



INSTITUTO FEDERAL GOIANO
CAMPUS URUTAÍ
NÚCLEO DE INFORMÁTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

ÍTALO GONÇALVES MEIRELES FARIA
JOÃO GABRIEL DE OLIVEIRA MEIRELES

SISTEMA DE GERENCIAMENTO CENTRALIZADO DE BUFFETS E EVENTOS

Urutaí, 13 de março de 2026

INSTITUTO FEDERAL GOIANO

NÚCLEO DE INFORMÁTICA

SISTEMAS DE INFORMAÇÃO

ÍTALO GONÇALVES MEIRELES FARIA
JOÃO GABRIEL DE OLIVEIRA MEIRELES

SISTEMA DE GERENCIAMENTO CENTRALIZADO DE BUFFETS E EVENTOS

Monografia apresentada ao Núcleo de Informática, curso de Sistemas de Informação, do Instituto Federal Goiano, como parte das exigências para obtenção do título de Bacharel em Sistemas de Informação.

Orientador(a):
Cristiane de Fátima dos Santos Cardoso

Urutaí, 13 de março de 2026

**Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema Integrado de Bibliotecas do IF Goiano - SIBi**

S623 Faria, Ítalo; Meireles, João Gabriel, .
SISTEMA DE GERENCIAMENTO CENTRALIZADO DE
BUFFETS E EVENTOS / . Faria, Ítalo; Meireles, João Gabriel.
Urutaí 2026.

153f. il.

Orientadora: Profª. Dra. CRISTIANE DE FÁTIMA DOS
SANTOS CARDOSO.
Tcc (Bacharel) - Instituto Federal Goiano, curso de 0120201 -
Bacharelado em Sistemas de Informação - Urutaí (Campus
Urutaí).
1. Buffets. 2. Eventos. 3. Aplicativo móvel. 4. Spring Boot. 5.
Angular. I. Título.

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO

PARA DISPONIBILIZAR PRODUÇÕES TÉCNICO-CIENTÍFICAS

NO REPOSITÓRIO INSTITUCIONAL DO IF GOIANO

Com base no disposto na Lei Federal nº 9.610, de 19 de fevereiro de 1998, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia Goiano a disponibilizar gratuitamente o documento em formato digital no Repositório Institucional do IF Goiano (RIIF Goiano), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IF Goiano.

IDENTIFICAÇÃO DA PRODUÇÃO TÉCNICO-CIENTÍFICA

- | | |
|--|---|
| ☐ Tese (doutorado) | ☐ Artigo científico |
| ☐ Dissertação (mestrado) | ☐ Capítulo de livro |
| ☐ Monografia (especialização) | ☐ Livro |
| ☒ TCC (graduação) | ☐ Trabalho apresentado em evento |
| ☐ Produto técnico e educacional - Tipo: | |

Nome completo do autor:

ÍTALO GONÇALVES M. FARIA e JOÃO GABRIEL DE O. MEIRELES

Matrícula:

2022101202010015

Título do trabalho:

SISTEMA DE GERENCIAMENTO CENTRALIZADO DE BUFFETS E EVENTOS

RESTRIÇÕES DE ACESSO AO DOCUMENTO

Documento confidencial: ☒ Não ☐ Sim, justifique:

Informe a data que poderá ser disponibilizado no RIIF Goiano: 13 /03 /26

O documento está sujeito a registro de patente? ☐ Sim ☒ Não

O documento pode vir a ser publicado como livro? ☐ Sim ☒ Não

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O(a) referido(a) autor(a) declara:

- Que o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
- Que obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autoria, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia Goiano os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
- Que cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia Goiano.

Documento assinado digitalmente
 ÍTALO GONÇALVES MEIRELES FARIA
Data: 13/03/2026 13:46:57-0300
Verifique em <https://validar.iti.gov.br>

Documento assinado digitalmente
 JOÃO GABRIEL DE OLIVEIRA MEIRELES
Data: 13/03/2026 13:49:05-0300
Verifique em <https://validar.iti.gov.br>

Urutaí

Local

13 /03 /26

Data

Assinatura do autor e/ou detentor dos direitos autorais

Ciente e de acordo:

Assinatura do(a) orientador(a)

Documento assinado digitalmente
 CRISTIANE DE FATIMA DOS SANTOS CARDOSO
Data: 13/03/2026 13:43:29-0300
Verifique em <https://validar.iti.gov.br>



SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA GOIANO

Formulário 13/2026 - CCBSI-URT/GE-UR/DE-UR/CMPURT/IFGOIANO

INSTITUTO FEDERAL GOIANO – CAMPUS URUTAÍ
DIRETORIA / GERÊNCIA DE GRADUAÇÃO E PÓS-GRADUAÇÃO
COORDENAÇÃO DOS CURSOS DA ÁREA DE INFORMÁTICA
CURSO SUPERIOR DE SISTEMAS DE INFORMAÇÃO

ATA DE APRESENTAÇÃO DE TRABALHO DE CURSO

Aos 26 dias do mês de fevereiro de dois mil e vinte e seis, reuniram-se os professores: Cristiane de Fátima dos Santos Cardoso, Nattane Luiza da Costa e Gabriel da Silva Vieira no núcleo de informática do Instituto Federal Goiano - Campus Urutaí, para avaliar o Trabalho de Curso do(s) acadêmico(s): **Ítalo Gonçalves Meireles Faria e João Gabriel de Oliveira Meireles**, como requisito necessário para a conclusão do Curso Superior de Sistemas de Informação desta Instituição. O presente TC tem como título: **SISTEMA DE GERENCIAMENTO CENTRALIZADO DE BUFFETS E EVENTOS**, foi orientado por Cristiane de Fátima dos Santos Cardoso.

Após análise, foram dadas as seguintes notas:

Professores	Aluno / Notas	
	João Gabriel de Oliveira Meireles	Ítalo Gonçalves Meireles Faria
1. Cristiane de Fátima dos Santos Cardoso	9	9,2
2. Nattane Luiza da Costa	9	9
3. Gabriel da Silva Vieira	8	8

MÉDIA FINAL:	8,7	8,7
---------------------	-----	-----

OBSERVAÇÕES: Correções conforme colocado pela banca dentro do período de 30 dias.

Por ser verdade firmamos a presente.

Documento assinado eletronicamente por:

- **Cristiane de Fatima dos Santos Cardoso**, COORDENADOR(A) DE CURSOS - FUC1 - CCBSI-URT , em 26/02/2026 12:52:55.
- **Nattane Luiza da Costa**, PROFESSOR ENS BASICO TECN TECNOLOGICO , em 26/02/2026 12:54:39.
- **Gabriel da Silva Vieira**, PROFESSOR ENS BASICO TECN TECNOLOGICO , em 26/02/2026 12:55:14.

Este documento foi emitido pelo SUAP em 26/02/2026. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifgoiano.edu.br/autenticar-documento/> e forneça os dados abaixo:

Código Verificador: 792972

Código de Autenticação: 8263691c24



INSTITUTO FEDERAL GOIANO

Campus Urutaí

Rodovia Geraldo Silva Nascimento, Km 2.5, SN, Zona Rural, URUTAÍ / GO, CEP 75790-000

(64) 3465-1900

AGRADECIMENTOS

Por Ítalo Faria

Agradeço primeiramente a Deus, por me conceder força, sabedoria e perseverança para superar os desafios e conduzir meus passos ao longo de toda a minha trajetória acadêmica. À minha família, expresso minha profunda gratidão pelo amor, apoio incondicional, incentivo constante e por acreditarem em meu potencial, sendo meu alicerce em todos os momentos desta caminhada. De maneira especial, agradeço à minha mãe, Daniella Paula, por nunca ter deixado faltar nada em nosso lar, por todos os esforços, renúncias e sacrifícios realizados para garantir meu bem-estar e minha educação, e por ser um exemplo de força, coragem e determinação diante das dificuldades. Ao meu avô, Silas Jetro, registro um agradecimento especial por ser uma grande figura paterna em minha vida, sempre presente com conselhos, ensinamentos e apoio, tornando-se um exemplo de caráter, responsabilidade e valores que levarei para toda a vida. Aos meus amigos, que estiveram presentes durante todo o processo, oferecendo apoio, motivação e companheirismo, registro minha sincera gratidão. Ao meu amigo João Gabriel, agradeço pela parceria, comprometimento e cooperação no desenvolvimento deste trabalho, tornando essa experiência mais produtiva e enriquecedora.

Por João Gabriel

Agradeço, primeiramente, a Deus, por me conceder força, saúde e perseverança ao longo de toda a graduação. Aos meus pais, Maria Lúcia e Carlúcio, meu eterno agradecimento por todo o apoio, incentivo e confiança. Obrigado por acreditarem em mim mesmo nos momentos mais difíceis e por nunca me deixarem desistir dos meus sonhos. Tudo o que conquistei até aqui também é fruto do amor e da dedicação de vocês. Ao meu colega e amigo, Ítalo, com quem tive a honra de desenvolver este trabalho. Obrigado pela parceria, companheirismo, paciência e amizade ao longo dessa jornada. Aos meus amigos Breno, Diego, Filipe, Flávio e José, agradeço por todos os momentos compartilhados, pelas risadas, conversas, apoio e por tornarem essa caminhada muito mais especial. À minha namorada Deise, meu sincero agradecimento por todo o apoio, compreensão e carinho. Mesmo nos momentos mais difíceis, você esteve ao meu lado, acreditando em mim e me dando forças para continuar. Sua presença foi essencial para que eu chegasse até aqui. Por fim, a todos que, direta ou indiretamente, contribuíram para a realização deste trabalho e para minha formação pessoal e acadêmica, meu muito obrigado.

A gestão de *buffets* e eventos ainda é frequentemente realizada por meio de processos descentralizados, como planilhas eletrônicas, aplicativos de mensagens e registros manuais, o que pode comprometer a organização das informações, dificultar o acompanhamento das reservas e aumentar a incidência de erros operacionais. Diante desse cenário, este trabalho apresenta o desenvolvimento de um aplicativo móvel para Android destinado ao gerenciamento centralizado de *buffets* e eventos, integrando cadastro de comidas e serviços, controle de reservas e personalização visual em um único ambiente. A metodologia adotada compreendeu levantamento e análise de requisitos, definição de perfis de usuários e modelagem conceitual por meio de diagramas *UML*. A solução proposta utiliza arquitetura *multi-tenant*, com *back-end* implementado em *Java* por meio do *Spring Boot* e exposição de serviços via *API RESTful*, enquanto o *front-end* foi desenvolvido com *Angular*, *Ionic* e *Capacitor*. Entre as funcionalidades implementadas destacam-se autenticação baseada em *JWT*, gerenciamento de comidas e serviços, criação e aprovação de reservas com geração automática de eventos e envio de notificações de estado. Como resultado, obteve-se uma aplicação funcional que atende aos requisitos definidos e centraliza, de forma estruturada, os processos de gestão de *buffets* e eventos em um único ambiente digital.

Palavras-chave:

Buffets, Eventos, Aplicativo móvel, *Spring Boot*, *Angular*, *Ionic*, *Multi-tenant*.

LISTA DE FIGURAS

4.1	Diagrama de casos de uso do sistema.	33
4.2	Diagrama de classes do sistema.	38
4.3	Diagrama de sequência do caso de uso UC01 – Autenticar usuário.	41
4.4	Diagrama de sequência do caso de uso UC05 – Criar reserva.	43
4.5	Diagrama de sequência do caso de uso UC06 – Aprovar reserva.	45
5.1	Estrutura de pacotes do Back-end	49
5.2	Diagrama do banco de dados.	76
6.1	Estrutura de pacotes do front-end baseada em Feature-Sliced Design.	80
7.1	Tela de abertura do aplicativo.	103
7.2	Tela de boas-vindas do aplicativo.	104
7.3	Tela de cadastro de novo cliente.	105
7.4	Tela de confirmação de cadastro.	106
7.5	Tela de Termos e Condições de Uso.	107
7.6	Tela de login.	108
7.7	Mensagem de erro de login inválido.	109
7.8	Tela de solicitação de recuperação de senha.	110
7.9	Tela de validação do código de recuperação.	111
7.10	Tela de redefinição de senha.	112
7.11	Tela inicial do cliente.	113
7.12	Catálogo de comidas disponíveis.	114
7.13	Detalhes de uma comida.	115
7.14	Catálogo de serviços disponíveis.	116
7.15	Detalhes de um serviço.	117
7.16	Carrinho de pedido com itens selecionados.	118
7.17	Tela de detalhes da reserva (nome, descrição, data, horário e quantidade de pessoas).	119
7.18	Tela de endereço do evento.	121
7.19	Tela de confirmação de reserva realizada.	122
7.20	Tela de perfil do cliente.	123
7.21	Tela de edição de perfil do cliente.	124
7.22	Tela de listagem de reservas.	125
7.23	Tela de detalhes da reserva.	127

7.24	Tela de edição completa da reserva.	128
7.25	Tela de listagem de eventos.	129
7.26	Tela de detalhes do evento.	131
7.27	Tela de edição de evento (cliente).	133
7.28	Tela de notificações.	134
7.29	Dashboard administrativo do buffet.	135
7.30	Tela de gerenciamento de comidas.	136
7.31	Formulário de cadastro/edição de comida.	137
7.32	Modal de confirmação de exclusão de comida.	138
7.33	Tela de gerenciamento de serviços.	139
7.34	Formulário de cadastro/edição de serviço.	140
7.35	Modal de confirmação de exclusão de serviço.	141
7.36	Modal de confirmação de saída.	142
7.37	Modals de confirmação de cancelamento.	143
7.38	Modal de aprovação de reserva com definição de valor.	144
7.39	<i>Badges</i> de estados de reservas e eventos.	146

LISTA DE TABELAS

2.1	Aspectos do atendimento ao cliente em sistemas de reserva.	18
2.2	Estrutura conceitual do JWT.	19
2.3	Comparação de características entre soluções existentes para gerenciamento de eventos.	20
3.1	Características do modelo multi-buffet aplicado ao sistema.	22
3.2	Princípios da arquitetura REST e sua aplicação conceitual no sistema.	22
3.3	Principais fluxos conceituais no gerenciamento de buffets.	24
3.4	Tabela consolidada de tecnologias, aspectos e descrições.	25
4.1	Requisitos funcionais do sistema.	29
4.2	Requisitos não funcionais do sistema.	30
4.3	Casos de uso do sistema.	34
4.4	Descrição do Caso de Uso UC02 – Autenticar usuário	35
4.5	Descrição do Caso de Uso UC05 – Criar reserva	36
4.6	Descrição do Caso de Uso UC11 – Aprovar reserva	37
5.1	Dependências utilizadas no projeto e suas descrições.	48
5.2	Endpoints de autenticação da API.	49
5.3	Endpoint de gerenciamento do perfil do cliente.	50
5.4	Endpoints de gerenciamento de buffets.	50
5.5	Endpoints de gerenciamento de comidas.	51
5.6	Endpoints de gerenciamento de serviços.	51
5.7	Endpoints de gerenciamento de eventos.	52
5.8	Endpoints de gerenciamento de reservas pelo buffet.	53
5.9	Endpoints de gerenciamento de reservas pelo cliente.	54
5.10	Endpoints de gerenciamento de notificações.	55
5.11	Endpoint de upload de arquivos.	55
5.12	Endpoints de recuperação de senha.	56
6.1	Principais dependências do <i>front-end</i> Angular/Ionic.	79
7.1	Comparativo entre soluções existentes e o eBuffet	147

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface.	Pag.22
HTTP	HyperText Transfer Protocol.	Pag.22
IDE	Integrated Development Environment.	Pag.25
JWT	<i>JSON Web Token</i> .	Pag.18
REST	Representational State Transfer.	Pag.22
SGBD	Sistema Gerenciador de Banco de Dados.	Pag.25
UI	User Interface.	Pag.25
UML	Unified Modeling Language.	Pag.32
URIs	Uniform Resource Identifiers.	Pag.22

1	INTRODUÇÃO	15
2	FUNDAMENTOS TEÓRICOS	17
2.1	Sistemas de Gerenciamento de Buffets e Eventos	17
2.2	Atendimento ao Cliente e Sistemas de Reserva	18
2.3	Segurança e Autenticação	18
2.4	Comparação com Soluções Existentes	19
3	MATERIAIS E MÉTODOS	21
3.1	Isolamento de Dados	21
3.2	Arquitetura REST e Aplicação Móvel Híbrida	22
3.3	Levantamento e Análise de Requisitos	23
3.4	Fluxos de Negócio	24
3.5	Definição de Tecnologias e Ferramentas	24
4	ANÁLISE E PROJETO DO SISTEMA	27
4.1	Perfis de Usuários	27
4.2	Requisitos Funcionais	28
4.3	Requisitos Não Funcionais	30
4.4	Regras de Negócio	30
4.5	Diagramas UML	32
4.5.1	Diagrama de Casos de Uso	33
4.5.2	Diagrama de Classes	37
4.5.3	Diagramas de Sequência	40
5	IMPLEMENTAÇÃO DO BACK-END (API RESTFUL)	47
5.1	Configuração Inicial	47
5.2	Estrutura de Pacotes do <i>Back-end</i>	48
5.3	Endpoints da API	49
5.3.1	Endpoints de Autenticação	49
5.3.2	Endpoints de Clientes	50
5.3.3	Endpoints de Buffets	50
5.3.4	Endpoints de Comidas	50
5.3.5	Endpoints de Serviços	51

5.3.6	Endpoints de Eventos	51
5.3.7	Endpoints de Reservas (Buffet)	53
5.3.8	Endpoints de Reservas (Cliente)	54
5.3.9	Endpoints de Notificações	55
5.3.10	Endpoints de Upload	55
5.3.11	Endpoints de Recuperação de Senha	55
5.4	Trechos de Código	56
5.4.1	Autenticação	56
5.4.2	Registro de usuários	59
5.4.3	Reservas	63
5.4.4	Evento	70
5.5	Projeto do Banco de Dados	76
6	IMPLEMENTAÇÃO DO <i>FRONT-END</i> (ANGULAR/IONIC)	78
6.1	Configuração inicial	78
6.2	Organização de pastas	79
6.3	<i>Login</i> e autenticação	81
6.4	Proteção de rotas e controle de perfil	84
6.5	Navegação e áreas por perfil	87
6.6	Redefinição de senha	88
6.7	Gerenciamento de comidas	88
6.8	Gerenciamento de serviços	89
6.9	Gerenciamento de reservas	90
6.10	Gerenciamento de eventos	91
6.11	Notificações	92
6.12	Upload e gerenciamento de imagens	92
6.13	Cliente <i>HTTP</i> centralizado	93
6.14	Componentes reutilizáveis	94
6.15	Arquitetura <i>Multi-Tenant</i>	95
6.15.1	Organização de recursos estáticos	96
6.15.2	Carregamento e aplicação dinâmica de tema	97
6.15.3	Isolamento de dados por <i>tenant</i>	99
6.15.4	Configuração de ambiente	100
6.15.5	Vantagens da arquitetura <i>multi-tenant</i>	101
7	RESULTADOS E DISCUSSÃO	102
7.1	Área Pública	103
7.1.1	Tela Inicial (<i>Index</i>)	103
7.1.2	Tela de Boas-Vindas	104
7.1.3	Tela de Cadastro	105
7.1.4	Tela de <i>Login</i>	108
7.1.5	Recuperação de Senha	110
7.2	Área do Cliente	113
7.2.1	<i>Home</i> do Cliente	113
7.2.2	Catálogo de Comidas	114
7.2.3	Catálogo de Serviços	116
7.2.4	Carrinho de Pedido (Nova Reserva)	118
7.2.5	Perfil do Cliente	123

7.3	Área Global (Cliente e Buffet)	125
7.3.1	Listagem de Reservas	125
7.3.2	Listagem de Eventos	129
7.3.3	Notificações	134
7.4	Área Administrativa do Buffet	135
7.4.1	<i>Dashboard</i> do Buffet	135
7.4.2	Gerenciamento de Comidas	136
7.4.3	Gerenciamento de Serviços	139
7.5	Modais de Confirmação	142
7.6	Estados Visuais dos Cards	145
7.6.1	Estados de Reservas	145
7.6.2	Estados de Eventos	145
7.6.3	Mapeamento Visual	146
7.7	Discussão	146
8	CONCLUSÃO	148
8.1	Limitações do Estudo	149

CAPÍTULO 1

INTRODUÇÃO

A organização de eventos e a prestação de serviços por buffets compõem um segmento dinâmico e multifacetado que engloba desde o planejamento logístico até a personalização de serviços para atender diferentes perfis de clientes. No contexto brasileiro, o setor de eventos e seu hub setorial representam aproximadamente 3,8% do Produto Interno Bruto (PIB), movimentando cerca de R\$ 291,1 bilhões anuais e envolvendo mais de 6,6 milhões de pessoas em toda a cadeia produtiva, evidenciando sua expressiva relevância econômica e social (ABRAPE, 2023). Buffets, protagonistas nesse cenário, precisam gerenciar reservas, disponibilizar suas comidas e serviços e cultivar a fidelização dos clientes. A carência de soluções tecnológicas centralizadas e flexíveis voltadas à gestão de buffets compromete a eficiência desses serviços e limita seu crescimento. Em paralelo, a transformação digital remodela a forma como empresas produzem, distribuem e disponibilizam serviços, impactando profundamente o setor de eventos ao passo que exige intensa interação entre pessoas, ambiente e gestão para proporcionar experiências memoráveis (PICCIONI, 2023).

Nos últimos anos, soluções baseadas em computação em nuvem e aplicativos móveis têm possibilitado avanços significativos no setor de hospitalidade, promovendo maior automatização dos processos, personalização dos serviços e acesso a informações em tempo real. Entretanto, as alternativas gratuitas e de código aberto atualmente disponíveis, como Attendize (Attendize, 2023), Odoo Events (Odoo, 2024) e Eventbrite (Eventbrite, 2023), apresentam foco predominante no gerenciamento genérico de eventos, não contemplando de forma adequada funcionalidades essenciais para a administração de buffets. Entre essas limitações destacam-se a ausência de mecanismos para o cadastro e a customização detalhada das comidas oferecidas, o controle

estruturado dos serviços disponibilizados e o gerenciamento da identidade visual da interface de forma individualizada para cada buffet. Além disso, pesquisas apontam que o uso de etiquetas de refeição em papel eleva os custos e sobrecarrega funcionários, enquanto a adoção de tablets e tecnologias *paperless* aumenta a eficiência, reduz despesas e agiliza a tomada de decisão (ALHARTHI et al., 2020). Esses fatores evidenciam a necessidade de uma solução que integre as operações de buffets e eventos em um único ambiente digital.

Nesse contexto, a proposta deste projeto surge para suprir tais lacunas, ao considerar aspectos como a separação entre reservas e eventos, a administração detalhada de comidas e serviços, a personalização por estabelecimento e a centralização do gerenciamento em uma única plataforma. Essa abordagem busca proporcionar maior aderência às necessidades operacionais dos buffets, bem como oferecer uma experiência mais clara, organizada e eficiente para os clientes, atendendo a um nicho que não é plenamente contemplado pelas soluções atualmente disponíveis no mercado.

Visando concretizar essa proposta, este trabalho apresenta o desenvolvimento do **eBuffet**, um aplicativo móvel para Android destinado ao gerenciamento centralizado de buffets e eventos, no qual clientes realizam reservas de forma estruturada e os responsáveis pelo buffet administram reservas, eventos, comidas, serviços e identidade visual em um único ambiente digital. Entre os principais diferenciais do sistema destacam-se a personalização visual da interface por estabelecimento, a separação conceitual entre reservas e eventos, funcionalidades específicas para o domínio de buffets e a integração completa entre *back-end* e *front-end* em uma arquitetura orientada a serviços. Essa iniciativa justifica-se pela demanda crescente por soluções que promovam integração, controle e autonomia no setor de eventos e pela ausência de plataformas especializadas que atendam às especificidades operacionais desse segmento.

Este trabalho encontra-se estruturado de forma progressiva. O Capítulo 2 apresenta os fundamentos teóricos que embasam a pesquisa. O Capítulo 3 descreve os materiais e métodos adotados, incluindo arquitetura, requisitos e tecnologias empregadas. O Capítulo 4 aborda a análise e o projeto do sistema, contemplando perfis de usuários, requisitos, regras de negócio e modelagens UML. O Capítulo 5 detalha a implementação do *back-end* por meio de uma API RESTful. O Capítulo 6 trata da implementação do *front-end* com Angular e Ionic. O Capítulo 7 apresenta a aplicação desenvolvida e suas funcionalidades. Por fim, o Capítulo 8 expõe as conclusões do trabalho.

CAPÍTULO 2

FUNDAMENTOS TEÓRICOS

Este capítulo apresenta os fundamentos teóricos que embasam o desenvolvimento do sistema proposto. São abordados conceitos relacionados a sistemas de gerenciamento de buffets e eventos, práticas de atendimento ao cliente e informatização de reservas, mecanismos de segurança e autenticação, bem como a análise comparativa de soluções existentes no mercado. Esses referenciais teóricos sustentam as decisões adotadas no projeto e delimitam o contexto tecnológico e funcional em que a solução está inserida.

2.1 Sistemas de Gerenciamento de Buffets e Eventos

Sistemas de gerenciamento de buffets e eventos têm como principal objetivo organizar e centralizar informações relacionadas à prestação de serviços em eventos sociais e corporativos. Esse tipo de sistema surge como resposta à complexidade operacional do setor, que envolve a gestão simultânea de reservas, oferta de comidas, serviços complementares e comunicação contínua com clientes (ALHARTHI et al., 2020).

Diferentemente de plataformas genéricas de gerenciamento de eventos, frequentemente voltadas à venda de ingressos ou controle de presença, sistemas destinados a buffets precisam atender a requisitos específicos do domínio. Entre esses requisitos destacam-se a gestão detalhada das comidas oferecidas, sua associação a eventos e reservas, bem como a personalização visual conforme a identidade de cada buffet. A baixa oferta de soluções especializadas costuma resultar em processos fragmentados, uso excessivo de ferramentas informais e dificuldade na consolidação das informações operacionais.

Nesse contexto, sistemas centralizados de gerenciamento favorecem a padronização dos processos, reduzem falhas operacionais e melhoram a organização das informações. Além disso, contribuem para a escalabilidade do negócio, ao permitir que diferentes buffets operem sobre uma mesma base tecnológica, mantendo autonomia sobre seus dados e serviços.

2.2 Atendimento ao Cliente e Sistemas de Reserva

No setor de eventos, a experiência do cliente está diretamente relacionada à clareza das informações, à organização do processo de reserva e à capacidade de resposta do buffet. Sistemas informatizados de reserva contribuem para a centralização das solicitações, registro histórico das interações e acompanhamento do status dos pedidos, reduzindo inconsistências típicas de atendimentos conduzidos por múltiplos canais (PICCIONI, 2023).

Nesse contexto, a Tabela 2.1 apresenta alguns dos principais aspectos do atendimento ao cliente que são impactados pela utilização de sistemas informatizados de reserva, evidenciando benefícios como maior rastreabilidade das solicitações, padronização das informações fornecidas aos clientes e redução da dependência de controles manuais.

Tabela 2.1: Aspectos do atendimento ao cliente em sistemas de reserva.

Aspecto	Impacto em sistemas de reserva
Registro histórico	Rastreabilidade de reservas e eventos ao longo do tempo.
Padronização do atendimento	Consistência nas informações fornecidas aos clientes.
Acompanhamento de status	Transparência no andamento das solicitações.
Redução de fricção	Menor dependência de mensagens e controles manuais.

A centralização dessas informações reduz falhas de comunicação, minimiza dependência de canais informais e melhora a previsibilidade operacional. Para buffets, isso se traduz em melhor planejamento de recursos, redução de conflitos de agenda e maior confiabilidade no atendimento.

2.3 Segurança e Autenticação

Sistemas que lidam com informações pessoais e contratuais devem adotar mecanismos de segurança adequados. A autenticação baseada em JWT (*JSON Web Token*) é amplamente utilizada por permitir controle de acesso seguro sem a necessidade de gerenciamento de sessões

no servidor, favorecendo cenários distribuídos e aplicações que consomem APIs (JONES et al., 2015).

O JWT é composto por cabeçalho, corpo e assinatura, permitindo que o servidor valide a integridade do token e utilize suas *claims* para apoiar decisões de autenticação e autorização.

Conforme apresentado na Tabela 2.2, o *header* identifica o algoritmo e o tipo do token, o *payload* armazena as informações e escopos de acesso, enquanto a *signature* garante a integridade do conteúdo, permitindo verificar se o token sofreu alterações indevidas.

Tabela 2.2: Estrutura conceitual do JWT.

Elemento	Função
Header	Define algoritmo e tipo do token.
Payload	Armazena informações e escopos de acesso.
Signature	Garante que o token não foi alterado.

2.4 Comparação com Soluções Existentes

Atualmente, existem diversas plataformas voltadas ao gerenciamento de eventos que atendem a um público amplo e a diferentes tipos de organização. Soluções como Attendize (Attendize, 2023), Odoo Events (Odoo, 2024) e Eventbrite (Eventbrite, 2023) disponibilizam funcionalidades consolidadas para cadastro de eventos, controle de participantes e divulgação, sendo amplamente adotadas em cenários genéricos de organização de eventos.

Apesar de sua popularidade, essas ferramentas apresentam limitações quando aplicadas a contextos mais específicos, como o gerenciamento de buffets. Em especial, observa-se a ausência de suporte à administração integrada de comidas, serviços complementares e reservas personalizadas por estabelecimento, bem como à operação simultânea de múltiplos buffets de forma independente dentro de uma mesma solução. Cabe ressaltar que tais restrições persistem mesmo nos planos pagos dessas plataformas, o que reforça a existência de um nicho funcional não atendido pelas alternativas atualmente disponíveis no mercado.

A Tabela 2.3 apresenta uma comparação conceitual entre as principais características dessas soluções existentes, elaborada a partir da análise direta de suas documentações e funcionalidades disponíveis, evidenciando seu foco predominantemente genérico e as restrições relacionadas à gestão específica de buffets e eventos personalizados.

Tabela 2.3: Comparação de características entre soluções existentes para gerenciamento de eventos.

Característica	Attendize	Odoo Events	Eventbrite
Gestão integrada de comidas	Não	Limitada	Não
Gerenciamento de serviços adicionais	Não	Limitado	Parcial
Personalização visual por estabelecimento	Não	Não	Parcial
Suporte a múltiplos buffets independentes	Não	Não	Não
Aplicativo móvel dedicado para clientes	Não	Não	Parcial
Foco em reservas personalizadas por buffet	Não	Não	Não

Dessa forma, observa-se que, embora as plataformas analisadas ofereçam funcionalidades consolidadas para a organização geral de eventos, não contemplam de forma estruturada as especificidades inerentes à gestão de buffets. As limitações identificadas quanto à administração detalhada de comidas, serviços personalizados e operação independente por estabelecimento evidenciam um recorte funcional ainda pouco explorado pelas soluções atualmente disponíveis.

Neste capítulo, foram apresentados os fundamentos teóricos que sustentam o desenvolvimento do sistema, incluindo conceitos sobre sistemas de gerenciamento de buffets e eventos, práticas de atendimento ao cliente e informatização de reservas, mecanismos de segurança baseados em JWT e uma comparação com as principais soluções existentes. A análise evidenciou que as plataformas disponíveis no mercado não contemplam de forma adequada as especificidades inerentes à gestão de buffets, reforçando a pertinência da proposta deste trabalho. Com esse embasamento teórico estabelecido, o próximo capítulo descreve os materiais e métodos adotados no desenvolvimento, abordando o modelo multi-buffet, a arquitetura REST, o levantamento de requisitos e a definição das tecnologias utilizadas.

CAPÍTULO 3

MATERIAIS E MÉTODOS

Este capítulo apresenta os materiais, métodos e decisões técnicas que fundamentaram o desenvolvimento do sistema proposto. Inicialmente, são descritos os aspectos conceituais relacionados ao modelo multi-buffet e aos fluxos de negócio do domínio. Em seguida, são abordados os fundamentos de integração por meio de APIs RESTful e da arquitetura *mobile* adotada. Posteriormente, são detalhadas as etapas de levantamento e análise de requisitos, bem como a definição das tecnologias e ferramentas utilizadas no projeto. Essa organização busca evidenciar a coerência metodológica adotada ao longo do desenvolvimento.

