ESPECIFICAÇÃO DE REQUISITOS DE SOFTWARE (SRS)

APLICATIVO MOBILE DE CUPCAKES GOURMET

Versão: 2.5
Data: 17/11/2025
Autor: Vinicius Alves LEon Rodriguez

Status: Aprovado

Conformidade:

- IEEE 830-1998 - Recommended Practice for Software Requirements Specifications
- IEEE 29148-2018 - Systems and Software Engineering
- ABNT NBR ISO/IEC/IEEE 29148:2013 - Engenharia de requisitos
- ABNT NBR ISO/IEC 12207:2017 - Processos do ciclo de vida de software
- ISO/IEC 25010:2011 - Software product quality model

ÍNDICE

1. INTRODUÇÃO
2. DESCRIÇÃO GERAL
3. REQUISITOS ESPECÍFICOS
4. MODELAGEM DO SISTEMA
5. REQUISITOS DE INTERFACE
6. REQUISITOS NÃO-FUNCIONAIS
7. ARQUITETURA E DESIGN PATTERNS
8. SEGURANÇA E COMPLIANCE
9. GESTÃO DO PROJETO
10. ANEXOS

1. INTRODUÇÃO

1.1 Propósito do Documento

Este documento especifica todos os requisitos funcionais e não funcionais do **Aplicativo Mobile de Cupcakes Gourmet**, com o objetivo de orientar o desenvolvimento, validação e manutenção do sistema. Destina-se a desenvolvedores, testadores, gerentes de projeto, stakeholders e equipe de manutenção.

1.2 Escopo do Produto

O **Aplicativo Mobile de Cupcakes Gourmet** é uma plataforma integrada de e-commerce que visa revolucionar a experiência de compra de cupcakes gourmet, oferecendo:

Para Clientes:

- Catálogo digital com ≥40 sabores (doces e salgados)
- Pedidos personalizados com rastreamento em tempo real
- Pagamentos seguros via Pix e cartão de crédito
- Recomendações inteligentes baseadas em ML/LLMs
- Notificações push sobre status de pedidos
- Avaliação e feedback de produtos

Para Administradores:

- Gerenciamento completo de catálogo e estoque
- Controle de ingredientes e custos unitários
- Dashboards analíticos com KPIs de vendas
- Gestão de usuários e entregadores
- Relatórios gerenciais e financeiros
- Monitoramento de segurança e logs

Para Entregadores:

- Visualização de pedidos atribuídos
- Atualização de status em tempo real
- Navegação integrada
- Histórico de entregas

1.3 Definição do Problema

A loja de cupcakes gourmet enfrenta desafios operacionais críticos:

- **Gestão Manual Ineficiente:** Pedidos via telefone/WhatsApp causam erros e atrasos
- **Falta de Rastreabilidade:** Clientes sem visibilidade do status do pedido
- **Estoque Desatualizado:** Dificuldade em controlar ingredientes e custos
- **Ausência de Análise:** Sem dados para decisões estratégicas

- **Experiência Limitada:** Cliente sem personalização ou recomendações

Solução Proposta:
Plataforma mobile integrada que une compras, pagamentos, notificações, gestão operacional e recomendações personalizadas via ML/LLMs.

1.4 Público-Alvo

Perfil	Descrição	Características
Clientes Finais	Consumidores de cupcakes gourmet	Usuários de smartphones (Android/iOS), idade 18-55 anos, conectados digitalmente
Administradores	Gestores da loja	Conhecimento básico de gestão, familiaridade com sistemas web
Entregadores	Responsáveis pela logística	Motoristas com smartphones, familiaridade com apps de navegação
Equipe Técnica	DevOps e desenvolvedores	Conhecimento em Python, Cloud, CI/CD, ML

1.5 Definições, Acrônimos e Abreviações

Termo	Definição
API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration/Continuous Delivery
DTO	Data Transfer Object
IaC	Infrastructure as Code
JWT	JSON Web Token
KPI	Key Performance Indicator
LLM	Large Language Model
ML	Machine Learning
MVP	Minimum Viable Product
OAuth2	Open Authorization 2.0
ORM	Object-Relational Mapping
RBAC	Role-Based Access Control
REST	Representational State Transfer
SRS	Software Requirements Specification
TDD	Test-Driven Development
UML	Unified Modeling Language
WCAG	Web Content Accessibility Guidelines

1.6 Referências

- IEEE 830-1998: Recommended Practice for Software Requirements Specifications
- IEEE 29148-2018: Systems and Software Engineering
- ABNT NBR ISO/IEC/IEEE 29148:2013: Engenharia de requisitos
- ISO/IEC 25010:2011: Software product quality model
- LGPD (Lei Geral de Proteção de Dados) - Lei nº 13.709/2018
- PCI-DSS: Payment Card Industry Data Security Standard
- Documento de Modelagem UML - Versão 1.0
- Documento de Casos de Uso - Versão 1.0

1.7 Visão Geral do Documento

Este SRS está organizado conforme as melhores práticas IEEE e ABNT:

