**Arthur do Nascimento Sita Gomes**
arthur.gomes.1388895@sga.pucminas.br

# Documento de Visão para o Sistema Swagger to SDK

**17 de setembro de 2025**

*Proposta do aluno Arthur do Nascimento Sita Gomes ao curso de Engenharia de Software como projeto de Trabalho de Conclusão de Curso (TCC) sob orientação de conteúdo do professor Cleiton Silva Tavares e orientação acadêmica do professor Leonardo Vlela Cardoso.*

---

## OBJETIVOS

Automatizar a geração de clientes de acesso a serviços baseados no padrão de Transferência de Estado Representacional (*REST*, do inglês Representational State Transfer) para aplicações de interface com o usuário, utilizando o *TypeScript* como linguagem principal e tomando como fonte de verdade, especificações em *Swagger/OpenAPI*. Baseada em uma interface gráfica dentro do Visual Studio Code (*VS Code*) ou diretamente em arquivos de especificação no formato JavaScript Object Notation (*JSON*, do inglês *JavaScript Object Notation*) ou *YAML* (YAML, do inglês *YAML Ain't Markup Language*), sem a necessidade de escrever código-fonte manual para definir as funções de consumo de *API*. Sendo necessário apenas informar a localização da especificação, parâmetros de configuração e preferências de organização para que um conjunto de tipos e funções seja gerado e integrado a projetos cliente, executados em navegadores ou ambientes com suporte ao Ecossistema *TypeScript* (*Node.js*), atendendo às necessidades de aplicações *front-end* que utilizarão o cliente gerado como principal meio de comunicação com serviços de *back-end*.

## ESCOPO

O sistema será uma ferramenta de apoio na forma de extensão do Editor de Código Visual (*VS Code*, do inglês *Visual Studio Code*) para a construção de uma Interface de Programação de Aplicação (*API*, do inglês *Application Programming Interface*) cliente que utiliza por padrão o Protocolo de Transferência de Hipertexto (*HTTP*, do inglês *Hypertext Transfer Protocol*) seguindo o estilo *REST*, realizando a geração de tipos e funções fortemente tipadas a partir de outras

aplicações ou microsserviços descritos por especificações *Swagger/OpenAPI*. Por meio apenas de especificações será possível definir o seu funcionamento diretamente no arquivo *OpenAPI* (*JSON* ou *YAML*) ou utilizando a interface de usuário do sistema. As especificações irão habilitar a criação de módulos *TypeScript* com funções que recebem parâmetros de caminho, consulta e cabeçalhos, além de corpo de requisição quando aplicável, e irão mapeá-los para chamadas *HTTP* padronizadas com `fetch`, respeitando os caminhos, métodos e esquemas definidos na especificação.

O arquivo *OpenAPI* funcionará, basicamente, como um arquivo de configuração. As informações escritas diretamente nele servirão apenas para contextualização. Todas as funcionalidades do gerador estarão implementadas de forma genérica, possibilitando que ele produza tipos e funções correspondentes às operações descritas, desde que sejam especificados os detalhes dessas operações. Para isso, no arquivo serão definidas informações sobre os servidores disponíveis, os caminhos e métodos expostos, as estruturas de dados de entrada e saída, seus tipos, obrigatoriedades e descrições, incluindo códigos de resposta mais comuns (200/201/204) e identificadores de operação. Na ausência de identificadores explícitos, nomes de funções serão sintetizados a partir de método e caminho, mantendo a rastreabilidade entre o contrato e o código gerado.

Além do próprio diferencial de centralizar e padronizar o consumo de serviços, a geração tipada potencializa, ainda mais, a segurança e a produtividade do desenvolvimento, haja vista que a alternativa direta mais comum, conhecida como escrita manual de *Software Development Kits* (SDKs) ou uso de bibliotecas genéricas fora do contexto do editor, exige manutenção recorrente e não oferece, por padrão, a mesma integração com o fluxo de trabalho do desenvolvedor. Para ambos os casos, a adoção de tipos estáticos pode reduzir erros em tempo de execução e melhorar a legibilidade e a consistência do código, pois promove a validação em tempo de compilação e expõe um único ponto de acesso organizado para as operações disponíveis.

Para auxiliar na construção de clientes de *API*s baseadas em *OpenAPI*, já existem bibliotecas e geradores dedicados no ecossistema *TypeScript*, contudo são focados no uso diretamente por linha de comando ou integração manual ao projeto, assim como não possuem uma estrutura base orientada ao editor. São soluções genéricas que permitem a criação de qualquer cliente, onde a organização dos módulos, a configuração de ambientes e a inserção no fluxo de trabalho do desenvolvedor precisam ser construídas do zero, assim como qualquer eventual alteração.

