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

GÉSSICA DA SILVA MEIRELES E RAQUEL BENEVIDES LOPES

DESENVOLVIMENTO DE UMA API PARA RESTAURANTES E PAINEL DO CLIENTE CONTENDO INFORMAÇÃO NUTRICIONAL

Urutaí, 13 de março de 2026

INSTITUTO FEDERAL GOIANO
NÚCLEO DE INFORMÁTICA
SISTEMAS DE INFORMAÇÃO

GÉSSICA DA SILVA MEIRELES E RAQUEL BENEVIDES LOPES

DESENVOLVIMENTO DE UMA API PARA RESTAURANTES E PAINEL DO CLIENTE CONTENDO INFORMAÇÃO NUTRICIONAL

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

L864d

Lopes, Raquel; Meireles, Géssica, .
DESENVOLVIMENTO DE UMA API PARA RESTAURANTES E PAINEL DO CLIENTE CONTENDO INFORMAÇÃO NUTRICIONAL / . Lopes, Raquel; Meireles, Géssica. Urutaí 2026.

82f. 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. API. 2. Restaurantes. 3. Cardápio on-line. 4. Informação nutricional. 5. Tabela TACO. I. Título.

INSTITUTO FEDERAL
Goiano

Repositório Institucional do IF Goiano - RIIF Goiano
Sistema Integrado de Bibliotecas

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

Nome completo do autor: Raquel Benevides Lopes e Gessica da Silva Meireles

Matrícula: 2021101202010168

Título do trabalho: DESENVOLVIMENTO DE UMA API PARA RESTAURANTES E PAINEL DO CLIENTE

RESTRIÇÕES DE ACESSO AO DOCUMENTO

Documento confidencial: ☑ Não ☐ Sim, justifique:

Informe a data que poderá ser disponibilizado no RIIF Goiano: 13 / 03 / 2026

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.

Urutaí (Local) 13 / 03 / 2026 (Data)

gov.br Documento assinado digitalmente
RAQUEL BENEVIDES LOPES
Data: 13/03/2026 08:28:20-0300
Verifique em https://validar.iti.gov.br

Gessica Meireles

Assinatura do autor e/ou detentor dos direitos au

Ciente e de acordo:

gov.br Documento assinado digitalmente
CRISTIANE DE FATIMA DOS SANTOS CARDOSO
Data: 13/03/2026 08:41:48-0300
Verifique em https://validar.iti.gov.br

Assinatura do(a) orientador(a)

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

Formulário 18/2026 - 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 quatro dias do mês de março de dois mil e vinte e seis, reuniram-se os professores: Cristiane de Fátima dos Santos Cardoso, Júnio César de Lima e Giovani Barbosa dos Santos filho no ambiente virtual google meet por meio do link https://meet.google.com/qcv-xfua-ckg?authuser=1, para avaliar o Trabalho de Curso do(s) acadêmico(s): **GÉSSICA DA SILVA MEIRELES E RAQUEL BENEVIDES LOPES**, como requisito necessário para a conclusão do Curso Superior de Sistemas de Informação desta Instituição. O presente TC tem como título: **DESENVOLVIMENTO DE UMA API PARA RESTAURANTES E PAINEL DO CLIENTE CONTENDO INFORMAÇÃO NUTRICIONAL ,** foi orientado por Cristiane de Fátima dos Santos Cardoso.

Após análise, foram dadas as seguintes notas:

Professores	**Aluno / Notas**	
	GÉSSICA DA SILVA MEIRELES	RAQUEL BENEVIDES LOPES
1. Cristiane de Fátima dos Santos Cardoso	8	8
2. Júnio César de Lima	8	8
3. Giovani Barbosa dos Santos Filho	8	8

MÉDIA FINAL:	8	8

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** , **PROFESSOR ENS BASICO TECN TECNOLOGICO** , em 04/03/2026 15:22:43.
- **Giovani Barbosa dos Santos Filho** , **PROF ENS BAS TEC TECNOLOGICO-SUBSTITUTO** , em 04/03/2026 15:23:54.
- **Junio Cesar de Lima** , **PROFESSOR ENS BASICO TECN TECNOLOGICO** , em 04/03/2026 15:24:38.

Este documento foi emitido pelo SUAP em 04/03/2026. 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: 795864
Código de Autenticação: 4ab56d5acd

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 Raquel Benevides

Agradeço, em primeiro lugar, a Deus, que foi meu sustento em todos os momentos. Nos dias de cansaço, Ele me fortaleceu; nos momentos de dúvida, concedeu-me direção; e, nas dificuldades, renovou minha fé para que eu não desistisse. Sem Ele, nada disso seria possível. Aos meus pais, Edson e Inácia, pelo amor, apoio e dedicação incondicional; ao meu irmão Lucas e à minha irmã Ruth, pelo companheirismo, incentivo e carinho; e a toda a minha família, que sempre esteve presente, torcendo e vibrando a cada conquista. À Vanessa, Jhennyf, Allefy, Fernanda, Amanda, Jamily, Bruno e Giselle, que já estavam comigo antes mesmo da faculdade, obrigada por sempre me apoiarem. Vocês conhecem meus sonhos, minhas inseguranças e minha vontade constante de ir além. Estiveram ao meu lado antes e durante essa jornada, acreditando em mim e me dando força nos momentos mais difíceis. Aos amigos que a faculdade me trouxe — Kaio, João Vitor, André, Farley e Arthur —, obrigada por tornarem essa caminhada mais leve. Entre desafios, trabalhos, noites de estudo e muitas conversas, construímos laços que levarei para a vida. Aos meus pastores e às irmãs do ciclo de oração, que foram instrumentos de Deus na minha vida, intercedendo por mim e fortalecendo minha fé ao longo dessa caminhada. Cada oração fez diferença. À minha orientadora, Cristiane Cardoso, pela dedicação, paciência e orientação fundamental para a realização deste trabalho. À minha dupla do TCC, Géssica, pela parceria, comprometimento e esforço compartilhado durante todo o desenvolvimento do projeto. Por fim, agradeço a todos que, de alguma forma, contribuíram para minha formação acadêmica, pessoal e espiritual. Esta conquista não é apenas minha, mas de todos que caminharam ao meu lado.

Por Géssica Meireles

Primeiramente, agradeço de todo o meu coração a Deus pelo privilégio de realizar este sonho. Por estar sempre ao meu lado durante toda essa jornada, concedendo-me força, sabedoria e fé para continuar, mesmo diante das dificuldades. Ele foi meu amparo e meu auxílio em todos os momentos, tornando possível a concretização desta etapa tão importante da minha vida. Agradeço aos meus pais, Gerson e Francisca, pelo amor, cuidado, dedicação e por todos os esforços realizados para que eu pudesse chegar até aqui. Aos meus avós, Manoel e Oterlina, João e Maria, e aos meus irmãos, Vanilson, Vanessa e João Paulo, e a todos os meus familiares, que sempre me apoiaram com palavras de carinho, incentivo e compreensão. Sem vocês, nada disso seria possível. Aos meus amigos de infância, Jhennyf, Fernanda, Ruth, Amanda, Bruno, Emanuelly e Allefy, que estiveram comigo em todos os momentos, compartilhando alegrias e desafios, oferecendo apoio, força e companheirismo ao longo dessa jornada. Aos meus amigos de curso, João, Kaio e Farley, que fizeram parte dessa fase tão importante da minha vida, sempre

me apoiando e contribuindo de maneira significativa para a minha trajetória acadêmica. Ao meu namorado, João Vitor, por estar sempre ao meu lado, incentivando-me e acreditando que esse sonho seria possível. Seu apoio foi fundamental para que eu chegasse até aqui. Aos meus pastores e às irmãs do círculo de oração, por todas as orações, palavras de incentivo e conforto, que fortaleceram minha fé e meu coração nos momentos mais desafiadores. Aos meus professores do curso de Sistemas de Informação, pela dedicação e pelos ensinamentos transmitidos ao longo da graduação, que foram fundamentais para a minha formação acadêmica e profissional. Em especial, à minha orientadora, Cristiane, pela dedicação, paciência e orientações, que foram essenciais para a conclusão deste trabalho. À minha dupla, Raquel, com quem tive a honra de compartilhar essa jornada. Muito obrigada por toda dedicação, empenho e companheirismo. Dividir essa vitória com você torna tudo ainda mais especial.Por fim, sou imensamente grata a todos que, de alguma forma, contribuíram para a minha formação acadêmica e pessoal. Esta conquista também é de vocês.

RESUMO

As tecnologias para restaurantes têm se mostrado grandes aliadas, oferecendo sistemas capazes de otimizar processos e facilitar o trabalho das equipes, tornando-se um diferencial competitivo quando bem utilizadas. Ao mesmo tempo, observa-se um público cada vez mais preocupado com a alimentação, e que muitas vezes, possui dificuldades em conseguir opções adequadas às suas demandas nutricionais ou seguras diante de restrições alimentares. Nesse contexto, o presente trabalho apresenta o desenvolvimento de um sistema denominado MenuNutri, composto por uma API para pedidos e controle de comandas em restaurantes, com o diferencial de incluir um painel do cliente contendo informações nutricionais. A metodologia adotada consiste do levantamento e análise de requisitos, prototipação, modelagem do sistema por meio de diagramas UML, implementação da API, desenvolvimento do banco de dados e integração da Tabela Brasileira de Composição de Alimentos (TACO). O levantamento é realizado a partir de entrevistas com nutricionistas e gestores, além da pesquisa de sistemas semelhantes. Como resultado, é gerado um sistema que permite o cadastro e gerenciamento de usuários e funcionários com diferentes níveis de acesso, além do controle de pratos, reservas e pedidos. Como resultado é apresentado o MenuNutri, uma ferramenta capaz de auxiliar restaurantes, proporcionando aos clientes acesso transparente às informações nutricionais, promovendo melhor experiência de uso e suporte à tomada de decisão alimentar.

Palavras-chave:

API, Restaurantes, Cardápio on-line, Informação nutricional, Tabela TACO.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API *Application Programming Interface*. Pag.18

DTOs Data Transfer Objects. Pag.19

JWT JSON Web Token. Pag.64

TACO Tabela Brasileira de Composição de Alimentos. Pag.16

UML *Unified Modeling Language*. Pag.29

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

A utilização da tecnologia para gerir operações no setor de restaurantes é uma tendência crescente, conforme apontam Gendron J.; Gendron (2018), Blöcher K.; Alt (2021). Esses sistemas são empregados para gerenciar processos internos e interagir com clientes; entretanto, muitos ainda enfrentam desafios relacionados à usabilidade, apresentando interfaces complexas e pouco intuitivas, o que pode dificultar uma operação eficiente.

Entre os sistemas existentes no mercado, destacam-se o Consumer, o Anota AI e o Best Restaurante, que oferecem diversas funcionalidades voltadas principalmente ao gerenciamento e controle administrativo do restaurante, contribuindo para a otimização do trabalho, organização e eficiência do estabelecimento (Consumer, 2026; Anota AI, 2026; Best Software, 2026).

Paralelamente, observa-se um aumento na preocupação da população com hábitos alimentares e com a qualidade da alimentação. De acordo com levantamento da Federação das Indústrias de São Paulo Conselho Federal de Nutricionistas (CFN) (2018), oito em cada dez brasileiros afirmam esforçar-se para manter uma alimentação saudável. Nesse contexto, cresce também a busca por informações relacionadas à composição nutricional dos alimentos consumidos. Entretanto, muitos dos sistemas utilizados atualmente na gestão de restaurantes, como Consumer, Anota AI e Best Restaurante, são voltados principalmente para a gestão de pedidos, controle de mesas e integração com serviços de delivery, não contemplando, de forma abrangente, a disponibilização de informações nutricionais detalhadas aos clientes.

Além disso, muitas pessoas possuem restrições alimentares que podem estar relacionadas a alergias, intolerâncias ou condições médicas específicas. Segundo o Ministério da Saúde (Ministério da Saúde (Brasil), 2022), alergias alimentares ocorrem quando o sistema imunoló-

gico reage de forma exagerada a determinadas substâncias presentes nos alimentos, podendo causar sintomas que variam de manifestações leves até reações mais graves. Já as intolerâncias alimentares estão associadas à dificuldade do organismo em digerir determinados componentes dos alimentos, geralmente resultando em sintomas gastrointestinais ((ASBAI), 2023). Há ainda restrições relacionadas a condições de saúde específicas, como a doença celíaca, cujo tratamento envolve uma dieta isenta de glúten (ARAúJO et al., 2010). Também há restrições alimentares ligadas a decisões pessoais, éticas ou culturais, que podem refletir valores e estilos de vida distintos. Essas escolhas são impulsionadas por fatores como saúde, bem-estar animal e a busca por hábitos alimentares mais equilibrados. Exemplos incluem o vegetarianismo, quando a pessoa opta por não consumir carnes, e o veganismo, que exclui qualquer alimento ou produto de origem animal. Questões culturais, sociais e religiosas, que podem influenciar os hábitos alimentares, também são discutidas por (JAYASINGHE S.; BYRNE, 2025).

Diante desse cenário, torna-se relevante disponibilizar informações nutricionais de forma clara e acessível aos consumidores. No entanto, muitos sistemas utilizados por restaurantes não oferecem esse tipo de informação, o que dificulta a realização de escolhas alimentares mais conscientes por parte dos clientes.

Assim, este trabalho propõe o desenvolvimento de um sistema voltado ao apoio na gestão operacional de restaurantes, com foco na organização de pedidos, controle de comandas e disponibilização digital do cardápio aos clientes. O sistema, designado por MenuNutri, busca proporcionar uma experiência simplificada e intuitiva, permitindo que os usuários consultem os pratos disponíveis e tenham acesso às informações nutricionais dos alimentos oferecidos. Utilizando como referência a Tabela Brasileira de Composição de Alimentos (TACO, 2020), a aplicação fornece dados detalhados sobre a composição nutricional dos pratos, auxiliando os clientes no processo de decisão alimentar. Além disso, o sistema permite o registro e gerenciamento de pedidos realizados no estabelecimento, contribuindo para uma organização mais eficiente do atendimento e do fluxo de comandas.

1.1 Estrutura do Trabalho

O Capítulo 1 apresenta o tema do projeto e aborda sistemas de restaurantes já existentes e seus desafios referentes à usabilidade e à ausência de informações nutricionais.

O Capítulo 2 descreve os fundamentos teóricos, apresentando, em cada seção, informações sobre sistemas gerenciais em restaurantes, dados importantes a respeito da Tabela TACO e

todas as tecnologias utilizadas para o desenvolvimento tanto da API quanto dos protótipos e do *front-end*, além de explicar como é realizado o desenvolvimento de uma API.

No Capítulo 3, é apresentado o levantamento dos requisitos para o desenvolvimento do sistema, incluindo todos os diagramas UML e os protótipos iniciais das telas.

O Capítulo 4 descreve o desenvolvimento do sistema, explicando todas as etapas necessárias para o desenvolvimento do *back-end* e do *front-end*, abordando a estrutura dos pacotes, as tecnologias e os *frameworks* utilizados, a arquitetura da aplicação, trechos de código e apresentando os resultados das telas.

CAPÍTULO 2

FUNDAMENTOS BÁSICOS

2.1 Sistemas gerenciais em restaurantes

Para o bom funcionamento de um restaurante, é necessário um sistema de gestão que atenda e supra todas as suas necessidades, capaz de gerenciar todas as áreas do estabelecimento e automatizar os processos de trabalho. Dessa forma, a criação e a popularização de sistemas de gestão para restaurantes são de suma importância, pois permitem que todos os setores garantam maior eficiência, agilidade e controle sobre suas operações. O uso de tecnologia para gerenciar operações no setor de restaurantes mostra uma tendência crescente, contribuindo para o melhor desempenho dos negócios (KOCAMAN, 2021).

Com o grande crescimento do setor gastronômico e o aumento do uso de tecnologias, estas passaram a ser aplicadas para auxiliar na coordenação de tarefas operacionais. Braga V (2023) afirma que empresas, em quase todos os setores, conduziram iniciativas para explorar novas tecnologias e seus benefícios, permitindo a transformação das principais operações. Entre os sistemas disponíveis no mercado, destacam-se o Consumer e o Anota AI, que são amplamente utilizados no setor alimentício.

O sistema Consumer é uma solução voltada para a gestão completa de restaurantes, oferecendo funcionalidades como controle de pedidos em tempo real e integração com caixa e gestão administrativa. Já o Anota AI apresenta uma proposta voltada para a automação do atendimento e ampliação dos canais de pedido, permitindo a gestão de pedidos realizados por diferentes meios, como delivery, salão, balcão e cardápio digital.

Embora existam diversos sistemas de gestão para restaurantes disponíveis no mercado, neste trabalho foram analisados apenas os sistemas Consumer e Anota AI. A escolha desses dois

sistemas se deve ao fato de serem soluções amplamente utilizadas por estabelecimentos do setor, além de apresentarem funcionalidades diretamente relacionadas ao desenvolvimento do sistema proposto neste trabalho.

A Tabela 2.1 mostra sistemas de gestão voltados para restaurantes, evidenciando suas diferenças. Alguns são mais completos e têm foco em grandes estabelecimentos, enquanto outros atendem a negócios menores. Dentre os sistemas mencionados, observam-se diferenças como a gestão e o controle de pedidos, integração com caixa e estoque, entre outros aspectos. Os sistemas apresentados na tabela são consolidados no mercado, oferecendo funcionalidades que abrangem desde a gestão de pedidos até o controle financeiro.

Dentre os sistemas apresentados, o Consumer se destaca por atender mais de 30 mil restaurantes, enquanto o Anota AI possui mais de 15 mil estabelecimentos automatizados, demonstrando que ambos apresentam bom desempenho no setor de serviços de alimentação. A gestão de pedidos dos dois sistemas apresenta funcionalidades completas. O Anota AI expande os canais de atendimento, permitindo o controle de pedidos por delivery, salão, balcão e cardápio digital; já o Consumer trabalha com pedidos em tempo real e integração com o caixa.

Critério	**Consumer**	**Anota AI**
Público-alvo	Mais de 30 mil restaurantes no Brasil.	Mais de 15 mil estabelecimentos automatizados.
Período de teste	Disponibiliza versão gratuita para testes.	Oferece teste gratuito de 7 dias.
Gestão de pedidos	Pedidos em tempo real, integrados ao caixa e estoque.	Controle de pedidos por delivery, salão, balcão, WhatsApp e cardápio digital.
Controle financeiro e de estoque	Controle automatizado, relatórios e ficha técnica.	Gestão financeira completa, cálculo automático de CMV e controle de insumos.
Integrações e notas fiscais	Integrações com iFood e emissão de NFe.	Emissão de NFC-e, integração com WhatsApp e redes sociais.
Marketing e fidelização	Programa de fidelidade, combos e QR Code para pedidos.	Campanhas de marketing, cashback, cupons e chatbot.
Suporte	Suporte online e ampla base de usuários.	Suporte via plataforma e atendimento automatizado.
Modelo de cobrança	Assinatura mensal, com versão gratuita.	Assinatura mensal por chatbot, conforme plano contratado.

Tabela 2.1: Comparativo entre os sistemas Consumer e Anota AI

2.2 Tabela TACO

A Tabela Brasileira de Composição de Alimentos (TACO) apresenta os dados sobre a composição dos principais alimentos consumidos no Brasil, com base em um plano de amostragem que garante valores representativos (TACO, 2020). Coordenada pelo NEPA/UNICAMP, trata-se de uma iniciativa que oferece informações sobre um expressivo número de nutrientes presentes em alimentos nacionais e regionais, obtidos por meio de análises laboratoriais.

A TACO apresenta alimentos cujas calorias são contabilizadas, abrangendo 597 alimentos consumidos no Brasil, e é utilizada por profissionais da área para elaborar dietas equilibradas. Ela também é empregada em sistemas computacionais, aplicativos e plataformas web voltadas à nutrição, como o Dietbox (DIETBOX, 2025). A TACO serve como base de dados confiável para calcular automaticamente o valor nutricional de pratos e receitas. Com essas informações, é possível obter dados precisos sobre o consumo alimentar, auxiliando na promoção de hábitos saudáveis.

Na TACO, as porções são padronizadas em 100 g, tanto para alimentos líquidos quanto para alimentos sólidos. A tabela está dividida em três partes: CMVcol, AGtaco e Aminoácidos, de forma a facilitar o armazenamento, a consulta e o processamento das informações nutricionais.

A primeira parte, CMVcol, concentra a composição centesimal e os micronutrientes dos alimentos, incluindo valores de umidade, energia (kcal e kJ), proteínas, lipídios totais, colesterol, carboidratos, fibra alimentar, cinzas, minerais (como cálcio, magnésio, ferro, sódio, potássio, manganês, cobre e zinco) e vitaminas (como retinol, equivalentes de retinol, tiamina, riboflavina, piridoxina, niacina e vitamina C). Todos os valores são calculados para uma porção padrão de 100 g do alimento.

A segunda parte, AGtaco, é responsável pelo detalhamento do perfil lipídico, apresentando informações sobre os ácidos graxos, como gorduras saturadas, monoinsaturadas e poli-insaturadas, além de tipos específicos, como C18:1, C18:2 n-6, C20:5 e C22:6.

Por fim, a terceira parte, denominada Aminoácidos, contém dados referentes à composição de aminoácidos das proteínas presentes nos alimentos, incluindo triptofano, leucina, lisina, valina, alanina, ácido aspártico e ácido glutâmico, o que permite análises mais específicas sobre a qualidade proteica.

A Figura 2.1 apresenta um recorte da Tabela Brasileira de Composição de Alimentos

[1]Consumer: <https://consumer.com.br>
[2]Anota AI: <https://www.anota.ai>

(TACO), utilizada como fonte de dados nutricionais no sistema proposto. A planilha apresenta informações detalhadas sobre a composição dos alimentos, como energia, macronutrientes, vitaminas e minerais. Observa-se também que a base de dados está organizada em diferentes partes, denominadas CMVcol, AGtaco3 e Aminoácidos TACO3, cada uma contendo conjuntos específicos de informações nutricionais.