- **Seção 1:** Contextualização e propósito
- **Seção 2:** Descrição geral do sistema e perspectivas
- **Seção 3:** Requisitos funcionais detalhados
- **Seção 4:** Modelagem UML e diagramas
- **Seção 5:** Especificação de interfaces
- **Seção 6:** Requisitos não-funcionais (performance, segurança, usabilidade)
- **Seção 7:** Arquitetura e padrões de design
- **Seção 8:** Segurança, compliance e LGPD
- **Seção 9:** Metodologia, cronograma e gestão
- **Seção 10:** Anexos técnicos

2. DESCRIÇÃO GERAL

2.1 Perspectiva do Produto

O Aplicativo Mobile de Cupcakes Gourmet é um sistema independente que integra múltiplos componentes:

TELA001 - Splash Screen

- Logo animado (2 segundos)
- Verificação de autenticação
- Transição suave

TELA002 - Onboarding (Primeira execução)

- 3 slides explicativos
- Botão "Pular" disponível
- Registro de conclusão

TELA003 - Login

- Campos: Email/Usuário, Senha
- Botões: "Entrar", "Esqueci minha senha"
- Link: "Criar conta"
- Opção: "Lembrar-me"

TELA004 - Cadastro

- Formulário em etapas:
  - Dados básicos (nome, email, senha)
  - Contato (telefone)
  - Endereço (opcional nesta etapa)
- Validação em tempo real
- Indicador de progresso

TELA005 - Home/Catálogo

- Header: Logo, busca, carrinho (badge com qtd)
- Carrossel de promoções
- Filtros: Categoria, Preço, Popularidade
- Grid de produtos (2 colunas)
- Bottom Navigation: Home, Pedidos, Perfil

TELA006 - Detalhes do Produto

- Galeria de imagens (swipe)
- Nome, preço, avaliação
- Descrição completa
- Ingredientes (lista)
- Info nutricional (expansível)
- Quantidade (contador)
- Botão: "Adicionar ao Carrinho" (fixo no bottom)

TELA007 - Carrinho

- Lista de itens com foto, nome, qtd, subtotal
- Campo para cupom de desconto

- Resumo: Subtotal, Taxa de entrega, Desconto, Total
- Botão: "Finalizar Pedido" (destaque)

TELA008 - Checkout

- Seleção de endereço
- Resumo do pedido
- Métodos de pagamento (Pix, Cartão)
- Observações (textarea)
- Botão: "Confirmar e Pagar"

TELA009 - Pagamento Pix

- QR Code centralizado
- Código "Pix Copia e Cola"
- Temporizador (10 minutos)
- Status: Aguardando pagamento

TELA010 - Pagamento Cartão

- Número do cartão (masked)
- Validade (MM/AA)
- CVV (password field)
- Nome do titular
- Botão: "Pagar R$ XX,XX"

TELA011 - Confirmação de Pedido

- Ícone de sucesso animado
- Número do pedido (destaque)
- Resumo do pedido
- Tempo estimado de entrega
- Botão: "Acompanhar Pedido"

TELA012 - Acompanhamento de Pedido

- Timeline vertical com status
- Mapa (quando "A Caminho")
- Dados do entregador (nome, foto, telefone)
- Tempo estimado atualizado
- Botão: "Avaliar Entrega" (após conclusão)

TELA013 - Meus Pedidos

- Abas: Em Andamento, Histórico
- Lista de pedidos com resumo
- Status visual (cores)
- Filtro por período

TELA014 - Perfil do Usuário

- Foto de perfil (editável)
- Dados pessoais
- Endereços salvos
- Métodos de pagamento
- Configurações
- Botão: "Sair"

TELA015 - Configurações

- Notificações (toggles)
- Modo escuro
- Idioma
- Sobre o app
- Termos e Política de Privacidade

5.2 Interface de Software

5.2.1 API REST - Especificação

Base URL: https://api.cupcakesgourmet.com/v1

Autenticação: Bearer Token (JWT)

Header padrão:

```
Authorization: Bearer {token}
Content-Type: application/json
Accept: application/json
```

Endpoints Principais:

```
POST   /auth/register          # Cadastro
POST   /auth/login             # Login
POST   /auth/refresh           # Renovar token
POST   /auth/reset-password    # Recuperar senha
```

```
GET    /products                # Listar produtos
GET    /products/:id            # Detalhar produto
POST   /products                # Criar produto (Admin)
PUT    /products/:id            # Atualizar produto (Admin)
DELETE /products/:id            # Deletar produto (Admin)

GET    /cart                    # Ver carrinho
POST   /cart/items              # Adicionar item
PUT    /cart/items/:id          # Atualizar item
DELETE /cart/items/:id          # Remover item

POST   /orders                  # Criar pedido
GET    /orders                  # Listar pedidos
GET    /orders/:id              # Detalhar pedido
PATCH  /orders/:id/status       # Atualizar status
(Admin/Entregador)
POST   /orders/:id/cancel       # Cancelar pedido

POST   /payments                # Processar pagamento
GET    /payments/:id            # Status do pagamento

GET    /users/profile           # Ver perfil
PUT    /users/profile           # Atualizar perfil
GET    /users/addresses         # Listar endereços
POST   /users/addresses         # Adicionar endereço

GET    /admin/dashboard         # Dashboard (Admin)
GET    /admin/reports           # Relatórios (Admin)
GET    /admin/users             # Listar usuários (Admin)

GET    /deliveries              # Pedidos atribuídos (Entregador)
POST   /deliveries/:id/accept   # Aceitar entrega (Entregador)
POST   /deliveries/:id/confirm  # Confirmar entrega (Entregador)
```

