INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS - *CAMPUS* BETIM
BACHARELADO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Marcial Faria Vasconcelos

SISTEMA DE AGENDAMENTOS BASEADO EM REPUTAÇÃO:
aplicação de sistemas web para gestão de recursos

Betim
2026

MARCIAL FARIA VASCONCELOS

SISTEMA DE AGENDAMENTOS BASEADO EM REPUTAÇÃO:
aplicação de sistemas web para gestão de recursos

Trabalho de Conclusão de Curso apresentado à banca examinadora do curso de Engenharia de Controle e Automação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais *Campus* Betim, como parte dos requisitos para obtenção do título de Bacharel em Engenharia de Controle e Automação.

Orientador: Prof. Me. Virgil Del Duca Almeida

Betim
2026

FICHA CATALOGRÁFICA

V331s Vasconcelos, Marcial Faria

Sistema de agendamentos baseado em reputação: aplicação de sistemas web para gestão de recursos / Marcial Faria Vasconcelos – 2026.

63 f. : il.

Trabalho de conclusão de curso (Bacharelado em Engenharia de Controle e Automação) - Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais, Câmpus Betim, 2026.

Orientação: Prof. Me. Virgil Del Duca Almeida

1. Sistemas web. 2. Sistemas modulares. 3. Soluções computacionais. 4. Controle de acesso. 5. Engenharia de Controle e Automação. I. Vasconcelos, Marcial Faria. II. Título.

CDU: 004.43

Ficha catalográfica elaborada pelo Bibliotecário Denísio Pereira Marcos CRB-6/3142

Marcial Faria Vasconcelos

SISTEMA DE AGENDAMENTOS BASEADO EM REPUTAÇÃO: aplicação de sistemas web para gestão de recursos

Trabalho de Conclusão de Curso apresentado à banca examinadora do curso de Engenharia de Controle e Automação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais *Campus* Betim, como parte dos requisitos para obtenção do título de Bacharel em Engenharia de Controle e Automação.

Aprovado em: 03 / 02 / 2026 pela banca examinadora:

Prof. Me. Virgil Del Duca Almeida (Orientador) - IFMG

Prof. Me. Deliene Costa Guimarães Barros - IFMG

Prof. Dr. Gustavo Lobato Campos - IFMG

RESUMO

A crescente demanda por organização e uso eficiente de recursos físicos compartilhados em ambientes institucionais evidencia a necessidade de soluções computacionais capazes de reduzir conflitos, otimizar a ocupação e apoiar a gestão operacional. Neste contexto, este trabalho apresenta o desenvolvimento de um sistema web para agendamento e gestão de recursos compartilhados, tais como vagas de estacionamento, salas, laboratórios e mesas de trabalho, incorporando um mecanismo de reputação baseado no histórico de utilização dos usuários. A solução proposta adota uma arquitetura modular e orientada a regras de negócio, permitindo a integração com diferentes tecnologias de monitoramento e controle de acesso, por meio de comunicação via protocolos HTTP e MQTT. A metodologia empregada baseia-se na aplicação de engenharia de requisitos aliada a um processo de desenvolvimento incremental e modular, assegurando rastreabilidade entre requisitos, implementações e critérios de aceitação. O sistema foi implementado com backend em Node.js, banco de dados relacional MariaDB e interfaces de programação de aplicações REST, além de um protótipo de integração com dispositivos físicos para validação de acesso. Os resultados obtidos demonstram a viabilidade da solução para apoiar o gerenciamento de recursos compartilhados, destacando-se o mecanismo de reputação como ferramenta para priorização automática de reservas e mitigação de conflitos em cenários de alta concorrência.

Palavras-chave: agendamento de recursos; sistemas web; controle de acesso; reputação de usuários; sistemas modulares.

ABSTRACT

The increasing need for efficient organization and utilization of shared physical resources in institutional environments highlights the importance of computational solutions capable of reducing conflicts, optimizing occupancy, and supporting operational management. In this context, this work presents the development of a web-based system for scheduling and managing shared resources, such as parking spaces, meeting rooms, laboratories, and workstations, incorporating a reputation mechanism based on users' historical behavior. The proposed solution adopts a modular, business-rule-oriented architecture, enabling integration with different monitoring and access control technologies through communication using HTTP and MQTT protocols. The methodology is grounded in requirements engineering combined with an incremental and modular development process, ensuring traceability between requirements, implementations, and acceptance criteria. The system was implemented with a Node.js backend, a MariaDB relational database, RESTful application programming interfaces, and a prototype for integration with physical access control devices. The obtained results demonstrate the feasibility of the proposed solution to support the management of shared resources, highlighting the reputation mechanism as an effective tool for automatic reservation prioritization and conflict mitigation in high-demand scenarios.

Keywords: resource scheduling; web systems; access control; user reputation; modular systems.

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface (Interface de Programação de Aplicações)

CORS – Cross-Origin Resource Sharing (Compartilhamento de Recursos de Origem Cruzada)

CRUD – Create, Read, Update, Delete (Criação, Leitura, Atualização e Exclusão)

DER – Entity-Relationship Diagram (Diagrama Entidade Relacionamento)

ESP32 – Espressif 32 (Microcontrolador Espressif 32)

GPS – Global Positioning System (Sistema de Posicionamento Global)

GPIO – General Purpose Input/Output (Entrada/Saída de Uso Geral)

HTTP – Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto)

HTTPS – Hypertext Transfer Protocol Secure (Protocolo de Transferência de Hipertexto Seguro)

I^2C – Inter-Integrated Circuit (Circuito Integrado Interconectado)

IETF – Internet Engineering Task Force (Grupo de Trabalho de Engenharia da Internet)

IFMG – Instituto Federal de Minas Gerais

JSON – JavaScript Object Notation (Notação de Objetos JavaScript)

JWT – JSON Web Token (Token Web JSON)

MQTT – Message Queuing Telemetry Transport (Transporte de Telemetria de Enfileiramento de Mensagens)

NFC – Near Field Communication (Comunicação de Campo Próximo)

OASIS – Organization for the Advancement of Structured Information Standards (Organização para o Avanço de Padrões de Informação Estruturada)

POC – Proof of Concept (Prova de Conceito)

QoS – Quality of Service (Qualidade de Serviço)

REST – Representational State Transfer (Transferência de Estado Representacional)

RFID – Radio-Frequency Identification (Identificação por Radiofrequência)

SPI – Serial Peripheral Interface (Interface Periférica Serial)

SPA – Single Page Application (Aplicação de Página Única)

SQL – Structured Query Language (Linguagem de Consulta Estruturada)

TCP/IP – Transmission Control Protocol/Internet Protocol (Protocolo de Controle de Transmissão/Protocolo de Internet)

TLS – Transport Layer Security (Segurança da Camada de Transporte)

UART – Universal Asynchronous Receiver/Transmitter (Transmissor/Receptor Assíncrono Universal)

UI – User Interface (Interface do Usuário)

URI – Uniform Resource Identifier (Identificador Uniforme de Recurso)

UX – User Experience (Experiência do Usuário)

SUMÁRIO

1 INTRODUÇÃO

A crescente necessidade de organizar e otimizar o uso de recursos físicos em ambientes corporativos, acadêmicos e institucionais tem impulsionado o desenvolvimento de sistemas capazes de gerenciar espaços de forma eficiente. Salas de reunião, laboratórios, mesas de trabalho e vagas de estacionamento são exemplos de recursos limitados cujo uso descoordenado resulta em conflitos, perda de produtividade e subutilização da infraestrutura disponível.

Com a popularização de modelos de trabalho híbridos e o aumento da circulação de pessoas em ambientes compartilhados, torna-se cada vez mais evidente a necessidade de ferramentas que proporcionem previsibilidade e controle sobre a distribuição desses recursos. A ausência de mecanismos adequados de agendamento e monitoramento frequentemente leva à superlotação de determinados espaços, ao uso desregulado de dependências institucionais e a dificuldades na gestão operacional.

Diante desse cenário, este trabalho apresenta o desenvolvimento de um sistema web voltado à administração e ao agendamento de recursos compartilhados, inicialmente projetado para estacionamentos, mas posteriormente expandido para outros tipos de ambientes. A solução proposta integra comunicação com o usuário, controle de acesso e registro histórico de utilização, além de um sistema de reputação capaz de avaliar comportamentos como presença, ausência e cancelamentos recorrentes, contribuindo para uma gestão mais eficiente e justa.

Serão discutidos aspectos teóricos e práticos relacionados ao desenvolvimento e à implementação do sistema, bem como sua aplicabilidade em diferentes cenários e a capacidade de auxiliar organizações a melhorar o uso de suas instalações.

1.1 Justificativa

O gerenciamento adequado de espaços físicos é fundamental para organizações que lidam com recursos compartilhados, especialmente aquelas de grande porte ou que operam com alta rotatividade de usuários. A capacidade de coletar e analisar dados referentes ao uso desses recursos permite compreender padrões de comportamento, antecipar demandas e otimizar processos internos.

Além disso, o modelo de trabalho híbrido, cuja adoção cresceu significativamente após o período de isolamento social imposto pela pandemia da Covid-19, intensificou a necessidade de ferramentas que auxiliem no planejamento da ocupação de ambientes (ADRJAN *et al.*, 2025). Em cenários nos quais colaboradores utilizam diferentes instalações em dias distintos, informações prévias sobre a intenção de uso tornam-se essenciais para manter a ordem, a eficiência e a segurança.

Com o sistema apresentado neste trabalho, não apenas será possível controlar a disponibilidade dos recursos, como também avaliar e registrar o comportamento dos usuários, permitindo

identificar horários de maior fluxo, frequência de utilização e práticas que possam comprometer a organização. Esses dados podem servir de base para decisões estratégicas relacionadas à infraestrutura e ao gerenciamento operacional da instituição.

1.2 Colocação do Problema

É possível, por meio de um sistema web que centraliza agendamentos, controle de acesso e um mecanismo de reputação baseado no histórico de uso, melhorar a organização e o aproveitamento de recursos compartilhados — como salas, laboratórios, mesas e estacionamentos — sem demandar infraestrutura complexa ou altos custos de implementação?

1.3 Objetivos

1.3.1 Geral

Desenvolver um sistema de agendamento para recursos escassos em ambientes compartilhados, integrando comunicação entre usuários e banco de dados, monitoramento do estado dos recursos e registro histórico para composição de um indicador de reputação. O sistema deve possibilitar que o usuário visualize e reserve recursos, bem como acessar dependências mediante validação adequada, de acordo com a infraestrutura disponível no local de instalação.

1.3.2 Específicos

- Desenvolver uma aplicação web para agendamento de recursos variados, incluindo vagas de estacionamento, salas, laboratórios e mesas;
- Implementar um módulo de autenticação e controle de acesso integrado à aplicação principal;
- Construir o protótipo de um dispositivo capaz de realizar a identificação do usuário e validar seu acesso aos recursos reservados;
- Integrar o sistema com equipamentos ou sensores já existentes na infraestrutura da instituição, quando aplicável;
- Implementar um sistema de reputação baseado no comportamento dos usuários, considerando presença, ausência, cancelamentos tardios e utilização indevida.

1.4 Organização do Trabalho

No Capítulo 1 apresenta-se o contexto geral, o problema e a solução proposta, bem como os objetivos do trabalho.

O Capítulo 2 apresenta os fundamentos teóricos que embasam o desenvolvimento do sistema, incluindo conceitos de gerenciamento de recursos, controle de acesso e sistemas de reputação.

No Capítulo 3 descrevem-se as etapas de desenvolvimento, a modelagem do sistema e as tecnologias utilizadas na implementação.

O Capítulo 4 apresenta os resultados obtidos, demonstrando o funcionamento do sistema, analisando o desempenho e discutindo os desafios encontrados durante a implementação.

Por fim, no Capítulo 5 são apresentadas as conclusões, relacionando a proposta inicial aos resultados alcançados, bem como sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

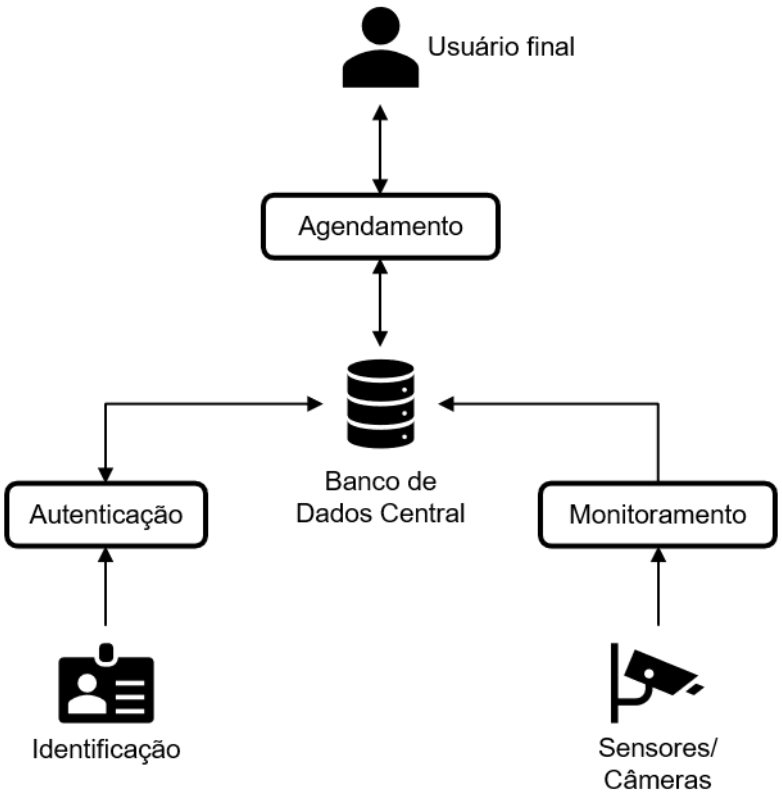
2.1 Análise do Problema

Para atingir os objetivos traçados, é necessário, primeiramente, compreender as necessidades do problema proposto e organizar esses requisitos em desafios de menor escala.

O projeto foi subdividido em módulos de menor complexidade que se comunicam entre si para compor um sistema completo. A Figura 1 apresenta as principais necessidades identificadas, que podem ser resumidas nos seguintes pontos:

1. Disponibilizar uma aplicação que permita aos usuários visualizar a disponibilidade dos recursos e realizar agendamentos.
2. Fornecer um sistema de controle de acesso capaz de autorizar ou negar a utilização de recursos conforme as regras institucionais.
3. Registrar e monitorar o uso dos recursos em tempo real, fornecendo informações atualizadas ao sistema central.
4. Estabelecer um conjunto de regras para gerar um índice qualificativo da reputação de cada usuário.

Figura 1 – Ilustração do Problema Proposto.



Fonte: Elaborado pelo autor, 2025.

5. Integrar todos os módulos a um banco de dados central, responsável pelo armazenamento das informações dos usuários, agendamentos e uso dos recursos.

Nos tópicos seguintes, serão discutidas as tecnologias consideradas para a implementação dos módulos do sistema, bem como trabalhos semelhantes que abordam problemas correlatos ou utilizam tecnologias similares.

2.2 Implementações similares e complementares

Durante o processo de levantamento bibliográfico, foram identificados diversos trabalhos acadêmicos que abordam, de forma parcial ou complementar, os desafios tratados neste projeto. Esses trabalhos exploram soluções relacionadas ao gerenciamento de vagas, controle de acesso, monitoramento de recursos compartilhados e integração de sistemas físicos com aplicações computacionais. A análise desses estudos permitiu compreender diferentes abordagens metodológicas, tecnologias empregadas e limitações práticas, contribuindo para a definição das escolhas realizadas neste trabalho.

