

A DIDACTIC APPROACH TO BACK-END DATA VALIDATION IN WEB APPLICATION DEVELOPMENT

**Moisés Bastos¹, Gabriel Nazaré², Afonso Garcia³, Almir Kimura Junior¹,
& Silvio Romero Adjar Marques⁴**

¹*Department of Control and Automation Engineering, Amazonas State University (Brazil)*

²*Ocean Manaus, Amazonas State University (Brazil)*

³*Department of Computer Engineering, Amazonas State University (Brazil)*

⁴*Department of Production Engineering, Amazonas State University (Brazil)*

Abstract

Data validation is a fundamental aspect of web application development, directly impacting data integrity, security, and system reliability. In the education of information technology professionals, understanding validation strategies across both front-end and back-end layers is essential for building robust and scalable applications. This article presents a didactic approach based on a structured lesson focused on data validation, combining theoretical foundations with practical implementation. Initially, the concepts of data validation in front-end and back-end environments are discussed, highlighting their roles, differences, and complementary nature within the application architecture. Subsequently, the study explores back-end validation through two approaches: manual implementation of validation rules and the use of specialized libraries, such as Joi, which enable the definition of complex schemas and constraints in a structured and reusable manner. The proposed lesson emphasizes best practices for error handling, data consistency, and security, demonstrating how validation libraries contribute to the development of scalable, maintainable, and more secure systems. The results indicate that this pedagogical strategy facilitates students' understanding of validation mechanisms and strengthens their practical skills in back-end development, reinforcing the importance of combining conceptual knowledge with hands-on experience in software engineering education. This approach highlights the potential of project-based practice in back-end education and its contribution to the consolidation of technical skills in educational contexts within the Amazon region.

Keywords: *Back-end, data validation, Joi.*

1. Introduction

Backend data validation ensures that information received by an application is correct before being processed or stored. This practice helps prevent inconsistencies and improves system security by blocking malicious or invalid inputs. For this reason, server-side validation is considered an essential practice in modern web application development (OWASP, 2023).

Validation libraries such as Joi, widely used in the Node.js ecosystem, allow developers to define clear rules for data, including type, size, and required fields. This approach makes the code more organized and reduces common errors in APIs. Such tools are commonly adopted to standardize validation processes and improve code maintainability (Hapijs, 2024).

This article aims to present a didactic approach to teaching backend data validation through a structured lesson that combines theoretical concepts with practical implementation.

2. Methodology

The lesson begins with a theoretical explanation about data validation and its importance in software development. Its main benefits are presented, such as ensuring the integrity, security, and consistency of information before processing or storage. Next, front-end validation is briefly mentioned, with a simple practical example only for context, since this is not the main focus of the lesson. At this point, it is also emphasized that backend validation is not limited to database data, and that validating all data received from the front-end is a fundamental practice before any processing occurs.

To enable the implementation of data validation, it is necessary to create an application with database storage. In this context, the lesson activity proposes that students practically develop an application that stores information about superheroes, including fields such as name, year of creation, and universe (DC or Marvel). The database used is MongoDB, chosen for its easy integration with JavaScript applications. To facilitate the initial tests, three superheroes are already inserted into the database.

2.1. Manual data validation

Since the backend application has already been created, it is possible to begin the implementation of data validation. At this stage, validations are first implemented manually so that students can understand the underlying logic, and later they are implemented with the support of the Joi library. Several common validations are applied, such as checking that the name does not contain only blank spaces, ensuring that all fields contain some value, verifying that the year is numeric, greater than 1900, and formatted with four digits, as well as performing a database check to ensure that no superhero with the same name has already been stored. Figure 1 shows the implemented validations.

Figure 1. Code for data validation implemented without the use of libraries (manual).

```

44 | if (nome.length>0 && (nome.trim().length)==0)
45 |   return res.status(422).send("ERRO: Favor inserir dados corretos no nome do herói");
46 |
47 | //verifica se o nome, ano e universo vieram vazios
48 | if(!nome)
49 |   return res.status(422).send("ERRO: Dados incompletos no nome do herói");
50 | else if(!ano)
51 |   return res.status(422).send("ERRO: Dados incompletos no ano de criação do herói");
52 | else if(!universo)
53 |   return res.status(422).send("ERRO: Dados incompletos no universo do herói");
54 |
55 | //validação do ano
56 | if (isNaN(parseInt(ano)) || parseInt(ano) <= 1900 || ano.length !=4)
57 |   return res.status(422).send("ERRO: Favor inserir um ano válido no formato XXXX");
58 |
59 | const buscanome = await collection.findOne({
60 |   nome: nome,
61 | })
62 | //verifica se o nome já existe no banco de dados
63 | if (buscanome!=null){
64 |   return res.send("ERRO: O super-herói " + nome + " já está cadastrado!!");
65 | }

```

2.2. Data validation using Joi

At this stage of the lesson, students begin implementing data validation using the Joi library. The use of this library provides several advantages, such as making the code cleaner, more reusable, and easier to maintain, since validation rules are organized through schemas. In addition, validation libraries reduce the repetition of logic in the code and allow complex validation rules to be implemented with only a few lines, contributing to the development of more consistent and secure APIs (Pautasso; Zimmermann; Leymann, 2017).

