ReservaFácil – Sistema de Reservas de Restaurante

Antônio Lisboa de Oliveira Neto
UFERSA
Pau dos Ferros, Brasil
antonio.neto62426@alunos.ufersa.edu.br

Darliany Elias Monte
Universidade Federal Rural do Semi-Árido
Pau dos Ferros, Brasil
darliany.monte@alunos.ufersa.edu.br

Débora Fernandes Costa
UFERSA
Pau dos Ferros, Brasil
debora.costa22396@alunos.ufersa.edu.br

Denylson de Freitas Feitoza
UFERSA
Pau dos Ferros, Brasil
denylson.feitoza@alunos.ufersa.edu.br

Francisco Alisson da Silva Queiroz
UFERSA
Pau dos Ferros, Brasil
francisco.queiroz10285@alunos.ufersa.edu.br

Jederson Yago Silva Rego
UFERSA
Pau dos Ferros, Brasil
jederson.rego@alunos.ufersa.edu.br

Maria Angelica da Costa Ciriaco
UFERSA
Pau dos Ferros, Brasil
maria.ciriaco@alunos.ufersa.edu.br

Vinícios Ramom de Oliveira Queiroz
UFERSA
Pau dos Ferros, Brasil
vinicios.queiroz@alunos.ufersa.edu.br

Zirlangio Correia da Silva Filho
UFERSA
Pau dos Ferros, Brasil
zirlangio.filho@alunos.ufersa.edu.br

***Resumo*—A digitalização do setor gastronômico tem impulsionado a adoção de soluções tecnológicas voltadas à otimização de processos e à melhoria da experiência do cliente. Entre esses processos, destaca-se o gerenciamento de reservas, que, quando realizado manualmente, está sujeito a falhas como sobreposição de agendamentos e baixa eficiência operacional. Este artigo apresenta o desenvolvimento do ReservaFácil, um sistema web para gestão de reservas em restaurantes, construído com uma arquitetura baseada em monorepositório. A aplicação oferece uma interface intuitiva para que os clientes realizem suas reservas de forma autônoma, além de um painel administrativo completo que permite ao restaurante controlar mesas, horários e usuários. Como diferencial metodológico, foi empregada a modelagem formal com notação Z, possibilitando a definição precisa de comportamentos e restrições do sistema, antecipando inconsistências lógicas ainda na fase de especificação. Além disso, foram aplicadas estratégias de testes manuais e automatizados, utilizando ferramentas como Jest e Postman, garantindo a confiabilidade da solução desenvolvida. Os resultados demonstram que o ReservaFácil é uma alternativa viável, segura e escalável para o gerenciamento de reservas, atendendo tanto às necessidades dos clientes quanto à gestão eficiente por parte dos administradores de restaurantes.**

***Index Terms*—Reservas, Testes, Funcionalidades.**

I. INTRODUÇÃO

O crescimento da demanda por soluções digitais no setor gastronômico tem impulsionado o desenvolvimento de sistemas que otimizam tanto a experiência dos clientes quanto a gestão administrativa dos estabelecimentos [10]. Dentre essas soluções, destaca-se a automação do processo de reservas, uma prática que, quando realizada manualmente, está sujeita a falhas de comunicação, duplicidade de agendamentos (*overbooking*) e baixa eficiência operacional, especialmente em micro e pequenos negócios, que vêm adotando ferramentas digitais para aumentar a competitividade e a eficiência operacional [13].

Nesse contexto, o presente trabalho propõe o desenvolvimento do **ReservaFácil**, um sistema de reservas para restaurantes que visa proporcionar uma interface intuitiva para clientes realizarem agendamentos de forma independente, ao mesmo tempo em que disponibiliza um painel administrativo completo para o controle de mesas, horários, estatísticas e usuários.

O sistema foi desenvolvido com base em um monorepositório, utilizando tecnologias modernas como Node.js [16], TypeScript [12], MongoDB [14] e React [11]. A escolha dessas ferramentas visa assegurar escalabilidade, segurança, tipagem estática e facilidade de manutenção, que são características essenciais para aplicações de médio a grande porte.