3.1 Isolamento de Dados

Plataformas multi-inquilino (*multi-tenant*) referem-se à capacidade de um sistema de atender simultaneamente a múltiplos clientes ou aplicações, garantindo o isolamento lógico de seus dados e recursos, mesmo quando compartilham a mesma infraestrutura computacional (GEORGIU et al., 2025).

Para o sistema proposto, foram consideradas duas abordagens arquiteturais: a *single-tenant*, na qual cada buffet operaria sobre uma instância de aplicação separada, e a *multi-tenant*, na qual uma única instância atende a múltiplos estabelecimentos com isolamento lógico de dados. Optou-se pelo modelo *multi-tenant* por oferecer maior escalabilidade e facilidade de manutenção, uma vez que a evolução da plataforma ocorre em um único código-base, enquanto as particularidades de cada buffet são tratadas por meio de parametrizações associadas à sua identificação. A Tabela 3.1 apresenta as principais características desse modelo e sua aplicação no sistema.

Tabela 3.1: Características do modelo multi-buffet aplicado ao sistema.

Elemento	Aplicação no sistema
Isolamento lógico por buffet	Entidades associadas a um buffet específico.
Identidade visual personalizada	Aplicação dinâmica de cores e logotipo.
Padronização de processos	Fluxos comuns com parametrização por buffet.
Escalabilidade operacional	Inclusão de novos buffets na mesma base tecnológica.

3.2 Arquitetura REST e Aplicação Móvel Híbrida

Uma API (Application Programming Interface) define um conjunto de regras e especificações que permitem a comunicação entre sistemas distintos, estabelecendo contratos de interoperabilidade entre aplicações (AMUNDSEN, 2022). No contexto do sistema desenvolvido neste trabalho, a API RESTful atua como camada intermediária entre o *front-end* e o banco de dados, promovendo a separação de responsabilidades e contribuindo para a escalabilidade, manutenção e organização da aplicação.

O estilo arquitetural REST (Representational State Transfer), formalizado por Fielding em sua tese de doutorado (FIELDING, 2000), fundamenta-se na manipulação de recursos identificados por URIs (Uniform Resource Identifiers), utilizando métodos do HTTP (HyperText Transfer Protocol) e adotando o princípio da comunicação sem estado (*stateless*). Nesse modelo, cada requisição realizada pelo cliente deve conter todas as informações necessárias para seu processamento, não havendo armazenamento de contexto no servidor entre requisições consecutivas. APIs RESTful promovem interoperabilidade, baixo acoplamento entre cliente e servidor e maior flexibilidade evolutiva dos sistemas (RICHARDSON; RUBY, 2007).

Os principais princípios que caracterizam a arquitetura REST e sua aplicação conceitual no sistema proposto estão sintetizados na Tabela 3.2.

Tabela 3.2: Princípios da arquitetura REST e sua aplicação conceitual no sistema.

Princípio REST	Aplicação conceitual
Cliente-servidor	Separação entre a interface de usuário (<i>front-end</i>) e a lógica de negócio implementada no <i>back-end</i> .
Stateless	Requisições independentes entre si, contendo todas as informações necessárias ao processamento.
Interface uniforme	Padronização na exposição e manipulação de recursos por meio de métodos HTTP.
Representações	Utilização de formatos estruturados, como JSON, para troca de dados entre cliente e servidor.

Adicionalmente, a arquitetura adotada neste trabalho integra uma aplicação *mobile* híbrida a uma API RESTful, permitindo que a lógica de negócio permaneça centralizada no *back-end*, enquanto a interface móvel opera como camada de apresentação e interação com o usuário (Ionic Framework, 2024; Ionic Team, 2024). Essa organização arquitetural favorece a manutenção e a escalabilidade do sistema, uma vez que as regras de negócio permanecem desacopladas da interface, possibilitando evolução independente dos componentes.

Dessa forma, assegura-se maior organização estrutural do sistema, pois cada camada desempenha funções específicas e bem delimitadas. A centralização das regras de negócio no servidor contribui para a padronização do processamento das informações e para a redução de inconsistências, além de facilitar futuras atualizações ou ajustes na aplicação, preservando a integridade das funcionalidades implementadas.

3.3 Levantamento e Análise de Requisitos

O levantamento de requisitos do sistema foi realizado a partir da análise da necessidade de organização enfrentada por um buffet que não dispunha de ferramenta digital adequada para gerenciar suas atividades. Observou-se que o controle de reservas, eventos, comidas e serviços era conduzido de forma descentralizada, com uso predominante de anotações manuais e registros em papel, o que dificultava o acompanhamento das informações e aumentava a possibilidade de erros operacionais.

Como estratégia de levantamento, realizou-se consulta à proprietária de um buffet, por meio de entrevistas informais e observação do funcionamento cotidiano do estabelecimento. A partir desse contato, foi possível compreender como os processos eram organizados e executados, identificando-se as principais dificuldades relacionadas à gestão de reservas, ao controle de serviços oferecidos e à necessidade de sistematização das informações. Essas interações permitiram estruturar as regras de negócio do sistema, posteriormente detalhadas no Capítulo 4.

Diante desse contexto, os requisitos foram definidos com base nos fluxos essenciais do domínio de buffets e eventos, considerando a interação entre clientes, responsáveis pela criação de reservas, e buffets, encarregados da avaliação, aprovação e organização dos eventos. O levantamento concentrou-se na identificação das funcionalidades necessárias para centralizar esses processos de forma coerente e estruturada em uma plataforma digital voltada à organização e ao gerenciamento de buffets e eventos.

A análise dos requisitos teve como objetivo organizar as necessidades identificadas para o desenvolvimento do sistema, classificando-as em requisitos funcionais e não funcionais, bem como definindo os perfis de usuários que interagem com a plataforma. Essa etapa possibilitou compreender como os diferentes papéis utilizam o sistema e quais responsabilidades assumem no domínio do problema, servindo de base para a modelagem conceitual e para as decisões relacionadas à implementação.

3.4 Fluxos de Negócio

Conforme foi percebido na fase de levantamento de requisitos, a gestão de buffets envolve processos recorrentes relacionados à oferta de comidas, disponibilização de serviços adicionais, cadastro de eventos e administração de reservas (ALHARTHI et al., 2020). Em ambientes digitais, esses fluxos devem ser organizados de maneira estruturada para reduzir inconsistências operacionais.

No sistema proposto, tais fluxos foram modelados de forma conceitual, delimitando o domínio do problema antes da formalização dos requisitos específicos. A Tabela 3.3 sintetiza os principais fluxos conceituais identificados e suas respectivas descrições.

Tabela 3.3: Principais fluxos conceituais no gerenciamento de buffets.

Fluxo	Descrição conceitual
Reservas	Organização da solicitação, análise e confirmação.
Comidas	Oferta e vinculação às reservas e eventos.
Serviços adicionais	Inclusão de serviços complementares.
Eventos	Consolidação das informações da execução.

3.5 Definição de Tecnologias e Ferramentas

As tecnologias foram selecionadas com base em critérios de robustez, escalabilidade, compatibilidade *mobile* e ampla adoção no mercado. Considerou-se ainda a disponibilidade de documentação e aderência às boas práticas de desenvolvimento.

A Tabela 3.4 apresenta as principais tecnologias e ferramentas utilizadas.

Tabela 3.4: Tabela consolidada de tecnologias, aspectos e descrições.

Tecnologia	Aspecto	Descrição
<i>Java</i>	Linguagem	Linguagem utilizada no desenvolvimento do back-end da aplicação, oferecendo robustez, portabilidade e amplo suporte a aplicações corporativas.
<i>Spring Boot</i>	Framework Back-end	Framework para aplicações Java responsável pela implementação da API REST, gerenciamento de dependências e organização das regras de negócio.
<i>PostgreSQL</i>	SGBD (Sistema Gerenciador de Banco de Dados)	Sistema gerenciador de banco de dados relacional utilizado para persistência das informações, garantindo integridade e consistência dos dados.
<i>Angular</i>	Framework Front-end	Framework utilizado na construção da interface do aplicativo, fornecendo estrutura baseada em componentes e alta produtividade no desenvolvimento.
<i>Ionic</i>	Framework <i>Mobile</i>	Framework híbrido utilizado para desenvolvimento do aplicativo <i>mobile</i> , permitindo a criação de interfaces compatíveis com Android e iOS.
<i>Capacitor</i>	Build <i>Mobile</i>	Ferramenta responsável pelo empacotamento do aplicativo Angular/Ionic e pelo acesso a recursos nativos do dispositivo móvel.
<i>Feature-Sliced Design</i>	Arquitetura de UI (User Interface)	Abordagem arquitetural adotada para organização do código front-end de forma modular, escalável e orientada a funcionalidades.
<i>Cloudinary</i>	Armazenamento de Imagens	Serviço utilizado para armazenamento, gerenciamento e otimização de imagens do sistema, como logotipos, banners e fotos de perfil.
<i>Git</i>	Versionamento de Código	Sistema de controle de versão distribuído utilizado para gerenciamento do código-fonte e controle do histórico de alterações.
<i>GitHub</i>	Repositório Remoto	Plataforma utilizada para hospedagem do repositório do projeto, colaboração entre desenvolvedores e controle de versões em nuvem.
<i>Postman</i>	Testes de API	Ferramenta utilizada para testes, validação e documentação das requisições realizadas na API REST do sistema.
<i>IntelliJ IDEA</i>	IDE (Integrated Development Environment)	Ambiente de desenvolvimento integrado utilizado no desenvolvimento do back-end em Java e Spring Boot.
<i>Android Studio</i>	IDE (Integrated Development Environment)	Ambiente de desenvolvimento utilizado para executar, testar e gerar o aplicativo Android.

Este capítulo apresentou os materiais e métodos que fundamentaram o desenvolvimento do sistema, descrevendo o modelo de isolamento de dados *multi-tenant*, a arquitetura *REST* integrada à aplicação *mobile* híbrida, o processo de levantamento e análise de requisitos junto a um buffet real, os fluxos de negócio do domínio e as tecnologias selecionadas para o projeto. Deve-se observar que a metodologia adotada buscou assegurar coerência entre as decisões técnicas e as necessidades operacionais identificadas. Assim, o próximo capítulo trata da análise e projeto do sistema, detalhando os perfis de usuários, os requisitos funcionais e não funcionais, as regras de negócio e a modelagem UML que orientou a implementação.

CAPÍTULO 4

ANÁLISE E PROJETO DO SISTEMA

Este capítulo apresenta a análise e o projeto do sistema, descrevendo os elementos que estruturam sua concepção e orientam a implementação. Inicialmente, são definidos os perfis de usuários considerados no domínio e suas respectivas responsabilidades. Em seguida, são especificados os requisitos funcionais e não funcionais, bem como as regras de negócio que delimitam o comportamento da plataforma. Por fim, são apresentados os principais artefatos de modelagem, por meio de diagramas UML, incluindo casos de uso, diagrama de classes e diagramas de sequência, com o objetivo de representar de forma padronizada as funcionalidades, a estrutura do domínio e as interações entre os componentes do sistema.

4.1 Perfis de Usuários

O escopo deste projeto delimita-se ao desenvolvimento de uma plataforma digital para o gerenciamento de buffets e eventos, contemplando dois perfis de usuários com responsabilidades distintas e sem a figura de um administrador global. Com base nesse escopo, foram estabelecidos dois perfis de usuários no sistema, correspondentes aos papéis essenciais no contexto de gerenciamento de buffets e eventos. A plataforma não possui administrador global, e cada perfil atua de forma independente dentro de seu respectivo contexto.

Cliente: usuário final da plataforma que utiliza o sistema com o objetivo de realizar reservas em buffets. Esse perfil interage com o sistema para consultar buffets disponíveis, visualizar as comidas e serviços oferecidos por cada estabelecimento e registrar reservas informando dados como data, horário e quantidade de pessoas. O cliente também acompanha o andamento de suas reservas, visualizando seus respectivos estados ao longo do processo.

Buffet: estabelecimento responsável pela oferta de serviços para eventos. Esse perfil utiliza o sistema para organizar suas operações, mantendo o cadastro de comidas e serviços, analisando reservas realizadas por clientes e decidindo sobre sua aprovação ou recusa. Cada buffet administra exclusivamente seus próprios dados e eventos, sem acesso a informações de outros estabelecimentos, garantindo isolamento lógico em um ambiente multi-buffet.

4.2 Requisitos Funcionais

Os requisitos funcionais descrevem as funcionalidades que o sistema deve prover para atender aos objetivos do projeto, expressando os serviços que o sistema deve oferecer e o comportamento esperado em resposta às interações dos usuários (SOMMERVILLE, 2016). Esses requisitos refletem as ações essenciais que clientes e buffets podem realizar na plataforma, considerando o fluxo de reservas, aprovação e gerenciamento de eventos.

A Tabela 4.1 apresenta a relação dos requisitos funcionais, com a identificação do ator principal responsável por cada funcionalidade.

Tabela 4.1: Requisitos funcionais do sistema.

ID	Nome	Descrição	Ator principal
RF01	Cadastro de usuário	Permite que somente clientes criem uma conta no sistema, fornecendo informações básicas e credenciais de acesso.	Cliente
RF02	Autenticação	Permite que usuários autenticados acessem a plataforma por meio de <i>login</i> , utilizando autenticação baseada em <i>token</i> JWT.	Cliente, Buffet
RF03	Gerenciamento de perfil	Permite que o cliente atualize seus dados pessoais, como nome, e-mail, telefone, senha e foto de perfil.	Cliente
RF04	Gerenciamento de comidas	Permite ao buffet cadastrar, listar, atualizar e remover comidas disponibilizadas para eventos, incluindo descrição e imagem.	Buffet
RF05	Gerenciamento de serviços	Permite ao buffet cadastrar, listar, atualizar e remover serviços complementares associados à realização de eventos.	Buffet
RF06	Criação de reserva	Permite ao cliente montar uma reserva selecionando um buffet, informando data, horário, quantidade de pessoas e escolhendo comidas e serviços desejados.	Cliente
RF07	Acompanhamento de reservas	Permite que clientes acompanhem o estado de suas reservas e que buffets visualizem reservas recebidas.	Cliente, Buffet
RF08	Aprovação de reserva	Permite ao buffet aprovar uma reserva realizada por um cliente, informando o valor estimado do evento e definindo a disponibilidade da data para novas aprovações.	Buffet
RF09	Recusa de reserva	Permite ao buffet recusar uma reserva pendente, impedindo sua conversão em evento.	Buffet
RF10	Cancelamento de reserva	Permite ao cliente cancelar uma reserva enquanto esta estiver em estado pendente.	Cliente
RF11	Gerenciamento de eventos	Permite ao buffet visualizar, acompanhar e atualizar o estado de eventos originados a partir de reservas aprovadas.	Buffet
RF12	Conclusão de evento	Permite ao buffet marcar um evento como concluído após sua realização.	Buffet
RF13	Notificações de <i>status</i>	O sistema deve notificar clientes e buffets sempre que houver alteração no estado de uma reserva ou evento.	Sistema
RF14	Gerenciamento de imagens	Permite o upload e a atualização de imagens, sendo fotos de perfil para clientes e imagens associadas às comidas e serviços cadastrados pelos buffets, por meio de serviço externo de armazenamento.	Cliente, Buffet
RF15	Redefinição de senha	Permite ao usuário redefinir sua senha por meio de verificação enviada ao e-mail cadastrado.	Cliente, Buffet

4.3 Requisitos Não Funcionais

Os requisitos não funcionais definem critérios de qualidade e restrições que o sistema deve atender para garantir seu correto funcionamento, segurança e manutenibilidade, incluindo propriedades como desempenho, confiabilidade e usabilidade (SOMMERVILLE, 2016). Esses requisitos não estão associados a funcionalidades específicas, mas impactam diretamente a experiência do usuário e a robustez da plataforma.

A Tabela 4.2 apresenta os requisitos não funcionais identificados para o sistema.

Tabela 4.2: Requisitos não funcionais do sistema.

ID	Nome	Descrição
RNF01	Usabilidade	A aplicação deve apresentar interface intuitiva, responsiva e adequada a dispositivos móveis, permitindo navegação simples e compreensão clara das funcionalidades.
RNF02	Segurança	O sistema deve utilizar comunicação segura (HTTPS), autenticação baseada em JWT e armazenamento seguro de credenciais, garantindo acesso apenas a recursos compatíveis com o perfil do usuário.
RNF03	Escalabilidade	A plataforma deve suportar múltiplos buffets operando simultaneamente, mantendo isolamento lógico de dados em um ambiente multi-buffet, sem necessidade de replicação da aplicação.
RNF04	Desempenho	As operações principais do sistema, como consulta, criação e atualização de reservas, devem apresentar tempo de resposta adequado, mesmo sob acesso simultâneo de múltiplos usuários.
RNF05	Confiabilidade	O sistema deve garantir consistência das informações, evitando conflitos de reservas e assegurando a integridade dos dados relacionados a eventos e disponibilidade de datas.
RNF06	Manutenibilidade	O código-fonte deve seguir padrões de organização e boas práticas de engenharia de software, facilitando manutenção, correções e evolução da plataforma.

4.4 Regras de Negócio

Além dos requisitos identificados, o sistema deve obedecer a um conjunto de regras de negócio que estabelecem e restringem aspectos do funcionamento do domínio, delimitando condições e comportamentos permitidos durante a operação da plataforma (Business Rules Group, 2000). Essas regras são essenciais para garantir a coerência do domínio de eventos e a organização adequada das operações realizadas na plataforma.

1. O cadastro de usuários na plataforma é permitido exclusivamente para clientes. Buffets não realizam cadastro diretamente no sistema.

2. O acesso dos buffets ocorre por meio de credenciais previamente fornecidas no momento da criação do aplicativo do buffet. O buffet pode utilizar o mecanismo de redefinição de senha disponibilizado pelo sistema, da mesma forma que o cliente.
3. Uma reserva é criada pelo cliente a partir da seleção de comidas e serviços oferecidos por um buffet, seguida do preenchimento dos dados da reserva, como data, horário e quantidade de pessoas.
4. O sistema não registra quantidades individuais para itens de comidas e serviços na seleção, tanto para reservas quanto para eventos. A quantidade de pessoas serve como referência geral de dimensionamento do evento.
5. A criação de reservas deve respeitar a disponibilidade do buffet, não sendo permitidas solicitações para datas e horários que conflitem com eventos já agendados no mesmo estabelecimento.
6. O sistema trabalha com estados distintos para reservas e eventos. Reservas podem assumir os estados *pendente*, *aprovada* ou *cancelada*. Eventos são criados exclusivamente a partir da aprovação de uma reserva e podem assumir os estados *agendado*, *cancelado* ou *concluído*.
7. Toda reserva aprovada é automaticamente convertida em um evento, com estado inicial *agendado*.
8. No momento da aprovação de uma reserva, o buffet deve obrigatoriamente informar o valor estimado do evento, sendo este o único valor financeiro tratado pelo sistema.
9. A cada aprovação de reserva, o buffet deve definir se a data do evento permanecerá disponível para novas aprovações ou se será restrita apenas aos eventos já existentes. Caso a data já tenha sido marcada como indisponível em uma aprovação anterior, novas reservas não poderão ser aceitas.
10. O cliente pode editar ou cancelar uma reserva apenas enquanto seu estado for *pendente*. Após a aprovação da reserva pelo buffet, nenhuma ação de edição ou cancelamento fica disponível ao cliente.
11. O buffet pode cancelar reservas enquanto estiverem no estado *pendente*, independentemente de ações do cliente.

12. Após a aprovação da reserva e sua conversão em evento, o cliente pode consultar o evento e, enquanto seu estado for *agendado*, pode editar seus dados e cancelá-lo. Eventos nos estados *cancelado* ou *concluído* não permitem alterações pelo cliente.
13. Para eventos em estado *agendado*, o buffet pode atualizar o valor estimado, cancelar o evento ou reverter um cancelamento previamente realizado. A reversão de cancelamento é uma ação exclusiva do buffet.
14. O buffet pode concluir um evento, alterando seu estado para *concluído*, indicando a finalização da prestação do serviço.
15. O cadastro, a atualização e a remoção de comidas e serviços são permitidos exclusivamente ao buffet. A remoção de um item implica sua inativação no sistema, impedindo novas associações a reservas ou eventos futuros, mesmo que o item permaneça vinculado a registros já existentes.
16. Cada buffet opera de forma independente no sistema, garantindo o isolamento lógico de comidas, serviços, reservas e eventos em relação a outros estabelecimentos.
17. O sistema deve gerar notificações automáticas sempre que ocorrer qualquer alteração no estado de uma reserva ou evento, informando tanto o cliente quanto o buffet.

4.5 Diagramas UML

Com o objetivo de representar de forma visual e padronizada os aspectos funcionais e estruturais do sistema, foram elaborados diagramas baseados na UML (Unified Modeling Language) , uma linguagem de modelagem amplamente utilizada para especificação, visualização, construção e documentação de sistemas orientados a objetos (BOOCH et al., 2005). Esses diagramas auxiliam na compreensão do domínio da aplicação, descrevendo as interações entre usuários e sistema, a organização das entidades principais e os fluxos de execução das funcionalidades centrais.

Os diagramas apresentados a seguir compreendem o diagrama de casos de uso, o diagrama de classes e os diagramas de sequência, todos alinhados aos requisitos funcionais, não funcionais e às regras de negócio previamente definidas.

4.5.1 Diagrama de Casos de Uso

O diagrama de casos de uso apresenta uma visão funcional do sistema, evidenciando as funcionalidades disponibilizadas e os atores responsáveis por sua execução (FOWLER, 2004). Esse tipo de diagrama permite representar as interações externas com o sistema, destacando os papéis envolvidos e os serviços oferecidos pela aplicação.

Foram identificados dois atores principais: **Cliente** e **Buffet**. Cada ator interage com o sistema de acordo com seu perfil de acesso e com as regras de negócio previamente estabelecidas.

A Figura 4.1 apresenta o diagrama de casos de uso do sistema, representando graficamente as funcionalidades disponíveis.



Figura 4.1: Diagrama de casos de uso do sistema.

Casos de uso identificados

A Tabela 4.3 apresenta a relação dos casos de uso identificados no sistema, bem como os respectivos atores responsáveis por sua execução.

Tabela 4.3: Casos de uso do sistema.

ID	Caso de Uso	Ator
UC01	Cadastrar usuário	Cliente
UC02	Autenticar usuário	Cliente, Buffet
UC03	Gerenciar perfil	Cliente
UC04	Consultar comidas e serviços	Cliente
UC05	Criar reserva	Cliente
UC06	Acompanhar reservas e eventos	Cliente, Buffet
UC07	Editar reserva (pendente)	Cliente
UC08	Cancelar reserva (pendente)	Cliente, Buffet
UC17	Editar evento (agendado)	Cliente
UC18	Cancelar evento (agendado)	Cliente, Buffet
UC09	Redefinir senha	Cliente, Buffet
UC10	Gerenciar reservas	Buffet
UC11	Aprovar reserva	Buffet
UC12	Informar valor estimado	Buffet
UC13	Definir disponibilidade da data	Buffet
UC14	Gerenciar comidas	Buffet
UC15	Gerenciar serviços	Buffet
UC16	Gerenciar eventos	Buffet
UC19	Reverter cancelamento	Buffet
UC20	Editar valor estimado do evento	Buffet
UC21	Concluir evento	Buffet

Descrição dos Casos de Uso

Antes de apresentar os casos de uso do sistema, é importante contextualizar que eles representam as interações entre os usuários (atores) e o sistema, descrevendo o comportamento esperado do sistema em diferentes situações. Cada caso de uso é descrito de forma estruturada, utilizando um modelo tabular que contempla os principais elementos, como atores envolvidos, pré e pós-condições, cenário principal, cenários de exceção, ações e restrições, de modo a facilitar o entendimento do funcionamento do sistema. A seguir, são apresentados os casos de uso considerados principais, que representam as funcionalidades centrais da aplicação.

UC02 – Autenticar usuário A Tabela 4.4 apresenta o caso de uso UC02, que descreve o processo de autenticação de usuários no sistema, permitindo o acesso às funcionalidades conforme o perfil do usuário autenticado.

Tabela 4.4: Descrição do Caso de Uso UC02 – Autenticar usuário

Nome do caso de uso	UC02 – Autenticar usuário
Ator principal	Cliente, Buffet
Atores secundários	–
Resumo	Permite que o usuário acesse o sistema por meio da validação de suas credenciais, habilitando funcionalidades conforme seu perfil.
Pré-condições	Usuário deve estar cadastrado (cliente) ou possuir credenciais previamente fornecidas (buffet).
Pós-condições	Usuário autenticado e com acesso às funcionalidades do sistema.
Cenário Principal	
Ações do ator	Ações do sistema
1) Informar e-mail e senha	2) Validar credenciais fornecidas
3) Confirmar autenticação	4) Conceder acesso ou exibir mensagem de erro
Cenários de Exceção	
E1 – Usuário não encontrado: O sistema não localiza o e-mail ou telefone informado no buffet selecionado. Uma mensagem de erro é exibida e o fluxo retorna ao passo 1.	
E2 – Senha incorreta: A senha informada não corresponde à senha cadastrada. O sistema exibe mensagem de credenciais inválidas e o fluxo retorna ao passo 1.	
E3 – Buffet incorreto: O usuário existe, porém em outro buffet. O sistema exibe mensagem de erro e o fluxo retorna ao passo 1.	
Restrições/Validações	
- As credenciais devem ser válidas e corresponder a um usuário ativo. - Usuários com credenciais inválidas não obtêm acesso ao sistema.	

UC05 – Criar reserva A Tabela 4.5 apresenta o caso de uso UC05, que descreve o processo de criação de uma reserva por parte do cliente, contemplando a seleção de buffet, comidas, serviços e o preenchimento dos dados do evento.

Tabela 4.5: Descrição do Caso de Uso UC05 – Criar reserva

Nome do caso de uso	UC05 – Criar reserva
Ator principal	Cliente
Atores secundários	Buffet
Resumo	Permite ao cliente criar uma reserva selecionando um buffet, comidas e serviços, informando data, horário e quantidade de pessoas.
Pré-condições	Cliente deve estar autenticado no sistema.
Pós-condições	Reserva registrada no sistema com estado <i>pendente</i> .
Cenário Principal	
Ações do ator	Ações do sistema
1) Selecionar buffet	2) Exibir comidas e serviços disponíveis
3) Selecionar comidas e serviços	4) Registrar seleção dos itens
5) Informar data, horário e quantidade	6) Validar disponibilidade e dados da reserva
7) Confirmar criação da reserva	8) Registrar reserva como pendente e notificar o buffet
Cenários de Exceção	
E1 – Buffet não encontrado: O sistema exibe mensagem de erro e o fluxo é encerrado.	
E2 – Dados obrigatórios não informados: O cliente não preencheu campos obrigatórios como data, horário, quantidade de pessoas ou endereço. O sistema exibe mensagem indicando os campos faltantes e o fluxo retorna ao passo 5.	
E3 – Data selecionada anterior à data atual: Apresentada mensagem de erro referente à data.	
Restrições/Validações	
- A reserva não pode conflitar com eventos já agendados do buffet. - Todos os dados obrigatórios devem ser informados. - Reservas para datas indisponíveis não são permitidas.	

UC11 – Aprovar reserva A Tabela 4.6 apresenta o caso de uso UC11, que descreve o processo de aprovação de uma reserva pendente pelo buffet, incluindo a definição do valor estimado do evento e da disponibilidade da data.

Tabela 4.6: Descrição do Caso de Uso UC11 – Aprovar reserva

Nome do caso de uso	UC11 – Aprovar reserva		
Ator principal	Buffet		
Atores secundários	Cliente		
Resumo	Permite ao buffet aprovar uma reserva pendente, informando o valor estimado do evento e definindo a disponibilidade da data.		
Pré-condições	Reserva deve estar em estado <i>pendente</i> .		
Pós-condições	Reserva aprovada e evento criado com estado <i>agendado</i> .		
Cenário Principal			
Ações do ator	Ações do sistema		
1) Selecionar reserva pendente	2) Exibir detalhes da reserva		
3) Informar valor estimado	4) Registrar valor estimado		
5) Definir disponibilidade da data	6) Atualizar disponibilidade do buffet		
7) Confirmar aprovação	8) Criar evento e notificar o cliente		
Cenários de Exceção			
E1 – Reserva não encontrada: A reserva selecionada não existe ou foi removida do sistema. O sistema exibe mensagem de erro e o fluxo é encerrado.			
E2 – Reserva não está pendente: A reserva já foi aprovada, cancelada ou possui outro estado diferente de <i>pendente</i> . O sistema exibe mensagem informando o estado atual e o fluxo é encerrado.			
E3 – Valor não informado: O buffet não informou o valor estimado do evento. O sistema exibe mensagem indicando a obrigatoriedade do campo e o fluxo retorna ao passo 3.			
Restrições/Validações			
- Apenas reservas em estado <i>pendente</i> podem ser aprovadas.			
- É obrigatório informar o valor estimado do evento.			
- O buffet não pode editar os dados da reserva, apenas aprová-la ou cancelá-la.			

4.5.2 Diagrama de Classes

O diagrama de classes representa a estrutura estática do sistema, modelando classes, atributos, operações e seus relacionamentos, incluindo multiplicidades (BOOCH et al., 2005). Trata-se de um dos principais artefatos da modelagem orientada a objetos, por permitir a organização estrutural das entidades do domínio.

A Figura 4.2 apresenta o diagrama de classes do sistema, evidenciando as entidades centrais, seus atributos e associações. Esse modelo apoia a implementação ao refletir as regras de negócio definidas, destacando a separação conceitual entre **reserva** e **evento**.

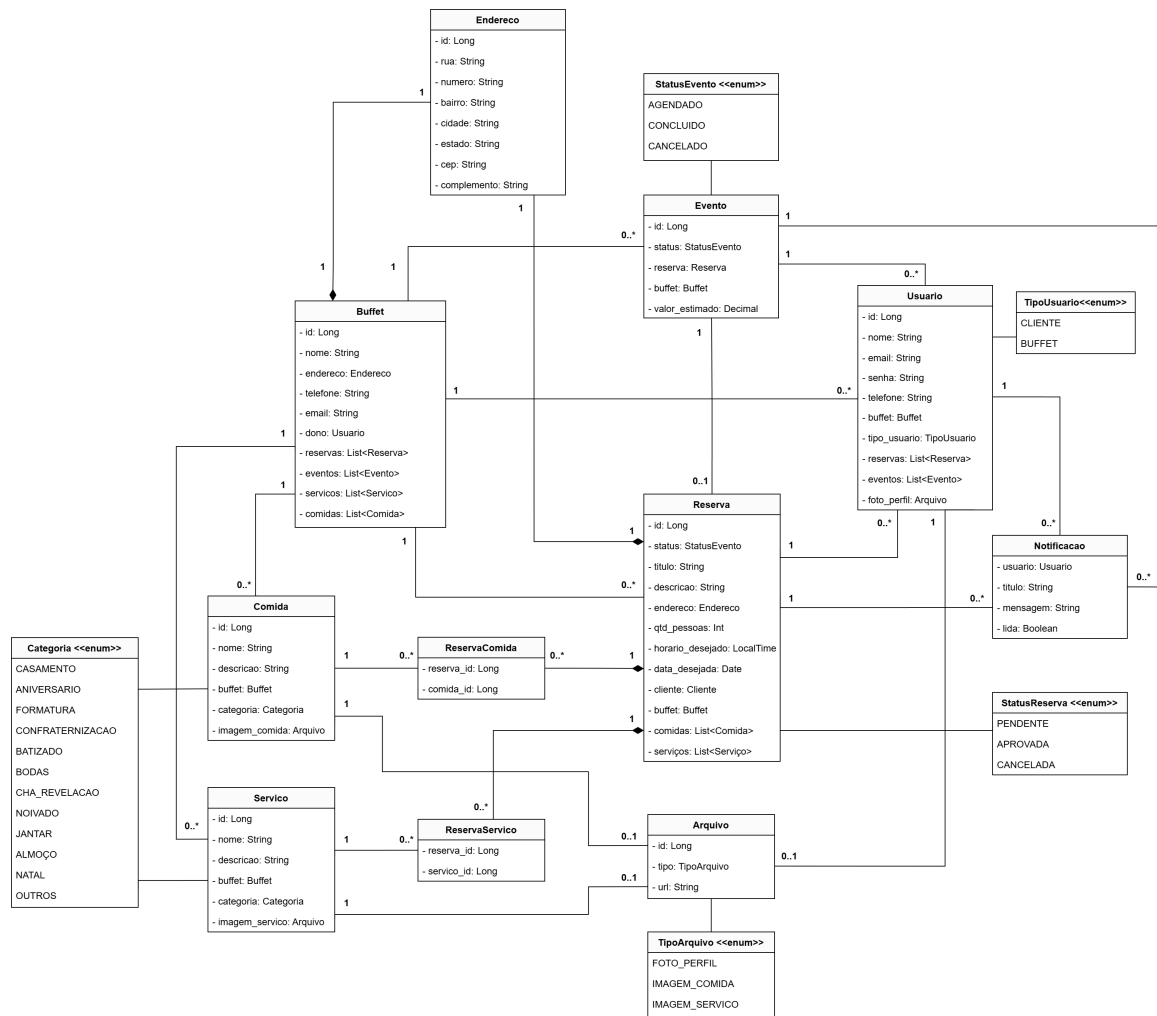


Figura 4.2: Diagrama de classes do sistema.

