INTERFACE TECNOLÓGICA

UTILIZAÇÃO DA FERRAMENTA ZOD COMO MEIO PARA VALIDAÇÃO DA ENTRADA DE DADOS EM UMA FUNÇÃO

USING THE ZOD TOOL TO VALIDATE DATA INPUT INTO A FUNCTION

Murilo Luiz Silva de Sousa – contato@silvamurilo.com.br
Faculdade de Tecnologia de Taquaritinga (Fatec) – Taquaritinga

Maurício de Oliveira Dian – mauricio.dian@fatec.sp.gov.br
Faculdade de Tecnologia de Taquaritinga (Fatec) – Taquaritinga

DOI: 10.31510/infa.v21i2.2037
Data de submissão: 23/09/2024
Data do aceite: 23/11/2024
Data da publicação: 20/12/2024

RESUMO

Este artigo explora a utilização da biblioteca Zod para a validação de esquemas em Typescript, demonstrando como essa técnica pode melhorar a entrada de dados em funções, garantindo não só a segurança, mas também a correção dos dados. Utilizando o método *parse* de Zod, o estudo não apenas verifica a conformidade dos dados com os esquemas predefinidos, mas também efetua a conversão dos dados para o formato adequado. Este processo é fundamental para manter a integridade dos dados e evitar erros comuns associados à manipulação de dados de entrada. A metodologia adotada envolveu uma análise detalhada de casos de uso práticos onde Zod foi implementado, proporcionando um entendimento claro de como a validação de esquemas pode contribuir para práticas de desenvolvimento mais seguras e eficientes. O artigo justifica a escolha do tema pela crescente necessidade de aplicações mais robustas e confiáveis, onde erros de dados podem ter consequências graves. O objetivo deste estudo é destacar a importância da validação de dados na melhoria da qualidade e confiabilidade das aplicações, evidenciando como Zod pode ser uma ferramenta valiosa para desenvolvedores que buscam implementar essas práticas rigorosas.

Palavras-chave: Typescript. Zod. Validação. Manipulação de dados.

ABSTRACT

This article explores the use of the Zod library for schema validation in Typescript, demonstrating how this technique can improve data entry into functions, guaranteeing not only the security but also the correctness of the data. Using Zod's parse method, the study not only checks that the data conforms to the predefined schemas, but also converts the data into the

appropriate format. This process is fundamental to maintaining data integrity and avoiding common errors associated with the manipulation of input data. The methodology adopted involved a detailed analysis of practical use cases where Zod has been implemented, providing a clear understanding of how schema validation can contribute to safer and more efficient development practices. The article justifies the choice of topic by the growing need for more robust and reliable applications, where data errors can have serious consequences. The aim of this study is to highlight the importance of data validation in improving the quality and reliability of applications, highlighting how Zod can be a valuable tool for developers looking to implement these rigorous practices.

Keywords: Typescript. Zod. Validation. Data manipulation.

1. INTRODUÇÃO

Zod é uma biblioteca de validação de esquemas em Typescript que simplifica a garantia de que os dados de entrada e saída nas aplicações estejam corretos e seguros. Ao definir esquemas de validação, ela ajuda a prevenir problemas como dados malformados, entradas inesperadas e vulnerabilidades de segurança, facilitando a manutenção da integridade dos dados (MCDONNELL, 2019).

Segundo Pocock (2020), Zod é uma ferramenta essencial para a validação de dados em aplicações, garantindo que as entradas desconhecidas ou não totalmente confiáveis sejam seguras e corretas antes de serem processadas. Ao utilizar Zod, os desenvolvedores podem evitar erros e prevenir vulnerabilidades, proporcionando uma camada adicional de segurança e robustez, especialmente em situações em que os dados vêm de fontes externas ou são manipuláveis pelos usuários, como em formulários ou APIs de terceiros. Ainda segundo Pocock (2020), este recurso de validação ajuda a identificar problemas potenciais no início, facilitando a depuração e manutenção do código, e mantendo a integridade da aplicação mesmo diante de mudanças inesperadas nas entradas de dados.

De acordo com Hall (2022), um *parser* é um componente de software que lê e analisa texto ou dados seguindo regras específicas, convertendo-os em uma estrutura organizada para processamento ou análise posterior. É fundamental em tarefas que requerem a interpretação de dados complexos, como linguagens de programação e formatos de arquivo, facilitando sua manipulação por outros componentes do software.

O foco deste trabalho é demonstrar a eficácia do Zod na validação de dados, destacando especialmente o uso de seu método *parse*. Este método não apenas verifica a conformidade dos dados com o esquema definido, mas também permite a conversão automática desses para o formato adequado.