II. FUNDAMENTAÇÃO TEÓRICA

O desenvolvimento de sistemas web modernos exige a integração de diversas tecnologias e boas práticas de engenharia de software, desde a concepção da interface do usuário até a persistência de dados e segurança. Nesta seção, são apresentados os principais conceitos e tecnologias que fundamentaram a construção do sistema **ReservaFácil**, com apoio em obras reconhecidas da literatura como Pressman (2016) [17], Jorgensen (2013) [9] e Spillner *et al.* (2014) [20].

A. Arquitetura Monorepo

A arquitetura baseada em monorepositórios (monorepo) permite centralizar diferentes partes de um sistema, como o *Front-end*, *Back-end* e bibliotecas compartilhadas, em um único repositório de controle de versão. Isso facilita o reúso de código, a integração contínua e a consistência entre os módulos [7]. O gerenciamento eficaz de configuração de software, incluindo controle de versões e integração de componentes, é essencial para a manutenção da qualidade do sistema ao longo de seu ciclo de vida [17].

B. API RESTful e Back-end com Node.js

A arquitetura REST (*Representational State Transfer*) [6], é amplamente adotada na construção de aplicações web dis-

tribuídas devido à sua simplicidade, escalabilidade e compatibilidade com protocolos HTTP. O *Back-end* do **ReservaFácil** foi desenvolvido em Node.js, que é indicado para aplicações que exigem alto desempenho e concorrência. A adoção do TypeScript contribui para a robustez e manutenibilidade do código por meio de tipagem estática, conforme orientações de engenharia de software moderna descritas [17].

C. Banco de Dados NoSQL

O sistema emprega o MongoDB, um banco de dados NoSQL orientado a documentos, que oferece flexibilidade na modelagem de dados e escalabilidade horizontal. Segundo Santos e Oliveira (2020) [19], esse tipo de banco é especialmente adequado para aplicações com esquemas dinâmicos ou em constante evolução. A biblioteca Mongoose facilita a definição de modelos, validações e regras de negócio, alinhando-se aos princípios de abstração de dados e modularização propostos por Jorgensen (2013) [9] no contexto de engenharia e teste de software.

D. Front-end com React

A interface de usuário foi construída com React, uma biblioteca *JavaScript* declarativa e baseada em componentes. Sua combinação com *TypeScript*, *React Hook Form* e *Styled Components* favorece a reutilização de código, consistência de estilos e produtividade no desenvolvimento. Essa abordagem modular é compatível com os princípios de engenharia de software orientada a objetos, como encapsulamento e separação de preocupações [17] [20].

E. Segurança e Validação

A segurança da aplicação foi assegurada por meio de práticas consolidadas, como autenticação via *JSON Web Tokens* (JWT) [1], limitação de requisições com *express-rate-limit* [18], proteção de cabeçalhos HTTP com Helmet [3], e criptografia de senhas utilizando bcrypt [2]. Além disso, a biblioteca Zod [22] foi empregada para validação de dados tanto no *front-end* quanto no *back-end*, promovendo a integridade e confiabilidade das entradas do usuário. A validação de entrada e a proteção contra ameaças, como injeções de código, são pilares fundamentais na construção de software seguro [5]. A engenharia de testes, por sua vez, reforça a importância da verificação contínua das funcionalidades e dos requisitos de segurança ao longo do ciclo de desenvolvimento [9].

III. ABORDAGEM

A construção do projeto **ReservaFácil** seguiu uma arquitetura moderna baseada em monorepositório, com foco em escalabilidade, segurança e facilidade de manutenção. A abordagem envolveu o planejamento da arquitetura, a modelagem formal com a notação Z, a implementação das funcionalidades utilizando tecnologias web, e a execução de testes para validação da aplicação.

