **Validação Contínua**: O servidor verifica a validade do token a cada operação
- **Rastreabilidade**: Todas as operações são registradas na base de dados associadas à matrícula do aluno

2.3 Fluxo Completo de Comunicação

```
1. Cliente conecta ao servidor via TCP
2. Cliente envia comando AUTH com sua matrícula
3. Servidor valida matrícula na base de dados
4. Servidor retorna token de sessão + dados do aluno
5. Cliente usa o token em todas as operações subsequentes
6. Cliente executa operações de negócio (OP commands)
7. Cliente encerra sessão com LOGOUT
8. Conexão TCP é fechada
```

2.4 Importantes Considerações de Implementação

Gerenciamento de Token

- **Armazenar o token**: Após receber o token na autenticação, seu cliente deve armazená-lo para uso em todas as operações
- **Incluir em todas as operações**: O token deve ser enviado como parâmetro em **todas** as chamadas aos métodos de negócio
- **Tratar expiração**: Tokens expiram após 1 hora de inatividade - implemente tratamento para re-autenticação quando necessário

Tratamento de Erros

- **Matrícula inválida**: O servidor retornará erro se a matrícula não estiver cadastrada
- **Token inválido/expirado**: Operações falharão se o token não for válido - reautenticar quando necessário
- **Sessão duplicada**: O sistema permite apenas uma sessão ativa por matrícula

Boas Práticas

- **Sempre fazer logout**: Encerre adequadamente a sessão para liberar recursos do servidor

- **Validar respostas**: Sempre verifique se a autenticação foi bem-sucedida antes de prosseguir
- **Implementar timeouts**: Configure timeouts adequados para conexões TCP

3. ESPECIFICAÇÕES TÉCNICAS

3.1 Protocolo Strings (Porta 8080)

Formato de Comunicação:

- Mensagens em texto plano
- Separadores: pipe (|) entre campos
- Codificação UTF-8
- Terminador: `\n`

Estrutura Geral:

```
COMANDO|param1=valor1|param2=valor2|FIM
```

Comandos Específicos:

Autenticação (AUTH):

```
AUTH|aluno_id=valor|timestamp=valor|FIM
```

Operações (OP):

```
OP|token=valor|operacao=nome_operacao|param1=valor1|param2=valor2|FIM
```

Logout (LOGOUT):

```
LOGOUT|token=valor|FIM
```

Exemplos:

```
AUTH|aluno_id=785445|timestamp=2025-10-10T10:00:00|FIM
OP|token=token123|operacao=echo|mensagem=Hello World|FIM
LOGOUT|token=token123|FIM
```

3.2 Protocolo JSON (Porta 8081)

Formato de Comunicação:

- Mensagens em formato JSON válido
- Codificação UTF-8
- Cada mensagem termina com `\n`

Estrutura de Requisição:

```
{
    "tipo": "autenticar|operacao|logout",
    "aluno_id": "785445",
    "token": "token_se_necessario",
    "operacao": "nome_da_operacao",
    "parametros": {
        // parâmetros específicos
    },
    "timestamp": "ISO 8601"
}
```

Estrutura de Resposta:

```
{
    "sucesso": true,
    "resultado": {
        // dados de resposta
    },
    "mensagem": "mensagem_do_servidor",
    "timestamp": "ISO 8601"
}
```

3.3 Protocolo Protocol Buffers (Porta 8082)

Formato de Comunicação:

- Mensagens binárias serializadas
- Cabeçalho: 4 bytes indicando tamanho da mensagem
- Payload: dados Protocol Buffers serializados

Schema (arquivo .proto fornecido):

```
syntax = "proto3";

message Requisicao {
    oneof tipo {
        Auth auth = 1;
        Operacao operacao = 2;
        Logout logout = 3;
    }
}

message Resposta {
    oneof resultado {
        RespostaOK ok = 1;
        RespostaErro erro = 2;
    }
}
```