	Umidade	Energia		Proteína	Lipídeos	Colesterol	Carbo-idrato	Fibra Alimentar	Cinzas	Cálcio	Magnésio	Número do Alimento	Manganês	Fósforo	Ferro	Sódio	Potássio	Cobre	Zinco	Retinol	RE	RAE	Tiamina	Riboflavina	Piridoxina	Niacina	Vitamina C
Descrição dos alimentos	(%)	(kcal)	(kJ)	(g)	(g)	(mg)	(g)	(g)	(g)	(mg)	(mg)		(mg)	(mg)	(mg)	(mg)	(mg)	(mg)	(mg)	(mcg)	(mcg)	(mg)	(mg)	(mg)	(mg)	(mg)
erivados																										
Arroz, integral, cozido	70,1	124	517	2,6	1,0	NA	25,8	2,7	0,5	5	59	1	0,63	106	0,3	1	75	0,02	0,7	NA			0,08	Tr	0,08	Tr	
Arroz, integral, cru	12,2	360	1505	7,3	1,9	NA	77,5	4,8	1,2	8	110	2	2,99	251	0,9	2	173	0,07	1,4	NA			0,26	Tr	0,18	4,18	
Arroz, tipo 1, cozido	69,1	128	537	2,5	0,2	NA	28,1	1,6	0,1	4	2	3	0,30	18	0,1	1	15	0,02	0,5	NA			Tr	Tr	Tr	Tr	
Arroz, tipo 1, cru	13,2	358	1497	7,2	0,3	NA	78,8	1,6	0,5	4	30	4	1,03	104	0,7	1	62	0,11	1,2	NA			0,16	Tr	0,07	1,12	
Arroz, tipo 2, cozido	68,7	130	544	2,6	0,4	NA	28,2	1,1	0,1	3	6	5	0,37	22	0,1	2	20	0,04	0,5	NA			Tr	Tr	Tr	Tr	
Arroz, tipo 2, cru	13,2	358	1498	7,2	0,3	NA	78,9	1,7	0,4	5	29	6	0,83	82	0,6	1	57	0,05	1,3	NA			0,16	Tr	0,05	0,92	
Aveia, flocos, crua	9,1	394	1648	13,9	8,5	NA	66,6	9,1	1,8	48	119	7	1,89	153	4,4	5	336	0,44	2,6	NA			0,55	0,03	Tr	4,47	1,4
Biscoito, doce, maisena	3,2	443	1853	8,1	12,0	NA	75,2	2,1	1,5	54	37	8	0,78	166	1,8	352	142	0,17	1,0	NA			1,01	0,42	0,23	3,91	6,2
Biscoito, doce, recheado com chocolate	2,2	472	1974	6,4	19,6	Tr	70,5	3,0	1,3	27	48	9	0,59	139	2,3	239	232	0,27	1,0	Tr			0,32	0,39	0,46	2,52	3,5
Biscoito, doce, recheado com morango	2,7	471	1971	5,7	19,6	Tr	71,0	1,5	1,0	36	27	10	0,66	138	1,5	230	113	0,13	0,7	Tr			0,90	0,42	0,21	1,50	Tr
Biscoito, doce, wafer, recheado de chocolate	1,2	502	2102	5,6	24,7	Tr	67,5	1,8	1,1	23	48	11	0,44	124	2,4	137	240	0,26	0,9	Tr			0,37	0,03	0,30	1,12	Tr
Biscoito, doce, wafer, recheado de morango	1,2	513	2148	4,5	26,4	1	67,4	0,8	0,6	14	19	12	0,28	73	1,1	120	75	0,08	0,5	Tr			0,36	Tr	0,94	0,37	
Biscoito, salgado, cream cracker	4,1	432	1806	10,1	14,4	NA	68,7	2,5	2,7	20	40	13	0,69	148	2,2	854	181	0,18	1,1	NA			0,71	0,13	0,17	7,14	
Bolo, mistura para	1,0	419	1752	6,2	6,1	Tr	84,7	1,7	2,0	59	28	14	0,46	333	1,2	463	75	0,15	0,6	NA			0,18	Tr	1,50	1,52	Tr
Bolo, pronto, aipim	34,1	324	1355	4,4	12,7	73	47,9	0,7	0,8	85	10	15	0,11	122	0,5	111	135	0,05	0,4	Tr			0,13	0,07	Tr	Tr	Tr
Bolo, pronto, chocolate	19,3	410	1715	6,2	18,5	77	54,7	1,4	1,3	75	28	16	0,38	197	2,1	283	212	0,05	0,7	Tr			0,13	0,09	0,05	Tr	Tr
Bolo, pronto, coco	29,3	333	1395	5,7	11,3	63	52,3	1,1	1,4	57	16	17	0,40	303	0,8	190	143	0,09	0,7	Tr	Tr	Tr	0,15	0,06	Tr	Tr	Tr
Bolo, pronto, milho	36,7	311	1303	4,8	12,4	82	45,1	0,7	1,0	83	10	18	0,11	128	0,7	134	118	0,04	0,4	57			0,11	0,05	Tr	Tr	Tr
Canjica, branca, crua	13,6	358	1496	7,2	1,0	NA	78,1	5,5	0,2	2	12	19	0,09	48	0,3	1	93	0,05	0,4	NA			Tr	Tr	Tr	Tr	Tr
Canjica, com leite integral	72,5	112	471	2,4	1,2	1	23,6	1,2	0,3	43	6	20	0,02	41	0,1	28	70	0,03	0,3	Tr			Tr	0,04	Tr	Tr	Tr
Cereais, milho, flocos, com sal	9,3	370	1546	7,3	1,6	NA	80,8	5,3	1,0	2	20	21	Tr	91	0,5	272	69	Tr	0,6	NA			0,12	Tr	0,06	Tr	Tr
Cereais, milho, flocos, sem sal	11,2	363	1520	6,9	1,2	NA	80,4	1,8	0,3	2	17	22	Tr	58	1,7	31	29	0,24	0,3	NA			0,05	Tr	0,04	Tr	Tr
Cereais, mingau, milho, infantil	4,7	394	1650	6,4	1,1	NA	87,3	3,2	0,5	219	16	23	0,11	169	3,0	399	82	0,04	0,4	21			3,72	0,47	5,08	24,16	109,4
Cereais, mistura para vitamina, trigo, cevada e aveia	4,4	381	1595	8,9	2,1	NA	81,6	5,0	3,0	584	72	24	2,28	515	12,6	1163	244	0,21	2,0	Tr			0,76	0,88	1,49	7,46	13,1
Cereal matinal, milho	5,5	365	1529	7,2	1,0	NA	83,8	4,1	2,5	143	11	25	0,06	101	3,1	655	83	0,06	7,6	NA	31	15	0,76	1,02	2,25	11,04	17,3
Cereal matinal, milho, açúcar	4,3	377	1576	4,7	0,7	NA	88,8	2,1	1,5	56	8	26	0,13	43	3,9	405	52	0,04	8,5	NA	31	15	0,73	1,11	0,79	10,13	14,6
Creme de arroz, pó	7,3	386	1615	7,0	1,2	NA	83,9	1,1	0,5	7	51	27	1,24	153	0,6	1	115	0,04	1,9	NA			0,22	Tr	0,05	Tr	Tr
Creme de milho, pó	5,7	333	1393	4,8	1,6	NA	86,1	3,7	1,7	323	30	28	0,20	303	4,3	594	166	0,04	0,8	NA			0,74	Tr	2,05	Tr	96,3
Curau, milho verde	81,6	78	328	2,4	1,6	5	13,9	0,5	0,5	53	16	29	0,06	75	0,4	21	162	0,03	0,4	12	20	16	0,04	0,07	Tr	Tr	Tr
Curau, milho verde, mistura para	3,9	402	1683	2,2	13,4	NA	79,8	2,5	0,7	31	9	30	0,05	45	0,9	223	55	0,03	0,2	NA			0,13	Tr	Tr	Tr	Tr
Farinha, de arroz, enriquecida	12,7	363	1519	1,3	0,3	NA	85,5	0,6	0,2	1	4	31	0,04	36	31,4	17	13	Tr	8,5	NA			3,23	Tr	3,47	24,42	173,6

	Umidade	Energia		Proteína	Lipídeos	Colesterol	Carbo-idrato	Fibra Alimentar	Cinzas	Cálcio	Magnésio	Número do Alimento	Manganês	Fósforo	Ferro	Sódio	Potássio	Cobre	Zinco	Retinol	RE	RAE	Tiamina	Riboflavina	Piridoxina	Niacina	Vitamina C
Descrição dos alimentos	(%)	(kcal)	(kJ)	(g)	(g)	(mg)	(g)	(g)	(g)	(mg)	(mg)		(mg)	(mg)	(mg)	(mg)	(mg)	(mg)	(mg)	(mcg)	(mcg)	(mcg)	(mg)	(mg)	(mg)	(mg)	(mg)
Farinha, de centeio, integral	10,8	338	1405	12,5	1,8	NA	73,3	15,5	1,7	34	120	32	3,86	340	4,7	41	334	0,56	2,7	NA			0,29	0,03	0,08	Tr	Tr
Farinha, de milho, amarela	11,8	351	1467	7,2	1,5	NA	79,1	5,5	0,5	1	31	33	Tr	84	2,3	45	58	0,27	0,6	NA	18	9	0,25	Tr	0,25	Tr	Tr
Farinha, de rosca	9,8	371	1550	11,4	1,5	NA	75,8	4,8	1,6	35	57	34	1,62	195	6,7	333	212	0,26	1,7	NA			0,25	Tr	0,09	Tr	Tr
Farinha, de trigo	13,0	360	1508	9,8	1,4	NA	75,1	2,3	0,8	18	31	35	0,46	115	1,0	1	151	0,15	0,8	NA			0,31	Tr	Tr	0,89	Tr
Farinha, láctea, de cereais	2,7	415	1736	11,9	5,8	11	77,8	1,9	1,9	196	58	36	1,49	296	8,7	125	368	0,19	1,7	492			1,43	1,13	1,14	9,50	24,3
Lasanha, massa fresca, cozida	59,6	164	685	5,8	1,2	NA	32,5	1,6	0,9	10	4	37	0,20	42	1,2	207	54	0,07	0,4	NA			0,04	Tr	Tr	3,96	
Lasanha, massa fresca, crua	45,0	220	922	7,0	1,3	NA	45,1	1,6	1,6	17	13	38	0,34	82	1,9	667	137	0,10	0,8	NA			0,08	Tr	Tr	Tr	
Macarrão, instantâneo	6,0	436	1824	8,8	17,2	NA	62,4	5,6	5,6	18	19	39	0,25	112	0,8	1516	148	0,10	0,5	NA			1,18	0,04	0,53	9,37	Tr
Macarrão, trigo, cru	10,2	371	1553	10,0	1,3	NA	77,9	2,9	0,5	17	28	40	0,53	100	0,9	7	147	0,15	0,8	NA			0,18	0,02	Tr	3,57	Tr
Macarrão, trigo, cru, com ovos	10,6	371	1550	10,3	2,0	18	76,6	2,3	0,5	19	Tr	41	0,40	118	0,9	15	134	0,14	0,8	Tr			0,11	0,05	0,03	4,37	Tr
Milho, amido, cru	12,2	361	1512	0,6	Tr	NA	87,1	0,7	0,1	1	3	42	0,02	13	0,1	8	9	0,02	0,1	NA			Tr	Tr	Tr	Tr	Tr
Milho, fubá, cru	11,5	353	1479	7,2	1,9	NA	78,9	4,7	0,6	3	41	43	0,34	108	0,9	Tr	168	0,08	1,1	NA	26	13	0,25	Tr	Tr	0,75	Tr
Milho, verde, cru	63,5	138	578	6,6	0,6	NA	28,6	3,9	0,7	2	33	44	0,12	113	0,4	1	185	0,05	0,5	NA	32	16	0,30	Tr	0,04	Tr	Tr

CMVCol taco3 | AGtaco3 | Aminoácidos TACO3

Figura 2.1: Print da tabela Taco

2.3 Fundamentos em tecnologia

O desenvolvimento de sistemas para restaurantes envolve diversas tecnologias e metodologias. Nesta seção, algumas dessas tecnologias são apresentadas.

2.3.1 Prototipação

Um protótipo é uma amostra ou modelo de trabalho que realiza a simulação de um produto real, tendo como objetivo apresentar ao cliente as funcionalidades do produto, sua interface e as jornadas do usuário.

Em relação à interface do usuário, a classificação pode se dar em termos de fidelidade, ou seja, o grau de similaridade entre o protótipo e a interface final. Os protótipos podem ser classificados em baixa fidelidade e alta fidelidade (OLIVEIRA et al., 2007).

Protótipos de baixa fidelidade são aqueles que apresentam grande diferença em relação ao produto final, possuindo poucos detalhes e representando apenas as funcionalidades principais.

Já os protótipos de alta fidelidade possuem grande proximidade com o produto final, com foco nos aspectos visuais e na imersão do usuário, permitindo que ele navegue como se estivesse utilizando o produto definitivo (LIMA; NASCIMENTO; NETO, 2025).

O Figma é uma plataforma colaborativa para construção de design de interfaces e prototipação. É utilizada para criar o design de produtos digitais, como sites e aplicativos para dispositivos móveis, oferecendo diversas funcionalidades que otimizam o trabalho e contribuem para o desenvolvimento do produto (FIGMA, 2025).

2.3.2 API

A API (*Application Programming Interface*) é um mecanismo que permite que dois componentes de software se comuniquem por meio de um conjunto de definições e protocolos. A arquitetura da API é explicada em termos de cliente e servidor: a aplicação que envia a solicitação é chamada de cliente, enquanto a aplicação que envia a resposta é denominada servidor.

Com o avanço do consumo de serviços interconectados pela web, o uso de APIs tornou-se cada vez mais comum e passou por constantes aprimoramentos ao longo dos anos (CANCIAN R. L.; CANAL, 2022). As APIs possibilitam a interoperabilidade entre diferentes sistemas, promovendo a comunicação eficiente entre aplicações e usuários (IBM, 2017). Seu funcionamento baseia-se na disponibilização de pontos de acesso que permitem a troca de informações entre aplicações e seus respectivos clientes, sejam eles usuários finais ou outras aplicações.

As APIs desempenham um papel relevante, pois são responsáveis por conectar serviços, dispositivos e aplicações que operam de forma distribuída. Dessa forma, assumem um papel essencial na integração, automação e escalabilidade dos sistemas, promovendo a interoperabilidade, ou seja, a capacidade de diferentes plataformas e tecnologias trabalharem em conjunto.

No contexto das APIs RESTful, os recursos do sistema são representados por URLs e manipulados por meio dos métodos HTTP, como GET (leitura), POST (criação), PUT (atualização) e DELETE (exclusão). As respostas geralmente são retornadas no formato JSON, que é legível e amplamente suportado por diversas linguagens de programação.

2.3.3 Desenvolvimento de uma API

O desenvolvimento de uma API envolve um conjunto de requisitos estruturais e técnicos que irão garantir seu funcionamento correto, desempenho adequado e facilidade de manutenção. Com o levantamento de requisitos funcionais e não funcionais, é possível descrever as funcio-

nalidades e serviços do sistema, bem como definir suas propriedades e restrições (TURINE M. A.; MASIERO, 1996).

A arquitetura de uma API é essencial para garantir organização e padronização na comunicação entre sistemas. A modelagem de sistemas, tanto no nível funcional quanto no nível de dados, é um requisito fundamental para a obtenção de produtos de software com maior qualidade e confiabilidade (ARAúJO, 2008). Uma modelagem bem estruturada auxilia na definição de entidades, atributos e relacionamentos que serão armazenados no banco de dados, facilitando a implementação e manutenção da API.

A API deve ser integrada a um sistema de gerenciamento de banco de dados responsável pelo armazenamento e recuperação dos dados. Para a definição do banco de dados, devem ser considerados fatores como volume de dados, desempenho, escalabilidade, facilidade de manutenção e integração com o sistema que será desenvolvido.

Para o desenvolvimento da API, será utilizado o framework *Spring Boot*, amplamente empregado na criação de aplicações web e serviços REST em Java. O *Spring Boot* facilita a configuração e o desenvolvimento de aplicações, oferecendo recursos que simplificam a integração com bancos de dados, gerenciamento de dependências e criação de APIs de forma mais rápida e organizada. Dessa forma, sua utilização contribui para tornar o processo de desenvolvimento mais eficiente e estruturado.

A implementação da API envolve o desenvolvimento das rotas, controladores e serviços responsáveis pelo processamento das requisições. Cada *endpoint* é definido de forma objetiva, especificando o método HTTP correspondente e sua finalidade dentro do sistema.

Para a troca de dados entre o *front-end* e o *back-end*, são utilizados os DTOs (Data Transfer Objects) , que têm como objetivo transportar apenas as informações necessárias para cada requisição e resposta da API, evitando o acoplamento direto com as entidades do banco de dados. A Figura 2.2 apresenta o fluxo de comunicação entre o usuário, o *front-end*, a API e o banco de dados, destacando os principais componentes responsáveis pelo processamento das requisições e pelo tratamento dos dados.

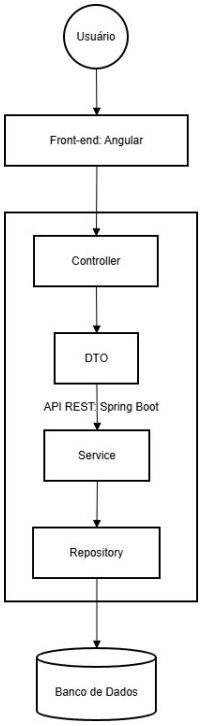


Figura 2.2: Fluxo de comunicação.

Os controladores são responsáveis por receber as requisições, validar os dados por meio dos DTOs e encaminhá-los para a camada de serviços, onde são executadas as regras de negócio, como o cálculo dos valores nutricionais. Ao final do processamento, as respostas são estruturadas novamente em DTOs, retornando ao *front-end* apenas os dados essenciais para a exibição da tabela nutricional.

A segurança de APIs consiste em um conjunto de estratégias voltadas para monitorar, testar e proteger os recursos disponibilizados (Okta, Inc., 2023). Trata-se de um requisito essencial, garantindo mecanismos de autenticação e autorização que asseguram que apenas usuários devidamente autorizados tenham acesso aos recursos do sistema.

A documentação de uma API corresponde ao conjunto de informações que descrevem como interagir com determinado software. Essa documentação pode incluir os *endpoints* disponíveis, os parâmetros aceitos, os métodos HTTP utilizados e os tipos de dados que a API espera ou retorna (OWASP API Security Project, 2019). A Tabela 2.2 apresenta os principais códigos de retorno HTTP e seus significados.

Código HTTP	Significado
201	Created: Requisição bem-sucedida que resultou na criação de um recurso
200	OK: Requisição bem-sucedida, o recurso foi retornado ou atualizado conforme esperado
401	Unauthorized: Token ausente ou inválido, usuário não autenticado
403	Forbidden: Usuário autenticado, mas não possui permissão para acessar o recurso
404	Not Found: Recurso solicitado não encontrado

Tabela 2.2: Códigos HTTP

2.3.4 *Framework Spring Boot*

Algumas definições atribuídas aos *frameworks* indicam que eles consistem em uma arquitetura desenvolvida com o objetivo de atingir a máxima reutilização de código, sendo representados como um conjunto de classes abstratas e concretas com grande potencial de especialização (PINTO, 2006).

O *Spring Boot* é um *framework* que facilita a criação de aplicações baseadas no ecossistema *Spring*, devido à sua abordagem opinativa sobre a plataforma e sobre bibliotecas de terceiros. Essa característica permite que o desenvolvedor invista o mínimo de tempo possível com configurações iniciais do projeto, tornando o processo de desenvolvimento mais ágil e produtivo (JUNIOR E. A.; ROCHA, 2021).

No contexto do desenvolvimento de sistemas, o *Spring Boot* destaca-se por oferecer uma estrutura robusta, escalável e de fácil manutenção. Entre seus principais recursos estão a injeção de dependências, o gerenciamento de componentes e o suporte à criação de *endpoints* para APIs REST. Essas funcionalidades tornam o *framework* amplamente utilizado em projetos modernos que buscam eficiência, modularidade, segurança e organização arquitetural.

2.3.5 *Front-end*

O termo *front-end* refere-se à interface gráfica do usuário, com a qual as pessoas interagem, sendo responsável pela interface e pela experiência do usuário, conceito central da Interação Humano-Computador (KOCAMAN, 2021). Essa camada é construída com tecnologias como HTML, CSS e JavaScript, tecnologias base da web que estruturam, estilizam e adicionam interatividade às páginas.

Alguns *frameworks* são utilizados para otimizar o tempo durante o desenvolvimento.

Um *framework* pode ser definido como uma estrutura base para o desenvolvimento de software de maneira mais rápida e eficiente. Ele define a arquitetura do projeto e fornece ferramentas necessárias que podem ser utilizadas como blocos de construção, podendo incluir bibliotecas e módulos, padrões de *design*, ferramentas de desenvolvimento e padrões de codificação.

O Angular é um *framework front-end* baseado em TypeScript, desenvolvido pelo Google, amplamente utilizado para a criação de aplicações web modernas e dinâmicas. Sua principal característica é a arquitetura baseada em componentes, que permite a construção de interfaces modulares, reutilizáveis e de fácil manutenção. Além disso, o Angular oferece recursos integrados, como injeção de dependências, roteamento, validação de formulários e comunicação com APIs, o que simplifica o desenvolvimento e torna o processo mais eficiente. Graças a essas funcionalidades, o *framework* possibilita a criação de interfaces responsivas e interativas, melhorando significativamente a experiência do usuário e garantindo que o sistema funcione de maneira fluida e organizada. Entre seus principais benefícios, destacam-se o melhor desempenho, um sistema de templates mais completo, maior suporte para testes e a possibilidade de utilização de componentes aninhados (BAGLIOTTI I. R.; GIBERTONI, 2020).

2.3.6 Banco de Dados

O banco de dados é uma coleção sistemática de dados armazenados eletronicamente (ELMASRI R.; NAVATHE, 2005). Pode conter diferentes tipos de informações, incluindo palavras, números, imagens, vídeos e arquivos, podendo ser considerado o equivalente eletrônico de um armário de arquivamento, ou seja, um repositório para uma coleção de arquivos de dados computadorizados (DATE, 2003). Um banco de dados é crucial para qualquer organização, pois suporta as operações internas da empresa, mantém informações administrativas e armazena dados especializados, como modelos econômicos ou de engenharia. Além disso, é essencial por diversos fatores, como escalabilidade eficiente, integridade dos dados, segurança da informação e análise de dados.

Bancos de dados relacionais e não relacionais representam dois métodos distintos de armazenamento de dados para aplicações. Um banco de dados relacional organiza as informações em formato tabular, estruturado em linhas e colunas. Por outro lado, os bancos de dados não relacionais utilizam diferentes modelos de dados para armazenar e gerenciar informações, podendo organizá-las como coleções de pares chave-valor, nas quais as chaves e os valores podem variar desde objetos simples até estruturas compostas e complexas.

O PostgreSQL é um sistema de gerenciamento de banco de dados objeto-relacional de código aberto, desenvolvido pelo Departamento de Ciência da Computação da Universidade da Califórnia, em Berkeley (PostgreSQL Global Development Group, 2025). A conformidade com os padrões SQL é uma de suas principais características, garantindo portabilidade e interoperabilidade entre diferentes sistemas.

O PostgreSQL é utilizado tanto em contextos acadêmicos quanto empresariais, sendo aplicado em sistemas corporativos, aplicações web e projetos de pesquisa, entre outros. Por ser um software de código aberto, contribui para a redução de custos com licenciamento e estimula sua adoção por organizações de diferentes portes (SILVA, 2023).

2.3.7 Back-end

O termo *back-end* refere-se à camada responsável pelo processamento das regras de negócio, comunicação com o banco de dados e gerenciamento das operações realizadas pelo sistema. Diferentemente do *front-end*, que está relacionado à interface com o usuário, o *back-end* atua nos bastidores da aplicação, garantindo que as funcionalidades solicitadas pelo usuário sejam executadas corretamente (SOMMERVILLE, 2019).

Essa camada é responsável por receber requisições, processar dados, aplicar regras de negócio e retornar respostas para o *front-end*. Para isso, são utilizadas linguagens de programação, *frameworks* e arquiteturas que permitem organizar o código, garantir segurança e facilitar a manutenção do sistema. Em aplicações web modernas, o *back-end* também é responsável pela criação de APIs, que permitem a comunicação entre diferentes sistemas e serviços.

Algumas definições atribuídas aos *frameworks* indicam que eles consistem em uma arquitetura desenvolvida com o objetivo de atingir a máxima reutilização de código, sendo representados como um conjunto de classes abstratas e concretas com grande potencial de especialização (PINTO, 2006).

Nesse contexto, destaca-se o *Spring Boot*, um *framework* que facilita a criação de aplicações baseadas no ecossistema *Spring*, devido à sua abordagem opinativa sobre a plataforma e sobre bibliotecas de terceiros. Essa característica permite que o desenvolvedor invista o mínimo de tempo possível com configurações iniciais do projeto, tornando o processo de desenvolvimento mais ágil e produtivo (JUNIOR E. A.; ROCHA, 2021).

No contexto do desenvolvimento de sistemas, o *Spring Boot* destaca-se por oferecer uma estrutura robusta, escalável e de fácil manutenção. Entre seus principais recursos estão a injeção

de dependências, o gerenciamento de componentes e o suporte à criação de *endpoints* para APIs REST. Essas funcionalidades tornam o *framework* amplamente utilizado em projetos modernos que buscam eficiência, modularidade, segurança e organização arquitetural.

CAPÍTULO 3

ANÁLISE E PROJETO DO SISTEMA

Nesta seção, são detalhadas as etapas iniciais para dar início ao desenvolvimento do sistema. Por conseguinte, o processo foi estruturado em etapas que incluem levantamento de requisitos, análise de requisitos, implementação, testes, correções e entrega, que serão descritos nas próximas seções.

3.1 Levantamento e Análise de Requisitos

O levantamento de requisitos, no contexto deste trabalho, foi direcionado ao desenvolvimento de uma API voltada para restaurantes, com ênfase no cardápio digital que apresenta informações nutricionais.

A análise levou em conta diferentes perfis de usuários, como gestores de restaurantes e clientes finais. Foram feitas entrevistas com gestores de restaurantes, com o objetivo de entender as principais dificuldades e demandas enfrentadas no gerenciamento dos sistemas, com foco nos cardápios e na disponibilização das informações aos clientes. A partir dessas entrevistas, foi possível identificar a necessidade de incluir informações nutricionais nos pratos e algumas necessidades relacionadas à otimização do sistema.

Além das entrevistas, foi realizada uma pesquisa com sistemas já utilizados no mercado, como plataformas de cardápio digital e sistemas de pedidos. A partir da análise desses sistemas, foi possível identificar funcionalidades comuns, como visualização de cardápio e organização por categorias. Também foram observadas algumas limitações ou a apresentação superficial de algumas informações.

Com todas as informações coletadas na pesquisa dos sistemas já existentes e nas entre-

vistas, foi possível definir requisitos mais alinhados às necessidades dos usuários. Esses dados contribuíram significativamente, auxiliando diretamente na especificação das funcionalidades da API e do painel do cliente, buscando oferecer uma solução que atenda tanto gestores quanto clientes.

Por último, foi consultada uma nutricionista, a fim de garantir a correta interpretação da tabela TACO.

3.1.1 Atores

- **Cliente**: Pode efetuar login no sistema, reservar mesa, ler o cardápio e realizar pedidos. O caso de uso "Realizar pedidos"é composto pelas ações de anotar pedido, listar pedido e cancelar pedido.
- **Garçom**: Possui acesso às funcionalidades de efetuar login, anotar pedido, listar pedido, cancelar pedido e atualizar o status da mesa.
- **Caixa**: É responsável por efetuar login, atualizar o status da mesa, finalizar pedidos e registrar pagamentos.
- **Chef de Cozinha**: Tem permissão para efetuar login, iniciar a preparação dos pedidos e finalizar a preparação quando concluída.
- **Gerente**: Possui todas as funcionalidades do garçom. Além disso, pode verificar os pedidos realizados, manter o cardápio do restaurante e gerenciar o cadastro de funcionários.