A. Arquitetura e Tecnologias Utilizadas

O projeto foi estruturado seguindo uma abordagem *fullstack* com TypeScript, promovendo uma divisão clara entre as camadas de *front-end*, *back-end* e banco de dados. Os principais componentes da arquitetura incluem:

- ***Front-end***: desenvolvido em React com TypeScript, o design priorizou responsividade e usabilidade;
- ***Back-end***: construído com Node.js, e feito a comunicação com o banco de dados MongoDB, e o esquema das rotas seguiu a arquitetura RESTful;
- **Banco de Dados**: o MongoDB foi escolhido por ser um banco NoSQL, oferecendo flexibilidade na modelagem dos documentos e agilidade no desenvolvimento;
- **Validação**: a biblioteca Zod foi utilizada para validação de dados no *back-end* e *front-end*, proporcionando maior robustez e segurança na manipulação das entradas do usuário;
- **Gerenciamento de Repositório**: o uso de um monorepositório permitiu manter o *back-end* e *front-end* no mesmo repositório, facilitando a gestão do projeto, padronização e reúso de código entre as camadas.

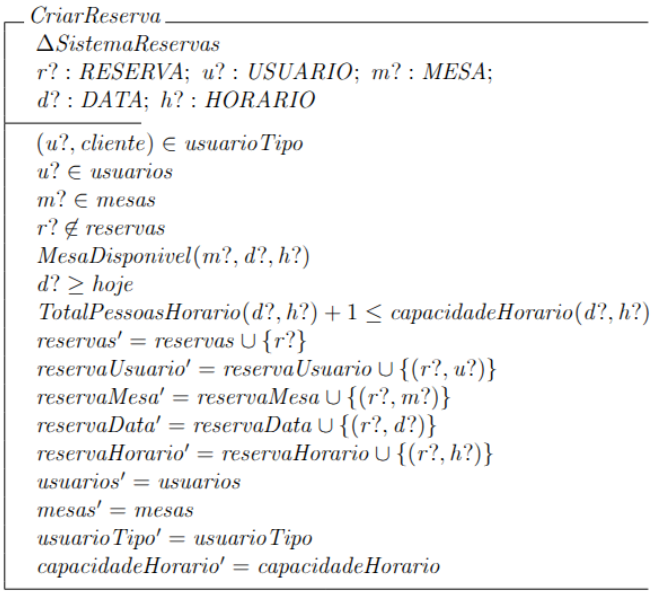
B. Modelagem Formal com Notação Z

Na fase de especificação, utilizou-se a notação Z, uma linguagem formal baseada em teoria dos conjuntos e lógica de predicados, empregada para modelagem matemática de sistemas computacionais, especialmente para garantir rigor e clareza na definição de estados e operações [21]. Com isso, modelaram-se formalmente os estados e operações do sistema, garantindo precisão e ausência de ambiguidades [21]. Foram representadas as estruturas de dados dos usuários e das reservas, além das operações CRUD *(Create, Read, Update, Delete)* associadas às reservas. Também foram especificadas restrições de negócio, como limites de reservas por horário, verificação de sobreposição e validação de dados obrigatórios.

A modelagem formal possibilitou antecipar inconsistências lógicas e orientar o planejamento dos testes funcionais e de integração, aumentando a confiabilidade da solução. Foram identificados, por exemplo, conflitos de reservas no mesmo horário para a mesma mesa, duplicidade de usuários e ausência de validação de campos obrigatórios. Tais problemas foram corrigidos ainda na fase de especificação. A operação de criação de reserva foi modelada com foco na correção dos estados e na consistência entre as entidades envolvidas.

A especificação formal da operação **CriarReserva**, exemplifica o uso da notação Z para definir os estados, pré-condições e pós-condições envolvidas nesse processo:

Figura 1. CriarReserva



A especificação da 1 assegura que somente usuários válidos possam criar reservas em mesas disponíveis, para datas futuras, respeitando a capacidade máxima definida para o horário. Esse modelo pode ser validado automaticamente e serve de base para uma implementação segura do sistema.

C. Estratégias de Teste

Diferentes estratégias de teste foram aplicadas para garantir a confiabilidade e robustez do sistema. Inicialmente, testes manuais validaram funcionalidades básicas, explorando login, cadastro, criação e cancelamento de reservas, o que permitiu identificar falhas visuais, comportamentos inesperados e inconsistências de uso [17].