4. FLUXO DE COMUNICAÇÃO E DIAGRAMAS

4.1 Fluxo Básico de Comunicação

1. Estabelecer Conexão TCP

```
Cliente → Servidor: TCP Connect
Servidor → Cliente: Connection Established
```

2. Autenticação

```
Cliente → Servidor: AUTH command
Servidor → Cliente: Token de sessão
```

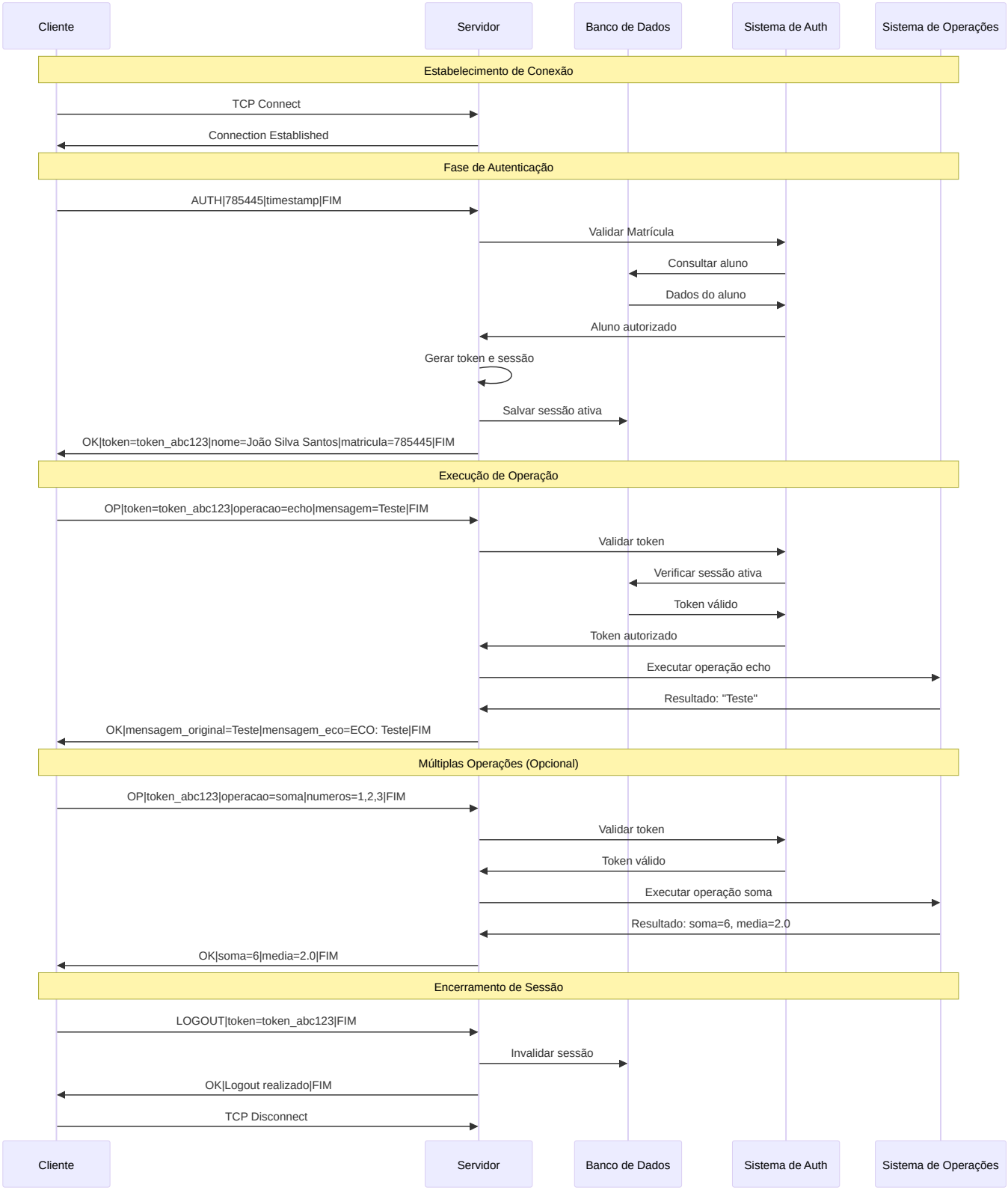
3. Operações

```
Cliente → Servidor: OP command + token
Servidor → Cliente: Resultado da operação
```