O estudo desenvolvido por Nwohiri e Muhammad (2020) apresenta um sistema de controle de acesso veicular aplicado a um campus universitário, utilizando leitores biométricos e câmeras de vigilância integradas a um microcomputador Raspberry Pi. O foco principal do trabalho está na segurança física e no controle de entrada, com ênfase na identificação do usuário no momento do acesso. Embora não trate diretamente de agendamentos ou gerenciamento dinâmico de recursos, o trabalho demonstra a viabilidade da integração entre dispositivos físicos, sistemas embarcados e aplicações computacionais centralizadas, aspecto diretamente relacionado à proposta deste projeto.

O trabalho de Melo, Silva e Santos (2022) apresenta o desenvolvimento de um aplicativo voltado à mobilidade urbana, com foco no gerenciamento inteligente de vagas de estacionamento em ambientes urbanos. A solução propõe a utilização de uma aplicação móvel para registro e consulta de vagas disponíveis em estacionamentos, buscando reduzir o tempo de procura por vagas. Diferentemente deste projeto, o estudo concentra-se na perspectiva do usuário final e na mobilidade urbana, não abordando aspectos como controle institucional, reputação de usuários ou integração com sistemas físicos de monitoramento.

Outro trabalho relevante é a dissertação de Calis (2018), que investiga a aplicação de redes neurais convolucionais para reconhecimento automático de placas veiculares. Embora o objetivo principal seja a identificação de veículos e não o gerenciamento de vagas, a pesquisa demonstra a viabilidade do uso de técnicas de visão computacional como mecanismos de autenticação e rastreamento, podendo ser utilizada de forma complementar em sistemas de controle de acesso e monitoramento, como o proposto neste trabalho.

Já o trabalho de Valipour *et al.* (2016) propõe um sistema automatizado para identificação

de vagas livres e ocupadas por meio de redes neurais convolucionais profundas. A solução emprega câmeras instaladas em estacionamentos e modelos de aprendizado profundo treinados para classificar cada vaga individualmente, com os resultados sendo disponibilizados em uma aplicação móvel. A principal contribuição desse estudo está na automação da coleta de dados de ocupação, eliminando a necessidade de sensores físicos dedicados por vaga. Entretanto, o trabalho não aborda mecanismos de agendamento prévio, controle de usuários ou regras de priorização, concentrando-se exclusivamente na detecção visual.

De forma semelhante, Nurullayev e Lee (2019) desenvolvem uma abordagem baseada em redes neurais convolucionais dilatadas para análise da ocupação de vagas de estacionamento. O estudo avalia diferentes arquiteturas e parâmetros de treinamento utilizando bases de dados públicas, como PKLot e CNRPark+EXT, com foco na robustez e generalização dos modelos. A pesquisa destaca desafios como variações de iluminação, oclusões e diferentes ângulos de câmera, reforçando a complexidade do monitoramento visual em ambientes reais. Embora altamente relevante no contexto de detecção automática, o trabalho não contempla a integração com sistemas de gestão, autenticação de usuários ou políticas de uso dos recursos.

O trabalho de Kl *et al.* (2021) propõe um sistema de reserva inteligente de vagas de estacionamento baseado em aplicação, integrando técnicas de reconhecimento automático de placas veiculares por meio da ferramenta *OpenALPR*. A principal contribuição do estudo está na utilização de visão computacional para identificação dos veículos no momento de entrada e saída do estacionamento, permitindo associar automaticamente a vaga reservada ao veículo detectado, reduzindo a necessidade de intervenção humana. Diferentemente da proposta deste trabalho, a solução apresentada concentra-se fortemente na etapa de reconhecimento e validação do veículo, tratando o processo de reserva de forma mais direta e menos flexível, sem incorporar mecanismos avançados de reputação, priorização de usuários ou políticas institucionais configuráveis. Ainda assim, o estudo demonstra a viabilidade prática da integração entre sistemas de reconhecimento visual e aplicações de gerenciamento de estacionamento, reforçando a relevância do uso de dados provenientes de fontes externas para atualização do estado das vagas, aspecto que pode ser explorado de forma complementar à arquitetura proposta neste projeto.

Diferentemente dos trabalhos analisados, a proposta apresentada neste trabalho busca integrar, em uma única solução, os seguintes aspectos: agendamento prévio de recursos, autenticação de usuários, monitoramento de status em tempo real, aplicação de regras baseadas em reputação e comunicação com sistemas externos por meio de protocolos padronizados como HTTP e MQTT. Essa abordagem permite maior flexibilidade, independência da tecnologia de monitoramento utilizada e adaptação a diferentes contextos institucionais, posicionando o sistema como uma solução extensível e alinhada às boas práticas da engenharia de software moderna.

2.3 Engenharia de Requisitos

A engenharia de requisitos define o que o sistema deve fazer e envolve um conjunto de atividades sistemáticas de descoberta, análise, especificação e validação dos requisitos. Segundo Valente (2020), essas atividades devem ser realizadas ao longo de todo o ciclo de vida do sistema, envolvendo o levantamento e o tratamento de requisitos funcionais e não-funcionais. Sommerville (2011) complementa que a engenharia de requisitos é o processo de descobrir, analisar, documentar e verificar as funções e restrições do sistema, gerando um documento de requisitos do sistema. Dessa forma, inicia-se com a elicitação de requisitos, passando pela análise e priorização, até a definição final do escopo do projeto (PRESSMAN; MAXIM, 2016).

A elicitação de requisitos consiste em interagir com os *stakeholders* para extrair suas necessidades e expectativas em relação ao sistema. Valente (2020) observa que essa etapa envolve técnicas como entrevistas, questionários, protótipos e observações do ambiente de negócio, com o objetivo de obter dos usuários os requisitos principais. A experiência com o usuário e o envolvimento de patrocinadores do projeto são essenciais nessa fase, garantindo que a visão do sistema seja corretamente compreendida. Além disso, protótipos e estudos de caso podem ser empregados para refinar as demandas iniciais e validar premissas de projeto (PRESSMAN; MAXIM, 2016).

Após a elicitação, realiza-se a análise de requisitos, que consiste em refinar e detalhar as informações obtidas, identificando conflitos, inconsistências ou omissões. Nessa etapa, os requisitos são classificados em funcionais e não-funcionais, e devem-se definir critérios de aceitação para cada um. Uma etapa crucial é a priorização dos requisitos, ou seja, estabelecer quais funcionalidades são mais críticas ou urgentes para o cliente. Pressman e Maxim (2016) destaca que a engenharia de requisitos inclui a classificação e priorização das necessidades, de modo a concentrar esforços nas partes de maior valor de negócio. Em suma, a análise e priorização permitem orientar o desenvolvimento incremental do sistema e reduzir riscos de retrabalho futuro.

2.4 Escolha da Arquitetura de Software

A arquitetura de software determina a estrutura global do sistema, definindo seus componentes principais, suas responsabilidades e a forma como eles se inter-relacionam. Para Pressman e Maxim (2016), a arquitetura é a estrutura do sistema de software, incluindo os componentes de software, suas propriedades externamente visíveis e as relações entre eles. Valente (2020) enfatiza que a arquitetura também envolve as decisões de design de mais alto nível, que raramente podem ser revertidas no futuro, como a escolha de padrões arquiteturais, linguagens de programação e bancos de dados. É, portanto, crucial selecionar uma arquitetura adequada às necessidades do sistema, uma vez que essa decisão influencia a escalabilidade, manutenibilidade e desempenho do software.

No desenvolvimento do sistema de agendamento de recursos, optou-se por uma arquitetura em camadas, dividindo a aplicação nas camadas de apresentação, negócio e persistência de dados. Essa escolha está alinhada com o princípio de modularidade destacado por Pressman e Maxim (2016), pois cada camada assume uma responsabilidade bem definida. Valente (2020) descreve padrões arquiteturais como MVC, que facilitam a separação de responsabilidades. A comunicação entre as camadas é realizada por meio de interfaces e APIs, adotando-se um estilo RESTful para as chamadas do cliente ao servidor. Esse estilo arquitetural define um conjunto de restrições para aplicações Web baseado no uso de recursos via protocolo HTTP e é amplamente utilizado em serviços distribuídos (FIELDING, 2000).

Pressman e Maxim (2016) ressalta que definir explicitamente a arquitetura no início do projeto permite analisar alternativas em estágios iniciais e reduzir riscos de projeto. Dessa forma, a escolha da arquitetura em camadas justifica-se pela sua maturidade e ampla adoção em sistemas corporativos, oferecendo escalabilidade e facilidade de manutenção. Além disso, o uso de APIs RESTful para comunicação torna o sistema extensível, permitindo, por exemplo, a integração futura com clientes móveis ou outros serviços.

2.5 Organização em Módulos

Uma arquitetura modular pressupõe que o sistema seja dividido em componentes ou subsistemas autônomos, cada qual com responsabilidade bem definida. Nesse sentido, Pressman e Maxim (2016) destaca que um bom projeto deve ser modular, ou seja, logicamente particionado em elementos ou subsistemas. Cada módulo do sistema de agendamento concentra funcionalidades relacionadas; por exemplo, podemos ter módulos separados para gestão de usuários, controle de reservas e administração de recursos, obedecendo ao princípio de responsabilidade única (VALENTE, 2020). A alta coesão dentro de cada módulo facilita a compreensão e a manutenção do código, enquanto o baixo acoplamento entre módulos (obtido através de interfaces claras) minimiza dependências indesejadas.

Nesta implementação a camada de apresentação do sistema consome serviços oferecidos pela camada de negócios através de uma API REST, o que permite um acoplamento frouxo entre cliente e servidor (VALENTE, 2020). Internamente, a camada de negócios invoca funções de acesso à base de dados definidas na camada de persistência, mantendo cada módulo isolado por uma API de acesso a dados. Valente (2020) observa que em sistemas distribuídos modernos (como microsserviços) a comunicação entre componentes geralmente envolve protocolos como HTTP/REST. De forma geral, foram adotado protocolos e formatos de troca de mensagens que asseguram que os módulos comuniquem-se de maneira robusta, tratando falhas e garantindo a integridade das operações.

2.6 Sistema de Agendamento: Aplicação Web

Para o sistema de agendamento, optou-se pela utilização de uma interface web por diversos motivos, sendo o principal a versatilidade que essa tecnologia possibilita, sendo possível acessa-la por meio de qualquer computador ou *smartphone*. Outro grande motivador para essa escolha foi a grande popularização deste tipo de aplicação nos últimos anos.

Projetos similares normalmente utilizam esse mesmo método para realizar a interface com os usuários. Exemplos desse tipo de aplicação são encontrados em trabalhos como o sistema de gerenciamento inteligente de estacionamento realizado por um grupo de pesquisadores da Universidade Reva (KL *et al.*, 2021) e o aplicativo de mobilidade urbana que realiza o registro de vagas em estacionamentos privados (MELO; SILVA; SANTOS, 2022).

2.6.1 Backend: Node.js

O *Node.js* é um ambiente de execução de código aberto e multiplataforma que permite a interpretação e execução de aplicações escritas em JavaScript fora do navegador. Baseado no motor *V8*, desenvolvido pelo Google, o Node.js adota um modelo de execução assíncrono e orientado a eventos, o que o torna especialmente adequado para aplicações que demandam alto desempenho e grande volume de conexões simultâneas, como serviços web e *APIs REST* (NODE.JS, 2024).

Uma das principais características do Node.js é o seu modelo de *I/O* não bloqueante, que possibilita a execução eficiente de múltiplas requisições concorrentes utilizando uma única *thread*. Essa abordagem reduz o consumo de recursos computacionais e favorece a escalabilidade do sistema, sendo amplamente adotada em arquiteturas modernas de aplicações distribuídas (TILKOV; VINOSKI, 2010).

No contexto deste trabalho, o Node.js foi empregado como base para o desenvolvimento do *backend* da aplicação, sendo responsável pela comunicação com o banco de dados, pela implementação das regras de negócio e pela disponibilização das interfaces de programação utilizadas pelo *frontend* e por sistemas externos.

Bibliotecas utilizadas

Para viabilizar o desenvolvimento do *backend*, foram utilizadas as seguintes bibliotecas:

- **Express**: *framework* minimalista para Node.js que simplifica a criação de servidores web e APIs. Ele fornece mecanismos para definição de rotas, tratamento de requisições HTTP e organização modular da aplicação, sendo amplamente utilizado no desenvolvimento de serviços RESTful (EXPRESS.JS, 2024);
- **CORS**: biblioteca responsável por gerenciar políticas de *Cross-Origin Resource Sharing*, permitindo o controle de acesso a recursos da API por aplicações hospedadas em domínios

distintos. Sua utilização é fundamental para garantir a comunicação segura entre o *frontend* e o *backend* (CORS, 2024);

- **Dotenv**: utilizada para o gerenciamento de variáveis de ambiente, possibilitando a separação entre configurações sensíveis, como credenciais de banco de dados e chaves criptográficas, e o código-fonte da aplicação. Essa prática contribui para a segurança e portabilidade do sistema (DOTENV, 2024);
- **Nodemon**: ferramenta de apoio ao desenvolvimento que monitora alterações nos arquivos do projeto e reinicia automaticamente o servidor. Seu uso agiliza o processo de desenvolvimento e testes, sem impactar o ambiente de produção (NODEMON, 2024);
- **MySQL2**: biblioteca que implementa o protocolo de comunicação com bancos de dados MySQL e MariaDB, oferecendo suporte a operações assíncronas e *prepared statements*. Essa biblioteca foi empregada para realizar a integração entre o *backend* e o banco de dados relacional do sistema (MYSQL2, 2024);
- **JSON Web Token (JWT)**: biblioteca utilizada para geração e validação de tokens de autenticação no padrão JSON Web Token. Essa abordagem permite a implementação de mecanismos de autenticação *stateless*, reduzindo a necessidade de armazenamento de sessões no servidor (IETF, 2015);
- **Bcrypt**: biblioteca destinada à criptografia de senhas, baseada em funções de derivação de chave adaptativas. Seu uso aumenta a segurança do sistema ao dificultar ataques de força bruta e engenharia reversa sobre credenciais armazenadas (PROVOS; MAZIERES, 1999).

2.6.2 Frontend: Vue.js

O *Vue.js* é um *framework* JavaScript progressivo voltado ao desenvolvimento de interfaces de usuário reativas e baseadas em componentes. Sua arquitetura facilita a separação entre lógica, estrutura e apresentação, permitindo a construção de aplicações modulares, organizadas e de fácil manutenção. O sistema de reatividade do Vue.js garante que alterações no estado da aplicação sejam automaticamente refletidas na interface, reduzindo a complexidade do código e aumentando a previsibilidade do comportamento da aplicação (VUE.JS, 2024).

No contexto deste trabalho, o *frontend* desempenha um papel secundário, sendo utilizado principalmente como suporte visual para validação e demonstração das funcionalidades do *backend*. Ainda assim, a utilização do Vue.js contribuiu para uma implementação mais ágil das interfaces e para uma melhor compreensão do fluxo de dados e das interações do sistema.

Bibliotecas utilizadas

- **Vue**: biblioteca principal do *framework*, responsável pelo sistema de reatividade, pelo modelo de componentes e pela renderização da interface do usuário;

- **Vue Router**: biblioteca oficial para gerenciamento de rotas em aplicações do tipo *Single Page Application* (SPA), permitindo a navegação entre diferentes telas e o controle de acesso a rotas específicas (ROUTER, 2024);
- **Pinia**: gerenciador de estado recomendado para o Vue.js, utilizado para centralizar e compartilhar dados entre componentes de forma organizada, previsível e com boa integração ao uso de tipagem estática (PINIA, 2024);
- **Vite**: ferramenta de desenvolvimento e empacotamento que oferece inicialização rápida do projeto, recarregamento automático durante o desenvolvimento e geração otimizada dos arquivos finais para produção (VITE, 2024);
- **TypeScript**: extensão do JavaScript que adiciona tipagem estática ao código, auxiliando na detecção antecipada de erros, na documentação implícita do sistema e na manutenção do código ao longo do tempo (Microsoft, 2024).

Mesmo tratando-se de um *frontend* de apoio, essas ferramentas permitem criar uma interface clara e confiável para validar fluxos do sistema e facilitar a demonstração das funcionalidades implementadas no *backend*.

