Projeto 1: Disparador de Webhooks

História do usuário

No PlugBoleto, uma das funcionalidades que temos é o Webhook, que consiste em notificar o cliente através de um envio por método POST direcionado a um endereço do sistema conectado em nosso produto, de atualizações nas cobranças geradas.

Mesmo com redundâncias do nosso lado, eventualmente os clientes por motivos diversos perdem essas notificações e não recebem a atualização necessária. Com isso, eles acessam o suporte e solicitam para os nossos clientes que o webhook seja reenviado para que a cobrança seja sincronizada entre a nossa aplicação e o sistema do cliente.

Esse processo requer do nosso time de suporte uma intervenção manual no produto, tendo que reenviar para a fila de notificações as cobranças solicitadas pelos clientes, gerando quantidades significativas de suporte.

A demanda consiste em construir um *disparador de webhooks,* que permita a própria empresa possa por conta própria disparar os webhooks quando precisar, acessando essa ferramenta, sem precisar acionar o nosso time de suporte.

Tecnologias a serem utilizadas

Backend: Node.js, TypeScript, Express.js, Fastify
Autenticação: JWT, OAuth 2.0
Mensageria: RabbitMQ, Kafka
Banco de Dados: PostgreSQL, TypeORM, Prisma
Cache: Redis
Logging e Monitoramento: Winston, Pino, Prometheus, Grafana
Infraestrutura: Docker, Kubernetes, Terraform
CI/CD: GitLab CI

Requisitos

1. Filtros e Consultas

- Implementar filtros avançados para consulta de notificações com os seguintes parâmetros:
    - **Período:** Definir intervalo de datas com limite máximo de dias para evitar consultas excessivas (24 horas).
    - **Cedente (autenticado):** Permitir filtragem por credenciais do emissor.
    - **Conta:** Possibilitar busca por conta específica vinculada à notificação.
    - **Convênio:** Filtragem por convênio bancário associado.
    - **IDs de Integração:** Consultas direcionadas por identificadores de integração.

2. Reprocessamento de Notificações

- Exigir dados completos do cliente para habilitar o reprocessamento de notificações.

- Implementar múltiplos tipos de reprocessamento, garantindo flexibilidade operacional:
    - **Webhook:** Reenvio automático para endpoints configurados.
    - **Agendado (E-mail):** Envio programado de notificações por e-mail.
    - **Eventos:** Gatilhos configuráveis para disparo de reprocessamentos com base em eventos específicos.

3. Gerenciamento de Situações das Notificações

- Permitir multi-seleção de situações ao consultar ou reprocessar notificações, contemplando:
    - **Registro:** Confirmação do cadastro do boleto.
    - **Baixa:** Cancelamento ou liquidação antecipada do título.
    - **Alteração:** Modificação de dados do boleto.
    - **Liquidação:** Confirmação do pagamento e quitação do boleto.

4. Requisições e Controle de Protocolos

- Implementar processamento assíncrono com geração de protocolos para rastreabilidade.
- Restringir novas requisições de mesmo tipo enquanto protocolos anteriores estiverem em aberto.
- Evitar duplicidade ao limitar novos protocolos idênticos a **uma requisição por hora**.

5. Gestão de Pacotes de Processamento

- Criar e processar pacotes de **50 boletos/notificações por lote** para operações de criação e deleção, otimizando a eficiência e reduzindo carga no sistema.

Sugestão de Arquitetura:

1. API Gateway (Express/Fastify)

O **API Gateway** será responsável por centralizar todas as requisições dos clientes, aplicar autenticação, validar restrições e encaminhar os pedidos para processamento assíncrono via fila de mensagens.

1.1. Recebimento de Requisições

- Implementação via **Express.js** ou **Fastify**, priorizando alta performance e baixa latência.
- Suporte para **REST API**.

1.2. Autenticação e Segurança

- **JWT:** Validação de identidade do cliente via tokens assinados.
- **OAuth 2.0:** Suporte a fluxos de autorização para clientes externos.
- **Rate Limiting:** Implementação de limites de requisição/protocolos via **Redis**.

1.3. Geração de Protocolo Único

Cada requisição recebe um identificador único (UUID v4) associado ao cliente e ao tipo de processamento.

Regras de Limite

- **Máximo de 3 protocolos por hora por cliente**, sendo:
  - 1 protocolo do tipo Webhook
  - 1 protocolo do tipo Notificação Agendada
  - 1 protocolo do tipo Evento
- Bloqueio de protocolos com conteúdo idêntico dentro do período de uma hora.

