v7.n11.2026

REVISTA CIENTÍFICA - RECIMA21 ISSN 2675-6218

DESENVOLVIMENTO DO VAI DAR JOGO: UMA PLATAFORMA WEB PARA AGENDAMENTO E GESTÃO DE QUADRAS ESPORTIVAS

DEVELOPMENT OF VAI DAR JOGO: A WEB-BASED SPORTS COURT BOOKING AND MANAGEMENT SYSTEM

DESARROLLO DE VAI DAR JOGO: UNA PLATAFORMA WEB PARA LA RESERVA Y GESTIÓN DE CANCHAS DEPORTIVAS

Vinícius de Freitas Yasuda[1], Felipe Diniz Dallilo[2]

e7119404
https://doi.org/10.47820/recima21.v7i11.9404

PUBLICADO: 11/2026

RESUMO

Este trabalho apresenta o desenvolvimento do Vai Dar Jogo, uma plataforma *web* para agendamento e gestão de quadras esportivas privadas. Parte dos estabelecimentos esportivos ainda gerencia reservas por controles manuais ou aplicativos de mensagens, o que dificulta a consulta autônoma de horários e a organização da agenda. Diante disso, propõe-se uma solução que integre busca, reserva e gestão operacional em um único ambiente. A pesquisa, de natureza aplicada e objetivo exploratório, abrangeu levantamento de requisitos em três estabelecimentos de Araraquara-SP, modelagem, implementação incremental e verificação técnica. O *back-end* foi construído em .NET 8 com base nos padrões *Clean Architecture*, DDD e CQRS, utilizando banco de dados SQL Server e controle transacional de concorrência, enquanto o *front-end* foi desenvolvido em React com TypeScript. A solução foi verificada por testes automatizados, incluindo um teste de integração com dez requisições simultâneas para o mesmo horário, além de testes de API e matriz de rastreabilidade. O protótipo resultante entrega autenticação, vitrine com filtros e avaliações, consulta de disponibilidade, painel financeiro e gestão de reservas, demonstrando a viabilidade técnica da proposta.

PALAVRAS-CHAVE: Aplicações *web*. Automação de processos. Engenharia de *Software*. Esportes. Sistema de agendamento.

ABSTRACT

This paper presents the development of Vai Dar Jogo, a web-based platform for booking and managing private sports courts. Many sports facilities still manage bookings through manual procedures or instant messaging apps, hindering autonomous availability lookup and schedule organization. A platform is proposed to integrate court discovery, booking, and management into a single environment. This applied and exploratory study comprised requirements elicitation across three venues in Araraquara-SP, modeling, incremental implementation, and technical verification. The back end was built in .NET 8 using Clean Architecture, DDD, and CQRS patterns with SQL Server and transactional concurrency control, while the front end was developed in React and TypeScript. Verification included automated tests, featuring an integration test with ten simultaneous booking requests for the same slot, alongside API testing and a traceability matrix. The resulting prototype provides authentication, a court showcase with filters and reviews, availability lookup computed at request time, a financial dashboard, and booking lifecycle management. The implementation demonstrates the technical feasibility of integrating sports court discovery, booking, and management functionalities into a web platform.

KEYWORDS: *Process automation. Scheduling system. Software Engineering. Sports. Web applications.*

[1] Graduando do Curso de Engenharia de Computação da Universidade de Araraquara – UNIARA. Araraquara-SP.
[2] Docente do Curso de Engenharia de Computação da Universidade de Araraquara – UNIARA. Araraquara-SP.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

RESUMEN
Este trabajo presenta el desarrollo de Vai Dar Jogo, una plataforma *web* para la reserva y gestión de canchas deportivas privadas. Parte de los establecimientos deportivos aún gestiona reservas por medio de controles manuales o aplicaciones de mensajería, dificultando la consulta autónoma de horarios y la administración de la agenda. Se propone una solución que integre búsqueda, reserva y gestión operativa en un único entorno. La investigación, aplicada y exploratoria, comprendió levantamiento de requisitos en tres establecimientos de Araraquara-SP, modelado, implementación incremental y verificación técnica. El *back-end* fue construido en .NET 8 con los patrones *Clean Architecture*, DDD y CQRS, utilizando SQL Server y control transaccional de concurrencia, mientras que el *front-end* se desarrolló en React con TypeScript. La solución se verificó por medio de pruebas automatizadas, incluida una prueba de integración con diez solicitudes simultáneas para el mismo horario, además de pruebas de API y matriz de trazabilidad. El prototipo resultante entrega autenticación, catálogo con filtros y valoraciones, consulta de disponibilidad, panel financiero y gestión de reservas, demostrando la viabilidad técnica de la propuesta.

PALABRAS CLAVE: *Aplicaciones web. Automatización de procesos. Deportes. Ingeniería de Software. Sistema de reservas.*

INTRODUÇÃO

Este trabalho apresenta o desenvolvimento de uma aplicação *web* voltada ao agendamento e à gestão de quadras esportivas privadas, denominada Vai Dar Jogo, cujo propósito é conectar usuários e proprietários em uma plataforma digital. O tema baseia-se nas limitações observadas nos meios habitualmente utilizados para divulgar e reservar esses espaços, nos quais as informações sobre localização, preço e disponibilidade costumam ficar espalhadas entre redes sociais e aplicativos de mensagens. Nesse contexto, o sistema proposto busca centralizar essas operações em uma única plataforma *web*.

O objetivo deste trabalho é desenvolver uma plataforma digital centralizada para a gestão e o agendamento de quadras esportivas privadas, facilitando a conexão entre atletas e proprietários. O projeto busca ampliar o acesso à informação sobre espaços esportivos, oferecendo maior clareza quanto à estrutura, às avaliações e à disponibilidade dos locais. Para o usuário final, o sistema oferece uma forma prática e autônoma de consultar horários e solicitar agendamentos; para os proprietários, atua como uma ferramenta de organização operacional e divulgação dos espaços, substituindo controles manuais por um fluxo digital estruturado.

A transformação digital tem modificado diretamente a forma como organizações estruturam seus processos e oferecem seus serviços. Esse movimento foi acelerado a partir de 2020 pela pandemia de COVID-19, que alterou rotinas de trabalho e consolidou o uso de canais digitais no cotidiano de empresas e consumidores. Atualmente, o acesso digital a serviços tornou-se uma expectativa recorrente do público, fazendo com que a adoção de ferramentas

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

tecnológicas represente um fator relevante de organização e competitividade para os negócios (VĂRZARU; BOCEAN, 2024).

Apesar desse avanço, parte dos estabelecimentos ainda realiza o agendamento por aplicativos de mensagens instantâneas, como o WhatsApp, ou por anotações manuais. Embora atendam operações de baixo volume, esses meios apresentam limitações práticas à medida que cresce o número de reservas: o cliente não consegue consultar horários livres sozinho, a confirmação depende de resposta humana, há risco de conflitos em pedidos feitos ao mesmo tempo e o histórico não fica organizado para conferência financeira. Enquanto agendas digitais de uso geral, a exemplo do Google Agenda, não oferecem vitrine pública por cidade, regras de bloqueio ou avaliações de usuários, plataformas comerciais do setor, como Playtomic e Appito, concentram sua atuação em grandes centros urbanos ou adotam modelos de cobrança por mensalidade que podem dificultar a adesão de pequenos proprietários. Segundo Hui e Teo (2024), plataformas digitais que exibem a disponibilidade de horários reduzem o tempo de espera no agendamento e auxiliam o controle operacional pelos responsáveis.