2.6.3 Sistema de Autenticação

Para possibilitar a associação entre a utilização dos recursos disponíveis e um usuário específico, bem como registrar eventos de início e término de uso, torna-se necessário um sistema de autenticação integrado à aplicação. Esse mecanismo permite identificar quem está utilizando determinado recurso e em quais condições, servindo de base para o controle operacional e para os processos de avaliação de comportamento previstos no sistema.

Um aspecto importante a ser destacado é que este módulo deve ser o mais adaptável possível. Considerando que muitas instituições já dispõem de soluções próprias de autenticação ou controle de acesso, o sistema proposto foi concebido para se adequar a diferentes cenários, podendo integrar-se a tecnologias já existentes ou operar de forma independente, sem comprometer o funcionamento dos demais módulos da aplicação.

Com esse objetivo, foi desenvolvido um protótipo didático para representar a etapa de autenticação, priorizando simplicidade de implementação e baixo custo. O protótipo tem a função de identificar o usuário e comunicar essa informação ao *backend*, permitindo que as regras de negócio do sistema sejam aplicadas de forma centralizada.

A identificação por radiofrequência (RFID) mostrou-se uma alternativa adequada para esse protótipo, em função de sua confiabilidade, alcance e custo reduzido (KL *et al.*, 2021; PALA; INANC, 2007). Outras tecnologias poderiam ser empregadas para a mesma finalidade, como reconhecimento de imagem, códigos visuais, diferentes tipos de *tags* (GPS ou Bluetooth) ou

leitores biométricos. Entretanto, essas alternativas geralmente apresentam maior complexidade de implementação, custos mais elevados ou requisitos de infraestrutura menos compatíveis com o escopo deste trabalho.

2.6.4 HTTP

O *Hypertext Transfer Protocol* (HTTP) é um protocolo de aplicação amplamente utilizado para comunicação cliente-servidor em aplicações web. Baseado no modelo requisição–resposta, o HTTP define métodos (por exemplo, `GET`, `POST`, `PUT`, `DELETE`) para manipulação de recursos identificados por URIs, e é especificado por normas da IETF. Em aplicações RESTful, o HTTP é comumente empregado para exposição de APIs que permitem operações CRUD sobre recursos remotos (FIELDING; RESCHKE, 2014).

No contexto deste trabalho, o HTTP é utilizado como um dos canais para o envio pontual de informações de estado ao *backend*. Fontes externas enviam requisições do tipo `POST` a endpoints REST, entregando um identificador de localidade e uma representação JSON da matriz de status associada às unidades dessa localidade. Entre as vantagens do uso de HTTP destacam-se a ampla compatibilidade com ferramentas e bibliotecas, o suporte nativo a TLS/HTTPS para confidencialidade e integridade e a facilidade de integração com serviços web existentes. Em contrapartida, o modelo request/response pode gerar overhead em cenários de atualização de alta frequência, exigindo cuidados quanto à latência e ao dimensionamento do servidor.

Boas práticas recomendadas incluem a utilização de HTTPS, autenticação e autorização de clientes (por exemplo, tokens JWT ou chaves de API), validação e sanitização de payloads no servidor e mecanismos de controle de carga (rate limiting, backpressure).

2.6.5 MQTT

O *Message Queuing Telemetry Transport* (MQTT) é um protocolo de mensagens leve baseado no padrão publish/subscribe, projetado para telemetria e dispositivos com recursos limitados. Normalmente opera sobre TCP/IP e dispõe de mecanismos de Quality of Service (QoS) que permitem ajustar garantias de entrega. O MQTT é apropriado para cenários de comunicação assíncrona e em tempo real, sendo padronizado por organizações como OASIS (BANKS; GUPTA, 2014).

Neste trabalho, o MQTT foi adotado como alternativa para recebimento contínuo de atualizações de status provenientes de dispositivos. Nesse modelo, cada localidade pode dispor de um tópico próprio onde publicações contendo a matriz de estado são enviadas; o *backend* subscreve os tópicos relevantes e processa as mensagens de forma reativa, atualizando o banco de dados conforme necessário. Vantagens do MQTT incluem baixo overhead por mensagem, eficiência em redes com largura de banda limitada e suporte a comunicação bidirecional entre dispositivos e servidor.

2.6.6 ESP32

Para a simulação prática da interação entre o sistema e dispositivos externos foi desenvolvido um protótipo didático baseado na placa ESP32. O ESP32 é um microcontrolador de baixo custo, com conectividade Wi-Fi e Bluetooth integrada, suficiente poder de processamento para tarefas de rede e suporte a periféricos típicos de prototipagem (UART, SPI, I^2C, GPIO), o que o torna adequado para demonstradores que emitem atualizações de status e replicam o comportamento de sensores ou leitores físicos (Espressif Systems, n.d.).

No protótipo, o ESP32 assume as seguintes funções principais: (i) interfacear-se com um sensor ou leitor (por exemplo, leitor RFID/NFC) e obter um identificador único de credencial; (ii) compor mensagens contendo o identificador da localidade e a matriz de status das unidades; e (iii) transmitir essas mensagens ao *backend* por meio de HTTP (`POST`) ou publicações MQTT em tópicos predefinidos (Espressif Systems, n.d.).

O uso do ESP32 como demonstrador permite replicar cenários reais com baixo custo e tempo de desenvolvimento reduzido, possibilitando validar os fluxos de integração, as rotinas cíclicas do sistema e o comportamento das políticas de confirmação e encerramento sem depender de infraestrutura externa definitiva.

3 METODOLOGIA

3.1 Abordagem Metodológica

A metodologia adotada neste trabalho baseia-se nos princípios apresentados no Capítulo 2, porém foi adaptada à condição prática em que o desenvolvimento foi conduzido por um único desenvolvedor. Em situações usuais recomenda-se a participação ativa de múltiplos *stakeholders* para elicitação e validação de requisitos; contudo, por se tratar de uma demanda pessoal, as etapas de levantamento, priorização e validação foram realizadas por meio de exercícios controlados de interpretação de papéis e por documentação explícita de premissas e hipóteses de uso. Essa adaptação metodológica visa minimizar vieses e garantir coerência entre fundamentação teórica e decisões técnicas, conforme discutido por Valente (2020), Pressman e Maxim (2016).

A opção por manter um processo incremental e orientado a funcionalidades permaneceu inalterada, pois favorece a entrega contínua de valor e reduz o risco técnico. No entanto, as práticas formais de elicitação (entrevistas e *workshops*) foram substituídas por reflexões com base em interpretação de papéis, nas quais o autor representou distintos perfis de usuário (administrador, usuário comum, operador de portaria) e documentou requisitos, cenários de uso e critérios de aceitação correspondentes, de acordo com o que julgou-se vantajoso exclusivamente pela parte interpretada no momento. Essa estratégia procurou reproduzir os benefícios da interação com múltiplos atores, preservando rastreabilidade entre requisitos e implementações.

3.2 Aplicação da Engenharia de Requisitos

A aplicação prática da engenharia de requisitos neste projeto seguiu as fases clássicas de elicitação, análise, especificação, validação e priorização (SOMMERVILLE, 2011; PRESSMAN; MAXIM, 2016; VALENTE, 2020). Em termos operacionais, as atividades realizadas incluíram:

- **Elicitação:** Diálogo interno entre os papéis de *stakeholders*, análise funcional e prototipagem de baixa fidelidade para validar hipóteses de uso e identificar cenários concretos de operação;
- **Análise:** classificação dos requisitos em funcionais e não funcionais, identificação de conflitos e definição de critérios de aceitação mensuráveis para cada requisito;
- **Priorização:** aplicação de critérios de valor de negócio, risco técnico e esforço estimado para ordenar o backlog de funcionalidades; priorizou-se primeiro a infraestrutura (banco de dados e APIs) e depois os recursos de orquestração (autenticação, agendamento, reputação) e por fim a apresentação (*frontend*);
- **Rastreabilidade:** manutenção de vínculo entre cada etapa de desenvolvimento e os requisitos originais, permitindo verificar que cada entrega atende aos critérios de aceitação definidos.

Essa sequência metodológica garantiu que as etapas de desenvolvimento que serão apresentadas posteriormente neste capítulo derivassem diretamente dos requisitos levantados, assegurando coerência entre fundamentação teórica e implementação prática.

3.3 Estratégia de Desenvolvimento Incremental e Modular

A estratégia de desenvolvimento combinou iterações curtas (*sprints*) com entrega incremental de funcionalidades agrupadas em etapas. Cada etapa possui critérios de aceitação e é acompanhada de testes unitários e de integração que comprovam o atendimento aos requisitos definidos.

Principais diretrizes adotadas:

- **Modularização**: o sistema foi dividido em módulos lógicos (usuários, localidades, agendamentos, avaliações, monitoramento) e em serviços reutilizáveis dentro do *backend*, favorecendo testes isolados e implantação incremental;
- **Critérios de aceitação**: cada etapa só é considerada concluída quando passam os testes e os critérios de aceitação definidos durante a fase de requisitos;
- **Iterações de validação**: entregas regulares foram apresentadas a *stakeholders* intermediários para coleta de feedback e priorização de ajustes nas iterações seguintes.

Essa estratégia permitiu controlar a complexidade do projeto, manter rastreabilidade entre requisitos e implementações e adaptar o escopo conforme necessidades identificadas durante o desenvolvimento.

3.4 Instrumentos e Artefatos de Suporte

Dada a natureza singular do processo, foram gerados artefatos adicionais destinados a documentar e sustentar as decisões tomadas:

- **Decision log**: registro cronológico das escolhas de projeto, motivação e alternativas consideradas;
- **Registro de premissas e restrições**: documento com todas as hipóteses adotadas (por exemplo, configuração de rede, uso de broker MQTT público para testes), incluindo data e justificativa;
- **Checklists de validação**: listas de verificação para assegurar que cada etapa atende aos critérios de aceitação antes da integração;

- **Provas de conceito e scripts de teste**: repositório com POCs (ESP32, endpoints de ingestão) e scripts para popular banco de dados com dados de teste.

Esses artefatos foram concebidos para fornecer transparência metodológica e permitir que avaliadores ou leitores reproduzam e avaliem as decisões adotadas, apesar da ausência de múltiplos *stakeholders*.

3.5 Limitações e Estratégias de Mitigação

Reconhecendo as limitações inerentes ao desenvolvimento conduzido por uma única pessoa, o trabalho adotou medidas para reduzir impactos sobre validade e qualidade:

- **Limitação:** ausência de validação por múltiplos *stakeholders*;
  Mitigação: interpretação de multiplos papéis pelo autor, contrução de protótipos para simulação e comparação com literatura e soluções acadêmicas;
- **Limitação:** riscos de viés individual nas priorizações;
  Mitigação: uso de critérios explícitos (valor de negócio, esforço e risco técnico) e registro das justificativas nas tomadas de decisão;
- **Limitação:** cobertura de testes possivelmente inferior à de equipes dedicadas;
  Mitigação: Utilização de ferramentas de Inteligência Artificial para gerar massas de testes com maior variabilidade.

Tais medidas não substituem a validação por usuários reais, mas aumentam a robustez metodológica do trabalho e fornecem base racional para as escolhas efetuadas.

3.6 Definições Estruturais

O desenvolvimento dos códigos deve ser realizado de maneira estruturada e organizada a fim de obter um resultado mais eficiente, compreensível e conciso. Para isso foram pré-estabelecidas uma série de políticas e restrições, que foram aplicadas durante o desenvolvimento deste trabalho. Nesta seção, serão descritos esses procedimentos.

3.6.1 Banco de dados

Primeiramente, para que todos os demais sistemas possam funcionar e realizar a comunicação entre si, é necessário um banco de dados funcional para centralizar as informações do projeto. Optou-se por utilizar o *MariaDB*, que consiste em uma implementação de código aberto do *mySQL*, um modelo de banco relacional que atende perfeitamente as necessidades deste projeto.

Será necessário armazenar três tipos de informação:

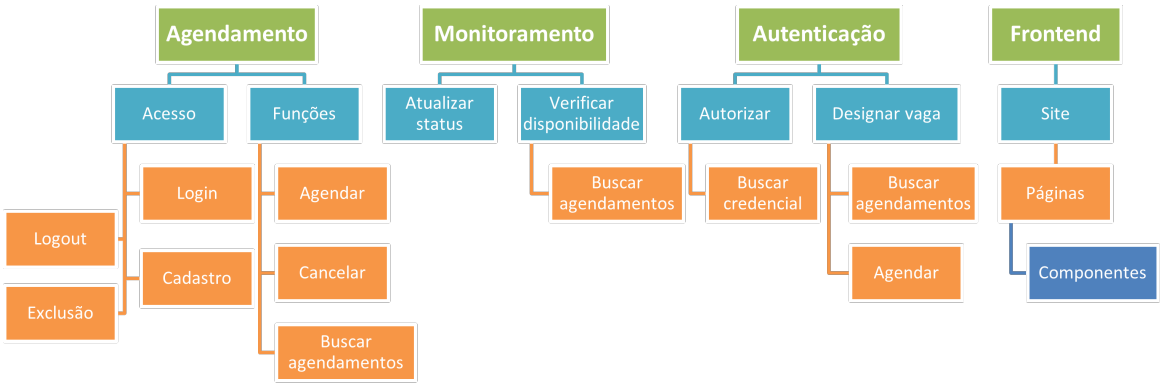
- Dados Cadastrais - Informações pessoais dos usuários e definições dos espaços físicos e recursos;
- Dados históricos - Registros de agendamentos, datas e horários de eventos realizados e avaliações;
- Status - Condição atual de cada recurso.

Por fim, será carregado um conjunto de dados de exemplo para realização dos testes dos demais sistemas.

3.6.2 Organização do código

Geralmente durante o processo de desenvolvimento de softwares é desejado que as aplicações sejam o mais objetivas e concisas o possível, contendo uma finalidade bem definida e um código claro. Para o desenvolvimento das aplicações deste trabalho foi aplicada esta lógica, como mostrado na estruturação proposta na Figura 2 abaixo.

Figura 2 – Subdivisão das funcionalidades básicas do sistema.



Fonte: Elaborado pelo autor, 2025.

O *backend* da aplicação foi subdividido em funcionalidades que, por sua vez, foram divididas em micro-serviços que realizam uma função menor e mais simples. Na Figura 2 é ilustrada essa repartição. Como pode ser observado, um mesmo serviço pode ser aplicado para múltiplos Sistemas, implicando que o código deve ser reutilizável e modular.

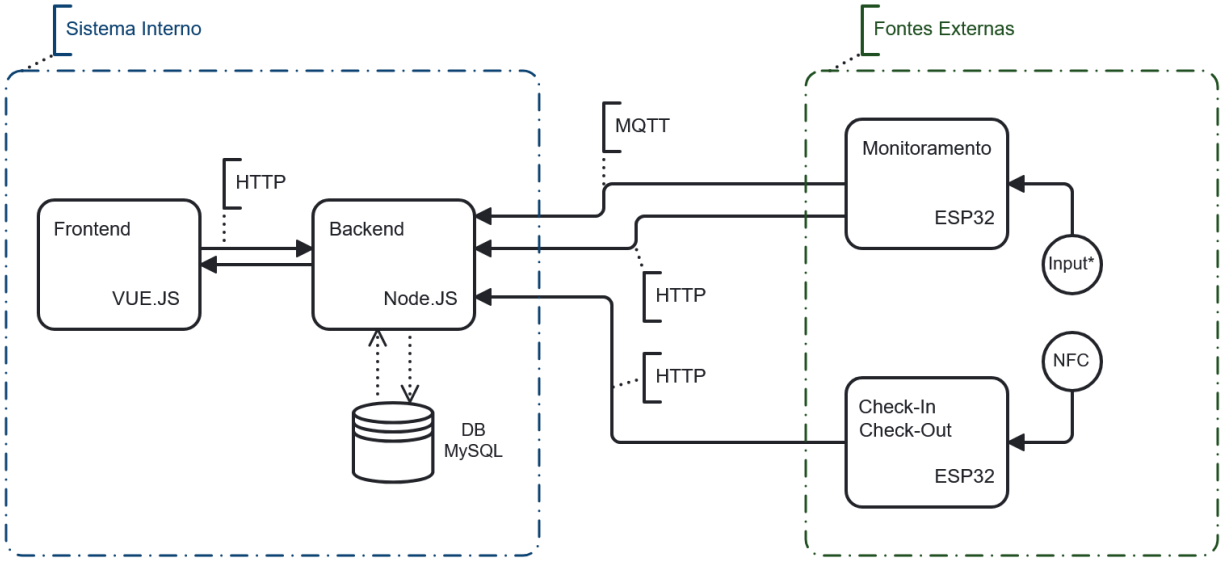
Para o *frontend* o procedimento será similar, porém seguindo a estrutura de um site. Primeiramente serão definidas as páginas necessárias e os usuários que terão acesso a elas, por exemplo administradores devem ter acesso a uma página de edição dos estacionamentos cadastrados, bem como um usuário de aplicação deverá ter acesso a uma página de agendamento especializada para a interação com pessoas na portaria. Em sequência, todos componentes de uma

página (cabeçalho, rodapé, botões, formulários, tabelas, gráficos, etc) serão construídos separadamente para serem importados e organizados nas páginas, também permitindo o reaproveitamento de cada componente.