4. Encerrar Sessão

```
Cliente → Servidor: LOGOUT command
Servidor → Cliente: Logout confirmation
Cliente → Servidor: TCP Disconnect
```

4.2 Diagrama de Sequência - Fluxo Completo



5. EXEMPLOS PRÁTICOS DE MENSAGENS

5.1 Protocolo de Strings (Porta 8080)

Autenticação

Requisição:

```
AUTH|aluno_id=785445|timestamp=2025-10-10T14:30:00|FIM
```

Resposta de Sucesso:

```
OK|token=token_abc123def456|nome=João Silva Santos|matricula=785445|FIM
```

Resposta de Erro:

```
ERROR|msg=Aluno não autorizado|timestamp=2025-10-10T14:30:01|FIM
```

Operação Echo com Token

Requisição:

```
OP|token=token_abc123def456|operacao=echo|mensagem=Olá Servidor|FIM
```

Resposta:

```
OK|mensagem_original=Olá Servidor|mensagem_eco=ECO: Olá Servidor|hash_md5=d6e2c3f876543210abcdef1234567890|FI
```

5.2 Protocolo JSON (Porta 8081)

Autenticação

Requisição:

```
{
    "tipo": "autenticar",
    "aluno_id": "785445",
    "timestamp": "2025-10-10T14:30:00"
}
```

Resposta de Sucesso:

```
{
    "sucesso": true,
    "token": "token_abc123def456",
    "dados_aluno": {
        "nome": "João Silva Santos",
        "matricula": "785445"
    },
    "mensagem": "Autenticação realizada com sucesso",
    "timestamp": "2025-10-10T14:30:01"
}
```

Resposta de Erro:

```
{
    "sucesso": false,
    "erro": "Aluno não autorizado",
    "timestamp": "2025-10-10T14:30:01"
}
```

Operação Echo com Token

Requisição:

```
{
    "tipo": "operacao",
    "token": "token_abc123def456",
    "operacao": "echo",
    "parametros": {
        "mensagem": "Olá Servidor"
    },
    "timestamp": "2025-10-10T14:30:05"
}
```

Resposta:

```
{
    "sucesso": true,
    "resultado": {
        "mensagem_original": "Olá Servidor",
        "mensagem_eco": "ECO: Olá Servidor",
        "timestamp_servidor": "2025-10-10T14:30:06",
        "tamanho_mensagem": 11,
        "hash_md5": "d6e2c3f876543210abcdef1234567890"
    },
    "timestamp": "2025-10-10T14:30:06"
}
```

5.3 Protocol Buffers (Porta 8082)

Autenticação

Estrutura da Mensagem (Formato Binário):

```
[4 bytes: tamanho] + [dados protobuf serializados]
```

Requisição (representação textual):

```
requisicao {
  auth {
    aluno_id: "785445"
    timestamp: "2025-10-10T14:30:00"
  }
}
```

Resposta de Sucesso (representação textual):

```
resposta {
  auth_response {
    sucesso: true
    token: "token_abc123def456"
    nome: "João Silva Santos"
    matricula: "785445"
    sessao_id: 12345
    timestamp: "2025-10-10T14:30:01"
  }
}
```

Resposta de Erro:

```
resposta {
  erro {
    codigo: "AUTH_FAILED"
    mensagem: "Aluno não autorizado"
    timestamp: "2025-10-10T14:30:01"
  }
}
```

Operação Echo com Token

Requisição:

```
requisicao {
  operacao {
    token: "token_abc123def456"
    nome_operacao: "echo"
    parametros {
      key: "mensagem"
      value: "Olá Servidor"
    }
    timestamp: "2025-10-10T14:30:05"
  }
}
```

Resposta:

```
resposta {
  operacao_response {
    sucesso: true
    resultado {
      key: "mensagem_original"
      value: "Olá Servidor"
    }
    resultado {
      key: "resposta"
      value: "Eco processado com sucesso"
    }
    timestamp: "2025-10-10T14:30:06"
  }
}
```

