Relatório Técnico – SCTEC

Sistema de Controle de Telescópio Espacial Compartilhado

Gustavo Nascimento Siqueira	RA: 10419057
Felipe Ujvari Gasparino de Sousa	RA: 10418415
Thomaz de Souza Scopel	RA: 10417183

16 de novembro de 2025

Resumo

Este documento descreve a implementação completa do **SCTEC** (Sistema de Controle de Telescópio Espacial Compartilhado), uma aplicação distribuída composta por um serviço de agendamento (Flask/Python) e um serviço coordenador (Node.js/Express) para garantir exclusão mútua em operações críticas de agendamento. O relatório consolida a arquitetura, modelagem de dados, especificação da API RESTful com HATEOAS, estratégia de logging (aplicação e auditoria), sincronização de tempo via Algoritmo de Cristian, containerização com Docker Compose e a validação por meio de testes de concorrência (demonstração do problema *sem lock* e da solução *com lock*). Incluímos seções com diagramas TikZ detalhados e espaço para evidências (prints de tela, saídas de logs e resultados).

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

O SCTEC foi desenvolvido no contexto da disciplina de *Computação Distribuída*, com os seguintes objetivos principais:

- Implementar uma **API RESTful** com **HATEOAS**;
- Resolver **condições de corrida** via **exclusão mútua** (serviço coordenador);
- Sincronizar o **tempo cliente-servidor** (Algoritmo de Cristian);
- Prover **observabilidade** com logs de aplicação e **auditoria** (correlation ID);
- Containerizar e orquestrar com **Docker Compose**.

Este relatório consolida o conteúdo dos documentos MODELOS.md, API.md, LOGGING.md, DOCKER.md e do README do projeto, estruturando a narrativa técnica e incluindo diagramas detalhados.

2 Arquitetura do Sistema

O sistema é composto por dois microserviços e um banco de dados SQLite. A comunicação se dá via HTTP/JSON, e o controle de concorrência é centralizado no coordenador.

2.1 Visão Geral

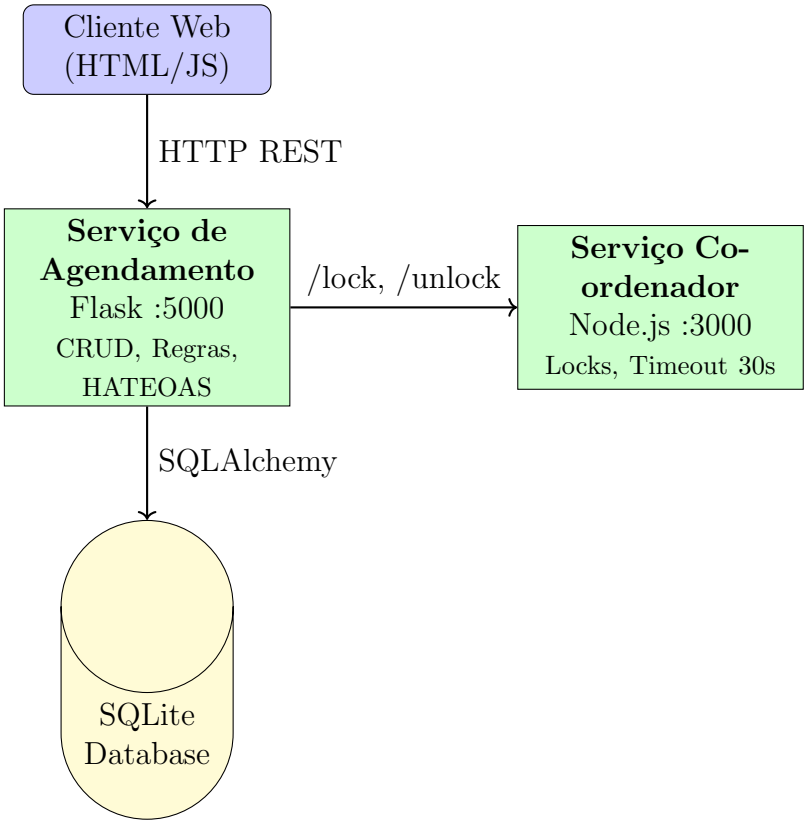

Figura 1: Arquitetura geral do sistema SCTEC

2.2 Divisão de Responsabilidades

Serviço de Agendamento (Flask):

- Fonte de verdade do estado do sistema
- Validação de regras de negócio
- Persistência em SQLite
- Exposição da API RESTful com HATEOAS
- Logging estruturado (aplicação + auditoria)

Serviço Coordenador (Node.js):

- Controle de acesso com *locks* por recurso
- Garantia de exclusão mútua
- Gerenciamento de timeouts (30s)
- Limpeza automática de locks expirados

3 Modelagem de Dados

3.1 Entidades Principais

Cientista:

- **Atributos:** id, nome, email (único), instituição, país, especialidade, data_cadastro, ativo
- **Papel:** Representa pesquisadores autorizados a usar o telescópio

Agendamento:

- **Atributos:** id, cientista_id, horário_início/fim (UTC), objeto_celeste, observações, status (AGENDADO, CANCELADO, CONCLUIDO), datas de criação/cancelamento, motivo
- **Papel:** Representa reservas de tempo no telescópio