Posteriormente, testes de integração foram conduzidos utilizando ferramentas como Insomnia e Postman, simulando requisições reais enviadas pelo *back-end* ao *front-end*. Os testes devem verificar o comportamento esperado dos componentes do sistema, incluindo a correta resposta das funcionalidades, tratamento de erros e a integridade dos dados retornados [17].

Com o avanço do desenvolvimento, foram implementados testes automatizados utilizando o framework `Jest` [4], com suporte do `MongoMemoryServer` [15] para simulação de banco de dados isolado. A suíte automatizada cobriu cerca de 82% do código em termos de linhas e statements, com um pouco menos de cobertura em branches (condições), algo comum em muitos projetos, pois é mais difícil cobrir todos os caminhos condicionais. A execução contabilizou 551 testes em 42 test suites, com 100% de sucesso.

D. Resumo Geral dos Testes Automatizados

Para avaliar a robustez do sistema, foram realizados testes automatizados abrangendo diferentes módulos da aplicação. A Tabela I apresenta um resumo das métricas dos testes executados, destacando a totalidade de testes aprovados e o tempo total de execução.

Tabela I
RESUMO GERAL DOS TESTES AUTOMATIZADOS

Métrica	Valor
Test Suites Executadas	42
Test Suites Aprovadas	42
Total de Testes	551
Testes com Sucesso	551 (100%)
Testes com Falha	0
Tempo Total de Execução	172.972 segundos
Ambiente	Local: *MongoMemoryServer*
Ferramenta	Jest

E. Cobertura por Categoria

A seguir, a Tabela II detalha a cobertura dos testes por categoria funcional, evidenciando as áreas da aplicação contempladas pelos casos de teste automatizados, que vão desde a validação dos modelos de dados até os fluxos de autenticação e segurança.

Tabela II
TESTES POR CATEGORIA

Categoria	Descrição
Integração	Fluxos administrativos, recuperação de senha, fluxo do usuário
Modelos	Validação, criação e regras de negócio para Reservation, User, Table
Controllers	Autenticação, dashboard, perfil, reservas (criação, cancelamento, confirmação, exclusão)
Serviços	Envio de e-mails (mockado), agendamento e limpeza automática
Middlewares	Segurança, limitação de requisições
Utils	Geração e validação de tokens, manipulação de horários e datas, logging
Testes Básicos	Verificação do ambiente

Os resultados demonstram que a suíte de testes abrange uma ampla gama de funcionalidades essenciais para o correto funcionamento do sistema, como autenticação, recuperação de senha, criação e cancelamento de reservas, verificação de regras de negócio e controle de acesso. Essa cobertura contribui para a garantia da qualidade e a redução de falhas durante o ciclo de desenvolvimento e manutenção.

F. Testes Pendentes

Ainda restam testes importantes a serem implementados, tais como:

- Testes de performance e carga (simulação com Artillery ou k6);
- Testes de segurança para validação de permissões e resistência a ataques;
- Testes de fluxos especiais (ex.: *token* expirado, exclusão de conta);
- Testes de usabilidade e acessibilidade no *front-end*.

Os testes realizados permitiram identificar e corrigir problemas importantes, incluindo validação incompleta, falhas de autenticação, inconsistências nos fluxos de reserva, mensagens

de erro pouco claras e sincronização entre *back-end* e *front-end*. Aspectos como a intuitividade, responsividade e clareza da interface foram avaliados com apoio da ferramenta Google Lighthouse [8]. As métricas de acessibilidade (entre 91 e 95) e boas práticas (100 em todas as páginas) indicam aderência a diretrizes de usabilidade, estrutura semântica adequada e boa experiência de navegação em diferentes dispositivos. Esses resultados confirmam que o sistema oferece uma interface amigável e eficiente para os diferentes perfis de usuários.

G. Validação e Segurança

A aplicação foi projetada com foco em validação de entradas e segurança de dados. Além do uso extensivo da biblioteca `Zod`, foram implementadas práticas como autenticação via *middleware*, geração e verificação de `JWT` (JSON *Web Tokens*) e criptografia de senhas com o algoritmo `bcrypt`.

Essas medidas garantem que apenas usuários autorizados tenham acesso a determinadas operações, que dados sensíveis sejam protegidos em trânsito e em repouso, e que o sistema seja resistente a ataques como acesso indevido ou manipulação de sessões.

