Universidade Federal do Ceará
Campus Quixadá
QXD0068 – Reuso de Software
Prof. Francisco Victor da Silva Pinheiro
Gabriel Alves
Juan Pimentel

Trabalho 2

Serviço Reutilizável (SOA/Microserviço) ou Mini-LPS

Quixadá-CE
2025

Sumário

1. Introdução

A definição arquitetural adotada para o desenvolvimento deste projeto fundamenta-se na **Opção A**, caracterizada pela implementação de um Serviço Reutilizável estruturado sob o paradigma de microsserviços. Tal escolha justifica-se pela observação crítica de que a geração de documentos — a exemplo de contratos, relatórios gerenciais e certificados acadêmicos — constitui uma demanda transversal e recorrente em múltiplos ecossistemas de software. Quando esse cenário não é gerenciado adequadamente, resulta em uma significativa duplicação de lógica e em um indesejável acoplamento entre as regras de negócio específicas de cada setor e as camadas de apresentação visual.

Nesse contexto, a solução proposta consiste na elaboração de um microsserviço agnóstico, projetado para operar mediante a recepção de dados estruturados em formato JSON, os quais são associados a um modelo pré-definido (*template*). O serviço encarrega-se integralmente do processamento, retornando o arquivo PDF finalizado e pronto para utilização. Esta abordagem estratégica promove o encapsulamento da complexidade inerente à manipulação de bibliotecas de renderização e otimiza o ciclo de desenvolvimento ao centralizar a funcionalidade. Consequentemente, assegura-se que qualquer subsistema corporativo — variando desde o departamento de Recursos Humanos até os setores de Vendas e Acadêmico — esteja apto a consumir este serviço de maneira padronizada via API, enviando exclusivamente as informações necessárias e abstraindo-se inteiramente das especificidades técnicas envolvidas na materialização dos documentos.

2. Descrição da Solução

Para este projeto, a abordagem arquitetural escolhida foi o desenvolvimento de um Micro Serviço independente, especializado na geração de documentos. A escolha por separar esta funcionalidade em um serviço autônomo, em vez de integrá-la como um módulo de uma aplicação monolítica, deve-se à natureza computacionalmente intensiva da renderização de PDFs. O domínio da aplicação foca na transformação de dados estruturados (JSON) e modelos visuais (templates) em documentos estáticos portáteis, atendendo a múltiplos contextos de negócio, como o setor acadêmico, financeiro e administrativo.

A arquitetura proposta segue o fluxo de uma API RESTful *stateless*. O cliente envia uma requisição HTTP contendo o nome do template desejado e os dados variáveis. O microserviço, atuando como uma caixa preta de processamento, realiza a validação, a compilação do HTML dinâmico e a renderização final utilizando um navegador *headless*. Todo o processo é orquestrado por um controlador central que gerencia filas de concorrência e tentativas de reprocessamento, garantindo que a aplicação principal não sofra degradação de performance durante a geração de relatórios complexos.

3. Tecnologias e Padrões Utilizados

A base tecnológica da solução foi construída sobre o Node.js em conjunto com o framework Express. A escolha do Node.js justifica-se pela sua arquitetura orientada a eventos e não bloqueante, características essenciais para um serviço que lida com operações de I/O intensas e

múltiplas requisições simultâneas. O Express complementa essa base oferecendo uma estrutura minimalista, porém robusta, para o roteamento e gerenciamento dos endpoints da API.

Para a renderização dos documentos, foi adotado o Puppeteer. Esta ferramenta controla uma instância do Chrome em modo headless (sem interface gráfica), permitindo que o sistema gere PDFs com fidelidade visual absoluta, suportando CSS moderno, fontes customizadas e layouts complexos exatamente como seriam vistos em um navegador. Diferente de bibliotecas que apenas aproximam o layout, o Puppeteer garante que o resultado final seja pixel-perfeito.

No que tange à construção dos layouts, utilizou-se o Handlebars (HBS). Esta template engine favorece a separação clara entre a lógica de dados e a camada de apresentação. Através do uso de Partials (componentes de interface), foi possível criar blocos de código reutilizáveis — como cabeçalhos, rodapés, tabelas e assinaturas — que são compartilhados entre diferentes tipos de documentos, promovendo a modularidade e facilitando a manutenção.

A infraestrutura e a confiabilidade do sistema são sustentadas pelo Docker e pelo Redis. O Docker encapsula a aplicação e suas dependências (incluindo as bibliotecas do sistema necessárias para o Chrome) em contêineres, garantindo consistência entre os ambientes de desenvolvimento e produção. Já o Redis desempenha um papel crucial no controle de concorrência, atuando como um armazenamento de chave-valor em memória para gerenciar locks distribuídos, evitando condições de corrida em requisições paralelas. Por fim, a documentação da API é exposta automaticamente via Swagger (OpenAPI), permitindo que desenvolvedores testem e integrem o serviço de forma intuitiva.

4. Padrões de Resiliência: Robustez e Otimização

Dada a complexidade envolvida na comunicação com um navegador headless, a resiliência foi um foco central da implementação.