Atualmente existem extensões para criação de SDKs no ambiente do VS Code, porém, nenhuma tem como foco o ecossistema *javacript/typescript* e uma integração com o fluxo de autenticação utilizado pelo desenvolvedor.

Em suma, o propósito geral desse projeto é ter uma ferramenta geradora de clientes *REST* tipados para *front-end*. Será possível que isso seja feito de forma visual por uma interface gráfica dentro do *VS Code* ou apontando diretamente para um arquivo *OpenAPI* dentro do diretório da aplicação. Apenas com essas especificações e com o código base já fornecido pela ferramenta, onde todas as funcionalidades de geração estarão implementadas de forma genérica, será possível produzir módulos que realizem as requisições especificadas de forma consistente. Por fim, o código gerado poderá ser utilizado no próprio ambiente de desenvolvimento ou incorporado a projetos clientes que serão executados em navegadores e implantados em servidores de aplicações *web* em sistemas operacionais *Windows*, *Linux* ou *macOS*, desde que haja suporte ao Ecossistema *TypeScript*.

## FORA DO ESCOPO

Este projeto não contempla protocolos e estilos de integração não-HTTP (*GraphQL*, *gRPC*, *SOAP*, *WebSockets*, *Server-Sent Events* ou mensageria via *TCP*). Não inclui a geração de servidores, *API Gateways*, *BFFs* ou qualquer camada de orquestração do lado do servidor. A aquisição de credenciais (fluxos *OAuth2/OIDC*, *PKCE*, *mTLS* e *SSO*) não será implementada; a ferramenta apenas permite injetar *tokens* já obtidos pela aplicação cliente. Fontes de especificação fora do *OpenAPI/Swagger* (*Postman Collections*, *RAML*, *WSDL* ou inferência por tráfego) não farão parte da primeira versão. Não haverá geração de *SDKs* para outras linguagens além de *TypeScript*, nem criação automática de componentes de *UI* (UI, do inglês User Interface), páginas, estados de aplicação ou *dashboards*. Publicação de pacotes no *npm*, configuração de pipelines de *CI/CD* (CI/CD, do inglês Continuous Integration/Continuous Development) e monitoramento/telemetria de execução não estão incluídos. Recursos avançados do *OpenAPI* como *callbacks*, *links* e discriminator terão suporte parcial ou posterior. Operações com *streaming* de alto volume (*upload/download*) não serão tratadas na fase inicial.

## GESTORES, USUÁRIOS E OUTROS INTERESSADOS

| | |
|---|---|
| **Nome** | Arthur do Nascimento Sita Gomes |
| **Qualificação** | Estudante/Desenvolvedor |
| **Responsabilidades** | Responsável pelo desenvolvimento do projeto ao longo do Trabalho de Conclusão do curso |

| | |
|---|---|
| **Tipo de usuário** | Pessoa desenvolvedora de *front-end* |

| | |
|---|---|
| **Como podera usar o sistema proposto** | Como poderá usar o sistema proposto Importar uma especificação *OpenAPI* local ou por *URL*, gerar um cliente *REST* tipado e consumi-lo diretamente em aplicações *web* com redução de código repetido e validação estática. |

## LEVANTAMENTO DE NECESSIDADES

1. Reduzir o esforço repetitivo de integração *REST* em aplicações *front-end*, mantendo paridade com contratos *OpenAPI*.
2. Permitir dentro do *VS Code* importar, validar e gerar clientes com poucos passos, inclusive modo assistir a alterações da especificação.
3. Oferecer tipagem estática de parâmetros e *payloads* (*path, query, headers, cookies e body*).
4. Permitir organização opinativa, porém configurável, da saída e conexão com o servidor (*URL base,* pastas, nomes de módulos).
5. Garantir compatibilidade de execução em navegadores e em Node.js por meio da *API* nativa *fetch*.
6. Expor *feedbacks* compreensíveis de erros de modelagem *OpenAPI* para antecipar correções.