Como ferramenta de desenvolvimento, optou-se por utilizar Node.js para construção do *backend* da aplicação e o Vue.js para o *frontend* pois, além de serem tecnologias amplamente utilizadas no mercado, já existe um conhecimento prévio delas por parte do autor, o que possibilita uma melhor distribuição do tempo dedicado às demais demandas do projeto.

Na Figura 3 é exemplificado como foi idealizada a integração entre os diversos componentes do sistema.

Figura 3 – Diagrama de comunicações entre Sistemas/Módulos [1]



Fonte: Elaborado pelo autor, 2025.

3.7 Frontend: Interface Básica

O desenvolvimento do *frontend* neste projeto possui caráter secundário e instrumental, não sendo tratado como o foco principal da solução proposta. Sua construção ocorre de forma paralela ao desenvolvimento do *backend*, com o objetivo principal de auxiliar na validação das funcionalidades implementadas, facilitar o processo de testes e fornecer uma interface visual que permita melhor compreensão do fluxo de dados e do funcionamento geral do sistema.

Dessa forma, o *frontend* não deve ser interpretado como uma interface final de produto, tampouco como uma solução voltada à experiência do usuário em um contexto de produção. As decisões de design e usabilidade foram tomadas de maneira pragmática, priorizando a rapidez de implementação e a clareza na visualização das operações realizadas pelo sistema.

[1] Input indica que a entrada de dados pode ser obtida por uma ampla variedade de métodos, tais como sensores físicos, câmeras de visão computacional ou outros dispositivos compatíveis com a infraestrutura disponível.

Em razão desse escopo reduzido, foi utilizado de forma intensiva o recurso do *GitHub Copilot* como ferramenta de apoio à geração de componentes visuais e estruturas básicas de interface. Essa abordagem permitiu a criação de telas funcionalmente adequadas e visualmente organizadas, sem comprometer o foco do trabalho, que está centrado na modelagem do sistema, nas regras de negócio e na comunicação entre os módulos da aplicação.

3.8 Desenvolvimento da Aplicação

O processo de desenvolvimento da aplicação será realizado de forma incremental, a partir da definição de funcionalidades bem delimitadas, organizadas em etapas de desenvolvimento. Cada etapa representa um conjunto coerente de comportamentos esperados do sistema, descrevendo de forma objetiva o que deve ser implementado e quais requisitos devem ser atendidos.

Essa abordagem permite dividir o sistema em partes menores e gerenciáveis, facilitando o planejamento, a implementação, os testes e a validação de cada funcionalidade de maneira isolada, antes de sua integração ao sistema como um todo. Dessa forma, reduz-se a complexidade do desenvolvimento e aumenta-se a rastreabilidade entre os requisitos definidos e as funcionalidades implementadas.

As etapas de desenvolvimento serão elaboradas com base nas necessidades dos diferentes perfis de usuários do sistema, como administradores, usuários comuns e operadores responsáveis pelo controle dos recursos físicos. A partir dessas etapas, são definidos os requisitos funcionais, que descrevem as ações que o sistema deve executar, e os requisitos não funcionais, que estabelecem restrições relacionadas a desempenho, segurança, usabilidade e manutenção.

Ao final da implementação de cada etapa, serão realizados testes para verificar se os requisitos definidos foram devidamente atendidos. Somente após essa validação a funcionalidade será considerada concluída e integrada às demais partes do sistema. Esse processo iterativo contribui para a qualidade do software, além de permitir ajustes e refinamentos ao longo do desenvolvimento.

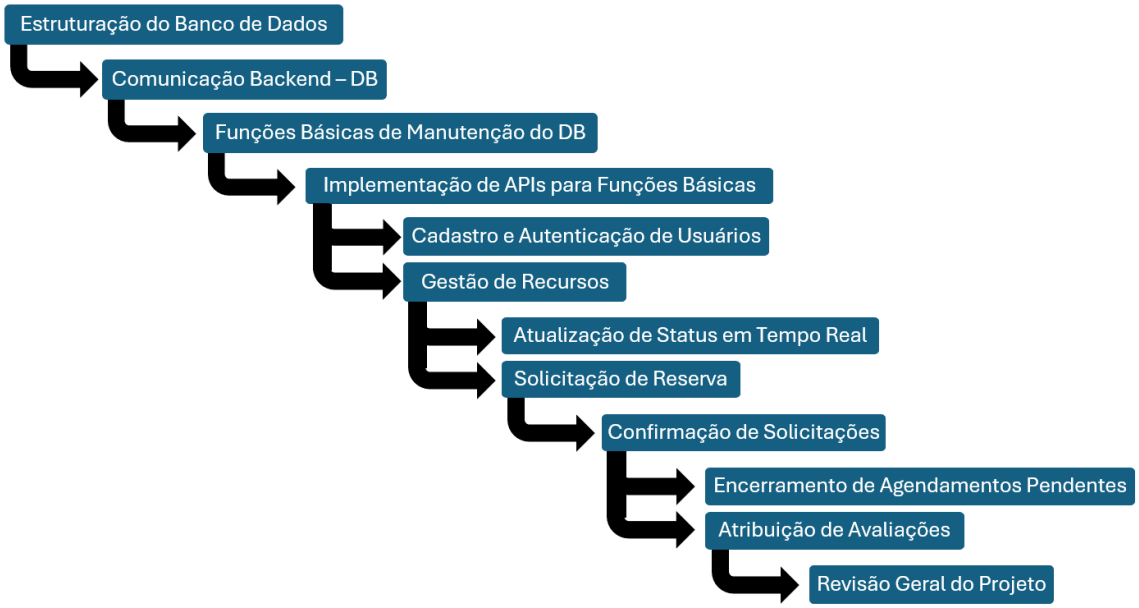
Na Figura 4 abaixo é representado em um diagrama de cascata as etapas do desenvolvimento e as suas respectivas dependências.

3.8.1 Estruturação do Banco de Dados

Esta etapa tem como objetivo a definição e implementação da estrutura inicial do banco de dados do sistema, a qual servirá como base para o desenvolvimento e integração dos demais módulos da aplicação. O banco de dados é responsável por centralizar e organizar as informações necessárias ao funcionamento do sistema, sendo, portanto, a primeira funcionalidade a ser desenvolvida no escopo do projeto.

Nesta etapa, tornou-se imprescindível estabelecer de forma clara as entidades, seus

Figura 4 – Diagrama em Cascata das Etapas de Desenvolvimento.



Fonte: Elaborado pelo autor, 2025.

atributos e os relacionamentos existentes, de modo a atender aos requisitos funcionais previamente definidos. A correta estruturação do banco de dados nesta etapa visa garantir a consistência das informações e viabilizar a implementação das funcionalidades previstas nas etapas subsequentes.

Foram definidas quatro entidades principais: *Usuário*, *Localidade*, *Agendamento* e *Avaliação*. Cada uma dessas entidades será implementada com um identificador único (*id*), utilizado como chave primária para assegurar a integridade e a rastreabilidade dos registros ao longo da operação do sistema.

A entidade *Usuário* será responsável por armazenar os dados pessoais dos usuários que utilizam a aplicação, incluindo informações de identificação e atributos necessários para autenticação e utilização dos serviços oferecidos. Trata-se de uma tabela de dados cadastrais, representando um recurso fundamental e relativamente estável ao longo do tempo.

Localidade terá como finalidade registrar as informações referentes aos espaços físicos disponibilizados para uso, como estacionamentos, salas ou outros recursos compartilhados. Serão armazenados características relevantes para o gerenciamento desses recursos, configurando também uma tabela de dados cadastrais que dará suporte às demais operações do sistema.

Agendamento corresponderá aos registros funcionais da aplicação, sendo responsável por associar um *Usuário* a uma determinada *Localidade* dentro de um intervalo de tempo previamente definido. Nessa entidade serão armazenadas informações como data, horário de início e término, além do estado do agendamento, indicando se a reserva foi atendida, cancelada ou não utilizada.

Por fim, a entidade *Avaliação* será utilizada para registrar o histórico de comportamento dos usuários no sistema. Esses registros servirão como base para o cálculo das métricas de reputação, podendo ser gerados automaticamente pelo próprio sistema, a partir do cumprimento

ou não dos horários agendados, ou inseridos manualmente por um administrador, conforme critérios definidos durante a operação da aplicação.

A estrutura conceitual definida nesta etapa foi planejada de modo a garantir consistência, flexibilidade e possibilidade de expansão futura, permitindo que novos requisitos sejam incorporados nas próximas etapas do desenvolvimento sem a necessidade de alterações estruturais significativas no banco de dados.

3.8.2 Comunicação *Backend* – Banco de Dados

Com o banco de dados previamente estruturado, esta etapa tem como objetivo estabelecer a comunicação entre a aplicação e o repositório central de dados por meio do *backend*. Nesta etapa, é iniciado o projeto responsável por intermediar as requisições dos demais módulos do sistema, garantindo o acesso controlado e consistente às informações armazenadas.

O escopo desta etapa concentra-se nas configurações iniciais do ambiente de desenvolvimento do *backend*, incluindo a criação do projeto, a definição de sua estrutura básica e a adição das dependências essenciais já discutidas no capítulo de Fundamentação Teórica. Também são realizadas as configurações necessárias para o correto gerenciamento de variáveis de ambiente, assegurando que credenciais e parâmetros sensíveis não sejam expostos diretamente no código-fonte.

Adicionalmente, são definidas as rotinas de inicialização do servidor e os módulos responsáveis pela conexão com o banco de dados. Ao final desta etapa, espera-se que a aplicação seja capaz de estabelecer uma conexão estável e bem-sucedida com o banco de dados, permitindo a execução de operações básicas de acesso, que servirão como base para a implementação das funcionalidades nas fases subsequentes.

Essa etapa é essencial para validar a infraestrutura da aplicação e garantir que o desenvolvimento das próximas funcionalidades possa ocorrer de forma incremental e controlada.

3.8.3 Funções Básicas de Manutenção das Tabelas

Antes da implementação de funcionalidades de maior complexidade, é necessário assegurar que o sistema seja capaz de executar operações fundamentais sobre cada uma das entidades definidas no banco de dados. Dessa forma, esta etapa tem como objetivo a implementação das funções básicas de manutenção das tabelas, garantindo o gerenciamento adequado dos dados ao longo de todo o ciclo de vida da aplicação.

Para cada uma das tabelas criadas, devem ser desenvolvidas funções responsáveis pelas operações de criação, atualização e exclusão de registros. Essas rotinas devem seguir o padrão de modularização adotado no *backend*, contribuindo para a clareza do código, facilitando a manutenção e permitindo o reaproveitamento das funcionalidades em diferentes contextos do sistema.

A implementação dessas operações básicas é essencial para validar a integridade das entidades definidas e assegurar que o sistema possua os mecanismos mínimos necessários para a manipulação dos dados de forma consistente e controlada.

3.8.4 Implementação de APIs para Funções Básicas

Além das operações de escrita, o sistema deve disponibilizar interfaces de programação (*APIs*) para a consulta dos dados armazenados. Essas funções de leitura devem permitir tanto a recuperação completa dos registros de uma entidade quanto consultas específicas, filtradas a partir de campos relevantes, como chaves primárias ou secundárias.

Esse mecanismo possibilita, por exemplo, a obtenção das informações de um usuário específico, a listagem de agendamentos associados a uma determinada localidade ou a recuperação de registros dentro de um intervalo de datas previamente definido. Dessa forma, o sistema oferece flexibilidade na consulta dos dados, atendendo às diferentes necessidades dos módulos consumidores da aplicação.

Essas operações poderão ser executadas remotamente por usuários ou sistemas que possuam as permissões adequadas. Para isso, torna-se necessária a definição de um ponto de escuta padrão para a aplicação, bem como a estruturação de rotas que direcionem corretamente as requisições recebidas às funções correspondentes. O tratamento das rotas deve considerar o tipo de requisição realizada, assegurando que cada operação seja acionada de forma controlada e em conformidade com as regras de acesso estabelecidas.

3.8.5 Cadastro e Autenticação de Usuários

Para que o sistema seja capaz de atender às demais demandas funcionais, é imprescindível a implementação de um mecanismo seguro e confiável de cadastro e autenticação de usuários. Dessa forma, esta etapa tem como objetivo definir e implementar os requisitos funcionais e não funcionais relacionados à criação de contas, validação de credenciais e controle de acesso à aplicação.

Requisitos Funcionais

Os seguintes requisitos funcionais devem ser atendidos para a conclusão desta etapa:

- Permitir o cadastro de novos usuários no sistema por meio de uma requisição do tipo *POST*,;
- Validar os dados fornecidos durante o processo de cadastro, verificando formatos, obrigatoriedade de campos e a inexistência de registros duplicados, como contas associadas a um mesmo endereço de e-mail;

- Diferenciar usuários comuns e usuários administradores, possibilitando a aplicação de regras distintas de acesso às funcionalidades do sistema;
- Permitir a autenticação dos usuários a partir da validação das credenciais informadas no processo de login;
- Gerar um *token* de autenticação após a validação bem-sucedida das credenciais, possibilitando a identificação do usuário em requisições subsequentes sem a necessidade de reenvio contínuo das informações de login.

Requisitos Não Funcionais

Além das funcionalidades descritas, esta etapa deve atender aos seguintes requisitos não funcionais:

- As credenciais de acesso devem ser armazenadas de forma segura, utilizando algoritmos de criptografia adequados para senhas, evitando o armazenamento de informações sensíveis em texto puro;
- As respostas às requisições de cadastro e autenticação devem ser claras e padronizadas, facilitando o tratamento de erros pelas aplicações clientes;
- O controle de acesso deve ser flexível, possibilitando a expansão futura do conjunto de permissões e perfis de usuário.

A implementação desta etapa estabelece a base de segurança e controle de acesso do sistema, sendo essencial para garantir que as funcionalidades subsequentes sejam utilizadas apenas por usuários devidamente autenticados e autorizados, de acordo com as políticas definidas.

3.8.6 Gestão de Recursos

De forma análoga ao cadastro de usuários, o sistema deve disponibilizar funcionalidades que permitam a gestão dos recursos compartilhados cadastrados. Esta etapa tem como objetivo implementar os mecanismos necessários para que usuários com perfil administrativo possam criar, editar e excluir recursos, representados pela entidade *Localidade*.

No que se refere aos requisitos funcionais, o sistema deve permitir o cadastro de novas localidades, bem como a atualização ou remoção de registros já existentes, garantindo que essas operações sejam restritas a usuários devidamente autenticados e autorizados. Também deve ser possível consultar os recursos cadastrados, de modo a apoiar as atividades de administração e validação das informações.