H. Funcionalidades do Sistema

O sistema **ReservaFácil** foi desenvolvido com foco na gestão eficiente de reservas de restaurantes. Entre as funcionalidades principais destacam-se: cadastro e *login* de usuários, criação e cancelamento de reservas com atualização de horários em tempo real, painel administrativo para controle de reservas e sistema de notificações visuais para novas atividades.

As interfaces foram projetadas para oferecer experiência intuitiva, com layout responsivo e informações organizadas de maneira clara para usuários comuns e administradores.

I. Colaboração e Controle de Versões

O desenvolvimento colaborativo foi essencial para o sucesso do projeto. O GitHub foi utilizado para controle de versões, gerenciamento de tarefas e revisão de código. Foram abertas diversas issues para correções e melhorias, como o ajuste de redirecionamento após login, a validação de campos obrigatórios e a responsividade do layout. O repositório do código-fonte está disponível em https://github.com/JedersonYago/ReservaDeRestaurante.

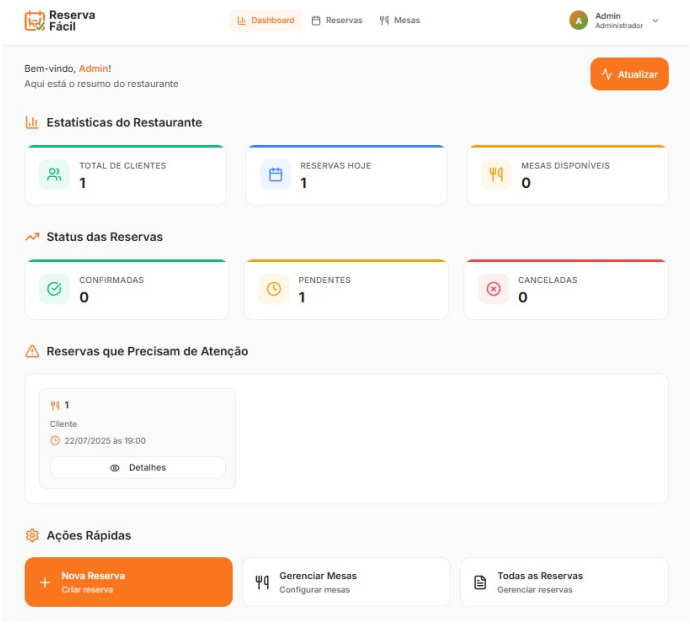
Cada funcionalidade ou correção foi associada a *commits* e *pull requests*, garantindo rastreabilidade das alterações. A comunicação da equipe foi feita por mensagens e reuniões, assegurando alinhamento e agilidade na tomada de decisões.

J. Interface do Sistema

Para facilitar a compreensão da proposta desenvolvida, são apresentadas a seguir duas telas principais do sistema: a interface do administrador e a interface do cliente. Ambas foram projetadas com foco na usabilidade e clareza das informações, de modo a atender às necessidades específicas de cada perfil de usuário.

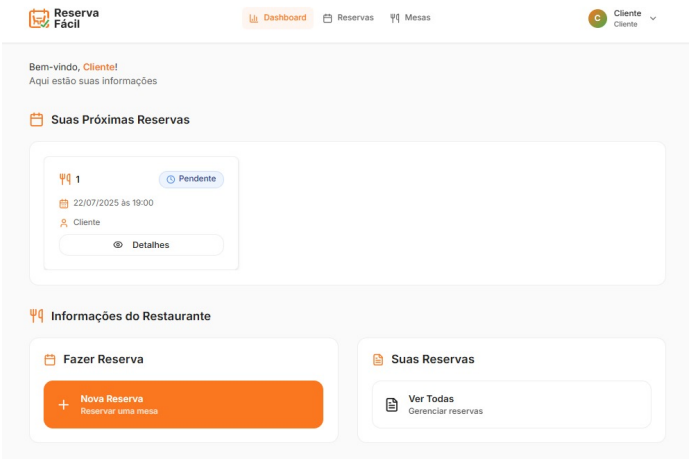
A Figura 2 ilustra o painel de controle acessado pelo administrador, onde é possível visualizar informações como o número total de reservas, avisos do sistema e ações rápidas para o gerenciamento do restaurante. Esta interface permite o controle centralizado de todas as operações administrativas, tornando a gestão mais eficiente.