3.2 Diagrama Entidade-Relacionamento

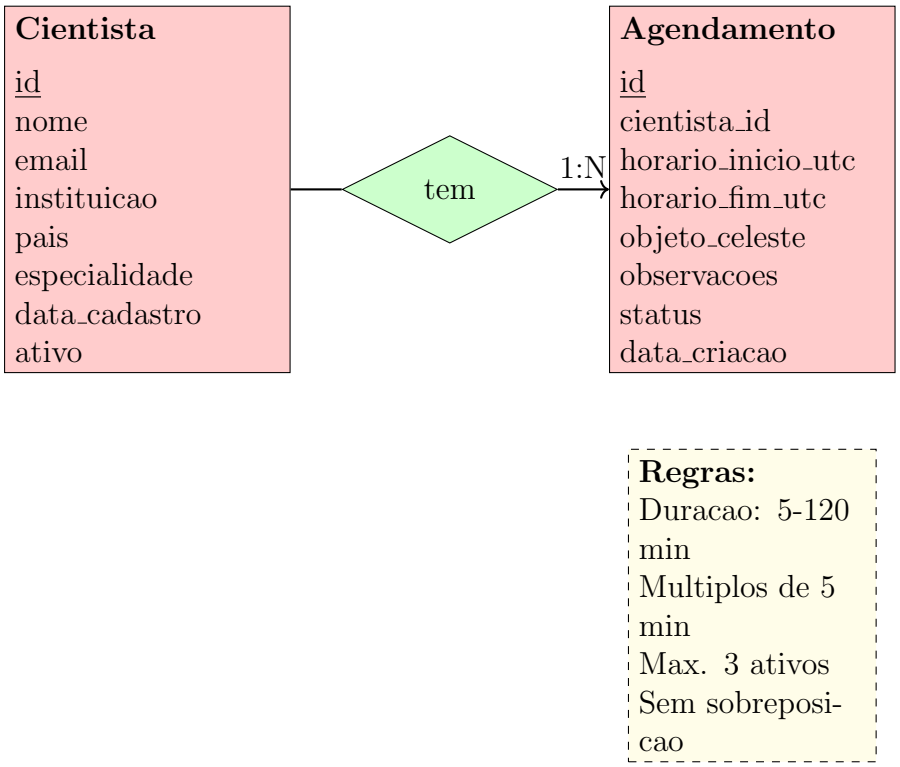

Figura 2: Diagrama Entidade-Relacionamento

3.3 Regras de Negócio

- Duração mínima de 5 min e máxima de 120 min
- Slots múltiplos de 5 minutos
- Não agendar no passado; antecedência mínima de 24h
- Sem sobreposição de horários com status AGENDADO
- Até 3 agendamentos ativos por cientista
- Somente agendamentos AGENDADOS podem ser cancelados

3.4 Índices e Constraints

- **Índices principais:** (horário_início, horário_fim, status), cientista_id, status
- **Constraints:** `horario_fim > horario_inicio`, domínio de status válido
- **Unique constraint:** Combinação de horários para status AGENDADO

4 API RESTful com HATEOAS

4.1 Convenções da API

- **Base URL:** `/api/v1`
- **Formato:** JSON
- **Timezone:** UTC para todos os timestamps
- **Headers obrigatórios:** `Content-Type: application/json`
- **Header de resposta:** `X-Correlation-ID`
- **Códigos HTTP:** 200 (OK), 201 (Created), 400 (Bad Request), 404 (Not Found), 409 (Conflict), 422 (Unprocessable Entity), 500 (Internal Server Error)

4.2 Endpoint de Sincronização de Tempo

O endpoint /time retorna o timestamp oficial do servidor para sincronização de tempo (Algoritmo de Cristian).

Resposta exemplo:

```
{
  "timestamp_utc": "2025-11-16T15:30:45.123456Z",
  "timezone": "UTC",
  "epoch_ms": 1731166245123,
  "_links": {
    "self": { "href": "/api/v1/time" },
    "agendamentos": { "href": "/api/v1/agendamentos" }
  }
}
```

Listing 1: Resposta do endpoint /time

4.3 Endpoints de Cientistas

- **GET** /cientistas: Lista com paginação, filtros e links HATEOAS
- **POST** /cientistas: Criação com validações (nome, email único)
- **GET** /cientistas/{id}: Detalhes de um cientista específico
- **GET** /cientistas/{id}/agendamentos: Agendamentos de um cientista

4.4 Endpoints de Agendamentos

- **GET** /agendamentos: Lista com filtros (período, status, cientista), paginação
- **POST** /agendamentos: Operação crítica com lock; validações completas
- **GET** /agendamentos/{id}: Detalhes com HATEOAS condicional
- **DELETE** /agendamentos/{id}: Soft delete com registro de motivo

5 Logging e Observabilidade

5.1 Logs de Aplicação

Formato: `[LEVEL] timestamp service correlation_id: mensagem`

Níveis de log:

- **DEBUG:** Detalhes técnicos para desenvolvimento
- **INFO:** Fluxo normal da aplicação
- **WARNING:** Situações incomuns mas tratadas
- **ERROR:** Erros que impedem operações
- **CRITICAL:** Falhas que comprometem o sistema

Pontos de log principais:

- Recebimento de requisição
- Tentativa/obtenção de lock
- Verificação de conflitos
- Persistência no banco
- Envio de resposta

5.2 Logs de Auditoria (JSON)

Eventos principais em formato JSON estruturado:

- `CIENTISTA_CRIADO`
- `AGENDAMENTO_CRIADO`
- `AGENDAMENTO_CANCELADO`
- `AGENDAMENTO_CONFLITO`

Cada evento registra: `timestamp_utc`, `event_type`, `service`, `correlation_id` e `details`.