As principais classes identificadas no diagrama são descritas a seguir:

- **Usuário:** representa qualquer pessoa cadastrada na plataforma e pode assumir o papel de cliente ou de responsável por um buffet. A classe armazena dados como `id`, `nome`, e-mail, senha e telefone. Cada usuário está vinculado a um `buffetId`, que identifica o buffet ao qual pertence, e possui um campo `tipoUsuario` definido pelo enumerado `TipoUsuario` (*CLIENTE* ou *BUFFET*). O usuário *CLIENTE* mantém associações com suas reservas e eventos e pode ter uma foto de perfil opcional, armazenada como um recurso da classe `Arquivo`.
- **Buffet:** representa o estabelecimento responsável pela prestação de serviços em eventos. Contém dados cadastrais (identificador, nome, endereço, telefone e e-mail) e relaciona-se com as demais entidades. O atributo `donoId` referencia o usuário que é o proprietário do buffet. Um buffet pode possuir vários usuários vinculados (funcionários ou um

administrador), bem como diversas reservas, eventos, comidas, serviços e arquivos de divulgação.

- **Reserva:** registra a solicitação feita por um cliente. A reserva contém data e horário desejados, quantidade de pessoas, descrição e endereço do evento. Cada reserva está associada a um usuário (cliente) e a um buffet específicos. Seu estado é controlado pelo enumerado `StatusReserva`, que assume os valores *pendente*, *aprovada* ou *cancelada*. As seleções de comidas e serviços são armazenadas por meio das listas de `Comida` e `Serviço` ou pelas entidades associativas `ReservaComida` e `ReservaServico`. Uma reserva aprovada pode originar no máximo um evento (multiplicidade 0..1).
- **Evento:** representa a concretização de uma reserva aprovada. Cada evento está vinculado à reserva que lhe deu origem, ao buffet responsável e ao cliente. O estado do evento é definido pelo enumerado `StatusEvento` (*agendado*, *concluído* ou *cancelado*). A classe também armazena o `valorEstimado`, que corresponde ao único dado financeiro tratado na plataforma.
- **Comida:** representa os itens de alimentação oferecidos pelo buffet. Possui atributos como identificador, nome e descrição, além da categoria (`Categoria`) que classifica o tipo de alimento (casamento, aniversário, formatura, confraternização, batizado, bodas, chá-revelação, noivado, jantar, almoço, Natal ou outros). Cada comida pertence a um único buffet e pode ser incluída em várias reservas por meio da entidade `ReservaComida`. Opcionalmente, uma comida possui uma imagem associada (`Arquivo`) do tipo `IMAGEMCOMIDA`.
- **Serviço:** similar a comida, representa os serviços adicionais disponibilizados pelo buffet (decoração, música, etc.). Possui identificador, nome, descrição e categoria (`Categoria`). Cada serviço pertence a um único buffet e pode ser selecionado por diferentes reservas através da entidade `ReservaServico`. Pode possuir uma imagem (`Arquivo`) do tipo `IMAGESERVICO`.
- **ReservaComida e ReservaServico:** classes associativas que materializam as relações muitos-para-muitos entre `Reserva` e `Comida` e entre `Reserva` e `Serviço`, respectivamente. Registram quais itens foram selecionados em cada reserva.
- **Endereço:** entidade reutilizável para armazenar dados de localização, com campos como

rua, número, bairro, cidade, estado, CEP e complemento. Um endereço é utilizado tanto pelo buffet quanto pela reserva para indicar o local do evento.

- **Arquivo**: representa recursos de mídia armazenados externamente (imagens). Cada arquivo possui um identificador, uma URL e um tipo definido pelo enumerado *TipoArquivo* (*FOTOPERFIL*, *IMAGEMCOMIDA* ou *IMAGEMSERVICO*). Arquivos são utilizados como fotos de perfil de usuários, imagens de comidas e serviços, ou como material de divulgação do buffet.
- **Notificação**: classe incorporada ao modelo para registrar mensagens enviadas aos usuários, como alterações no *status* de reservas e eventos. Cada notificação armazena o usuário destinatário, título, mensagem e um indicador de leitura (*lida*).

Os tipos enumerados presentes no diagrama — *StatusReserva*, *StatusEvento*, *Categoria*, *TipoArquivo* e *TipoUsuario* — padronizam estados e classificações, fortalecendo a consistência do modelo e garantindo que as transições de estado e as categorias sigam as regras de negócio estabelecidas para a plataforma.

4.5.3 Diagramas de Sequência

Os diagramas de sequência representam, sob uma perspectiva temporal, a troca de mensagens entre atores e componentes do sistema durante a execução de um caso de uso específico (FOWLER, 2004). Esses diagramas permitem visualizar a ordem das interações, os fluxos principais e alternativos e a colaboração entre camadas da arquitetura, como interface, controladores, serviços e repositórios, evidenciando o comportamento dinâmico da aplicação.

As Figuras 4.3, 4.4 e 4.5 apresentam, respectivamente, os diagramas de sequência referentes aos casos de uso: autenticação do usuário (UC01), criação de reserva (UC05) e aprovação de reserva com conversão em evento (UC06). A seguir, descrevem-se detalhadamente os fluxos representados em cada diagrama.

Diagrama de Sequência – Autenticar usuário

A Figura 4.3 ilustra o fluxo de autenticação no sistema, aplicável tanto para o perfil **Cliente** quanto para o perfil **Buffet**. O diagrama evidencia a interação temporal entre os participantes: *Cliente/Buffer*, *App (Interface)*, *AuthController*, *AuthService*, *UserRepository* e o *Banco de Dados*.

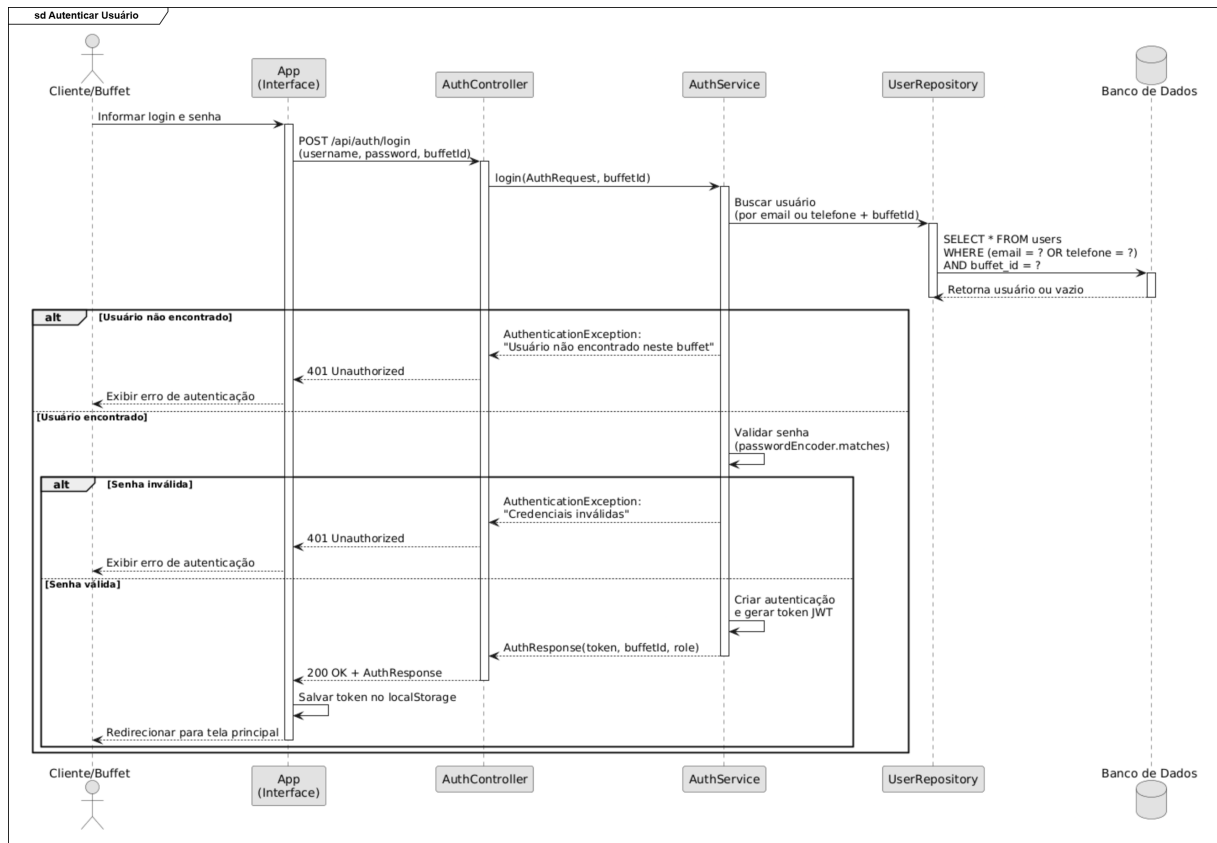


Figura 4.3: Diagrama de sequência do caso de uso UC01 – Autenticar usuário.

O fluxo se inicia quando o usuário informa suas credenciais na interface do aplicativo, enviando *login* e *senha*. Em seguida, o *App* realiza a requisição ao *AuthController* por meio do endpoint `POST /api/auth/login`, enviando os dados de autenticação e o identificador do buffet (*buffetId*). Esse parâmetro é necessário para garantir que a autenticação ocorra no contexto correto em um ambiente *multi-buffet*.

Ao receber a requisição, o *AuthController* delega o processamento ao *AuthService*, responsável pelas validações e pela emissão do *token*. O serviço, por sua vez, consulta o *UserRepository* para buscar o usuário correspondente ao *login,e-mail* ou *telefone* combinados com o *buffetId*. Essa consulta é realizada no banco de dados.

O diagrama contempla dois caminhos alternativos (*alt*):

- **Usuário não encontrado:** caso a consulta não retorne registro, o *AuthService* lança uma exceção de autenticação, que resulta em resposta 401 *Unauthorized*. O *App* exibe mensagem de erro ao usuário.
- **Usuário encontrado:** se o usuário existir, o *AuthService* realiza a validação da senha utilizando *passwordEncoder.matches*. Se a senha for inválida, ocorre novamente retorno 401 *Unauthorized*. Caso a senha seja válida, o serviço cria o contexto de autenticação e gera um *token* JWT, retornando um *AuthResponse* contendo o *token*, o *buffetId* e o perfil (*role*).

Por fim, o *AuthController* retorna 200 OK ao *App*. O aplicativo armazena o *token* localmente (ex.: *localStorage*) e redireciona o usuário para a tela principal compatível com seu perfil, concluindo o processo de autenticação.

Diagrama de Sequência – Criar reserva

A Figura 4.4 apresenta o diagrama de sequência referente ao caso de uso de criação de reserva, realizado pelo **Cliente**. O diagrama evidencia a interação temporal entre: *Cliente*, *App (Interface)*, *ReservaClienteController*, *ReservaService*, repositórios de domínio (*UserRepository*, *BuffetRepository*, *ServicoRepository*, *ComidaRepository*, *ReservaRepository*), o *NotificacaoService* e o *Banco de Dados*.

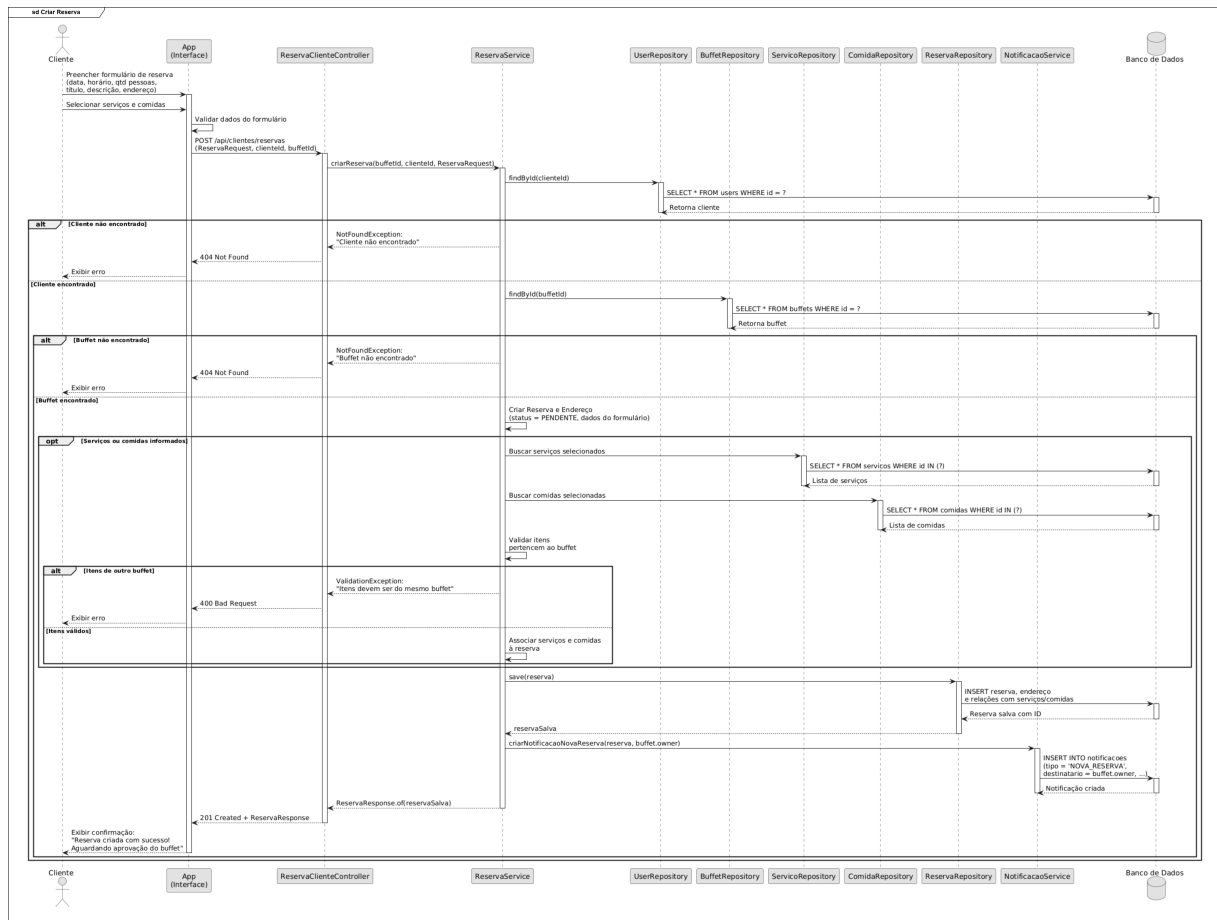


Figura 4.4: Diagrama de sequência do caso de uso UC05 – Criar reserva.

O fluxo se inicia quando o cliente preenche o formulário de reserva no aplicativo, informando data, horário, quantidade de pessoas, título, descrição e endereço. Além disso, o cliente seleciona comidas e serviços desejados. Antes do envio, o *App* realiza validações locais de preenchimento e consistência dos dados do formulário.

Em seguida, o *App* envia a requisição ao *ReservaClienteController* via `POST /api/clientes/reservas`, encaminhando *ReservaRequest*, *clienteId* e *buffetId*. O controlador delega a lógica ao *ReservaService*, responsável por validar as entidades envolvidas e persistir o registro da reserva.

O serviço executa validações sequenciais:

- **Validação do cliente:** consulta o *UserRepository* (`findById`) para garantir que o cliente existe. Caso não exista, o fluxo retorna `404 Not Found`.
- **Validação do buffet:** consulta o *BuffetRepository* (`findById`) para garantir que o buffet selecionado é válido. Caso não exista, retorna `404 Not Found`.

Com cliente e buffet válidos, o *ReservaService* cria a entidade *Reserva* e o *Endereço* associado, definindo o *status* inicial como *PENDENTE* e armazenando os dados do formulário.

O diagrama inclui um bloco opcional (*opt*) para o caso de haver itens selecionados. Nessa etapa, o serviço busca os serviços no *ServicoRepository* e as comidas no *ComidaRepository*. Após obter as listas, o serviço valida se todos os itens pertencem ao mesmo buffet selecionado. Se houver itens de outro buffet, o serviço retorna 400 Bad Request. Caso os itens sejam válidos, o serviço cria as associações necessárias entre a reserva e os itens escolhidos.

Após as validações, o *ReservaRepository* persiste a reserva no banco de dados, incluindo o endereço e os relacionamentos com comidas e serviços. Com a reserva salva e seu identificador gerado, o serviço aciona o *NotificacaoService* para registrar uma notificação de nova reserva destinada ao responsável do buffet. Por fim, o *ReservaService* retorna um *ReservaResponse*, e o controlador responde com 201 Created. O *App* exibe a confirmação ao cliente, indicando que a reserva foi registrada e permanecerá aguardando avaliação do buffet.

Diagrama de Sequência – Aprovar reserva

A Figura 4.5 apresenta o diagrama de sequência referente ao caso de uso de aprovação de reserva por parte do **Dono do Buffet**, etapa que concretiza a principal regra do domínio: uma reserva aprovada é convertida em evento. O diagrama evidencia a interação temporal entre: *Dono do Buffet*, *App (Interface)*, *ReservaBuffetController*, *ReservaService*, *ReservaRepository*, *NotificacaoService* e o *Banco de Dados*.

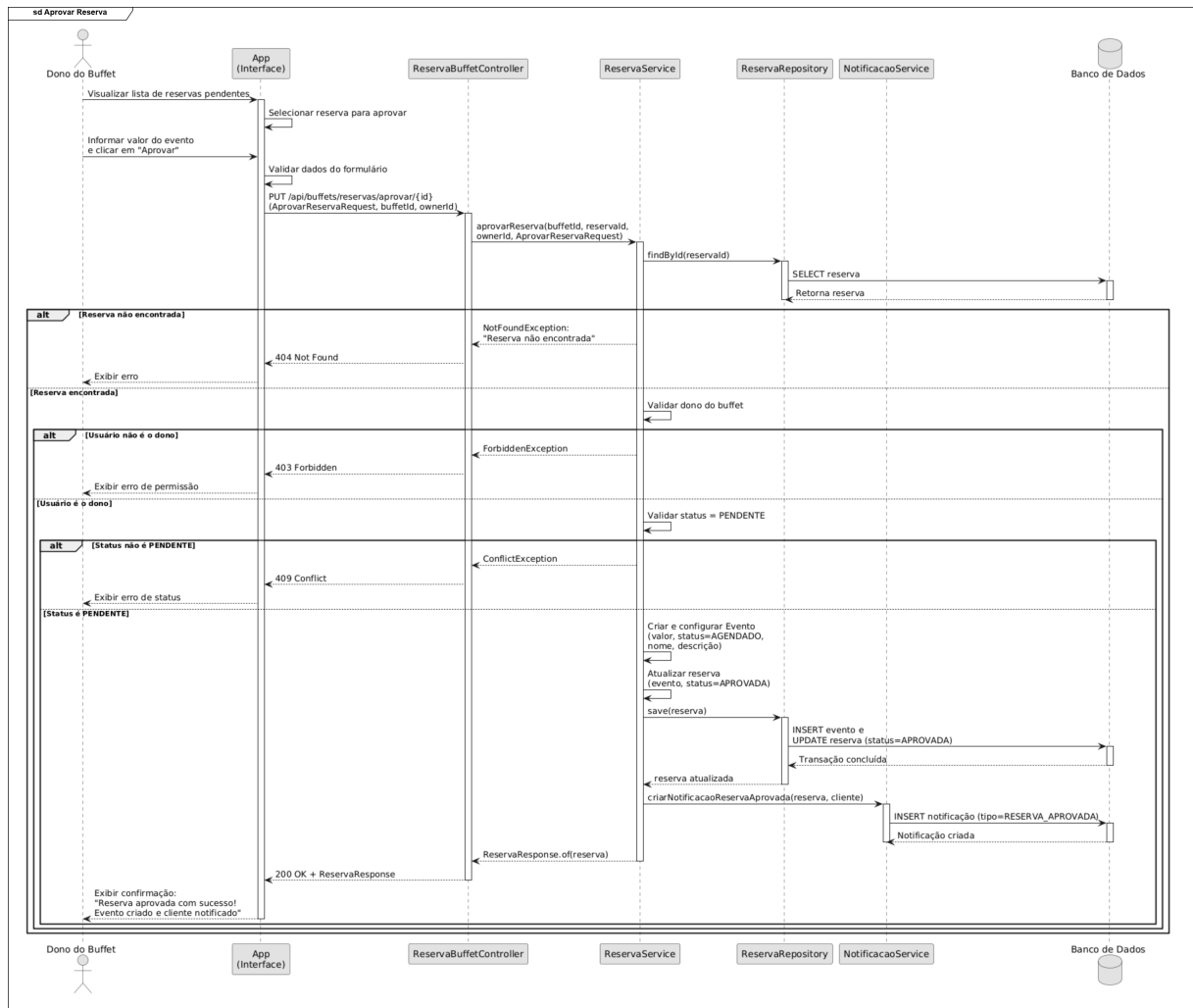


Figura 4.5: Diagrama de sequência do caso de uso UC06 – Aprovar reserva.

O fluxo se inicia quando o dono do buffet acessa a lista de reservas pendentes no aplicativo e seleciona uma reserva para aprovação. Em seguida, informa o valor estimado do evento e confirma a ação. O *App* valida o formulário e envia a requisição ao *ReservaBuffetController* por meio do endpoint `PUT /api/buffets/reservas/aprovar/{id}`, encaminhando *AprovarReservaRequest*, *buffetId* e *ownerId*.

Ao receber a solicitação, o controlador encaminha o processamento ao *ReservaService*, que inicia a busca da reserva no *ReservaRepository* (`findById`). Caso a reserva não exista, retorna `404 Not Found`. Caso exista, o serviço aplica validações essenciais representadas por blocos alternativos (*alt*):

- **Validação de permissão:** o serviço confirma se o usuário autenticado (*ownerId*) é o dono do buffet responsável pela reserva. Se não for, retorna `403 Forbidden`.

- **Validação de *status*:** o serviço verifica se o *status* da reserva é *PENDENTE*. Se a reserva não estiver pendente, retorna 409 *Conflict*.

Com as validações atendidas, o serviço cria o *Evento* a partir dos dados da reserva, definindo o *status* do evento como *AGENDADO* e registrando o valor informado. Em seguida, o serviço atualiza a reserva, alterando seu *status* para *APROVADA* e vinculando o evento criado. A persistência ocorre por meio do *ReservaRepository*, que executa a transação de inserção do evento e atualização da reserva.

Após a conclusão da transação no banco, o serviço aciona o *NotificacaoService* para registrar uma notificação de reserva aprovada destinada ao cliente. Por fim, o *ReservaService* retorna *ReservaResponse* e o controlador responde com 200 OK. O *App* exibe a confirmação ao dono do buffet, indicando que a reserva foi aprovada, o evento foi criado e o cliente foi notificado.

Neste capítulo, foram apresentados os elementos que estruturam a concepção do sistema, incluindo a definição dos perfis de usuários (cliente e buffet), a especificação dos requisitos funcionais e não funcionais, as regras de negócio que delimitam o comportamento da plataforma e os artefatos de modelagem UML, diagramas de casos de uso, de classes e de sequência. Essa análise forneceu a base conceitual necessária para orientar a etapa de codificação. Dessa forma, o próximo capítulo detalha a implementação do *back-end* por meio de uma *API RESTful* desenvolvida em Java com Spring Boot, abordando a configuração inicial, a estrutura de pacotes, as entidades, os serviços e o projeto do banco de dados.

CAPÍTULO 5

IMPLEMENTAÇÃO DO *BACK-END* (API RESTFUL)

Com os requisitos definidos no Capítulo 4, deu-se início à construção do sistema pela camada de *back-end*, encarregada do processamento das regras de negócio, do armazenamento das informações e da disponibilização dos recursos acessados pela interface móvel. A implementação utilizou **Java 21** e **Spring Boot 3** como base tecnológica, com autenticação por *tokens* **JWT** gerenciada pelo **Spring Security**, mapeamento objeto-relacional por meio do **JPA/Hibernate**, banco de dados **PostgreSQL** e controle de versões do esquema via **Flyway**. O conjunto dessas tecnologias viabiliza uma arquitetura organizada e de fácil evolução, além de suportar o modelo *multi-tenant* adotado, que garante o isolamento lógico entre os dados dos diferentes buffets cadastrados na plataforma.

5.1 Configuração Inicial

A primeira etapa do desenvolvimento do sistema foi a criação do projeto Spring Boot, utilizando a ferramenta Spring Initializr. Para garantir a integração de todos os componentes necessários, foram adicionadas as seguintes dependências ao arquivo `pom.xml`:

Tabela 5.1: Dependências utilizadas no projeto e suas descrições.

Dependência	Descrição
spring-boot-starter-data-jpa	Integração com JPA/Hibernate para persistência de dados no banco PostgreSQL.
spring-boot-starter-validation	Validação de dados de entrada com Bean Validation (Jakarta Validation).
spring-boot-starter-web	Suporte para construção de APIs REST com Spring MVC.
spring-boot-starter-security	Autenticação e autorização de usuários na API.
spring-boot-starter-test	Suporte para testes unitários e de integração.
springdoc-openapi-starter-webmvc-ui	Documentação automática da API com Swagger UI.
postgresql	Driver JDBC para conexão com banco de dados PostgreSQL.
h2	Banco de dados em memória para testes e desenvolvimento local.
lombok	Redução de boilerplate com anotações para getters, setters e construtores.
java-jwt	Geração e validação de tokens JWT para autenticação stateless.
cloudinary-http44	Upload e gerenciamento de imagens na nuvem (fotos de comidas, serviços e perfil).
jasperreports	Geração de relatórios em PDF.
flyway-core	Controle de versão e migração do esquema do banco de dados.
flyway-database-postgresql	Suporte específico do Flyway para PostgreSQL.
spring-security-test	Utilitários para testes de segurança.
spring-boot-starter-mail	Utilizado para envio de emails.

5.2 Estrutura de Pacotes do *Back-end*

A Figura 5.1 apresenta a estrutura de pacotes do *back-end*. A figura (a) mostra os pacotes da aplicação; a figura (b) apresenta os subpacotes de *controller* e *model*; a figura (c) detalha o subpacote de *dto*; e a figura (d) detalha o subpacote de *repository* e *service*.

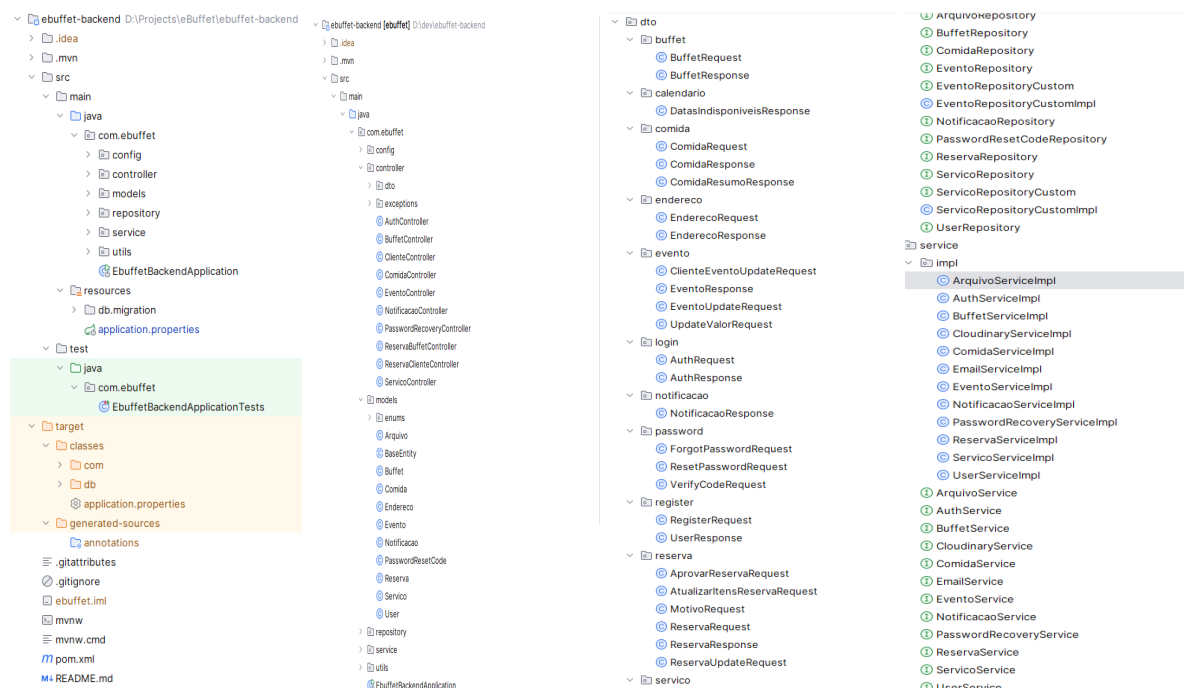


Figura 5.1: Estrutura de pacotes do Back-end

5.3 Endpoints da API

5.3.1 Endpoints de Autenticação

A autenticação é um mecanismo essencial para assegurar que apenas usuários devidamente autorizados tenham acesso aos recursos do sistema. O processo de autenticação no sistema é baseado em tokens JWT (JSON Web Token), que são gerados após a validação das credenciais do usuário e utilizados para autorizar requisições subsequentes. Os endpoints de autenticação permitem o registro de novos usuários, tanto clientes quanto proprietários de buffet, além de fornecer acesso às informações do usuário autenticado. Os principais endpoints relacionados ao processo de autenticação estão descritos na Tabela 5.2.

Tabela 5.2: Endpoints de autenticação da API.

Método	Endpoint	Descrição
POST	/api/auth/login	Realiza login e retorna token JWT
POST	/api/auth/register	Cadastra novo usuário cliente
POST	/api/auth/register/buffet	Cadastra novo usuário dono de buffet
GET	/api/auth/me	Retorna dados do usuário autenticado

5.3.2 Endpoints de Clientes

Os endpoints de clientes permitem o gerenciamento do perfil dos usuários que utilizam a plataforma para contratar serviços de buffet. Através destes recursos, o cliente autenticado pode atualizar suas informações pessoais, incluindo nome, endereço de email, número de telefone e foto de perfil. A atualização do perfil é fundamental para manter os dados de contato atualizados, garantindo que o buffet possa se comunicar adequadamente com o cliente sobre suas reservas e eventos. O endpoint disponível para gerenciamento do perfil do cliente está descrito na Tabela 5.3.

Tabela 5.3: Endpoint de gerenciamento do perfil do cliente.

Método	Endpoint	Descrição
PUT	/api/clientes/me	Atualiza perfil do cliente autenticado (nome, email, telefone, foto)

5.3.3 Endpoints de Buffets

Os endpoints de buffets constituem o núcleo do gerenciamento dos estabelecimentos cadastrados na plataforma. Estes recursos permitem que proprietários criem, visualizem, atualizem e removam seus buffets, incluindo informações como nome comercial, dados de contato e endereço de localização. Cada buffet funciona como uma entidade central que agrega comidas, serviços, reservas e eventos. Os endpoints disponíveis para o gerenciamento de buffets estão descritos na Tabela 5.4.

Tabela 5.4: Endpoints de gerenciamento de buffets.