Diante desse cenário, dificuldades de localização e reserva podem restringir o acesso a determinados espaços esportivos, enquanto os proprietários enfrentam obstáculos no controle manual de bloqueios, cancelamentos e receitas. Formula-se, assim, a seguinte pergunta norteadora: como uma plataforma *web* pode integrar funcionalidades de busca, agendamento e gestão de quadras esportivas privadas, atendendo aos requisitos identificados entre usuários e proprietários?

Como hipótese, considera-se que a centralização desse fluxo em uma plataforma *web* atende às demandas funcionais de consulta autônoma de horários e de controle gerencial das reservas, limitando-se este estudo à implementação e à verificação técnica do protótipo.

A pesquisa classifica-se como aplicada, de natureza tecnológica e orientação exploratória, com abordagem qualitativa, combinando revisão bibliográfica e observação direta de três estabelecimentos esportivos em Araraquara-SP, conforme detalhado na Seção 2.

A partir dos requisitos levantados, o sistema foi implementado com *front-end* em React e TypeScript e *back-end* em .NET 8 na linguagem C#, utilizando arquitetura em camadas com DDD e CQRS e banco de dados SQL Server. Ao final, verificou-se a solução por meio de 171 testes automatizados e de uma matriz de rastreabilidade, apresentados na Seção 3.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

1. REFERENCIAL TEÓRICO

1.1. C# e .NET

O ecossistema formado pela linguagem C# e pela plataforma .NET, desenvolvido pela Microsoft, é amplamente adotado no desenvolvimento de aplicações *web* e corporativas. Isso ocorre porque a plataforma combina produtividade, desempenho e ferramentas integradas de desenvolvimento (ALBAHARI, 2024). A tipagem estática de C# permite identificar incompatibilidades de tipos durante o desenvolvimento e contribui para reduzir determinadas classes de erros (MICROSOFT, 2026a). Além disso, o suporte nativo à programação assíncrona com os operadores *async* e *await* e o gerenciamento automático de memória do .NET favorecem o atendimento de requisições concorrentes e a manutenção do código ao longo do tempo (MICROSOFT, 2023b).

1.2. HTML5, CSS3, TypeScript e React

A interface do sistema opera como uma *Single Page Application* estruturada com HTML5 semântico e folhas de estilo CSS3 responsivas por meio de *Media Queries* (MOZILLA, 2026b). Na lógica de apresentação, utilizam-se em conjunto a biblioteca React e o TypeScript, um *superset* do JavaScript cuja tipagem estática auxilia a detectar incompatibilidades de dados em tempo de compilação (TYPESCRIPT, [s.d.]). A arquitetura baseada em componentes e os mecanismos de atualização seletiva da interface via *Virtual DOM* podem favorecer a organização do código e a eficiência da renderização, dependendo das características da implementação e do gerenciamento de estado (MOZILLA, 2026a; REACT, [s.d.]).

1.3. SQL Server e Integridade Transacional

O Microsoft SQL Server é um Sistema Gerenciador de Banco de Dados Relacional (SGBDR) cuja modelagem física utiliza chaves primárias, chaves estrangeiras e índices para garantir a integridade referencial dos dados (CÓDIGO FÁCIL, 2026; MICROSOFT, 2026b). Cabe ressaltar, contudo, que o uso isolado de um SGBDR não impede conflitos lógicos em sistemas de agendamento: duas transações simultâneas podem ler o mesmo horário como livre antes que uma delas conclua a gravação, problema conhecido na literatura como *write skew* (KLEPPMANN, 2017). A prevenção de reservas duplicadas exige combinar transações ACID com níveis de isolamento restritivos, como o modo *Serializable*, ou com bloqueios exclusivos por recurso no banco de dados, a exemplo do procedimento *sp_getapplock*.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

1.4. *Domain-Driven Design* (DDD)

O *Domain-Driven Design* (DDD) concentra a modelagem do *software* nas regras de negócio e estabelece uma linguagem comum, chamada de Linguagem Ubíqua, alinhando os termos usados no levantamento de requisitos às classes do código-fonte (EVANS, 2003; FULL CYCLE, 2019). No chamado Design Tático, Vernon (2013) apresenta os elementos que estruturam a camada de domínio, incluindo as Entidades, que possuem identificador único ao longo do tempo, os Objetos de Valor, definidos apenas por seus atributos, além dos Agregados e dos Serviços de Domínio, mantendo o núcleo do sistema independente da tecnologia de banco de dados (MICROSOFT, 2022).

1.5. *Command Query Responsibility Segregation* (CQRS)

O padrão *Command Query Responsibility Segregation* (CQRS) separa as operações de escrita, representadas por Comandos voltados à validação de regras e alteração de estado, das operações de leitura, representadas por Consultas que apenas retornam dados sem modificar o sistema (MICROSOFT, 2025). Embora esse padrão permita dividir leitura e escrita em bancos separados em cenários de grande escala, neste trabalho, que adota um monolito modular conectado a um único banco SQL Server, a separação é aplicada de forma apenas lógica no nível de código, utilizando consultas não rastreadas com *AsNoTracking* e *handlers* independentes via MediatR, ficando a separação física de infraestrutura como uma possibilidade futura.

1.6. *Clean Architecture* e Arquitetura em Camadas

A *Clean Architecture*, proposta por Martin (2017), é um padrão arquitetural que organiza o *software* em camadas concêntricas orientadas pela Regra da Dependência, segundo a qual o código das camadas internas não deve depender de detalhes técnicos das camadas externas. Essa divisão mantém as regras de negócio isoladas de *frameworks*, interfaces visuais e mecanismos de persistência. Em conjunto com os princípios do DDD, essa estrutura facilita a criação de testes automatizados e a manutenção do sistema, pois permite alterar detalhes de banco de dados ou de interface sem modificar as entidades e validações centrais.

Na prática, o *software* divide-se em quatro camadas principais com responsabilidades bem definidas. A camada de Domínio é o núcleo da aplicação, reunindo entidades, regras de negócio, contratos e serviços de domínio, sem depender de bibliotecas externas. A camada de Aplicação coordena os casos de uso por meio de comandos e consultas, organizando o fluxo de dados sem concentrar regras negociais. A camada de Infraestrutura implementa o acesso a

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

recursos externos, como o banco de dados SQL Server e os serviços de geração de *tokens*. Por fim, a camada de Apresentação recebe as requisições HTTP da interface *web* e retorna respostas padronizadas (MICROSOFT, 2023a).

1.7. Diagramas UML

A *Unified Modeling Language* (UML) oferece notações visuais padronizadas para documentar a estrutura e o comportamento de sistemas orientados a objetos (FOWLER, 2004; BOOCH; RUMBAUGH; JACOBSON, 2005). Neste projeto, utilizam-se o Diagrama de Casos de Uso, que representa os requisitos funcionais e as interações dos perfis de usuário com o sistema por meio de relações como *include* e *extend*, e o Diagrama de Classes, que descreve as entidades, os objetos de valor e as relações de associação e composição do domínio.

1.8. Engenharia de Requisitos, Qualidade e Verificação de *Software*

A Engenharia de Requisitos compreende o levantamento, a especificação, a validação e o rastreamento dos requisitos funcionais e não funcionais de um *software* (SOMMERVILLE, 2018). Para avaliar a qualidade do produto, a norma ISO/IEC 25010 (ISO/IEC, 2023) define características como adequação funcional, confiabilidade, segurança, eficiência de desempenho e usabilidade. Enquanto a adequação funcional e a confiabilidade em acessos simultâneos podem ser verificadas por testes automatizados de unidade e de integração, a análise de usabilidade e de desempenho sob alto volume exige testes com usuários e ferramentas específicas de carga (SOMMERVILLE, 2018; KLEPPMANN, 2017). Além disso, sistemas que coletam dados cadastrais devem observar os princípios da Lei Geral de Proteção de Dados Pessoais (LGPD), instituída pela Lei nº 13.709/2018 (BRASIL, 2018).