5.3 Correlation ID

- UUID gerado por requisição
- Propagado entre todos os serviços
- Incluído em todos os logs
- Retornado no header `X-Correlation-ID`
- Permite *traceability* ponta-a-ponta

6 Serviço Coordenador (Locks)

6.1 API do Coordenador

- **POST** /lock: Tenta adquirir lock de um recurso (200/409)
- **POST** /unlock: Libera lock (200/404)
- **GET** /locks: Debug e observabilidade de locks ativos
- **GET** /health: Health check do serviço

6.2 Características dos Locks

- **Timeout:** 30 segundos (prevenção de deadlock)
- **Cleanup:** Periódico (a cada 1 minuto)
- **Identificação:** Por nome de recurso (ex: `Hubble-Acad_2025-12-01T03:00:00Z`)
- **Holder tracking:** Registra correlation_id do detentor

6.3 Fluxo de Agendamento com Sucesso

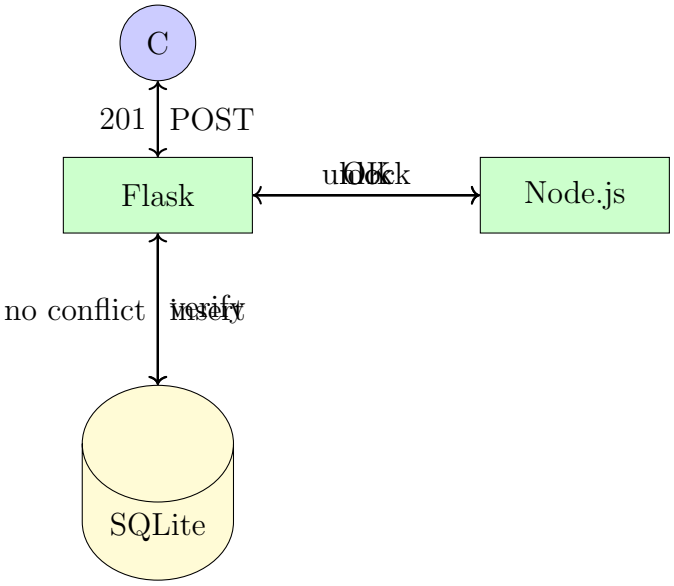

Figura 3: Fluxo simplificado de agendamento com lock

6.4 Fluxo Detalhado com Múltiplas Requisições

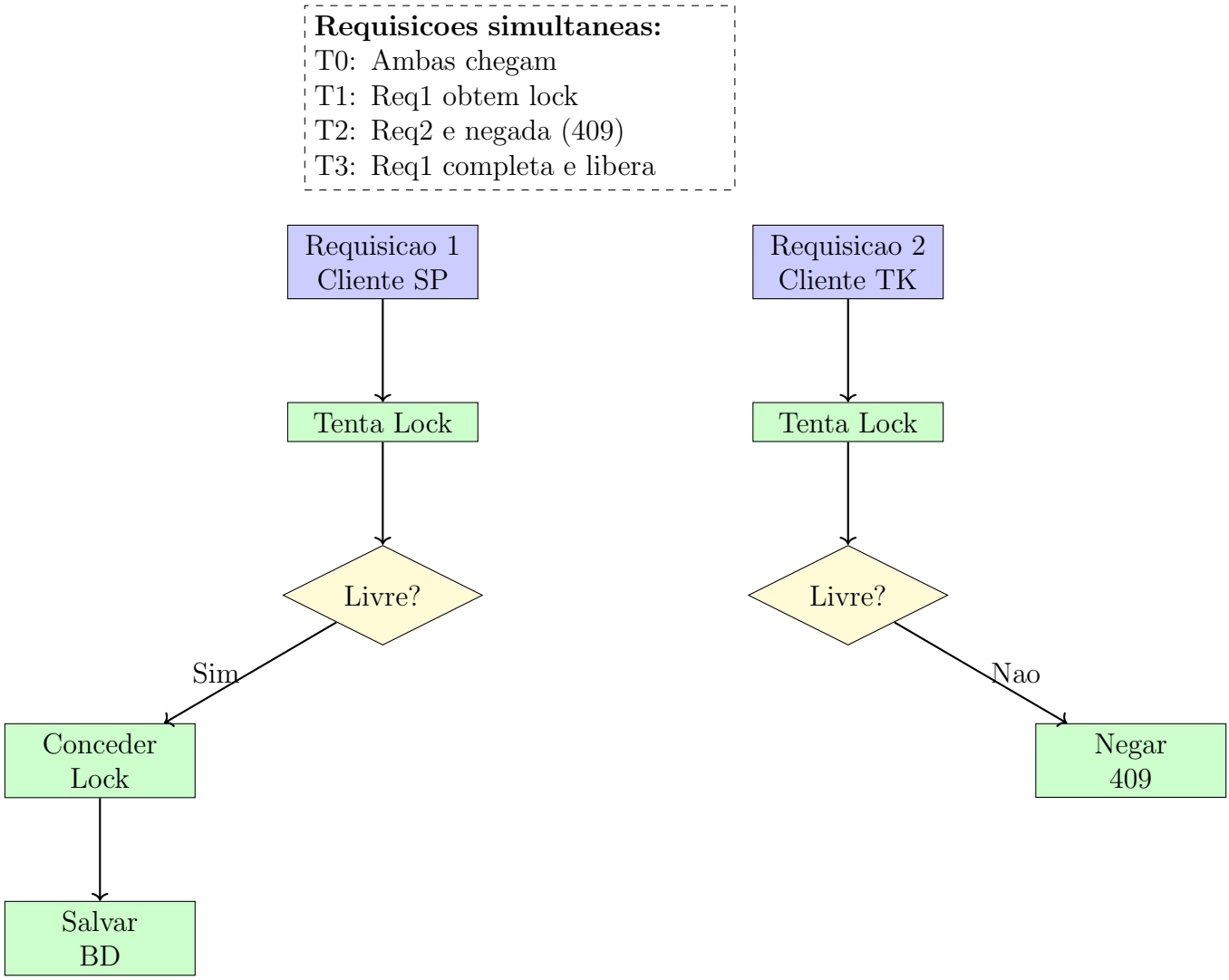

Figura 4: Exclusão mútua - Duas requisições concorrentes

7 Sincronização de Tempo (Algoritmo de Cristian)

O cliente realiza medições de *round-trip time* (RTT) com o endpoint /time e calcula um *offset* assumindo latência simétrica.

7.1 Equação do Offset

$$\text{offset} = (T_{\text{servidor}} + \tfrac{RTT}{2}) - T_{\text{cliente}} \tag{1}$$

Onde:

- T_{servidor}: Timestamp retornado pelo servidor
- RTT: Round-Trip Time (tempo total da requisição)
- T_{cliente}: Timestamp local do cliente

7.2 Fluxograma do Algoritmo

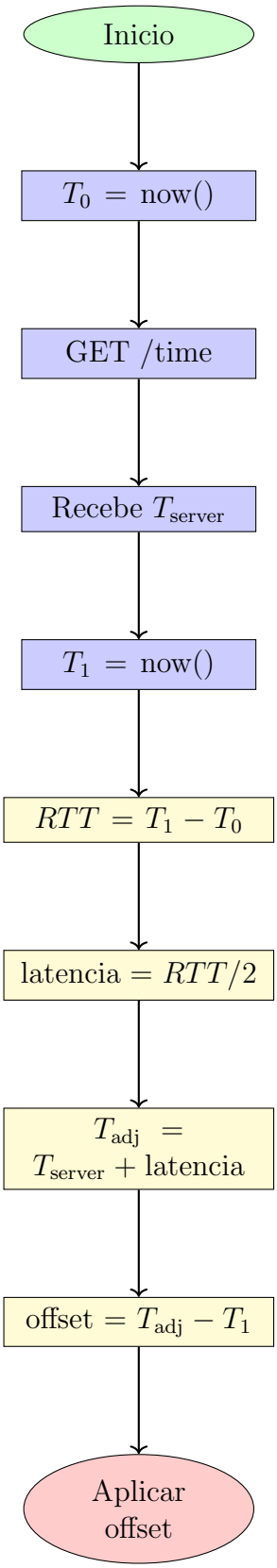

Figura 5: Fluxograma do Algoritmo de Cristian

7.3 Implementação no Cliente

A correção é aplicada localmente ao preparar pedidos de agendamento:

- Cliente calcula offset periodicamente (a cada 30s)
- Atualiza display de tempo em tempo real
- Aplica correção antes de enviar requisições POST

8 HATEOAS e Cliente Web

As respostas da API incluem `_links` com ações possíveis, implementando o conceito de HATEOAS (Hypermedia as the Engine of Application State).

8.1 Exemplo de Resposta HATEOAS

```
{
  "id": 123,
  "cientista_id": 7,
  "horario_inicio_utc": "2025-12-01T03:00:00Z",
  "status": "AGENDADO",
  "_links": {
    "self": { "href": "/api/v1/agendamentos/123" },
    "cientista": { "href": "/api/v1/cientistas/7" },
    "cancelar": {
      "href": "/api/v1/agendamentos/123",
      "method": "DELETE"
    }
  }
}
```

Listing 2: Resposta de agendamento com links

8.2 Benefícios do HATEOAS

- **Desacoplamento:** Cliente não precisa conhecer regras de negócio
- **Descoberta dinâmica:** Ações disponíveis são fornecidas pela API
- **Exemplo prático:** Botão "Cancelar" aparece apenas quando `_links.cancelar` existe