Este artigo também se propõe a explorar aspectos essenciais e práticos da implementação dessa funcionalidade. Segundo Laso (2023), esta capacidade revela-se fundamental em validações de formulário, onde o *parse* assegura que todas as entradas do usuário sejam rigorosamente avaliadas contra os requisitos específicos antes de qualquer processamento adicional.

A metodologia adotada neste estudo envolveu a implementação prática da biblioteca Zod em vários cenários de entrada de dados, permitindo uma análise detalhada de sua funcionalidade e eficiência. Optou-se por este tema devido à crescente necessidade de aplicações mais seguras e confiáveis, onde a integridade dos dados é crucial e o objetivo foi avaliar até que ponto o Zod pode contribuir para a robustez e segurança na manipulação de dados, demonstrando, através de casos práticos, como essa ferramenta se adapta e responde às exigências de validação de dados complexos.

2. FUNDAMENTAÇÃO TEÓRICA

A seguir serão abordados conceitos chave e exemplos de uso, destacando o *parse* na garantia de integridade e na correção dos dados em aplicações Typescript.

2.1 O que é Zod

Bachovas (2023), afirma que o Zod facilita a verificação e a tipagem de dados recebidos, o que simplifica e permite a definição de esquemas de objetos e a validação de dados em conformidade, em tempo de execução. Isso contribui para a integridade dos dados, assegurando que estejam alinhados com critérios de validação estipulados. Destaca ainda alguns pontos, como:

- A habilidade de estabelecer esquemas de objetos durante a execução do programa.

- A capacidade de validar dados inseridos com base nesses esquemas.
- Suporte para a criação de esquemas complexos e definição de tipos personalizados.
- Sua API de fácil manuseio para a configuração e validação de esquemas.

De acordo com McDonnell (2019), a ferramenta suporta vários tipos especiais que aumentam sua flexibilidade. Os *unums* permitem definir um conjunto fixo de valores constantes. *unions* e *intersections*, similares às estruturas do Typescript, facilitam a combinação de diferentes tipos de maneira flexível. Tipos como *any*, *unknown* e *never* são utilizados para diferentes situações, da aceitação total (*any)* até a restrição total (*never)*.

Segundo Colandrea (2023), o Zod se destaca em projetos Typescript por sua capacidade de definir e validar estruturas de dados com precisão. Este duplo mecanismo aumenta a segurança, prevenindo as inconsistências de dados e erros comuns. Colandrea (2023), ainda destaca, com uma API clara e poderosa, Zod permite aos desenvolvedores criarem esquemas detalhados com minimalismo de código, tirando vantagem da inferência de tipos para assegurar a conformidade dos dados com os requisitos estabelecidos, reduzindo riscos e falhas nas aplicações.

Esquemas são ferramentas úteis para garantir que os valores utilizados em um sistema atendam a critérios específicos de validação, o que pode ser crucial em muitos cenários de desenvolvimento de software (LASO, 2023).

De acordo com Osayerie (2023), esquemas podem ser utilizados sozinhos ou como parte de esquemas mais complexos, permitindo a construção de validações detalhadas e específicas para os dados da sua aplicação. Essa abordagem ajuda a garantir que os dados não apenas existam, mas também estejam no formato correto e dentro dos parâmetros esperados.

Uma forma de criar um esquema para validar cadeias de caracteres é usar o método *z.string()* de forma simples, como:

```
const passwordSchema = z.string().min(8).max(64);
```

O primeiro esquema é focado em senhas, impõe limites de comprimento entre 8 e 64 caracteres, fortalecendo a segurança e exemplificando como a ferramenta ajuda a otimizar o desenvolvimento, assegurando a integridade e adequação dos dados nas aplicações.

Outro caso é a utilização do método *z.number()*. Este é usado para validar números e suporta a adição de restrições, como valores mínimo e máximo.

const minMaxNumberSchema: z.number().min(0).max(100);

O esquema anterior valida se o valor inserido é um número válido, sem impor restrições adicionais, ideal para contextos em que qualquer número é aceitável.

2.2 Funcionalidades da ferramenta

Segundo Hall (2022), a utilização do *parser* da biblioteca Zod representa uma abordagem eficaz para a validação e transformação de dados de acordo com esquemas previamente definidos. Este método garante que tanto os dados de entrada quanto de saída estejam alinhados às especificações técnicas exigidas, além de identificar e reportar erros sempre que os dados desviarem das normas estabelecidas.