2. METODOLOGIA

Esta seção descreve o percurso metodológico deste estudo, classificado como pesquisa aplicada, de natureza tecnológica e objetivo exploratório. O desenvolvimento foi realizado de forma iterativa e incremental em cinco etapas. Na primeira etapa, realizou-se a identificação do problema por meio da observação direta e informal da rotina de agendamento em três estabelecimentos privados de quadras esportivas na cidade de Araraquara-SP, durante o primeiro semestre de 2026, ocasião em que se verificou o uso exclusivo de WhatsApp e de controles manuais, sem consulta autônoma de horários, sem regras automáticas de cancelamento e sem histórico gerencial. Na segunda etapa, fez-se o levantamento de requisitos, transformando as necessidades observadas nos requisitos funcionais e não funcionais descritos

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

na Seção 2.2 e no Quadro 2. Na terceira etapa, elaboraram-se o projeto arquitetural e a modelagem, definindo a divisão em *Clean Architecture*, DDD e CQRS e a estrutura das tabelas no SQL Server, conforme as Seções 2.3 e 2.4. Na quarta etapa, executou-se a implementação do *back-end* em .NET 8 e do *front-end* em React com TypeScript. Por fim, na quinta etapa, realizou-se a verificação técnica da solução com o auxílio das bibliotecas xUnit, FluentAssertions e NSubstitute, somando 171 testes automatizados, testes de *endpoints* e matriz de rastreabilidade.

2.1. Ambiente e Ferramentas de Desenvolvimento

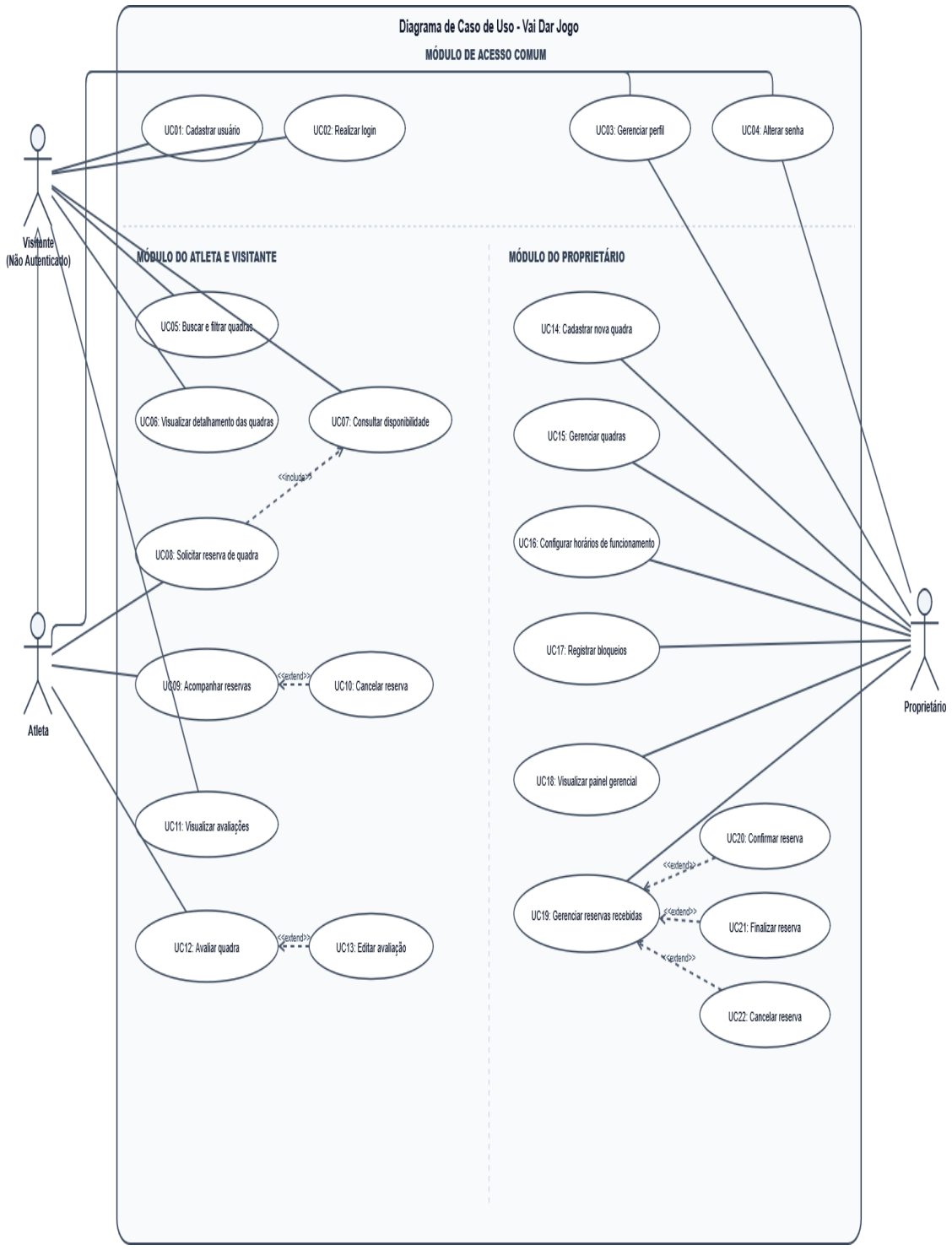
A implementação utilizou o ambiente Microsoft Visual Studio para o *back-end* em .NET 8, o editor Visual Studio Code para o *front-end* em React com TypeScript, o sistema Git para controle de versão e organização de ramificações de código (CHACON; STRAUB, 2014) e a ferramenta Postman para testar as requisições HTTP, a autenticação por *tokens* JWT e o formato JSON das respostas da API.

2.2. Diagrama de Casos de Uso

A especificação funcional da plataforma foi estruturada com base no mapeamento dos requisitos de negócio e das permissões atribuídas a cada perfil de usuário. A fim de formalizar as interações entre os atores e os módulos do sistema, foi elaborado o Diagrama de Casos de Uso, apresentado na Figura 1, conforme os conceitos fundamentados na Seção 1.7.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 1. Diagrama de casos de uso



Fonte: Elaborado pelos autores (2026).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

O sistema contempla três perfis de usuários com permissões distintas:

a) Visitante, correspondente ao usuário não autenticado: perfil público que pode acessar a plataforma para consultar quadras, informações de estrutura e horários disponíveis apenas para visualização;

b) Atleta: além das consultas públicas, pode selecionar horários, solicitar reservas, consultar o histórico de agendamentos, acompanhar as próximas partidas, avaliar espaços e alterar ou cancelar suas reservas, sendo exigida antecedência mínima de duas horas para o cancelamento de horários já confirmados e retornando a reserva ao *status Pending* quando houver reagendamento;

c) Proprietário: gerencia apenas as quadras pelas quais é responsável, configurando horários de funcionamento e bloqueios de agenda, acompanhando indicadores financeiros e aprovando, recusando, cancelando ou finalizando reservas, não sendo permitido reservar a própria quadra nem concluir partidas antes do término do horário agendado.