9 Diagrama de Estados do Agendamento

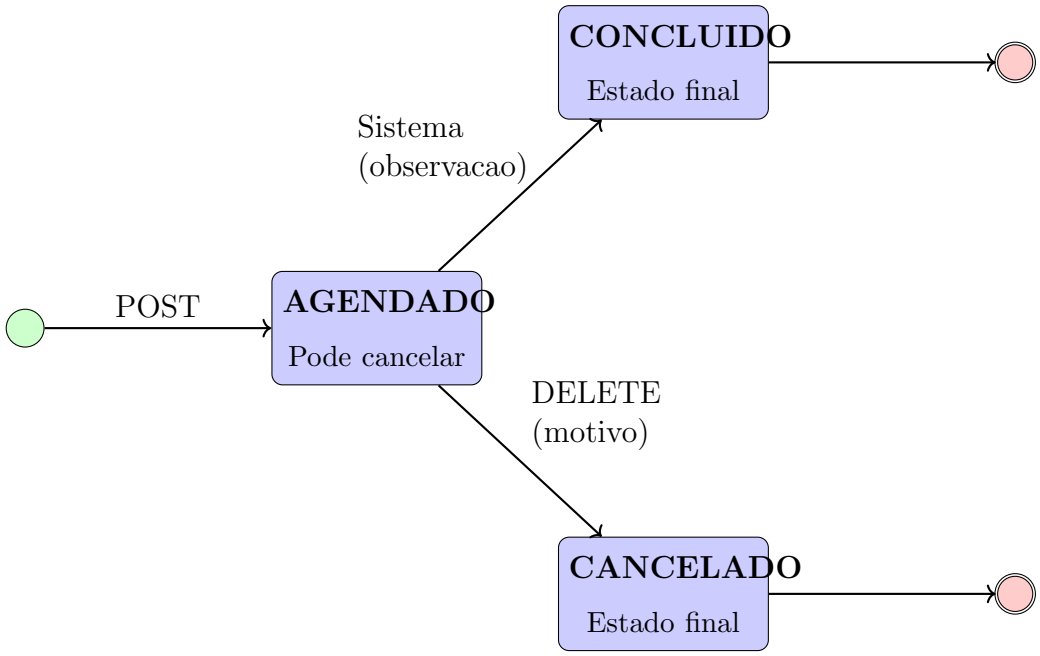

Figura 6: Diagrama de estados de um agendamento

10 Sistema de Toggle COM/SEM Lock

Para fins didáticos e demonstração, o sistema implementa um toggle que permite alternar entre duas versões:

- **Versão SEM LOCK (Entrega 2):** Demonstra o problema da condição de corrida
- **Versão COM LOCK (Entrega 3):** Demonstra a solução com exclusão mútua

10.1 Configuração

```
# Versao COM lock (Producao)
USE_LOCK=true

# Versao SEM lock (Demonstracao do problema)
# USE_LOCK=false
```

Listing 3: Arquivo .env - Toggle de lock

10.2 Scripts de Demonstração

- `demo_sem_lock.ps1`: Configura e executa versão sem lock
- `demo_com_lock.ps1`: Configura e executa versão com lock
- `demo_comparacao.ps1`: Executa ambas as versões sequencialmente

11 Testes de Concorrência e Evidências

11.1 Entrega 2 – SEM Lock (Demonstração do Problema)

Objetivo: Demonstrar a existência da condição de corrida.
Expectativa:

- Múltiplos agendamentos criados para o mesmo horário
- Logs entrelaçados mostrando race condition
- Múltiplos eventos `AGENDAMENTO_CRIADO` para o mesmo slot
- Estado inconsistente no banco de dados

Placeholder: Incluir screenshot da saída do teste
`python tests\test_concorrencia.py 10`

Deve mostrar:

- 2-10 sucessos (HTTP 201)
- 0-8 conflitos
- Consulta ao BD mostrando múltiplos registros

Figura 7: Saída do teste de estresse SEM lock

Placeholder: Incluir trechos de `logs/app.log`

Destacar:

- Logs entrelaçados de múltiplas threads
- Ausência de tentativas de lock
- Múltiplas verificações de conflito simultâneas

Figura 8: Trechos de logs mostrando condição de corrida

11.2 Entrega 3 – COM Lock (Solução)

Objetivo: Demonstrar a solução com exclusão mútua.
Expectativa:

- Apenas 1 sucesso (HTTP 201)
- 9 conflitos (HTTP 409) se 10 threads
- Banco consistente com único registro para o slot
- 1 evento `AGENDAMENTO_CRIADO`

- 9 eventos `AGENDAMENTO_CONFLITO`

Placeholder: Incluir screenshot da saída do teste
`python tests\test_com_lock.py 10`

Deve mostrar:

- 1 sucesso (HTTP 201)
- 9 conflitos (HTTP 409)
- Consulta ao BD mostrando 1 único registro
- Tempos de resposta (com overhead do lock)

Figura 9: Saída do teste de estresse COM lock

Placeholder: Incluir logs coordenados

Mostrar:

- Logs do Flask tentando adquirir lock
- Logs do Node.js concedendo/negando locks
- Correlation IDs correlacionados
- Sequência ordenada de eventos