6. OPERAÇÕES DO SISTEMA

6.1 Autenticação (AUTH)

Parâmetros:

- `aluno_id` : Matrícula do aluno (fornecida pelo professor)

Resposta:

- `token` : Token de sessão para operações subsequentes
- `nome` : Nome completo do aluno
- `matricula` : Número de matrícula

6.2 Operações Disponíveis (OP)

6.2.1 Echo

- **Descrição:** Retorna a mensagem enviada com informações adicionais
- **Parâmetros:**
  - `mensagem` : Texto a ser ecoado (string)
- **Retorno:**
  - `mensagem_original` : Mensagem enviada pelo cliente
  - `mensagem_eco` : Mensagem prefixada com "ECO: "
  - `timestamp_servidor` : Momento do processamento
  - `tamanho_mensagem` : Tamanho em caracteres
  - `hash_md5` : Hash MD5 da mensagem original
- **Exemplo de Uso:** Testar conectividade e integridade dos dados

6.2.2 Soma

- **Descrição:** Calcula estatísticas matemáticas de uma lista de números
- **Parâmetros:**
  - `numeros` : Lista de números (formato varia por protocolo)
- **Retorno:**
  - `numeros_processados` : Lista dos números recebidos
  - `soma` : Soma de todos os números
  - `media` : Média aritmética
  - `maximo` : Maior número
  - `minimo` : Menor número
  - `quantidade` : Total de números processados
- **Validações:**
  - Lista não pode estar vazia
  - Todos os elementos devem ser números válidos

6.2.3 Timestamp

- **Descrição:** Retorna informações temporais do servidor
- **Parâmetros:** Nenhum
- **Retorno:**
  - `timestamp_formatado` : Data/hora legível
  - `timestamp_unix` : Timestamp Unix
  - `timezone` : Fuso horário do servidor

- `dia_semana` : Dia da semana
- `informacoes_adicionais` : Outros dados temporais

6.2.4 Status

- **Descrição:** Informações sobre o estado do servidor
- **Parâmetros Opcionais:**
  - `detalhado` : Boolean para informações extras
- **Retorno Básico:**
  - `status` : Estado geral do servidor
  - `operacoes_processadas` : Contador total
  - `tempo_ativo` : Tempo desde inicialização
- **Retorno Detalhado** (se solicitado):
  - `sessoes_ativas` : Número de sessões ativas
  - `estatisticas_banco` : Dados do banco de dados
  - `memoria_uso` : Informações de memória
  - `conexoes_recentes` : Histórico de conexões

6.2.5 Histórico

- **Descrição:** Histórico de operações do aluno logado
- **Parâmetros Opcionais:**
  - `limite` : Número máximo de registros (padrão: 10)
- **Retorno:**
  - `operacoes` : Lista de operações anteriores
  - `total_encontrado` : Total de operações no histórico
  - `estatisticas` : Resumo estatístico das operações
- **Estrutura de Cada Operação:**
  - `id` : ID único da operação
  - `operacao` : Nome da operação executada
  - `timestamp` : Momento da execução
  - `sucesso` : Se foi executada com sucesso
  - `parametros` : Parâmetros utilizados
  - `resultado` : Resultado obtido

6.3 Logout (LOGOUT)

Parâmetros Obrigatórios:

- `token` : Token da sessão a ser encerrada

Retorno:

- `mensagem` : Confirmação do logout
- `sessao_encerrada` : ID da sessão encerrada

Comportamento:

- Token torna-se inválido imediatamente
- Operações subsequentes com o token falharão
- Recursos da sessão são liberados

7. TRATAMENTO DE ERROS E TIMEOUTS

7.1 Tipos de Erro

1. **Erro de Autenticação**
   - Causa: Aluno não autorizado
   - Solução: Verificar ID do aluno - utilizar seu **número de matrícula**
2. **Erro de Token**
   - Causa: Token inválido, expirado ou não fornecido
   - Solução: Realizar nova autenticação
3. **Erro de Operação**
   - Causa: Operação não suportada
   - Solução: Verificar lista de operações disponíveis
4. **Erro de Parâmetros**
   - Causa: Parâmetros obrigatórios ausentes ou inválidos
   - Solução: Verificar especificação da operação
5. **Erro Interno**
   - Causa: Problema no servidor
   - Solução: Tentar novamente ou contatar suporte