Quanto aos requisitos não funcionais, o sistema deve assegurar a integridade dos dados durante operações de edição ou exclusão, evitando conflitos com agendamentos previamente

registrados. Além disso, na criação de novos recursos, deve ser realizada a validação das informações mínimas necessárias para o correto funcionamento das funcionalidades de agendamento. A solução deve ainda ser projetada de forma flexível, permitindo a expansão futura do modelo de recursos sem a necessidade de alterações estruturais significativas.

3.8.7 Solicitação de Reserva

A funcionalidade de solicitação de reserva constitui o núcleo operacional do sistema, sendo responsável por permitir que os usuários realizem o agendamento dos recursos disponíveis. Para que esta etapa atenda adequadamente ao seu objetivo, foram definidos os requisitos, conforme descrito a seguir.

Requisitos Funcionais

- Disponibilizar ao usuário uma lista dos tipos de localidade cadastrados no sistema;
- Retornar a lista de localidades disponíveis a partir da seleção de um tipo específico;
- Exibir a disponibilidade dos recursos com base em uma data e horário informados pelo usuário;
- Disponibilizar um método para a criação de registros na tabela de agendamentos.

Requisitos Não Funcionais

- Validar a existência de conflitos de agenda antes da efetivação do registro, impedindo a criação de reservas sobrepostas para um mesmo recurso.
- Assegurar que o processo de criação de reservas seja executado de forma controlada e confiável, evitando inconsistências nos dados armazenados.

3.8.8 Atualização de Status em Tempo Real

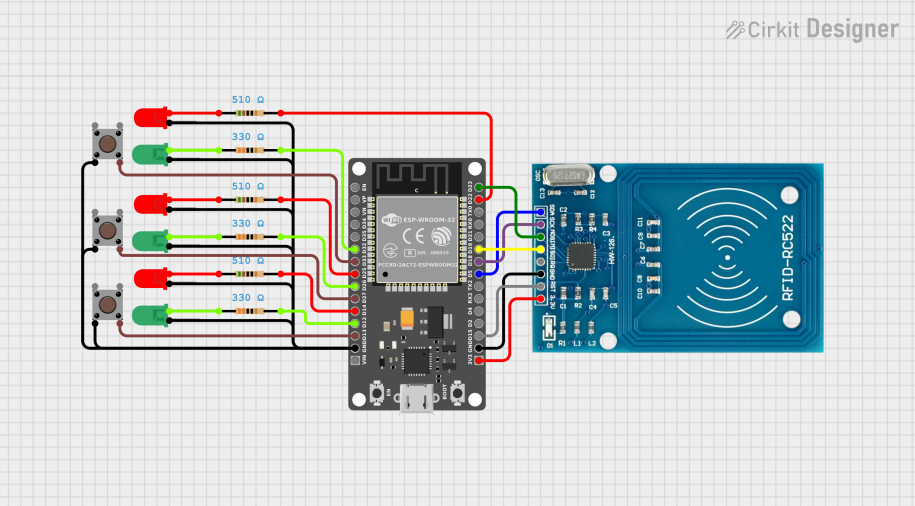
Um diferencial funcional do sistema é a capacidade de receber e exibir informações de status dos recursos em tempo real, uma funcionalidade especialmente relevante para cenários como estacionamentos, embora aplicável a outros tipos de recursos. Esta etapa tem como objetivo tratar exclusivamente da recepção, processamento e disponibilização dessas informações ao usuário final.

Os mecanismos responsáveis pela obtenção dos dados de status — como sensores físicos ou sistemas de visão computacional — estão fora do escopo deste projeto. Dessa forma, o sistema deve apenas consumir informações provenientes de fontes externas previamente configuradas, sem interferir no método de coleta.

O sistema deve, portanto, disponibilizar meios para o recebimento das informações de status por meio dos protocolos HTTP e MQTT. As mensagens recebidas devem conter um identificador da localidade e um conjunto de dados em formato de array representando o estado de cada vaga ou recurso associado. A partir dessas informações, o sistema deve atualizar o estado dos recursos no banco de dados e disponibilizar os dados atualizados para visualização pelos usuários.

Para validar o devido funcionamento desta etapa, será utilizado um protótipo construído com um microcontrolador ESP32 para simular os sinais que se esperar receber das localidades físicas. Na Figura 5 a seguir é demonstrado o esquemático proposto para este protótipo, que possuirá 3 vagas simuladas, cada uma contendo LEDs para indicar o estado e um botão para alternar esse estado.

Figura 5 – Protótipo para simulação de monitoramento



Fonte: Elaborado pelo autor, 2025.

3.8.9 Confirmação de Solicitações

O próximo objetivo é definir o processo de confirmação automática das solicitações de agendamento, com base em parâmetros configuráveis e no histórico de reputação dos usuários. Para isso, o sistema deverá executar uma rotina periódica, em intervalo ajustável por administradores, responsável por avaliar os agendamentos que ainda não tenham sido confirmados ou cancelados.

Do ponto de vista funcional, a rotina deverá analisar cada solicitação pendente, considerando o coeficiente numérico de reputação do usuário responsável, representado por um valor real no intervalo de 1 a 5. Com base nesse valor, os usuários serão classificados em três faixas de reputação — baixa, média e alta — que determinarão se a solicitação pode ser confirmada imediatamente ou se deverá aguardar um tempo mínimo adicional. Os valores limiares que definem cada faixa, assim como os tempos mínimos de espera associados, deverão ser configuráveis por usuários administradores.

Em relação aos requisitos não funcionais, o sistema deve permitir o ajuste da frequência de execução da rotina de verificação, de modo a possibilitar maior controle sobre o desempenho e o consumo de recursos. Essa abordagem permite automatizar o processo de confirmação das reservas, ao mesmo tempo em que incorpora critérios de reputação e flexibilidade administrativa, contribuindo para um uso mais equilibrado e confiável dos recursos do sistema.

3.8.10 Encerramento de Agendamentos Pendentes e Atribuição de Avaliações

Para concluir o ciclo de vida de cada agendamento, torna-se necessário o encerramento formal do registro, bem como o armazenamento das informações referentes ao horário efetivo de término e à respectiva avaliação do usuário. Esta etapa tem como objetivo garantir a correta finalização dos agendamentos e a consolidação dos dados utilizados para os cálculos de reputação.

O sistema deverá disponibilizar uma função responsável por encerrar um agendamento e atribuir automaticamente uma avaliação com base no único critério objetivo disponível, que corresponde à diferença entre os horários previamente agendados e os horários efetivamente cumpridos. Essa abordagem permite uma avaliação padronizada e isenta de intervenção manual para os casos regulares de utilização.

Além disso, deverá ser implementada uma rotina periódica capaz de identificar agendamentos pendentes cujo período de validade já tenha sido ultrapassado. Nesses casos, o sistema deverá realizar o encerramento automático do registro e atribuir uma avaliação negativa, caracterizando o não cumprimento do agendamento por parte do usuário.

Por fim, o sistema deverá oferecer um mecanismo para a atribuição manual de avaliações por usuários administradores. Essa funcionalidade permitirá a consideração de fatores externos ao controle automático do sistema, como comportamento inadequado, uso indevido ou danos aos recursos disponibilizados, complementando assim o modelo de reputação adotado.

3.9 Recursos

A seguir, estão listados os recursos que serão necessários durante o desenvolvimento deste trabalho:

- Computador com acesso a ferramentas de desenvolvimento. - Sem custo adicional;
- Computador para execução das aplicações. - Sem custo adicional;
- Leitor RFID - aproximadamente R$ 35,00;
- Microcontrolador *esp-32*. - aproximadamente R$ 40,00.

4 RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados obtidos com o desenvolvimento do sistema proposto e discute sua adequação aos objetivos e requisitos definidos ao longo do trabalho. A análise concentra-se na validação funcional da aplicação e na avaliação das decisões arquiteturais adotadas, considerando as limitações de escopo e tempo do projeto.

Os resultados são discutidos de forma alinhada às etapas implementadas, permitindo avaliar a evolução incremental do sistema e a contribuição de cada funcionalidade para a consolidação da solução. São abordados aspectos relacionados à organização do banco de dados, à comunicação entre os módulos, à implementação das regras de negócio e à integração com fontes externas de informação.

Além dos aspectos funcionais, são considerados fatores não funcionais relevantes, como modularidade, flexibilidade e a possibilidade de expansão futura. Por fim, o capítulo relaciona os resultados obtidos com o problema proposto, destacando as limitações identificadas e apontando oportunidades de aprimoramento para trabalhos futuros.

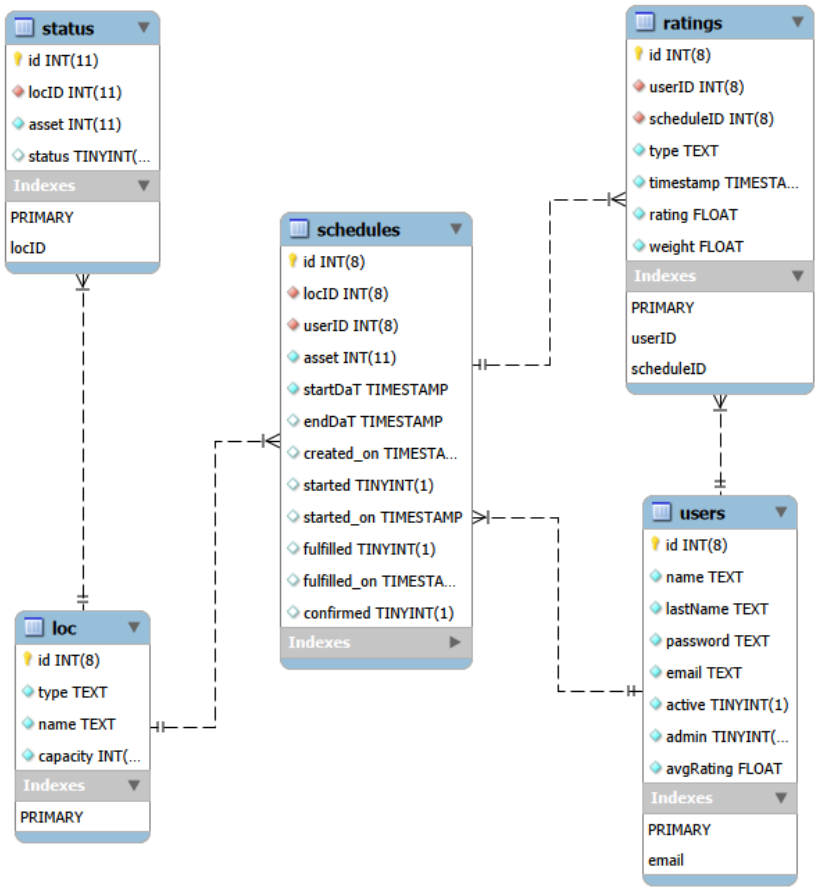
Os códigos-fonte desenvolvidos neste trabalho foram disponibilizados em repositórios públicos na plataforma GitHub, podendo ser acessados por meio de (VASCONCELOS, 2026a) e (VASCONCELOS, 2026b).

4.1 Implementação do Banco de Dados

O esquema físico do banco de dados foi implementado utilizando o sistema gerenciador MariaDB e contempla quatro tabelas principais: `users`, `loc`, `schedules` e `ratings`. A modelagem adotada atende aos requisitos funcionais definidos no escopo do sistema, garantindo integridade referencial, organização dos dados e suporte às funcionalidades de agendamento e avaliação de usuários.

Na Figura 6 a seguir, é apresentado o Diagrama Entidade Relacionamento do banco construído mostrando todos os campos e estruturas definidos e as relações entre eles.

Figura 6 – Diagrama Entidade Relacionamento



Fonte: Elaborado pelo autor, 2025.

No Apêndice A, apresenta-se a descrição completa das tabelas, de seus campos e das decisões de implementação consideradas relevantes para o funcionamento do sistema.

4.1.1 Relacionamentos

Os relacionamentos entre as tabelas foram definidos a partir dos campos identificadores, utilizando chaves primárias e estrangeiras para garantir a integridade referencial do banco de dados. Todas as relações estabelecidas são do tipo um-para-muitos (1:n), configuração que atende aos requisitos funcionais do sistema.

Foram definidos os seguintes relacionamentos principais:

- Cada registro de *agendamento* está associado a um único usuário e a uma única localidade, enquanto um mesmo usuário ou localidade pode possuir múltiplos agendamentos ao longo do tempo;

- Cada registro de *avaliação* está associado a um único agendamento e a um único usuário, permitindo que um usuário possua diversas avaliações vinculadas a diferentes utilizações do sistema;
- Cada registro de *status* está associado a uma única localidade.

Embora a associação direta entre as tabelas `ratings` e `users` não seja estritamente necessária, uma vez que a identificação do usuário já está presente no registro do agendamento, optou-se por manter esse relacionamento explícito. Essa decisão visa simplificar e otimizar consultas frequentes baseadas no usuário avaliado, reduzindo a necessidade de junções adicionais e contribuindo para um melhor desempenho e legibilidade das operações de consulta.

4.2 Funções de Manutenção de Tabelas

O *backend* da aplicação foi organizado em módulos independentes, cada um responsável pela manipulação de uma entidade específica do banco de dados. Essa abordagem visa manter o código estruturado, modular e de fácil manutenção.

Para cada entidade, foram implementadas funções assíncronas responsáveis pelas interações necessárias com o banco de dados. Embora, em nível conceitual, as operações de manutenção apresentem finalidades semelhantes entre as diferentes entidades, as particularidades estruturais de cada tabela exigem comandos SQL distintos, o que justifica a segregação das funções em módulos específicos.

No Apêndice B foram descritas em mais detalhes as funções, seus parâmetros de entrada e saídas. Subdividiu-se a explicação entre as funções que possuem um comportamento comum entre as tabelas e aquelas que possuem objetivos específicos para um processo.

4.3 APIs e Tratativas de Requisições

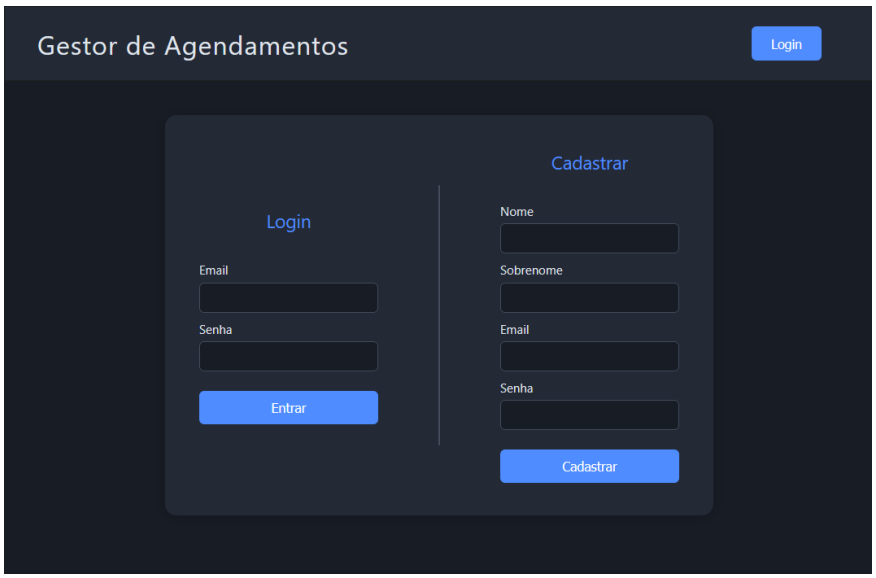
Para o recebimento, tratamento e resposta às requisições externas, foi estabelecido um subdiretório denominado *Handlers*, no qual cada módulo é responsável por tratar uma entidade específica do sistema, seguindo a mesma organização adotada anteriormente para os módulos de manipulação do banco de dados. Essa estrutura contribui para a modularidade do código e facilita a manutenção. Os códigos correspondentes a esses módulos podem ser consultados em (VASCONCELOS, 2026a).