Figura 10: Logs coordenados (Flask + Node.js) com correlation ID

11.3 Resumo Comparativo

Tabela 1: Comparação detalhada: Sistema SEM vs COM lock

Métrica	**SEM lock**	**COM lock**	**Observação**
Agendamentos criados	> 1	1	Exclusão mútua
Conflitos (HTTP 409)	0 a $n-1$	$n-1$	n = threads
Consistência do BD	Comprometida	Garantida	Estado único
Eventos AUDIT criação	Múltiplos	Único	1 por slot
Race condition	Presente	Resolvida	Lock funciona
Tempo médio (10 threads)	$\sim$150ms	$\sim$180ms	+20% overhead

12 Containerização e Orquestração

O sistema utiliza Docker Compose para orquestração dos microserviços.

12.1 Arquitetura Docker

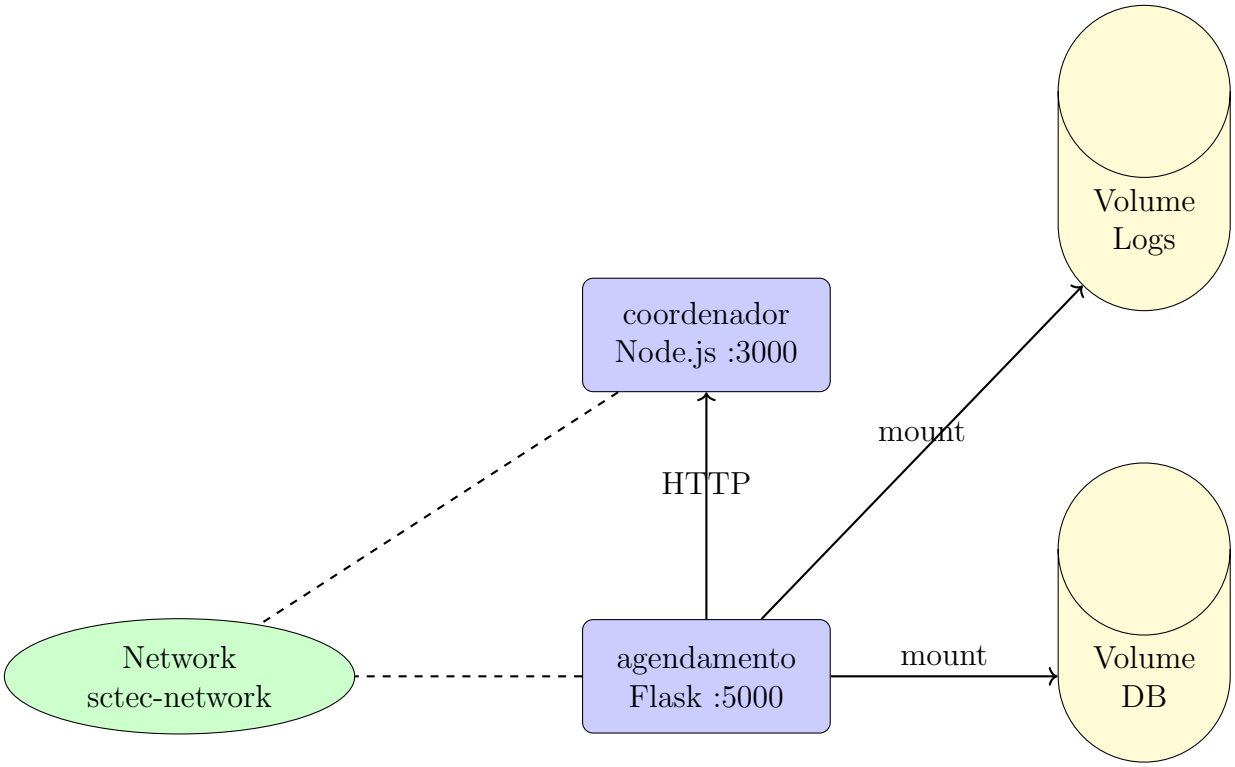

Figura 11: Arquitetura de containerização

12.2 Características do Docker Compose

- **Healthchecks:** Monitoramento automático de saúde dos serviços
- **Volumes:** Persistência de banco de dados e logs
- **Network:** Rede compartilhada para comunicação inter-serviços
- **Restart policy:** `unless-stopped` para robustez
- **Logging:** Rotação automática (max 10MB, 3 arquivos)
- **Dependencies:** Agendamento aguarda Coordenador estar saudável

12.3 Comandos Docker

```
# Iniciar sistema
docker-compose up --build -d

# Ver logs em tempo real
docker-compose logs -f

# Ver logs de um servico especifico
docker-compose logs -f agendamento

# Parar sistema
docker-compose stop
```

```
12
13 # Reiniciar servico
14 docker-compose restart coordenador
15
16 # Remover tudo (incluindo volumes)
17 docker-compose down -v
```

Listing 4: Comandos básicos Docker Compose

Placeholder: Screenshot de `docker-compose logs -f`

Mostrar:

- Logs entrelaçados dos dois serviços
- Cores diferentes por serviço
- Timestamps sincronizados
- Fluxo de uma requisição através dos serviços