Figure 2. Code for data validation manual versus Joi.

<pre> //Create -> [POST] /herois app.post("/herois", async function(req,res){ // extrai o nome, ano e universo do corpo da requisição const nome = req.body.nome; const ano = req.body.ano_criacao; const universo = req.body.universo; // verifica se o nome contém apenas espaços em branco if (nome.length>0 && (nome.trim().length)==0) return res.status(422).send("ERRO: Favor inserir dados corretos no nome do herói"); //verifica se o nome, ano e universo vieram vazios if(!nome) return res.status(422).send("ERRO: Dados incompletos no nome do herói"); else if(!ano) return res.status(422).send("ERRO: Dados incompletos no ano de criação do herói"); else if(!universo) return res.status(422).send("ERRO: Dados incompletos no universo do herói"); //validação do ano if (isNaN(parseInt(ano)) parseInt(ano) <= 1900 ano.length !=4) return res.status(422).send("ERRO: Favor inserir um ano válido no formato XXXX"); const buscanome = await collection.findOne({ nome: nome, }) </pre>	<pre> 44 // Schema Joi 45 const schema = Joi.object({ 46 nome: Joi.string().trim().min(1).required().messages({ 47 'string.empty': 'O nome é obrigatório', 48 'string.min': 'O nome não pode estar em branco' 49 }), 50 ano_criacao: Joi.number().integer().min(1900).required().messages({ 51 'number.base': 'Ano deve conter apenas números', 52 'number.min': 'Ano mínimo é 1900' 53 }), 54 universo: Joi.string().trim().required().messages({ 55 'string.empty': 'O universo é obrigatório' 56 }) 57 }); 58 59 // 1. Validar dados com Joi 60 const { error, value } = schema.validate(req.body, { abortEarly: false }); 61 62 if (error) { 63 const mensagens = error.details.map(e => e.message).join('\n'); 64 return res.status(422).send("ERRO(s): " + mensagens); 65 } </pre>
--	---

The Joi library allows the creation of schemas that define the expected structure and data types, such as strings, numbers, arrays, and objects. These schemas use a chained syntax that enables both simple and complex validations in an organized way. During the lesson, students create a schema containing the same validations previously implemented manually. Figure 2 presents a comparison between the manual code (highlighted on the left) and the schema developed in class using Joi (on the right). The validation used to check whether a superhero name already exists in the database, preventing duplicate records, is

shown in Figure 3. This implementation is performed inside a try...catch block to capture possible failures, such as database connection errors. This approach prevents the server from crashing and allows clearer error messages to be returned to the client. It also helps maintain a secure application flow in asynchronous operations.

Figure 3. Code for database validation using Joi.

```
--
67
68
69
70
71
72
73
74
75
76
77
78
try {
  // 2. Verificar duplicidade no banco (case-insensitive)
  const existente = await collection.findOne({
    nome: { $regex: `^${value.nome}$`, $options: 'i' }
  });
  if (existente) {
    return res.status(400).send(" ERRO: Herói já cadastrado ");
  }
} catch (err) {
  console.error(err);
  res.status(500).send(" ERRO: 'Erro no servidor' ");
}
```

3. Conclusion

The lesson presents, in a progressive way, the concepts and practices related to data validation in backend applications. Initially, the theoretical foundations of validation are discussed, highlighting their importance for ensuring the integrity, security, and consistency of information. Students then develop an application with a MongoDB database to register superheroes and implement manual validation, allowing them to understand the logic behind the process. Afterwards, validation is improved using the Joi library, enabling a comparison of approaches and demonstrating how specialized libraries can make code more organized, reusable, and easier to maintain.

The classes have been taught since 2025, allowing students to gain knowledge in data validation and receive certification at no cost. This initiative also highlights the importance of providing technological education to people in the Amazon region, who often face logistical challenges in accessing educational opportunities in major urban centers in Brazil. Figure 4 shows a class being conducted at the Ocean Institute in Manaus.

Figure 4. Class being taught at Ocean Institute.



Acknowledgements

We would like to express our gratitude to the Ocean Digital Fabrication Laboratory for providing the resources and facilities that made this project possible. Special thanks to our mentors, the coordinating professor, and all team members for their guidance, support, and collaborative efforts throughout the development of this work. Their contributions were invaluable to the success of this project. The Ocean Center Research and Development (R&D) project is funded by Samsung, using resources from the Western Amazon Informatics Law (Federal Law No. 8,387/1991), with its dissemination in line with Article 39 of Decree No. 10,521/2020.

References

- Hapijs. *Joi Validation Library Documentation*. (2024). Available at: <https://joi.dev>
- OWASP Foundation (2023). *OWASP Top Ten: The Ten Most Critical Web Application Security Risks*. Available at: <https://owasp.org>
- Pautasso, C., Zimmermann, O., & Leymann, F. (2017). RESTful Web Services vs. “Big” Web Services: Making the Right Architectural Decision. *Proceedings of the 17th International World Wide Web Conference (WWW)*. ACM.