Além disso, atletas e proprietários compartilham as funcionalidades de gestão de conta, que incluem cadastro, *login*, atualização de dados de perfil e alteração de senha.

2.3. Arquitetura do sistema e modelo de domínio

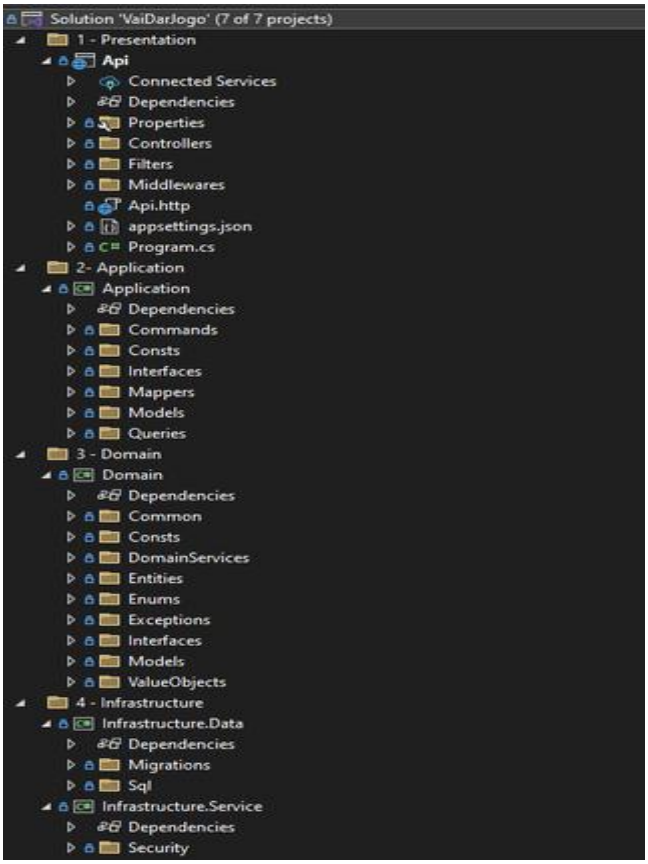
A arquitetura do sistema foi projetada com base nos princípios da *Clean Architecture*, em combinação com os padrões táticos do DDD e a segregação de responsabilidades via CQRS. O objetivo dessa decisão é isolar as regras de negócio para garantir maior manutenibilidade e testabilidade.

2.3.1. Arquitetura em camadas

A solução foi organizada no ecossistema .NET 8 em projetos independentes, respeitando a Regra da Dependência discutida na Seção 1.6. A Figura 2 apresenta a árvore de projetos da solução implementada no Visual Studio, mostrando a separação física entre as camadas.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 2. Estrutura de camadas no Visual Studio



Fonte: Elaborado pelos autores (2026).

As responsabilidades de cada camada estão delimitadas da seguinte forma:

1. *Domain*: projeto central e totalmente livre de dependências externas. Concentra as entidades de negócio, agregados, *Value Objects*, enumerações, exceções de domínio e contratos de serviços de domínio;

2. *Application*: orquestra os casos de uso por meio do padrão CQRS utilizando a biblioteca MediatR. Define *Commands*, *Queries* e seus respectivos *Handlers*, intermediando as requisições sem acoplar a interface às regras internas;

3. *Infrastructure*: contém as implementações concretas de acesso a recursos externos; divide-se em *Infrastructure.Data*, que lida com interações com o banco de dados, e *Infrastructure.Service*, que é responsável por serviços de segurança e geração de *tokens*;

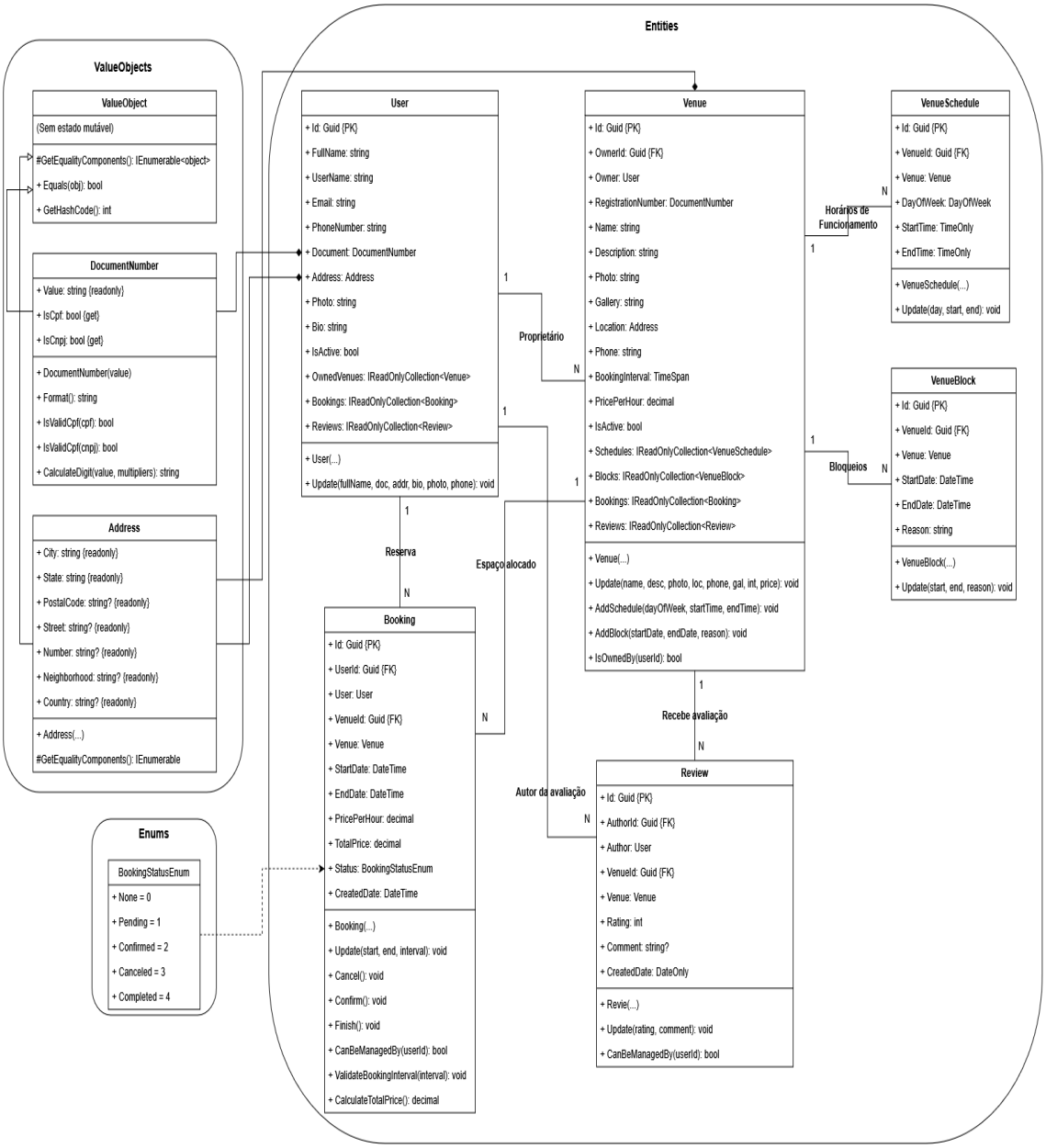
4. *Presentation*: camada mais externa, exposta como uma API RESTful, responsável por receber requisições HTTP e retornar respostas padronizadas.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

2.3.2. Diagrama de Classes

O núcleo da aplicação foi desenvolvido com base nos princípios do DDD, refletindo os termos do domínio de gestão de quadras esportivas diretamente nas classes do sistema. A Figura 3 apresenta o Diagrama de Classes, seguindo as notações descritas na Seção 1.7.