Figura 12: Logs agregados com Docker Compose

13 Como Executar

13.1 Opção 1: Docker (Recomendado)

Pré-requisitos:

- Docker Desktop instalado
- Portas 3000 e 5000 livres

Windows:

```
1 start.bat
```

Linux/Mac:

```
1 chmod +x start.sh
2 ./start.sh
```

Acessar:

- Interface Web: http://localhost:5000
- API: http://localhost:5000/api/v1
- Coordenador: http://localhost:3000

13.2 Opção 2: Desenvolvimento Local

Terminal 1 - Coordenador (Node.js):

```
1 cd servico-coordenador
2 npm install
3 npm start
```

Terminal 2 - Agendamento (Python):

```
cd servico-agendamento
python -m venv venv
venv\Scripts\activate  # Windows
# source venv/bin/activate  # Linux/Mac
pip install -r requirements.txt
python run.py
```

14 Segurança e Boas Práticas

- **Variáveis sensíveis:** Sempre via ambiente, nunca no código
- **Secret Key:** Trocar em produção (`SECRET_KEY`)
- **Usuário não-root:** Preferir em containers
- **Dados sensíveis:** Não logar (emails, IDs devem ser sanitizados em produção)
- **Auditoria:** Apenas eventos de negócio, nunca dados pessoais completos
- **Healthchecks:** Implementados para detecção de falhas
- **Restart policy:** Recuperação automática de falhas
- **CORS:** Configurado apropriadamente
- **Input validation:** Todas as entradas são validadas

15 Conclusões e Trabalhos Futuros

15.1 Objetivos Atingidos

O sistema SCTEC atende completamente aos requisitos propostos:

- ✓ **API RESTful com HATEOAS:** Implementada e testada
- ✓ **Exclusão Mútua:** Resolvida via serviço coordenador
- ✓ **Sincronização de Tempo:** Algoritmo de Cristian funcional
- ✓ **Logging Completo:** Aplicação + Auditoria com correlation ID
- ✓ **Containerização:** Docker Compose operacional
- ✓ **Testes:** Demonstração clara do problema e da solução

15.2 Lições Aprendidas

1. **Complexidade de Sistemas Distribuídos:** A necessidade de coordenação entre serviços adiciona overhead mas é essencial para consistência
2. **Importância do Logging:** Em sistemas distribuídos, logs estruturados com correlation IDs são fundamentais para debugging
3. **Tradeoffs:** A exclusão mútua adiciona latência ($\sim$20%) mas garante consistência
4. **Design Modular:** Separação em microserviços facilita manutenção e escalabilidade

15.3 Trabalhos Futuros

Sugestões de evolução do sistema:

- **Autenticação/Autorização:** JWT tokens para cientistas, roles e permissões
- **PostgreSQL:** Migração de SQLite para banco production-ready
- **Redis:** Locks distribuídos com suporte a cluster
- **WebSockets:** Atualização em tempo real da interface
- **Kubernetes:** Deploy em cluster para alta disponibilidade
- **Métricas:** Prometheus + Grafana para monitoramento
- **CI/CD:** Pipeline automatizado com GitHub Actions
- **Testes Unitários:** Cobertura completa do código
- **Load Balancer:** Múltiplas instâncias do serviço de agendamento
- **Message Queue:** RabbitMQ ou Kafka para comunicação assíncrona

A Apêndice A: Sumário de Endpoints

Endpoint	Descrição
GET /api/v1/time	Timestamp oficial do servidor para sincronização + links HATEOAS
GET /api/v1/cientistas	Lista paginada com filtros; inclui links HATEOAS por recurso
POST /api/v1/cientistas	Criação de cientista com validação completa e auditoria
GET /api/v1/cientistas/{id}	Detalhes de um cientista específico + links HATEOAS
GET /api/v1/cientistas/{id}/agendamentos	Lista de agendamentos de um cientista específico
GET /api/v1/agendamentos	Lista paginada com filtros (data, status, cientista) e ordenação

Endpoint	Descrição
POST /api/v1/agendamentos	Criação de agendamento (operação crítica com lock obrigatório)
GET /api/v1/agendamentos/{id}	Detalhes de agendamento + HATEOAS condicional (status-based)
DELETE /api/v1/agendamentos/{id}	Cancelamento de agendamento (soft delete com motivo)
POST /lock	(Coordenador) Tenta adquirir lock de um recurso
POST /unlock	(Coordenador) Libera lock de um recurso
GET /locks	(Coordenador) Lista locks ativos (debug)
GET /health	(Coordenador) Health check do serviço