Método	Endpoint	Descrição
POST	/api/buffets	Cria um novo buffet
GET	/api/buffets/{id}	Busca um buffet pelo ID
GET	/api/buffets/me	Lista buffets do proprietário autenticado (paginado)
PUT	/api/buffets/{id}	Atualiza dados de um buffet
DELETE	/api/buffets/{id}	Remove um buffet

5.3.4 Endpoints de Comidas

Os endpoints de comidas permitem o gerenciamento completo do cardápio oferecido por cada buffet. Através destes recursos, o proprietário pode cadastrar novos itens alimentícios, incluindo informações como nome, descrição, categoria do evento e imagem ilustrativa. O

sistema possibilita a organização do cardápio por categorias de eventos (casamento, aniversário, formatura, entre outros), facilitando a seleção pelos clientes durante o processo de reserva. A listagem de comidas suporta filtros por categoria e status, permitindo uma visualização organizada do catálogo. Os endpoints disponíveis para o gerenciamento de comidas estão descritos na Tabela 5.5.

Tabela 5.5: Endpoints de gerenciamento de comidas.

Método	Endpoint	Descrição
POST	/api/comidas	Cadastra nova comida no cardápio do buffet
GET	/api/comidas/{id}	Busca uma comida pelo ID
GET	/api/comidas	Lista comidas do buffet com filtros (categoria, status)
PUT	/api/comidas/{id}	Atualiza dados de uma comida
DELETE	/api/comidas/{id}	Remove uma comida (soft ou hard delete)

5.3.5 Endpoints de Serviços

Os endpoints de serviços possibilitam o gerenciamento do portfólio de serviços adicionais oferecidos pelos buffets, como decoração, música ao vivo, fotografia, filmagem e outras amenidades. Estes serviços complementam a experiência do evento e podem ser selecionados pelos clientes durante a criação de uma reserva. Assim como as comidas, os serviços são categorizados por tipo de evento, permitindo sugestões contextualizadas. O sistema oferece funcionalidades de busca textual e filtros avançados para facilitar a navegação pelo catálogo de serviços disponíveis. Os endpoints disponíveis para o gerenciamento de serviços estão descritos na Tabela 5.6.

Tabela 5.6: Endpoints de gerenciamento de serviços.

Método	Endpoint	Descrição
POST	/api/servicos	Cadastra novo serviço no portfólio do buffet
GET	/api/servicos/{id}	Busca um serviço pelo ID
GET	/api/servicos	Lista serviços do buffet com filtros (categoria, status, busca)
PUT	/api/servicos/{id}	Atualiza dados de um serviço
DELETE	/api/servicos/{id}	Remove um serviço (soft ou hard delete)

5.3.6 Endpoints de Eventos

Os endpoints de eventos gerenciam o ciclo de vida completo dos eventos confirmados no sistema. Um evento é criado automaticamente quando uma reserva é aprovada pelo buffet, representando o compromisso firmado entre as partes. Estes recursos permitem visualizar a

agenda de eventos, atualizar informações como data, horário e valor, além de controlar o status do evento através de ações como conclusão e cancelamento. O sistema também oferece consulta de datas indisponíveis, auxiliando clientes na escolha de datas disponíveis para novos eventos. Os endpoints disponíveis para o gerenciamento de eventos estão descritos na Tabela 5.7.

Tabela 5.7: Endpoints de gerenciamento de eventos.

Método	Endpoint	Descrição
GET	/api/eventos/{id}	Busca um evento pelo ID
GET	/api/eventos	Lista eventos do buffet com filtros (status, período)
GET	/api/eventos/me	Lista eventos do cliente autenticado
GET	/api/eventos/datas-indisponiveis	Retorna datas com eventos confirmados no período
PUT	/api/eventos/{id}/valor	Atualiza valor cobrado pelo evento (até 3 dias antes)
PUT	/api/eventos/{id}/concluir	Marca evento como concluído
PUT	/api/eventos/{id}/cancelar	Cancela um evento (proprietário do buffet)
PUT	/api/eventos/{id}/reverter-cancelamento	Reverte o cancelamento de um evento
PUT	/api/eventos/{id}/cliente	Atualiza evento pelo cliente (até 5 dias antes)
PUT	/api/eventos/{id}/cliente/cancelar	Cancela evento pelo cliente
DELETE	/api/eventos/{id}	Remove um evento (soft ou hard delete)

5.3.7 Endpoints de Reservas (Buffet)

Os endpoints de reservas para o buffet permitem que o proprietário do estabelecimento gerencie as solicitações recebidas dos clientes. Através destes recursos, o buffet pode visualizar todas as reservas pendentes, aprovar ou recusar solicitações, e gerenciar os itens (comidas e serviços) associados a cada reserva. O fluxo de aprovação é fundamental para o funcionamento do sistema, pois a aprovação de uma reserva gera automaticamente um evento na agenda do buffet. Estas funcionalidades garantem controle total sobre a demanda e capacidade do estabelecimento. Os endpoints disponíveis para o gerenciamento de reservas pelo buffet estão descritos na Tabela 5.8.

Tabela 5.8: Endpoints de gerenciamento de reservas pelo buffet.

Método	Endpoint	Descrição
GET	/api/buffets/reservas	Lista reservas do buffet (paginado)
PUT	/api/buffets/reservas/aprovar/{id}	Aprova uma reserva pendente
PUT	/api/buffets/reservas/recusar/{id}	Recusa uma reserva pendente
PUT	/api/buffets/reservas/{id}/itens	Atualiza comidas e serviços da reserva
GET	/api/buffets/reservas/{id}/servicos	Lista serviços da reserva
POST	/api/buffets/reservas/{id}/servicos/{servicoId}	Adiciona serviço à reserva
DELETE	/api/buffets/reservas/{id}/servicos/{servicoId}	Remove serviço da reserva

5.3.8 Endpoints de Reservas (Cliente)

Os endpoints de reservas para o cliente permitem que usuários da plataforma solicitem a contratação de serviços de buffet para seus eventos. O cliente pode criar novas reservas informando data desejada, quantidade de convidados, local do evento e selecionando comidas e serviços do catálogo do buffet. O sistema implementa regras de negócio específicas, como a possibilidade de edição apenas de reservas pendentes e restrições de cancelamento para reservas já aprovadas com menos de 7 dias para o evento. Estes recursos oferecem autonomia ao cliente no gerenciamento de suas solicitações. Os endpoints disponíveis para o gerenciamento de reservas pelo cliente estão descritos na Tabela 5.9.

Tabela 5.9: Endpoints de gerenciamento de reservas pelo cliente.

Método	Endpoint	Descrição
POST	/api/clientes/reservas	Cria nova reserva de evento
GET	/api/clientes/reservas/{id}	Busca uma reserva pelo ID
GET	/api/clientes/reservas/me	Lista reservas do cliente (paginado)
PUT	/api/clientes/reservas/{id}	Atualiza dados da reserva (apenas pendentes)
PUT	/api/clientes/reservas/{id}/itens	Atualiza comidas e serviços da reserva
PUT	/api/clientes/reservas/{id}/cancelar	Cancela uma reserva
GET	/api/clientes/reservas/{id}/servicos	Lista serviços da reserva
POST	/api/clientes/reservas/{id}/servicos/{servicoId}	Adiciona serviço à reserva
DELETE	/api/clientes/reservas/{id}/servicos/{servicoId}	Remove serviço da reserva

5.3.9 Endpoints de Notificações

Os endpoints de notificações gerenciam o sistema de comunicação assíncrona com os usuários da plataforma. As notificações são geradas automaticamente pelo sistema em resposta a eventos importantes, como a criação de uma nova reserva, aprovação ou recusa de solicitações, e atualizações de status de eventos. O usuário pode consultar suas notificações, verificar a quantidade de mensagens não lidas e marcar notificações como lidas. Este mecanismo mantém todos os envolvidos informados sobre o andamento de suas reservas e eventos. Os endpoints disponíveis para o gerenciamento de notificações estão descritos na Tabela 5.10.

Tabela 5.10: Endpoints de gerenciamento de notificações.

Método	Endpoint	Descrição
GET	/api/notificacoes	Lista notificações do usuário (paginado)
GET	/api/notificacoes/unread-count	Retorna quantidade de notificações não lidas
PUT	/api/notificacoes/{id}/mark-as-read	Marca notificação como lida
DELETE	/api/notificacoes/{id}	Remove uma notificação

5.3.10 Endpoints de Upload

O endpoint de upload fornece a funcionalidade de envio de arquivos de mídia para o sistema, utilizando o serviço de armazenamento em nuvem Cloudinary. Este recurso foi inicialmente utilizado para testes de upload de imagens e atualmente foi mantido para caso seja necessário algum teste. O endpoint disponível para upload de arquivos está descrito na Tabela 5.11.

Tabela 5.11: Endpoint de upload de arquivos.

Método	Endpoint	Descrição
POST	/api/upload	Faz upload de arquivo para o Cloudinary

5.3.11 Endpoints de Recuperação de Senha

Os endpoints de recuperação de senha fornecem a funcionalidade de envio, validação e recuperação de senha. Utilizando o Gmail SMTP como provedor de e-mail. Graças ao plano gratuito é permitido o envio de até 500 e-mails por dia, o que é suficiente para recuperação de senha. Os endpoints disponíveis para recuperação de senha estão descritos na Tabela 5.12.

Tabela 5.12: Endpoints de recuperação de senha.

Método	Endpoint	Descrição
POST	/api/auth/password/forgot	Recebe o e-mail do usuário e envia um código de 4 dígitos para ele (válido por 15 min)
POST	/api/auth/password/verify-code	Verifica se o código informado é válido e ainda não expirou, retornando true ou false
POST	/api/auth/password/reset	Valida o código novamente e, se válido, atualiza a senha do usuário para a nova senha informada

5.4 Trechos de Código

Os códigos abaixo foram selecionados das classes de serviço (ServiceImpl), que contêm a implementação das regras de negócio do sistema. Dessa forma permitindo visualizar como as validações, transformações de dados e integrações entre componentes são realizadas na prática.

5.4.1 Autenticação

O Quadro 5.1, mostra o serviço de autenticação que é responsável por validar as credenciais do usuário e gerar o token JWT que será utilizado nas requisições subsequentes. Na Figura 5.1, é apresentado o método de login, que implementa a busca do usuário por email ou telefone, validação da senha com BCrypt e geração do token.

```

1 @Service
2 public class AuthServiceImpl implements AuthService {
3
4     private final JwtUtil jwtUtil;
5     private final UserRepository userRepository;
6     private final PasswordEncoder passwordEncoder;
7
8     public AuthServiceImpl(PasswordEncoder passwordEncoder,
9                           JwtUtil jwtUtil,
10                          UserRepository userRepository) {
11         this.passwordEncoder = passwordEncoder;
12         this.jwtUtil = jwtUtil;
13         this.userRepository = userRepository;
14     }
15
16     @Override
17     public AuthResponse login(AuthRequest request, Long buffetId) {
18         try {
19             User user = userRepository.
20                 findByEmailIgnoreCaseAndBuffetId(
21                     request.getUsername(),

```

```
21         buffetId
22     )
23     .or(() -> userRepository.
24         findByTelefoneAndBuffetId(
25             request.getUsername(),
26             buffetId
27         ))
28     .orElseThrow(() ->
29         new AuthenticationException("Usuario nao
30             encontrado neste buffet") {});
31
32     if (!passwordEncoder.matches(request.getPassword(), user.
33         getSenha())) {
34         throw new AuthenticationException("Credenciais
35             invalidas") {};
36     }
37
38     Authentication auth = new
39         UsernamePasswordAuthenticationToken(
40             user, null, user.getAuthorities());
41     SecurityContextHolder.getContext().setAuthentication(auth
42 );
43
44     String token = jwtUtil.generateToken(user);
45
46     return new AuthResponse(token, buffetId, user.getRole());
47
48 } catch (AuthenticationException e) {
49     throw new AuthenticationException("Credenciais invalidas
50         ou buffet incorreto") {};
51 }
52
53 }
```

Quadro 5.1 Método de login da API

O método login implementa as seguintes funcionalidades:

- Busca do usuário: Permite autenticação tanto por email quanto por telefone, utilizando o método `or()` do `Optional` para encadear as buscas (atualmente utilizado apenas email como modo de login).
- Isolamento multi-tenant: A busca considera o `buffetId`, garantindo que cada buffet tenha seu próprio contexto de usuários.
- Validação segura de senha: Utiliza o `PasswordEncoder` (BCrypt) para comparar a senha informada com o hash armazenado, sem expor a senha original.

- Integração com Spring Security: Configura o SecurityContextHolder com as informações do usuário autenticado.
- Geração de token JWT: Delega a criação do token para a classe utilitária JwtUtil, seguindo o princípio de responsabilidade única.

O Quadro 5.2, mostra o DTO de entrada referente a requisição de login. Os atributos possuem anotações para validações de entrada.

```
1
2 @AllArgsConstructor
3 @NoArgsConstructor
4 @Getter
5 @Setter
6 public class AuthRequest {
7
8     @NotNull
9     private String username;
10
11     @NotNull
12     private String password;
13 }
```

Quadro 5.2 DTO de entrada para login

O Quadro 5.3, mostra o DTO de saída referente a resposta de login. Esses dados serão fundamentais para o cliente utilizar em suas próximas requisições ao sistema.

```
1 @AllArgsConstructor
2 @NoArgsConstructor
3 @Getter
4 @Setter
5 public class AuthResponse {
6
7     private String token;
8
9     private Long buffetId;
10
11     private EnumUserRole role;
12
13     public AuthResponse(String token) {
14         this.token = token;
15     }
16
17 }
```

Quadro 5.3 DTO de saída para login

5.4.2 Registro de usuários

O Quadro 5.4, apresenta o método de registro de usuários, que possui validações de unicidade e normalização de dados.

```
1
2 @Service
3 public class UserServiceImpl implements UserService {
4
5     @Autowired
6     private UserRepository repository;
7
8     @Autowired
9     private BuffetRepository buffetRepository;
10
11     @Autowired
12     private PasswordEncoder encoder;
13
14     @Transactional
15     @Override
16     public User register(RegisterRequest req) {
17         if (!buffetRepository.existsById(req.getBuffetId())) {
18             throw new IllegalArgumentException("Buffet nao encontrado");
19         }
20
21         User u = buildUserFromRequest(req);
22         u.setRole(EnumUserRole.CLIENTE);
23         u.setBuffetId(req.getBuffetId());
24         return repository.save(u);
25     }
26
27     private User buildUserFromRequest(RegisterRequest req) {
28         String email = req.getEmail().trim().toLowerCase();
29         String telefone = req.getTelefone() != null
30             ? req.getTelefone().replaceAll("\\D+", "")
31             : null;
32
33         if (repository.existsByEmailIgnoreCaseAndBuffetId(email, req.
34             getBuffetId())) {
35             throw new ConflictException("E-mail ja esta em uso neste buffet."
36                 );
37         }
38
39         if (telefone != null &&
40             repository.existsByTelefoneAndBuffetId(telefone, req.getBuffetId
41                 ())) {
42             throw new ConflictException("Telefone ja esta em uso neste buffet
43                 .");
44         }
45
46         User u = new User();
```



```
43     u.setNome(req.getNome().trim());
44     u.setEmail(email);
45     u.setTelefone(telefone);
46     u.setSenha(encoder.encode(req.getSenha()));
47     u.setStatus(EnumStatus.ATIVO);
48     return u;
49 }
50 }
```

Quadro 5.4 método de Cadastro de Usuários

O método `register` e o método auxiliar `buildUserFromRequest` implementam:

- Validação de pré-condições: Verifica se o buffet existe antes de prosseguir com o cadastro.
- Normalização de dados: O email é convertido para minúsculas e o telefone tem caracteres não numéricos removidos, garantindo consistência.
- Validação de unicidade contextualizada: Email e telefone são validados como únicos dentro do contexto do buffet, permitindo o mesmo dado em buffets diferentes.
- Criptografia de senha: A senha é criptografada com BCrypt antes do armazenamento, seguindo boas práticas de segurança.
- Controle transacional: A anotação `@Transactional` garante atomicidade da operação.

O Quadro 5.5, mostra o DTO de entrada referente ao registro de usuário.

```
1 @AllArgsConstructor
2 @NoArgsConstructor
3 @Getter
4 @Setter
5 public class RegisterRequest {
6
7     @NotBlank
8     private String nome;
9
10    @Email
11    @NotBlank
12    private String email;
13
14    @NotBlank
15    @Size(min = 6, max = 18)
16    private String senha;
17
18    private String telefone;
19
20    @NotNull(message = "BuffetId obrigatório")
```

```

21     private Long buffetId;
22 }

```

Quadro 5.5 DTO de entrada para registro

O Quadro 5.6, mostra a classe de modelo referente aos usuários do sistema, que representa uma entidade que resulta em uma tabela no banco de dados. Posteriormente é convertida para um DTO de resposta pelo controller

```

1  @Entity
2  @Table(name = "users",
3        uniqueConstraints = {
4            @UniqueConstraint(name="uk_users_email", columnNames
5                               = "email"),
6            @UniqueConstraint(name="uk_users_telefone",
7                               columnNames = "telefone")
8        })
9  @Getter
10 @Setter
11 public class User extends BaseEntity implements UserDetails {
12
13     @Id
14     @GeneratedValue(strategy = GenerationType.IDENTITY)
15     private Long id;
16
17     @NotBlank
18     private String nome;
19
20     @Email
21     @NotNull
22     private String email;
23
24     private String telefone;
25
26     @NotNull
27     @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
28     private String senha;
29
30     @Column(name = "buffet_id")
31     private Long buffetId;
32
33     @OneToMany(mappedBy = "cliente", fetch = FetchType.LAZY)
34     private List<Reserva> reservas = new ArrayList<>();
35
36     @Column(name = "role", nullable = false)
37     @Enumerated(EnumType.STRING)
38     private EnumUserRole role;
39
40     @ManyToOne(fetch = FetchType.LAZY)
41     @JoinColumn(name = "foto_id", foreignKey = @ForeignKey(name = "
42         fk_user_foto"))

```

```
40     private Arquivo foto;
41
42     @Override
43     public Collection<? extends GrantedAuthority> getAuthorities() {
44         if (role == null) {
45             return Collections.emptyList();
46         }
47         return List.of(new SimpleGrantedAuthority("ROLE_" + role.name
48             ());
49     }
50
51     @Override
52     public String getPassword() {
53         return senha;
54     }
55
56     @Override
57     public String getUsername() {
58         return email != null ? email : telefone;
59     }
60
61     @Override
62     public boolean isAccountNonExpired() {
63         return true;
64     }
65
66     @Override
67     public boolean isAccountNonLocked() {
68         return true;
69     }
70
71     @Override
72     public boolean isCredentialsNonExpired() {
73         return true;
74     }
75
76     @Override
77     public boolean isEnabled() {
78         return true;
79     }
79 }
```

Quadro 5.6 Entidade User

5.4.3 Reservas

O Quadro 5.7, mostra métodos da classe service de Reservas, cujo objetivo é gerenciar a criação de reservas. Os métodos utilizados na criação de reservas orquestram a validação de dados, criação de entidades relacionadas e notificação automática.

```

1
2 @Service
3 public class ReservaServiceImpl implements ReservaService {
4
5     private final ReservaRepository reservaRepo;
6     private final BuffetRepository buffetRepo;
7     private final UserRepository userRepo;
8     private final ServicoRepository servicoRepo;
9     private final ComidaRepository comidaRepo;
10    private final NotificacaoService notificacaoService;
11
12    public ReservaServiceImpl(ReservaRepository reservaRepo,
13        BuffetRepository buffetRepo, UserRepository userRepo,
14        ServicoRepository servicoRepo, ComidaRepository comidaRepo,
15        NotificacaoService notificacaoService) {
16        this.reservaRepo = reservaRepo;
17        this.buffetRepo = buffetRepo;
18        this.userRepo = userRepo;
19        this.servicoRepo = servicoRepo;
20        this.comidaRepo = comidaRepo;
21        this.notificacaoService = notificacaoService;
22    }
23
24    @Transactional
25    @Override
26    public ReservaResponse criarReserva(Long buffetId, Long clienteId,
27        ReservaRequest req) {
28        User cliente = userRepo.findById(clienteId)
29            .orElseThrow(() -> new NotFoundException("Cliente
30                n o encontrado"));
31
32        Buffet buffet = buffetRepo.findById(req.getBuffetId())
33            .orElseThrow(() -> new NotFoundException("Buffet n o
34                encontrado"));
35
36        boolean dataBloqueada = eventoRepo.existsDataBloqueada(
37            buffet.getId(), req.getDataDesejada(), EnumStatus.
38                ATIVO);
39        if (dataBloqueada) {
40            throw new ConflictException("N o    poss vel criar uma
41                reserva para este dia. A data " + req.getDataDesejada
42                () + " est  bloqueada no calend rio.");
43        }
44
45        Reserva r = new Reserva();

```

```
37     r.setCliente(cliente);
38     r.setBuffet(buffet);
39     r.setQtdPessoas(req.getQtdPessoas());
40     r.setDataDesejada(req.getDataDesejada());
41     r.setHorarioDesejado(req.getHorarioDesejado());
42     r.setStatusReserva(EnumStatusReserva.PENDENTE);
43     r.setStatus(EnumStatus.ATIVO);
44     r.setTitulo(req.getTitulo());
45     r.setDescricao(req.getDescricao());
46
47     Endereco e = new Endereco();
48     EnderecoRequest er = req.getEndereco();
49     e.setRua(er.getRua());
50     e.setNumero(er.getNumero());
51     e.setBairro(er.getBairro());
52     e.setCidade(er.getCidade());
53     e.setEstado(er.getEstado());
54     e.setCep(er.getCep());
55     e.setComplemento(er.getComplemento());
56     e.setStatus(EnumStatus.ATIVO);
57     r.setEndereco(e);
58
59     if (req.getServicoIds() != null && !req.getServicoIds().
60         isEmpty()) {
61         List<Servico> servicos = servicoRepo.findByIdIn(req.
62             getServicoIds());
63         validarServicosDoMesmoBuffet(servicos, buffet.getId());
64         r.setServicos(servicos);
65     }
66
67     if (req.getComidaIds() != null && !req.getComidaIds().isEmpty
68         ()) {
69         List<Comida> comidas = comidaRepo.findByIdIn(req.
70             getComidaIds());
71         validarComidasDoMesmoBuffet(comidas, buffet.getId());
72         r.setComidas(comidas);
73     }
74
75     Reserva reservaSalva = reservaRepo.save(r);
76
77     notificacaoService.criarNotificacaoNovaReserva(reservaSalva,
78         buffet.getOwner());
79
80     return ReservaResponse.of(reservaSalva);
81 }
82
83 private void validarServicosDoMesmoBuffet(List<Servico> servicos
84     , Long buffetId) {
85     if (servicos.stream().anyMatch(s -> !s.getBuffet().getId().
86         equals(buffetId))) {
87         throw new ConflictException("Todos os servi os devem
```

```
81         pertencer ao buffet da reserva");  
82     }  
83  
84     private void validarComidasDoMesmoBuffet(List<Comida> comidas,  
85         Long buffetId) {  
86         if (comidas.stream().anyMatch(c -> !c.getBuffet().getId().  
87             equals(buffetId))) {  
88             throw new ConflictException("Todas as comidas do  
89                 card pio devem pertencer ao buffet da reserva");  
90         }  
91     }  
92 }
```

Quadro 5.7 Métodos do service de Reservas

O método `criarReserva` demonstra várias práticas importantes:

- Validação de entidades relacionadas: Verifica a existência de cliente e buffet antes de criar a reserva.
- Composição de entidades: Cria o endereço como entidade embutida na reserva, demonstrando o relacionamento `OneToOne` com cascata.
- Validação de data bloqueada: Valida se a data selecionada para reserva não está bloqueada, dessa forma mantendo um controle para que o buffet não tenha problemas de várias reservas para um mesmo dia.
- Validação de integridade referencial: Os métodos `validarServicosDoMesmoBuffet` e `validarComidasDoMesmoBuffet` garantem que apenas itens do próprio buffet sejam associados à reserva.
- Notificação automática: Após a criação, o sistema notifica automaticamente o proprietário do buffet sobre a nova reserva.
- Conversão para DTO: O retorno utiliza o método estático `ReservaResponse.of()` para converter a entidade em DTO.

O Quadro 5.8, mostra o DTO de entrada referente a requisição de criação de reserva. Os atributos possuem anotações para validações de entrada.

```
1 @Getter
2 @Setter
3 public class ReservaRequest {
4
5     @NotNull
6     private Long buffetId;
7
8     @NotNull
9     private Integer qtdPessoas;
10
11     @NotNull
12     private LocalDate dataDesejada;
13
14     @NotNull
15     private LocalTime horarioDesejado;
16
17     @Valid
18     @NotNull
19     private EnderecoRequest endereco;
20
21     private List<Long> servicoIds;
22     private List<Long> comidaIds;
23     private String titulo;
24     private String descricao;
25
26 }
27
```

Quadro 5.8 DTO de entrada para login

O Quadro 5.9, mostra o DTO de saída referente a resposta de criação da reserva. Dados que são fundamentais para listagem e visualização da reserva criada no aplicativo.

```
1 @Getter
2 @Setter
3 public class ReservaResponse {
4
5     private Long id;
6     private EnumStatusReserva statusReserva;
7     private EnumStatus status;
8
9     private Long buffetId;
10    private Long clienteId;
11    private String nomeCliente;
12    private String emailCliente;
13    private String telefoneCliente;
14    private Long eventoId;
15
16    private LocalDate dataDesejada;
17    private LocalTime horarioDesejado;
18    private Integer qtdPessoas;
19
20 }
```

```

19
20     private String titulo;
21     private String descricao;
22     private EnderecoResponse endereco;
23     private List<ComidaResumoResponse> comidas;
24     private List<ServicoResumoResponse> servicos;
25
26     public static ReservaResponse of(Reserva r) {
27         ReservaResponse dto = new ReservaResponse();
28         dto.id = r.getId();
29         dto.statusReserva = r.getStatusReserva();
30         dto.status = r.getStatus();
31         dto.bufferId = r.getBuffet().getId();
32         dto.clienteId = r.getClient().getId();
33         dto.nomeCliente = r.getClient().getNome();
34         dto.emailCliente = r.getClient().getEmail();
35         dto.telefoneCliente = r.getClient().getTelefone();
36         dto.eventoId = (r.getEvento() != null ? r.getEvento().getId()
37             : null);
38         dto.dataDesejada = r.getDataDesejada();
39         dto.horarioDesejado = r.getHorarioDesejado();
40         dto.qtdPessoas = r.getQtdPessoas();
41         dto.titulo = r.getTitulo();
42         dto.descricao = r.getDescricao();
43         dto.endereco = EnderecoResponse.of(r.getEndereco());
44
45         if (r.getComidas() != null) {
46             dto.comidas = r.getComidas().stream()
47                 .map(ComidaResumoResponse::of).toList();
48         } else {
49             dto.comidas = List.of();
50         }
51
52         if (r.getServicos() != null) {
53             dto.servicos = r.getServicos().stream()
54                 .map(ServicoResumoResponse::of).toList();
55         } else {
56             dto.servicos = List.of();
57         }
58
59         return dto;
60     }

```

Quadro 5.9 DTO de saída para reserva

O Quadro 5.10, mostra o método da classe service de Reservas, cujo objetivo é aprovar uma reserva criada anteriormente por um cliente e gerar de forma automática um evento na agenda do buffet.


```
1
2
3 @Transactional
4 @Override
5 public ReservaResponse aprovarReserva(Long buffetId, Long
6     reservaId, Long ownerId, AprovarReservaRequest req) {
7     Reserva r = reservaRepo.findById(reservaId)
8         .orElseThrow(() -> new NotFoundException("Reserva
9         n o encontrada"));
10
11     if (!r.getBuffet().getOwner().getId().equals(ownerId))
12         throw new ForbiddenException("Voc n o o dono deste
13         buffet");
14
15     if (r.getStatusReserva() != EnumStatusReserva.PENDENTE)
16         throw new ConflictException("Somente reservas PENDENTES
17         podem ser aprovadas");
18
19     Evento ev = new Evento();
20     ev.setReserva(r);
21     ev.setBuffet(r.getBuffet());
22
23     ev.setStatusEvento(EnumStatusEvento.AGENDADO);
24
25     boolean shouldBlockDay = req.getBlockDay() != null && req.
26         getBlockDay();
27     ev.setBloquearCalendario(shouldBlockDay);
28
29     ev.setValor(req.getValor());
30     ev.setStatus(EnumStatus.ATIVO);
31
32     ev.setNome(r.getTitulo());
33     ev.setDescricao(r.getDescricao());
34
35     ev.setDataEvento(r.getDataDesejada());
36     ev.setHoraEvento(r.getHorarioDesejado());
37
38     r.setEvento(ev);
39     r.setStatusReserva(EnumStatusReserva.APROVADA);
40
41     notificacaoService.criarNotificacaoReservaAprovada(r, r.
42         getCliente());
43
44     return ReservaResponse.of(r);
45 }
```

Quadro 5.10 Método de aprovação de reserva

O método de aprovação demonstra:

- Validação de autorização: Apenas o proprietário do buffet pode aprovar reservas.

- Validação de estado: Somente reservas pendentes podem ser aprovadas.
- Criação automática de evento: Um evento é criado e associado à reserva no momento da aprovação.
- Bloqueio de dia: Ao aprovar a reserva o buffet pode optar por bloquear aquele dia, dessa forma outro cliente não poderá criar uma nova reserva caso o dia esteja bloqueado.
- Relacionamento bidirecional: A reserva e o evento são vinculados mutuamente.
- Notificação ao cliente: O cliente é notificado automaticamente sobre a aprovação.

O Quadro 5.11, mostra o método da classe service de Reservas, cujo objetivo é permitir ao buffet reverter o cancelamento de alguma reserva cancelada anteriormente por ele ou por algum cliente.

```
1      @Transactional
2      @Override
3      public ReservaResponse reverterCancelamentoReserva(Long buffetId,
4      Long reservaId, Long ownerId) {
5          Reserva r = reservaRepo.findById(reservaId)
6              .orElseThrow(() -> new NotFoundException("Reserva nao
7                  encontrada"));
8
9          if (!r.getBuffet().getOwner().getId().equals(ownerId)) {
10              throw new ForbiddenException("Voce nao e o dono deste buffet"
11                  );
12          }
13
14          if (!r.getBuffet().getId().equals(buffetId)) {
15              throw new ConflictException("Reserva nao pertence ao buffet
16                  informado");
17          }
18
19          if (r.getStatusReserva() != EnumStatusReserva.CANCELADA) {
20              throw new ConflictException("So e possivel reverter reservas
21                  canceladas");
22          }
23
24          LocalDate hoje = LocalDate.now();
25          if (r.getDataDesejada().isBefore(hoje)) {
26              throw new ConflictException("Nao e possivel reverter uma
27                  reserva cuja data ja passou");
28          }
29
30          if (r.getEvento() != null) {
31              r.getEvento().setStatusEvento(EnumStatusEvento.AGENDADO);
32          }
```

```

27         r.setStatusReserva(EnumStatusReserva.APROVADA);
28     } else {
29         r.setStatusReserva(EnumStatusReserva.PENDENTE);
30     }
31
32     r.setStatus(EnumStatus.ATIVO);
33
34     notificacaoService.criarNotificacaoReservaReativada(r, r.
35         getCliente());
36
37     return ReservaResponse.of(reservaRepo.save(r));
38 }

```

Quadro 5.11 Método de reverter cancelamento de reserva

O método de reversão de reserva implementa as seguintes funcionalidades:

- Validação temporal: Não permite reverter reservas cuja data já passou.
- Reversão inteligente: Se tinha evento, volta para APROVADA e reativa o evento; senão, volta para PENDENTE.
- Reativação completa: Status da reserva volta para ATIVO.
- Notificação ao cliente: O cliente é informado que sua reserva foi reativada.