## FUNCIONALIDADES DO PRODUTO

| **Necessidade:** Reduzir o esforço repetitivo de integração *REST* em aplicações *front-end*, mantendo paridade com contratos *OpenAPI*. | |
|---|---|
| **Funcionalidade** | **Categoria** |
| 1. Importar e validar especificações *OpenAPI* (arquivo local/*URL*) | Crítico |
| 2. Gerar tipos *TypeScript* para *schemas*, parâmetros e respostas | Crítico |
| 3. Gerar funções por operação com assinatura tipada completa | Crítico |
| 4. Nomear funções por *operationId* ou por método+caminho | Importante |

| **Necessidade:** Permitir dentro do *VS Code* importar, validar e gerar clientes com poucos |
|---|

| passos, inclusive modo assistir a alterações da especificação. | |
|---|---|
| **Funcionalidade** | **Categoria** |
| 1. Comandos na Command Palette para importar, validar e gerar dos *templates* | Crítico |
| 2. Retorno de erros com sugestões de correção quando possível | Importante |

| **Necessidade:** Oferecer tipagem estática de parâmetros e *payloads* (*path, query, headers, cookies e body*). | |
|---|---|
| **Funcionalidade** | **Categoria** |
| 1. Mapeamento de *path*, *query*, *headers*, *cookie* e *requestBody* | Crítico |
| 2. Tipos por código de resposta (200/201/204 e configuráveis) | Crítico |
| 3. Geração de utilitários de verificação em tempo de execução | Útil |

| **Necessidade:** Permitir organização opinativa, porém configurável, da saída e conexão com o servidor (*URL base,* pastas, nomes de módulos). | |
|---|---|
| **Funcionalidade** | **Categoria** |
| 1. Fornecer *templates* de conexão com o servidor (*adapters*) | Crítico |
| 2. Definir *url* base para todas as requisições | Importante |
| 3. Definir diretórios de saída e nomes de módulos | Importante |
| 4. Integração com *Prettier/ESLint* do projeto | Útil |

| **Necessidade:** Garantir compatibilidade de execução em navegadores e em Node.js por meio da *API* nativa *fetch*. | |
|---|---|
| **Funcionalidade** | **Categoria** |
| 1. Orientação para uso da API nativa *fetch* no *Node* | Crítico |

| **Necessidade:** Expor *feedbacks* compreensíveis de erros de modelagem *OpenAPI* para antecipar correções. |
|---|

| Funcionalidade | Categoria |
|---|---|
| 1. Retornar mensagens de erro ao importar documentação *OpenAPI* | Importante |

## INTERLIGAÇÃO COM OUTROS SISTEMAS

A extensão integra-se apenas com o próprio *VS Code* por meio de suas *APIs* de interface, arquivos e *workspace*, sem dependências em serviços externos. As especificações são lidas de arquivos locais do projeto ou de *URLs* públicas; referências remotas autenticadas não fazem parte da primeira versão. O código gerado destina-se a projetos *TypeScript* que utilizem *bundlers* como *Vite* ou *Webpack*, compatível com *TypeScript* 5+ e *Node* 18+. No navegador e Node utiliza-se *fetch* nativo. Opcionalmente a geração respeita formatação e regras já existentes no projeto por meio de *Prettier* e *ESLint*.

## RESTRIÇÕES

- A cobertura do *OpenAPI* prioriza caminhos, métodos, parâmetros, *requestBody*, *responses* e *schemas*; recursos como *callbacks*, *links* e *discriminator* não terão suporte.
- Políticas de *CORS* e restrições do servidor não são contornadas pela geração do cliente.
- O ambiente mínimo requerido inclui *VS Code* em versão compatível, *Node.js* 18 ou superior e *TypeScript* 5 ou superior.
- Especificações muito extensas ou com cadeias profundas de referências podem impactar desempenho de validação e geração.
- Operações com *streaming* de alto volume não estão contempladas na versão inicial.
- A saída é exclusivamente em *TypeScript* nesta fase.

## DOCUMENTAÇÃO

Será elaborado um manual de uso orientado à interface do *VS Code*, cobrindo requisitos, instalação, importação por arquivo/*URL*, validação, geração, atualização e configuração de saída. Será produzido um guia de integração com exemplos de consumo no navegador e no Node, incluindo orientações de polyfill de *fetch*, upload multipart e tratamento de erros. Haverá uma referência de configuração detalhando opções de geração (pastas, nomenclatura, *templates*, mapeamento de *status*), bem como uma matriz de compatibilidade *OpenAPI* com recomendações de modelagem para melhor tipagem. A documentação técnica interna descreverá arquitetura e módulos (*parser*, *validador*, gerador, *templates*, escritor), além do fluxo importação➜validação➜planejamento➜geração➜formatação e decisões de projeto. Um repositório de exemplos disponibilizará especificações de diferentes portes e a respectiva saída

gerada. Por fim, haverá registro de versões com critérios de *breaking change* e uma seção de perguntas frequentes com soluções para erros comuns de modelagem.