3.1.2 Requisitos do sistema

Os principais requisitos funcionais do sistema foram definidos conforme a tabela 3.1

Tabela 3.1: Requisitos Funcionais

Código	Identificação	Classificação	Ator	Objetivo
RF01	Efetuar login	Essencial	Cliente	Permitir que o cliente acesse o sistema para fazer o pedido
RF02	Efetuar login	Essencial	Garçom, chef de cozinha, caixa e gerente	Permitir que efetuem login para acessar funcionalidades conforme seu perfil
RF03	Ler cardápio	Essencial	Cliente	Permitir que o cliente visualize o cardápio
RF04	Realizar pedidos	Essencial	Cliente/Garçom	Permitir que o cliente faça pedidos

Código	Identificação	Classificação	Ator	Objetivo
RF05	Anotar pedidos	Essencial	Garçom	Permitir que o garçom anote o pedido do cliente
RF06	Listar pedidos	Essencial	Garçom	Permitir que o garçom visualize os pedidos realizados
RF07	Cancelar pedido	Opcional	Garçom	Permitir que o garçom cancele um pedido em caso de necessidade
RF08	Modificar pedido	Opcional	Garçom	Permitir que o garçom altere pedidos realizados
RF09	Atualizar status da mesa	Essencial	Garçom	Permitir que o garçom atualize o status da mesa (livre, ocupada, aguardando pagamento)
RF10	Iniciar preparação	Essencial	Chef de cozinha	Permitir que o chef inicie a preparação do pedido
RF11	Finalizar preparação	Essencial	Chef de cozinha	Permitir que o chef finalize a preparação do pedido
RF12	Finalizar pedido	Essencial	Garçom/Cliente	Permitir que o garçom finalize o pedido do cliente após a refeição
RF13	Finalizar conta	Essencial	Garçom/Cliente	Permitir que o garçom finalize a conta do cliente
RF14	Registrar pagamento	Essencial	Caixa	Permitir que o caixa registre o pagamento realizado pelo cliente
RF15	Visualizar informações nutricionais	Essencial	Cliente	Permitir que o cliente visualize informações nutricionais dos alimentos
RF16	Manter cardápio	Essencial	Gerente	Permitir que o gerente atualize, adicione ou remova itens do cardápio
RF17	Verificar pedidos	Essencial	Gerente	Permitir que o gerente acompanhe os pedidos realizados
RF18	Manter funcionário	Essencial	Gerente	Permitir que o gerente cadastre, edite ou remova funcionários no sistema
RF19	Reservar mesa	Essencial	Cliente	Permitir que o cliente reserve uma mesa no restaurante
RF20	Aplicar Filtros Alimentares Personalizados	Essencial	Cliente	Permitir que o cliente filtre os alimentos conforme suas preferências ou restrições alimentares

3.1.3 Requisitos não funcionais

Tabela 3.2: Requisitos Não Funcionais

Código	Identificação	Classificação	Objetivo
RNF01	Usabilidade	Essencial	Interface deve ser intuitiva e fácil de usar para todos os usuários.
RNF02	Performance	Essencial	O sistema deve carregar as páginas em até 3 segundos.
RNF03	Segurança	Essencial	As informações dos usuários devem ser armazenadas de forma segura.
RNF04	Portabilidade	Opcional	O sistema deve funcionar em dispositivos móveis e desktops.
RNF05	Confiabilidade	Essencial	O sistema deve garantir 99% de disponibilidade.

3.1.4 Regras de Negócio - MenuNutri

1. O cliente pode acessar o aplicativo de qualquer lugar; porém, só poderá efetuar um pedido se estiver autenticado no sistema. Caso não esteja logado, o sistema exigirá login antes de permitir a criação do pedido.

2. Para criar um pedido, o cliente deverá informar o número da mesa. O sistema vincula automaticamente a mesa ao pedido no momento da criação. Caso a mesa não exista, o pedido não será criado.

3. O cliente só poderá possuir um pedido com status **ABERTO** por vez. Caso já exista um pedido em aberto, o sistema impedirá a criação de um novo pedido.

4. Após a criação, o pedido deverá seguir obrigatoriamente o fluxo de status definido pelo sistema, respeitando a seguinte sequência: **ABERTO** → **AUTORIZADO** → **EM_ANDAMENTO** → **PRONTO** → **FINALIZADO** → **PAGO**. Não é permitido alterar o status fora da ordem estabelecida.

5. O cliente só poderá adicionar ou remover itens enquanto o pedido estiver com status **ABERTO**. Após a finalização do pedido, não serão permitidas alterações.

6. O pagamento é realizado no caixa. Após a confirmação do pagamento, o pedido é marcado como **PAGO** e a mesa é liberada automaticamente pelo sistema.

7. As informações nutricionais dos pratos são calculadas automaticamente com base na quantidade dos ingredientes cadastrados no sistema, garantindo que os valores exibidos sejam proporcionais à porção servida.

8. O sistema estabelece um limite máximo de três reservas ativas por cliente, a fim de evitar o bloqueio excessivo de mesas e garantir maior disponibilidade para outros clientes.

3.1.5 Diagrama de Casos de Uso

Os diagramas *Unified Modeling Language* (UML) utilizados no desenvolvimento de sistemas são ferramentas visuais. Eles auxiliam na compreensão do funcionamento do software, facilitando a comunicação entre desenvolvedores e analistas envolvidos no projeto.

O diagrama de caso de uso mostrado na Figura 3.1 tem como objetivo representar as principais funcionalidades do sistema e todas as interações entre o usuário e o sistema. No diagrama desenvolvido são apresentados seis atores principais: cliente, funcionário, garçom, chefe de cozinha, gerente e caixa.

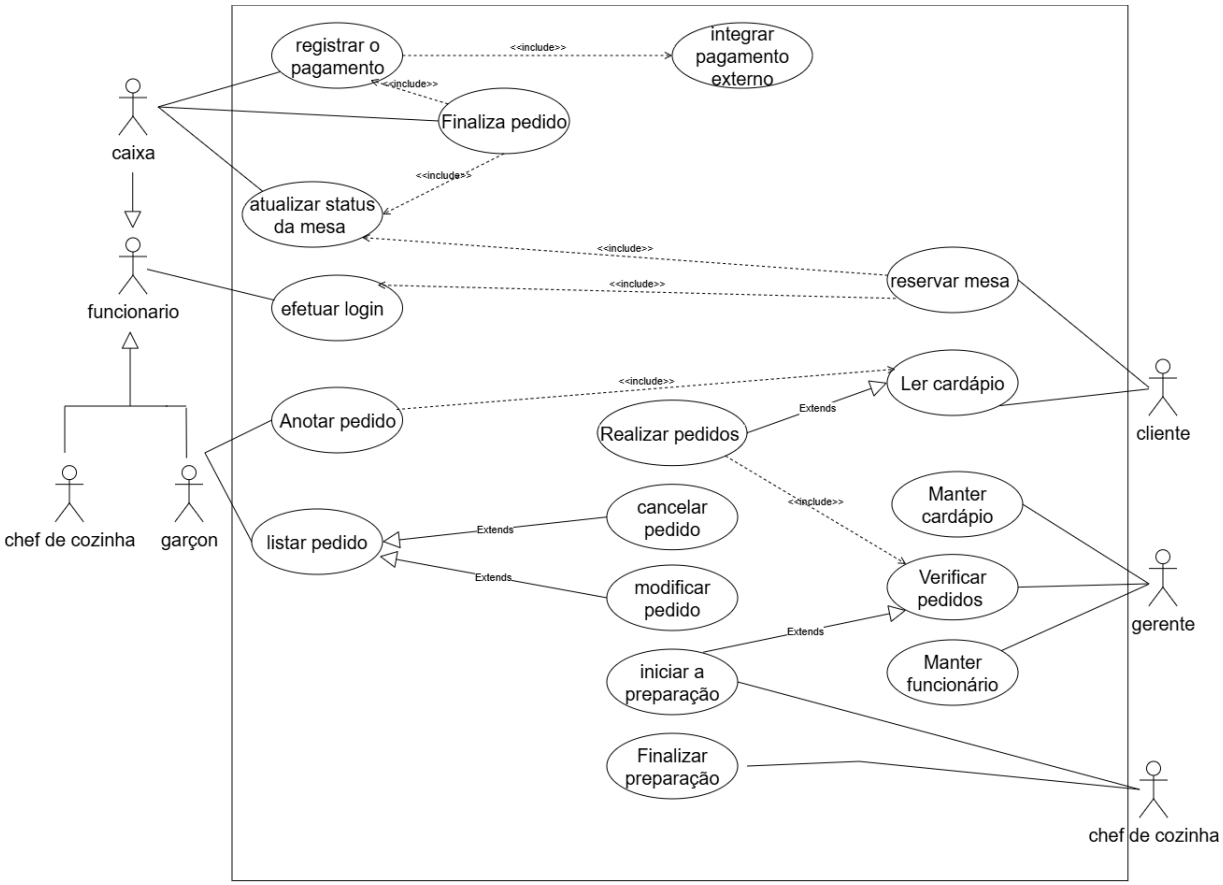


Figura 3.1: Diagrama de casos de uso.

A partir dos requisitos funcionais identificados, foram definidos diversos casos de uso

para representar as interações entre os atores e o sistema. Neste trabalho, são apresentados apenas os principais casos de uso do sistema, selecionados por representarem as funcionalidades centrais da aplicação, diretamente ligadas ao painel do cliente.

UC01 - Realizar Login

Nome do caso de uso	UC01 - Realizar Login
Ator principal	Cliente
Atores secundários	-
Resumo	O usuario realiza login atravez do google.
pré-condições	Usuário deve ter uma conta do google ativa.
pós-condições	O usuario deve estar autenticado para acessar as funcionalidades restritas do sistema.
	Cenário Principal
Ações do ator	Ações do sistema
1) O usuario acessa a tela de login	2) Seleciona "login com google"
	3) Insere as credenciais
4) O sistema valida os dados	5) O sistema autentica o usuario e redireciona para a tela principal

UC02 - Ler cardápio

Nome do caso de uso	UC02 - Ler cardapio
Ator principal	Cliente
Atores secundários	-
Resumo	Permite que o cliente visualize o cardápio com categorias, pratos, descrições e informações nutricionais.
pré-condições	
pós-condições	O cliente consegue visualizar os detalhes dos pratos e informações nutricionais.
	Cenário Principal
Ações do ator	Ações do sistema
1) Acessa a tela principal do sistema	2) O sistema lista os pratos separados por categoria
3) Seleciona o prato desejado	
	4) O sistema exibe as informações do prato
	Cenário Alternativo
5) Seleciona "informações nutricionais"	6) O sistema exibe informações nutricionais do prato

UC03 - Realizar Pedido

Nome do caso de uso	UC03 - Realizar Pedido
Ator principal	Cliente
Atores secundários	Funcionarios e Garçom
Resumo	Permite que o cliente selecione pratos e envie um pedido ao sistema para preparo.
pré-condições	O cliente precisa estar logado no sistema.
pós-condições	O pedido é registrado no sistema e enviado ao gerente para ser avaliado e encaminhado para a cozinha.
	Cenário Principal
Ações do ator (Cliente)	Ações do sistema
1) Executa o caso de uso "ler cardápio"	2) Adiciona o prato ao pedido
3) Visualiza a atualização da lista de itens do pedido	4) Atualiza a lista de itens do pedido
5) Verifica os itens e adiciona observações	6) Exibe a lista de itens atualizada
7) Confirma o pedido	8) Registra o pedido e o envia ao gerente
	Cenário Alternativo I
Ações do ator (Cliente)	Ações do sistema
8) Seleciona a opção "Adicionar mais itens"	9) O sistema executa novamente o caso de uso "Ler cardápio"
	Cenário Alternativo II
Ações do ator (Cliente)	Ações do sistema
10) Tenta realizar o pedido sem estar logado	11) O sistema solicita autenticação e executa o caso de uso "Realizar login"

UC04 - Reservar Mesa

Nome do caso de uso	UC02 - Rersevar Mesa
Ator principal	Cliente
Atores secundários	
Resumo	Permite que o cliente selecione a data, a hora e o número de pessoas para reservar uma mesa no restaurante.
pré-condições	O cliente deve estar autenticado no sistema.
pós-condições	A mesa é reservada e resgistrada no sistema para a data e hora selecionadas.
	Cenário Principal
Ações do ator (Cliente)	Ações do sistema
1) Acessa a funcionalidade "Reservar Mesa"	2) Exibe o formulário de reserva
3) Informa a data, hora e número de pessoas	4) Recebe e valida as informações inseridas
5) Confirma os dados da reserva	6) Verifica a disponibilidade de mesas
	7) Efetua a reserva e apresenta mensagem de sucesso ao cliente
	Cenário Alternativo I
	Cenário Alternativo I - Mesa indisponível
Ações do ator (Cliente)	Ações do sistema
6) Informa a data, hora e número de pessoas	7) Verifica disponibilidade e detecta que não há mesas disponíveis
	8) Informa a indisponibilidade e solicita que o cliente escolha outra data ou horário
	Cenário Alternativo II - Cliente não autenticado
Ações do ator (Cliente)	Ações do sistema
9) Tenta acessar a funcionalidade "Reservar Mesa"	10) Detecta que o cliente não está autenticado
	11) Solicita autenticação e executa o caso de uso "Realizar Login"

3.1.6 Diagrama Entidade-Relacionamento

O diagrama entidade-relacionamento é a representação visual da estrutura lógica de um banco de dados. Esse modelo auxilia no entendimento da organização das informações e serve como base para a criação do modelo lógico do banco de dados. A estrutura do banco demonstra as entidades envolvidas e seus relacionamentos.

No contexto do projeto, o diagrama da Figura 3.2 representa a estrutura do banco de dados, mostrando as entidades e seus relacionamentos. A entidade **Pessoa** é generalizada em

Cliente e **Funcionário**, indicando atributos comuns a ambos. O cliente realiza reservas de mesas, enquanto o funcionário é responsável por gerenciar pedidos e atualizar o status das mesas. Esse modelo facilita a organização dos dados e garante a consistência das informações utilizadas pela aplicação.

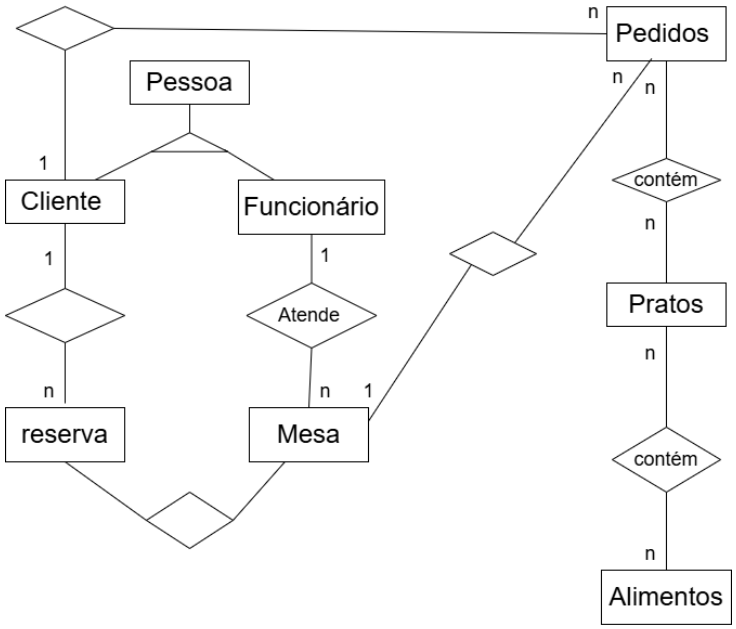


Figura 3.2: Diagrama entidade e relacionamento.

3.1.7 Diagrama de Atividades

O diagrama de atividades é definido como uma representação visual utilizada para ilustrar o fluxo contínuo de atividades, ações e processos no sistema.

No diagrama, o garçom recebe o pedido e o encaminha para o gerente. Depois de receber e verificar o pedido, o gerente autoriza a solicitação. Com o pedido autorizado, a cozinha recebe a solicitação e inicia o preparo. Após a finalização, a cozinha encaminha o pedido para o garçom, que, por sua vez, leva o pedido até o cliente.

Após receber o pedido, o cliente pode solicitar o fechamento da conta. Os pedidos feitos pelo cliente são direcionados para o caixa, que realiza o processo de pagamento e finaliza o atendimento.

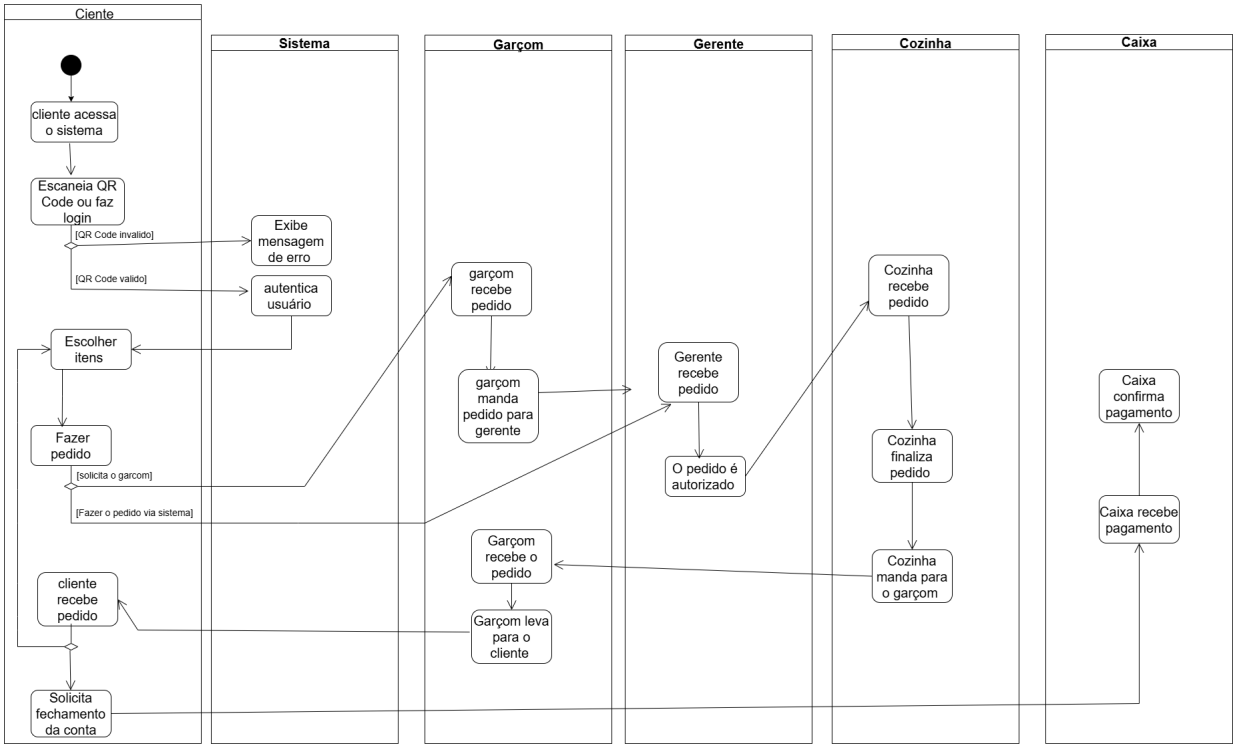


Figura 3.3: Diagrama de atividade.

3.1.8 Diagrama de Sequência

O diagrama de sequência representa a interação entre diferentes componentes e atores do sistema ao longo do tempo, sendo fundamental para compreender os fluxos de comunicação. Ele permite visualizar a ordem em que as ações acontecem e como cada elemento contribui para o processo.

O primeiro diagrama refere-se à funcionalidade *Realizar Pedido* (Figura 3.4). Nesse processo, o cliente inicia a ação de fazer pedido e o sistema apresenta o cardápio disponível.

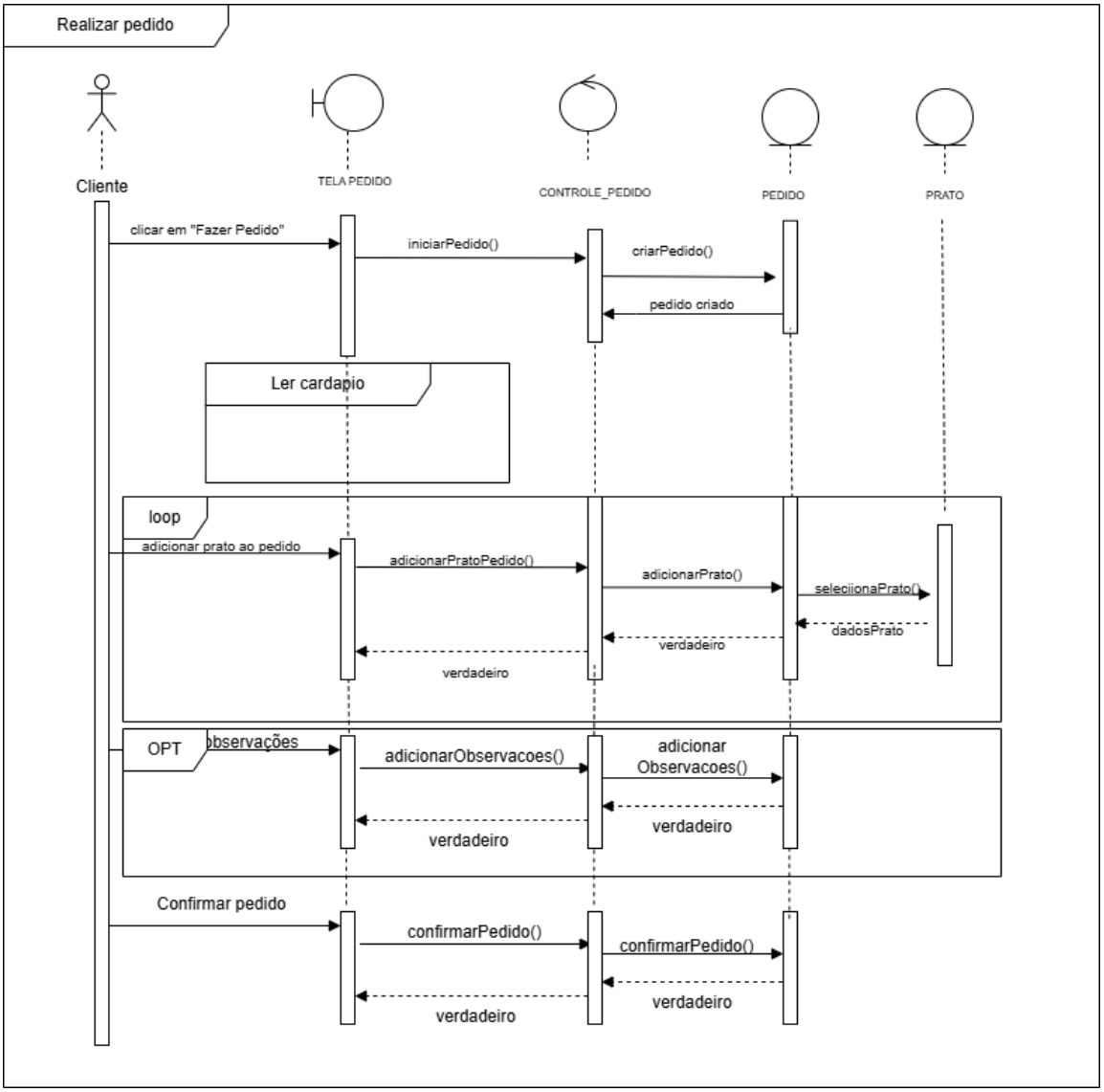


Figura 3.4: Diagrama de sequencia.

O diagrama da Figura 3.5 representa o processo de reserva realizado pelo cliente. Ao selecionar a mesa, o sistema verifica sua disponibilidade e também checa se o cliente já possui outras reservas, impedindo que sejam reservadas mais de três mesas.

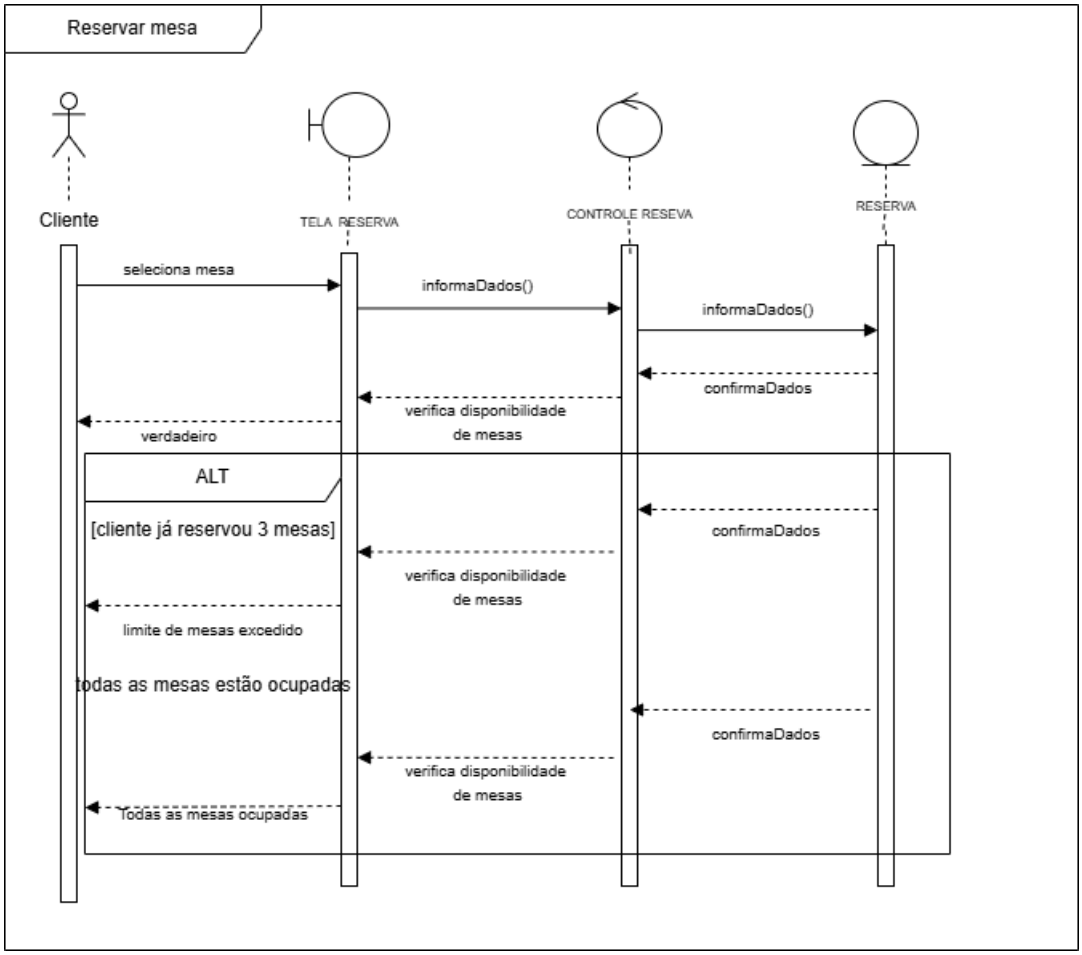


Figura 3.5: Diagrama de sequencia reserva.