Retry Pattern (Padrão de Tentativa): Operações de renderização podem falhar devido a instabilidades transitórias, como timeouts de rede ou sobrecarga momentânea da thread do navegador. Para mitigar isso, implementamos o padrão de Retry utilizando a biblioteca async-retry. Conforme observado no módulo pdfGenerator.js, o sistema encapsula a lógica de geração do PDF em um bloco de tentativas. Caso ocorra uma falha na renderização, o serviço aguarda um breve intervalo e tenta novamente (configurável via MAX_RETRIES), aumentando significativamente a taxa de sucesso sem a necessidade de intervenção do usuário final.

Concurrency Control e Idempotência (Rate Limiter Adaptado): Para proteger o servidor de desperdício de recursos computacionais, implementamos um controle de concorrência baseado em Distributed Locks com Redis. No controller.js, cada requisição recebe uma assinatura única (hash SHA256) gerada a partir do seu conteúdo (template e dados). Antes de iniciar o processamento, o sistema verifica no Redis se já existe um processamento em andamento para aquele hash exato. Se detectado, a requisição duplicada é bloqueada imediatamente (retornando status 429), impedindo que o servidor renderize o mesmo documento duas vezes simultaneamente. Isso atua como um rate limiter inteligente, focado na idempotência e na economia de CPU.

5. Avaliação da Reusabilidade

A principal força deste micro serviço reside na sua capacidade de atender a domínios de negócio completamente distintos sem alterações no código-fonte, apenas através da injeção de novos templates e dados. A arquitetura suporta uma variabilidade ampla, demonstrada nos seguintes cenários exemplificados:

- **Cenário 1: Acadêmico (Certificados)**

Utilizando o template certificado.hbs, o sistema processa dados de alunos, cursos, cargas horárias e datas. A lógica de layout se adapta para preencher os campos variáveis, mantendo a formatação oficial da instituição e garantindo a autenticidade visual do documento.

- **Cenário 2: Financeiro (Notas Fiscais)**

No contexto financeiro, o template nota_fiscal.hbs demonstra a capacidade do sistema de lidar com dados tabulares complexos e elementos gráficos. O serviço integra a geração de QR Codes (para pagamentos via Pix, por exemplo) e cálculos de totais, formatando uma fatura profissional pronta para envio ao cliente.

- **Cenário 3: Administrativo (Contratos)**

Para o setor jurídico e administrativo, o template contrato.hbs permite a geração dinâmica de documentos legais. Cláusulas, nomes das partes, valores e condições são injetados em um texto corrido padronizado, automatizando a burocracia de preenchimento manual e reduzindo erros humanos.

- **Cenário 4: Construtor Dinâmico (Builder)**

Além dos modelos fixos, o sistema conta com um template builder.hbs e um arquivo de configuração JSON associado. Este cenário representa o nível máximo de reuso, onde o próprio cliente define a estrutura do documento (títulos, parágrafos, imagens) via JSON, e o serviço monta o PDF dinamicamente, funcionando como um gerador de relatórios ad-hoc.

6. Conclusão

A implementação deste microserviço de geração de PDFs atingiu os objetivos de desacoplamento e robustez esperados. Ao centralizar a lógica de renderização em uma aplicação *stateless* e containerizada, garantimos que a escalabilidade do serviço possa ser gerenciada independentemente das aplicações principais. A padronização visual foi assegurada pelo uso de componentes reutilizáveis (partials) no Handlebars, eliminando a duplicação de código HTML/CSS.

Como lições aprendidas, destacamos a complexidade de gerenciar recursos em processos *headless* como o Puppeteer. O consumo de memória e CPU pode ser elevado, tornando vital a implementação de estratégias de *lock* distribuído (via Redis) e o padrão de *Retry*. Estas salvaguardas impedem que gargalos ou falhas pontuais comprometam a disponibilidade do sistema. Conclui-se que a extração dessa responsabilidade para um serviço dedicado, protegido

por padrões de resiliência, é a arquitetura ideal para garantir performance e manutenibilidade a longo prazo em sistemas que dependem de documentação estática.

A abordagem de microserviço para tarefas computacionalmente pesadas, como a geração de PDFs, mostrou-se ideal. Conseguimos desacoplar essa responsabilidade dos sistemas principais, assegurando que a falha ou a lentidão na geração de um PDF não comprometa a operação do sistema como um todo. O uso de padrões como Retry e Partials garantiu a robustez e a manutenibilidade exigidas por um serviço moderno.

7. Referências

- **Slides da disciplina.**
- **Express Documentation.** Disponível em: https://expressjs.com/
- **Puppeteer: Headless Chrome Node.js API.** Disponível em: https://pptr.dev/
- **Handlebars.js Documentation.** Disponível em: https://handlebarsjs.com/
- **Redis: The Open Source, In-memory Data Store.** Disponível em: https://redis.io/
- **Microservices Patterns: With examples in Java.** Richardson, C. Manning Publications, 2018.