Ao definir um esquema em Zod, são detalhados a estrutura e os tipos de dados esperados, o que permite ao *parser* avaliar a conformidade dos dados antes de realizar quaisquer operações críticas ou responder a requisições. Esta prática é essencial para assegurar a segurança, consistência e confiabilidade nas interações entre APIs e clientes, sendo um pilar para o desenvolvimento de interfaces de programação robustas e protegidas (LASO, 2023).

Na sequência, será explorado como o Zod pode ser empregado para aprimorar a validação de dados na entrada de funções, destacando o papel crucial do seu mecanismo de *parse*.

3. PROCEDIMENTOS METODOLÓGICOS

O método adotado neste trabalho foi a revisão bibliográfica através de uma análise de artigos teóricos e práticos recentes e documentação oficial da biblioteca. Com esta metodologia, não somente aspectos teóricos de validação de dados, mas também a implementação prática desses conceitos foi utilizada para ilustrar a funcionalidade e a importância do Zod no processo

de validação. Através desta abordagem prática, foi possível não só interpretar os dados com clareza, mas também verificar sua veracidade e a aplicabilidade das técnicas discutidas, proporcionando uma compreensão mais profunda e embasada sobre como Zod opera e se integra às práticas de desenvolvimento de software, garantindo a consistência e a segurança dos dados manipulados.

Essa pesquisa demonstra-se de cunho qualitativo, uma vez que busca explorar o tema com a finalidade de trazer a relevância e a importância do uso de ferramentas de validações de entrada de dados em sistemas e interfaces de programação.

4. RESULTADOS E DISCUSSÃO

Nessa sessão será criado uma função que requisita o *endpoint* de dados meteorológicos atuais (disponível em: *https://openweathermap.org/current*). Para isso são necessários 3 parâmetros: latitude, longitude e a chave de autenticação da API (*API Key*).

A *API Key* é acessada pelo "*.env",* por motivos de segurança, ele não será exposto. Então a nossa função irá precisar apenas de latitude e longitude. Na figura 1 é possível acompanhar o *endpoint* da API e os dados que iremos precisar.

Figura 1 – Dados necessários para a requisição

```
/*
lat     → Latitude
lon     → Longitude
appid   → API Key
https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}
*/
```

Fonte: Próprio Autor (2024)

A Figura 2, ilustra o esquema que será utilizado na função que irá requisitar a API. Esse esquema exige um objeto com dois pares do tipo "chave: valor", indicados como latitude e longitude.

Figura 2 – Esquema da Função

```
15  const getWeatherSchema = z.object({
16    lat: z.number(),
17    lon: z.number(),
18  });
```

Fonte: Próprio Autor (2024)

A Figura 3, ilustra um exemplo de *payload* nos padrões do esquema.

Figura 3 – Payload de Acordo com Esquema

```
51  const payload = {
52    lat: 40.7128,
53    lon: -74.006,
54  };
```

Fonte: Próprio Autor (2024)

Na Figura 4, pode-se ver que o Zod infere o tipo do esquema, isso faz com que a entrada da função seja tipada (com garantias de tipo) de acordo com o esquema. A função em si, utiliza o *payload* enviado e faz a requisição a API.

Figura 4 - Função Usada para Requisição

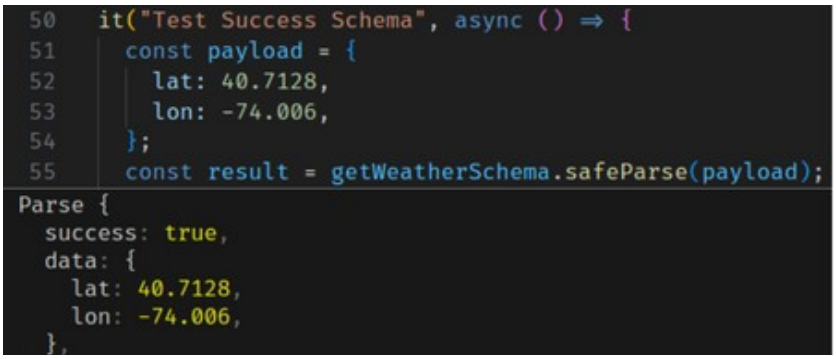
```
33  const service = (headers?: HeadersInit) ⇒
34    makeService("https://api.openweathermap.org/data/2.5", {
35      headers,
36    });
37
38  async function getWeather(payload: z.infer<typeof getWeatherSchema>) {
39    const apiKey = process.env.API_KEY_WEATHER;
40    const lat = payload.lat;
41    const lon = payload.lon;
42    const response = await service({}).post(
43      `/weather?lat=${lat}&lon=${lon}&appid=${apiKey}`
44    );
45    return await response.json();
46  };
```

Fonte: Próprio Autor (2024)

Sabendo o que é necessário, basta utilizar o *payload* e verificar com o Zod.