Figura 2. Tela de Dashboard de Administrador



Já a Figura 3 apresenta a visualização do painel do cliente. Nessa interface, o usuário tem acesso às suas reservas ativas, histórico de visitas e opções para realizar novas reservas. O layout foi pensado para ser intuitivo, com ênfase na experiência do usuário e na praticidade do acesso às funcionalidades disponíveis.

Figura 3. Tela de dashboard do cliente



Essas telas evidenciam a separação de papéis dentro do sistema e refletem o cumprimento dos requisitos funcionais definidos durante o processo de especificação. A especificação foi conduzida com base em uma abordagem híbrida, combinando descrição textual e formalização matemática com

a notação Z. Os principais requisitos, como o processo de criação de reservas, foram modelados com pré e pós-condições formais, garantindo a integridade das operações. Essa base formal permitiu identificar inconsistências antecipadamente e orientar o desenvolvimento e os testes do sistema.

IV. CONSIDERAÇÕES FINAIS

O desenvolvimento do sistema **ReservaFácil** representou uma importante iniciativa, permitindo a aplicação prática de conceitos de engenharia de software, testes, validações e boas práticas de desenvolvimento. O projeto, hospedado em um monorepositório com a organização dos diretórios web, *API* e *common*, utilizou tecnologias modernas como React.js, Node.js, TypeScript, Zod e MongoDB, o que contribuiu significativamente para a qualidade, escalabilidade e manutenibilidade da aplicação.

A proposta principal foi criar um sistema intuitivo para gerenciamento de reservas de mesas em restaurantes, contemplando tanto a perspectiva do cliente, que realiza a reserva, quanto da administração do restaurante, que acompanha e organiza os horários e disponibilidade.

Durante a execução do projeto, observou-se que a adoção de um *monorepo* favoreceu a padronização e reutilização de código, facilitando a comunicação entre *front-end* e *back-end*. A estrutura comum para validações, como o uso do Zod em conjunto com o TypeScript, proporcionou maior previsibilidade no desenvolvimento e segurança nos dados trafegados entre as camadas da aplicação. Essa abordagem reduziu significativamente inconsistências entre validação de formulários no *front-end* e checagem no *back-end*, o que resultou em menor taxa de erros durante a execução.

Além disso, a integração com bibliotecas de autenticação, autenticação com JWT, e o uso de *middlewares* para proteger rotas sensíveis contribuiu para um sistema mais seguro e aderente a práticas modernas de desenvolvimento. A experiência do usuário também foi considerada uma prioridade, com um *front-end* responsivo, limpo e de fácil navegação.

A fase de testes foi fundamental para a evolução do projeto. Foram realizados testes manuais e exploratórios, focados principalmente na navegação, fluxo de criação de reservas, cadastro de restaurantes, autenticação e permissões. O feedback obtido por meio desses testes permitiu a correção de erros de fluxo, como inconsistências em horários disponíveis e restrições de acesso a páginas administrativas. Houve também a aplicação de testes em nível de API para verificar o correto funcionamento dos *endpoints* REST.

O uso da Notação Z foi um diferencial metodológico importante. A especificação formal permitiu modelar matematicamente os comportamentos e restrições do sistema ainda na fase de concepção, trazendo clareza sobre os estados possíveis e antecipando inconsistências. A modelagem abordou aspectos como lógica de reserva, disponibilidade de mesas e autenticação. Embora pouco utilizada no cotidiano do desenvolvimento de software, sua aplicação neste projeto contribuiu para maior confiabilidade do sistema, porque evidenciou inconsistências lógicas na fase de modelagem, permitindo correções antecipadas e maior confiabilidade nas operações críticas.