3.1.9 Diagrama de Classes

O diagrama de classes é uma representação visual que descreve a estrutura estática de um sistema, apresentando suas classes, atributos, métodos e os relacionamentos entre elas. Ele facilita a comunicação entre desenvolvedores e analistas, além de servir como guia para o desenvolvimento do software.

A Figura 3.6 representa a modelagem estrutural do sistema de gerenciamento de restaurante. As principais entidades do domínio são **Usuário, Cliente, Funcionário, Pedido, ItemPedido, Prato, Ingrediente, Reserva e Mesa**, além de seus respectivos relacionamentos.

A classe **Usuário** é especializada em **Cliente** e **Funcionário**, permitindo o controle por meio do tipo de perfil. O **Pedido** é composto por um ou mais **ItemPedido**, estabelecendo uma

relação de composição, em que cada item está associado a um prato. Já na parte de gestão de **Reservas**, estas são vinculadas ao **Cliente** e às **Mesas**, possibilitando o controle de ocupação e disponibilidade.

Foram utilizadas enumerações para representar estados e classificações do sistema, como status de pedido, status de reserva, status de mesa, categoria de prato e restrições alimentares.

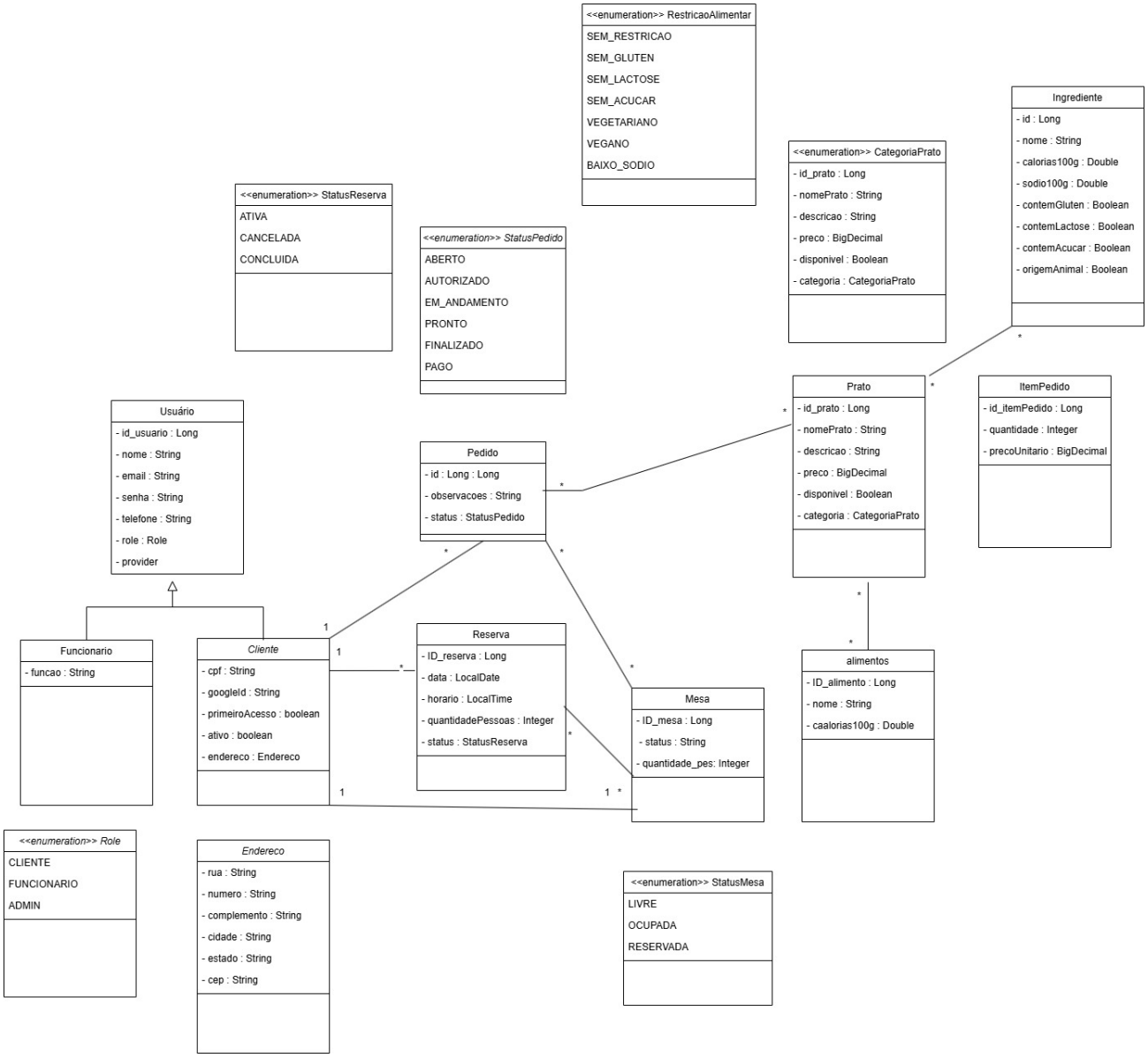


Figura 3.6: Diagrama de classe.

3.2 Protótipo

Com o objetivo de validar a proposta do sistema e garantir uma melhor experiência para os usuários, foi desenvolvido um protótipo de alta fidelidade utilizando a ferramenta Figma. O protótipo consiste em uma simulação navegável da interface, permitindo visualizar as telas

e o fluxo de interação entre elas. Essa abordagem possibilita demonstrar a organização das funcionalidades e a navegação do sistema antes da implementação final. As telas foram projetadas com foco na usabilidade, acessibilidade e clareza das informações, seguindo boas práticas de design e organização de conteúdo. A seguir, são apresentadas as principais telas que compõem o sistema, com suas respectivas descrições e funcionalidades.

3.2.1 Descrição do protótipo

A Figura 3.7 apresenta a tela de login, a qual recepciona o usuário com uma imagem ilustrativa de um prato, acompanhada de uma mensagem de boas-vindas e de um botão com o texto “Ver cardápio”. Essa tela tem como objetivo principal direcionar o usuário para o cardápio, facilitando a navegação inicial no sistema.

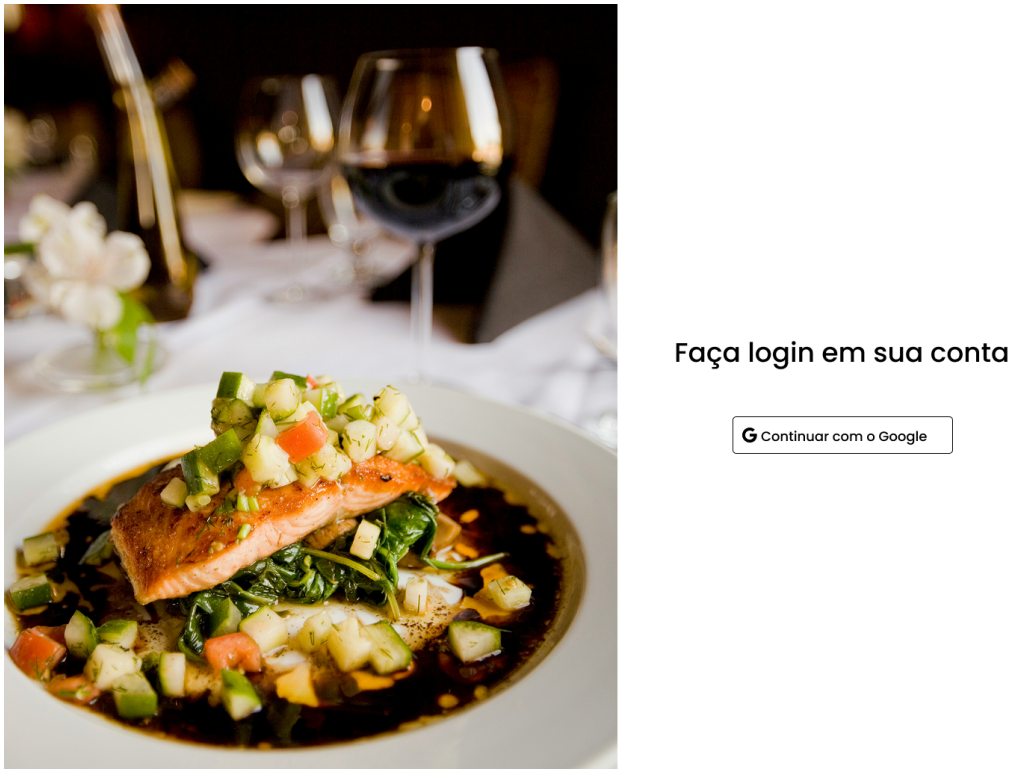


Figura 3.7: Tela de login.

A Figura 3.8 apresenta a página de cardápio digital do sistema, com um layout limpo e organizado que facilita a navegação do usuário. Na parte superior, há uma barra de navegação contendo as opções “Home”, “Reserva”, “Conta” e “Sacola”, acompanhada de um banner com imagens de pratos que reforçam o tema gastronômico. Logo abaixo, encontra-se um botão de “Restrições”, que permite aplicar filtros relacionados a preferências ou limitações alimentares, seguido de ícones circulares que representam as categorias do cardápio, como “Entrada”, “Prato

Principal", "Acompanhamentos", "Sobremesas" e "Bebidas". A seção principal exibe os pratos divididos por categoria, cada um apresentado em um card com imagem ilustrativa, nome e preço.

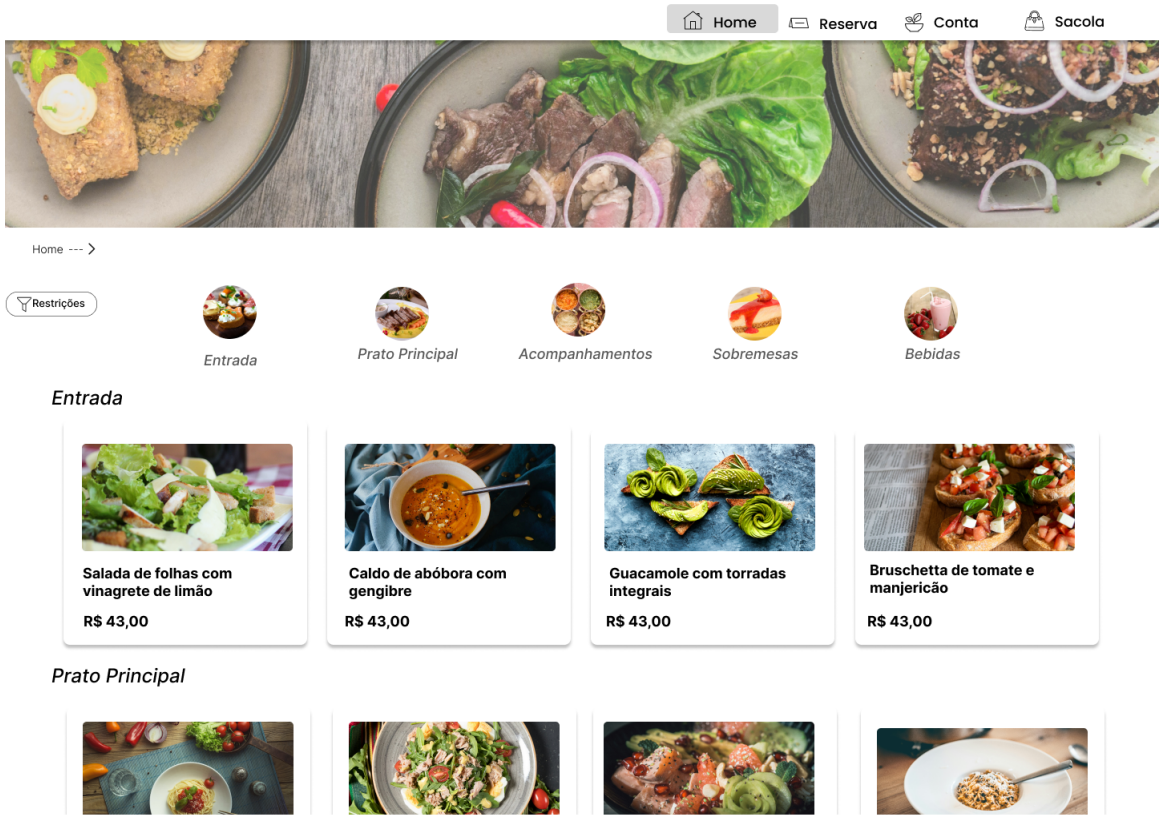


Figura 3.8: Tela do Cardapio

A Figura 3.9 permite que o usuário personalize a visualização do cardápio conforme suas necessidades alimentares. No lado esquerdo, há uma lista de filtros com opções como sem lactose, sem glúten, sem frutos do mar, sem soja e vegano, entre outras. Ao selecionar as opções desejadas, o sistema exibe apenas os pratos compatíveis. À direita, o cardápio é organizado por categorias (prato principal, acompanhamentos, sobremesas e bebidas), com imagens, nomes e preços dos pratos disponíveis.

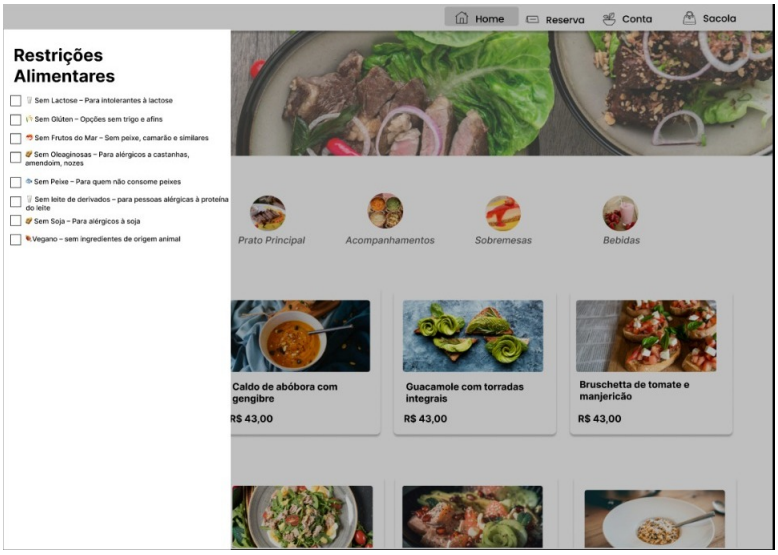


Figura 3.9: Tela de filtros

A Figura 3.10 mostra os detalhes de um prato selecionado. O usuário pode visualizar uma imagem ampliada do prato, a descrição completa e a tabela com informações nutricionais. A tabela é baseada na Tabela Brasileira de Composição de Alimentos (TACO), incluindo dados como valor energético, proteínas, carboidratos, gorduras e fibras.

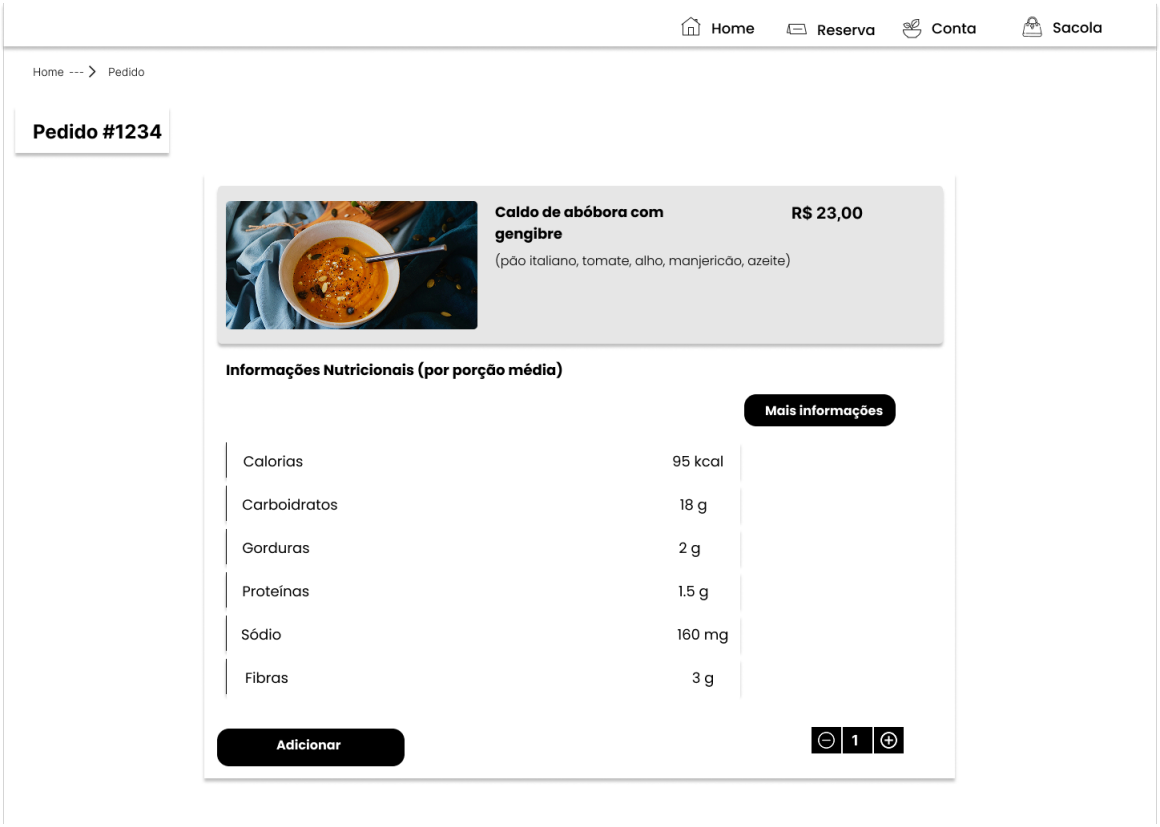


Figura 3.10: Tela de detalhes do pedido.

A Figura 3.11 exibe os itens adicionados ao pedido, permitindo ao usuário revisá-los

antes da finalização. Nela, é mostrado o número do pedido e o status do processo (em andamento ou finalizado). Cada item aparece em um card com imagem, nome, preço, controle de quantidade e opção de exclusão.

Há ainda um campo para observações e botões de ação: “Realizar Pedido” e “Adicionar Mais Itens”. No topo, a navegação contém os links principais (Home, Reserva, Conta e Sacola) e um botão de “Solicitar Atendimento”.

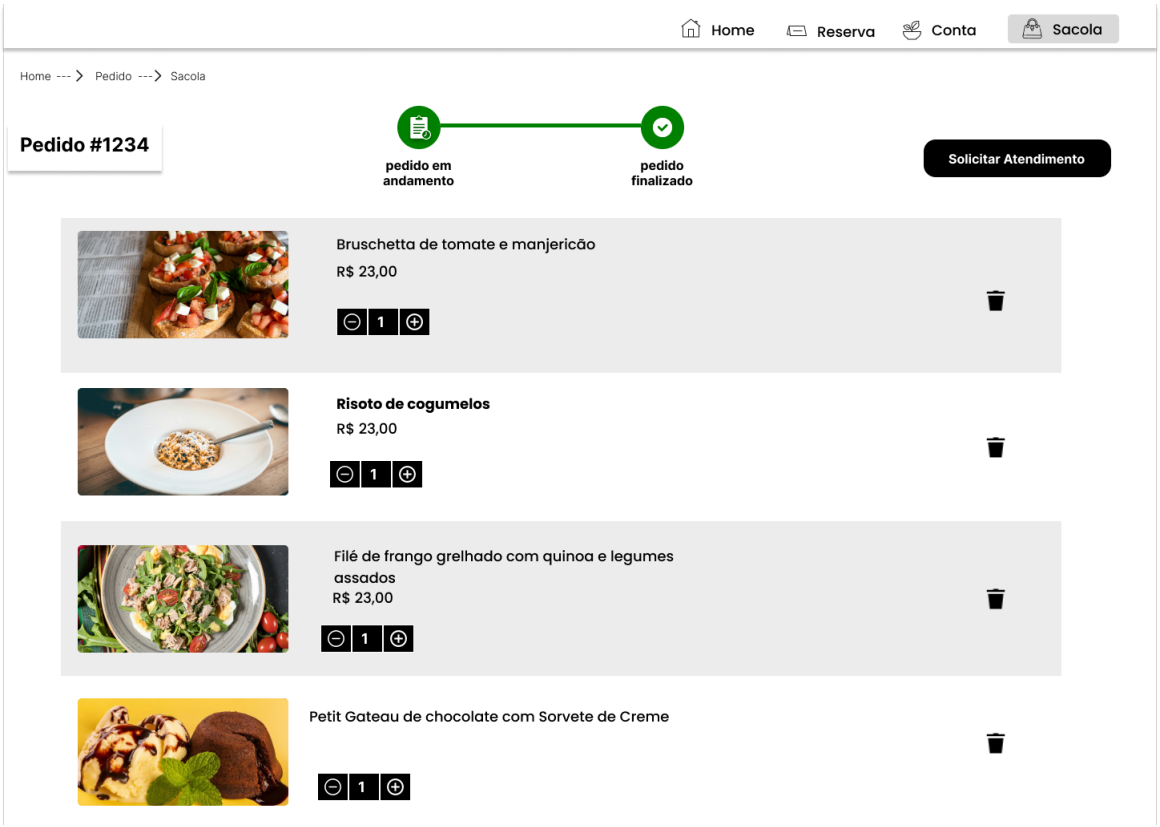


Figura 3.11: Tela da sacola

A Figura 3.12 exibe uma janela informando que o pedido já foi finalizado.

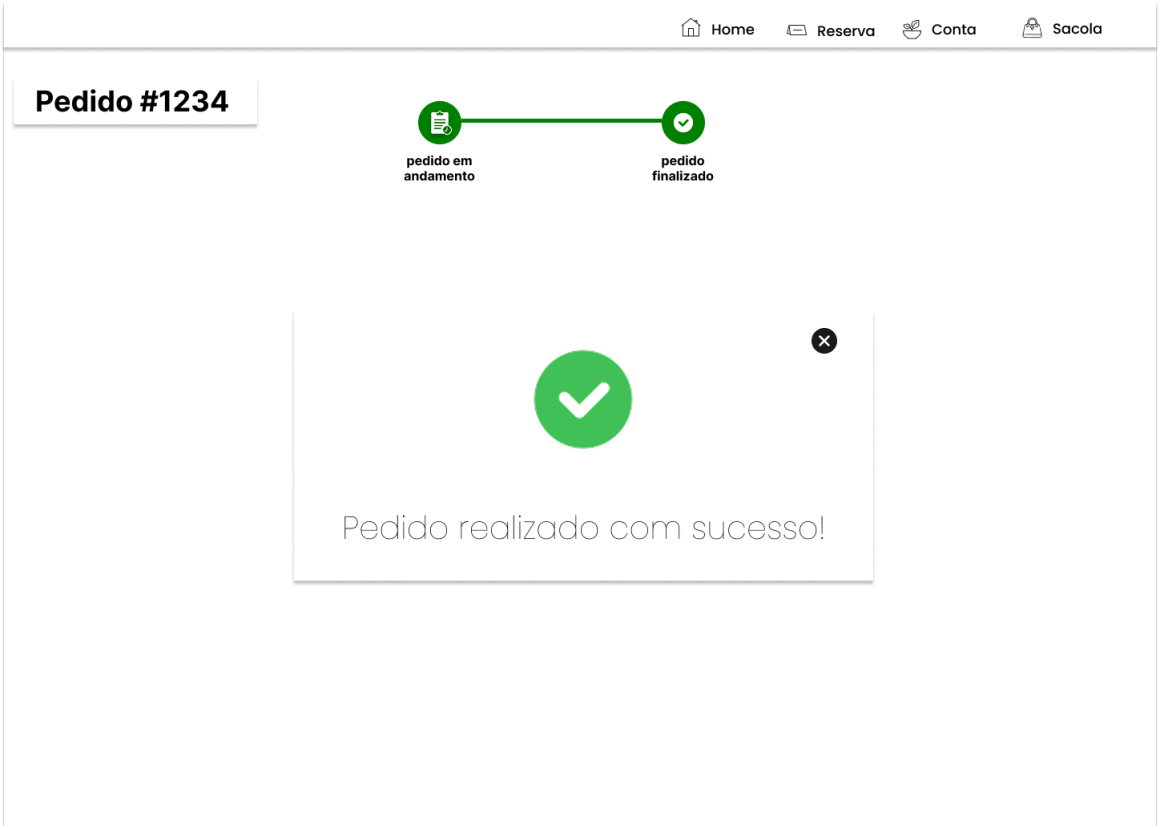


Figura 3.12: Pedido realizado

A Figura 3.13 exibe o histórico dos pedidos já realizados e um botão “Finalizar a conta”, permitindo que o cliente solicite o fechamento da conta.