Figura 3. Diagrama de Classes



Fonte: Elaborado pelos autores (2026).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

2.3.3. Lógica de disponibilidade para agendamento

Em sistemas de agendamento, a validação de disponibilidade e a prevenção de conflitos de horários constituem desafios centrais da solução. No DDD, regras que envolvem múltiplos agregados independentes são atribuídas a Serviços de Domínio, preservando a coesão das entidades (EVANS, 2003).

Assim, o serviço *AvailabilityDomainService* implementa o método *CalculatePeriodsAvailability*, exibido na Figura 4, para montar a grade de horários para consulta.

Figura 4. Implementação do algoritmo de disponibilidade

```
1 reference
public class AvailabilityDomainService : IAvailabilityDomainService
{
    2 references
    public IEnumerable<AvailabilityPeriod> CalculatePeriodsAvailability(
        DateOnly currentDate,
        IEnumerable<VenueSchedule> schedules,
        IEnumerable<VenueBlock> blocks,
        IEnumerable<Booking> bookings
    )
    {
        var periods = new List<AvailabilityPeriod>();

        foreach (var schedule in schedules)
        {
            var scheduleStart = schedule.StartTime;
            var scheduleEnd = schedule.EndTime;

            var schedulePeriods = new List<AvailabilityPeriod>();

            AddBlockedPeriods(schedulePeriods, blocks, currentDate, scheduleStart, scheduleEnd);
            AddBookedPeriods(schedulePeriods, bookings, currentDate, scheduleStart, scheduleEnd);
            AddAvailablePeriods(schedulePeriods, scheduleStart, scheduleEnd);

            periods.AddRange(schedulePeriods);
        }

        return periods.OrderBy(p => p.StartTime);
    }
```

Fonte: Elaborado pelos autores (2026).

O algoritmo funciona da seguinte maneira:

1. Para cada registro de horário de funcionamento ativo da quadra no dia da semana solicitado, obtêm-se os horários de abertura e fechamento;

2. É chamado o método *AddBlockedPeriods*, que encontra os bloqueios registrados para a data e insere os intervalos de indisponibilidade com *status Blocked*;

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

3. É chamado o método *AddBookedPeriods*, que verifica as reservas ativas no período e adiciona os períodos ocupados com *status Booked*;

4. É chamado o método *AddAvailablePeriods*, que calcula os intervalos disponíveis dentro do horário de funcionamento e que não foram ocupados por bloqueios ou reservas, adicionando com *status Available*;

5. Por fim, os períodos gerados são adicionados em uma coleção e ordenados pelo horário de início, fornecendo, assim, à camada de aplicação as informações necessárias para a montagem da disponibilidade na vitrine da plataforma.

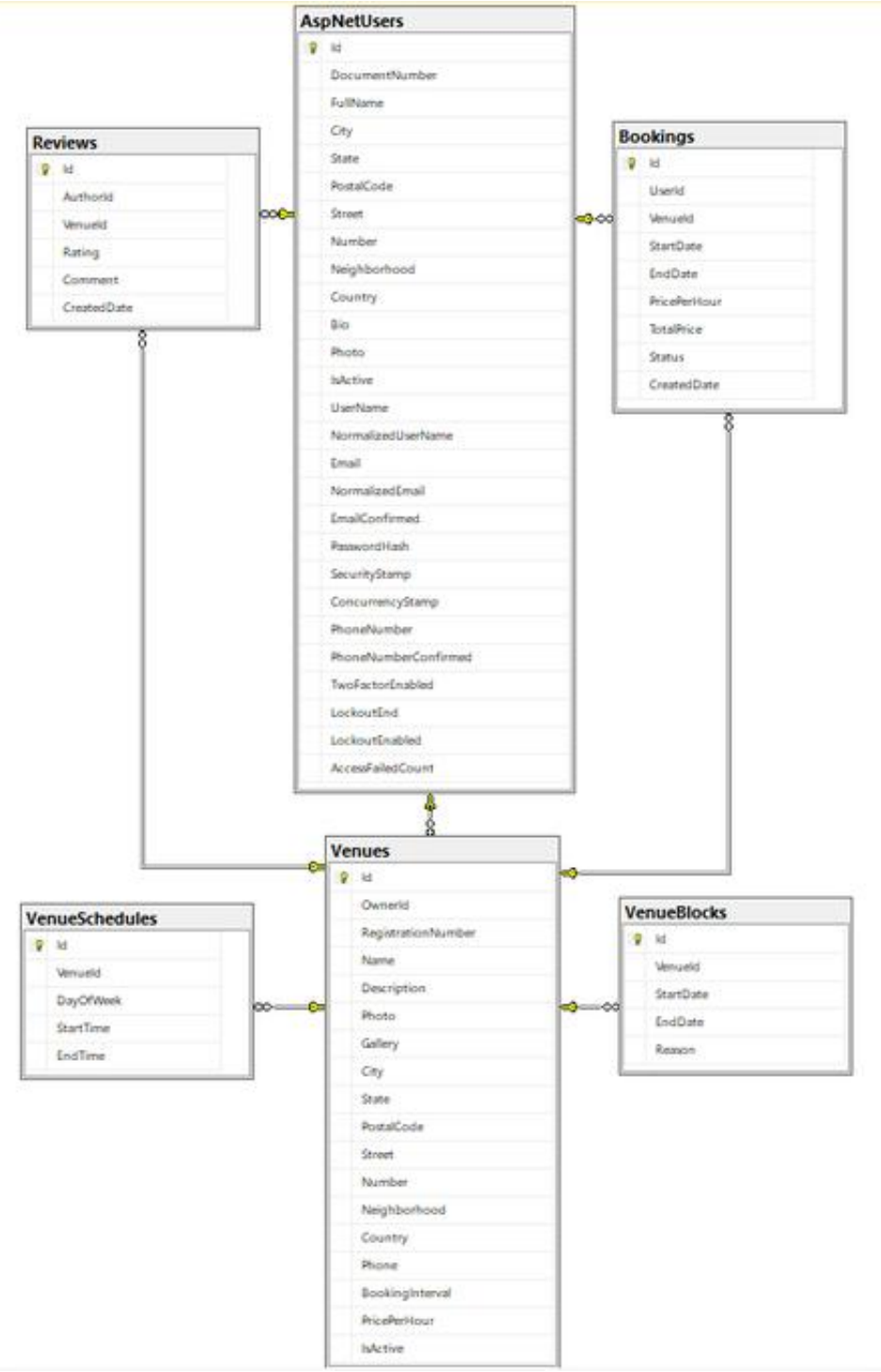
Cabe destacar que o método *CalculatePeriodsAvailability* atende apenas à consulta visual da grade, enquanto a criação e a alteração de reservas passam pela validação do serviço *BookingDomainService*. Esse serviço rejeita com exceção *409 Conflict* os pedidos para horários no passado, fora do expediente da quadra, desalinhados com a duração mínima ou que apresentem sobreposição de horário com bloqueios e reservas ativas, verificada quando o início solicitado é menor que o término existente e o término solicitado é maior que o início existente. Para evitar condições de corrida, também chamadas de *race conditions*, quando dois usuários tentam reservar o mesmo horário simultaneamente, o método *ExecuteWithVenueLockAsync* reúne a leitura de conflitos, a validação de regras e a gravação com *SaveChangesAsync* dentro de uma transação configurada com *IsolationLevel.Serializable* e associada ao bloqueio exclusivo *sp_getapplock* do SQL Server por quadra, garantindo o processamento sequencial das solicitações para um mesmo espaço.