V. TRABALHOS FUTUROS

Embora o sistema atenda bem à proposta inicial, há diversas oportunidades para expansão. Entre elas, destacam-se o desenvolvimento de um aplicativo móvel com notificações *push* para reservas, a implementação de um chat em tempo real para comunicação entre clientes e restaurante, a criação de um sistema de gerenciamento de filas de espera com alertas automáticos e a melhoria dos recursos de acessibilidade para garantir maior autonomia a usuários com deficiência.

REFERÊNCIAS

[1] AUTH0. Introduction to json web tokens. https://www.jwt.io/introduction, 2025. Acesso em: 27 jul. 2025.

[2] BCRYPT.JS CONTRIBUTORS. bcrypt.js. https://github.com/dcodeIO/bcrypt.js, 2025. Acesso em: 27 jul. 2025.

[3] CONTRIBUTORS, H. Helmet: Help secure express apps with http headers. https://helmetjs.github.io/, 2025. Acesso em: 27 jul. 2025.

[4] FACEBOOK, INC. Jest — delightful javascript testing framework. https://jestjs.io/, 2025. Acesso em: 27 jul. 2025.

[5] FÁTIMA, G. P. D., SANTOS, B. J. D., MACIEL, Z. A., AND RUBEM, F. *Testes de Software e Gerência de Configuração*. Editora Érica, 2020.

[6] FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[7] GITLAB. Monorepos. https://docs.gitlab.com/user/project/repository/monorepos/, 2025. Acesso em: 28 jul. 2025.

[8] GOOGLE LLC. Lighthouse: Ferramenta de auditoria para performance e qualidade web. https://developer.chrome.com/docs/lighthouse/overview?hl=pt-br, 2023. Acesso em: 27 jul. 2025.

[9] JORGENSEN, P. C. *Software Testing: A Craftsman's Approach*. CRC Press, 2013.

[10] M, C. A., AND G, M. Intuitive online table booking system for restaurants. In *2024 International Conference on Smart Technologies for Sustainable Development Goals (ICSTSDG)* (2024), IEEE.

[11] META PLATFORMS, INC. React – a javascript library for building user interfaces. https://react.dev, 2024. Acesso em: 28 jul. 2025.

[12] MICROSOFT. Typescript: Javascript with syntax for types. https://www.typescriptlang.org, 2024. Acesso em: 28 jul. 2025.

[13] MIRANDA, I. L. D., AND VIEIRA, L. B. Uso de ferramentas digitais em micro e pequenos negócios tradicionais: uma análise no setor de restaurantes. Trabalho de Conclusão de Curso (Bacharelado em Administração) – Universidade Federal de Santa Catarina, 2023. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/266162/TCC%20FINAL%20ASSINADO%20%20PDFA.pdf?sequence=1&isAllowed=y. Acesso em: 24 jul. 2025.

[14] MONGODB INC. Mongodb. https://www.mongodb.com, 2024. Acesso em: 28 jul. 2025.

[15] MONGODB, INC. Mongomemoryserver — in-memory mongodb for testing. https://github.com/nodkz/mongodb-memory-server, 2025. Acesso em: 27 jul. 2025.

[16] NODE.JS FOUNDATION. Node.js. https://nodejs.org, 2024. Acesso em: 28 jul. 2025.

[17] PRESSMAN, R. S. *Engenharia de Software: Uma abordagem profissional*. McGraw-Hill, 2016.

[18] RATE-LIMIT CONTRIBUTORS, E. express-rate-limit. https://github.com/express-rate-limit/express-rate-limit, 2025. Acesso em: 27 jul. 2025.

[19] SANTOS, J. C., AND OLIVEIRA, L. Uma análise comparativa entre bancos de dados relacionais e nosql em aplicações web. *Revista Tecnologia e Sociedade* (2020).

[20] SPILLNER, A., LINZ, T., AND SCHAEFER, H. *Software Testing Foundations: A Study Guide for the Certified Tester Exam*. Rocky Nook, 2014.

[21] SPIVEY, J. M. *The Z Notation: A Reference Manual*, 2nd ed. Prentice Hall, Hemel Hempstead, UK, 1992.

[22] ZOD CONTRIBUTORS. Zod: Typescript-first schema validation with static type inference. https://zod.dev/, 2025. Acesso em: 27 jul. 2025.