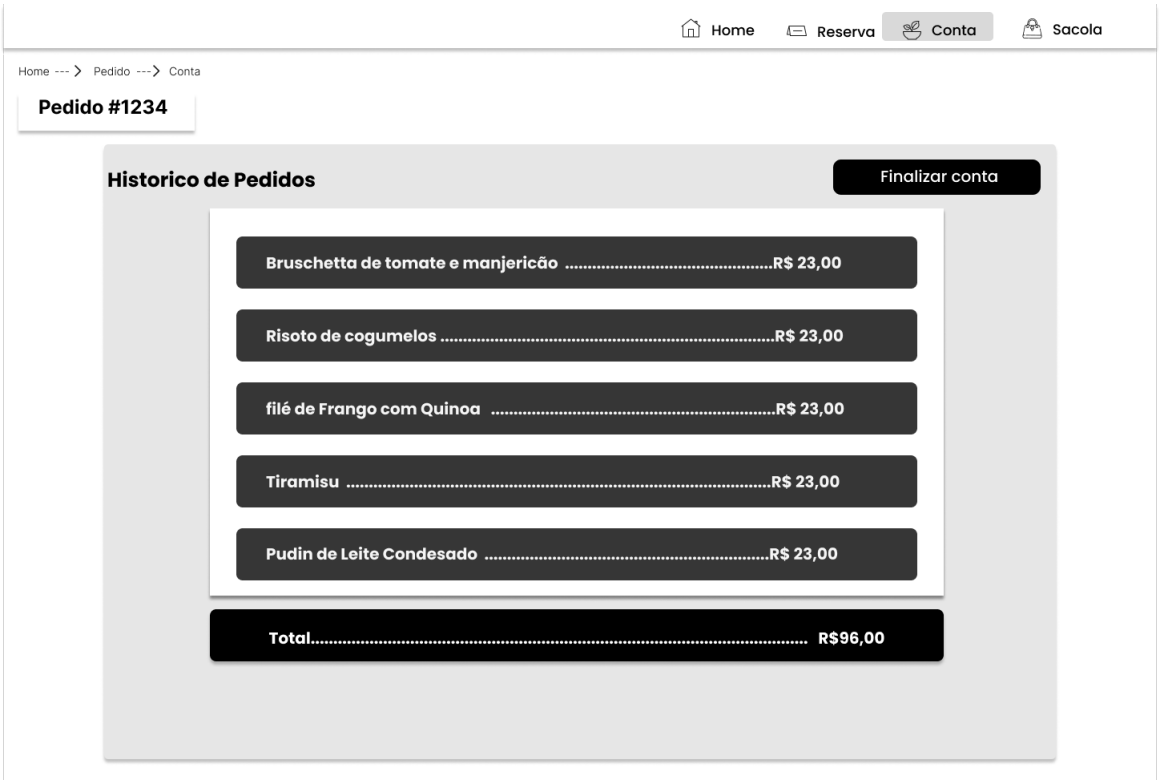


Figura 3.13: Tela da conta

A Figura 3.14 exibe uma janela informando que, para finalizar a conta, é necessário ir ao caixa para realizar o pagamento.

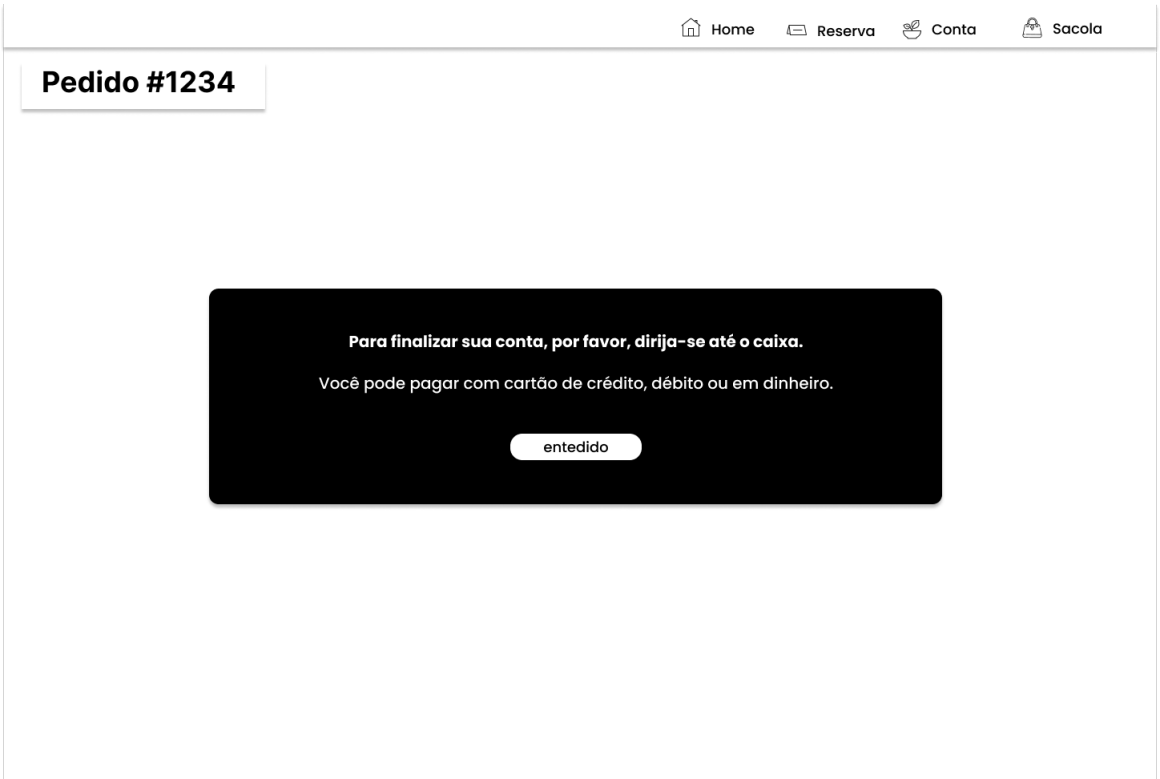


Figura 3.14: Tela da conta aviso

A Figura 3.15 exibe as mesas disponíveis, apresentadas em cinza, que podem ser selecionadas pelo cliente. Já as mesas indisponíveis são exibidas em outra cor e não permitem interação.

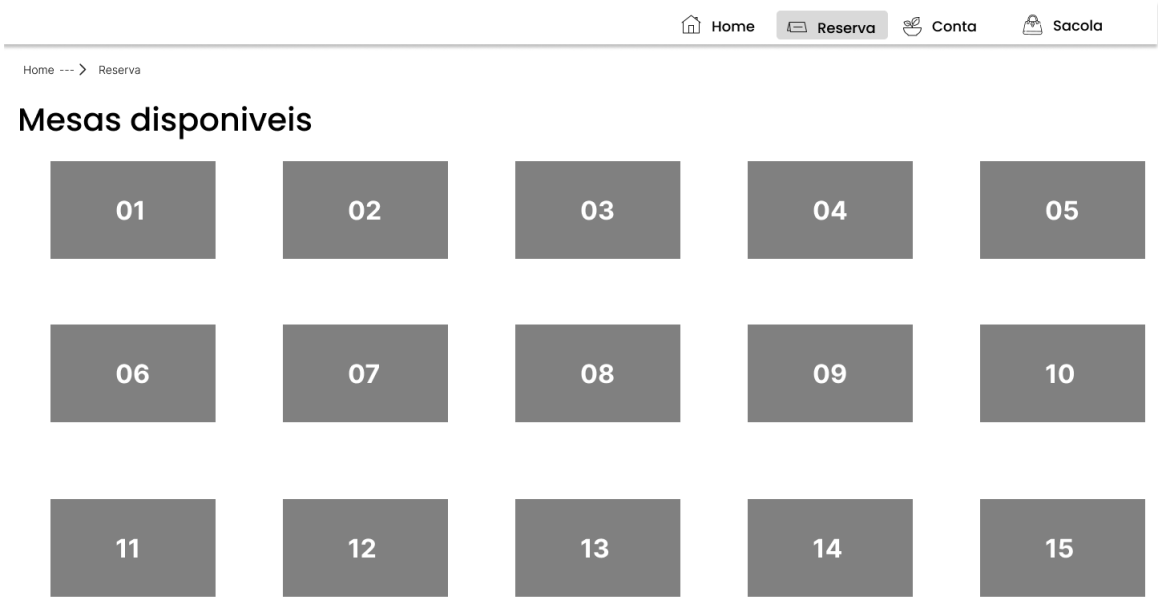


Figura 3.15: Tela de reserva

A Figura 3.16 exibe um formulário no qual o usuário deve informar a data, o horário e a quantidade de pessoas para verificar a disponibilidade e confirmar a reserva no sistema.

Figura 3.16: Tela de informações da reserva

A Figura 3.17 exibe uma mensagem de sucesso, informando que a mesa foi reservada com êxito.

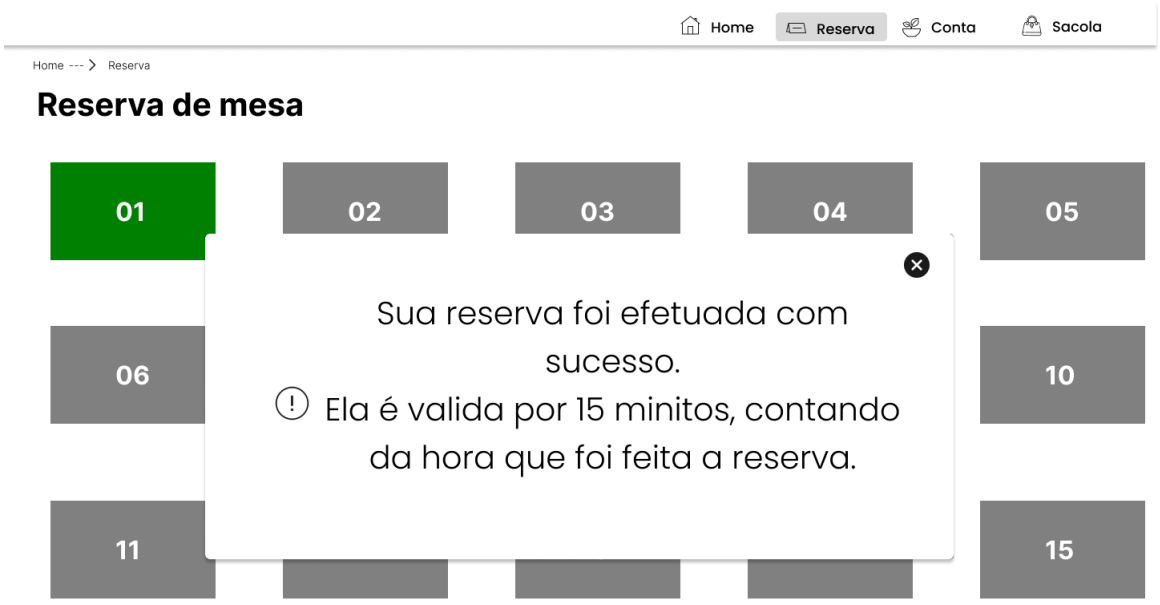


Figura 3.17: Tela de aviso reserva feita

A Figura 3.18 exibe um aviso de restrição informando que o cliente não pode reservar mais de três mesas.

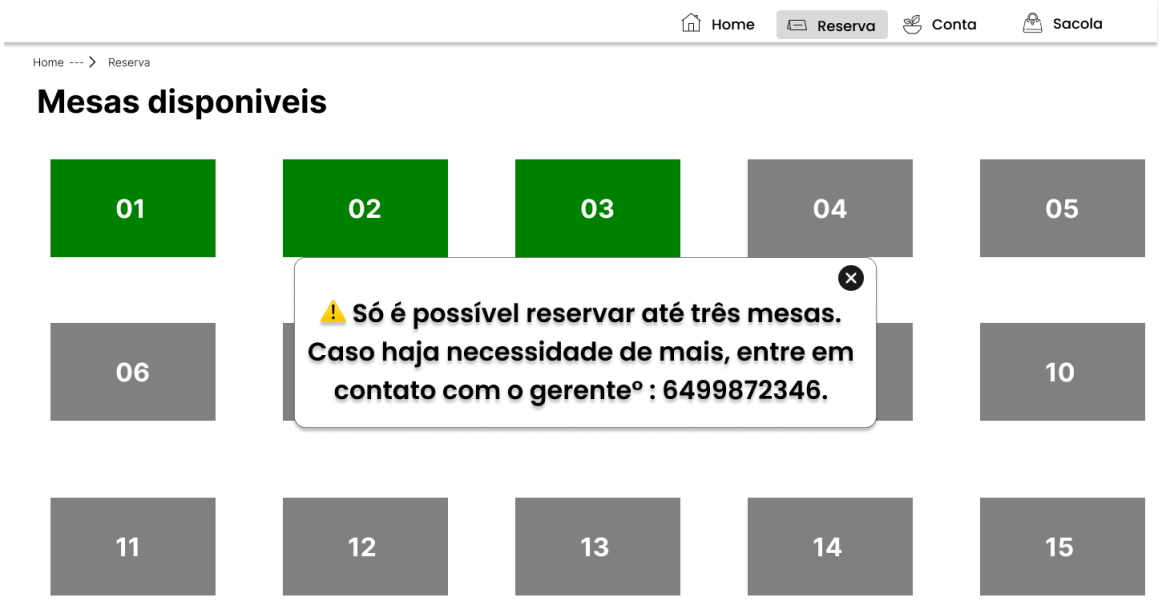


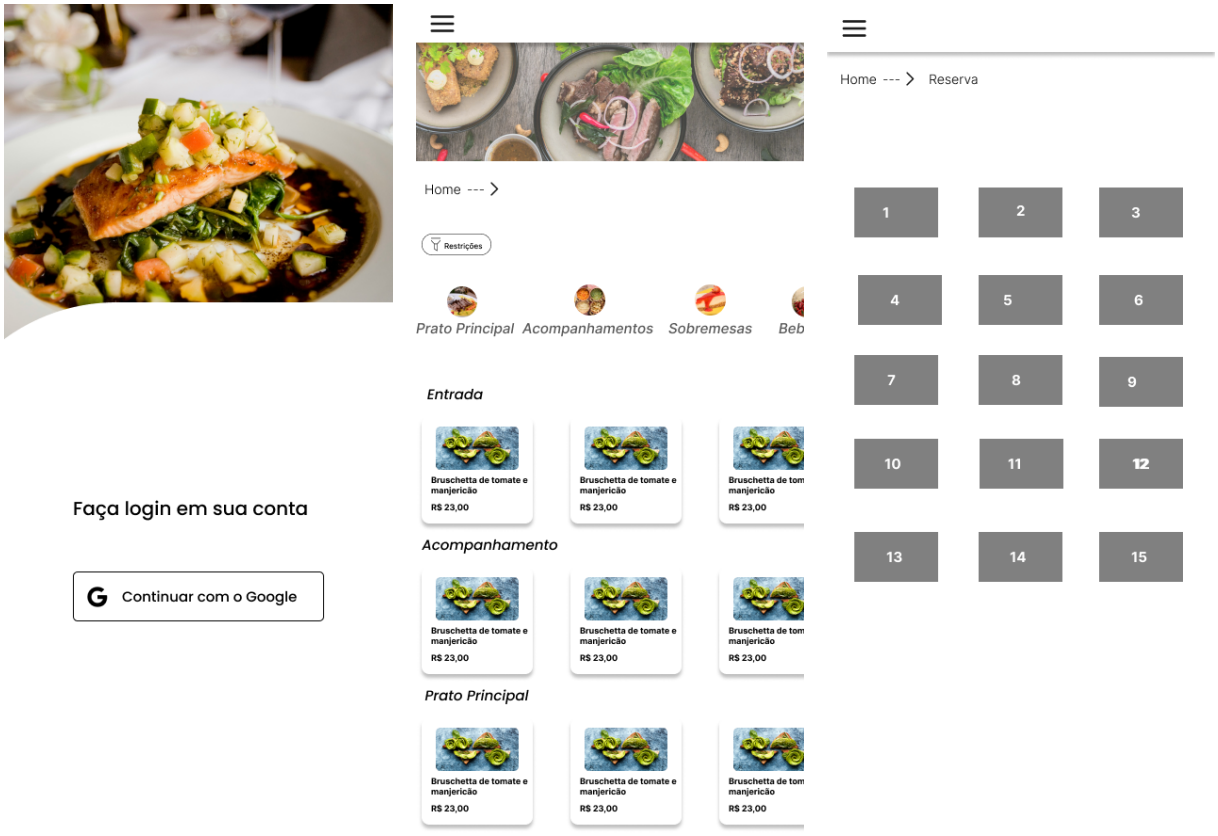
Figura 3.18: Tela de aviso de limite reserva

A Figura **??** exibe uma mensagem de indisponibilidade, informando que não há mesas

livres para o horário selecionado. Não há fila de espera implementada; o cliente precisa selecionar outro horário ou data.

Figura 3.19: Tela de aviso de mesas indisponiveis

A Figura 3.20 apresenta a versão mobile do sistema, composta por três telas principais: tela de login, tela de cardápio e tela de reserva de mesas. Essas telas foram projetadas para dispositivos móveis, mantendo a responsividade e garantindo que os usuários possam acessar todas as funcionalidades principais do sistema com clareza e facilidade, independentemente do tamanho da tela do dispositivo.



(a) Tela de Login (b) Tela de Cardapio (c) Tela de Reservas

Figura 3.20: Protótipos de interface mobile

3.2.2 Avaliação do protótipo

Com o objetivo de coletar *feedback* qualitativo e quantitativo sobre a experiência dos usuários com o protótipo de telas desenvolvido, foi realizada uma pesquisa por meio de um formulário online. Ressalta-se que a avaliação considerou apenas o protótipo navegável, e não o sistema implementado, permitindo analisar principalmente aspectos de usabilidade, clareza das informações e facilidade de navegação. A metodologia adotada teve como foco avaliar a usabilidade, identificar pontos de melhoria e compreender a percepção geral do público-alvo em relação às funcionalidades e informações apresentadas.

Instrumento de Coleta de Dados: A ferramenta utilizada para a pesquisa foi o Google Forms, pela sua praticidade, acessibilidade e facilidade de organização dos dados. O formulário continha perguntas objetivas e subjetivas, estruturadas para abranger aspectos técnicos e experienciais. As perguntas foram divididas nos seguintes blocos:

- Avaliação geral da experiência com o site (escala de 1 a 5);

- Facilidade de uso e navegação;
- Identificação de erros ou falhas no sistema;
- Funcionalidades mais apreciadas;
- Dúvidas ou dificuldades encontradas;
- Sugestões de melhorias ou funcionalidades ausentes;
- Espaço para comentários finais.

Participantes: A pesquisa foi disponibilizada a um grupo intencional de usuários, incluindo pessoas do público-alvo do projeto, com interesse em alimentação saudável e informações nutricionais. Ao todo, foram recebidas cinco respostas completas. Embora o número seja limitado, os participantes foram selecionados para refletir o perfil típico de usuários do sistema, incluindo variação de idade, gênero e familiaridade com navegação em dispositivos digitais.

Procedimentos: O link do formulário foi compartilhado com os participantes por meios digitais (como WhatsApp e e-mail). Todos os participantes foram informados de que a pesquisa era voluntária e que os dados coletados seriam utilizados exclusivamente para fins acadêmicos e de melhoria do sistema.

Após o período de coleta, os dados foram exportados em formato CSV e analisados com o auxílio de ferramentas de visualização de dados e geração de relatórios. A Figura 3.21 apresenta o resultado final da avaliação. No gráfico, o eixo horizontal (eixo X) representa a escala de avaliação, variando de 1 a 5, enquanto o eixo vertical (eixo Y) representa cada participante que respondeu ao formulário. Cada barra horizontal corresponde à nota atribuída individualmente por um usuário.

De acordo com o gráfico, a avaliação geral apresentou notas 4 e 5, indicando uma análise positiva quanto à usabilidade e ao funcionamento do protótipo. Observa-se apenas uma avaliação com nota 3, o que pode ser justificado pelas respostas complementares do participante, que indicou ter encontrado um erro no sistema — especificamente a não exibição de valores — além de classificar a facilidade de uso como “neutro”.

Esses resultados demonstram que, apesar de uma boa aceitação geral, ajustes técnicos ainda são necessários para aprimorar a experiência do usuário.

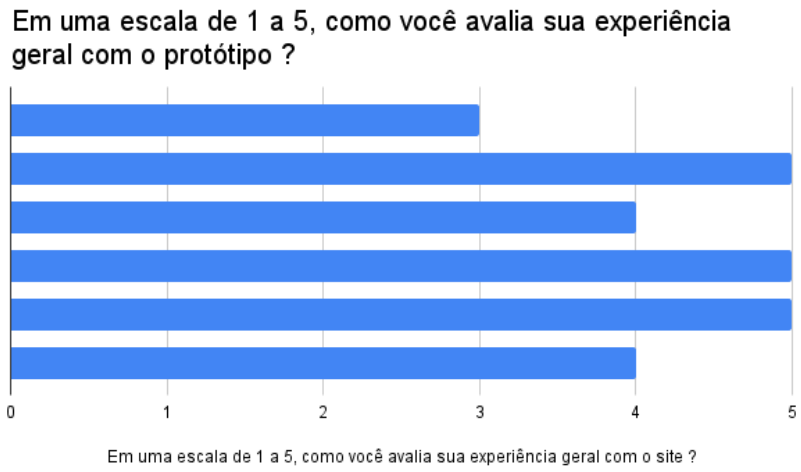


Figura 3.21: Avaliação do protótipo

CAPÍTULO 4

IMPLEMENTAÇÃO DO SISTEMA

Este capítulo apresenta a arquitetura e a implementação do sistema desenvolvido. Inicialmente, são descritas a arquitetura do sistema, a separação entre *front-end* e *back-end* e a comunicação entre as camadas, destacando os padrões e as tecnologias escolhidos para atender aos objetivos do projeto. Em seguida, detalha-se a implementação, incluindo a estrutura do *back-end*, a API, a regra de cálculo nutricional, os mecanismos de segurança, o banco de dados e o *front-end*, finalizando com a apresentação do sistema em funcionamento.

4.1 Arquitetura do Sistema

A arquitetura do sistema foi planejada seguindo o padrão em camadas, visando organizar melhor o código, garantir escalabilidade e facilitar a manutenção. O sistema é composto por duas camadas principais:

- **Front-end**: responsável pela interface com o usuário, pela interação com as telas e pela apresentação das informações, desenvolvido em Angular.
- **Back-end**: responsável pela lógica de negócio, pelo acesso ao banco de dados e pelo processamento das regras da aplicação, implementado em Java com Spring Boot.

A comunicação entre *front-end* e *back-end* ocorre por meio de uma API REST, garantindo uma troca eficiente de informações e uma separação clara entre apresentação e processamento. Essa arquitetura permite que cada camada evolua de forma independente, facilitando futuras atualizações ou expansões do sistema.

Além disso, as tecnologias escolhidas foram o *Spring Boot* e o Angular, que foram selecionadas para atender aos objetivos do projeto. Com esse conjunto de escolhas, obtém-se uma arquitetura organizada, escalável, segura e com facilidade de integração entre as camadas.

A Figura 4.1 apresenta as principais camadas que compõem a aplicação, evidenciando a interação entre o cliente web, desenvolvido em Angular, a API REST implementada em Spring Boot, responsável pela lógica de negócio, e a camada de dados, onde são armazenadas as informações do sistema. Além disso, são representados os principais módulos da aplicação, como os serviços de usuários, cardápio, pedidos e reservas, que organizam as funcionalidades do sistema e permitem o gerenciamento das operações relacionadas ao restaurante e às informações nutricionais dos pratos.