5.4.4 Evento

O Quadro 5.12, mostra o método da classe service de Evento, cujo objetivo é permitir ao cliente atualizar seu evento. Como regra o cliente só pode atualizar o evento até 5 dias antes da data agendada.

```

1
2 @Transactional
3 @Override
4 public EventoResponse atualizarEventoPeloCliente(Long buffetId, Long
5     id, Long clienteId, ClienteEventoUpdateRequest req) {
6
7     Evento e = repository.findById(id)
8         .orElseThrow(() -> new NotFoundException("Evento nao
9             encontrado"));
10
11     if (!e.getBuffet().getId().equals(buffetId)) {
12         throw new ConflictException("Evento nao pertence ao buffet
13             informado");
14     }
15
16     Reserva reserva = e.getReserva();

```

```
14     if (reserva == null) {
15         throw new ConflictException("Evento nao possui reserva
16             associada");
17     }
18
19     if (!reserva.getClient().getId().equals(clienteId)) {
20         throw new ForbiddenException("Este evento nao pertence a voce
21             ");
22     }
23
24     if (e.getStatusEvento() != EnumStatusEvento.AGENDADO &&
25         e.getStatusEvento() != EnumStatusEvento.PENDENTE) {
26         throw new ConflictException("So e possivel editar eventos
27             agendados ou pendentes");
28     }
29
30     LocalDate hoje = LocalDate.now();
31     LocalDate limiteEdicao = e.getDataEvento().minusDays(5);
32     if (hoje.isAfter(limiteEdicao)) {
33         throw new ConflictException(
34             "So e possivel editar o evento ate 5 dias antes. Prazo
35             limite: " + limiteEdicao);
36     }
37
38     if (req.getQtdPessoas() != null) {
39         reserva.setQtdPessoas(req.getQtdPessoas());
40     }
41
42     if (req.getDataEvento() != null) {
43         if (req.getDataEvento().isBefore(hoje)) {
44             throw new ConflictException("A nova data do evento nao
45                 pode estar no passado");
46         }
47         e.setDataEvento(req.getDataEvento());
48         reserva.setDataDesejada(req.getDataEvento());
49     }
50
51     if (req.getHoraEvento() != null) {
52         e.setHoraEvento(req.getHoraEvento());
53         reserva.setHorarioDesejado(req.getHoraEvento());
54     }
55
56     if (req.getEndereco() != null) {
57         Endereco endereco = reserva.getEndereco();
58         if (endereco == null) {
59             endereco = new Endereco();
60             endereco.setStatus(EnumStatus.ATIVO);
61             reserva.setEndereco(endereco);
62         }
63         endereco.setRua(req.getEndereco().getRua());
64         endereco.setNumero(req.getEndereco().getNumero());
```

```

60     endereco.setBairro(req.getEndereco().getBairro());
61     endereco.setCidade(req.getEndereco().getCidade());
62     endereco.setEstado(req.getEndereco().getEstado());
63     endereco.setCep(req.getEndereco().getCep());
64     endereco.setComplemento(req.getEndereco().getComplemento());
65 }
66
67 if (req.getComidaIds() != null) {
68     List<Comida> comidas = comidaRepository.findByIdIn(req.
69         getComidaIds());
70     validarComidasDoMesmoBuffet(comidas, buffetId);
71     reserva.setComidas(comidas);
72 }
73
74 if (req.getServicoIds() != null) {
75     List<Servico> servicos = servicoRepository.findByIdIn(req.
76         getServicoIds());
77     validarServicosDoMesmoBuffet(servicos, buffetId);
78     reserva.setServicos(servicos);
79 }
80
81 return new EventoResponse(repository.save(e));
82 }

```

Quadro 5.12 Método de atualizar evento

O método `atualizarEventoPeloCliente` implementa as seguintes funcionalidades:

- Prazo de 5 dias: O cliente só pode alterar o evento até 5 dias antes da data.
- Atualização parcial: Apenas os campos informados são atualizados.
- Sincronização de dados: Alterações no evento são refletidas na reserva associada.
- Mensagem informativa: A exceção inclui a data limite para orientar o usuário.

O Quadro 5.13, mostra o DTO de entrada que permite atualização parcial dos campos do evento.

```

1 @Getter
2 @Setter
3 public class ClienteEventoUpdateRequest {
4
5     private List<Long> comidaIds;
6
7     private List<Long> servicoIds;
8
9     @Min(value = 1, message = "Quantidade de pessoas deve ser maior
10         que zero")
11     private Integer qtdPessoas;

```

```

11
12     private LocalDate dataEvento;
13
14     private LocalTime horaEvento;
15
16     @Valid
17     private EnderecoRequest endereco;
18 }

```

Quadro 5.13 DTO de entrada para atualização de evento pelo cliente

O Quadro 5.14, mostra o DTO de saída com as informações do evento.

```

1 @Getter
2 @Setter
3 @NoArgsConstructor
4 @AllArgsConstructor
5 public class EventoResponse {
6
7     private Long id;
8     private String nome;
9     private EnumStatusEvento statusEvento;
10    private EnumStatus status;
11
12    private Long buffetId;
13    private Long reservaId;
14    private Long clienteId;
15
16    private LocalDate dataEvento;
17    private LocalTime horaEvento;
18    private BigDecimal valor;
19    private String descricao;
20
21    public EventoResponse(Evento e) {
22        this.id = e.getId();
23        this.nome = e.getNome();
24        this.statusEvento = e.getStatusEvento();
25        this.status = e.getStatus();
26        this.buffetId = e.getBuffet() != null ? e.getBuffet().getId()
27            : null;
28        this.reservaId = (e.getReserva() != null) ? e.getReserva().
29            getId() : null;
30        this.clienteId = (e.getReserva() != null && e.getReserva().
31            getCliente() != null)
32            ? e.getReserva().getCliente().getId() : null;
33        this.dataEvento = e.getDataEvento();
34        this.horaEvento = e.getHoraEvento();
35        this.valor = e.getValor();
36        this.descricao = e.getDescricao();
37    }
38 }

```

Quadro 5.14 DTO de saída para evento

O Quadro 5.15, mostra o método da classe service de Evento, cujo objetivo é permitir ao buffet alterar os valores orçados para o evento. Como regra o buffet pode atualizar o valor do evento até 3 dias antes da data.

```
1
2 @Transactional
3 @Override
4 public EventoResponse updateValor(Long buffetId, Long id, BigDecimal
   valor, Long ownerId) {
5     Evento e = repository.findById(id)
6         .orElseThrow(() -> new NotFoundException("Evento nao
           encontrado"));
7
8     if (!e.getBuffet().getId().equals(buffetId)) {
9         throw new ConflictException("Evento nao pertence ao buffet
           informado");
10    }
11
12    if (!e.getBuffet().getOwner().getId().equals(ownerId)) {
13        throw new ForbiddenException("Voce nao e o dono deste buffet"
14        );
15    }
16
17    if (e.getStatusEvento() != EnumStatusEvento.AGENDADO) {
18        throw new ConflictException("So e possivel alterar o valor de
           eventos agendados");
19    }
20
21    LocalDate hoje = LocalDate.now();
22    LocalDate limiteEdicao = e.getDataEvento().minusDays(3);
23    if (hoje.isAfter(limiteEdicao)) {
24        throw new ConflictException(
25            "So e possivel alterar o valor ate 3 dias antes do evento
26            . Prazo limite: "
27            + limiteEdicao);
28    }
29
30    BigDecimal valorAntigo = e.getValor();
31    e.setValor(valor);
32
33    Evento eventoAtualizado = repository.save(e);
34
35    if (eventoAtualizado.getReserva() != null) {
36        notificacaoService.criarNotificacaoEventoValorAtualizado(
37            eventoAtualizado,
38            eventoAtualizado.getReserva().getCliente(),
39            valorAntigo,
40            valor
41        );
42    }
```

```
42     return new EventoResponse(eventoAtualizado);  
43 }
```

Quadro 5.15 Método de atualização de valor do evento

O método `updateValor` implementa as seguintes funcionalidades:

- Prazo de 3 dias: Alterações de valor são permitidas até 3 dias antes do evento.
- Notificação completa: O cliente recebe notificação com valor antigo e novo.
- Validação de estado: Apenas eventos AGENDADOS podem ter valor alterado.

O Quadro 5.16 mostra o DTO de entrada para atualização de valor.

```
1 @Getter  
2 @Setter  
3 public class UpdateValorRequest {  
4  
5     @NotNull(message = "Valor e obrigatorio")  
6     @Positive(message = "Valor deve ser positivo")  
7     private BigDecimal valor;  
8 }
```

Quadro 5.16 DTO de entrada para atualização de valor

5.5 Projeto do Banco de Dados

A geração das tabelas, colunas e relacionamentos do banco de dados foi realizada automaticamente pelo próprio Spring, a partir das classes de entidade (POJOs) criadas com as anotações específicas para classes e atributos. A Figura 5.2 apresenta o diagrama do banco de dados gerado.

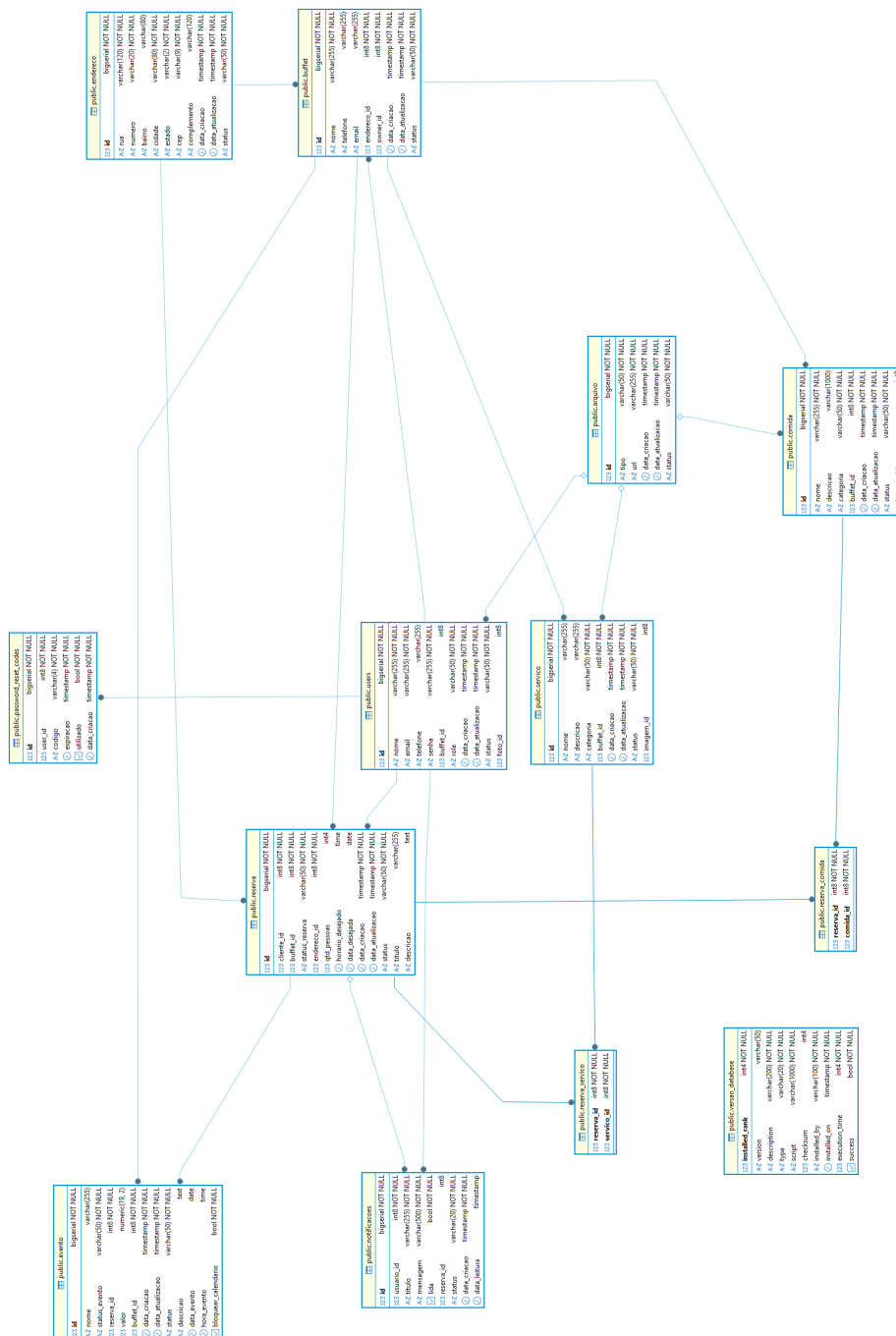


Figura 5.2: Diagrama do banco de dados.

Este capítulo detalhou a implementação do *back-end* do sistema, abrangendo a configuração inicial do projeto Spring Boot, a estrutura de pacotes adotada, a definição das entidades e DTOs, a lógica dos serviços, os controladores REST e o projeto do banco de dados com geração automática pelo JPA/Hibernate. A *API RESTful* construída constitui a camada central de regras de negócio, expondo os endpoints consumidos pela interface do usuário. Com o *back-end* implementado e funcional, o próximo capítulo apresenta a implementação do *front-end* com Angular e Ionic, descrevendo a organização do código segundo a arquitetura Feature-Sliced Design, os componentes desenvolvidos e a integração com a API.

CAPÍTULO 6

IMPLEMENTAÇÃO DO *FRONT-END* (ANGULAR/IONIC)

Após o levantamento e análise dos requisitos apresentados no Capítulo 4 e a implementação do *back-end* detalhada no Capítulo 5, iniciou-se o desenvolvimento da interface do usuário do sistema. O *front-end* foi construído com **Angular 20** e **Ionic 8**, utilizando **TypeScript** e empacotamento híbrido via **Capacitor 7** para execução em ambiente móvel Android. Essa escolha permite reutilização de código, boa experiência em dispositivos móveis e integração direta com a API REST desenvolvida no capítulo anterior, garantindo comunicação entre cliente e servidor de forma padronizada.

6.1 Configuração inicial

A inicialização do projeto foi realizada por meio do **Ionic CLI**, que configura automaticamente o Angular e define a estrutura base de módulos, páginas e configuração de *build*. O projeto utiliza a arquitetura *standalone components*, dispensando a necessidade de módulos NgModule tradicionais, o que simplifica a organização e melhora a *performance* em tempo de execução.

O desenvolvimento foi conduzido em ambiente compatível com TypeScript 5.8, com suporte a compilação incremental e verificação estrita de tipos. A configuração do projeto, definida no arquivo `angular.json`, especifica o diretório de saída (`www`), os arquivos de estilo globais e as configurações de *build* utilizadas pelo **Capacitor** para empacotar a aplicação no dispositivo Android.

O ponto de entrada da aplicação é definido no arquivo `main.ts`, que configura os *providers* globais, incluindo o roteamento, cliente HTTP com *interceptors*, e inicializadores de tema e sessão (`APP_INITIALIZER`). Essa configuração garante que serviços essenciais sejam

carregados antes da inicialização completa do aplicativo.

As principais dependências de *runtime* utilizadas no *front-end* são apresentadas na Tabela 6.1.

Tabela 6.1: Principais dependências do *front-end* Angular/Ionic.

Dependência	Versão	Descrição
@angular/core	20.0.0	Núcleo do <i>framework Angular</i> , responsável por injeção de dependências, ciclo de vida de componentes e sistema de módulos.
@angular/router	20.0.0	Módulo de roteamento do <i>Angular</i> , gerenciando navegação entre páginas e <i>guards</i> de autenticação.
@angular/forms	20.0.6	Suporte a formulários reativos e <i>template-driven</i> , validações e controle de estado.
@angular/common/http	20.0.0	Cliente HTTP para comunicação com a API REST, incluindo suporte a <i>interceptors</i> .
@ionic/angular	8.0.0	Biblioteca de componentes UI do Ionic, otimizados para dispositivos móveis.
@capacitor/core	7.4.1	<i>Runtime</i> do <i>Capacitor</i> para acesso a funcionalidades nativas do dispositivo.
@capacitor/android	7.4.1	<i>Plugin</i> do <i>Capacitor</i> para empacotamento e execução em Android.
@capacitor/camera	7.0.2	<i>Plugin</i> para acesso à câmera e galeria de fotos do dispositivo.
@capacitor/preferences	7.0.2	Armazenamento local persistente de preferências do usuário.
rxjs	7.8.0	Biblioteca para programação reativa, utilizada para gerenciar <i>streams</i> assíncronos.
ionicons	7.0.0	Biblioteca de ícones otimizada para aplicações <i>Ionic</i> .
@angular/material	20.2.3	Componentes Material Design integrados ao Angular.

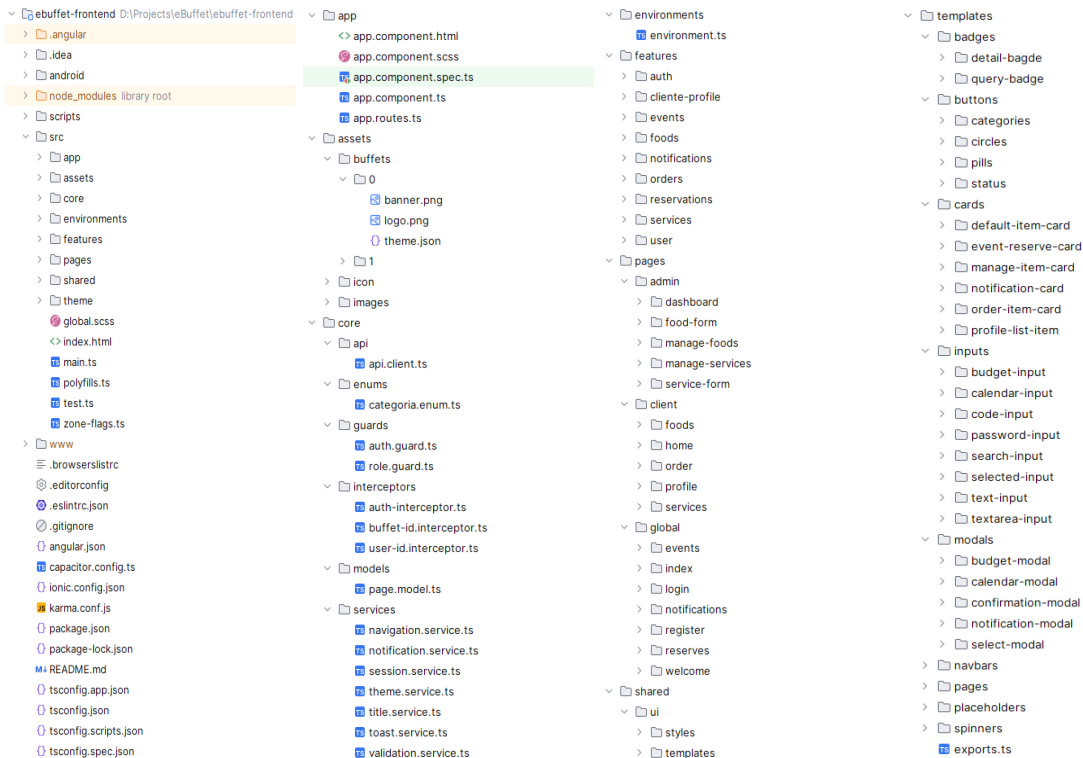
Além das dependências de *runtime*, o projeto utiliza ferramentas de desenvolvimento como @angular/cli para *scaffolding* e *build*, ESLint para *linting* de código, e TypeScript como linguagem principal de desenvolvimento.

6.2 Organização de pastas

Para facilitar manutenção e evolução do código, o *front-end* adota uma organização modular inspirada na arquitetura **Feature-Sliced Design (FSD)**, separando responsabilidades por camadas e por funcionalidades do domínio. Essa abordagem melhora a escalabilidade do

projeto, reduz acoplamento entre módulos e organiza o crescimento do sistema conforme novas funcionalidades são incorporadas.

A estrutura principal do projeto está organizada no diretório `src/`, que contém as seguintes camadas, conforme ilustrado na Figura 6.1:



(a) Estrutura geral. (b) Estrutura subpacotes 1. (c) Estrutura subpacotes 2. (d) Estrutura subpacotes 3.

Figura 6.1: Estrutura de pacotes do front-end baseada em Feature-Sliced Design.

De forma geral, a divisão segue os seguintes objetivos:

- **app:** concentra configurações globais do aplicativo, incluindo o componente raiz (`app.component.ts`), definição de rotas (`app.routes.ts`) e inicialização do *framework*. Este diretório é responsável por orquestrar a navegação inicial baseada no estado de autenticação do usuário.
- **pages:** agrupa as páginas do sistema, organizadas por perfil de usuário. A subpasta `global` contém telas acessíveis a todos (ex.: *login*, registro, notificações, reservas e eventos). A subpasta `client` reúne páginas exclusivas do cliente (*home*, catálogo de comidas e serviços, carrinho de pedido e perfil). A subpasta `admin` concentra funcionalidades do buffet (*dashboard*, gerenciamento de comidas e serviços).
- **features:** agrupa funcionalidades de negócio coesas, organizadas por domínio. Cada *feature* contém seus próprios modelos de dados (`model`) e serviços (`services`), que

encapsulam tanto a comunicação com a API quanto a lógica de negócio específica da funcionalidade. As *features* implementadas incluem: `auth` (autenticação), `foods` (comidas), `services` (serviços), `reservations` (reservas), `events` (eventos), `notifications` (notificações), `orders` (carrinho de pedido), `user` (perfil do usuário) e `client-profile` (gestão do perfil do cliente).

- **core:** contém infraestrutura compartilhada e transversal ao sistema. A subpasta `api` centraliza o cliente HTTP (`api.client.ts`). A subpasta `guards` implementa proteções de rota (`auth.guard.ts` para autenticação e `role.guard.ts` para controle de perfil). A subpasta `interceptors` define *interceptors* HTTP para injeção automática de *token* JWT e outros *headers*. A subpasta `services` agrupa serviços globais como `session.service.ts` (gerenciamento de sessão), `toast.service.ts` (notificações *toast*), `validation.service.ts` (validações de formulário) e `theme.service.ts` (controle de tema). A subpasta `enums` centraliza enumerações compartilhadas (ex.: `categoria.enum.ts`).
- **shared:** inclui componentes de interface reutilizáveis, organizados na subpasta `ui/templates`. Contém componentes de *layout* (pages), barras de navegação (`navbars`), botões (`buttons`), *inputs* (`inputs`), *cards* (`cards`), modais (`modals`), *badges* (`badges`), *placeholders* (`placeholders`) e *spinners* (`spinners`).
- **assets:** armazena recursos estáticos como imagens, ícones e fontes. Este diretório possui papel fundamental na implementação da arquitetura *multi-tenant*, conforme detalhado na Seção 6.15, organizando recursos específicos de cada buffet em subdiretórios isolados.
- **environments:** define variáveis de ambiente para diferentes contextos (desenvolvimento, produção), incluindo a URL base da API e o identificador do buffet (`buffetId`) utilizado na personalização da interface.
- **theme:** centraliza arquivos de estilo globais e definição de variáveis CSS para customização do tema Ionic.

6.3 Login e autenticação

A autenticação controla o acesso às funcionalidades internas do sistema, garantindo que apenas usuários autenticados consumam os recursos protegidos da API. O fluxo de *login* é imple-

mentado no componente LoginComponent (pages/global/login), que utiliza formulários reativos do Angular para capturar e validar as credenciais do usuário.

O processo de autenticação segue as seguintes etapas:

1. O usuário preenche o formulário de *login* com e-mail e senha.
2. O componente valida os campos utilizando o `ValidationService`, verificando presença de valores, formato de e-mail e tamanho mínimo da senha (6 caracteres).
3. Se as validações passarem, o `AuthService` envia as credenciais para o *endpoint* `/auth/login` da API.
4. Em caso de sucesso, a API retorna um *token* JWT e os *roles* do usuário.
5. O *token* é armazenado no `localStorage` com a chave `access_token`.
6. O serviço então consulta o *endpoint* `/auth/me` para obter os dados completos do usuário autenticado (id, nome, e-mail, telefone e *roles*).
7. Os dados do usuário são armazenados no `SessionService`, que gerencia o estado de autenticação globalmente utilizando o `Capacitor Preferences` para persistência.
8. Com base nos *roles* retornados, o sistema redireciona o usuário para a área apropriada: `/admin/dashboard` para usuários com *role* `BUFFET` ou `ADMIN`, e `/client/home` para usuários com *role* `CLIENT`.

O Quadro 6.1 apresenta a implementação do método de *login* no componente.

```
1 async toDoLogin(event: any): Promise<void> {  
2   event.target.blur();  
3  
4   const email = this.loginForm.get('email')?.value;  
5   const password = this.loginForm.get('password')?.value;  
6  
7   const requiredValidation = this.validationService.  
8     validateRequiredFields(  
9       { email, password }  
10    );  
11   if (!requiredValidation.isValid) {  
12     await this.toastService.warning(requiredValidation.message!);  
13     return;  
14   }  
15 }
```

```
15  const emailValidation = this.validationService.validateEmail(email)
16      ;
17  if (!emailValidation.isValid) {
18      await this.toastService.warning(emailValidation.message!);
19      return;
20  }
21  const passwordValidation = this.validationService.validatePassword(
22      password, 6
23  );
24  if (!passwordValidation.isValid) {
25      await this.toastService.warning(passwordValidation.message!);
26      return;
27  }
28
29  const loginResult = await this.authService.login(
30      email, password, this.sessionService
31  );
32
33  if (!loginResult.success) {
34      await this.toastService.error(
35          'E-mail ou senha incorretos. Tente novamente.'
36      );
37      return;
38  }
39
40  try {
41      if (loginResult.roles?.includes('BUFFET') ||
42          loginResult.roles?.includes('ADMIN')) {
43          await this.navCtrl.navigateRoot('/admin/dashboard',
44              { animated: true });
45      } else {
46          await this.navCtrl.navigateRoot('/client/home',
47              { animated: true });
48      }
49  } catch (error) {
50      await this.toastService.error(
51          'Erro ao redirecionar. Tente novamente.'
52      );
53  }
54  }
```

Quadro 6.1 Método de *login* no componente LoginComponent.

O sistema implementa regras de negócio específicas para o cadastro, conforme definido no Capítulo 4:

- **Cadastro permitido apenas para clientes:** o cliente pode criar conta informando dados básicos através do componente RegisterComponent. Já o buffet não realiza auto-cadastro, acessando o sistema por credenciais previamente fornecidas.

- **Controle de rotas protegidas:** páginas internas exigem autenticação via AuthGuard; além disso, páginas específicas exigem o perfil correspondente (cliente ou buffet) via RoleGuard.

O armazenamento do *token* e a inclusão automática do JWT nas requisições são realizados por meio do *authInterceptor*, apresentado no Quadro 6.2.

```
1 import { HttpInterceptorFn } from "@angular/common/http";
2
3 export const authInterceptor: HttpInterceptorFn = (req, next) => {
4   const token = localStorage.getItem('access_token');
5
6   if (token) {
7     const headers = { 'Authorization': `Bearer ${token}` };
8     return next(req.clone({ setHeaders: headers }));
9   }
10
11   return next(req);
12 };
```

Quadro 6.2 *Interceptor* HTTP para injeção automática do *token* JWT.

Este *interceptor* é registrado globalmente na configuração do Angular (*main.ts*), garantindo que todas as requisições HTTP incluam automaticamente o *header* *Authorization* quando um *token* estiver disponível.

6.4 Proteção de rotas e controle de perfil

A proteção de rotas é implementada por meio de dois *guards*: *AuthGuard* e *RoleGuard*. O *AuthGuard* verifica se o usuário está autenticado e se o *token* JWT é válido (não expirado). Em caso negativo, redireciona para a página de boas-vindas.

O Quadro 6.3 apresenta a implementação do *AuthGuard*.

```
1 @Injectable({ providedIn: 'root' })
2 export class AuthGuard implements CanActivate {
3   constructor(
4     private sessionService: SessionService,
5     private navCtrl: NavController
6   ) {}
7
8   canActivate(): boolean {
9     const isLoggedIn = this.sessionService.isAuthenticated();
10    const token = localStorage.getItem('access_token');
11
12    if (!isLoggedIn || !token) {
13      this.sessionService.logout();
```

```

14     this.navCtrl.navigateRoot('/welcome', { animated: true });
15     return false;
16 }
17
18 if (!isJwtValid(token)) {
19     this.sessionService.logout();
20     this.navCtrl.navigateRoot('/welcome', { animated: true });
21     return false;
22 }
23
24 return true;
25 }
26 }
27
28 function isJwtValid(token: string): boolean {
29     try {
30         const payload = JSON.parse(atob(token.split('.')[1]));
31         if (!payload?.exp) return true;
32         const now = Math.floor(Date.now() / 1000);
33         return payload.exp > now;
34     } catch {
35         return false;
36     }
37 }

```

Quadro 6.3 Guard de autenticação verificando token JWT.

O RoleGuard verifica se o usuário possui um dos *roles* necessários para acessar determinada rota, redirecionando para a área apropriada caso contrário.

```

1 @Injectable({ providedIn: 'root' })
2 export class RoleGuard implements CanActivate {
3     constructor(
4         private sessionService: SessionService,
5         private navCtrl: NavController
6     ) {}
7
8     canActivate(route: ActivatedRouteSnapshot): boolean {
9         const requiredRoles = route.data['roles'] as string[];
10
11         if (!requiredRoles || requiredRoles.length === 0) {
12             return true;
13         }
14
15         const user = this.sessionService.getUser();
16         const userRoles = user?.roles || [];
17
18         const hasRole = requiredRoles.some(
19             role => userRoles.includes(role)
20         );
21
22         if (!hasRole) {

```

```
23     if (userRoles.includes('BUFFET') ||
24         userRoles.includes('ADMIN')) {
25         this.navCtrl.navigateRoot('/admin/dashboard',
26             { animated: true });
27     } else {
28         this.navCtrl.navigateRoot('/client/home',
29             { animated: true });
30     }
31     return false;
32 }
33
34 return true;
35 }
36 }
```

Quadro 6.4 *Guard de controle de perfil (roles).*

A configuração de rotas (`app.routes.ts`) utiliza esses *guards* para proteger áreas específicas, conforme apresentado no Quadro 6.5.

```
1 export const routes: Routes = [
2     // Rotas publicas
3     { path: '', redirectTo: 'index', pathMatch: 'full' },
4     { path: 'index', component: IndexComponent },
5     { path: 'welcome', component: WelcomeComponent },
6     { path: 'login', component: LoginComponent },
7     { path: 'register', component: RegisterComponent },
8     { path: 'request-password-reset', component:
9         RequestPasswordResetComponent },
10    { path: 'forgot-password', component: ForgotPasswordComponent },
11    { path: 'new-password', component: NewPasswordComponent },
12
13    // Rotas globais protegidas (autenticacao obrigatoria)
14    {
15        path: 'notifications',
16        component: NotificationsComponent,
17        canActivate: [AuthGuard]
18    },
19    {
20        path: 'reserves',
21        component: ReservesComponent,
22        canActivate: [AuthGuard]
23    },
24    {
25        path: 'events',
26        component: EventsComponent,
27        canActivate: [AuthGuard]
28    },
29
30    // Rotas do cliente (role CLIENT obrigatorio)
31    {
32        path: 'client',
```

```

32   canActivate: [AuthGuard, RoleGuard],
33   data: { roles: [ 'CLIENT' ] },
34   children: [
35     { path: 'home', component: HomeComponent },
36     { path: 'foods', component: FoodsComponent },
37     { path: 'order', component: OrderComponent },
38     { path: 'services', component: ServicesComponent },
39     { path: 'profile', component: ProfileComponent }
40   ]
41 },
42
43 // Rotas do buffet (role BUFFET obrigatorio)
44 {
45   path: 'admin',
46   canActivate: [AuthGuard, RoleGuard],
47   data: { roles: [ 'BUFFET' ] },
48   children: [
49     { path: 'dashboard', component: DashboardComponent },
50     { path: 'manage-foods', component: ManageFoodsComponent },
51     { path: 'food-form', component: FoodFormComponent },
52     { path: 'manage-services', component: ManageServicesComponent },
53     { path: 'service-form', component: ServiceFormComponent }
54   ]
55 }
56 ];

```

Quadro 6.5 Configuração de rotas com *guards* de autenticação e perfil.

6.5 Navegação e áreas por perfil

O sistema disponibiliza funcionalidades distintas conforme o perfil autenticado, seguindo os dois perfis estabelecidos no Capítulo 4: **Cliente** e **Buffet**. O cliente utiliza a aplicação para consultar comidas e serviços, montar pedidos, criar reservas e acompanhar reservas e eventos. O buffet utiliza a aplicação para gerenciar comidas, serviços, aprovar/recusar reservas recebidas e gerenciar eventos originados a partir de reservas aprovadas.

Essa separação é essencial em um cenário *multi-tenant*, garantindo que cada buffet visualize e administre exclusivamente seus próprios dados, mantendo isolamento lógico conforme definido no Capítulo 4. A implementação completa da arquitetura *multi-tenant* é detalhada na Seção 6.15, incluindo mecanismos de personalização visual e isolamento de dados por estabelecimento.