Na Figura 5 a variável *payload* é um objeto com dois pares do tipo “chave: valor” com números. Na linha 55, a constante *result* está fazendo o *parse* dos dados. No terminal da IDE (*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado) utilizada é possível ver o resultado desse processo.

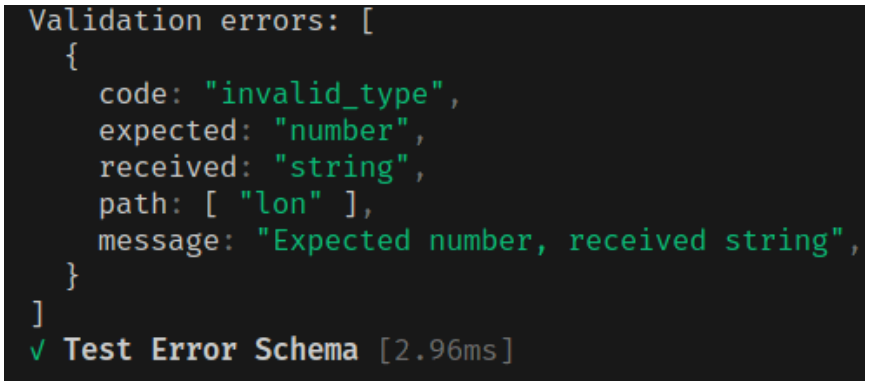
Figura 5 – Resultado do *parser* do payload



Fonte: Próprio Autor (2024)

A Figura 6 ilustra como é a execução esperada sem *parsear* o *payload* com sucesso. Veja que o valor recebido não está nos padrões que o esquema exigia.

Figura 6 – Zod tratando input com erro



Fonte: Próprio Autor (2024)

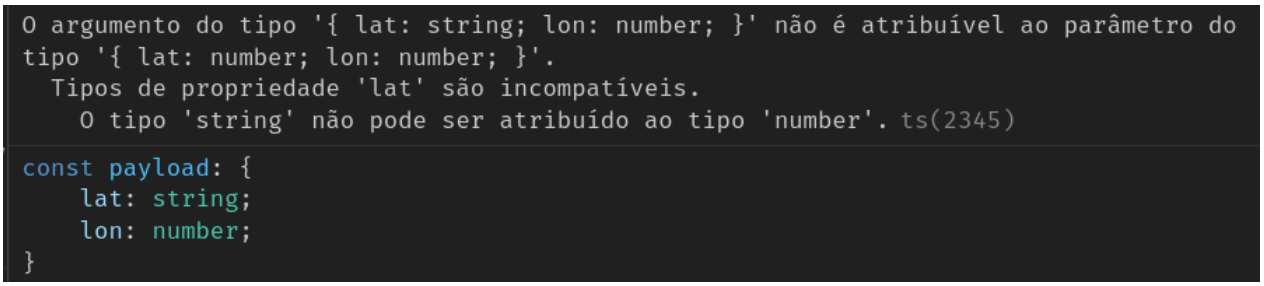
O teste esperava um *resultado falso,* pois o valor passado em longitude estava como *string* e não *number*. No console do terminal, é possível visualizar o retorno da ferramenta indicando que o esperado deveria ser um valor numérico e não uma string.

Cavalcante (2019) discute a Experiência do Desenvolvedor (DX), destacando sua importância para proporcionar uma boa interação dos desenvolvedores com produtos digitais. Similar à Experiência do Usuário (UX – *User Experience),* a DX foca nas necessidades específicas dos desenvolvedores, abrangendo aspectos como confiança, clareza e facilidade de uso. Ele ressalta que a qualidade da documentação e o suporte da comunidade são essenciais para uma DX eficaz.

A falta de uma boa DX pode resultar na rejeição de produtos, enquanto produtos com alta DX são frequentemente recomendados e mais utilizados, conforme exemplificado por empresas como Stripe, Atlassian e Firebase. Uma DX bem planejada pode significativamente aumentar a satisfação e a eficiência dos desenvolvedores, demonstrando a crescente valorização da DX no desenvolvimento de software (CAVALCANTE, 2019).

O Zod auxilia quanto ao DX. A Figura 7 ilustra a ferramenta alertando ao desenvolvedor que aquele *payload* não está com os tipos corretos de acordo com o esquema, uma vez que era esperado um objeto com duas *strings* e na realidade está recebendo um *number* e uma *string*.