2.4. Modelagem de Dados

A modelagem conceitual representada no diagrama de classes deu origem à estrutura física do banco de dados por meio das *migrations* do Entity Framework Core. As tabelas utilizam identificadores únicos do tipo *Guid* como chave primária, colunas numéricas decimais com duas casas para valores financeiros e chaves estrangeiras configuradas com restrição *NoAction* para preservar o histórico de vínculos entre usuários, quadras, horários, bloqueios e reservas. Adicionalmente, definiu-se um índice único composto pelas colunas *AuthorId* e *VenueId* na tabela *Reviews* para impedir que um mesmo atleta avalie a mesma quadra mais de uma vez. A Figura 5 apresenta o diagrama físico das tabelas gerado na ferramenta SQL Server Management Studio.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 5. Modelo físico



Fonte: Elaborado pelos autores (2026).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

2.5. Implementação da API RESTful

A API RESTful do sistema Vai Dar Jogo foi construída com o ASP.NET Core Web API, seguindo as convenções REST para rotas e métodos HTTP. Ao todo, a aplicação disponibiliza 36 *endpoints* distribuídos em sete módulos funcionais, conforme mostra o Quadro 1. Todas as requisições de escrita passam pela validação de dados de entrada com a biblioteca *FluentValidation* e pela checagem das regras de domínio: na rota *POST /api/v1/bookings*, por exemplo, um agendamento válido retorna o código *201 Created*, dados incompletos ou fora do formato retornam *400 Bad Request* e conflitos de horário retornam *409 Conflict* acompanhados de mensagem explicativa em formato JSON.

Quadro 1. Módulos da API e distribuição dos *endpoints*

Módulo	Total de *endpoints*	Descrição
Authentication	3	Registro, *login* e alteração de senha.
Bookings	6	Gerenciamento de agendamentos.
Reviews	3	Avaliações de espaços esportivos.
Users	7	Perfil, reservas, quadras, painel gerencial.
Venues	13	Gestão de quadras e disponibilidade.
VenueBlocks	2	Bloqueios de agenda.
VenueSchedules	2	Horário de funcionamento.

Fonte: Elaborado pelos autores (2026).

2.5.1. Autenticação e Autorização

O sistema utiliza autenticação *stateless* baseada em *tokens* JWT com validade de 60 minutos e verificação de emissor, audiência e assinatura digital. O controle de credenciais é feito pelo ASP.NET Core Identity, que aplica algoritmo PBKDF2 com *hash* e *salt* para proteger as senhas, bloqueia temporariamente a conta por 15 minutos após cinco tentativas incorretas e limita as rotas de *login* por meio de *rate limiting* a cinco requisições por minuto para cada endereço IP.

Além da verificação do perfil de acesso, identificado pela propriedade *Role* no *token*, todos os *handlers* de escrita conferem no banco de dados se o usuário autenticado é de fato o

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

proprietário da quadra, por meio do método *IsOwnedBy*, ou o autor da reserva ou avaliação, retornando o código *403 Forbidden* caso alguém tente modificar registros de terceiros. Como a plataforma armazena dados pessoais básicos, incluindo nome, *e-mail*, telefone e CPF ou CNPJ, o cadastro coleta apenas as informações estritamente necessárias à identificação das reservas, em alinhamento ao princípio da necessidade da LGPD (BRASIL, 2018), ficando a tela de consentimento expresso e o fluxo automatizado de exclusão de conta como etapas prévias ao lançamento comercial.

2.5.2. Tratamento de erros e paginação

Para manter a padronização das respostas da API, o componente *ExceptionHandlingMiddleware* intercepta falhas de validação ou de negócio e devolve objetos JSON com os respectivos códigos HTTP 400, 403, 404, 409 ou 500, planejando-se como melhoria futura a adoção do formato *Problem Details* especificado na norma RFC 9457. Nas rotas de listagem, as consultas aplicam paginação diretamente no banco de dados com os operadores *Skip* e *Take* e o retorno do total de registros, evitando o tráfego excessivo de informações em uma única chamada.

3. RESULTADOS E DISCUSSÃO

Esta seção apresenta as telas do protótipo funcional, a verificação automatizada e a rastreabilidade dos requisitos. A interface foi projetada com alternância entre modos claro e escuro e *layout* responsivo, decisões de projeto visual que não foram submetidas a testes formais com usuários nesta etapa.

3.1. Busca e detalhamento de quadras

A tela de busca de quadras, apresentada na Figura 6, funciona como a vitrine principal da plataforma. Nela, o usuário pode explorar os espaços esportivos disponíveis, filtrando por nome, cidade e faixa de preço, e ordenar os resultados por avaliação, preço ou nome. Cada item possui foto de capa, preço por hora, localização, descrição, nota média de avaliações e indicador se o local está aberto ou fechado no momento.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

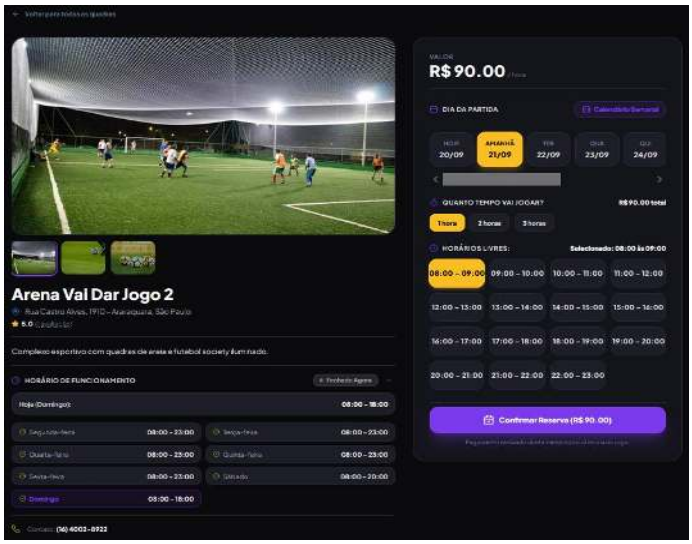
Figura 6. Tela de busca de quadras



Fonte: Elaborado pelos autores (2026).

Ao selecionar uma quadra, o usuário é direcionado à página de detalhamento correspondente, conforme a Figura 7. Esta tela apresenta fotos, preço por hora, horários de funcionamento semanal, telefone de contato e seção de avaliações. Na lateral direita, encontra-se o painel de agendamento com seleção de dia, duração desejada e horários livres. Os horários disponíveis são exibidos em destaque, enquanto horários já reservados, bloqueados ou passados são exibidos como indisponíveis para seleção. Após a seleção, o usuário pode confirmar a reserva com o valor total calculado dinamicamente.