6.6 Redefinição de senha

O sistema implementa o requisito RF15 (Redefinição de senha) através de um fluxo completo que permite ao usuário recuperar o acesso à sua conta. O processo é dividido em três etapas principais:

1. **Solicitação de recuperação:** o usuário acessa a tela de solicitação de recuperação (`RequestPasswordResetComponent`) e informa seu e-mail cadastrado. O sistema valida o formato do e-mail e envia uma requisição ao *back-end*, que gera um código de 6 dígitos e envia por e-mail.
2. **Validação do código:** o usuário acessa a tela de validação (`ForgotPasswordComponent`) e insere o código de 6 dígitos recebido por e-mail. O sistema valida a autenticidade e validade do código.
3. **Nova senha:** após validação bem-sucedida do código, o usuário é direcionado para a tela de nova senha (`NewPasswordComponent`), onde deve informar e confirmar a nova senha. O sistema valida a correspondência entre as senhas e o tamanho mínimo (6 caracteres) antes de processar a alteração.

Essa funcionalidade está disponível tanto para clientes quanto para buffets, conforme estabelecido nos requisitos do Capítulo 4.

6.7 Gerenciamento de comidas

O gerenciamento de comidas é exclusivo do perfil buffet e permite:

- **Cadastro:** inclusão de comida com nome, descrição, categoria e imagem através do componente `FoodFormComponent`.
- **Listagem e consulta:** exibição das comidas cadastradas pelo buffet no componente `ManageFoodsComponent`, com suporte a pesquisa e visualização de detalhes.
- **Atualização:** edição de dados e substituição de imagem quando necessário, reutilizando o mesmo `FoodFormComponent` em modo de edição.
- **Remoção/Inativação:** a remoção é realizada via *soft delete*, impedindo novas associações a reservas futuras, mantendo consistência com registros históricos, conforme regra de negócio definida no Capítulo 4.

O serviço `FoodsApiService` (`features/foods/services/food.service.ts`) encapsula toda a comunicação com a API, fornecendo métodos para listagem paginada, criação, atualização e remoção. O *upload* de imagens é realizado através de `FormData`, enviando simultaneamente os dados JSON da comida e o arquivo de imagem selecionado.

O Quadro 6.6 apresenta o método de criação de comida com *upload* de imagem.

```
1 create(request: ComidaRequest, ownerId: number,
2     imagem?: File): Observable<ComidaResponse> {
3     const params = { ownerId: String(ownerId) };
4
5     const formData = new FormData();
6
7     const comidaBlob = new Blob(
8         [JSON.stringify(request)],
9         { type: 'application/json' }
10    );
11    formData.append('comida', comidaBlob);
12
13    if (imagem) {
14        formData.append('imagem', imagem);
15    }
16
17    return this.api.post<ComidaResponse>(
18        this.basePath, formData, params
19    );
20 }
```

Quadro 6.6 Método de criação de comida com *upload* de imagem.

O componente `FoodFormComponent` utiliza formulários *template-driven* do Angular para capturar os dados, validando campos obrigatórios antes do envio. A seleção de imagem é implementada através do *plugin* `@capacitor/camera`, permitindo ao usuário escolher uma foto da galeria ou tirar uma nova foto com a câmera do dispositivo.

6.8 Gerenciamento de serviços

De forma análoga às comidas, o gerenciamento de serviços também é exclusivo do perfil buffet. As operações incluem cadastro, listagem, atualização e remoção/inativação, implementadas nos componentes `ServiceFormComponent` e `ManageServicesComponent`. O serviço de API correspondente é `ServicesApiService` (`features/services/services/services.service.ts`).

Esses serviços representam itens complementares do evento (ex.: decoração, música, fotografia etc.), conforme o domínio definido. A categoria dos serviços utiliza a mesma enumeração `EnumCategoria` das comidas, garantindo consistência no sistema.

6.9 Gerenciamento de reservas

A reserva é criada pelo cliente através do fluxo de pedido (`OrderComponent`), onde o usuário seleciona comidas e serviços de um buffet específico, montando seu carrinho. O serviço `OrderService` gerencia o estado do carrinho utilizando `BehaviorSubject` do RxJS, permitindo que múltiplos componentes observem as mudanças em tempo real.

Após montar o carrinho, o cliente prossegue para as telas de detalhes do pedido (`OrderDetailsComponent`), endereço (`OrderAddressComponent`) e confirmação (`OrderConfirmationComponent`), onde informa data, horário, quantidade de pessoas e endereço do evento. Ao confirmar, a reserva é enviada para a API através do `ReservationsApiService` com estado inicial **PENDENTE**.

O ciclo de vida da reserva no *front-end* segue as regras estabelecidas no Capítulo 4:

- **Edição e cancelamento pelo cliente:** permitidos apenas enquanto a reserva estiver em estado *PENDENTE*, através do componente `ReserveEditComponent`. Após aprovação, essas ações ficam indisponíveis ao cliente.
- **Aprovação e recusa pelo buffet:** o buffet pode aprovar a reserva informando obrigatoriamente o valor estimado através de modal de confirmação, ou recusá-la. Na aprovação, a reserva é convertida em evento pelo *back-end*.
- **Validação de disponibilidade:** a criação de reservas deve respeitar a disponibilidade do buffet, não sendo permitidas solicitações para datas e horários que conflitem com eventos já agendados no mesmo estabelecimento.

As telas de reservas (`ReservesComponent` e `ReserveDetailsComponent`) incluem listagem com filtros por estado, detalhes completos da reserva e ações condicionadas ao estado atual, garantindo que o usuário visualize apenas as operações permitidas no contexto.

O Quadro 6.7 apresenta o método de criação de reserva.

```
1 create(clienteId: number, body: ReservaRequest):  
2   Observable<ReservaResponse> {  
3     return this.api.post<ReservaResponse>(  
4       '/clientes/reservas', body, { clienteId }  
5     );  
6   }
```

Quadro 6.7 Método de criação de reserva pelo cliente.

6.10 Gerenciamento de eventos

Eventos são criados exclusivamente a partir da aprovação de uma reserva pelo buffet, iniciando com estado **AGENDADO**. O *front-end* permite acompanhamento e atualização conforme as regras definidas no Capítulo 4:

- **Cliente:** pode consultar o evento através do componente `EventsComponent` e `EventDetailsComponent`, visualizando todos os detalhes (data, horário, itens selecionados, endereço e valor estimado). Enquanto estiver *AGENDADO*, pode editar dados através do `EventEditComponent` e cancelar. Em estados *CANCELADO* ou *CONCLUIDO*, não há alterações permitidas.
- **Buffet:** pode atualizar valor estimado, cancelar e reverter cancelamento (ação exclusiva do buffet, conforme UC19 definido no Capítulo 4), além de concluir o evento ao final da prestação do serviço. Essas operações são implementadas no `EventoService`.

A apresentação do evento inclui informações originadas da reserva (data, horário, itens selecionados e endereço) e o valor estimado informado na aprovação. O componente utiliza *badges* coloridos para indicar visualmente o estado atual do evento.

O serviço `EventoService` (`features/events/services/evento.service.ts`) fornece métodos para listagem por cliente ou buffet, consulta individual, atualização de valor, conclusão e cancelamento, conforme apresentado no Quadro 6.8.

```
1 updateValor(id: number, valor: number, ownerId: number):
2     Observable<EventoResponse> {
3     const body = { valor };
4     const params = { ownerId: String(ownerId) };
5     return this.api.put<EventoResponse>(
6         `${this.basePath}/${id}/valor`, body, params
7     );
8 }
9
10 concluir(id: number, ownerId: number):
11     Observable<EventoResponse> {
12     const params = { ownerId: String(ownerId) };
13     return this.api.put<EventoResponse>(
14         `${this.basePath}/${id}/concluir`, {}, params
15     );
16 }
17
18 cancelar(id: number, ownerId: number):
19     Observable<EventoResponse> {
20     const params = { ownerId: String(ownerId) };
```



```
21 return this.api.put<EventoResponse>(  
22 `_${this.basePath}/${id}/cancelar`, {}, params  
23 );  
24 }
```

Quadro 6.8 Métodos principais do EventoService.

6.11 Notificações

O sistema registra e apresenta notificações relacionadas a alterações de estado de reservas e eventos através do componente `NotificationsComponent` (`pages/global/notifications`). Sempre que ocorre uma mudança relevante (ex.: reserva aprovada, reserva cancelada, evento cancelado, evento concluído), uma notificação é gerada pelo *back-end* e disponibilizada ao usuário destinatário.

No *front-end*, a tela de notificações apresenta listagem ordenada por data de criação (mais recentes primeiro), status de leitura (lida/não lida) indicado por badge, e detalhamento da mensagem em modal. O componente permite marcar notificações como lidas individualmente ou todas de uma vez, possibilitando que cliente e buffet acompanhem atualizações sem depender de comunicação externa.

O serviço de notificações utiliza polling periódico ou pull-to-refresh para buscar novas notificações, garantindo atualização em tempo adequado sem sobrecarregar o servidor.

6.12 Upload e gerenciamento de imagens

O sistema permite upload e atualização de imagens vinculadas a entidades do domínio, contemplando:

- **Foto de perfil de cliente** (através do componente `ProfileEditComponent`);
- **Imagens de comidas** (através do `FoodFormComponent`);
- **Imagens de serviços** (através do `ServiceFormComponent`).

No *front-end*, os componentes de formulário incluem componentes customizados de seleção de imagem (`ImagePlaceholderComponent` e `ImageCircleComponent`) que permitem:

- Seleção de imagem da galeria do dispositivo
- Captura de nova foto com a câmera

- Visualização prévia da imagem antes do envio
- Validações básicas de tipo e tamanho de arquivo

A integração com o dispositivo móvel é realizada através do plugin `@capacitor/camera`, que fornece API unificada para acesso à câmera e galeria em Android e iOS. Os arquivos são enviados ao back-end como `FormData` junto com os dados JSON da entidade, conforme demonstrado anteriormente no Quadro 6.6.

O *back-end* integra o armazenamento externo (ex.: AWS S3, Cloudinary ou sistema de arquivos local) e retorna a URL associada ao recurso. Dessa forma, a aplicação mantém desempenho adequado ao carregar imagens por URLs otimizadas, evitando armazenamento local pesado no dispositivo e permitindo cache eficiente pelo navegador.

6.13 Cliente *HTTP* centralizado

Para padronizar a comunicação com a API REST, o projeto implementa um cliente HTTP centralizado (`ApiClient`) que encapsula o `HttpClient` do Angular, fornecendo métodos tipados para as operações CRUD padrão (GET, POST, PUT, DELETE).

O `ApiClient` normaliza automaticamente os parâmetros de query string, convertendo todos os valores para string e removendo valores nulos ou indefinidos. Além disso, centraliza a URL base da API (definida em `environment.ts`), facilitando mudanças de ambiente.

```
1 @Injectable({ providedIn: 'root' })
2 export class ApiClient {
3   private http = inject(HttpClient);
4   private baseUrl = environment.API_URL;
5
6   private normalizeParams(params?: Record<string, any>):
7     Record<string, string> | undefined {
8     if (!params) return undefined;
9     const normalized: Record<string, string> = {};
10    for (const [key, value] of Object.entries(params)) {
11      if (value !== undefined && value !== null) {
12        normalized[key] = String(value);
13      }
14    }
15    return normalized;
16  }
17
18  get<T>(path: string, params?: Record<string, any>):
19    Observable<T> {
20    const normalized = this.normalizeParams(params);
21    return this.http.get<T>(
```

```

22     `${this.baseUrl}${path}`,
23     { params: normalized }
24   );
25 }
26
27 post<T>(path: string, body?: any,
28     params?: Record<string, any>): Observable<T> {
29   const normalized = this.normalizeParams(params);
30   return this.http.post<T>(
31     `${this.baseUrl}${path}`, body,
32     { params: normalized }
33   );
34 }
35
36 put<T>(path: string, body?: any,
37     params?: Record<string, any>): Observable<T> {
38   const normalized = this.normalizeParams(params);
39   return this.http.put<T>(
40     `${this.baseUrl}${path}`, body,
41     { params: normalized }
42   );
43 }
44
45 delete<T>(path: string, params?: Record<string, any>):
46     Observable<T> {
47   const normalized = this.normalizeParams(params);
48   return this.http.delete<T>(
49     `${this.baseUrl}${path}`,
50     { params: normalized }
51   );
52 }
53 }

```

Quadro 6.9 Cliente HTTP centralizado ApiClient.

Todos os serviços de API das *features* (AuthService, FoodsApiService, ReservationsApiService, EventoService, etc.) injetam e utilizam o ApiClient, garantindo consistência na comunicação com o *back-end*.

6.14 Componentes reutilizáveis

Para acelerar o desenvolvimento e garantir consistência visual, o projeto implementa uma biblioteca interna de componentes reutilizáveis organizados em `shared/ui/templates`. Esses componentes são *standalone* e podem ser importados diretamente por qualquer página ou *feature*. Os principais grupos incluem:

- **Pages:** componentes de *layout* de página (ModelPageComponent, FormPageCompo-

nent) que definem estrutura padrão com *header*, conteúdo e *footer*.

- **Navbars:** barras de navegação customizadas para cliente (`ClientNavbarComponent`) com navegação por *tabs*.
- **Buttons:** botões estilizados (`PrimaryButtonComponent`, `OutlineButtonComponent`, `SecondaryButtonComponent`) que seguem o *design system* do projeto.
- **Inputs:** campos de formulário customizados (`TextInputComponent`, `PasswordInputComponent`, `TextareaInputComponent`, `SelectedInputComponent`) com validação integrada e estados visuais.
- **Cards:** componentes de *card* para exibição de itens em listas (`FoodCardComponent`, `ServiceCardComponent`, `OrderItemCardComponent`, `ReserveCardComponent`, `EventCardComponent`).
- **Modals:** modais reutilizáveis (`ConfirmationModalComponent`, `SelectModalComponent`) para confirmações e seleções.
- **Badges:** indicadores visuais de estado (`StatusBadgeComponent`) com cores configuráveis.
- **Placeholders:** componentes de *placeholder* de imagem (`ImagePlaceholderComponent`, `ImageCircleComponent`) com suporte a *upload*.
- **Spinners:** indicadores de carregamento (`LoadingSpinnerComponent`) para *feedback* visual em operações assíncronas.

Todos esses componentes recebem configurações via `@Input()` (ex.: cores, tamanhos, textos) e emitem eventos via `@Output()` (ex.: cliques, mudanças de valor), permitindo alta flexibilidade e reutilização em diferentes contextos.

6.15 Arquitetura *Multi-Tenant*

O sistema implementa uma arquitetura *multi-tenant* que permite que múltiplos buffets utilizem a mesma aplicação com personalização visual individualizada e isolamento completo de dados. Essa arquitetura garante que cada estabelecimento tenha identidade visual própria (cores,

logotipo e *banner*) e acesse exclusivamente suas informações (comidas, serviços, reservas e eventos), sem interferência ou visualização de dados de outros buffets.

A implementação do *multi-tenancy* no *front-end* é composta por três pilares fundamentais: organização de recursos estáticos por *tenant*, personalização dinâmica da interface e isolamento de dados via *interceptors* HTTP.

6.15.1 Organização de recursos estáticos

O diretório `assets/buffets` armazena recursos específicos de cada buffet, organizados em subdiretórios numerados de acordo com o identificador único do estabelecimento. Cada subdiretório contém os seguintes arquivos essenciais para personalização visual:

- `theme.json`: arquivo de configuração que define as cores primária, secundária e de destaque (*accent*), além dos caminhos para logotipo e *banner*. Este arquivo também armazena o identificador do buffet (`buffetId`) e nome do estabelecimento (`nameBuffet`).
- `logo.png`: imagem do logotipo do buffet, utilizada em cabeçalhos, telas de apresentação e áreas de identificação visual.
- `banner.png`: imagem de *banner* utilizada em telas específicas como *home*, proporcionando identidade visual diferenciada para cada estabelecimento.

A estrutura de diretórios segue o padrão apresentado no Quadro 6.10.

```
1 assets/  
2   buffets/  
3     0/  
4       theme.json  
5       logo.png  
6       banner.png  
7     1/  
8       theme.json  
9       logo.png  
10      banner.png
```

Quadro 6.10 Estrutura de diretórios de recursos por buffet.

O arquivo `theme.json` segue o formato apresentado no Quadro 6.11.

```
1 {  
2   "primaryColor": "#9747FF",  
3   "secondaryColor": "#48008C",  
4   "accentColor": "#FFFFFF",  
5   "logo": "assets/buffets/0/logo.png",
```

```
6  "banner": "assets/buffets/0/banner.png",  
7  "buffetId": 0,  
8  "nameBuffet": "eBuffet"  
9  }
```

Quadro 6.11 Estrutura do arquivo theme.json de configuração de tema.

6.15.2 Carregamento e aplicação dinâmica de tema

O serviço ThemeService (core/services/theme.service.ts) é responsável por carregar o tema correspondente ao buffet configurado e aplicar as personalizações à interface. Este serviço é inicializado antes da renderização da aplicação através do mecanismo APP_INITIALIZER do Angular, garantindo que a interface seja apresentada com o tema correto desde o primeiro carregamento.

O processo de carregamento de tema segue as seguintes etapas:

1. O serviço consulta o identificador do buffet configurado no arquivo environments/environment.ts através da propriedade buffetId.
2. Com base neste identificador, constrói o caminho para o arquivo theme.json correspondente (ex.: assets/buffets/1/theme.json).
3. Realiza requisição HTTP para carregar o arquivo de configuração do tema.
4. Valida a presença das propriedades obrigatórias (primaryColor, secondaryColor, accentColor).
5. Armazena a configuração carregada e atualiza os BehaviorSubjects que permitem componentes observarem mudanças nas cores e recursos visuais.
6. Aplica as cores dinamicamente às variáveis CSS do Ionic através de setProperty no document.documentElement, incluindo variações de *shade*, *tint*, contraste e valores RGB.
7. Remove a classe theme-loading do body, permitindo a exibição da interface.
8. Em caso de falha no carregamento, aplica tema padrão (*fallback*) com cores pré-definidas para evitar quebra da aplicação.

O Quadro 6.12 apresenta o método principal de carregamento de tema.

```

1 async loadTheme(): Promise<void> {
2   try {
3     const themePath = `assets/buffets/${environment.buffetId}/theme.
      json`;
4
5     const theme = await this.http.get<ThemeConfig>(themePath).
      toPromise();
6
7     if (!theme) {
8       throw new Error('Theme data is empty or invalid');
9     }
10
11    if (!theme.primaryColor || !theme.secondaryColor ||
12        !theme.accentColor) {
13      throw new Error('Theme is missing required color properties');
14    }
15
16    this.currentTheme = theme;
17    this.updateColorSubjects(theme);
18    this.buffetIdSubject.next(theme.buffetId ?? null);
19    this.applyTheme(theme);
20
21    document.body.classList.remove('theme-loading');
22  } catch (error) {
23    console.error('Failed to load theme:', error);
24    this.applyFallbackTheme();
25    document.body.classList.remove('theme-loading');
26  }
27 }
28
29 private applyTheme(theme: ThemeConfig): void {
30   const root = document.documentElement;
31
32   root.style.setProperty('--ion-color-primary', theme.primaryColor);
33   root.style.setProperty('--ion-color-primary-rgb',
34     this.hexToRgb(theme.primaryColor));
35   root.style.setProperty('--ion-color-primary-contrast',
36     this.getContrastColor(theme.primaryColor));
37
38   root.style.setProperty('--ion-color-secondary', theme.
39     secondaryColor);
40   root.style.setProperty('--ion-color-secondary-rgb',
41     this.hexToRgb(theme.secondaryColor));
42
43   root.style.setProperty('--ion-color-accent', theme.accentColor);
44
45   root.style.setProperty('--theme-logo', `url('${theme.logo}')`);
46   root.style.setProperty('--theme-banner', `url('${theme.banner}')`);
47 }

```

Quadro 6.12 Método de carregamento de tema no ThemeService.

O serviço também fornece *observables* públicos que permitem que componentes reajam dinamicamente a mudanças de tema:

- `primaryColor$`: cor primária do tema
- `secondaryColor$`: cor secundária do tema
- `accentColor$`: cor de destaque
- `logo$`: caminho do logotipo
- `banner$`: caminho do *banner*
- `buffetId$`: identificador do buffet

Componentes podem se inscrever nesses *observables* para atualizar sua apresentação conforme o tema carregado, garantindo consistência visual em toda a aplicação.

6.15.3 Isolamento de dados por *tenant*

Para garantir que cada buffet acesse exclusivamente seus próprios dados, o sistema implementa *interceptors* HTTP que injetam automaticamente cabeçalhos de identificação em todas as requisições à API. O principal *interceptor* responsável por esse isolamento é o `buffetIdInterceptor` (`core/interceptors/buffet-id.interceptor.ts`).

Este *interceptor* funciona da seguinte forma:

1. Injeta o `ThemeService` para obter o identificador do buffet atualmente configurado.
2. Verifica se existe um `buffetId` válido carregado do tema.
3. Clona a requisição HTTP original e adiciona o cabeçalho `X-Buffer-Id` com o identificador do buffet.
4. Passa a requisição modificada para o próximo *handler* da cadeia de *interceptors*.

O Quadro 6.13 apresenta a implementação completa do *interceptor*.

```
1 import { HttpInterceptorFn } from '@angular/common/http';
2 import { inject } from '@angular/core';
3 import { ThemeService } from "../services/theme.service";
4
5 export const buffetIdInterceptor: HttpInterceptorFn = (req, next) =>
  {
```



```
6  const themeService = inject(ThemeService);
7  const buffetId = themeService.getBuffetIdSync?();
8
9  if (!buffetId) return next(req);
10
11  return next(req.clone({
12    setHeaders: { 'X-Buffer-Id': String(buffetId) }
13  }));
14  });
```

Quadro 6.13 *Interceptor* para injeção automática do identificador do buffet.

O *back-end*, ao receber requisições com o cabeçalho `X-Buffer-Id`, aplica filtros automáticos nas consultas ao banco de dados para retornar exclusivamente os dados pertencentes ao buffet especificado. Esse mecanismo, detalhado no Capítulo 5, garante isolamento completo de dados mesmo em nível de aplicação, complementando o isolamento implementado no nível de banco de dados.

Adicionalmente, o sistema utiliza *interceptors* complementares para injeção de identificadores de usuário (`user-id.interceptor.ts`), garantindo que operações específicas do cliente (ex.: criação de reservas, consulta de perfil) sejam sempre associadas ao usuário autenticado.

6.15.4 Configuração de ambiente

A seleção do buffet a ser utilizado pela aplicação é realizada através do arquivo `environments/environment.ts`, que define variáveis globais do ambiente de execução. O Quadro 6.14 apresenta a estrutura deste arquivo.

```
1  export const environment = {
2    production: true,
3    buffetId: 1,
4    API_URL: 'http://localhost:8080/api',
5    TENANT_HEADER: 'X-Buffer-Id'
6  };
```

Quadro 6.14 Arquivo de configuração de ambiente.

A propriedade `buffetId` determina qual conjunto de recursos visuais será carregado do diretório `assets/buffets` e qual identificador será utilizado nos cabeçalhos HTTP. Essa abordagem permite que múltiplas instâncias da aplicação sejam compiladas e distribuídas, cada uma configurada para um buffet específico, mantendo isolamento completo entre estabelecimentos.

Para ambientes de desenvolvimento, é possível criar variações do arquivo de ambiente (ex.: `environment.development.ts`) com configurações específicas, incluindo URLs de API de homologação e identificadores de buffets de teste.

6.15.5 Vantagens da arquitetura *multi-tenant*

A implementação da arquitetura *multi-tenant* no *front-end* proporciona os seguintes benefícios ao sistema:

- **Identidade visual personalizada:** cada buffet possui interface com cores, logotipo e *banner* próprios, reforçando a marca do estabelecimento junto aos clientes.
- **Isolamento de dados:** garantia de que cada buffet visualize e manipule exclusivamente seus próprios dados, atendendo requisitos de privacidade e segurança.
- **Reutilização de código:** uma única base de código atende múltiplos estabelecimentos, reduzindo custos de desenvolvimento e manutenção.
- **Facilidade de expansão:** adição de novos buffets realizada simplesmente criando novo diretório em `assets/buffets` com os recursos correspondentes, sem necessidade de modificação do código-fonte.
- **Manutenção centralizada:** correções de *bugs* e implementação de novas funcionalidades beneficiam automaticamente todos os buffets cadastrados.
- **Consistência de experiência:** apesar da personalização visual, a estrutura de navegação e funcionalidades permanecem consistentes, facilitando treinamento e suporte.

Essa arquitetura posiciona o sistema como solução escalável capaz de atender múltiplos estabelecimentos simultaneamente, mantendo qualidade e personalização adequadas às necessidades de cada buffet.

Neste capítulo, foi apresentada a implementação do *front-end* do sistema, construído com Angular, Ionic e Capacitor, detalhando a configuração inicial, a organização de pastas baseada em *Feature-Sliced Design*, os principais componentes e serviços desenvolvidos, além da estratégia de personalização visual multi-buffet. A arquitetura adotada promove modularidade, reutilização de código e facilidade de manutenção, permitindo que múltiplos estabelecimentos operem sobre a mesma base com identidade visual própria. Com as camadas de *back-end* e *front-end* implementadas, o próximo capítulo apresenta o resultado final da aplicação por meio de suas interfaces, demonstrando as funcionalidades em operação para os perfis de cliente e buffet.

Nos capítulos anteriores foram apresentados os fundamentos teóricos, a análise de requisitos, a implementação do *back-end* e a implementação do *front-end* do sistema. Neste capítulo, apresenta-se o resultado final do desenvolvimento por meio das interfaces do usuário, demonstrando as funcionalidades implementadas desde a tela inicial até o gerenciamento completo de reservas e eventos. As telas são organizadas conforme os perfis de acesso (público, cliente e buffet), evidenciando a aplicação prática dos requisitos e regras de negócio definidos no Capítulo 4 e implementados nos Capítulos 5 e 6.

O sistema foi desenvolvido utilizando Angular com Ionic, proporcionando uma experiência nativa em dispositivos móveis através do *framework* Capacitor, que permite acesso a recursos do dispositivo como câmera, armazenamento local e notificações. A arquitetura *Feature-Sliced Design* (FSD) adotada no *front-end* facilita a manutenção e escalabilidade, separando responsabilidades em camadas bem definidas: páginas (*pages*), funcionalidades de negócio (*features*), componentes compartilhados (*shared*) e infraestrutura transversal (*core*).

A personalização visual de cada buffet, implementada através da arquitetura *multi-tenant* detalhada no Capítulo 6, garante que cores, logotipos e *banners* específicos sejam carregados dinamicamente conforme o estabelecimento configurado, proporcionando identidade visual única para cada buffet que utiliza o sistema.

7.1 Área Pública

As telas apresentadas nesta seção são acessíveis a todos os usuários, estando eles autenticados ou não. Compreendem o fluxo de entrada no sistema, desde a tela inicial até a autenticação e cadastro de novos usuários.

7.1.1 Tela Inicial (*Index*)

A Figura 7.1 apresenta a tela inicial do sistema, exibida automaticamente durante o carregamento do aplicativo. Esta interface apresenta o logotipo do buffet com fundo colorido seguindo a identidade visual configurada no tema do estabelecimento, proporcionando uma experiência visual personalizada ao usuário enquanto o sistema verifica o estado de autenticação e carrega os recursos necessários para a sessão. Caso o usuário já esteja autenticado, ocorre redirecionamento automático para a área apropriada (cliente ou buffet); caso contrário, o sistema direciona para a tela de boas-vindas.



Figura 7.1: Tela de abertura do aplicativo.

7.1.2 Tela de Boas-Vindas

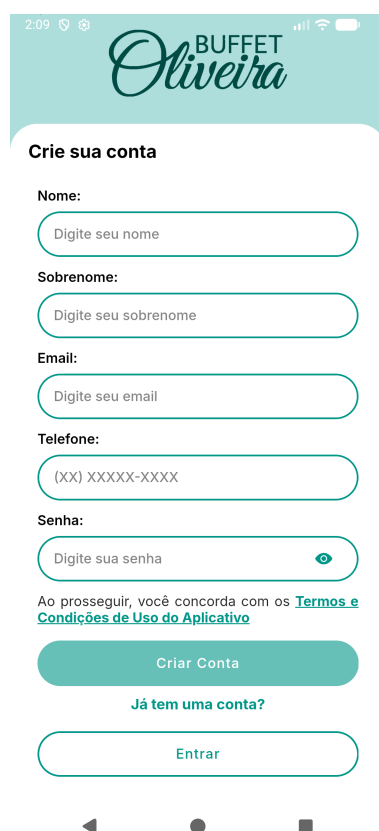
A Figura 7.2 apresenta a tela de boas-vindas, exibida aos usuários não autenticados. Esta interface apresenta o logotipo do buffet, uma mensagem de boas-vindas e dois botões principais: “Criar Conta”, que permite a novos usuários registrarem-se no sistema, e “Entrar”, destinado a usuários previamente cadastrados. A interface foi projetada seguindo princípios de usabilidade, tornando-a intuitiva e acolhedora, facilitando o primeiro contato do usuário com o aplicativo.



Figura 7.2: Tela de boas-vindas do aplicativo.

7.1.3 Tela de Cadastro

A Figura 7.3 apresenta o formulário de cadastro de novos clientes. Devem ser preenchidas informações básicas como nome, sobrenome, e-mail, telefone e senha. O campo de telefone possui formatação automática no padrão brasileiro (XX) XXXXX-XXXX. Ao final do formulário, encontra-se um link para os Termos e Condições de Uso do Aplicativo, que devem ser aceitos implicitamente ao criar a conta. O botão “Criar Conta” é habilitado somente quando todos os campos obrigatórios são preenchidos corretamente. Usuários já cadastrados podem acessar o botão “Entrar” para serem redirecionados ao login.



A imagem mostra a interface de usuário para a criação de uma nova conta no aplicativo "Buffet Oliveira". No topo, há uma barra de status com o horário 2:09 e ícones de conexão. Abaixo, o cabeçalho do aplicativo exibe o nome "Buffet Oliveira" em uma fonte cursiva verde. O formulário, intitulado "Crie sua conta", contém os seguintes campos: "Nome:" com o placeholder "Digite seu nome"; "Sobrenome:" com o placeholder "Digite seu sobrenome"; "Email:" com o placeholder "Digite seu email"; "Telefone:" com o placeholder "(XX) XXXXX-XXXX"; e "Senha:" com o placeholder "Digite sua senha" e um ícone de olho para alternar a visibilidade. Abaixo dos campos, há um texto informando: "Ao prosseguir, você concorda com os [Termos e Condições de Uso do Aplicativo](#)". No final, há dois botões: "Criar Conta" em um botão verde arredondado e "Já tem uma conta?" seguido por um botão "Entrar" em um botão branco arredondado com contorno verde. Na base da tela, os ícones de navegação padrão do Android são visíveis.

Figura 7.3: Tela de cadastro de novo cliente.

A Figura 7.4 apresenta a tela de confirmação de cadastro, exibida após o usuário criar sua conta com sucesso. Esta tela informa que o cadastro foi realizado e instrui o usuário a fazer login para acessar o sistema. O botão “Fazer Login” redireciona o usuário diretamente para a tela de login.

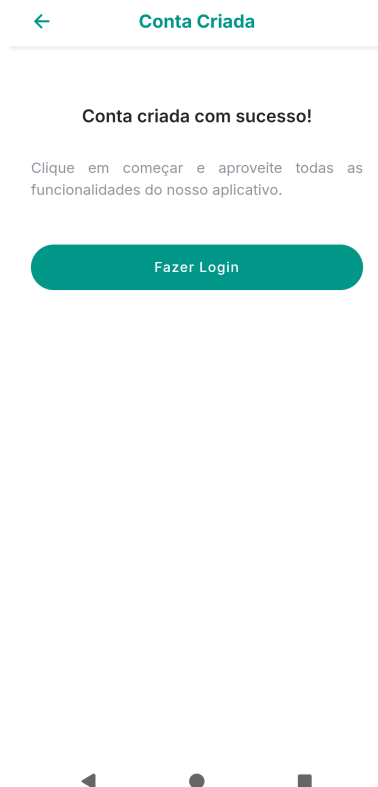


Figura 7.4: Tela de confirmação de cadastro.

A Figura 7.5 apresenta a tela de Termos e Condições de Uso do Aplicativo. O usuário pode acessar esta tela clicando no link disponível durante o cadastro. Os termos descrevem as regras de uso do sistema, políticas de privacidade e responsabilidades. Após a leitura, o usuário pode fechar a tela e retornar ao formulário de cadastro.



