Documentação do Módulo de SLA Tracking

1. Visão Geral e Objetivos do Sistema

O **Módulo de SLA Tracking** é um sistema robusto desenhado para monitorizar, calcular e reportar o cumprimento de Acordos de Nível de Serviço (SLAs). O seu principal objetivo é fornecer uma visão clara e precisa sobre a disponibilidade de serviços, registar incidentes, janelas de manutenção e violações de SLA, permitindo uma gestão proativa da infraestrutura tecnológica.

O sistema foi construído em **TypeScript** com **Node.js** e **Express**, utilizando **Prisma** como ORM para interagir com uma base de dados **SQLite**. A sua arquitetura é modular, separando responsabilidades em controladores, serviços, repositórios e tarefas agendadas (jobs), o que facilita a manutenção e escalabilidade.

Os principais objetivos são:

- **Calcular a Disponibilidade (Uptime):** Processar dados de status de serviços para calcular a percentagem de tempo em que um serviço esteve operacional.
- **Gerir Políticas de SLA:** Permitir a criação, atualização e consulta de políticas de SLA que definem as metas de disponibilidade para cada serviço.
- **Registar Eventos Relevantes:** Manter um registo histórico de incidentes (períodos de downtime) e janelas de manutenção programada.
- **Detetar e Alertar Violações:** Identificar quando o nível de serviço de um componente fica abaixo da meta definida na sua política de SLA e registar essa violação.
- **Fornecer Dados para Dashboards:** Expor endpoints de API que entregam dados consolidados sobre o estado dos SLAs, permitindo a sua visualização em plataformas como o InfraWatch.

2. Arquitetura e Implementação

O sistema segue uma arquitetura em camadas, o que promove a separação de conceitos e a reutilização de código.

Componentes Principais:

- **server.ts**: Ponto de entrada da aplicação. É responsável por inicializar o servidor Express, configurar middlewares (como o CORS e o parser de JSON), registar as rotas da API e iniciar as tarefas agendadas (jobs).
- **API (Directório api)**:
    - **routes.ts / routes-with-swagger.ts**: Define os endpoints da API, associando cada rota (URL e método HTTP) a um controlador específico. A versão com

Swagger também gera a documentação da API de forma automática.
    - **Controladores (Directório controllers)**: Recebem os pedidos HTTP, validam os dados de entrada (parâmetros, body) e orquestram a execução da lógica de negócio, invocando os serviços apropriados. No final, formatam e enviam a resposta ao cliente.
- **Lógica de Negócio (Directório services)**:
    - Contém a lógica central da aplicação. Por exemplo, o windowCalculator.ts é crucial para calcular a disponibilidade dentro de janelas de tempo, considerando incidentes e manutenções. O backfill.ts permite reprocessar dados históricos.
- **Acesso a Dados (Directório repositories)**:
    - Abstrai a comunicação com a base de dados. Cada repositório (ex: policyRepo.ts, incidentRepo.ts) é responsável pelas operações de CRUD (Create, Read, Update, Delete) de uma entidade específica, utilizando o Prisma para executar as queries.
- **Tarefas Agendadas (Directório jobs)**:
    - Processos que correm em segundo plano, de forma periódica.
        - **ingestStatus.job.ts**: Ingere dados de status de uma fonte externa (presumivelmente, o sistema de monitorização principal como o InfraWatch) e armazena-os.
        - **rollupWindow.job.ts**: Processa os dados brutos de status em janelas de tempo agregadas (ex: a cada 5 minutos), calculando o uptime/downtime para cada janela.
        - **alerting.job.ts**: Verifica periodicamente se ocorreram violações de SLA e, no futuro, poderá acionar alertas.
- **Base de Dados (Directório db)**:
    - **prisma.schema**: Define o esquema da base de dados, incluindo tabelas, colunas, tipos de dados e relações. As entidades principais são SlaPolicy, SlaIncident, SlaMaintenance, SlaStatusWindow e SlaViolation.
    - **migrations**: Contém os scripts SQL para criar e atualizar a estrutura da base de dados.

3. Fluxo de Processamento

O coração do sistema é um processo contínuo de ingestão, cálculo e agregação de dados, que funciona da seguinte forma:

1. **Definição de Políticas:** Um utilizador (ou sistema externo) utiliza o endpoint POST /policies para definir um SLA para um determinado serviço (identificado por checkId). A política define a meta de disponibilidade (ex: 99.9%).