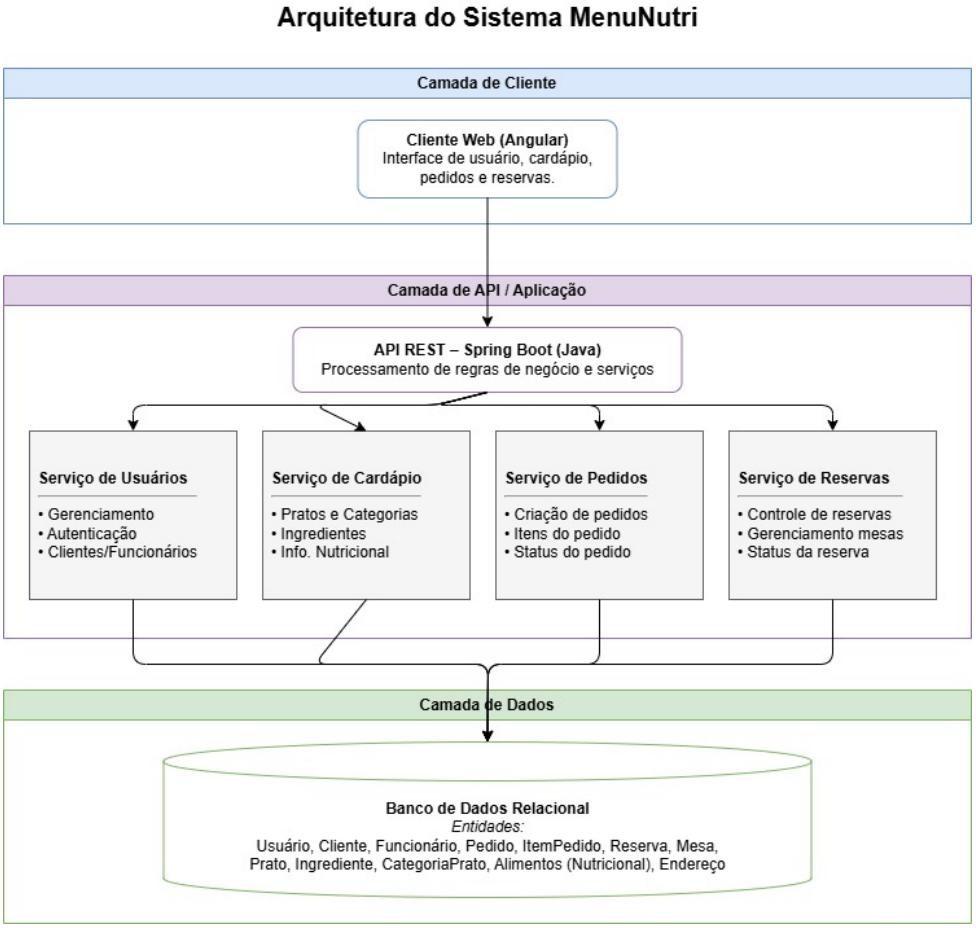


Figura 4.1: Diagrama de arquiteturas em camadas

4.2 Implementação do Back-end

O back-end do sistema foi desenvolvido utilizando a linguagem Java e o *framework* Spring Boot, por meio de uma arquitetura em camadas que proporciona maior organização e

escalabilidade à aplicação.

4.2.1 Estrutura do Sistema

O projeto foi estruturado seguindo as boas práticas recomendadas para aplicações REST. O código-fonte está organizado em pacotes que representam as principais responsabilidades da aplicação, permitindo a separação entre as camadas de apresentação, negócio e persistência de dados. Essa organização facilita a manutenção, a escalabilidade e a compreensão da aplicação.

- O pacote *controller* é responsável por concentrar os controladores da aplicação, que expõem os *endpoints* da API e realizam a comunicação entre o *front-end* e o *back-end* por meio de requisições HTTP.
- O pacote *service* contém a lógica de negócio do sistema, sendo responsável pelo processamento das regras da aplicação, cálculos e validações, como o cálculo das informações nutricionais dos pratos com base na Tabela Brasileira de Composição de Alimentos (TACO).
- O pacote *repository* é responsável pelo acesso aos dados, utilizando o Spring Data JPA para a realização de operações de persistência no banco de dados, como criação, leitura, atualização e exclusão de registros.
- O pacote *model* contém as entidades do sistema, representando as tabelas do banco de dados e seus respectivos atributos, o que possibilita o mapeamento objeto-relacional.
- O pacote *dto* é utilizado para a transferência de dados entre as camadas da aplicação e o *front-end*, garantindo maior controle sobre as informações trafegadas e evitando a exposição direta das entidades do banco de dados.
- O pacote *security* é utilizado para armazenar as configurações relacionadas à autenticação e autorização.
- O pacote *exception* contém as classes responsáveis pelo tratamento de exceções personalizadas.
- O pacote *config* armazena as classes de configurações gerais do sistema.
- O pacote *util* armazena as classes utilitárias.

As Figuras 4.2, 4.3, 4.4, 4.5, 4.6, 4.7 e 4.8 apresentam a estrutura de pastas do *back-end* da aplicação.

A Figura 4.2 apresenta a estrutura geral de diretórios do *back-end* da aplicação. O projeto segue o padrão de organização utilizado em aplicações desenvolvidas com Spring Boot, no qual o código-fonte principal está localizado no diretório *src/main/java*. Dentro desse diretório encontram-se os pacotes responsáveis pela implementação das diferentes camadas da aplicação, permitindo uma melhor organização do código e a separação de responsabilidades.

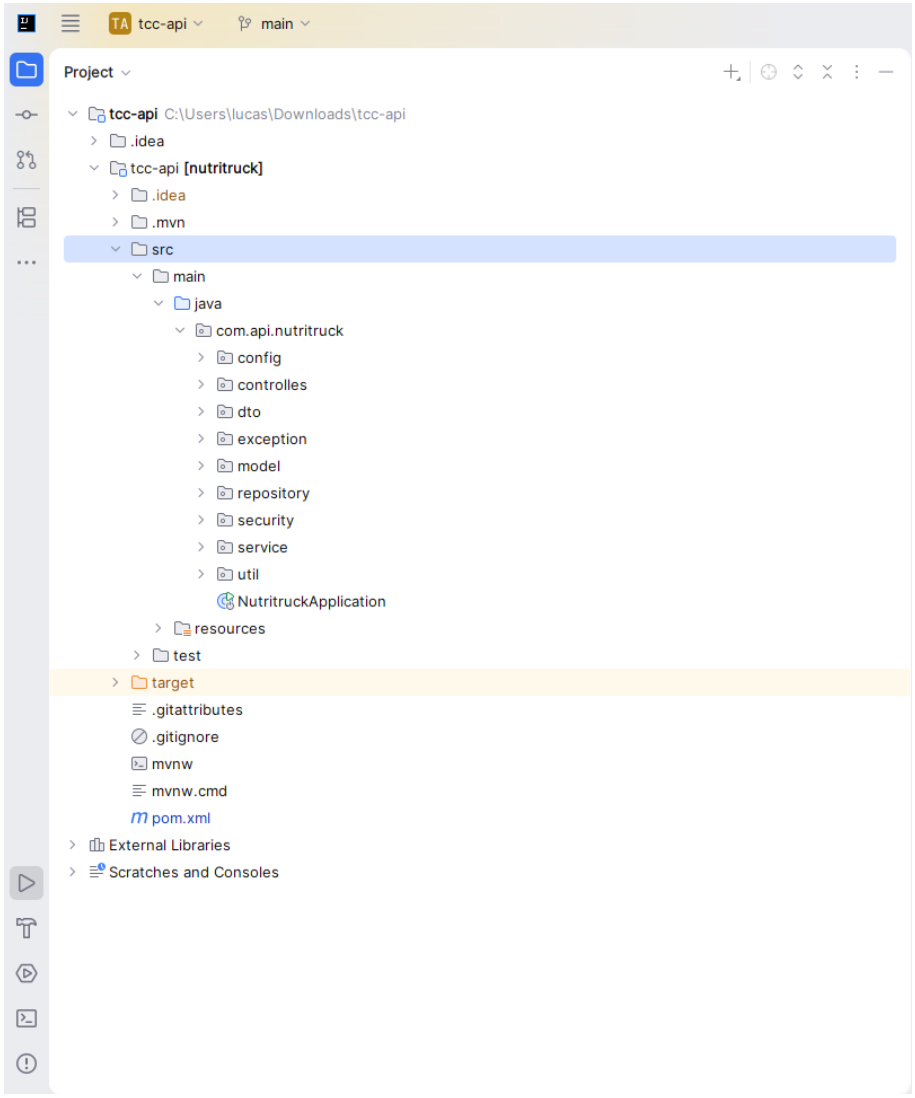


Figura 4.2: Estrutura geral das pastas do back-end

A Figura 4.3 apresenta os pacotes *controller* e *config*. O pacote *controller* é responsável por receber as requisições HTTP enviadas pelos usuários e encaminhá-las para as camadas responsáveis pelo processamento das regras de negócio. Já o pacote *config* contém as configurações da aplicação, como definições de segurança, interceptadores e outras configurações gerais

necessárias para o funcionamento do sistema.

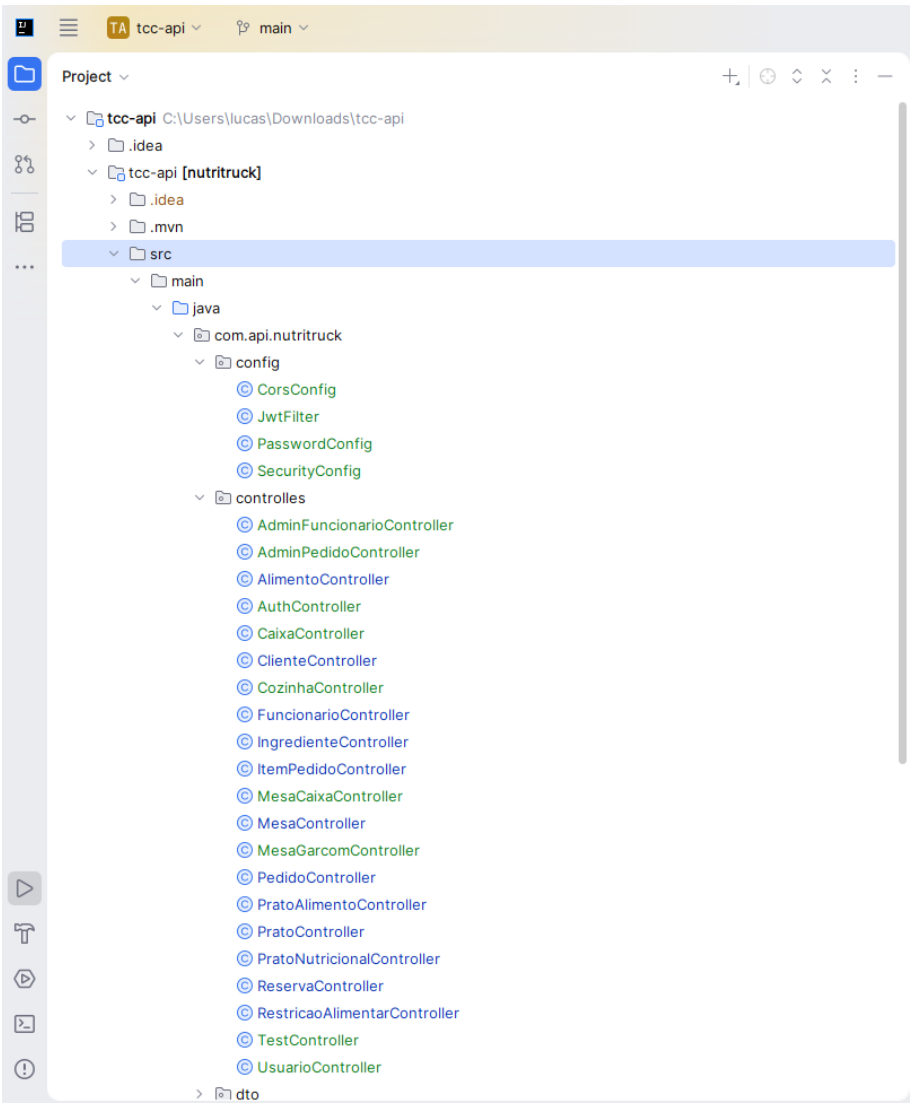


Figura 4.3: Subpacotes do back-end: controller e config

A Figura 4.4 apresenta os pacotes *dto* e *exception*. O pacote *dto* (*Data Transfer Object*) é utilizado para transportar dados entre as diferentes camadas da aplicação, garantindo que apenas as informações necessárias sejam expostas. Já o pacote *exception* é responsável pelo tratamento de exceções do sistema, permitindo capturar erros e retornar mensagens adequadas ao usuário.

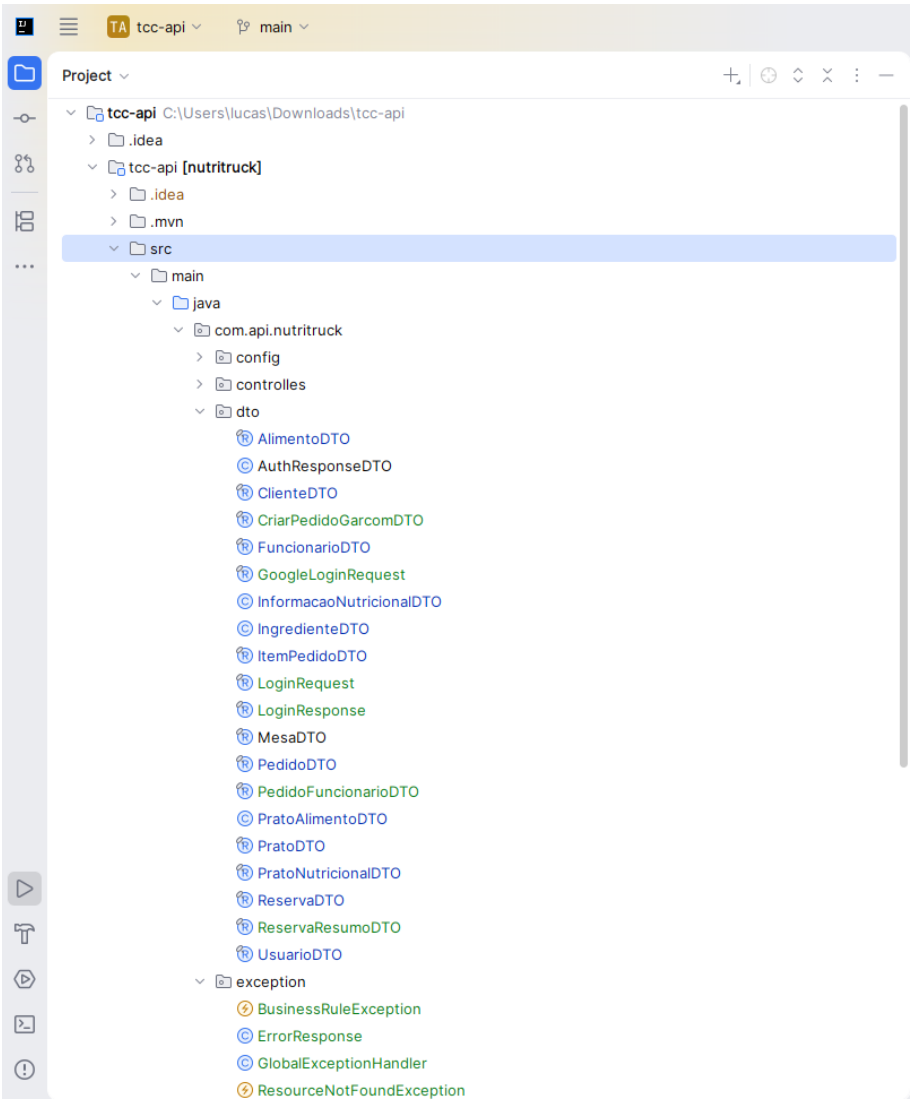


Figura 4.4: Subpacotes do back-end: dto e exception

A Figura 4.5 apresenta os pacotes *model* e *repository*. O pacote *model* contém as entidades da aplicação, que representam as estruturas de dados armazenadas no banco de dados. Já o pacote *repository* é responsável pela comunicação com o banco de dados, realizando operações como inserção, consulta, atualização e remoção de registros.

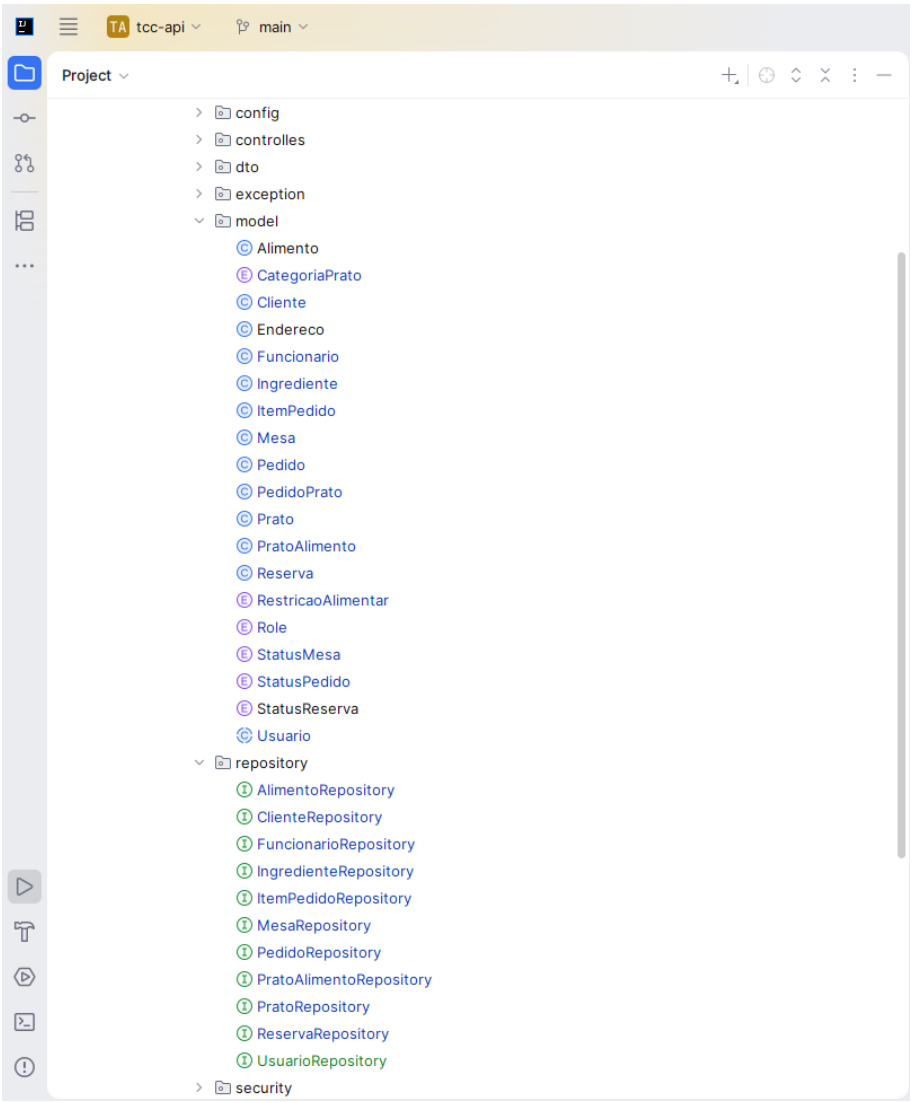


Figura 4.5: Subpacotes do back-end: model e repository

A Figura 4.6 apresenta os pacotes *repository* e *security*. O pacote *repository* contém as interfaces responsáveis pela comunicação com o banco de dados e pelas operações de persistência das entidades do sistema. Já o pacote *security* concentra as configurações e componentes relacionados à segurança da aplicação, como autenticação, autorização e controle de acesso dos usuários.

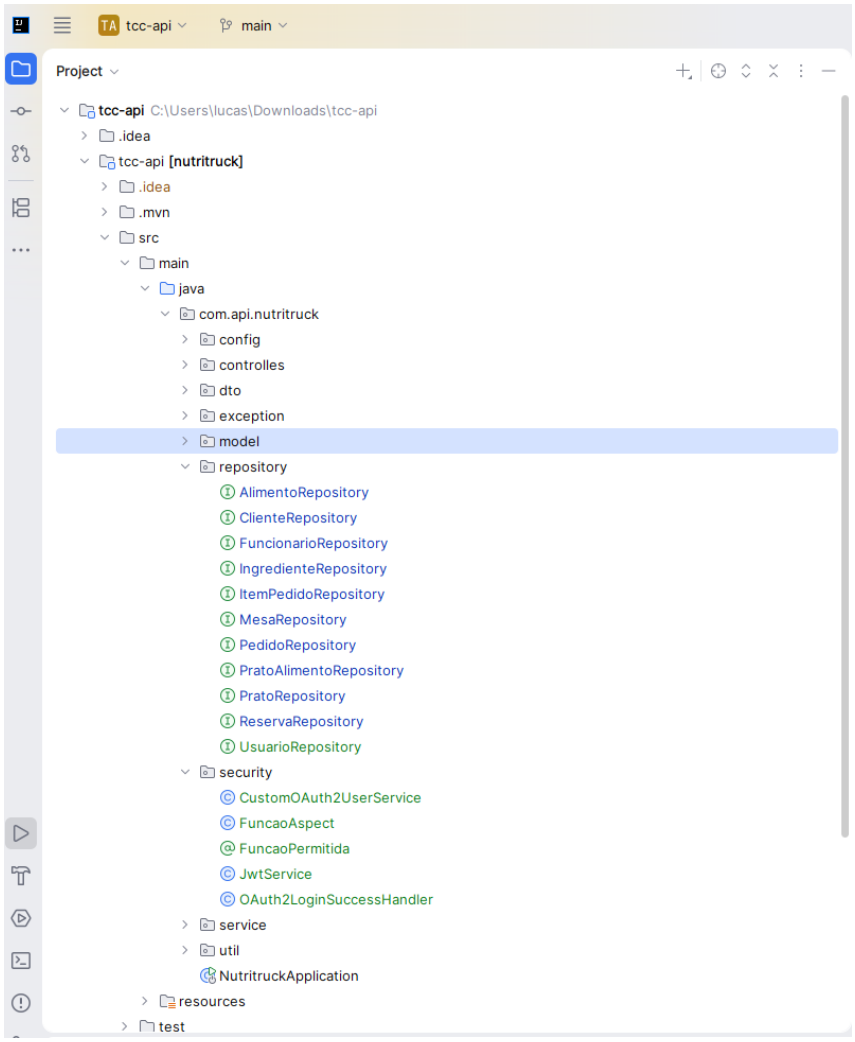


Figura 4.6: Subpacotes do back-end: repository e security

A Figura 4.7 apresenta os pacotes *service* e *util*. O pacote *service* contém as classes responsáveis pela implementação das regras de negócio do sistema, realizando o processamento das informações recebidas pelos controladores e interagindo com os repositórios. O pacote *util* reúne classes auxiliares utilizadas em diferentes partes da aplicação, oferecendo funções de apoio que facilitam a organização e a reutilização do código.

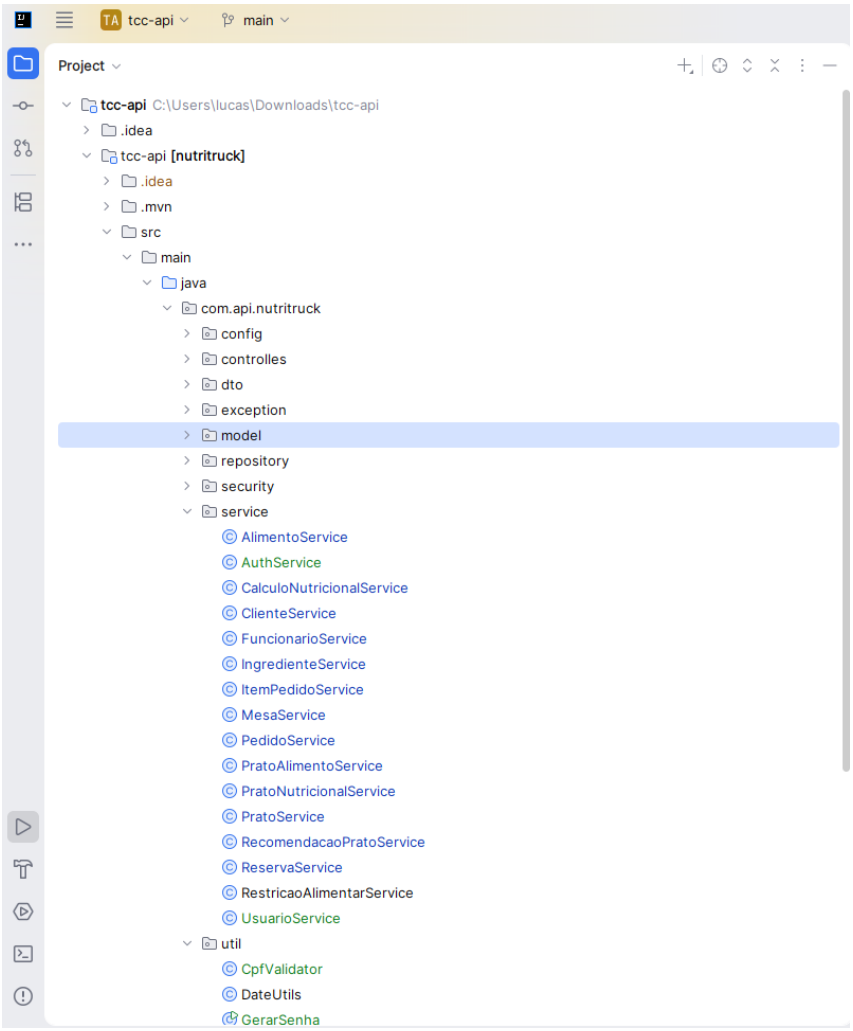


Figura 4.7: Subpacotes do back-end: service e util

A Figura 4.8 apresenta os pacotes destinados aos testes automatizados do sistema. Esses pacotes contêm testes relacionados principalmente às camadas *service* e *util*, com o objetivo de verificar o funcionamento correto das regras de negócio e das funcionalidades auxiliares implementadas, contribuindo para a qualidade e a confiabilidade da aplicação.