Figura 7. Tela de detalhes da quadra

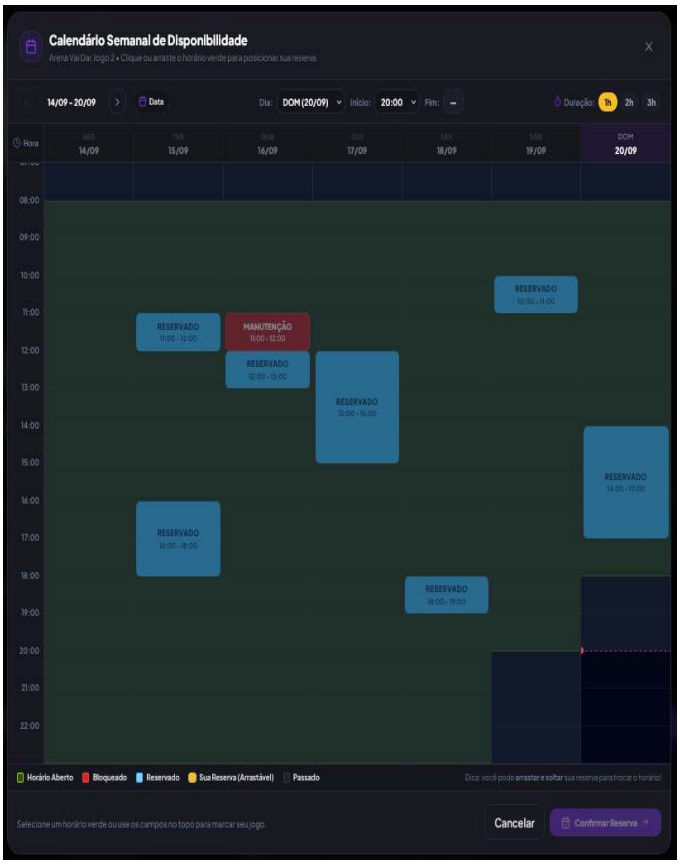


Fonte: Elaborado pelos autores (2026).

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Adicionalmente, o sistema oferece a funcionalidade de calendário semanal, apresentado na Figura 8, que proporciona uma visão consolidada da disponibilidade ao longo das semanas para apoiar a escolha de horários com antecedência.

Figura 8. Calendário semanal



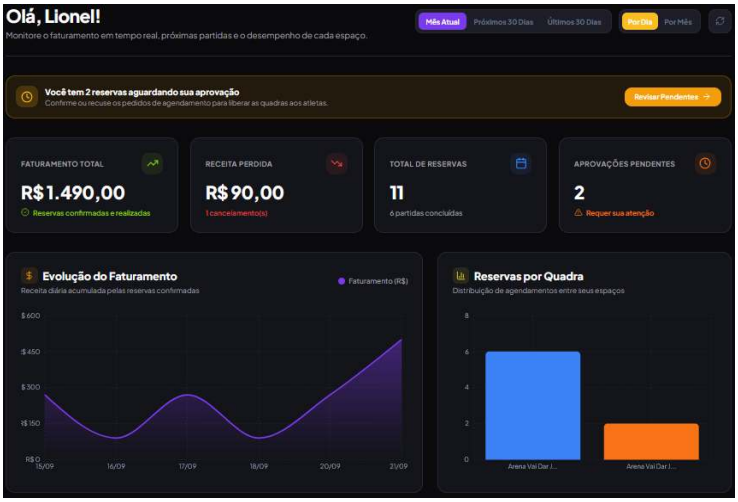
Fonte: Elaborado pelos autores (2026).

3.2. Painel Gerencial

O painel gerencial do proprietário, apresentado na Figura 9, reúne os principais indicadores dos espaços cadastrados: faturamento total, valor que deixou de ser recebido por conta de cancelamentos, total de reservas e pedidos pendentes de aprovação, acompanhados de gráficos de evolução de receita e de distribuição de reservas por quadra, que podem ser filtrados por período e agrupados por dia ou por mês. Na parte inferior da tela, uma tabela exibe as próximas partidas agendadas com botões de ação rápida para aprovar, recusar ou finalizar o agendamento, além da lista de quadras do proprietário com atalhos para a configuração da agenda e para a página pública.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

Figura 9. Painel gerencial do proprietário



Fonte: Elaborado pelos autores (2026).

3.3. Avaliações

O módulo de avaliações permite que atletas autenticados atribuam uma nota de 1 a 5 e escrevam um comentário sobre o estabelecimento, sendo permitido apenas um registro por atleta em cada quadra. A tela exibe a nota média da quadra, o total de avaliações recebidas e a distribuição por quantidade de estrelas, além de permitir que o próprio autor edite ou exclua seu comentário, conforme mostra a Figura 10.

Figura 10. Seção de avaliações



Fonte: Elaborado pelos autores (2026).

3.4. Verificação automatizada e rastreabilidade de requisitos

A verificação técnica foi realizada com os *frameworks* xUnit, FluentAssertions e NSubstitute, somando 171 testes automatizados executados com 100% de aprovação e nenhuma falha. Desse total, 83 testes validam as regras das entidades de domínio *Booking*, *Venue*, *Review*, *VenueBlock* e *VenueSchedule*; 22 testam os objetos de valor *DocumentNumber*

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

e *Address*; 22 verificam os serviços de domínio de disponibilidade, reservas e quadras; 40 testam os *handlers* e validadores da camada de aplicação; 3 conferem as regras de configuração de *tokens* JWT; e 1 consiste no teste de integração de concorrência no banco SQL Server. Nesse teste de integração, foram enviadas simultaneamente dez requisições paralelas tentando reservar exatamente o mesmo horário de uma mesma quadra sob a proteção do bloqueio *sp_getapplock* e do isolamento *Serializable*, o que resultou em apenas uma reserva gravada com sucesso e nove solicitações recusadas por conflito de horário com código HTTP 409. O Quadro 2 apresenta a matriz de rastreabilidade que relaciona os requisitos levantados às evidências verificadas.

Quadro 2. Matriz de rastreabilidade e verificação de requisitos

ID	Requisito	Módulo	Critério e Evidência Verificada
RF01	Autenticação e cadastro de atletas e proprietários	*Authentication*	Emissão de JWT, *hash* PBKDF2 e validação Módulo 11 de CPF e CNPJ, verificados por 7 testes em *DocumentNumberTests* e 3 em *JwtOptionsValidationTests*.
RF02	Vitrine pública de quadras com filtros	*Venues*	Filtragem por cidade, nome e preço, ordenação e paginação, verificadas nos *endpoints* REST e na Figura 6.
RF03	Cálculo de disponibilidade diária e semanal	*Venues*	Classificação de intervalos livres, reservados e bloqueados, verificada por 4 testes em *AvailabilityDomainServiceTests* e nas Figuras 7 e 8.
RF04	Criação de reserva sem sobreposição	*Bookings*	Rejeição com erro HTTP 409 de horários sobrepostos, passados ou fora da grade, verificada por 12 testes em *BookingDomainServiceTests* e 5 em *CreateBookingHandlerTests*.
RF05	Alteração e cancelamento com regras de domínio	*Bookings*	Antecedência mínima de 2h para reserva confirmada e retorno ao estado *Pending* ao reagendar, verificados por 21 testes em *BookingTests* e 9 nos *handlers* de alteração e cancelamento.
RF06	Aprovação, recusa e conclusão de reservas	*Bookings*	Ações restritas ao proprietário da quadra e conclusão após o término do horário, verificadas por 9 testes nos *handlers* de confirmação e finalização.
RF07	Gestão de quadras, horários e bloqueios	*Venues* e *Blocks*	Validação de propriedade com retorno HTTP 403 para terceiros, verificada por 34 testes de domínio e 4 em *DeleteVenueHandlerTests*.
RF08	Avaliação de espaços	*Reviews*	Limite de uma avaliação por atleta e quadra via

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

	esportivos de 1 a 5		índice único e validação, verificado por 11 testes em *ReviewTests* e na Figura 10.
RF09	Painel gerencial com indicadores financeiros	*Users*	Cálculo de faturamento, perdas por cancelamento e gráficos por período, conforme demonstrado na Figura 9.
RNF01	Prevenção de condições de corrida em reservas	*BaseRepository*	Transação *Serializable* combinada ao *sp_getapplock*, com 1 sucesso e 9 conflitos em 10 chamadas paralelas no teste *BookingConcurrencyIntegrationTests*.