7.2 Formato de Erro por Protocolo

JSON:

```
{
    "sucesso": false,
    "erro": "Token inválido ou expirado",
    "detalhes": {
        "codigo": "ERRO_TOKEN"
    },
    "timestamp": "2025-10-10T10:00:00.000Z"
}
```

Strings:

```
ERROR|msg=Token inválido ou expirado|codigo=ERRO_TOKEN|FIM
```

Protocol Buffers: Utiliza a mensagem `RespostaErro` do schema.

7.3 Timeouts e Limites

Timeouts

- **Conexão**: 30 segundos
- **Sessão**: 1 hora de inatividade
- **Operação**: 10 segundos por operação

Limites

- **Mensagem máxima**: 64KB
- **Números na soma**: 1000 elementos
- **Histórico**: 100 registros por consulta

- **Conexões simultâneas**: 50 por servidor

8. COMPARAÇÃO ENTRE PROTOCOLOS

8.1 Eficiência

- **Protocol Buffers**: Mais eficiente (50-80% menor)
- **JSON**: Moderadamente eficiente, legível
- **Strings**: Menos eficiente, muito simples

8.2 Complexidade de Implementação

- **Strings**: Mais simples
- **JSON**: Moderada (parsing necessário)
- **Protocol Buffers**: Mais complexa (requer compilador)

8.3 Debugging

- **Strings**: Mais fácil (texto plano)
- **JSON**: Fácil (estruturado e legível)
- **Protocol Buffers**: Mais difícil (binário)

8.4 Tipagem

- **Protocol Buffers**: Tipagem forte
- **JSON**: Tipagem fraca mas estruturada
- **Strings**: Sem tipagem (tudo é string)

8.5 Compatibilidade

- **JSON**: Universalmente suportado
- **Strings**: Suportado em qualquer linguagem
- **Protocol Buffers**: Requer biblioteca específica

9. BOAS PRÁTICAS PARA IMPLEMENTAÇÃO

9.1 Gestão de Conexões

1. Usar timeout adequado nas conexões
2. Fechar conexões adequadamente
3. Reusar conexões quando possível

9.2 Tratamento de Dados

1. Validar todos os dados recebidos
2. Implementar encoding/decoding correto
3. Gerenciar buffers adequadamente
4. Tratar casos edge (dados vazios, muito grandes)

9.3 Monitoramento

1. Implementar logs de debug
2. Medir tempos de resposta
3. Contar sucessos e falhas
4. Implementar métricas de desempenho

9.4 Testes

1. Testar todos os cenários de sucesso
2. Testar todos os cenários de erro
3. Testar cenários de rede (desconexão, timeout)
4. Validar comportamento com dados inválidos

10. REQUISITOS DE IMPLEMENTAÇÃO

10.1 Requisitos Obrigatórios

1. **Três Clientes Funcionais**: Implementar cliente para cada protocolo
2. **Autenticação Completa**: Sistema de login/logout funcional
3. **Todas as Operações**: Implementar todas as 5 operações listadas
4. **Tratamento de Erros**: Gerenciamento adequado de erros de rede e protocolo
5. **Interface Consistente**: Interface similar entre os três clientes

10.2 Funcionalidades Mínimas

- Conexão com servidor remoto
- Autenticação com credenciais válidas
- Execução de todas as operações
- Exibição clara dos resultados
- Logout e encerramento de sessão
- Tratamento de timeouts e desconexões

10.3 Linguagens Suportadas

Os alunos podem implementar os clientes em qualquer linguagem que suporte:

- Sockets TCP/IP
- JSON parsing (para protocolo JSON)
- Protocol Buffers (para protocolo binário)

Sugestões: Python, Java, C++, JavaScript (Node.js), Go, C#

11. ENTREGÁVEIS

11.1 Repositório GitHub

Código Fonte Obrigatório:

- Três implementações de cliente (uma para cada protocolo)
- Documentação de instalação e execução (README.md)
- Scripts de exemplo e/ou interface gráfica
- Código organizado em estrutura de pastas clara
- Arquivo de dependências (requirements.txt, pom.xml, etc.)