2. **Ingestão de Status (Job):** O ingestStatus.job corre a cada minuto. A sua função é obter o estado atual (UP/DOWN) de cada serviço monitorizado a partir de uma fonte externa. Estes dados brutos são guardados na tabela SlaStatus.
3. **Agregação em Janelas (Job):** O rollupWindow.job corre a cada 5 minutos. Ele analisa os dados brutos da SlaStatus dos últimos 5 minutos e cria uma entrada na SlaStatusWindow. Esta entrada resume o período, indicando o tempo total de uptime e downtime para cada serviço nessa janela de 5 minutos.
4. **Cálculo de Disponibilidade (API):** Quando um utilizador consulta o endpoint GET /status/:checkId, o sistema:
    - Obtém a política de SLA para o checkId.
    - Consulta todas as SlaStatusWindow para o período de tempo desejado (ex: últimos 30 dias).
    - Obtém também todos os SlaIncident e SlaMaintenance registados.
    - O windowCalculator.ts entra em ação: ele soma todo o downtime das janelas, subtrai o tempo que estava coberto por uma janela de manutenção aprovada e calcula a percentagem final de disponibilidade.
5. **Registo de Incidentes e Manutenções (API):**
    - Os endpoints POST /incidents e POST /maintenance permitem registar manualmente (ou via automação) os períodos de indisponibilidade ou manutenção. Estes registos são cruciais para o cálculo correto do SLA, pois o tempo de manutenção planeada geralmente não conta como downtime para o cálculo do SLA.
6. **Deteção de Violações (Job):** O alerting.job corre periodicamente, utiliza o mesmo windowCalculator para verificar o SLA atual de cada serviço e, se a disponibilidade estiver abaixo da meta definida na política, regista uma entrada na tabela SlaViolation.

4. Endpoints da API

A seguir, uma descrição detalhada dos endpoints disponíveis.

Políticas de SLA (/policies)

GET /policies

- **Objetivo:** Obter a lista de todas as políticas de SLA configuradas.
- **Parâmetros de Query:**
    - checkId (opcional, string): Filtra as políticas por um ID de serviço específico.
- **Retorno (200 OK):** Um array de objetos de política.

```
[
 {
```

```
    "id": "clv7yq8c3000108l760a3g9h2",
    "checkId": "service-api-prod",
    "name": "API Principal",
    "description": "SLA para a API de produção.",
    "targetUptime": 99.95,
    "period": "MONTHLY",
    "createdAt": "2024-08-14T14:30:00.000Z",
    "updatedAt": "2024-08-14T14:30:00.000Z"
  }
]
```


POST /policies

- **Objetivo:** Criar uma nova política de SLA.
- **Corpo do Pedido (Body):**


```
{
  "checkId": "service-web-prod",
  "name": "Website de Produção",
  "targetUptime": 99.9,
  "period": "MONTHLY"
}
```


- **Retorno (201 Created):** O objeto da política criada.

PUT /policies/:id

- **Objetivo:** Atualizar uma política de SLA existente.
- **Parâmetros de Rota:**
  - id (obrigatório, string): O ID da política a ser atualizada.
- **Corpo do Pedido (Body):** Campos a serem atualizados.


```
{
  "targetUptime": 99.98
}
```


- **Retorno (200 OK):** O objeto da política atualizada.

DELETE /policies/:id

- **Objetivo:** Apagar uma política de SLA.
- **Parâmetros de Rota:**
  - id (obrigatório, string): O ID da política a ser apagada.

- **Retorno (204 No Content):** Resposta vazia em caso de sucesso.

Incidentes (/incidents)

GET /incidents

- **Objetivo:** Obter a lista de incidentes registados.
- **Parâmetros de Query:**
  - checkId (opcional, string): Filtra por ID do serviço.
  - startTime (opcional, ISO 8601 string): Início do intervalo de tempo.
  - endTime (opcional, ISO 8601 string): Fim do intervalo de tempo.
- **Retorno (200 OK):** Um array de objetos de incidente.

POST /incidents

- **Objetivo:** Registar um novo incidente (período de downtime).
- **Corpo do Pedido (Body):**

```
{
  "checkId": "service-db-prod",
  "startTime": "2024-08-14T10:00:00.000Z",
  "endTime": "2024-08-14T10:15:00.000Z",
  "title": "Falha na Base de Dados Primária"
}
```

- **Retorno (201 Created):** O objeto do incidente criado.

Manutenções (/maintenance)

GET /maintenance

- **Objetivo:** Obter a lista de janelas de manutenção.
- **Parâmetros de Query:**
  - checkId (opcional, string): Filtra por ID do serviço.
  - startTime (opcional, ISO 8601 string): Início do intervalo de tempo.
  - endTime (opcional, ISO 8601 string): Fim do intervalo de tempo.
- **Retorno (200 OK):** Um array de objetos de manutenção.

POST /maintenance

- **Objetivo:** Registar uma nova janela de manutenção.
- **Corpo do Pedido (Body):**

```
{
  "checkId": "service-api-prod",
  "startTime": "2024-08-15T02:00:00.000Z",
  "endTime": "2024-08-15T03:00:00.000Z",
```

```
  "title": "Atualização de segurança do servidor"
}
```


- **Retorno (201 Created):** O objeto da manutenção criada.