Figura 7.5: Tela de Termos e Condições de Uso.

7.1.4 Tela de *Login*

A Figura 7.6 apresenta a tela de autenticação do sistema, onde devem ser inseridos e-mail e senha para acesso ao aplicativo. O formulário possui validação em tempo real, verificando o formato do endereço eletrônico e o tamanho mínimo da senha (6 caracteres). Encontra-se disponível o *link* “Esqueceu sua senha?”, que redireciona o usuário para o fluxo de recuperação de credenciais. O botão “Entrar” é habilitado somente quando as credenciais são preenchidas corretamente. Usuários sem cadastro prévio podem acessar o botão “Criar Conta” para serem redirecionados ao formulário de registro.



A imagem mostra a interface de login de um aplicativo. No topo, há uma barra de status com o horário 2:08 e ícones de conexão. Abaixo, um cabeçalho com uma seta de retorno e o logo "BUFFET Oliveira". O formulário "Login" contém campos para "Email:" e "Senha:". O campo de e-mail tem o placeholder "Digite seu email". O campo de senha tem o placeholder "Digite sua senha" e um ícone de olho para alternar a visibilidade. Abaixo dos campos, há um link "Esqueceu sua senha?". Um botão verde "Entrar" está centralizado. Abaixo dele, o texto "Não possui uma conta?" precede um botão "Criar Conta". Na base da tela, há ícones de navegação padrão de Android.

Figura 7.6: Tela de login.

A Figura 7.7 mostra a mensagem de erro exibida quando o usuário fornece credenciais inválidas. A mensagem “E-mail ou senha incorretos. Tente novamente.” aparece como notificação *toast* na parte superior da tela, orientando o usuário a corrigir as informações.



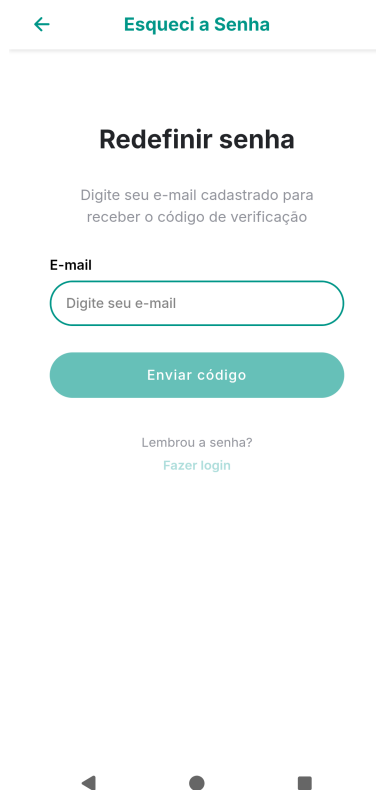
Figura 7.7: Mensagem de erro de login inválido.

7.1.5 Recuperação de Senha

O sistema implementa um fluxo completo de recuperação de senha em três etapas, garantindo segurança e usabilidade no processo de redefinição de credenciais.

Solicitação de Recuperação

A Figura 7.8 apresenta o primeiro passo do processo de recuperação de senha. O usuário deve informar o endereço eletrônico cadastrado no sistema. Ao acionar o botão “Enviar Código”, o sistema valida o formato do e-mail e, sendo válido, envia um código de verificação de 6 dígitos para a caixa de entrada do usuário. Uma mensagem de sucesso é exibida informando que o código foi enviado. Encontra-se disponível também o *link* “Voltar ao Login”, que permite retornar à tela de autenticação a qualquer momento.

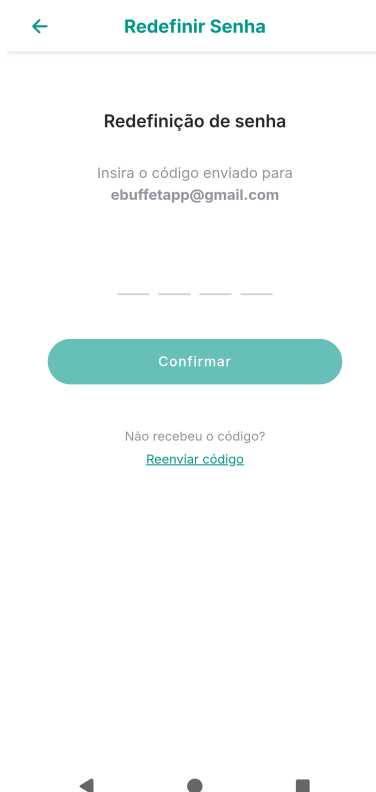


A interface de usuário para a recuperação de senha. No topo, há um link de navegação "Esqueci a Senha" com uma seta para trás. O título principal é "Redefinir senha". Abaixo dele, uma instrução pede ao usuário para digitar seu e-mail cadastrado para receber o código de verificação. Há um campo de entrada rotulado "E-mail" com o placeholder "Digite seu e-mail". Abaixo do campo, há um botão verde "Enviar código". Na base, há um link "Fazer login" sob o texto "Lembrou a senha?". Na parte inferior da tela, há uma barra de navegação com três ícones: uma seta para trás, um círculo e um quadrado.

Figura 7.8: Tela de solicitação de recuperação de senha.

Validação do Código

A Figura 7.9 apresenta a segunda etapa do processo, onde o usuário deve inserir o código de 4 dígitos recebido por e-mail. O campo de entrada possui formatação específica para códigos numéricos e validação em tempo real. O endereço eletrônico para o qual o código foi enviado é exibido na tela para confirmação. Ao inserir o código correto e acionar “Confirmar”, o sistema valida a autenticidade do código e redireciona para a tela de redefinição de senha. Caso o código esteja incorreto ou expirado, uma mensagem de erro é exibida solicitando novo código.



← Redefinir Senha

Redefinição de senha

Insira o código enviado para
ebuffetapp@gmail.com

— — — —

Confirmar

Não recebeu o código?
[Reenviar código](#)

◀ ● ■

Figura 7.9: Tela de validação do código de recuperação.

Redefinição de Senha

A Figura 7.10 apresenta a etapa final do processo de recuperação. Devem ser informadas a nova senha e sua confirmação. O sistema valida a correspondência entre as senhas e verifica se atendem aos requisitos mínimos de segurança (mínimo de 6 caracteres). Os campos de senha possuem ícone de visualização que permite alternar entre texto visível e oculto. Ao acionar “Redefinir Senha”, a alteração é processada no servidor e, em caso de sucesso, uma mensagem de confirmação é exibida e ocorre redirecionamento automático para a tela de autenticação, permitindo que o usuário faça login com a nova senha.



A interface de redefinição de senha é apresentada em um layout limpo. No topo, há uma barra de navegação com uma seta verde para voltar e o título "Nova Senha" em verde. Abaixo, o título "Nova senha" é centralizado. Segue uma instrução: "Defina e confirme a sua nova senha para utilizar nosso aplicativo!". Há dois campos de entrada com bordas arredondadas e ícones de olho para alternar a visibilidade: "Digite uma nova senha" e "Confirme a senha". Abaixo dos campos, um botão verde arredondado com o texto "Confirmar" está centralizado. Na base da tela, há uma barra de status com três ícones: uma seta para voltar, um círculo no meio e um quadrado para avançar.

Figura 7.10: Tela de redefinição de senha.

7.2 Área do Cliente

7.2.1 Home do Cliente

A Figura 7.11 apresenta a tela inicial do cliente após autenticação no sistema. O cabeçalho exibe mensagem de boas-vindas personalizada com o nome do usuário e ícone de notificações. A interface contém três seções principais:

- **Carrossel de destaques:** apresenta *banners* promocionais e destaques do buffet, utilizando o *banner* configurado no tema do estabelecimento;
- **Calendário:** exibe as datas de eventos e reservas do cliente, permitindo visualização rápida dos dias disponíveis para serem realizadas reservas;
- **Categorias:** oferece acesso rápido às principais categorias de comidas através de *cards* visuais.

Na parte inferior da tela, encontra-se uma barra de navegação (*tab bar*) com quatro ícones: *Home*, Comidas, Serviços e Perfil, permitindo navegação rápida entre as principais áreas do aplicativo destinadas ao cliente.

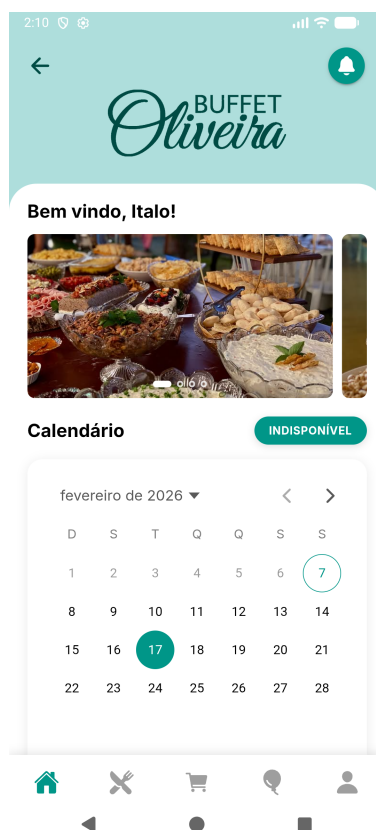


Figura 7.11: Tela inicial do cliente.

7.2.2 Catálogo de Comidas

A Figura 7.12 apresenta o catálogo de comidas disponíveis no buffet. A interface possui campo de pesquisa no topo, permitindo busca de comidas pelo nome. Logo abaixo, encontram-se filtros por categoria (Almoço, Jantar, Casamento, Aniversário, Formatura, entre outras), facilitando a navegação. As comidas são exibidas em formato de *cards* com imagem, nome, descrição resumida e categoria. Ao selecionar um *card*, ocorre redirecionamento para a tela de detalhes da comida. A listagem implementa *scroll* infinito, carregando mais itens conforme a rolagem da página.



Figura 7.12: Catálogo de comidas disponíveis.

A Figura 7.13 apresenta a tela de detalhes de uma comida específica. Podem ser visualizados a imagem em destaque, nome completo, descrição detalhada e categoria da comida. Na parte inferior, encontra-se o botão “Adicionar à Reserva”, que permite incluir o item no carrinho de pedido. Encontra-se disponível também um botão de retorno no cabeçalho para voltar ao catálogo.



Figura 7.13: Detalhes de uma comida.

7.2.3 Catálogo de Serviços

A Figura 7.14 apresenta o catálogo de serviços complementares oferecidos pelo buffet (decoração, música, fotografia, entre outros). A estrutura é idêntica à tela de comidas, apresentando campo de pesquisa, filtros por categoria e *cards* informativos. Ao selecionar um *card*, o usuário é redirecionado para a tela de detalhes do serviço. A listagem também implementa *scroll* infinito para carregamento dinâmico de mais itens.

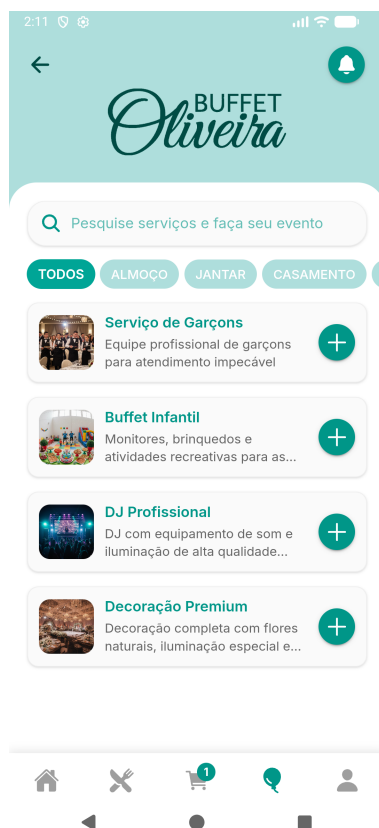


Figura 7.14: Catálogo de serviços disponíveis.

A Figura 7.15 apresenta a tela de detalhes de um serviço específico, com layout similar à tela de detalhes de comida. O usuário visualiza imagem, nome, descrição e categoria, e pode adicionar o serviço ao carrinho de pedido.



Figura 7.15: Detalhes de um serviço.

7.2.4 Carrinho de Pedido (Nova Reserva)

O processo de criação de uma nova reserva é dividido em três etapas sequenciais e uma tela de confirmação da reserva, garantindo que todas as informações necessárias sejam coletadas de forma organizada e validada.

Seleção de Itens (Carrinho)

A Figura 7.16 apresenta a tela de carrinho de pedido, primeira etapa do processo, onde são visualizados todos os itens (comidas e serviços) selecionados previamente nas telas de catálogo. A interface está dividida em duas seções principais:

- **Comidas:** lista todas as comidas adicionadas ao carrinho, exibindo imagem, nome, descrição, categoria e botão de remoção individual;
- **Serviços:** lista todos os serviços adicionados da mesma forma que as comidas.

Ao tocar em um item, ocorre redirecionamento para sua tela de detalhes, permitindo revisão das informações. O botão “Continuar” na parte inferior, destacado nas cores do tema do buffet, conduz à próxima etapa do processo. Também a opção de remover o item do carrinho clicando no ícone de lixeira do respectivo item.

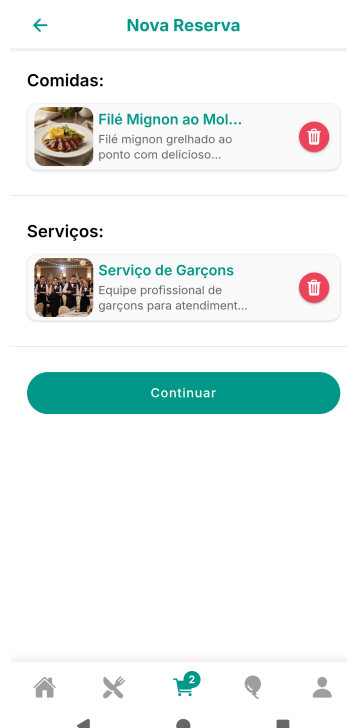


Figura 7.16: Carrinho de pedido com itens selecionados.

Detalhes da Reserva

A Figura 7.17 apresenta a tela de detalhes da reserva, segunda etapa do processo de criação. Devem ser informados os seguintes dados obrigatórios:

- **Nome da reserva:** título identificador do evento;
- **Descrição:** informações adicionais sobre o evento;
- **Data do evento:** selecionada através de calendário modal que exibe datas indisponíveis (baseadas em eventos já agendados no buffet);
- **Horário desejado:** horário de início do evento;
- **Quantidade de pessoas:** número estimado de convidados.

O campo de data possui integração com o sistema de disponibilidade do buffet e destacando datas que já possuem eventos agendados, impedindo que o cliente solicite reserva para datas indisponíveis. Todos os campos são validados antes da progressão. O botão “Continuar” permanece desabilitado até que todos os campos obrigatórios sejam preenchidos corretamente, conduzindo então à etapa de endereço.

← Detalhes da Reserva

Nome da reserva: 0/20
Digite o nome de sua reserva

Descrição da reserva: 0/255
Digite sobre qual intuito da reserva

Quantidade de pessoas:
Digite a quantidade de pessoas

Horário desejado:

Data desejada:
Escolha a data desejada

Continuar

Figura 7.17: Tela de detalhes da reserva (nome, descrição, data, horário e quantidade de pessoas).

Endereço do Evento

A Figura 7.18 apresenta o formulário de endereço do evento, terceira etapa do processo. Devem ser preenchidos os seguintes campos:

- **CEP:** código postal do local do evento;
- **Logradouro:** nome da rua, avenida ou praça;
- **Número:** número do imóvel;
- **Complemento:** informações adicionais como apartamento, bloco (opcional);
- **Bairro:** nome do bairro;
- **Cidade:** nome da cidade;
- **Estado:** seleção através de modal com lista de estados brasileiros (UF).

O campo de estado abre um modal de seleção contendo todos os estados do Brasil, facilitando a escolha e evitando erros de digitação. Todos os campos, exceto complemento, são obrigatórios. O sistema valida o preenchimento antes de habilitar o botão “Continuar”, que conduz à tela final de confirmação da reserva.

The image shows a mobile application interface for entering an event address. At the top, there is a back arrow and the title "Endereço da Reserva". Below this, there are seven input fields, each with a label and a placeholder text:

- Rua:** Digite a rua
- Número:** Digite o número de residência
- Complemento:** Digite o complemento
- Bairro:** Digite o bairro
- Cidade:** Digite a cidade
- Estado:** Selecione o estado (with a dropdown arrow icon)
- CEP:** 00000-000

At the bottom of the form is a large teal button labeled "Fazer Reserva". Below the button are three small black icons representing mobile navigation: a back arrow, a home circle, and a recent apps square.

Figura 7.18: Tela de endereço do evento.

Reserva Confirmada

A Figura 7.19 apresenta a tela de confirmação de reserva, exibida após a conclusão das três etapas do processo de criação (seleção de itens, detalhes da reserva e endereço do evento). A interface confirma visualmente que a solicitação foi enviada com sucesso ao buffet.

A tela utiliza as cores personalizadas do tema do buffet, apresentando no centro um ícone de confirmação circular em tamanho grande. O título “Reserva realizada!” é seguido pela mensagem: “Aguarde ao buffet autorizar e confirmar o agendamento de sua reserva para o evento”, informando que a reserva foi criada com estado **PENDENTE** e aguarda aprovação.

Na parte inferior, encontram-se três botões de ação:

- **Minhas Reservas:** redireciona para a listagem de reservas, permitindo acompanhamento do estado da solicitação;
- **Entrar em contato:** abre conversa no WhatsApp com o buffet, enviando mensagem pré-formatada para facilitar a comunicação;
- **Voltar:** retorna à tela inicial do cliente.

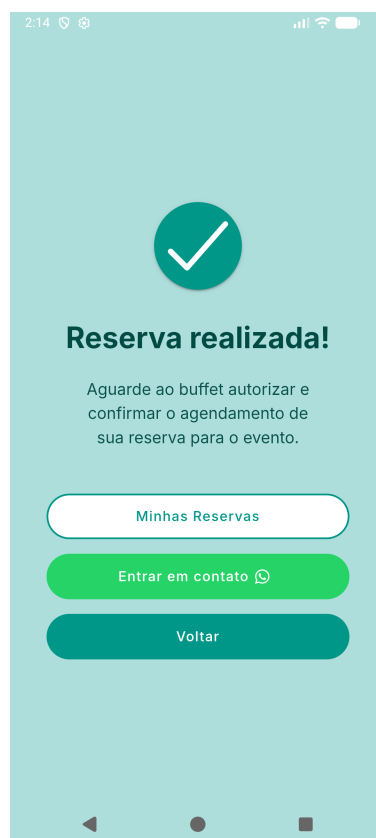


Figura 7.19: Tela de confirmação de reserva realizada.

7.2.5 Perfil do Cliente

A Figura 7.20 apresenta a tela de perfil do cliente, onde são exibidas suas informações pessoais:

- Foto de perfil (com opção de alterar);
- Nome completo;
- E-mail;
- Telefone.

Encontram-se disponíveis dois botões principais: “Editar Perfil”, que redireciona para a tela de edição, e “Sair”, que realiza *logout* do sistema e retorna à tela de boas-vindas.

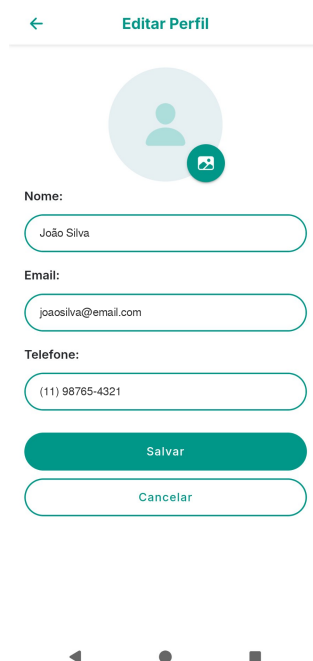


Figura 7.20: Tela de perfil do cliente.

A Figura 7.21 apresenta o formulário de edição de perfil. Podem ser atualizados os seguintes dados:

- Foto de perfil;
- Nome;
- Sobrenome;
- Telefone;

O campo de endereço eletrônico não é editável por questões de segurança. O botão “Salvar” processa as alterações e, em caso de sucesso, exibe mensagem de confirmação e retorna à tela de perfil.



A interface de usuário para a edição de perfil do cliente. No topo, há uma barra de navegação com uma seta verde para voltar e o título "Editar Perfil". Abaixo, há um círculo cinza para a foto de perfil com um ícone de câmera verde. Seguem-se os campos de texto: "Nome:" com o valor "João Silva", "Email:" com o valor "joaosilva@email.com", e "Telefone:" com o valor "(11) 98765-4321". No final, há dois botões: "Salvar" em verde e "Cancelar" em cinza. Na base da tela, há ícones de navegação padrão de um aplicativo móvel.

Figura 7.21: Tela de edição de perfil do cliente.

7.3 Área Global (Cliente e Buffet)

7.3.1 Listagem de Reservas

A Figura 7.22 apresenta a tela de listagem de reservas, acessível tanto por clientes quanto por buffets (com visualizações diferentes conforme o perfil). A interface possui:

- Campo de pesquisa para filtrar reservas por nome/título;
- Filtros por estado (Todas, Pendente, Aprovada, Recusada, Cancelada);
- Lista de cards de reservas com título, descrição resumida e badge de estado.

Ao selecionar um card, ocorre redirecionamento para a tela de detalhes da reserva. O ícone de seta no canto do card também permite acesso rápido aos detalhes.

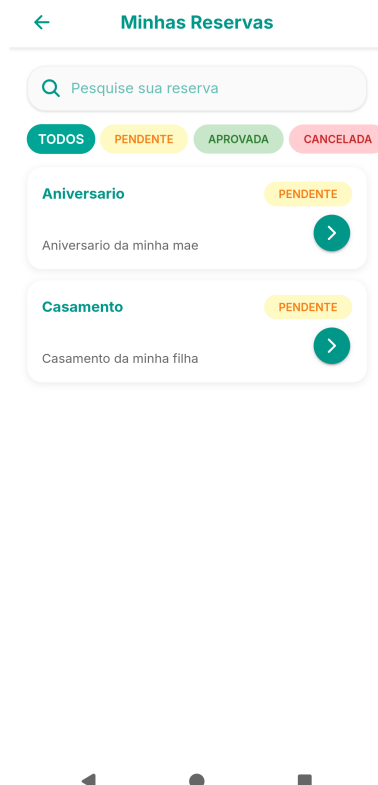


Figura 7.22: Tela de listagem de reservas.

Detalhes da Reserva

A Figura 7.23 apresenta a tela de detalhes de uma reserva específica, acessível tanto por clientes quanto pelo buffet. A interface exibe o estado atual da reserva através de *badge* colorido (Pendente, Aprovada ou Cancelada), seguido pelas seguintes informações organizadas em seções:

- **Cliente:** nome completo, e-mail e telefone formatado (visível apenas para o buffet);
- **Data:** data do evento no formato brasileiro (DD/MM/AAAA);
- **Descrição:** informações detalhadas sobre o evento;
- **Endereço:** localização completa formatada (rua, número, bairro, cidade, estado, CEP e complemento);
- **Horário:** hora de início do evento;
- **Quantidade de pessoas:** número estimado de convidados;
- **Comidas:** lista de itens alimentícios selecionados com imagem, nome e descrição;
- **Serviços:** lista de serviços contratados com imagem, nome e descrição.

Os itens de comida e serviço são clicáveis, redirecionando para suas respectivas telas de detalhes em modo de visualização.

As ações disponíveis variam conforme o perfil e estado da reserva:

Ações do Cliente:

- **Reserva pendente:** pode editar todos os dados da reserva ou cancelá-la. Disponibiliza botões “Editar”, “Entrar em Contato” (WhatsApp com o buffet) e “Cancelar”;
- **Reserva aprovada ou cancelada:** apenas visualização das informações. O botão “Entrar em Contato” permanece disponível.

Ações do Buffet:

- **Reserva pendente:** pode aprovar a reserva informando valor orçado obrigatório e opcionalmente bloqueando a data no calendário, ou pode recusá-la (cancelar). Disponibiliza botões “Aprovar”, “Entrar em Contato” (WhatsApp com o cliente) e “Cancelar”;

- **Reserva cancelada:** pode reverter o cancelamento, retornando a reserva ao estado pendente. Disponibiliza botões “Reverter Cancelamento” e “Entrar em Contato”;
- **Reserva aprovada:** apenas visualização, pois a reserva foi convertida em evento.

Ao aprovar uma reserva, o buffet deve informar o valor orçado através de modal específico, que também permite marcar a opção de bloquear a data no calendário, impedindo novas reservas para o mesmo dia. Ao cancelar ou reverter cancelamento, um modal de confirmação é exibido para evitar ações acidentais.



Figura 7.23: Tela de detalhes da reserva.

Edição de Reserva

A Figura 7.24 apresenta a tela de edição de reserva, acessível apenas pelo cliente enquanto a reserva está com estado **PENDENTE**. Esta interface permite modificação completa de todos os dados da reserva originalmente informados:

- **Nome da reserva:** título identificador do evento;
- **Descrição:** informações adicionais sobre o evento;
- **Data do evento:** através de calendário que valida disponibilidade;
- **Horário desejado:** horário de início do evento;
- **Quantidade de pessoas:** número estimado de convidados;
- **Endereço completo:** todos os campos de localização (rua, número, complemento, bairro, cidade, estado e CEP);
- **Itens de comida e serviço:** visualização dos itens selecionados (a modificação de itens é realizada através de fluxo específico);

O campo de data possui integração com a disponibilidade do buffet, impedindo seleção de datas já ocupadas por outros eventos. O botão “Salvar Alterações” processa as mudanças e atualiza a reserva, mantendo seu estado como **PENDENTE** até análise do buffet.

The figure displays three sequential screenshots of the 'Editar Reserva' (Edit Reservation) screen in a mobile application. Each screen has a back arrow and the title 'Editar Reserva' at the top.

- Left Screenshot:** Shows the 'Comidas' (Food) and 'Serviços' (Services) sections. Under 'Comidas', there is an item 'Lasanha à Bolonhesa' with a description 'Lasanha tradicional com molho bolonhesa e queijo...' and a red trash icon. Below it is a green button 'Adicionar comidas'. Under 'Serviços', there is an item 'Decoração Premium' with a description 'Decoração completa com flores naturais, iluminação...' and a red trash icon. Below it is a green button 'Adicionar serviços'.
- Middle Screenshot:** Shows the 'Nome da reserva' (11/20) field with the value 'Aniversario'. Below it is the 'Descrição da reserva' (24/255) field with the value 'Aniversario da minha mae'. Below that is the 'Quantidade de pessoas' field with the value '20'. Below that is the 'Horário' field with the value '12:30 PM'. Below that is the 'Data' field with the value '23/02/2026' and a calendar icon.
- Right Screenshot:** Shows the 'Endereço completo' section. It includes fields for 'Rua' (Justino), 'Número' (12), 'Complemento' (Casa), 'Bairro' (Centro), 'Cidade' (Pires do Rio), 'Estado' (GO - Goiás), and 'CEP' (75720000). At the bottom is a large green button labeled 'Salvar'.

Figura 7.24: Tela de edição completa da reserva.

7.3.2 Listagem de Eventos

A Figura 7.25 apresenta a tela de listagem de eventos, similar à tela de reservas. Eventos são criados automaticamente quando o buffet aprova uma reserva. A interface possui:

- Campo de pesquisa para filtrar eventos;
- Filtros por estado (Todos, Agendado, Cancelado, Concluído);
- Lista de cards de eventos com título, descrição e badge de estado.

Ao selecionar um card, acessa-se os detalhes do evento.



Figura 7.25: Tela de listagem de eventos.

Detalhes do Evento

A Figura 7.26 apresenta a tela de detalhes de um evento específico, acessível tanto por clientes quanto pelo buffet. Eventos são criados automaticamente quando o buffet aprova uma reserva. A interface exibe o estado atual do evento através de *badge* colorido (Agendado, Cancelado ou Concluído), seguido pelas seguintes informações organizadas em seções:

- **Cliente:** nome completo, e-mail e telefone formatado (visível apenas para o buffet);
- **Data:** data do evento no formato brasileiro (DD/MM/AAAA);
- **Valor Orçado:** valor definido pelo buffet na aprovação da reserva, formatado em reais (R\$);
- **Descrição:** informações detalhadas sobre o evento;
- **Endereço:** localização completa formatada (rua, número, bairro, cidade, estado, CEP e complemento);
- **Horário:** hora de início do evento;
- **Quantidade de pessoas:** número estimado de convidados;
- **Comidas:** lista de itens alimentícios com imagem, nome e descrição;
- **Serviços:** lista de serviços contratados com imagem, nome e descrição.

Os itens de comida e serviço são clicáveis, redirecionando para suas respectivas telas de detalhes em modo de visualização.

As ações disponíveis variam conforme o perfil e estado do evento:

Ações do Cliente:

- **Evento agendado (aprovado):** pode editar informações do evento ou cancelá-lo. Disponibiliza botões “Editar”, “Entrar em Contato” (WhatsApp com o buffet) e “Cancelar”;
- **Evento cancelado ou concluído:** apenas visualização das informações. O botão “Entrar em Contato” permanece disponível.

Ações do Buffet:

- **Evento agendado:** pode editar o valor orçado, concluir o evento ou cancelá-lo. Disponibiliza botões “Editar Valor Orçado”, “Concluir”, “Entrar em Contato” (WhatsApp com o cliente) e “Cancelar”;
- **Evento cancelado:** pode reverter o cancelamento, retornando o evento ao estado agendado. Disponibiliza botões “Reverter Cancelamento” e “Entrar em Contato”;
- **Evento concluído:** apenas visualização das informações. O botão “Entrar em Contato” permanece disponível.

Ao editar o valor orçado, um modal específico é exibido permitindo a atualização do valor. Ao concluir ou cancelar, modais de confirmação são apresentados para evitar ações acidentais. A funcionalidade de reverter cancelamento é exclusiva do buffet, permitindo reativar eventos cancelados.

← Aniversario

Status:

AGENDADO

Data:

23/02/2026

Valor Orçado:

R\$ 1000,00

Descrição:

Aniversario da minha mae

Endereço:

Justino, 12 - Centro - Pires do Rio, GO - 75720000 - Casa

Horário:

12:30 horas

Quantidade de pessoas:

20 pessoas

Comidas:

Lasanha à Bolonhesa

Lasanha tradicional com molho bolonhesa e queijos especiais

← Aniversario

Horário:

12:30 horas

Quantidade de pessoas:

20 pessoas

Comidas:

Lasanha à Bolonhesa

Lasanha tradicional com molho bolonhesa e queijos especiais

Serviços:

Decoração Premium

Decoração completa com flores naturais, iluminação especial e arranjos...

Editar

Entrar em Contato

Cancelar

Figura 7.26: Tela de detalhes do evento.

Edição de Evento

A Figura 7.27 apresenta a tela de edição de evento, acessível por clientes quando o evento está nos estados pendente ou agendado. Esta interface permite modificação completa dos dados do evento, organizados em seções:

Comidas e Serviços:

- Lista de comidas selecionadas com opção de remover individualmente;
- Botão “Adicionar comidas” para incluir novos itens;
- Lista de serviços contratados com opção de remover individualmente;
- Botão “Adicionar serviços” para incluir novos itens.

Informações do Evento:

- **Nome do evento:** campo de texto com limite de 100 caracteres;
- **Descrição do evento:** área de texto com limite de 255 caracteres, ambos com contador visual.

Data e Hora:

- **Data e hora de início:** campos separados para data e horário de início do evento;
- **Data e hora de fim:** campos separados para data e horário de término do evento.

Dados Adicionais:

- **Valor do evento:** campo numérico para visualização do valor orçado;
- **Quantidade de pessoas:** número estimado de convidados;
- **Endereço completo:** rua, número, complemento (opcional), bairro, cidade, estado (seleção via modal) e CEP com formatação automática.

O campo de estado abre modal de seleção com lista completa de UFs brasileiras. O CEP possui formatação automática no padrão brasileiro (XXXXX-XXX). Todos os campos são validados antes do envio. O botão “Salvar” permanece desabilitado até que todos os campos obrigatórios sejam preenchidos corretamente.

Ao remover comidas ou serviços, modais de confirmação são exibidos para evitar exclusões acidentais. As alterações são processadas e o evento é atualizado no servidor, mantendo seu estado atual.

Observação: O buffet possui acesso limitado à edição, podendo apenas atualizar o valor orçado através de modal específico na tela de detalhes, sem acesso à tela completa de edição.