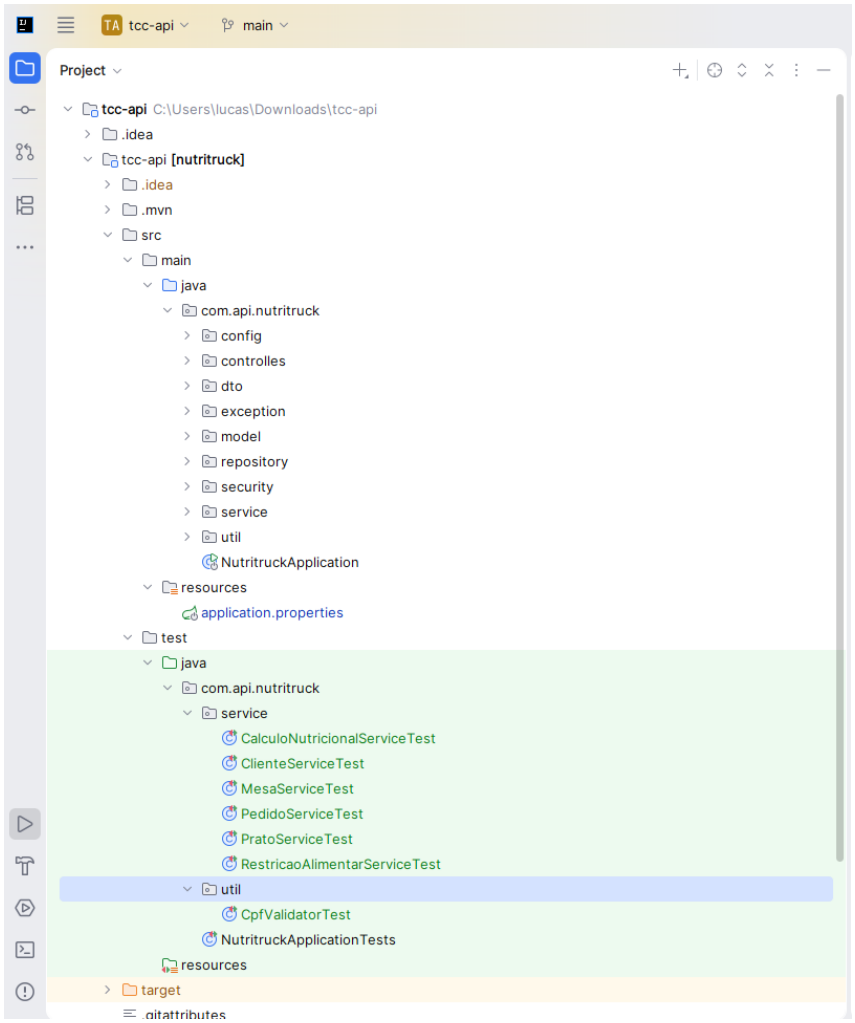


Figura 4.8: Pacotes de testes do sistema

4.2.2 API e endpoints

Responsável por intermediar a comunicação entre o *front-end* e o *back-end*, a API foi implementada utilizando o Spring Boot, seguindo o padrão de arquitetura REST, o que possibilita a troca de informações por meio de requisições HTTP de forma estruturada e padronizada.

Os endpoints da API foram definidos de acordo com as funcionalidades do sistema, permitindo operações como cadastro, consulta, atualização e remoção de dados. Cada endpoint está associado a um recurso específico e utiliza métodos HTTP como GET, POST, PUT e DELETE, garantindo a correta manipulação das informações.

Para possibilitar a comunicação entre o *front-end* e o *back-end*, foram definidos diversos endpoints da API REST, cada um associado a um recurso específico do sistema. A seguir, são apresentadas as tabelas que descrevem os endpoints organizados por funcionalidade, incluindo método HTTP, URL e objetivo de cada rota.

A Tabela 4.1 apresenta os endpoints responsáveis pela autenticação de usuários e funcionários, diferenciando os tipos de perfil (cliente, garçom, caixa e chefe de cozinha). Cada endpoint garante que o usuário seja autenticado corretamente, permitindo acesso às funcionalidades correspondentes ao seu perfil.

Endpoints de autenticação

Método	Endpoint	Descrição
POST	/criar_funcionario	Realiza o cadastro de um funcionário no sistema com sua respectiva função.
POST	/login_funcionario_garcon	Realiza o login do funcionário do tipo garçom.
POST	/login_caixa	Realiza o login do funcionário do tipo caixa.
POST	/login_chefe_cozinha	Realiza o login do funcionário do tipo chefe de cozinha.
POST	/login_cliente	Realiza o login do cliente utilizando e-mail e senha.

Tabela 4.1: Endpoints de autenticação

A Tabela 4.2 apresenta os endpoints relacionados ao cadastro e gerenciamento de alimentos e pratos do cardápio. Essas rotas incluem operações de criação, listagem, associação de alimentos a pratos e remoção, garantindo que o sistema mantenha as informações atualizadas e consistentes.

Endpoints de alimentos e pratos

Método	Endpoint	Descrição
POST	/criar_alimento	Cadastra um novo alimento no sistema.
GET	/listar_alimento	Retorna a listagem de alimentos cadastrados.
POST	/criar_prato	Cadastra um novo prato no sistema.
GET	/listar_prato	Retorna a listagem de pratos cadastrados.
DELETE	/deletar_prato	Remove um prato do sistema.
POST	/criar_pratoAlimento	Associa alimentos a um prato específico.

Tabela 4.2: Endpoints de alimentos e pratos

A Tabela 4.3 apresenta os endpoints responsáveis pelo cadastro e listagem das mesas do restaurante, permitindo que o sistema controle a disponibilidade e organize as reservas.

Endpoints de mesas

Método	Endpoint	Descrição
POST	/criar_mesa	Realiza o cadastro de uma nova mesa.
GET	/listar_mesa	Retorna a listagem de mesas cadastradas.

Tabela 4.3: Endpoints de mesas

A Tabela 4.4 apresenta os endpoints utilizados pelo garçom para registrar pedidos, atualizar o status das mesas e visualizar listas de pedidos e mesas, facilitando o processo de atendimento.

Endpoints do garçom

Método	Endpoint	Descrição
POST	/anotar_pedido_garcon	Permite ao garçom registrar um pedido.
GET	/listar_pedido_garcon	Retorna a listagem de pedidos.
PUT	/atualizar_status_mesa_garcon	Atualiza o status da mesa pelo garçom.
GET	/listar_mesa_garcon	Lista as mesas atendidas pelo garçom.

Tabela 4.4: Endpoints do garçom

A Tabela 4.5 apresenta os endpoints que permitem à cozinha consultar os pedidos recebidos e atualizar o status de preparo ou finalização, garantindo que a preparação dos pratos seja controlada de forma eficiente.

Endpoints da cozinha

Método	Endpoint	Descrição
GET	/listar_pedido_cozinha	Retorna os pedidos enviados para a cozinha.
POST	/iniciar_preparo_pedido	Marca o pedido como em preparo.
POST	/finalizar_preparo_pedido	Marca o pedido como finalizado.

Tabela 4.5: Endpoints da cozinha

A Tabela 4.6 apresenta os endpoints utilizados pelo caixa para gerenciar o fechamento de pedidos, atualizar o status de pedidos e mesas, e listar mesas e pedidos registrados no sistema.

Endpoints do caixa

Método	Endpoint	Descrição
POST	/caixa_fechar_pedido	Realiza o fechamento do pedido no caixa.
PUT	/editar_pedido	Atualiza o status do pedido para FINALIZADO ou PAGO.
PUT	/atualizar_status_mesa	Atualiza o status da mesa (DISPONÍVEL ou OCUPADA).
GET	/listar_mesa	Lista as mesas cadastradas.
GET	/listar_pedidos	Lista os pedidos registrados no sistema.

Tabela 4.6: Endpoints do caixa

A Tabela 4.7 apresenta os endpoints utilizados pelo cliente para criar ou cancelar reservas de mesas, permitindo o controle das reservas ativas e evitando a sobreposição de horários.

Endpoints de reserva

Método	Endpoint	Descrição
POST	/criar_reserva	Permite ao cliente realizar uma nova reserva no sistema.
PUT	/cancelar_reserva	Realiza o cancelamento de uma reserva existente.

Tabela 4.7: Endpoints de reserva

A Tabela 4.8 apresenta os endpoints disponíveis para o gerente, permitindo autorizar pedidos antes de seu encaminhamento para a cozinha, garantindo maior controle sobre as operações do restaurante.

Endpoints do gerente

Método	Endpoint	Descrição
POST	/autorizar_pedido	Permite ao gerente autorizar um pedido antes de seu encaminhamento para preparo.

Tabela 4.8: Endpoints do gerente

4.2.3 Implementação da Regra de Cálculo Nutricional

A Tabela TACO apresenta os valores nutricionais dos alimentos considerando uma porção padrão de 100 gramas, tanto para alimentos líquidos quanto sólidos. Para apresentar a contagem de calorias de um prato, é necessário realizar um cálculo proporcional, no qual as calorias (ou nutrientes) totais são obtidas por meio da multiplicação do valor energético do alimento, presente na Tabela TACO, pela quantidade do ingrediente utilizada no prato, dividida por 100. Dessa forma, obtém-se o valor proporcional correspondente à quantidade efetivamente utilizada na preparação do prato.

O Quadro 4.1 apresenta o código correspondente à implementação do cálculo nutricional.

O código apresentado no Quadro 4.1 implementa a regra responsável pelo cálculo das informações nutricionais de um prato a partir dos ingredientes que o compõem. Essa implementação está localizada na camada de serviço (*Service*) da aplicação, responsável por concentrar as regras de negócio do sistema.

A classe *CalculoNutricionalService* recebe como parâmetro uma lista de ingredientes (*PratoAlimento*) associados a determinado prato. Cada item da lista contém a quantidade do ingrediente utilizada na receita e uma referência ao alimento cadastrado na base de dados, que possui os valores nutricionais correspondentes a 100 gramas, conforme definidos pela Tabela TACO.

Inicialmente, o método cria um objeto do tipo *InformacaoNutricionalDTO*, responsável por armazenar o resultado final do cálculo nutricional. Em seguida, são inicializadas variáveis que irão acumular os valores de calorias, carboidratos, proteínas, gorduras, fibras e sódio.

O cálculo é realizado por meio de um laço de repetição que percorre todos os ingredientes do prato. Para cada ingrediente, é calculado um fator de proporção, obtido pela divisão da quantidade utilizada no prato por 100. Esse fator é utilizado para ajustar os valores nutricionais do alimento, que na base TACO estão definidos para porções de 100 gramas. Dessa forma, cada nutriente é multiplicado por esse fator e somado ao total acumulado.

Após o processamento de todos os ingredientes, os valores obtidos são arredondados e atribuídos ao objeto *InformacaoNutricionalDTO*, que é então retornado como resultado do método. Dessa forma, o serviço centraliza a lógica responsável pelo cálculo das informações nutricionais e disponibiliza os resultados para outras camadas da aplicação, como os controladores da API.

```
1 package com.api.nutritruck.service;
```

```java
import com.api.nutritruck.dto.InformacaoNutricionalDTO;
import com.api.nutritruck.model.Alimento;
import com.api.nutritruck.model.PratoAlimento;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class CalculoNutricionalService {

    public InformacaoNutricionalDTO calcular(List<PratoAlimento>
       ingredientes) {
        InformacaoNutricionalDTO total = new InformacaoNutricionalDTO
           ();

        double calorias = 0;
        double carboidratos = 0;
        double proteinas = 0;
        double gorduras = 0;
        double fibras = 0;
        double sodio = 0;

        for (PratoAlimento item : ingredientes) {

            double fator = item.getQuantidadeGramas() / 100;

            Alimento ing = item.getAlimento();

            calorias += ing.getCalorias100g() * fator;
            carboidratos += ing.getCarboidratos100g() * fator;
            proteinas += ing.getProteinas100g() * fator;
            gorduras += ing.getGorduras100g() * fator;
            fibras += ing.getFibras100g() * fator;
            sodio += ing.getSodio100g() * fator;
        }

        total.setCalorias((double) Math.round(calorias));
        total.setCarboidratos(arredondar(carboidratos));
        total.setProteinas(arredondar(proteinas));
        total.setGorduras(arredondar(gorduras));
        total.setFibras(arredondar(fibras));
        total.setSodio(arredondar(sodio));

        return total;
    }

    private double arredondar(double valor) {
        return Math.round(valor * 10.0) / 10.0;
    }
}
```

Quadro 4.1: Cálculo nutricional a partir da tabela TACO

4.2.4 Segurança da Aplicação

A segurança do sistema foi implementado utilizando o *framework spring security* com a autenticação baseada A JSON Web Token (JWT) e controle de acesso por perfis de usuário.

Autenticação

A autenticação dos usuários no sistema é realizada por meio do protocolo OAuth2, utilizando a conta do google como provedor de identidade. Durante esse processo no qual o usuário realiza o login com sua conta google e as credenciais são validadas pelos servidores do provedor de autenticação, é gerado um token JWT contendo informações com email, perfil e função do usuário. Esse token é enviado no cabeçalho das requisições no formato *Bearer Token*.

O processo de autenticação da API é realizado por meio de um endpoint responsável por receber as credenciais do usuário e encaminhá-las para o serviço de autenticação. O Quadro 4.2 apresenta a implementação do endpoint responsável pelo login no sistema.

```
1 @PostMapping("/login")
2 public LoginResponse login(@RequestBody LoginRequest request) {
3     return authService.autenticar(request);
4 }
```

Quadro 4.2: Endpoint de Autenticação

O código apresentado no Quadro 4.2 demonstra a implementação do endpoint responsável pela autenticação dos usuários no sistema. Na linha 1, a anotação @PostMapping("/login") define que o método será acessado por meio de uma requisição HTTP do tipo POST no caminho /login.

Na linha 2, é declarado o método login, que recebe um objeto do tipo LoginRequest. A anotação @RequestBody indica que os dados enviados na requisição serão convertidos automaticamente para esse objeto.

Na linha 3, o método chama o serviço de autenticação authService, responsável por validar as credenciais do usuário. Como retorno, é enviado um objeto LoginResponse, contendo as informações da autenticação, como o token de acesso utilizado nas demais requisições da API.

Validação do Token

A validação do token é feita por um filtro personalizado (JwtFilter), ele é responsável por verificar a autenticidade do token e inserir as permissões no contexto de segurança. O Quadro

4.3 apresenta a implementação do filtro responsável pela verificação do token de autenticação.

O código apresentado no Quadro 4.3 demonstra a implementação do filtro responsável por validar o token JWT em cada requisição realizada à API. Na linha 2, a classe JwtFilter estende OncePerRequestFilter, garantindo que o filtro seja executado uma única vez para cada requisição.

Entre as linhas 5 e 8 é definido o método doFilterInternal, responsável por interceptar as requisições HTTP. Na linha 10, o código obtém o cabeçalho Authorization, onde o token de autenticação é enviado pelo cliente.

Nas linhas 12 a 15 é realizada a verificação inicial do cabeçalho. Caso ele esteja ausente ou não esteja no formato Bearer, a requisição segue normalmente sem autenticação. Na linha 17, o token JWT é extraído do cabeçalho.

Entre as linhas 19 e 25 ocorre a validação do token por meio do serviço jwtService, que também extrai informações do usuário, como e-mail, perfil e função. Em seguida, nas linhas 27 a 34, são definidas as permissões de acesso do usuário utilizando objetos do tipo SimpleGrantedAuthority.

Por fim, nas linhas 36 a 41 é criado o objeto de autenticação do usuário e ele é registrado no contexto de segurança da aplicação. Dessa forma, as demais partes do sistema conseguem identificar o usuário autenticado e aplicar as regras de autorização necessárias.

```
1  @Component
2  public class JwtFilter extends OncePerRequestFilter {
3      @Override
4      protected void doFilterInternal(HttpServletRequest request,
5                                      HttpServletResponse response,
6                                      FilterChain filterChain)
7              throws ServletException, IOException {
8
9          final String authHeader = request.getHeader("Authorization");
10
11         if (authHeader == null || !authHeader.startsWith("Bearer "))
               {
12             filterChain.doFilter(request, response);
13             return;
14         }
15
16         final String token = authHeader.substring(7);
17
18         if (jwtService.isValid(token)) {
19
20             String email = jwtService.extractEmail(token);
21             String role = jwtService.getClaims(token)
22                                     .get("role", String.class);
```

```
23              String funcao = jwtService.getClaims(token)
24                                      .get("funcao", String.class);
25
26              List<SimpleGrantedAuthority> authorities = new ArrayList
                    <>();
27              authorities.add(new SimpleGrantedAuthority("ROLE_" + role
                    ));
28
29              if (funcao != null && !funcao.isBlank()) {
30                  authorities.add(
31                      new SimpleGrantedAuthority(funcao.toUpperCase())
32                  );
33              }
34
35              UsernamePasswordAuthenticationToken authentication =
36                  new UsernamePasswordAuthenticationToken(
37                      usuario, null, authorities);
38
39              SecurityContextHolder.getContext()
40                                   .setAuthentication(authentication);
41          }
42
43          filterChain.doFilter(request, response);
44      }
45  }
```

Quadro 4.3: Filtro de autenticação utilizando JWT

Autorização

O controle de acesso do sistema é realizado por meio da definição de perfis de usuário, que determinam quais recursos podem ser acessados dentro da aplicação. Esses perfis são representados por uma enumeração chamada *Role*.

O código apresentado no Quadro 4.4 demonstra a definição da enumeração Role, utilizada para representar os perfis de acesso existentes no sistema. Na linha 1 é declarada a enumeração Role, que define um conjunto fixo de valores utilizados para identificar o tipo de usuário da aplicação.

Entre as linhas 2 e 4 são definidos os perfis disponíveis: ADMIN, responsável pela administração do sistema; FUNCIONARIO, utilizado para usuários que desempenham funções operacionais no restaurante; e CLIENTE, destinado aos usuários que acessam o sistema para realizar pedidos ou reservas.

```
public enum Role {
    ADMIN,
    FUNCIONARIO,
    CLIENTE
}
```

Quadro 4.4: Enum Role

A utilização dessa enumeração permite padronizar o controle de acesso dentro da aplicação, facilitando a implementação das regras de autorização e garantindo que cada usuário possua permissões adequadas de acordo com seu perfil.

As permissões de acesso aos endpoints são configuradas na classe de segurança da aplicação.

O código apresentado no Quadro 4.5 demonstra um trecho da configuração de segurança responsável por definir as permissões de acesso aos endpoints da aplicação. Na linha 1 é iniciada a configuração das regras de autorização das requisições HTTP.

Na linha 2, o endpoint ‘/auth/**‘ é liberado para acesso público por meio do método ‘permitAll()‘, permitindo que qualquer usuário acesse rotas relacionadas à autenticação, como o login.

Entre as linhas 3 e 4 é definida uma regra que permite a realização de requisições do tipo POST no endpoint ‘/pedido/**‘ apenas para usuários que possuem o perfil ‘CLIENTE‘.

Nas linhas 5 e 6, é estabelecido que os endpoints iniciados por ‘/admin/**‘ só podem ser acessados por usuários com o perfil ‘ADMIN‘.

Já nas linhas 7 e 8, o endpoint ‘/caixa/**‘ exige que o usuário possua a autoridade ‘CAIXA‘, relacionada a uma função específica dentro do sistema.

Por fim, na linha 9, a regra ‘anyRequest().authenticated()‘ determina que qualquer outra requisição não especificada nas regras anteriores exige que o usuário esteja autenticado para acessar o recurso.

```
.authorizeHttpRequests(auth -> auth
    .requestMatchers("/auth/**").permitAll()
    .requestMatchers(HttpMethod.POST, "/pedido/**")
        .hasRole("CLIENTE")
    .requestMatchers("/admin/**")
        .hasRole("ADMIN")
    .requestMatchers("/caixa/**")
        .hasAuthority("CAIXA")
    .anyRequest().authenticated()
)
```

Quadro 4.5: Trecho da Configuração de Segurança

4.2.5 Banco de Dados

O sistema utilizado foi o PostgreSQL. O mesmo foi escolhido por sua confiabilidade, robustez, sendo muito usado em aplicações de médio e grande porte. Foi projetado para armazenar as informações gerenciais do sistema e os dados nutricionais provenientes da tabela Taco.

A tabela TACO foi incorporada ao banco de dados com o objetivo de fornecer uma base padronizada de valores nutricionais, o que permite que a API realize o cálculo das calorias e nutrientes dos pratos.

As tabelas presentes no banco de dados foram criadas automaticamente a partir do Spring Boot, por meio do mapeamento das entidades da aplicação, permitindo, assim, a geração do esquema do banco de dados. O uso do mapeamento objeto-relacional garante a integração eficiente entre a camada de persistência e a API.

Para realizar o mapeamento entre as classes da aplicação e as tabelas do banco de dados, foi utilizada a especificação JPA (Java Persistence API), implementada pelo Hibernate. Nesse processo, as entidades da aplicação são anotadas com metadados que indicam como seus atributos devem ser armazenados no banco de dados.

O Quadro 4.6 apresenta um exemplo de entidade utilizada no sistema e seu respectivo mapeamento para uma tabela no banco de dados.

```
1  package com.api.nutritruck.model;
2
3  @Entity
4  @Table(name = "pedido")
5  public class Pedido {
6
7      @Id
8      @GeneratedValue(strategy = GenerationType.IDENTITY)
9      private Long id;
10
11     @Column(length = 255)
12     private String observacoes;
13
14     @Enumerated(EnumType.STRING)
15     @Column(nullable = false)
16     private StatusPedido status;
17
18     @JsonIgnoreProperties({"hibernateLazyInitializer", "handler"})
19     @ManyToOne(fetch = FetchType.LAZY)
20     @JoinColumn(name = "cliente_id", nullable = false)
21     private Cliente cliente;
22
```

```
23      @ManyToOne(fetch = FetchType.LAZY)
24      @JoinColumn(name = "mesa_id", nullable = true)
25      private Mesa mesa;
26
27
28
29      @Column(name = "data_finalizacao")
30      private LocalDateTime dataFinalizacao;
31
32      @Column(name = "data_pagamento")
33      private LocalDateTime dataPagamento;
34
35
36
37
38      @OneToMany(mappedBy = "pedido", cascade = CascadeType.ALL,
            orphanRemoval = true)
39      private List<PedidoPrato> pedidoPratos;
40
41      @OneToMany(mappedBy = "pedido", cascade = CascadeType.ALL,
            orphanRemoval = true )
42      private List<ItemPedido> itens = new ArrayList<>();
```

Quadro 4.6: entidade de pedido

Como exemplo, o Quadro 4.6 apresenta um trecho da entidade Pedido. A anotação @Entity indica que a classe representa uma tabela no banco de dados, enquanto @Table(name = "pedido") define o nome da tabela.

O atributo id é definido como chave primária por meio da anotação @Id, e @GeneratedValue indica que seu valor é gerado automaticamente pelo banco. As anotações @Column definem propriedades das colunas, como tamanho ou obrigatoriedade.

Os relacionamentos entre tabelas também são definidos por anotações, como @ManyToOne, que estabelece a relação entre um pedido e o cliente responsável pela solicitação, além da possível associação com uma mesa do restaurante. Já a anotação @OneToMany representa o relacionamento entre um pedido e os itens que o compõem.

No banco de dados, essa entidade é representada pela tabela *pedido*, que possui colunas como *id*, *observacoes*, *status*, *data_finalizacao* e *data_pagamento*. Além disso, a tabela contém chaves estrangeiras, como *cliente_id* e *mesa_id*, que estabelecem o relacionamento com as tabelas *cliente* e *mesa*.

Os itens associados a cada pedido são armazenados na tabela *item_pedido*, que contém atributos como *id*, *quantidade*, *preco_unitario* e *subtotal*. Essa tabela possui as chaves estrangeiras *pedido_id* e *prato_id*, que relacionam cada item ao pedido correspondente e ao prato

selecionado. Dessa forma, o mapeamento objeto-relacional permite representar no banco de dados as relações existentes entre pedidos, pratos e clientes dentro do sistema.

4.3 Implementação do Front-end

O desenvolvimento do *front-end* foi realizado com foco na usabilidade, na integração com o *back-end* e na organização visual, garantindo que as funcionalidades do sistema fossem acessadas de forma clara e objetiva.

4.3.1 Configuração Inicial

A implementação do *front-end* foi iniciada na IDE Visual Studio Code utilizando o *framework* Angular, cuja linguagem principal é o TypeScript, permitindo a execução da aplicação nos navegadores e a interação com a API desenvolvida para comunicação com o sistema.

Inicialmente, foi preparado o ambiente de desenvolvimento com a instalação das dependências e a configuração do gerenciador de pacotes, o que possibilitou o controle e a manutenção das bibliotecas utilizadas ao longo do projeto.

O projeto Angular foi estruturado seguindo as boas práticas recomendadas, com a divisão entre componentes, serviços e módulos, o que permite uma melhor organização do código, facilitando a escalabilidade e a manutenção da aplicação. O uso de componentes *standalone* possibilitou uma organização mais direta, na qual cada componente declara explicitamente suas dependências.

Por outro lado, os serviços foram utilizados para centralizar toda a comunicação com o *back-end*, enquanto os componentes ficaram responsáveis pela construção das interfaces e pela interação com o usuário.

Ao longo do desenvolvimento, também foram configurados os mecanismos de roteamento, possibilitando a navegação entre as diferentes telas da aplicação, bem como o consumo da API por meio de requisições HTTP. Além disso, foram definidos padrões iniciais de estilização e organização visual, garantindo consistência entre as telas desenvolvidas.