Adicionalmente, foi implementado um arquivo `router.js` (cujas definições estão apresentadas no Apêndice C) responsável pelo roteamento das requisições recebidas. Esse módulo realiza o direcionamento adequado com base na URL acessada e no método HTTP utilizado, garantindo que cada solicitação seja encaminhada à função de tratamento correspondente.

4.3.1 Criação e Autenticação de Usuários

Conforme ilustrado na Figura 7, foram desenvolvidos dois formulários básicos para a realização do cadastro e da autenticação de usuários. Após a verificação do preenchimento e da validade das informações fornecidas, é enviada uma requisição do tipo POST ao *backend*, responsável por validar as credenciais informadas ou efetuar a criação de um novo registro de usuário.

Figura 7 – Tela de Login



Fonte: Elaborado pelo autor, 2025.

Em caso de autenticação bem-sucedida, além da confirmação do acesso, é gerado um token temporário que passa a ser utilizado nas requisições subsequentes. Essa abordagem elimina a necessidade do reenvio constante das credenciais, aumentando a segurança e a eficiência da comunicação entre o cliente e o servidor.

4.3.2 Gestão de Localidades, Recursos e Usuários

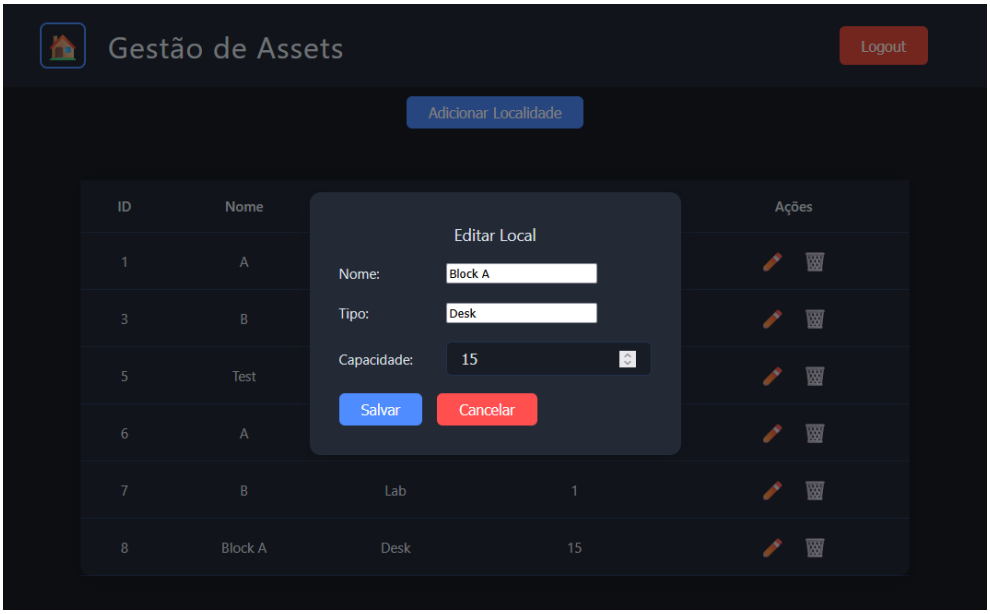
Usuários com permissões administrativas possuem acesso às funcionalidades de gestão tanto de localidades e recursos quanto de contas de usuários. Essas funcionalidades são implementadas a partir de interfaces que permitem a listagem dos registros existentes, a recuperação detalhada de um item específico por meio de seu identificador, bem como a edição ou exclusão de registros utilizando requisições do tipo PATCH e DELETE. Também é disponibilizada a opção de criação de novos registros.

As Figuras 8 e 9 apresentam exemplos das interfaces destinadas à gestão de localidades e recursos, enquanto as Figuras 10 e 11 ilustram as telas utilizadas para a administração de usuários do sistema.

Figura 8 – Tela de Gestão de Recursos

Fonte: Elaborado pelo autor, 2025.

Figura 9 – Edição de um registro de recurso

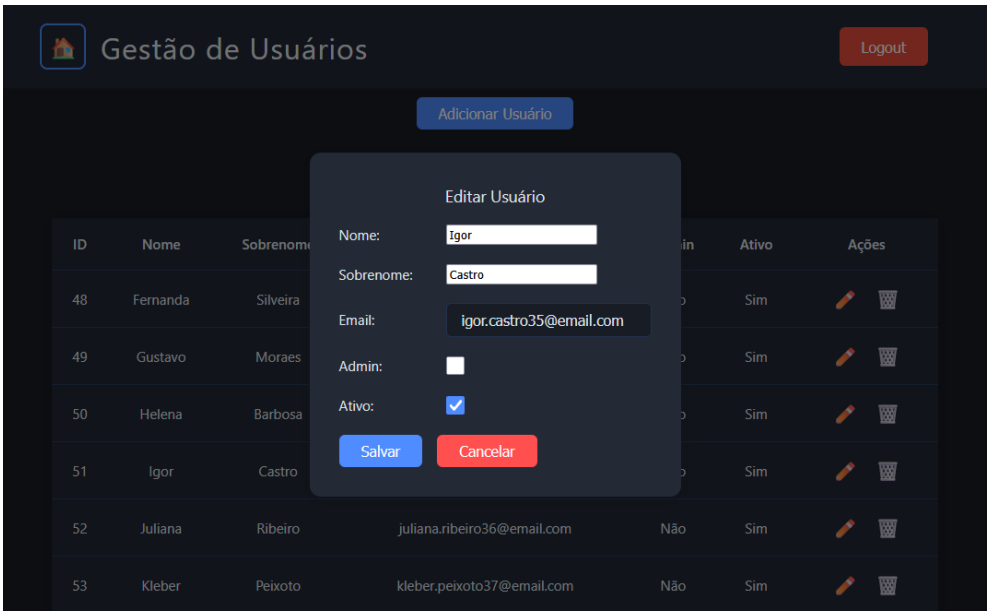


Fonte: Elaborado pelo autor, 2025.

Figura 10 – Tela de Gestão de Usuários

Fonte: Elaborado pelo autor, 2025.

Figura 11 – Edição de um registro de usuário

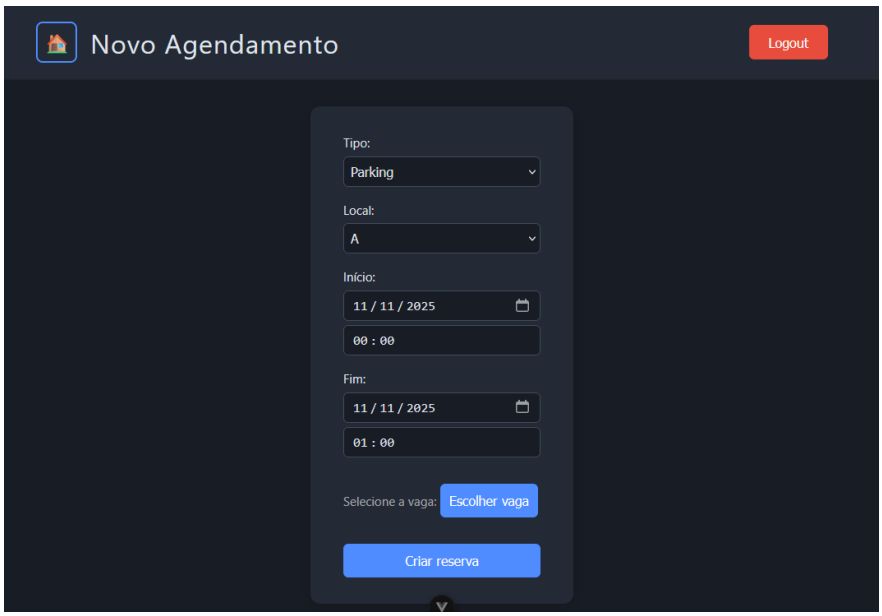


Fonte: Elaborado pelo autor, 2025.

4.3.3 Processo de Agendamento

Foi desenvolvido um formulário específico para a solicitação de agendamentos, por meio do qual o usuário informa a data, o horário desejado e seleciona o recurso a ser utilizado. No carregamento inicial do formulário, que pode ser visto na Figura 12 o sistema realiza automaticamente uma consulta aos tipos de localidade cadastrados, populando uma lista suspensa com as opções disponíveis. A partir da seleção de um tipo pelo usuário, é efetuada uma nova consulta ao banco de dados para obtenção das localidades associadas à categoria escolhida.

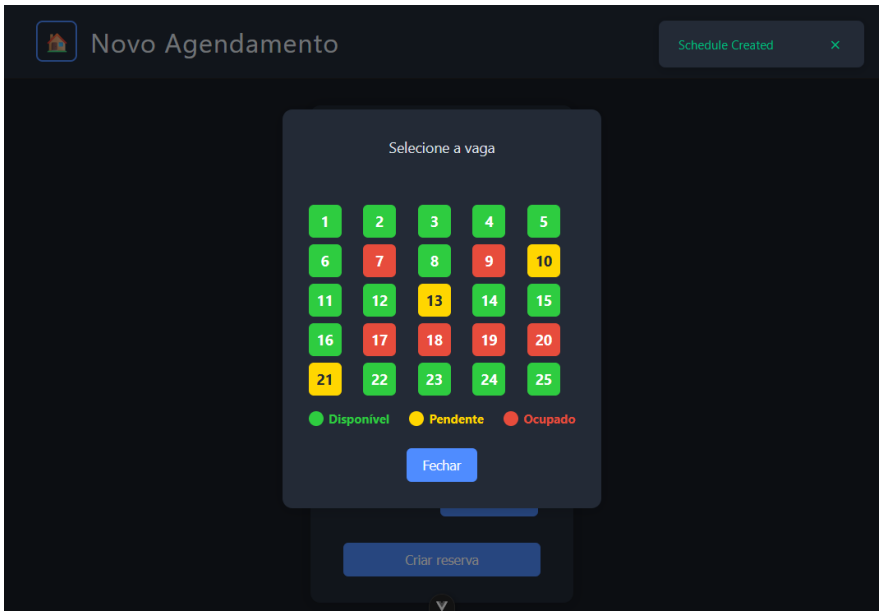
Figura 12 – Interface de solicitação de agendamento



Fonte: Elaborado pelo autor, 2025.

Após a definição da localidade, do horário inicial e do horário final, o sistema apresenta ao usuário a lista de recursos pertencentes à localidade selecionada, juntamente com a indicação de seu estado de disponibilidade no intervalo informado. Os recursos podem ser classificados como livres, pendentes ou ocupados, sendo que os recursos já ocupados não ficam disponíveis para novas solicitações de agendamento, conforme pode ser observado na Figura 13.

Figura 13 – Exibição da disponibilidade dos recursos



Fonte: Elaborado pelo autor, 2025.

Essa abordagem permite ao usuário visualizar previamente a situação dos recursos antes da confirmação da solicitação, contribuindo para a redução de conflitos de agenda e para uma

melhor experiência de uso do sistema.

4.3.4 Recebimento e Exibição de Status

Para a comunicação entre o *backend* da aplicação e as múltiplas e potenciais fontes de dados de monitoramento, optou-se pela utilização do protocolo MQTT (*Message Queuing Telemetry Transport*), em razão de suas principais vantagens, tais como baixo consumo de largura de banda, reduzida sobrecarga de comunicação, modelo assíncrono baseado em publicação/assinatura e elevada escalabilidade. Essas características tornam o protocolo especialmente adequado para cenários com dispositivos heterogêneos, recursos computacionais limitados e necessidade de atualização frequente de estados em tempo quase real.

Para fins de validação do funcionamento do sistema, foi utilizado o EMQX, um *broker* MQTT público, considerando que o volume de dados trafegado seria reduzido e que os requisitos de segurança não eram críticos nesse estágio do desenvolvimento. Ressalta-se, contudo, que a arquitetura do sistema foi projetada de forma a permitir a utilização de outros *brokers*, sendo necessária apenas a alteração da URL de conexão do cliente MQTT e do identificador do tópico ao qual o sistema realiza a inscrição. Em um ambiente produtivo, recomenda-se a utilização de um servidor dedicado à aplicação ou a contratação de um serviço de *broker* privado, de modo a garantir maior confiabilidade, disponibilidade e segurança das informações transmitidas.

Para a visualização dos dados recebidos, conforme ilustrado na Figura 14, foi desenvolvida uma interface no *frontend* da aplicação, semelhante à tela de criação de agendamentos. Nessa interface, o usuário pode selecionar uma determinada localidade e visualizar, em tempo real, o estado atual de seus recursos, ou, alternativamente, um indicador de ausência de informações quando não há dados atualizados para a localidade selecionada.

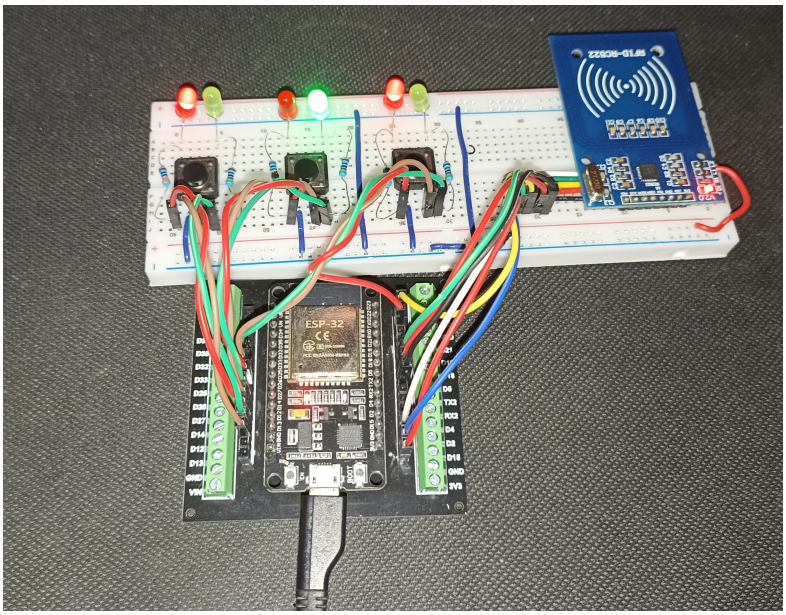
Figura 14 – Visualização de status em tempo real

Fonte: Elaborado pelo autor, 2025.

Com o objetivo de validar a integração e a consistência das informações recebidas via MQTT, foi construído um protótipo didático simples, destinado exclusivamente a atuar como

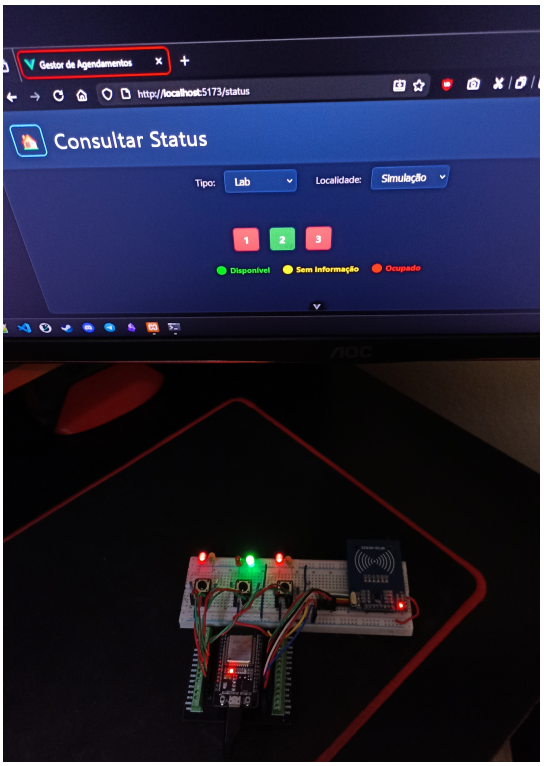
uma fonte externa de dados para a aplicação, simulando sensores de uma localidade contendo três recursos. Para tal, utilizou-se um microcontrolador ESP32, LEDs para representação visual do estado de cada recurso e *push buttons* para permitir a alternância manual entre os diferentes estados das vagas. As Figuras 15 e 16 são fotografias que demonstram, respectivamente, a montagem do *hardware* deste protótipo e a sua comunicação em tempo real com o sistema.