The figure displays three sequential screens of a mobile application for editing an event. Each screen has a back arrow and the title 'Editar Evento'.

Screen 1: Item Selection

- Comidas:** A card for 'Lasanha à Bolonhesa' (Lasagna traditional with meat sauce and cheese) with a delete icon. Below it is a green button 'Adicionar comidas'.
- Serviços:** A card for 'Decoração Premium' (Premium decoration with natural flowers, lighting, etc.) with a delete icon. Below it is a green button 'Adicionar serviços'.
- Nome do evento:** A text field with 'Aniversario' and a character count '11/100'.
- Descrição do evento:** A text area with 'Aniversario da minha mae' and a character count '24/255'.
- Data e Hora de Início:** A section header for the start date and time.

Screen 2: Event Details

- Data e Hora de Início:** Fields for 'Data de início:' (02/23/2026) and 'Hora de início:' (12:30 PM).
- Data e Hora de Fim:** Fields for 'Data de fim:' (02/23/2026) and 'Hora de fim:' (12:30 PM).
- Valor do evento:** A field for the event value (1000).
- Quantidade de pessoas:** A field for the number of people (20).
- Rua:** A field for the street name (Justino).
- Número:** A field for the street number.

Screen 3: Address and Location

- Rua:** A field for the street name (Justino).
- Número:** A field for the street number (12).
- Complemento:** A field for the complement (Casa).
- Bairro:** A field for the neighborhood (Centro).
- Cidade:** A field for the city (Pires do Rio).
- Estado:** A dropdown menu for the state (GO - Goiás).
- CEP:** A field for the postal code (75720000).
- Salvar:** A large green button at the bottom to save the changes.

Figura 7.27: Tela de edição de evento (cliente).

7.3.3 Notificações

A Figura 7.28 apresenta a tela de notificações do sistema. As notificações são geradas automaticamente quando ocorrem mudanças relevantes no estado de reservas e eventos.

Cada notificação exibe:

- Título da notificação;
- Mensagem descritiva;
- Badge verde indicando se foi lida ou não.

Uma notificação pode ser selecionada para visualizar detalhes completos em modal, e deve ser marcada como lida individualmente ao abrir e fechar a o modal da notificação.

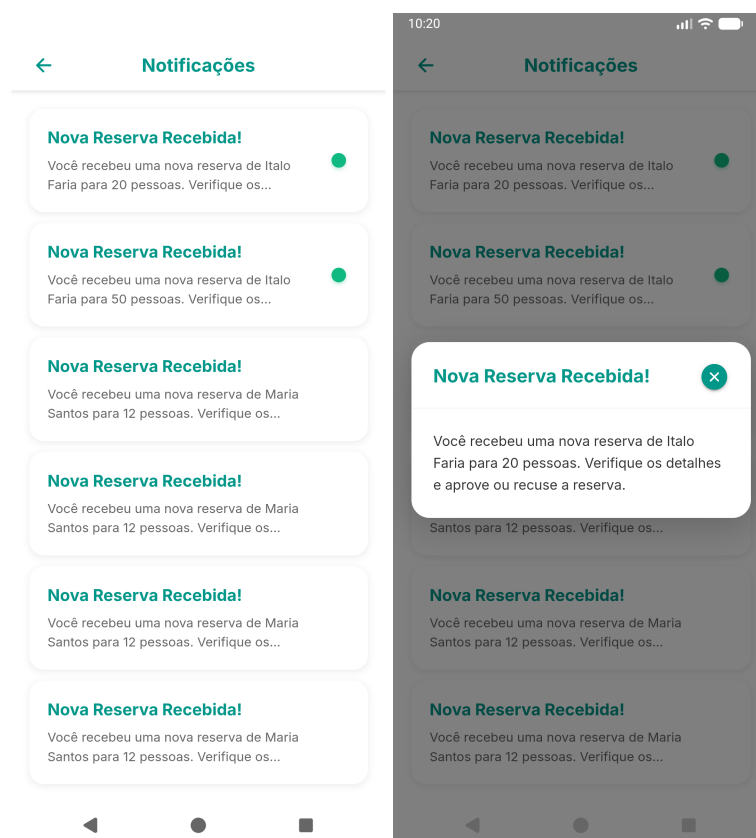


Figura 7.28: Tela de notificações.

7.4 Área Administrativa do Buffet

7.4.1 Dashboard do Buffet

A Figura 7.29 apresenta o painel de controle administrativo do buffet, exibido após a autenticação de usuários com perfil **BUFFET**. A tela contém:

- **Grid de atalhos:** quatro *cards* clicáveis para acesso rápido às principais áreas administrativas, sendo uma página para Comidas (gerenciamento de comidas), Serviços (gerenciamento de serviços), Reservas (acompanhamento e gestão de reservas), Eventos (acompanhamento e gestão de eventos).
- **Calendário mensal:** visualização completa dos dias que estão indisponíveis e disponíveis, permitindo ao buffet ter visão geral da agenda do estabelecimento.

O ícone de notificações no cabeçalho permite acesso rápido às notificações e o botão de retorno no canto superior esquerdo permite realizar *logout* do sistema.

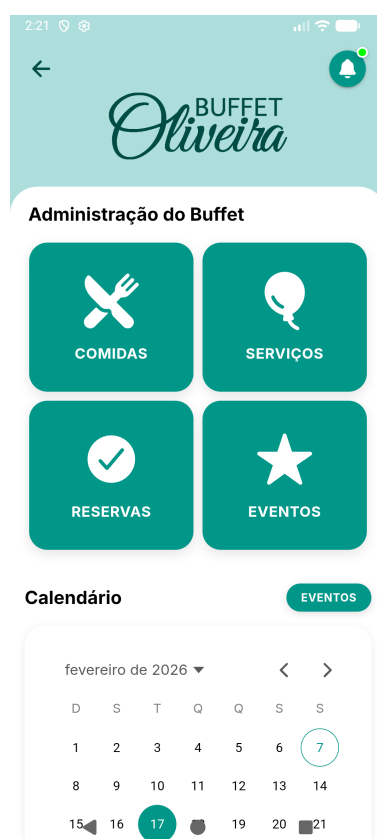


Figura 7.29: Dashboard administrativo do buffet.

7.4.2 Gerenciamento de Comidas

A Figura 7.30 apresenta a tela de gerenciamento de comidas do buffet. Esta interface permite ao administrador:

- Visualizar todas as comidas cadastradas;
- Pesquisar comidas pelo nome;
- Filtrar por categoria;
- Editar, Adicionar e Excluir comidas;

Cada card de comida exibe imagem, nome, descrição e categoria. A listagem implementa *scroll* infinito para melhor desempenho com grandes volumes de dados. Quando não há comidas cadastradas ou nenhuma corresponde aos filtros, a mensagem “Nenhuma comida encontrada” é exibida.

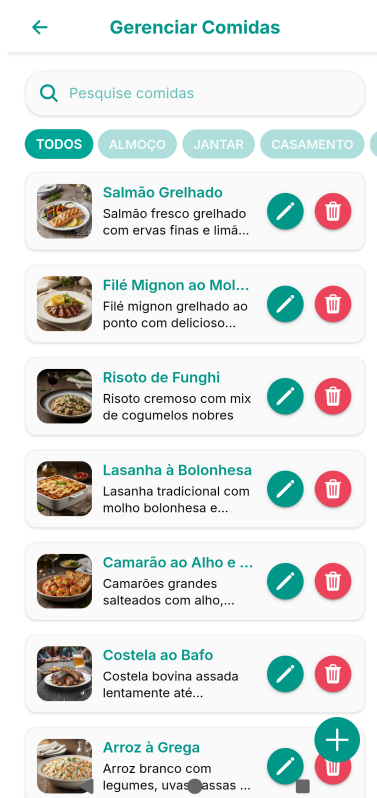


Figura 7.30: Tela de gerenciamento de comidas.

A Figura 7.31 apresenta o formulário de cadastro/edição de comida. Devem ser preenchidos os seguintes campos:

- Nome da comida;
- Descrição detalhada;
- Categoria (seleção única);
- Imagem (upload através de câmera ou galeria).

Ao editar uma comida existente, os campos são preenchidos automaticamente com os dados atuais. A imagem pode ser visualizada antes do envio. O botão “Salvar” processa o formulário e, em caso de sucesso, retorna à tela de gerenciamento. Validações garantem que todos os campos obrigatórios sejam preenchidos.

The figure displays two mobile application screens for food management, side-by-side.

Left Screen: Editar Comida

- Header: Back arrow, "Editar Comida"
- Image: A photo of a grilled salmon dish with vegetables and lemon slices.
- Nome: 15/20. Input field containing "Salmão Grelhado".
- Descrição: 56/255. Input field containing "Salmão fresco grelhado com ervas finas e limão siciliano".
- Categoria: Dropdown menu showing "Casamento".
- Buttons: "Salvar" (green), "Excluir" (red).

Right Screen: Nova Comida

- Header: Back arrow, "Nova Comida"
- Image: Placeholder for a food image.
- Nome: 0/20. Input field with placeholder "Digite o nome da comida".
- Descrição: 0/255. Input field with placeholder "Digite sobre detalhes da comida".
- Categoria: Dropdown menu with placeholder "Defina a categoria da comida".
- Buttons: "Salvar" (green).

Figura 7.31: Formulário de cadastro/edição de comida.

A Figura 7.32 apresenta o modal de confirmação de exclusão de comida. Quando o ícone de lixeira é acionado em um card, este modal é exibido solicitando confirmação da ação. A mensagem informa o nome da comida que será excluída. A ação pode ser confirmada (botão “Sim”) ou cancelada (botão “Não”).

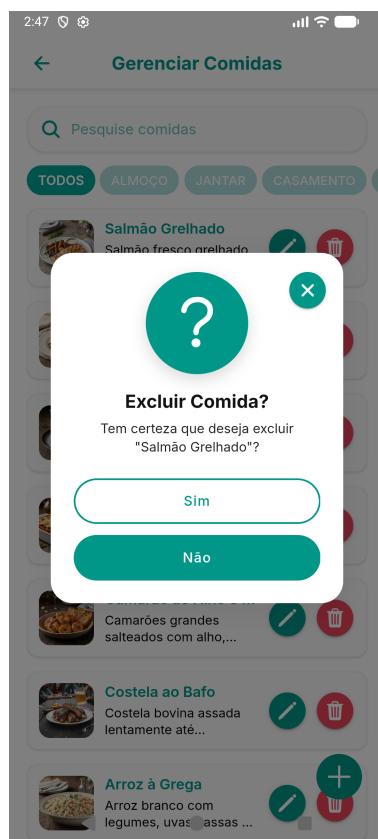


Figura 7.32: Modal de confirmação de exclusão de comida.

7.4.3 Gerenciamento de Serviços

A Figura 7.33 apresenta a tela de gerenciamento de serviços, com estrutura idêntica à tela de gerenciamento de comidas. O administrador pode:

- Visualizar todos os serviços cadastrados;
- Pesquisar serviços pelo nome;
- Filtrar por categoria;
- Editar, Adicionar e Excluir serviços;

Cada card de serviço exibe imagem, nome, descrição e categoria. O botão circular flutuante no canto inferior direito abre o formulário de cadastro de novo serviço.

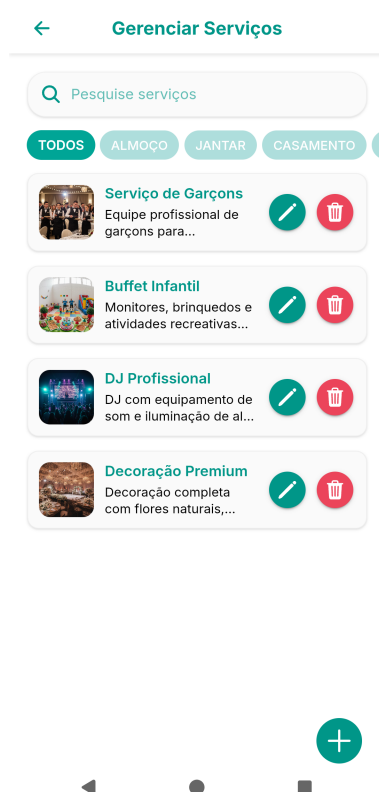


Figura 7.33: Tela de gerenciamento de serviços.

A Figura 7.34 apresenta o formulário de cadastro/edição de serviço, com campos equivalentes ao formulário de comida:

- Nome do serviço;
- Descrição detalhada;
- Categoria;
- Imagem.

O comportamento é idêntico ao formulário de comidas, com validações, visualização prévia de imagem e mensagens de feedback ao usuário.

The figure displays two mobile application screens for service management. The left screen, titled 'Novo Serviço', is for creating a new service. It features a back arrow, a placeholder image, and three input fields: 'Nome:' (0/20) with the placeholder 'Digite o nome do serviço', 'Descrição:' (0/255) with 'Digite sobre detalhes do serviço', and 'Categoria:' with a dropdown menu showing 'Defina a categoria do serviço'. A green 'Salvar' button is at the bottom. The right screen, titled 'Editar Serviço', is for editing an existing service. It also has a back arrow and a preview image of waiters. The input fields are: 'Nome:' (18/20) with 'Serviço de Garçons', 'Descrição:' (57/255) with 'Equipe profissional de garçons para atendimento impecável', and 'Categoria:' with a dropdown menu showing 'Confraternização'. It includes both a green 'Salvar' button and a red 'Excluir' button at the bottom.

Figura 7.34: Formulário de cadastro/edição de serviço.

A Figura 7.32 apresenta o modal de confirmação de exclusão de serviço. Quando o ícone de lixeira é acionado em um card, este modal é exibido solicitando confirmação da ação. A mensagem informa o nome do serviço que será excluída. A ação pode ser confirmada (botão “Sim”) ou cancelada (botão “Não”).

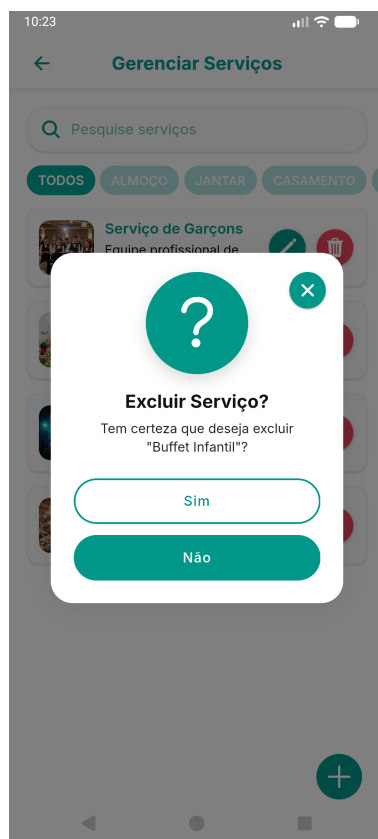


Figura 7.35: Modal de confirmação de exclusão de serviço.

7.5 Modais de Confirmação

O sistema utiliza modais de confirmação para ações críticas que requerem validação explícita do usuário, evitando exclusões ou alterações acidentais.

A Figura 7.36 apresenta o modal exibido quando se tenta sair do aplicativo. O modal questiona “Deseja sair?” e informa que ocorrerá redirecionamento para a tela inicial. Os botões “Sim” e “Não” permitem confirmar ou cancelar a ação.

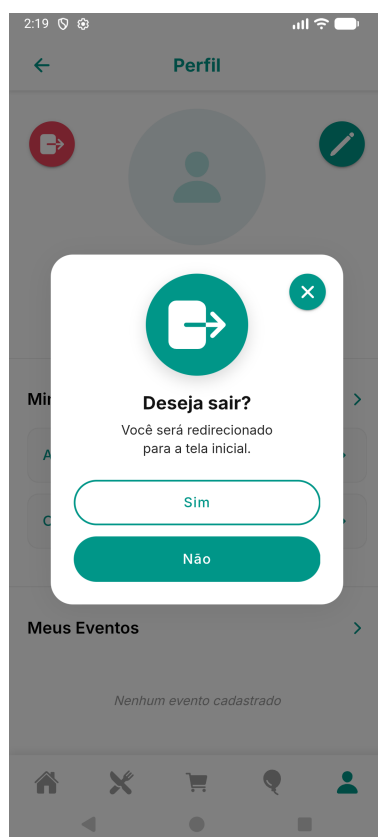


Figura 7.36: Modal de confirmação de saída.

A Figura 7.37 apresenta o modal exibido quando se tenta cancelar uma reserva ou evento. O modal solicita confirmação para que caso o usuário tenha certeza de que está cancelando a reserva ou evento.

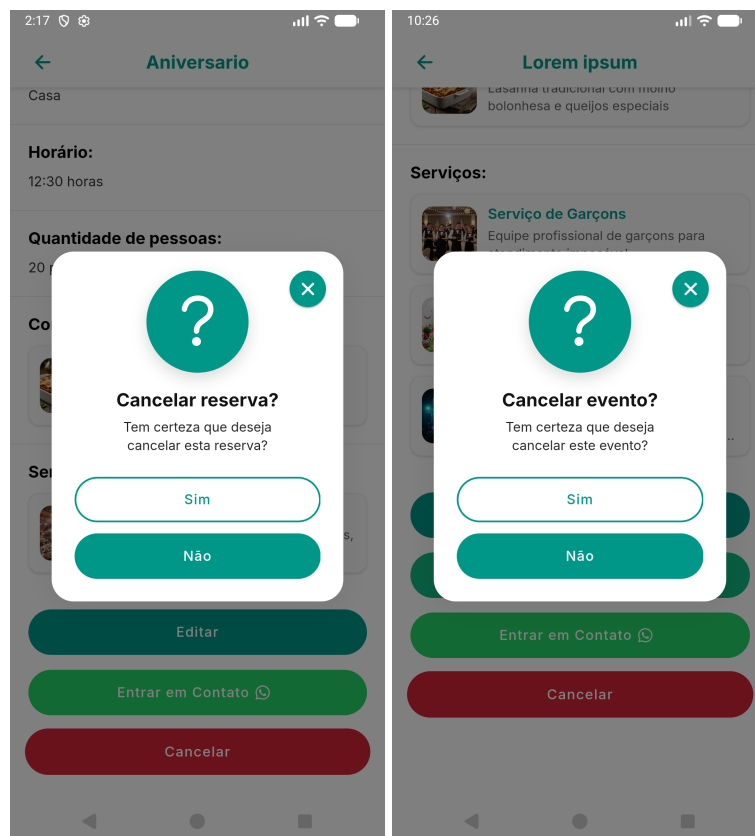


Figura 7.37: Modals de confirmação de cancelamento.

A Figura 7.38 apresenta o modal exibido quando o buffet aprova uma reserva. O modal solicita que seja informado obrigatoriamente o valor estimado do evento e também se a data da reserva deve ser bloqueada e marcada como indisponível para novas reservas.



Figura 7.38: Modal de aprovação de reserva com definição de valor.

7.6 Estados Visuais dos Cards

O sistema utiliza *badges* coloridos para representar visualmente os diferentes estados de reservas e eventos, facilitando a identificação rápida do status atual em listagens e telas de detalhes.

7.6.1 Estados de Reservas

As reservas podem assumir os seguintes estados, cada um com representação visual específica através de *badges* coloridos:

- **PENDENTE:** reserva criada pelo cliente, aguardando análise e aprovação do buffet. Neste estado, o cliente pode editar todos os dados da reserva ou cancelá-la. O buffet pode aprovar (informando valor estimado obrigatório) ou recusar a solicitação;
- **APROVADA:** reserva aprovada pelo buffet com valor estimado definido. Ao ser aprovada, a reserva é automaticamente convertida em evento pelo *back-end*, assumindo este estado como registro histórico da aprovação. Neste estado, não permite mais edições, pois o fluxo continua através do evento criado;
- **CANCELADA:** reserva cancelada pelo cliente antes da aprovação, ou recusada pelo buffet. Quando o buffet recusa uma reserva, esta assume o estado CANCELADA, sendo tratada de forma idêntica a cancelamentos iniciados pelo cliente. Serve como registro histórico, não permitindo mais alterações.

Reservas não possuem estado CONCLUÍDA, pois ao serem aprovadas tornam-se eventos, e apenas eventos podem ser concluídos. Os três estados acima (PENDENTE, APROVADA, CANCELADA) são os únicos possíveis para reservas no sistema.

7.6.2 Estados de Eventos

Os eventos, criados automaticamente a partir da aprovação de reservas, podem assumir os seguintes estados:

- **AGENDADO:** evento confirmado e agendado para ocorrer. Este é o estado inicial de todo evento criado. Neste estado, o cliente pode cancelar (mas não editar), e o buffet pode atualizar valor estimado, cancelar ou concluir o evento;

- **CONCLUÍDO:** evento finalizado com sucesso após sua realização. Ação exclusiva do buffet. Neste estado, nenhuma alteração é permitida, servindo como registro histórico definitivo;
- **CANCELADO:** evento cancelado antes de sua realização, seja por iniciativa do cliente ou do buffet. O buffet possui ação exclusiva de reverter o cancelamento, retornando o evento ao estado AGENDADO. Clientes não podem reverter cancelamentos.

7.6.3 Mapeamento Visual

A Figura 7.39 apresenta os *badges* utilizados no sistema:

- **Amarelo:** reservas pendentes;
- **Verde:** reservas aprovadas e eventos agendados;
- **Vermelho:** reservas e eventos cancelados;
- **Azul:** eventos concluídos.



Figura 7.39: *Badges* de estados de reservas e eventos.

7.7 Discussão

O desenvolvimento do **eBuffet** permitiu verificar, na prática, as limitações identificadas nas soluções existentes durante o levantamento apresentado no Capítulo 2. A Tabela 7.1 retoma a comparação realizada na Tabela 2.3, acrescida de uma coluna referente ao sistema desenvolvido, evidenciando como cada característica foi contemplada.

Tabela 7.1: Comparativo entre soluções existentes e o **eBuffet**.

Característica	Attendize	Odoo Events	Eventbrite	eBuffet
Gestão de cardápio por categorias	Não	Parcial	Não	Sim
Personalização visual por estabelecimento	Não	Não	Não	Sim
Controle de reservas com ciclo de estados	Não	Parcial	Não	Sim
Arquitetura <i>multi-tenant</i>	Não	Sim	Não	Sim
Aplicativo móvel nativo (Android)	Não	Não	Sim	Sim
Notificações automáticas de estado	Não	Parcial	Sim	Sim
Foco específico em buffets	Não	Não	Não	Sim

Conforme evidenciado na Tabela 7.1, as plataformas analisadas apresentam foco genérico no gerenciamento de eventos, não contemplando funcionalidades essenciais para a administração de buffets, como o cadastro e a customização de cardápios por categorias de evento e o controle do ciclo de vida das reservas com aprovação e negociação de valor. O **eBuffet** preenche essas lacunas ao oferecer um conjunto de funcionalidades direcionado especificamente ao domínio de buffets, integrando em uma única plataforma móvel o gerenciamento de comidas, serviços, reservas e eventos, com personalização visual individualizada por estabelecimento.

Do ponto de vista tecnológico, a escolha de **Angular** com **Ionic** e **Capacitor** permitiu o desenvolvimento de uma aplicação híbrida com experiência próxima à nativa em Android, reduzindo o esforço de implementação em relação ao desenvolvimento nativo. A arquitetura *multi-tenant* implementada no *front-end* e no *back-end* garante que múltiplos estabelecimentos possam operar sobre a mesma base de código, com identidade visual e dados completamente isolados, o que posiciona o sistema como solução escalável e de baixo custo de manutenção.

Neste capítulo foi apresentado o resultado final do desenvolvimento por meio das interfaces do sistema, demonstrando as funcionalidades implementadas desde as telas de autenticação e cadastro até o gerenciamento completo de reservas e eventos. Com a aplicação apresentada em sua totalidade, o próximo capítulo se dedica à exposição das considerações finais do trabalho, sintetizando os resultados obtidos, as limitações identificadas e as sugestões para trabalhos futuros.

Este trabalho apresentou o desenvolvimento do **eBuffet**, um aplicativo móvel para Android voltado ao gerenciamento centralizado de buffets e eventos. O objetivo principal do projeto foi suprir a lacuna existente no mercado por soluções que integrem, em uma única plataforma, funcionalidades específicas para a administração de buffets, incluindo o cadastro de comidas e serviços, o controle de reservas e eventos, e a personalização visual da interface conforme a identidade de cada estabelecimento.

O desenvolvimento foi estruturado em etapas bem definidas, iniciando pelo levantamento e análise de requisitos, que permitiu compreender as necessidades do domínio e estabelecer os perfis de usuários (cliente e buffet), os requisitos funcionais e não funcionais, além das regras de negócio que regem o funcionamento do sistema. A modelagem do sistema, apresentada por meio de diagramas UML, consolidou a estrutura conceitual e serviu de base para a implementação.

A arquitetura adotada combinou uma API RESTful desenvolvida em Java com Spring Boot e um aplicativo móvel híbrido construído com Angular, Ionic e Capacitor. Essa combinação proporcionou separação clara entre as camadas de apresentação e negócio, escalabilidade, facilidade de manutenção e experiência nativa em dispositivos móveis. A organização do *front-end* seguiu a abordagem *Feature-Sliced Design*, que promoveu modularidade e reutilização de componentes em todo o projeto.

Entre as principais funcionalidades implementadas, destacam-se: o sistema de autenticação baseado em JWT com controle de perfis; o gerenciamento completo de comidas e serviços pelo buffet, incluindo *upload* de imagens via integração com serviço externo; o fluxo de criação de reservas pelo cliente com seleção de itens, definição de data, horário e endereço; a aprovação

de reservas pelo buffet com conversão automática em eventos; e o sistema de notificações que mantém ambos os perfis informados sobre alterações de estado.

O **eBuffet** também implementou o conceito de *multi-buffet*, permitindo que múltiplos estabelecimentos operem sobre a mesma base tecnológica com isolamento lógico de dados e personalização visual, demonstrando a viabilidade de uma plataforma centralizada para o setor de eventos.

Os resultados obtidos demonstram que o **eBuffet** atende aos objetivos propostos, oferecendo uma solução moderna, intuitiva e funcional para a gestão de buffets e eventos. A interface desenvolvida prioriza a usabilidade, com componentes padronizados, *feedback* visual adequado e navegação simplificada para ambos os perfis de usuário.

8.1 Limitações do Estudo

Apesar dos resultados alcançados, o trabalho apresenta algumas limitações que devem ser consideradas. O levantamento de requisitos foi realizado com base em entrevista junto à proprietária de um único estabelecimento, o que pode não refletir a totalidade das necessidades do setor. Além disso, não foram conduzidos testes formais de usabilidade com usuários reais, de modo que a avaliação da interface se baseou na experiência dos próprios desenvolvedores e nos critérios estabelecidos nos requisitos. Do ponto de vista técnico, o aplicativo foi disponibilizado apenas para a plataforma Android, e a ausência de notificações *push* nativas reduz a efetividade dos alertas quando o aplicativo não está em uso ativo. Por fim, embora o conceito de *multi-buffet* tenha sido implementado com isolamento lógico de dados, não foram realizados testes de carga para mensurar o comportamento do sistema diante de um volume elevado de estabelecimentos e usuários simultâneos.

Vale ressaltar que a dependência de conexão com a internet é uma **característica** inerente à arquitetura cliente-servidor adotada, necessária para a comunicação em tempo real com a API e a sincronização de dados entre os perfis de cliente e buffet, não configurando uma limitação do projeto.

O **eBuffet** também não contempla um módulo de pagamento integrado, o que exige que a negociação financeira entre cliente e buffet ocorra por meios externos à plataforma. A comunicação entre cliente e buffet restringe-se ao sistema de notificações, não havendo um canal direto de mensagens dentro da aplicação para tratar particularidades de cada evento.

Como trabalhos futuros, sugere-se a implementação de funcionalidades adicionais que podem agregar ainda mais valor à plataforma:

- **Integração de pagamentos:** implementação de módulo para pagamento *online*, permitindo que clientes realizem pagamentos de sinal ou valor integral através do aplicativo, integrando com *gateways* como Stripe, PagSeguro ou Mercado Pago.
- **Sistema de avaliações:** permitir que clientes avaliem buffets após a conclusão de eventos, com sistema de notas e comentários que auxiliem outros usuários na escolha do estabelecimento.
- **Versão para iOS:** expansão do aplicativo para a plataforma iOS, aproveitando a arquitetura híbrida já implementada com Ionic e Capacitor.
- **Notificações push:** implementação de notificações *push* nativas para alertar usuários sobre mudanças de estado mesmo quando o aplicativo não estiver em primeiro plano.
- **Chat integrado:** implementação de sistema de mensagens entre cliente e buffet para facilitar a comunicação durante o processo de negociação e organização de eventos.

Portanto, o **eBuffet** representa uma contribuição relevante para o setor de eventos, oferecendo uma alternativa tecnológica moderna que centraliza e otimiza os processos de gestão de buffets. Atualmente, o aplicativo é distribuído por meio de arquivo APK para instalação direta em dispositivos Android, não estando disponível na Play Store nem possuindo versão *web* ou *desktop* para acesso pelo proprietário do buffet, aspectos que configuram oportunidades de evolução para versões futuras. A solução desenvolvida estabelece uma base sólida para futuras evoluções, podendo ser expandida e adaptada para atender a novos requisitos e funcionalidades.

REFERÊNCIAS BIBLIOGRÁFICAS

- ABRAPE. *Números do Setor de Eventos no Brasil*. 2023. <<https://www.abrape.com.br/numeros-do-setor/>>. Dados sobre faturamento anual, participação no PIB e geração de empregos do setor de eventos no Brasil. Acesso em: 14 fev. 2026.
- ALHARTHI, K.; YÜKSEL, A. S.; CANKAYA, I. A.; CANKAYA, S. F. *IoT for Hospitality Industry: Paperless Buffet Management*. [S.l.], 2020. Article in International Journal of Computer Applications, 177(25):1–5. Disponível em: <https://www.researchgate.net/publication/338301272_IoT_for_Hospitality_Industry_Paperless_Buffet_Management>.
- AMUNDSEN, M. *RESTful Web APIs*. [S.l.]: O'Reilly Media, 2022. ISBN 9781098106748.
- Attendize. *Attendize – Open-source Ticket Selling & Event Management Platform*. 2023. <<https://www.attendize.com/>>. Accessed: 6 January 2026.
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *The Unified Modeling Language User Guide*. 2. ed. Boston: Addison-Wesley, 2005. ISBN 9780321267979.
- Business Rules Group. *Defining Business Rules – What Are They Really?* 2000. Disponível em: <https://www.businessrulesgroup.org/first_paper/BRG-whatBR_3ed.pdf>.
- Eventbrite. *Eventbrite – Event Management and Ticketing Platform*. 2023. <<https://www.eventbrite.com/>>. Accessed: 6 January 2026.
- FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. [S.l.], 2000. Doctoral dissertation. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- FOWLER, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3. ed. Boston: Addison-Wesley, 2004. ISBN 9780321193681.
- GEORGIU, J.; SYMEONIDES, M.; PALLIS, G.; DIKAIKOS, M. D. Automating multi-tenancy performance evaluation on edge compute nodes. In: *2025 IEEE International Conference on Edge Computing and Communications (EDGE)*. IEEE, 2025. Disponível em: <<https://arxiv.org/abs/2506.10461>>.
- Ionic Framework. *Ionic Documentation*. 2024. <<https://ionicframework.com/docs>>. Accessed: 6 January 2026.

Ionic Team. *Capacitor Documentation*. 2024. <<https://capacitorjs.com/docs>>. Accessed: 6 January 2026.

JONES, M.; BRADLEY, J.; SAKIMURA, N. *JSON Web Token (JWT)*. [S.l.], 2015. Request for Comments. Disponível em: <<https://www.rfc-editor.org/rfc/rfc7519>>.

Odoo. *Odoo Events – All-in-One Event Management*. 2024. <<https://www.odoo.com/app/events>>. Accessed: 6 January 2026.

PICCIONI, N. From physical to metaversal events: An exploratory study. *Italian Journal of Marketing*, v. 2023, n. 2, p. 119–134, 2023. Exploratory study on digital and hybrid events. Disponível em: <https://ideas.repec.org/a/spr/ijmark/v2023y2023i2d10.1007_s43039-023-00068-1.html>.

RICHARDSON, L.; RUBY, S. *RESTful Web Services*. [S.l.]: O'Reilly Media, 2007. ISBN 9780596529260.

SOMMERVILLE, I. *Software Engineering*. 10. ed. Boston: Pearson, 2016. ISBN 9780133943030.