Violações de SLA (/violations)

GET /violations

- **Objetivo:** Obter a lista de violações de SLA registadas.
- **Parâmetros de Query:**
  - checkId (opcional, string): Filtra por ID do serviço.
- **Retorno (200 OK):** Um array de objetos de violação.


```
[
 {
  "id": "clv8z12a4000108l7abcd1234",
  "policyId": "clv7yq8c3000108l760a3g9h2",
  "checkId": "service-api-prod",
  "actualUptime": 99.85,
  "startTime": "2024-08-01T00:00:00.000Z",
  "endTime": "2024-08-14T15:00:00.000Z",
  "createdAt": "2024-08-14T15:00:05.000Z"
 }
]
```


Status do SLA (/status)

GET /status/:checkId

- **Objetivo:** Obter o estado de cumprimento do SLA para um serviço específico.
- **Parâmetros de Rota:**
  - checkId (obrigatório, string): O ID do serviço a ser consultado.
- **Parâmetros de Query:**
  - range (opcional, string, ex: 7d, 30d, 90d): Define o período de cálculo. Se não for fornecido, usa o período da política (ex: MONTHLY).
- **Retorno (200 OK):** Um objeto detalhado com o estado do SLA.


```
{
 "checkId": "service-api-prod",
 "policy": {
  "targetUptime": 99.95,
  "period": "MONTHLY"
 },
```

```
  "range": {
    "startTime": "2024-08-01T00:00:00.000Z",
    "endTime": "2024-08-14T15:10:00.000Z"
  },
  "uptime": {
    "percentage": 99.96,
    "totalTimeMs": 1177800000,
    "upMs": 1177328880,
    "downMs": 471120,
    "maintenanceMs": 0
  },
  "isViolation": false,
  "errorBudget": {
    "totalBudgetMs": 591000,
    "consumedMs": 471120,
    "remainingMs": 119880,
    "consumedPercentage": 79.71
  }
}
```


5. Integração com o InfraWatch

A integração deste módulo com a plataforma **InfraWatch** é o que materializa o seu valor, fornecendo os dados necessários para os dashboards e relatórios de SLA. A integração pode ser feita da seguinte forma:

1. **Fonte de Dados para os Jobs:**
    - O **ingestStatus.job.ts** precisa de ser configurado para consumir dados do motor de monitorização do InfraWatch. O InfraWatch, que já realiza a monitorização (via ping, SNMP, etc.), deve expor um endpoint interno que o módulo de SLA possa consultar a cada minuto para obter o status (up/down) de todos os serviços (checks).
    - A configuração deste endpoint de origem deve ser feita no ficheiro .env do módulo de SLA.
2. **Exibição de Dados no Frontend do InfraWatch:**
    - O frontend do InfraWatch (React/Angular) deve fazer chamadas à API do Módulo de SLA para popular os seus componentes visuais.
    - **Dashboard Principal:** Para cada serviço exibido no dashboard do InfraWatch, uma chamada a GET /status/:checkId pode ser feita para obter o

uptime.percentage e isViolation, permitindo exibir um ícone (verde/vermelho) ou o valor percentual do SLA atual.

- **Página de Relatórios e SLA:** Esta página no InfraWatch pode ser mais detalhada.
  - Pode listar todas as políticas (GET /policies) e, para cada uma, mostrar o estado atual do SLA (GET /status/:checkId).
  - Pode incluir gráficos históricos, obtendo dados de SlaStatusWindow (endpoint a ser criado, se necessário) para mostrar a evolução da disponibilidade ao longo do tempo.
  - Uma tabela de incidentes (GET /incidents) e manutenções (GET /maintenance) pode ser exibida para dar contexto a qualquer período de downtime.
  - A lista de violações (GET /violations) pode ser usada para um relatório de conformidade.

3. **Gestão de Políticas a partir do InfraWatch:**
   - O frontend do InfraWatch pode implementar uma interface de administração onde os utilizadores podem criar, editar e apagar políticas de SLA, utilizando os endpoints POST, PUT, e DELETE do recurso /policies.

Exemplo de Fluxo de Integração:

- Um utilizador acede à página de detalhes de um serviço no **InfraWatch**.
- O frontend do InfraWatch faz uma chamada para GET https://<sla_module_host>/api/status/service-api-prod?range=30d.
- A resposta JSON é usada para preencher um gráfico de "pizza" com a percentagem de uptime, um indicador de "Error Budget" e uma lista dos últimos incidentes relacionados.

6. Configuração e Definições

As configurações principais da aplicação estão no ficheiro .env:

- DATABASE_URL: A connection string para a base de dados. Ex: file:./dev.db para SQLite.
- PORT: A porta onde o servidor da API irá correr. Ex: 3000.
- CORS_ORIGIN: O URL do frontend (InfraWatch) que terá permissão para aceder a esta API. Ex: http://localhost:3001.
- BACKFILL_START_DATE: (Opcional) Data de início para o reprocessamento de dados históricos.
- INGEST_STATUS_CRON: A frequência (formato cron) com que o job de ingestão de status corre. Padrão: * * * * * (a cada minuto).
- ROLLUP_WINDOW_CRON: A frequência com que o job de agregação corre.

Padrão: */5 * * * * (a cada 5 minutos).

- ALERTING_CRON: A frequência com que o job de verificação de violações corre. Padrão: */15 * * * * (a cada 15 minutos).