Figura 15 – Protótipo para simulação de monitoramento



Fonte: Elaborado pelo autor, 2025.

Figura 16 – Protótipo para simulação de monitoramento



Fonte: Elaborado pelo autor, 2025.

Definiu-se que, caso uma localidade não receba novas informações dentro de um período de dez minutos, os dados na tabela *de status* deverão ser limpos para que informações desatualizadas não sejam exibidas.

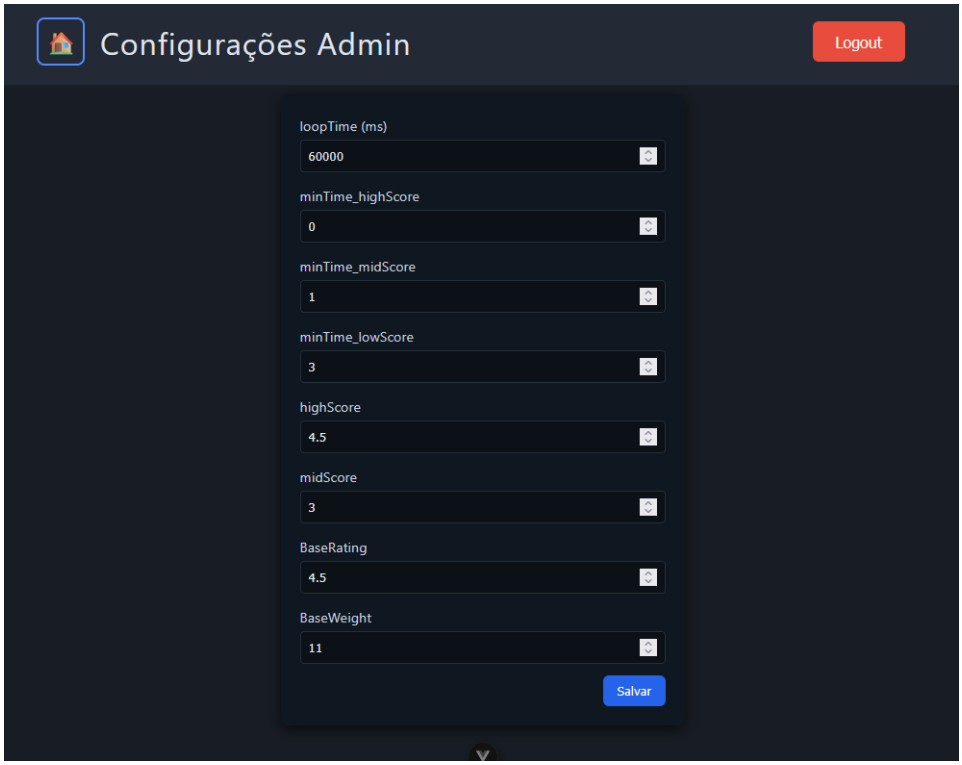
Em (VASCONCELOS, 2026c) encontra-se, o código-fonte do microcontrolador ESP32, utilizado na implementação e validação do protótipo descrito.

4.4 Parâmetros de Configuração

No *backend* da aplicação foi definido um arquivo de configuração no formato *JSON*, destinado ao armazenamento de parâmetros variáveis do sistema. Esses parâmetros concentram valores que influenciam diretamente o comportamento da aplicação em tempo de execução, permitindo maior flexibilidade sem a necessidade de alterações no código-fonte. O arquivo utilizado no ambiente onde foram realizados os testes está disponível em (VASCONCELOS, 2026a).

Na Figura 17 mostra-se a implementação de um simples formulário para realizar a atualização dos parâmetros de configuração remotamente, disponível apenas para usuários administradores.

Figura 17 – Tela Administrativa para configurações de parâmetros



Fonte: Elaborado pelo autor, 2025.

A utilização desse arquivo possibilita a adaptação do sistema a diferentes cenários de uso, regras de negócio e condições de infraestrutura, como intervalos de execução de rotinas

periódicas, limites associados a processos automáticos e valores limiares utilizados em decisões internas do sistema. Dessa forma, ajustes operacionais podem ser realizados de maneira simples e controlada, por usuários administradores, sem comprometer a estabilidade ou a manutenção do código.

4.5 Rotinas Cíclicas do Sistema

De acordo com os parâmetros configuráveis definidos no sistema, são executadas rotinas periódicas responsáveis por realizar validações automáticas e garantir o correto funcionamento das regras de negócio. Essas rotinas atuam de forma independente da interação direta dos usuários.

4.5.1 Confirmação Baseada em Reputação

Um dos principais diferenciais do sistema proposto é o mecanismo de confirmação de agendamentos baseado na reputação do usuário solicitante. Para atender a esse requisito, definiu-se que todas as reservas devem ser inicialmente registradas como solicitações pendentes, independentemente da disponibilidade imediata do recurso, sendo aprovadas ou rejeitadas posteriormente pelo sistema.

Periodicamente, a rotina de confirmação consulta todos os agendamentos pendentes e avalia a nota média do usuário responsável pela solicitação. Para cada registro, é verificado se o tempo mínimo de espera, definido de acordo com a faixa de reputação, já foi atingido. Caso esse critério seja atendido, o sistema verifica a existência de conflitos de agenda. Havendo conflito entre solicitações concorrentes, o agendamento associado ao usuário com maior reputação é confirmado, enquanto os demais são rejeitados. Em situações de empate, é priorizada a solicitação realizada primeiro.

Para fins de teste e validação do sistema, foram definidos os seguintes parâmetros iniciais, onde a reputação é um coeficiente numérico construído a partir da avaliação média do usuário:

- Reputação média considerada a partir de 3,0;
- Reputação alta considerada a partir de 4,5;
- Tempo mínimo de espera para usuários de reputação baixa: três horas;
- Tempo mínimo de espera para usuários de reputação média: uma hora;
- Confirmação imediata para usuários de reputação alta.

Dessa forma, uma solicitação realizada por um usuário com reputação 4,0 somente será confirmada caso, após uma hora, o recurso permaneça disponível. Já solicitações efetuadas por usuários com reputação igual ou superior a 4,5 são confirmadas de forma imediata, desde que

não exista conflito. Em cenários de solicitações simultâneas, a de maior reputação é priorizada, sendo as demais automaticamente rejeitadas.

4.5.2 Encerramento e Atribuição de Avaliações

O encerramento dos agendamentos pode ocorrer de duas maneiras: por meio do envio de informações de *check-in* e *check-out* a partir de um dispositivo físico, ou de forma automática pelo próprio sistema.

O encerramento padrão, bem como a definição do início de um agendamento, ocorre a partir do recebimento de uma requisição informando que as credenciais de um usuário foram lidas fisicamente em uma determinada localidade.

No protótipo construído, conforme mostrado anteriormente na Figura 5, utilizou-se um módulo leitor de RFID para a identificação de *tags* contendo as credenciais de acesso. Entretanto, o sistema foi concebido de forma a permitir a utilização de diferentes métodos de autenticação, de maneira isolada ou combinada, como leitores biométricos ou sistemas de reconhecimento facial, dentre outras possibilidades.

Ao receber o evento de *check-in* do usuário, o sistema realiza uma busca por agendamentos previamente confirmados e, caso identifique uma confirmação compatível com o horário atual ou até um tempo de tolerância (parâmetro padrão de dez minutos), retorna a autorização de acesso, marca o agendamento como iniciado e registra o horário correspondente.

Durante o processo de *check-out*, é efetuado o encerramento do agendamento e a verificação do cumprimento do prazo estabelecido. Caso o encerramento ocorra dentro do horário previsto, é atribuída a nota máxima. Se ocorrer dentro de uma tolerância de cinco minutos, é atribuído o valor correspondente à *nota alta* configurada nos parâmetros do sistema. Para atrasos superiores a uma hora, é atribuída a nota mínima (1,0), enquanto para atrasos inferiores a esse limite é aplicada uma pontuação inversamente proporcional ao tempo de atraso.

Caso um agendamento permaneça vencido por um tempo superior ao definido nos parâmetros (vinte e quatro horas por padrão), ele é encerrado automaticamente. Para isso, o sistema consulta todos os registros confirmados cujo campo `fulfilled` esteja definido como falso e cuja data de término (`endDate`) apresente diferença superior a vinte e quatro horas em relação ao horário atual. Se o agendamento não tiver sido iniciado (`started = 0`), é atribuída automaticamente uma avaliação de nota 2,0 com peso 1. Caso o agendamento tenha sido iniciado, mas não finalizado corretamente, a avaliação atribuída é de nota 1,0, também com peso 1.

4.5.3 Reavaliação da Reputação

Sempre que um novo registro é inserido na tabela `ratings`, o sistema executa automaticamente o recálculo da nota média do usuário correspondente. Esse novo valor é então atualizado

na tabela `users`, garantindo que a reputação reflita de forma contínua o histórico mais recente de avaliações e comportamentos registrados.

4.6 Análise Integrada do Sistema

A aplicação desenvolvida demonstrou funcionamento integrado e coerente entre seus principais módulos, desde a interação inicial do usuário até o encerramento completo de um agendamento. O fluxo do sistema inicia-se no *frontend*, responsável por coletar as entradas do usuário e apresentar visualmente os dados relevantes, e se estende ao *backend*, onde são realizadas as validações, regras de negócio e persistência das informações no banco de dados.

O processo de autenticação garante o controle de acesso e a diferenciação de permissões, permitindo que apenas usuários autorizados realizem determinadas operações. A partir desse ponto, o usuário pode consultar recursos disponíveis, solicitar agendamentos e acompanhar o estado de suas solicitações. Essas requisições são tratadas por meio das APIs desenvolvidas, que atuam como intermediárias entre a interface e o banco de dados, assegurando consistência e integridade das informações.

Um dos principais diferenciais observados nos resultados foi a atuação das rotinas cíclicas do sistema, responsáveis por automatizar processos críticos, como a confirmação de agendamentos baseada na reputação dos usuários e o encerramento automático de agendas pendentes. A integração dessas rotinas com o modelo de reputação demonstrou-se eficaz para priorizar usuários com histórico positivo, ao mesmo tempo em que penaliza comportamentos inadequados de forma objetiva e mensurável.

De forma geral, os testes realizados indicaram que o sistema é capaz de operar de maneira contínua e autônoma, atendendo aos requisitos funcionais definidos na metodologia e validando a viabilidade da proposta como uma solução flexível para o gerenciamento e agendamento de recursos compartilhados.

5 CONCLUSÕES

Este trabalho teve como objetivo geral o desenvolvimento de um sistema de agendamento para recursos escassos em ambientes compartilhados, integrando funcionalidades de reserva, autenticação de usuários, monitoramento do estado dos recursos e registro histórico para composição de um indicador de reputação. A solução proposta buscou possibilitar a visualização e a reserva de recursos, bem como o controle de acesso às dependências, respeitando as limitações e a infraestrutura disponível no local de implantação.

Os objetivos definidos no início do trabalho foram atendidos de forma satisfatória. Foi desenvolvida uma aplicação web capaz de gerenciar diferentes tipos de recursos, como vagas de estacionamento, salas, laboratórios e mesas, permitindo o agendamento e a consulta de disponibilidade. Implementou-se um módulo de autenticação e controle de acesso integrado ao sistema, assegurando que apenas usuários devidamente cadastrados e autorizados possam utilizar as funcionalidades disponíveis. Adicionalmente, foi construído um protótipo didático utilizado para simular a identificação do usuário e a validação do acesso aos recursos reservados.

A integração com sistemas e dispositivos externos foi viabilizada por meio de protocolos padronizados de comunicação, permitindo que o sistema consuma informações de status provenientes de diferentes fontes, independentemente da tecnologia empregada na coleta dos dados. Também foi implementado um mecanismo de reputação fundamentado no comportamento dos usuários ao longo do uso do sistema, considerando critérios objetivos como cumprimento de horários, ausências e atrasos, contribuindo para uma gestão mais equilibrada e responsável dos recursos compartilhados.

Embora o sistema atenda aos objetivos propostos neste trabalho, diversas melhorias podem ser consideradas em trabalhos futuros, como o desenvolvimento de um *frontend* mais robusto e orientado ao usuário final, incorporando princípios de *User Experience* (UX) e *User Interface* (UI) para adequação a ambientes produtivos, a implementação de autenticação multifator e políticas mais rígidas de controle de acesso, proteção contra ataques comuns a APIs (como *rate limiting* e validações abrangentes de dados), integração com provedores externos como login via Google, aumento da robustez das APIs em cenários de falhas e alta concorrência e otimizações de desempenho. Adicionalmente, pode ser implementado a integração com módulos avançados de monitoramento baseados em processamento de imagens por redes neurais para automação da identificação de estados de recursos, além de mecanismos de auditoria, geração de relatórios analíticos e adaptação da arquitetura para ambientes distribuídos ou em nuvem, ampliando a escalabilidade e o alcance da solução.

Os resultados obtidos demonstram a viabilidade técnica da solução e sua aderência aos princípios de modularidade, extensibilidade e organização. Dessa forma, conclui-se que o projeto alcança sua finalidade acadêmica e prática, servindo como base para evoluções posteriores e para a adaptação do sistema a diferentes contextos institucionais.

REFERÊNCIAS

ADRJAN, P. *et al.* Working from home after covid-19: Evidence from job postings in 20 countries. **Labour Economics**, v. 96, p. 102751, 2025. ISSN 0927-5371. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0927537125000752>. Citado na página 9.

BANKS, A.; GUPTA, R. (Ed.). **MQTT Version 3.1.1**. 2014. OASIS Standard. Acesso em: 2025. Disponível em: <https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>. Citado na página 20.

CALIS, J. O. G. **APLICAÇÃO DE REDES NEURAIS CONVOLUCIONAIS PARA RECONHECIMENTO AUTOMÁTICO DE PLACAS DE VEÍCULOS**. Tese (Doutorado) — Universidade Estadual Paulista "Júlio de Mesquita Filho, 2018. Citado na página 13.

CORS. **CORS Middleware for Node.js**. 2024. Acesso em: 2025. Disponível em: <https://www.npmjs.com/package/cors>. Citado na página 18.

DOTENV. **Dotenv Documentation**. 2024. Acesso em: 2025. Disponível em: <https://www.npmjs.com/package/dotenv>. Citado na página 18.

Espressif Systems. **ESP32 Series – Product Page**. n.d. <https://www.espressif.com/en/products/socs/esp32>. Acesso em: 2025. Citado na página 21.

EXPRESS.JS. **Express.js Documentation**. 2024. Acesso em: 2025. Disponível em: <https://expressjs.com>. Citado na página 17.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doutorado) — University of California, Irvine, 2000. Disponível em: <https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf>. Citado na página 16.

FIELDING, R. T.; RESCHKE, J. **Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**. 2014. Request for Comments 7231, RFC Editor. Acesso em: 2025. Disponível em: <https://www.rfc-editor.org/info/rfc7231>. Citado na página 20.

IETF. **JSON Web Token (JWT)**. 2015. RFC 7519. Disponível em: <https://www.rfc-editor.org/rfc/rfc7519>. Citado na página 18.

KL, T.; YADAV, U.; KRISHNAN, V.; KS, V.; PM, M. Application based smart parking reservation system using openalpr. v. 83, p. 5004–5008, 01 2021. Citado 3 vezes nas páginas 14, 17 e 19.

MELO, A. da S.; SILVA, I. M. da; SANTOS, R. T. dos. Estudo e desenvolvimento de um aplicativo para mobilidade urbana, explorando o gerenciamento inteligente de vagas de estacionamento. **Episteme Transversalis**, v. 13, n. 2, 2022. Citado 2 vezes nas páginas 13 e 17.

Microsoft. **TypeScript: JavaScript with syntax for types**. 2024. Disponível em: <https://www.typescriptlang.org>. Citado na página 19.