Figura 7 – Zod apontando erro no payload antes de executar



Fonte: Próprio Autor (2024)

De acordo com Gado (2024), a validação de dados é um aspecto crítico no desenvolvimento de software, especialmente em aplicações que dependem da precisão e integridade dos dados para operar corretamente. A biblioteca Zod emerge como uma solução

poderosa e flexível para este desafio, oferecendo uma abordagem declarativa e tipo-segura para definir esquemas de validação que se integram perfeitamente ao ecossistema Typescript.

Seu método *parse* simplifica a validação de dados, fornecendo um mecanismo robusto para assegurar que os dados de entrada e saída estejam alinhados com as expectativas do desenvolvedor. Através dos exemplos práticos apresentados, fica evidente que o Zod não apenas facilita a implementação de validações de dados complexas, mas também contribui para a manutenção da consistência, segurança e confiabilidade das aplicações (RIOS, 2024).

Os desenvolvedores poderão adotar o Zod como parte de suas práticas de validação de dados para melhorar a qualidade e a confiabilidade de suas aplicações Typescript.

5. CONSIDERAÇÕES FINAIS

Conclui-se que o Zod se demonstra como uma ferramenta poderosa e eficaz na validação de dados de entrada em funções. Ela evidencia e reforça a confiabilidade e a eficiência no desenvolvimento. Ela não apenas simplifica o processo de validação através de esquemas declarativos e robustos, mas também contribui para a redução de erros e vulnerabilidades comuns associadas à manipulação de dados. Ao automatizar a validação e fornecer feedbacks detalhados sobre os erros, ela facilita a detecção e correção precoce de problemas, melhorando a qualidade do código e a experiência do usuário final.

Percebe-se que a integração do Zod em aplicações que consomem ou fornecem APIs eleva o padrão de desenvolvimento, promovendo práticas de codificação mais confiáveis e um código mais limpo e manutenível. Isso, por sua vez, se traduz em uma redução no tempo de desenvolvimento e nos custos associados à manutenção de software, além de reforçar a proteção contra-ataques mal-intencionados. Em suma, esses fatores justificam o porquê muitos desenvolvedores acabam utilizando tal ferramenta para avançar significativamente nas práticas de validação de dados contemporâneos. É uma ferramenta simples, eficiente e a capaz de lidar com validações complexas, tornando-se de grande valia para diversos tipos e níveis de programadores.

REFERÊNCIAS

BACHOVAS, R. G. **Validação de formulário com Zod e React Hook Form: Garantindo a integridade dos dados nos seus formulários.** Disponível em: <https://medium.com/@rbgadotti/validação-de-formulário-com-Zod-e-react-hook-form-garantindo-a-integridade-dos-dados-nos-seus-b1802aa329f1>. Acesso em: 10 fev 2024.

CAVALCANTE, A. **O que é DX (Developer Experience)**. 2019. Disponível em: <https://medium.com/@albertcavalcante/o-que-%C3%A9-dx-developer-experience-375f53eadede>. Acesso em: 25 mar 2024.

COLANDREA, D. **Master schema validation in Typescript with Zod.** 2023. Disponível em: <https://dev.to/_domenicocolandrea/master-schema-validation-in-typescript-with-zod-28dc>. Acesso em: 20 dez 2023.

GADO, W. **O que é zod?**. 2024. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-o-zod>. Acesso em: 05 mai 2024.

HALL, T. **Using Zod to Parse Function Schemas**. 2022. Disponível em: <https://blog.hyper.io/using-Zod-to-parse-function-schemas>. Acesso em: 09 fev 2024.

LASO, C. S. **A step-by-step guide to using Zod for backend data validation**. 2023. Disponível em: <https://eagerworks.com/blog/zod-for-backend-data-validation>. Acesso em: 15 mar 2024.

MCDONNELL, C. **Zod: Typescript-first schema validation with static type inference**. Disponível em: <https://zod.dev>. Acesso em: 29 mar 2024>.

OSAYERIE, A. **Schema Validation with Zod in 2023.** 2023. Disponível em: <https://www.turing.com/blog/data-integrity-through-Zod-validation>. Acesso em: 20 fev 2024.

POCOCK, M. **When should you use Zod?**. 2020. Disponível em: <https://www.totalTypescript.com/when-should-you-use-Zod>. Acesso em: 10 jan 2024.

RIOS, V. **Validação de Formulários com React-Hook-Form e Zod**. 2024. Disponível em: <https://dev.to/vitorrios1001/validacao-de-formularios-com-react-hook-form-e-zod-a6k>. Acesso em: 05 mai 2024.