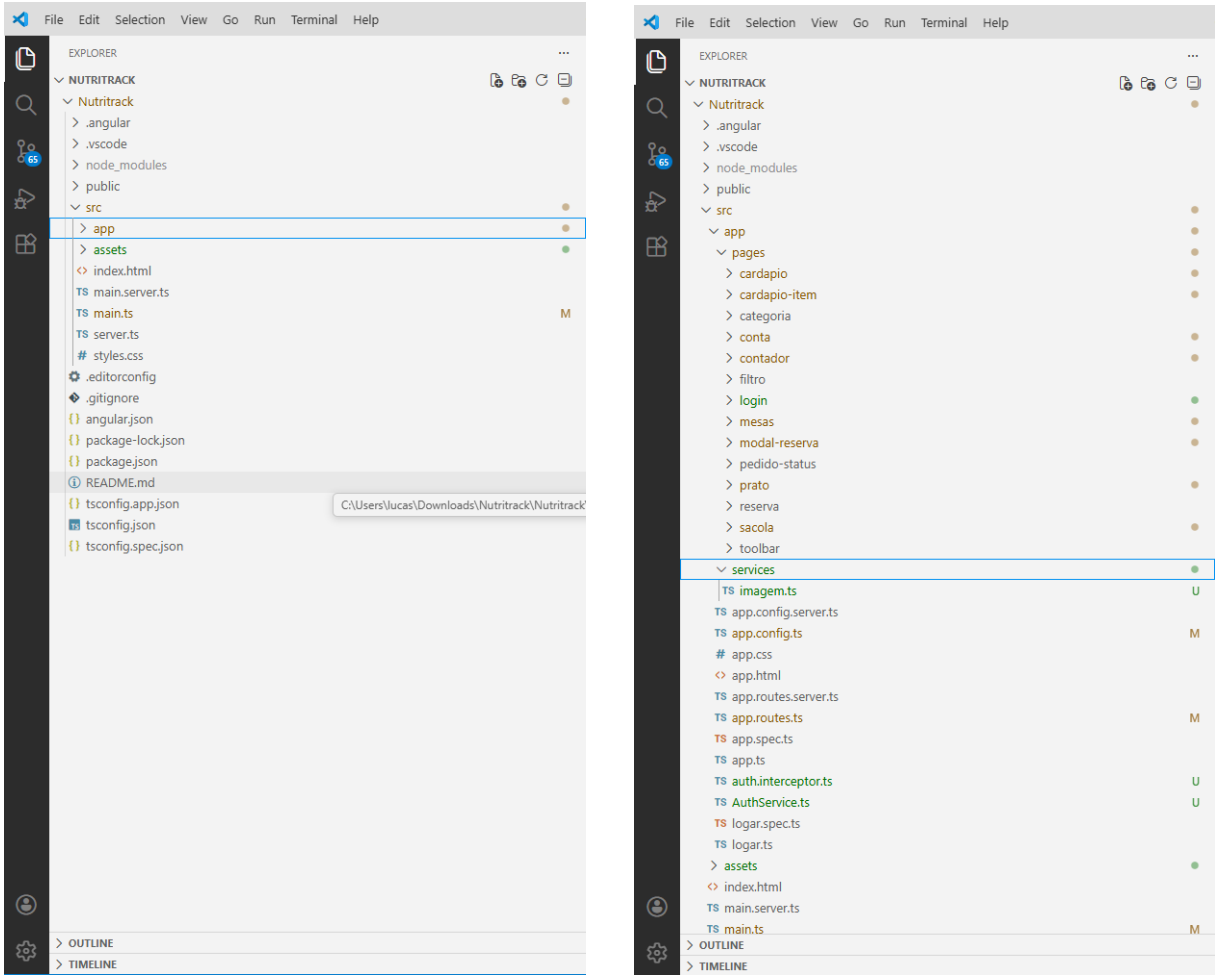
4.3.2 Estrutura das Pastas

Foi utilizado o Angular para o desenvolvimento do *front-end* do sistema, adotando a estrutura padrão gerada pelo CLI da ferramenta. A pasta principal contém os arquivos de

configuração essenciais para o funcionamento da aplicação, como *package.json*, *angular.json* e *node_modules*, que armazena todas as bibliotecas utilizadas pelo projeto.

A pasta *src* concentra o código-fonte da aplicação. Nela estão localizados os componentes, páginas e serviços do sistema. Os componentes são responsáveis por definir a interface de cada parte da aplicação, como cartões de pratos, formulários de reserva ou tabelas de pedidos.

As páginas (ou *views*) organizam os componentes em telas completas, como login, cardápio e reserva de mesas. Já os serviços são responsáveis pela comunicação com o *back-end*, realizando requisições HTTP para os endpoints da API, como criar pedidos, listar pratos ou consultar reservas.



(a) Estrutura geral do front-end **(b)** Subpacotes do front-end

Figura 4.9: Estrutura de pacotes do front-end

4.3.3 Exemplo de Implementação no Front-end

Além da organização em componentes e serviços, o front-end também implementa mecanismos de segurança e comunicação com a API do sistema. Um exemplo disso é o uso de

um *Interceptor* no Angular, responsável por interceptar automaticamente as requisições HTTP realizadas pela aplicação.

Esse recurso permite adicionar o token de autenticação no cabeçalho das requisições enviadas ao back-end, garantindo que apenas usuários autenticados possam acessar determinadas funcionalidades do sistema.

O Quadro 4.7 apresenta um exemplo de implementação desse interceptor, no qual o token é obtido pelo serviço de autenticação e incluído no cabeçalho da requisição utilizando o padrão *Bearer Token*.

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(private authService: AuthService) {}

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<
     HttpEvent<any>> {

    const token = this.authService.getToken();

    if (token) {
      const cloned = req.clone({
        headers: req.headers.set('Authorization', `Bearer ${token}`)
      });
      return next.handle(cloned);
    }

    return next.handle(req);
  }
}
```

Quadro 4.7: Interceptor de autenticação utilizado no front-end

4.4 Resultado Final da Aplicação

Nessa seção é apresentado o sistema implementado, destacando as principais telas e trechos de código responsáveis pelo funcionamento da aplicação.

4.4.1 Visão Geral do Sistema Funcionando

No desenvolvimento do sistema foi possível implementar uma aplicação funcional constituída pelo *front-end* e *back-end* integrados. O sistema permite visualizar o cardápio, consultar informações nutricionais de cada prato, adicionar e excluir itens do pedido, acompanhar pedidos

realizados e efetuar reservas de mesas. Essas operações, realizadas na interface, são processadas pela API e armazenadas no banco de dados.

A Figura 4.10 apresenta os pratos cadastrados no cardápio.

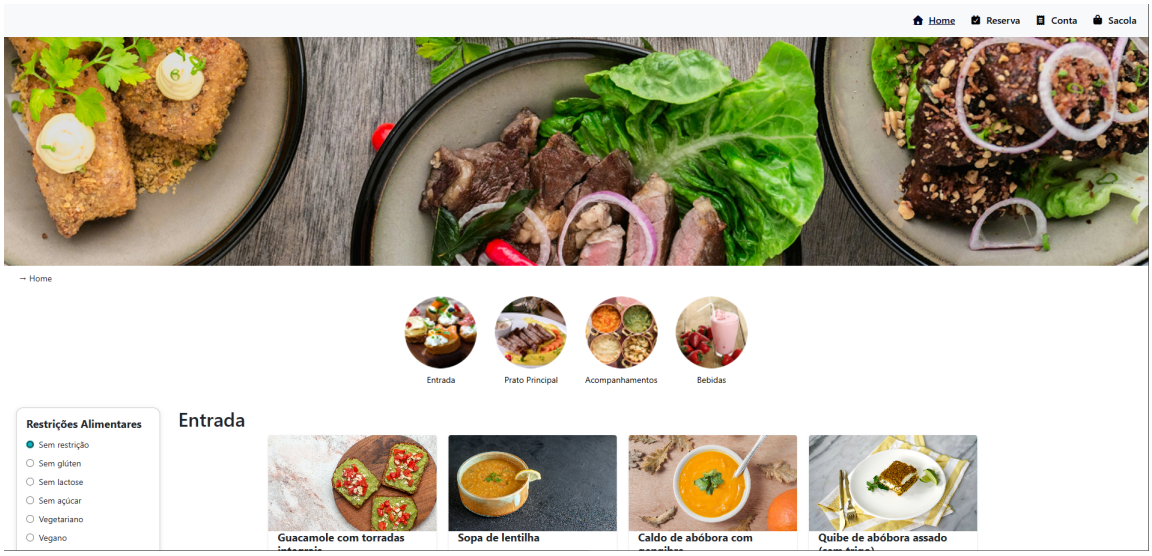


Figura 4.10: cardapio

O funcionamento dessa tela ocorre por meio de uma requisição HTTP realizada pelo *front-end* ao *back-end* para buscar os pratos cadastrados no sistema. O Quadro 4.8 apresenta um exemplo dessa requisição implementada no *front-end* da aplicação.

```
buscarPrato(id: string) {
  this.http.get<any>(`http://localhost:8080/pratos/${id}`)
    .subscribe(prato => this.prato = prato);
}
```

Quadro 4.8: Requisição para buscar pratos cadastrados

A Figura 4.11 apresenta o prato selecionado e suas informações nutricionais.

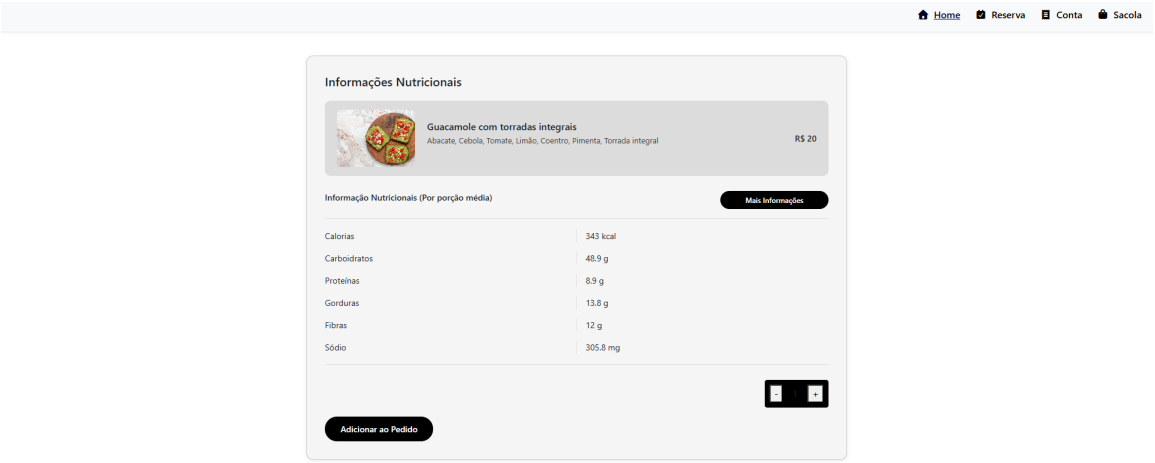


Figura 4.11: prato-nutricional

As informações nutricionais exibidas na interface são obtidas por meio de uma requisição à API responsável por retornar os dados nutricionais do prato selecionado. O Quadro 4.9 apresenta um exemplo dessa requisição realizada no front-end da aplicação.

```
buscarNutricao(id: string) {
  this.http.get<any>(`http://localhost:8080/pratos/${id}/nutricao`)
    .subscribe(n => this.nutricao = n);
}
```

Quadro 4.9: Requisição para buscar informações nutricionais

A Figura 4.12 apresenta o formulário para ser feita a reserva.

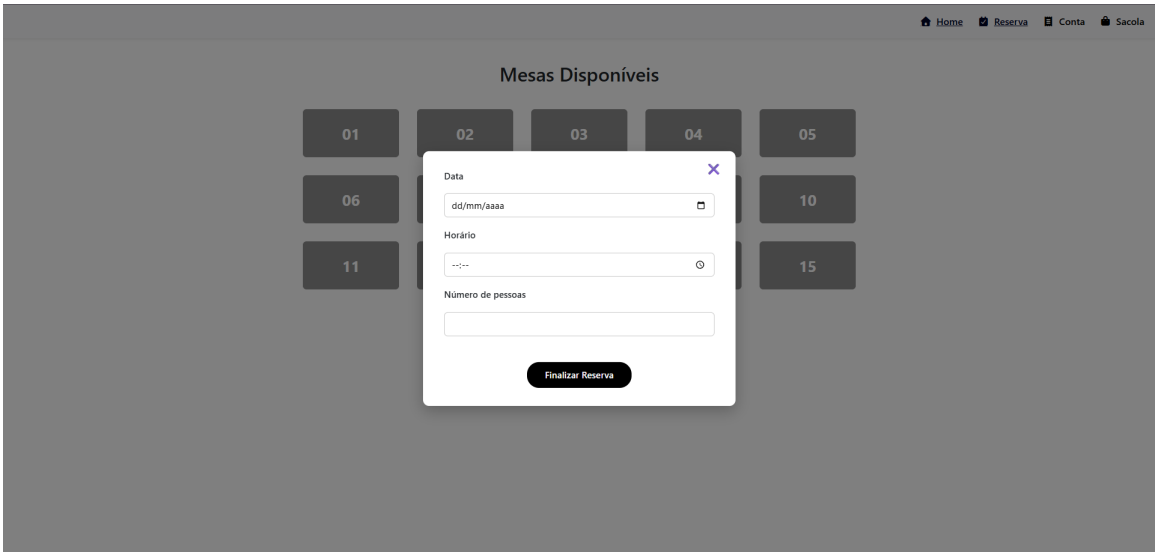


Figura 4.12: Reserva

Ao preencher o formulário, os dados são enviados para o back-end por meio de uma

requisição HTTP responsável por registrar a reserva no sistema. O Quadro 4.10 apresenta um exemplo dessa requisição utilizada para realizar o cadastro de uma reserva na aplicação.

```
this.http.post('http://localhost:8080/reserva', dadosReserva)
  .subscribe({
    next: () => console.log("Reserva realizada"),
    error: (err) => console.error(err)
  });
```

Quadro 4.10: Exemplo de envio de reserva

CAPÍTULO 5

CONCLUSÃO

Este trabalho apresentou o desenvolvimento de uma API para gerenciamento de pedidos de restaurante e controle de comandas, integrada a um sistema web com *front-end* voltado ao cliente, com destaque para a disponibilização de informações nutricionais e filtros relacionados a restrições alimentares. A proposta surgiu a partir da análise de sistemas existentes no mercado, como Consumer e Anota AI, que, apesar de oferecerem funcionalidades administrativas relevantes, ainda apresentam limitações quanto à usabilidade e à transparência das informações nutricionais.

O desenvolvimento do sistema foi estruturado em etapas bem definidas, iniciando pelo levantamento e análise de requisitos, o que possibilitou compreender as necessidades do domínio, identificar os perfis de usuários (cliente, atendente e administrador), definir requisitos funcionais e não funcionais e estabelecer as regras de negócio. A modelagem do sistema consolidou a estrutura conceitual e serviu de base para a implementação da solução proposta.

A arquitetura adotada combinou uma API REST com implementação segura e organizada, baseada na Tabela Brasileira de Composição de Alimentos (TACO) para obtenção das informações nutricionais, e um *front-end* voltado ao cliente, desenvolvido com foco em usabilidade, acessibilidade e clareza das informações. Essa separação entre *back-end* e *front-end* permitiu maior escalabilidade, modularidade e facilidade de manutenção, além de proporcionar uma experiência funcional e intuitiva para os usuários. Entre as funcionalidades implementadas, destacam-se: gerenciamento de clientes, pedidos, pratos, reservas e comandas; filtros nutricionais e de restrições alimentares; e integração completa entre *back-end* e *front-end*.

Os resultados obtidos demonstram que o sistema atende aos objetivos propostos, ofere-

cendo uma solução moderna e funcional que combina eficiência operacional e transparência nutricional. Além disso, o desenvolvimento do projeto contribuiu significativamente para o crescimento acadêmico e profissional, proporcionando experiência prática em arquitetura de sistemas, modelagem, segurança, banco de dados, desenvolvimento de APIs e construção de interfaces. O trabalho exigiu tomada de decisão técnica, resolução de problemas e constante aprimoramento, consolidando competências essenciais na área de tecnologia.

Apesar dos resultados alcançados, algumas limitações foram identificadas. O sistema ainda não contempla integração direta com meios de pagamento na API, nem funcionalidades avançadas de análise de dados e geração de relatórios estratégicos para o restaurante. Além disso, o levantamento de requisitos foi realizado com um número limitado de estabelecimentos, o que pode restringir a generalização de algumas funcionalidades implementadas.

5.1 Trabalhos Futuros

Como trabalhos futuros, sugerem-se:

- Desenvolvimento de um painel administrativo completo, com *dashboards* e indicadores estratégicos, como volume de pedidos, faturamento, controle de reservas e relatórios analíticos.
- Ampliação dos mecanismos de autenticação, incluindo login via e-mail e senha, aumentando a segurança e a flexibilidade do sistema.
- Expansão da base de dados nutricional, integrando novas fontes de informação para oferecer maior detalhamento dos pratos.
- Integração de meios de pagamento digital, permitindo que os pedidos sejam concluídos de forma totalmente online.
- Implementação de documentação interativa da API por meio do *Swagger*, facilitando testes e validação de *endpoints*, parâmetros e métodos HTTP.
- Integração com plataformas de *delivery*, ampliando o alcance do restaurante e automatizando processos operacionais.

Essas evoluções têm o potencial de tornar o sistema mais robusto, escalável e aderente às demandas contemporâneas do setor alimentício, consolidando a proposta como uma solução

tecnológica completa que alia gestão eficiente, transparência nutricional e responsabilidade social.

REFERÊNCIAS BIBLIOGRÁFICAS

Anota AI. *Sistema de automação de pedidos para restaurantes e delivery*. 2026. <https://anota.ai/>. Acesso em: 5 mar. 2026.

ARAúJO, H. M. C.; ARAúJO, W. M. C.; BOTELHO, R. B. A.; ZANDONADI, R. P. Doença celíaca, hábitos e práticas alimentares e qualidade de vida. *Revista de Nutrição*, Campinas, v. 23, n. 3, p. 467–474, maio/jun. 2010. Acesso em: 16 out. 2025.

ARAúJO, M. A. P. Modelagem de dados – teoria e prática. *Saber Digital*, v. 1, n. 1, p. 27–64, 2008.

(ASBAI), A. B. de Alergia e I. *Alergia Alimentar*. São Paulo: Associação Brasileira de Alergia e Imunologia, 2023. <https://asbai.org.br/alergia-alimentar-perguntas-e-respostas/>. Departamento Científico de Alergia Alimentar. Acesso em: 16 out. 2025.

BAGLIOTTI I. R.; GIBERTONI, D. Reusabilidade no desenvolvimento de um sistema web utilizando o framework angular. *Interface Tecnológica*, Taquaritinga, São Paulo, Brasil, v. 17, n. 1, 2020. Acesso em: 18 out. 2025.

Best Software. *Best Restaurante: sistema de gestão para restaurantes*. 2026. Disponível em: https://centraldosoftware.com.br/restaurante. Acesso em: 9 mar. 2026.

BLöCHER K.; ALT, R. Ai and robotics in the european restaurant sector: Assessing potentials for process innovation in a high-contact service industry. *Electronic Markets*, v. 31, p. 529–551, 2021. Acesso em: 25 fev. 2026.

BRAGA V, T. Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia Acesso em: 21 out. 2025, *Desenvolvimento do protótipo de uma API para integração dos pedidos de aplicativos de delivery e sistemas próprios em restaurantes*. Braga: [s.n.], 2023. Orientador: Paulo Henrique Ribeiro Gabriel.

CANCIAN R. L.; CANAL, A. P. *Desenvolvimento de uma API REST utilizando os princípios SOLID para proporcionar o acesso a dados públicos para aplicações no Brasil*. 2022. Trabalho de Conclusão de Curso - Curso de Bacharelado em Ciência da Computação, Universidade Franciscana (UFN), Santa Maria, RS, Brasil. Disponível em: <https://tfgonline.lapinf.ufn.edu.br/media/midias/Artigo_V644bJn.pdf>. Acesso em: 21 out. 2025.

Conselho Federal de Nutricionistas (CFN). *Pesquisa mostra que 80% dos brasileiros buscam alimentação saudável*. 2018. <https://cfn.org.br/

pesquisa-mostra-que-80-dos-brasileiros-buscam-alimentacao-saudavel/>. Acesso em: 27 fev. 2026.

Consumer. *Sistema para restaurantes e delivery*. 2026. <https://consumer.com.br/>. Acesso em: 5 mar. 2026.

DATE, C. J. *Introdução a Sistemas de Bancos de Dados*. 8ª edição. ed. Rio de Janeiro: [s.n.], 2003. Tradução autorizada da edição americana *An Introduction to Database Systems*. ISBN 978-85-352-8445-4.

DIETBOX. *Planos*. 2025. <https://pay.dietbox.me/>. Acesso em: 4 nov. 2025.

ELMASRI R.; NAVATHE, S. B. *Sistemas de Banco de Dados*. 4. ed. São Paulo: Pearson Addison Wesley, 2005.

FIGMA. *Recently Viewed / Compartilhados*. 2025. <https://www.figma.com/@figma>. Acesso em: 25 fev. 2026.

GENDRON J.; GENDRON, B. W. E. Hospitality, big data, and restaurateurs: does simulation have a seat at the table? *MODSIM World*, v. 21, p. 24–26, 2018.

IBM. *O que é API REST?* 2017. <https://www.ibm.com/br-pt/think/topics/rest-apis>. Acesso em: 21 out. 2025.

JAYASINGHE S.; BYRNE, N. M. H. A. P. Cultural influences on dietary choices. *Progress in Cardiovascular Diseases*, Elsevier, Amsterdam, Netherlands, v. 90, p. 22–26, 2025. Acesso em: 25 fev. 2026.

JUNIOR E. A.; ROCHA, R. D. M. R. d. S. G. Desenvolvimento de api rest com spring boot. *Revista Científica do UniRios*, UniRios, v. 15, n. 29, 2021. Acesso em: 25 fev. 2026.

KOCAMAN, M. E. Operational effects of using restaurant management system: An assessment according to business features. *International Journal of Gastronomy and Food Science*, Elsevier, v. 25, p. 100408, 2021.

LIMA, L. D.; NASCIMENTO, V. A. M.; NETO, G. H. Desenvolvimento de um protótipo para controle de vendas de uma loja atuante no segmento de comercio de roupas. *Revista Eletrônica de Sistemas de Informação e Gestão Tecnológica*, Centro Universitario Municipal de Franca, v. 15, n. 2, 2025.

Ministério da Saúde (Brasil). *Entenda as diferenças entre alergia e intolerância alimentar*. 2022. <https://www.gov.br/saude/pt-br/assuntos/noticias/2022/setembro/entenda-as-diferencas-entre-alergia-e-intolerancia-alimentar>. Acesso em: 16 out. 2025.

Okta, Inc. *API Security Fundamentals*. [S.l.], 2023.

OLIVEIRA, K. M. A. d.; AGUIAR, Y. P.; JúNIOR, B. L.; CHAVES, L. C. R.; GUEDES, G.; VIEIRA, D. A.; CARVALHO, Y. O.; LIMA J. G. DE ALVES, M. d. O. O uso de modelos e múltiplos protótipos na concepção de interface do usuário. *Principia*, Universidade Federal da Paraíba, João Pessoa, n. 15, p. 15–29, dez. 2007.

OWASP API Security Project. *OWASP API Security Top 10 — 2019*. 2019. <https://owasp.org/API-Security/editions/2019/pt-BR/dist/owasp-api-security-top-10-pt-br.pdf>. Disponível em: <https://owasp.org/API-Security/editions/2019/pt-BR/dist/owasp-api-security-top-10-pt-br.pdf>. Acesso em: 25 fev. 2026.

PINTO, F. A. O. *Frameworks: Conceitos Gerais*. Dissertação (Dissertação de Mestrado em Informática) — Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, jul. 2006. Capítulo 2 da dissertação sobre o uso de frameworks e reuso de software, abordando classificações, papéis, benefícios e desafios de adoção.

PostgreSQL Global Development Group. *PostgreSQL*. 2025. Disponível em: <https://www.postgresql.org/>. Acesso em: 16 jan. 2026. Disponível em: <https://www.postgresql.org/>.

SILVA, J. V. F. S. *Comparação de desempenho entre os bancos de dados PostgreSQL e Neo4j para acesso a dados complexos*. Dissertação (Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação)) — Universidade Federal de Uberlândia, Uberlândia, MG, Brasil, 2023.

SOMMERVILLE, I. *Engenharia de Software*. 10. ed. São Paulo: Pearson, 2019. Acesso em: 12 mar. 2026. Disponível em: <https://www.facom.ufu.br/~william/Disciplinas%202018-2/BSI-GSI030-EngenhariaSoftware/Livro/engenhariaSoftwareSommerville.pdf>.

TACO. *Tabela Brasileira de Composição de Alimentos (TACO)*. 2020. <https://cfn.org.br/wp-content/uploads/2017/03/taco_4_edicao_ampliada_e_revisada.pdf>. Acesso em: 31 maio 2024.

TURINE M. A.; MASIERO, P. C. *Especificação de Requisitos: Uma Introdução*. São Carlos, SP, Brasil, 1996.