MYSQL2. **MySQL2 Documentation**. 2024. Acesso em: 2025. Disponível em: <https://www.npmjs.com/package/mysql2>. Citado na página 18.

NODE.JS. **Node.js Documentation**. 2024. Acesso em: 2025. Disponível em: <https://nodejs.org>. Citado na página 17.

NODEMON. **Nodemon Documentation**. 2024. Acesso em: 2025. Disponível em: <https://www.npmjs.com/package/nodemon>. Citado na página 18.

NURULLAYEV, S.; LEE, S.-W. Generalized parking occupancy analysis based on dilated convolutional neural network. **Sensors**, v. 19, n. 2, 2019. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/19/2/277>. Citado na página 14.

NWOHIRI, A.; MUHAMMAD, A. Vehicle access control and security system: University of lagos security gate as a case. **LAUTECH Journal of Engineering and Technology**, v. 13, n. 2, p. 52–59, Mar. 2020. Disponível em: <http://laujet.com/index.php/laujet/article/view/347>. Citado na página 13.

PALA, Z.; INANC, N. Smart parking applications using rfid technology. In: **2007 1st Annual RFID Eurasia**. [S.l.: s.n.], 2007. p. 1–3. Citado na página 19.

PINIA. **Pinia — State Management for Vue**. 2024. Disponível em: <https://pinia.vuejs.org>. Citado na página 19.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software: Uma Abordagem Profissional**. 8. ed. New York: McGraw-Hill Education, 2016. Citado 3 vezes nas páginas 15, 16 e 22.

PROVOS, N.; MAZIERES, D. A future-adaptable password scheme. **USENIX Annual Technical Conference**, p. 81–92, 1999. Disponível em: <https://www.usenix.org/legacy/events/usenix99/provos.html>. Citado na página 18.

ROUTER, V. **Vue Router**. 2024. Disponível em: <https://router.vuejs.org>. Citado na página 19.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Education, 2011. Citado 2 vezes nas páginas 15 e 22.

TILKOV, S.; VINOSKI, S. Node.js: Using javascript to build high-performance network programs. **IEEE Internet Computing**, v. 14, n. 6, p. 80–83, 2010. Citado na página 17.

VALENTE, M. T. **Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade**. [S.l.]: Editora: Independente, 2020. Citado 3 vezes nas páginas 15, 16 e 22.

VALIPOUR, S.; SIAM, M.; STROULIA, E.; JAGERSAND, M. Parking-stall vacancy indicator system, based on deep convolutional neural networks. In: **2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)**. [S.l.: s.n.], 2016. p. 655–660. Citado na página 13.

VASCONCELOS, M. F. **ScheduleManager: Backend do Sistema de Agendamento de Recursos**. 2026. <https://github.com/MarcialVasconcelos/ScheduleManager_Backend>. Código-fonte disponível em repositório GitHub. Acesso em: 2026. Citado 3 vezes nas páginas 35, 37 e 44.

____. **ScheduleManager: Frontend do Sistema de Agendamento de Recursos**. 2026. <https://github.com/MarcialVasconcelos/ScheduleManager_Frontend>. Código-fonte disponível em repositório GitHub. Acesso em: 2026. Citado na página 35.

____. **ScheduleManager: Protótipo Desenvolvido com ESP32**. 2026. <https://github.com/MarcialVasconcelos/ScheduleManager_PrototypeESP32>. Código-fonte disponível em repositório GitHub. Acesso em: 2026. Citado na página 44.

VITE. **Vite — A next generation frontend tooling**. 2024. Disponível em: <https://vitejs.dev>. Citado na página 19.

VUE.JS. **Vue.js**. 2024. Disponível em: <https://vuejs.org>. Citado na página 18.

APÊNDICE A – DETALHAMENTO TÉCNICO DA ESTRUTURA DO BANCO DE DADOS

Tabela `users`:

Responsável por armazenar os dados cadastrais dos usuários, bem como informações relacionadas às permissões de acesso e à reputação no sistema. Os principais campos definidos são:

- `id` — chave primária do tipo `INT UNSIGNED`, com incremento automático.

  Identificador único dos registros, utilizado como referência nas demais tabelas do sistema.

- `name`, `lastName` — tipo `TEXT`.

  Campos destinados ao armazenamento das informações de identificação do usuário.

- `email` — tipo `TEXT`, com restrição de unicidade.

  Campo utilizado como identificador lógico do usuário, impedindo o cadastro de múltiplas contas associadas ao mesmo endereço de correio eletrônico.

- `password` — tipo `TEXT`.

  Armazena o *hash* da senha do usuário, garantindo que as credenciais não sejam persistidas em formato legível.

- `active`, `admin` — tipo `TINYINT`.

  Campos de comportamento booleano, nos quais o valor 0 indica, respectivamente, usuário inativo ou usuário comum, e o valor 1 indica usuário ativo ou administrador.

- `avgRating` — tipo `FLOAT`.

  Valor real no intervalo de 1 a 5 que representa a reputação média do usuário, calculada a partir das avaliações registradas no sistema.

Tabela `loc`:

Armazena as informações referentes às localidades ou recursos disponíveis para agendamento. O conteúdo desta tabela é classificado como dados cadastrais e serve de base para os registros de utilização do sistema.

- `id` — chave primária do tipo `INT UNSIGNED`, com incremento automático.

  Identificador único da localidade.

- `name` — tipo `TEXT`.

  Nome ou identificação textual da localidade.

- `type` — tipo `TEXT`.

  Indica a categoria do recurso, como estacionamento, sala, laboratório ou mesa compartilhada.

- `capacity` — tipo `INT UNSIGNED`.

  Número natural que define a quantidade máxima de vagas ou unidades disponíveis na localidade.

Tabela `schedules`:

Representa os registros funcionais de agendamento do sistema, associando usuários a localidades em intervalos de tempo previamente definidos.

- `id` — chave primária do tipo `INT UNSIGNED`, com incremento automático.

  Identificador único do agendamento.

- `userID` — chave estrangeira associada à tabela `users`.

  Indica o usuário responsável pelo agendamento.

- `locID` — chave estrangeira associada à tabela `loc`.

  Define a localidade ou recurso reservado.

- `asset` — tipo `INT UNSIGNED`.

  Número natural que representa o identificador cardinal da vaga ou recurso utilizado.

- `startDaT`, `endDaT` — tipo `TIMESTAMP`.

  Determinam o intervalo temporal do agendamento.

- `created_on`, `started_on`, `fulfilled_on` — tipo `TIMESTAMP`.

  Deixa registrado respectivamente o momento em que o agendamento foi criado, iniciado e finalizado.

- `started`, `fulfilled` — tipo `TINYINT`.

  Campos de comportamento booleano, nos quais os valores indicam, respectivamente, se o agendamento foi iniciado e finalizado.

- `confirmed` — tipo `TINYINT`.

  Indica a condição do agendamento, como pendente (NULL), confirmado (1) ou recusado (0).

Tabela `ratings`:

Destinada ao registro das avaliações associadas ao comportamento dos usuários, servindo como base para o cálculo da reputação.

- `id` — chave primária do tipo `INT UNSIGNED`, com incremento automático.

  Identificador único da avaliação.

- `user_id` — chave estrangeira associada à tabela `users`.

  Usuário avaliado.

- `schedule_id` — chave estrangeira associada à tabela `schedules`.

  Agendamento ao qual a avaliação se refere.

- `type` — tipo `TEXT`.

  Identifica qual o tipo da atribuição, para distinguir notas automáticas do sistemas daquelas inseridas manualmente por administradores.

- `timestamp` — tipo `TIMESTAMP`.

  Deixa registrado o momento em que a nota foi atribuída.

- `rating` — tipo `FLOAT`.

  Nota atribuída ao usuário, utilizada no cálculo da reputação.

- `weight` — tipo `FLOAT`.

  Multiplicador de peso da nota.

Tabela `status`:

Armazena os dados em tempo real da condição dos recursos.

- `id` — chave primária do tipo `INT UNSIGNED`, com incremento automático.

  Identificador único do status.

- `locID` — chave estrangeira associada à tabela `loc`.

  Define a localidade ou recurso reservado.

- `asset` — tipo `INT UNSIGNED`.

  Número natural que representa o identificador cardinal da vaga ou recurso utilizado.

- `status` — tipo `TINYINT`.

  Indica a condição atual do recurso, como sem informação (NULL), livre (0) ou ocupado (1).

APÊNDICE B – BACKEND: FUNÇÕES DE MANUTENÇÃO DE TABELAS

B.1 Funções comuns

As funções descritas nesta subseção seguem um padrão lógico comum a todas as entidades do sistema, variando apenas de acordo com os campos e tipos de dados de cada tabela. Essas rotinas implementam as operações básicas de criação, leitura, atualização e exclusão de registros (*CRUD*).

Tabela 1 – Funções comuns de manutenção das tabelas

Função	**Parâmetros**	**Retorno**	**Descrição**
`selectAll`	N/A	Array de objetos	Retorna todos os registros da tabela correspondente.
`selectBy<Campo>`	Valor do campo	Array de objetos	Retorna registros filtrados com base em um campo específico.
`insert`	Objeto da entidade	N/A	Realiza a inserção de um novo registro na tabela associada.
`update`	ID do registro, objeto da entidade	N/A	Atualiza os dados do registro identificado pelo ID informado.
`delete`	ID do registro	N/A	Remove o registro correspondente ao ID informado.

Fonte: Elaborado pelo autor.

B.2 Funções específicas

Além das funções genéricas, foram desenvolvidas rotinas específicas para atender a regras de negócio próprias do sistema de agendamentos e avaliação de usuários. Essas funções concentram lógicas que não se aplicam de forma genérica a todas as entidades.

Tabela 2 – Funções específicas do sistema

Função	**Parâmetros**	**Retorno**	**Descrição**
`authenticateUser`	Email, senha	Booleano	Valida as credenciais do usuário por meio da comparação do *hash* da senha utilizando *bcrypt*.
`getAllTypes`	N/A	Array de strings	Retorna os valores distintos referentes aos tipos de localidades cadastradas.
`checkAvailability`	Objeto `schedule`	Booleano	Verifica a existência de conflitos de agendamento no período informado.
`getAvailabilityArray`	ID da localidade, início, fim	Array	Retorna a disponibilidade individual de cada recurso da localidade no intervalo informado.
`updateAvgRating`	ID do usuário	N/A	Recalcula e atualiza a média de avaliações do usuário na tabela `users`.

Fonte: Elaborado pelo autor.

APÊNDICE C – DOCUMENTAÇÃO DAS ROTAS DA API

C.1 Visão Geral

A aplicação utiliza o framework Express.js para gerenciar requisições HTTP e implementa autenticação por token JWT, com exceções para os endpoints de login, cadastro de usuários e atualização de status com autenticação básica.

C.2 Middleware de Autenticação

Todos os endpoints, exceto os listados abaixo, requerem um token JWT válido no cabeçalho `Authorization` da requisição:

- `POST /login`
- `POST /users`

C.3 Endpoints da API

C.3.1 Autenticação

Tabela 3 – Endpoints de autenticação

Método	Rota	Parâmetros	Retorno
POST	`/login`	Email e senha do usuário	Token JWT de autenticação

Fonte: Elaborado pelo autor.

C.3.2 Usuários

Tabela 4 – Endpoints de gerenciamento de usuários

Método	Rota	Parâmetros	Retorno
GET	`/users`	Nenhuma	Lista de todos os usuários
GET	`/users/:id`	ID do usuário (parâmetro de rota)	Dados do usuário específico
POST	`/users`	Dados do novo usuário (corpo da requisição)	Usuário criado com ID gerado
PATCH	`/users`	ID e dados a atualizar (corpo da requisição)	Usuário atualizado
DELETE	`/users`	ID do usuário (corpo da requisição)	Confirmação de exclusão

Fonte: Elaborado pelo autor.

C.3.3 Localizações

Tabela 5 – Endpoints de gerenciamento de localizações

Método	**Rota**	**Parâmetros**	**Retorno**
GET	`/locations`	Nenhuma	Lista de todas as localizações
GET	`/locations/:id`	ID da localização (parâmetro de rota)	Dados da localização específica
POST	`/locations`	Dados da nova localização (corpo da requisição)	Localização criada com ID gerado
PATCH	`/locations`	ID e dados a atualizar (corpo da requisição)	Localização atualizada
PATCH	`/locations/:id`	ID (parâmetro de rota) e dados a atualizar (corpo)	Localização atualizada
DELETE	`/locations`	ID da localização (corpo da requisição)	Confirmação de exclusão

Fonte: Elaborado pelo autor.

C.3.4 Tipos de Localização

Tabela 6 – Endpoints de tipos de localização

Método	**Rota**	**Parâmetros**	**Retorno**
GET	`/loctypes`	Nenhuma	Array de tipos de localização disponíveis
GET	`/loctypes/:type`	Tipo de localização (parâmetro de rota)	Localizações do tipo especificado

Fonte: Elaborado pelo autor.

C.3.5 Agendamentos

Tabela 7 – Endpoints de gerenciamento de agendamentos

Método	**Rota**	**Parâmetros**	**Retorno**
GET	`/schedules`	Nenhuma	Lista de todos os agendamentos
GET	`/schedules/UID/:userID`	ID do usuário (parâmetro de rota)	Agendamentos do usuário específico
POST	`/schedules`	Dados do novo agendamento (corpo da requisição)	Agendamento criado com ID gerado
PATCH	`/schedules`	ID e dados a atualizar (corpo da requisição)	Agendamento atualizado
PATCH	`/schedules/:id`	ID (parâmetro de rota) e dados a atualizar (corpo)	Agendamento atualizado
DELETE	`/schedules`	ID do agendamento (corpo da requisição)	Confirmação de exclusão
GET	`/schedules/status`	Nenhuma	Array de disponibilidades de agendamento
POST	`/schedules/checkinout`	Dados de entrada/saída (corpo da requisição)	Confirmação de check-in/check-out

Fonte: Elaborado pelo autor.

C.3.6 Avaliações

Tabela 8 – Endpoints de gerenciamento de avaliações

Método	**Rota**	**Parâmetros**	**Retorno**
GET	`/ratings`	Nenhuma	Lista de todas as avaliações
GET	`/ratings/:id`	ID da avaliação (parâmetro de rota)	Dados da avaliação específica
POST	`/ratings`	Dados da nova avaliação (corpo da requisição)	Avaliação criada com ID gerado
PATCH	`/ratings`	ID e dados a atualizar (corpo da requisição)	Avaliação atualizada
DELETE	`/ratings`	ID da avaliação (corpo da requisição)	Confirmação de exclusão

Fonte: Elaborado pelo autor.

C.3.7 Status

Tabela 9 – Endpoints de gerenciamento de status

Método	**Rota**	**Parâmetros**	**Retorno**
GET	`/status`	Nenhuma	Lista de todos os status
GET	`/status/:id`	ID do status (parâmetro de rota)	Dados do status específico
GET	`/status/loc/:locId`	ID da localização (parâmetro de rota)	Status da localização especificada
POST	`/status`	Dados do novo status (corpo da requisição)	Status criado com ID gerado
PATCH	`/status`	ID e dados a atualizar (corpo da requisição)	Status atualizado
DELETE	`/status`	ID do status (corpo da requisição)	Confirmação de exclusão

Fonte: Elaborado pelo autor.

C.3.8 Configurações

Tabela 10 – Endpoints de gerenciamento de configurações

Método	**Rota**	**Parâmetros**	**Retorno**
GET	`/config`	Nenhuma	Configurações atuais da aplicação
POST	`/config`	Novas configurações (corpo da requisição)	Confirmação de configurações salvas

Fonte: Elaborado pelo autor.

C.4 Tratamento de Erros

Todas as requisições para rotas não definidas retornam um status HTTP 403 (Forbidden) com mensagens de erro padrão definidas em `messages.json`. Requisições sem token JWT válido (exceto para as rotas de autenticação e status com autenticação básica) também serão rejeitadas com status 403.