Tabela 2: Sumário completo de endpoints da API

B Apêndice B: Exemplos de Logs

B.1 Log de Aplicação

```
[INFO] 2025-11-16T18:00:04.500Z servico-agendamento a1b2c3d4: Requisicao recebida para POST /agendamentos
[INFO] 2025-11-16T18:00:04.505Z servico-agendamento a1b2c3d4: Validando dados do agendamento
[INFO] 2025-11-16T18:00:04.510Z servico-agendamento a1b2c3d4: Tentando adquirir lock para o recurso Hubble-Acad_2025-12-01T03:00:00Z
[INFO] 2025-11-16T18:00:05.120Z servico-agendamento a1b2c3d4: Lock adquirido com sucesso
[INFO] 2025-11-16T18:00:05.122Z servico-agendamento a1b2c3d4: Iniciando verificacao de conflito no BD
[INFO] 2025-11-16T18:00:05.123Z servico-agendamento a1b2c3d4: Salvando novo agendamento no BD
[INFO] 2025-11-16T18:00:05.125Z servico-agendamento a1b2c3d4: Agendamento salvo com ID 123
[INFO] 2025-11-16T18:00:05.127Z servico-agendamento a1b2c3d4: Liberando lock
[INFO] 2025-11-16T18:00:05.130Z servico-agendamento a1b2c3d4: Resposta 201 enviada
```

Listing 5: Exemplo de logs de aplicação

B.2 Log de Auditoria

```
{
  "timestamp_utc": "2025-11-16T18:00:05.297Z",
  "level": "AUDIT",
  "event_type": "AGENDAMENTO_CRIADO",
  "service": "servico-agendamento",
  "correlation_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "details": {
    "agendamento_id": 123,
    "cientista_id": 7,
    "cientista_nome": "Marie Curie",
```

```
    "horario_inicio_utc": "2025-12-01T03:00:00Z",
    "horario_fim_utc": "2025-12-01T03:30:00Z",
    "objeto_celeste": "NGC 1300",
    "duracao_minutos": 30
  }
}
```

Listing 6: Evento AUDIT: AGENDAMENTO_CRIADO

```
{
  "timestamp_utc": "2025-11-16T19:15:22.458Z",
  "level": "AUDIT",
  "event_type": "AGENDAMENTO_CANCELADO",
  "service": "servico-agendamento",
  "correlation_id": "b2c3d4e5-f6a7-8901-bcde-f12345678901",
  "details": {
    "agendamento_id": 123,
    "cientista_id": 7,
    "cientista_nome": "Marie Curie",
    "horario_inicio_utc": "2025-12-01T03:00:00Z",
    "motivo": "Condicoes meteorologicas desfavoraveis"
  }
}
```

Listing 7: Evento AUDIT: AGENDAMENTO_CANCELADO

C Apêndice C: Comandos Úteis

C.1 Docker

```
# Ver status dos containers
docker-compose ps

# Ver logs em tempo real
docker-compose logs -f

# Ver logs de um servico especifico
docker-compose logs -f agendamento
docker-compose logs -f coordenador

# Ver ultimas 100 linhas de log
docker-compose logs --tail=100 agendamento

# Verificar health status
docker inspect --format='{{.State.Health.Status}}' sctec-agendamento
docker inspect --format='{{.State.Health.Status}}' sctec-coordenador

# Acessar shell do container
docker exec -it sctec-agendamento /bin/bash
docker exec -it sctec-coordenador /bin/sh

# Ver uso de recursos
docker stats

# Inspecionar rede
docker network inspect sctec-network
```

```

# Inspecionar volumes
docker volume inspect sctec-agendamento-db
docker volume inspect sctec-agendamento-logs
```

Listing 8: Comandos Docker para diagnóstico

C.2 Análise de Logs

```
# Ver todos os eventos de auditoria
cat servico-agendamento/logs/audit.log | jq '.event_type'

# Filtrar eventos especificos
cat servico-agendamento/logs/audit.log | jq 'select(.event_type=="
    AGENDAMENTO_CRIADO")'

# Buscar por correlation_id
cat servico-agendamento/logs/app.log | grep "a1b2c3d4"

# Ver tentativas de lock
cat servico-agendamento/logs/app.log | grep -E '(Tentando adquirir lock|
    Lock adquirido|Falha ao adquirir)'

# Contar eventos por tipo
cat servico-agendamento/logs/audit.log | jq -r '.event_type' | sort |
    uniq -c
```

Listing 9: Comandos para análise de logs

C.3 Banco de Dados

```
# Acessar banco de dados
sqlite3 servico-agendamento/instance/telescopio.db

# Queries uteis
SELECT * FROM cientistas;
SELECT * FROM agendamentos WHERE status = 'AGENDADO';
SELECT COUNT(*) FROM agendamentos GROUP BY status;

# Ver estrutura das tabelas
.schema cientistas
.schema agendamentos

# Exportar dados
.mode csv
.output agendamentos.csv
SELECT * FROM agendamentos;
.output stdout
```

Listing 10: Comandos SQLite para inspeção

D Apêndice D: Troubleshooting

D.1 Problemas Comuns

Problema	**Solução**
Porta em uso	Verificar processos: `netstat -ano | findstr ":5000"`. Matar processo ou usar porta alternativa.
Coordenador offline	Verificar logs: `docker-compose logs coordenador`. Reiniciar: `docker-compose restart coordenador`.
BD corrompido	Remover volume: `docker-compose down -v`. Reiniciar: `docker-compose up -d`.
Lock travado	Aguardar timeout (30s) ou reiniciar coordenador: `docker-compose restart coordenador`.
Build falha	Limpar cache: `docker-compose build --no-cache`. Verificar logs de build.

Tabela 3: Soluções para problemas comuns