Fonte: Elaborado pelos autores (2026).

4. CONSIDERAÇÕES FINAIS

O presente trabalho desenvolveu e verificou o protótipo da plataforma *web* Vai Dar Jogo, reunindo funcionalidades de busca, agendamento e gestão de quadras esportivas privadas a partir de necessidades observadas em estabelecimentos de Araraquara-SP. A arquitetura construída em .NET 8 com os padrões *Clean Architecture*, DDD e CQRS, banco de dados SQL Server e interface em React com TypeScript atendeu aos requisitos funcionais e não funcionais definidos no projeto, conforme demonstrado na matriz de rastreabilidade do Quadro 2 e confirmado pela aprovação dos 171 testes automatizados, incluindo o bloqueio transacional contra agendamentos duplicados em acessos simultâneos.

Em comparação ao controle manual feito por aplicativos de mensagens e anotações, o sistema entrega funcionalidades verificadas de consulta autônoma de horários livres, validação automática do prazo mínimo para cancelamento, registro de bloqueios de agenda, publicação de avaliações e painel financeiro para os proprietários. Ressalta-se que esses resultados se referem às funcionalidades implementadas e testadas em nível de protótipo. Como limitações deste estudo, destacam-se a ausência de testes formais de usabilidade com usuários finais, como a aplicação de questionários padronizados, a não realização de testes de estresse sob alto volume de acessos simultâneos, o levantamento exploratório restrito a três estabelecimentos de uma mesma cidade e a necessidade de implementar fluxos completos de consentimento da LGPD e revisões de acessibilidade antes do uso comercial.

A plataforma apresenta potencial para contribuir com a digitalização dos processos de divulgação, consulta de disponibilidade e gerenciamento de reservas, dependendo de sua adoção e avaliação em contextos reais de utilização. Como trabalhos futuros, sugerem-se a integração de pagamentos digitais via Pix para confirmação automática das reservas no momento do agendamento, o desenvolvimento de um aplicativo móvel com envio de notificações

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

push aos usuários e a realização de estudos práticos de usabilidade e desempenho em estabelecimentos em operação.

REFERÊNCIAS

ALBAHARI, Joseph. *C# 12 in a Nutshell: The Definitive Reference*. Sebastopol: O'Reilly Media, 2024.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. The Unified Modeling Language User Guide. 2. ed. Boston: Addison-Wesley, 2005.

BRASIL. **Lei nº 13.709, de 14 de agosto de 2018**. Lei Geral de Proteção de Dados Pessoais (LGPD). Brasília, DF: Presidência da República, 2018. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm. Acesso em: 10 abr. 2026.

CHACON, Scott; STRAUB, Ben. ***Pro Git***. 2. ed. Nova York: Apress, 2014. Disponível em: https://git-scm.com/book/pt-br/v2. Acesso em: 30 maio 2026.

CÓDIGO FÁCIL. Microsoft SQL Server: o que é? ***Código Fácil***, 2026. Disponível em: https://codigofacil.com.br/microsoft-sql-server-o-que-e/. Acesso em: 4 abr. 2026.

EVANS, Eric. ***Domain-Driven Design****: Tackling Complexity in the Heart of Software*. Boston: Addison-Wesley, 2003.

FOWLER, Martin. ***UML Distilled****: A Brief Guide to the Standard Object Modeling Language*. 3. ed. Boston: Addison-Wesley, 2004.

FULL CYCLE. Domain-Driven Design (DDD): o que é e como aplicar na prática. ***Full Cycle***, 2019. Disponível em: https://fullcycle.com.br/domain-driven-design/. Acesso em: 5 abr. 2026.

HUI, Lim Yew; TEO, Poh-Chuin. An implementation of digital platform to enhance the appointment scheduling system. ***International Journal of Academic Research in Business and Social Sciences***, v. 14, n. 2, p. 402-416, 2024. DOI: 10.6007/IJARBSS/v14-i2/20635. Disponível em: https://hrmars.com/papers_submitted/20635/an-implementation-of-digital-platform-to-enhance-the-appointment-scheduling-system.pdf. Acesso em: 21 mar. 2026.

ISO/IEC. ***ISO/IEC 25010:2023****: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model*. Genebra: ISO, 2023.

KLEPPMANN, Martin. ***Designing Data-Intensive Applications****: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Sebastopol: O'Reilly Media, 2017.

MARTIN, Robert C. ***Clean Architecture****: A Craftsman's Guide to Software Structure and Design*. Boston: Prentice Hall, 2017.

MICROSOFT. C# docs – get started, tutorials, reference. ***Microsoft Learn***, 2026a. Disponível em: https://learn.microsoft.com/en-us/dotnet/csharp/. Acesso em: 3 abr. 2026.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.

MICROSOFT. CQRS pattern. ***Azure Architecture Center***, 2025. Disponível em: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs. Acesso em: 5 abr. 2026.

MICROSOFT. Common web application architectures. ***Microsoft Learn***, 2023a. Disponível em: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures. Acesso em: 20 mar. 2026.

MICROSOFT. Design a microservice domain model. ***Microsoft Learn***, 2022. Disponível em: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice. Acesso em: 5 abr. 2026.

MICROSOFT. Documentação do SQL Server. ***Microsoft Learn***, 2026b. Disponível em: https://learn.microsoft.com/pt-br/sql/sql-server/?view=sql-server-ver17. Acesso em: 4 abr. 2026.

MICROSOFT. What is .NET? ***Microsoft Learn***, 2023b. Disponível em: https://learn.microsoft.com/en-us/dotnet/core/introduction. Acesso em: 3 abr. 2026.

MOZILLA. Começando com React. ***MDN Web Docs***, 2026a. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn_web_development/Core/Frameworks_libraries/React_getting_started. Acesso em: 3 abr. 2026.

MOZILLA. Documentação de HTML, CSS e JavaScript. ***MDN Web Docs***, 2026b. Disponível em: https://developer.mozilla.org/pt-BR/. Acesso em: 3 abr. 2026.

REACT. Visão geral da referência do React. ***React.dev***, [s.d.]. Disponível em: https://pt-br.react.dev/reference/react. Acesso em: 3 abr. 2026.

SOMMERVILLE, Ian. ***Engenharia de Software***. 10. ed. São Paulo: Pearson Education do Brasil, 2018.

TYPESCRIPT. *TypeScript: JavaScript With Syntax For Types*. **Microsoft**, [s.d.]. Disponível em: https://www.typescriptlang.org/. Acesso em: 3 abr. 2026.

VĂRZARU, Anca Antoaneta; BOCEAN, Claudiu George. Digital transformation and innovation: the influence of digital technologies on turnover from innovation activities and types of innovation. ***Systems***, v. 12, n. 9, art. 359, 2024. DOI: 10.3390/systems12090359. Disponível em: https://www.mdpi.com/2079-8954/12/9/359. Acesso em: 21 mar. 2026.

VERNON, Vaughn. ***Implementing Domain-Driven Design***. Boston: Addison-Wesley, 2013.

Este artigo é publicado em acesso aberto (Open Access) sob a licença Creative Commons Atribuição 4.0 Internacional (CC-BY), que permite uso, distribuição e reprodução irrestritos em qualquer meio, desde que o autor original e a fonte sejam creditados.