Estrutura Sugerida do Repositório:

```
nome-do-projeto/
├── README.md
├── cliente-strings/
│   ├── cliente_strings.py (ou .java, .cpp, etc.)
│   └── documentacao.md
├── cliente-json/
│   ├── cliente_json.py
│   └── documentacao.md
├── cliente-protobuf/
│   ├── cliente_protobuf.py
│   ├── schema.proto (se necessário)
│   └── documentacao.md
├── docs/
│   └── relatorio_tecnico.md
└── assets/
    └── screenshots/
```

Requisitos do GitHub:

- Repositório público ou privado (com acesso para o professor)
- Commits com mensagens descritivas
- README.md principal com instruções de instalação e execução
- Documentação técnica adequada

11.2 Vídeo Demonstrativo

Especificações do Vídeo:

- **Duração:** 10-15 minutos
- **Hospedagem:** YouTube (pode ser não listado) ou similar

Conteúdo Obrigatório do Vídeo:

Parte 1: Apresentação

- Identificação do aluno (nome, matrícula, turma)
- Visão geral do projeto e objetivos
- Arquitetura implementada

Parte 2: Demonstração Prática

- **Execução dos três clientes** conectando aos servidores
- **Demonstração das operações:** echo, soma, timestamp, status, histórico
- **Fluxo completo:** autenticação → operações → logout
- **Tratamento de erros** (pelo menos um cenário)

Parte 3: Explicação do Código

- **Trechos relevantes do código** de cada protocolo
- **Decisões de implementação** e desafios enfrentados
- **Comparação entre protocolos** com base na experiência prática
- **Análise de desempenho** (se implementada)

Orientações para Gravação:

- Use ferramentas como OBS Studio, Camtasia, ou similar
- Prepare um roteiro para evitar pausas longas
- Teste áudio e vídeo antes da gravação final
- Mostre o código funcionando em tempo real
- Explique as diferenças práticas entre os protocolos

11.3 Entrega

Formato de Entrega:

- **GitHub:** Link do repositório público ou convite de acesso
- **Vídeo:** Link do YouTube (pode ser não listado) ou plataforma similar
- **Submissão:** Via sistema da disciplina com ambos os links

Prazo:

- Data limite: [A ser definida pelo professor]

12. INFORMAÇÕES COMPLEMENTARES

12.1 Endereços dos Servidores

Importante: Os endereços IP serão fornecidos pelo professor no início do trabalho, pois as instâncias EC2 serão criadas dinamicamente.

Formato:

- Servidor Strings: `tcp://[IP_PUBLICO]:8080`
- Servidor JSON: `tcp://[IP_PUBLICO]:8081`
- Servidor Protocol Buffers: `tcp://[IP_PUBLICO]:8082`

12.2 Arquivos de Apoio

O professor fornecerá:

- Schema Protocol Buffers (`.proto`)
- Documentação detalhada da API
- Exemplos de mensagens para cada protocolo
- Bibliotecas ou dependências necessárias

12.3 Prazo de Entrega

Data de Entrega: [A ser definida pelo professor]
Formato: Repositório Git com código e documentação

Apresentação: [Data da apresentação a ser definida]

13. RECURSOS ADICIONAIS

13.1 Bibliografia Recomendada

- Coulouris, G. et al. "Distributed Systems: Concepts and Design"
- Tanenbaum, A. S. "Distributed Systems: Principles and Paradigms"
- Documentação oficial do Protocol Buffers

13.2 Ferramentas Sugeridas

- Wireshark (análise de tráfego de rede)
- Postman ou similar (testes de API)
- Git (controle de versão)
- IDE/Editor de código de preferência

13.3 Suporte

- Fórum da disciplina para dúvidas técnicas
- Horário de atendimento do professor
- Documentação técnica disponível no repositório da disciplina

Bom trabalho! Este projeto proporcionará uma oportunidade de aprender sobre diferentes abordagens de comunicação em sistemas distribuídos.