1.4. Envio Assíncrono para a Fila (RabbitMQ/Kafka)

- O protocolo gerado é enviado para a **fila de mensagens** garantindo desacoplamento e escalabilidade.
- **Headers e Metadata** da requisição são mantidos para rastreabilidade.

1.5. Consulta do Status do Protocolo

- Endpoint específico para verificar o status do processamento no **Banco de Dados**.

2. Fila de Mensagens (RabbitMQ/Kafka)

Responsável pelo processamento assíncrono e desacoplamento entre a API Gateway e os microserviços de backend.

2.1. Garantia de Entrega

- **RabbitMQ (fila tradicional):** Mensagens são persistidas e consumidas na ordem FIFO.
- **Kafka (event streaming):** Permite reprocessamento e alto throughput.

2.2. Benefícios

- **Evita sobrecarga na API Gateway.**
- **Garante resiliência ao processamento,** permitindo retries em caso de falha.
- **Distribuição de carga entre múltiplos consumidores (scalability).**

3. Microserviço Worker (Processamento de Notificações)

O **Worker** é responsável por consumir mensagens da fila, executar a lógica de processamento e atualizar o status da requisição.

3.1. Consumo da Fila

- Conexão direta com **RabbitMQ/Kafka**, garantindo processamento eficiente.
- Uso de **workers concorrentes** para escalabilidade horizontal.

3.2. Processamento da Solicitação

- Identificação do tipo de notificação (Webhook, Notificação Agendada).
- Execução das rotinas necessárias (envio de e-mail, disparo de webhook, etc.).
- Aplicação de **regras de reprocessamento** conforme necessário.

3.3. Atualização do Status no Banco de Dados

- O protocolo recebe atualização de status (**pendente, em processamento, concluído, falha**).
- Logs de erro e auditoria são registrados para troubleshooting.

4. Banco de Dados (PostgreSQL)

O **PostgreSQL** será utilizado para armazenar e rastrear o status dos protocolos e notificações.

4.1. Estrutura do Banco de Dados

- **Tabela de Protocolos:**
  - `id (UUID, PK)`
  - `cnpj (VARCHAR: removendo caracteres especiais)`
  - `tipo (ENUM: webhook, notificação agendada, evento)`
  - `status (ENUM: pendente, processando, concluído, falha)`
  - `data_criacao (TIMESTAMP)`
  - `data_atualizacao (TIMESTAMP)`
- **Tabela de Logs e Auditoria:**
  - Histórico de eventos e mudanças nos protocolos.

4.2. Restrições e Regras

- **Validação de restrições da API:** Impedir criação de protocolos duplicados dentro do período estipulado (1 hora).
- **Índices e Otimizações:** Uso de índices em colunas de busca frequente (`cnpj`, `tipo`, `status`).

Materiais de apoio

Recursos de Referência:

- Monitor SEFAZ da Tecnospeed: https://monitor.tecnospeed.com.br/

Utilizar o Monitor da SEFAZ como referência para a construção do Monitorador Bancário, lembrando-se de utilizar a logo e identidade da Tecnospeed e do Plugbank.

- Documentação do Plugboleto

Utilizar o nosso produto para fazer os disparos de requisições na nossa API. Assim, utilizam-se semelhantes para envios. Documentação pública: atendimento.tecnospeed.com.br.

- Repositório de demonstrações de uso do Plugboleto

Caso haja dúvidas sobre a forma de implementar o produto, possuímos um repositório de códigos que podem ser utilizados como exemplo. Repositório público: https://github.com/tecnospeed/Componente-Boleto/tree/master/demonstracoes/API

- PageSpeed Test: PageSpeed Insights (web.dev)

Para aprimorar a escalabilidade e responsividade, pode ser utilizada a ferramenta PageSpeed Test para avaliar o projeto. Lembrando que para que ele funcione, é necessário que o projeto esteja hospedado em um ambiente em nuvem.

- Identidade visual da Tecnospeed

Link com arquivos da identidade visual da Tecnospeed para utilizar no projeto: https://drive.google.com/drive/folders/1Oky--tKCYfEZBj5p9DtJp0EMY6fVSJmX?usp=drive_link

Diferenciais no projeto

Serão diferenciais nos projetos apresentados:

- Ter a maior quantidade de bancos e serviços disponíveis no produto (conforme a lista apresentada nos requisitos).
- Fidelidade no uso das tecnologias listadas aqui com as apresentadas no projeto
- Maior proximidade/similaridade com o layout e design system da Tecnospeed
- Aplicação de técnicas de escalabilidade dos serviços, para suportar grandes capacidades de acesso
- Responsividade do front-end para dispositivos móveis
- Ter aplicação de técnicas de SEO (Search Engine Optimization) para facilitar o usuário encontrar o monitor no Google.