Respostas Padrão:

```
// Sucesso
{
  "success": true,
  "data": { ... },
  "message": "Operação realizada com sucesso"
}
```

```
// Erro
{
  "success": false,
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Email já cadastrado",
    "details": {
      "field": "email",
      "value": "user@example.com"
    }
  }
}
```

Códigos HTTP:

- 200: OK
- 201: Created
- 400: Bad Request
- 401: Unauthorized
- 403: Forbidden
- 404: Not Found
- 422: Validation Error
- 500: Internal Server Error

5.2.2 Integração com Mercado Pago

SDK: mercadopago-python v2.2.0

Fluxo de Pagamento:

1. Criar Preferência de Pagamento

```
import mercadopago

sdk = mercadopago.SDK(os.getenv("MERCADOPAGO_ACCESS_TOKEN"))

preference_data = {
    "items": [
        {
            "title": pedido.descricao,
            "quantity": 1,
            "unit_price": float(pedido.valor_total)
```

```
        }
    ],
    "payer": {
        "email": usuario.email
    },
    "back_urls": {
        "success":
"https://app.cupcakesgourmet.com/pagamento/sucesso",
        "failure":
"https://app.cupcakesgourmet.com/pagamento/falha",
        "pending":
"https://app.cupcakesgourmet.com/pagamento/pendente"
    },
    "notification_url":
"https://api.cupcakesgourmet.com/v1/webhooks/mercadopago"
}

result = sdk.preference().create(preference_data)
```

2. Webhook de Confirmação

```
@app.post("/webhooks/mercadopago")
async def mercadopago_webhook(request: Request):
    body = await request.json()

    if body["type"] == "payment":
        payment_id = body["data"]["id"]
        payment = sdk.payment().get(payment_id)

        if payment["status"] == 200:
            if payment["response"]["status"] == "approved":
                # Atualizar pedido como pago
                await
atualizar_status_pedido(payment["response"]["external_reference"],
"Pago")

    return {"received": True}
```

5.2.3 Integração Firebase Cloud Messaging

Envio de Notificação Push:

```
from firebase_admin import messaging, initialize_app

initialize_app()

def enviar_notificacao(device_token: str, titulo: str, corpo: str,
dados: dict):
    message = messaging.Message(
        notification=messaging.Notification(
            title=titulo,
            body=corpo,
        ),
        data=dados,
        token=device_token,
        android=messaging.AndroidConfig(
            priority='high',
            notification=messaging.AndroidNotification(
                icon='ic_notification',
                color='#E91E63'
            )
        ),
        apns=messaging.APNSConfig(
            payload=messaging.APNSPayload(
                aps=messaging.Aps(badge=1, sound='default')
            )
        )
    )

    response = messaging.send(message)
    return response
```

5.3 Interface de Hardware

5.3.1 Requisitos de Dispositivo Móvel

Smartphones Android:

- Sistema Operacional: Android 10.0 (API 29) ou superior
- RAM: Mínimo 2GB (Recomendado 4GB)
- Armazenamento: 100MB livres
- Processador: Quad-core 1.5 GHz ou superior
- GPS: Obrigatório para funcionalidades de entrega

- Câmera: Desejável (para comprovante de entrega)
- Conectividade: Wi-Fi ou 3G/4G/5G

Dispositivos iOS:

- Sistema Operacional: iOS 14.0 ou superior
- Dispositivos: iPhone 7 ou posterior
- RAM: Mínimo 2GB
- Armazenamento: 100MB livres
- GPS: Obrigatório
- Câmera: Desejável
- Conectividade: Wi-Fi ou 3G/4G/5G

5.3.2 Servidores Cloud (AWS)

Ambiente de Produção:

- **EC2 Instances:** t3.medium (2 vCPUs, 4GB RAM) - Auto Scaling 2-10 instâncias
- **RDS MySQL:** db.t3.medium (2 vCPUs, 4GB RAM) - Multi-AZ
- **DocumentDB (MongoDB):** t3.medium - 3 nodes
- **ElastiCache (Redis):** cache.t3.small
- **S3:** Standard storage para imagens
- **CloudFront:** CDN para entrega de assets
- **Lambda:** Serverless para processos assíncronos
- **ECS/EKS:** Orquestração de containers

5.4 Interface de Comunicação

5.4.1 Protocolos

- **HTTP/HTTPS:** REST API (TLS 1.2+)
- **WebSocket:** Atualizações em tempo real (rastreamento)
- **gRPC:** Comunicação entre microserviços
- **MQTT:** Telemetria de entregadores (GPS)

5.4.2 Formatos de Dados

- **JSON:** Padrão para APIs REST
- **Protocol Buffers:** Comunicação gRPC
- **Base64:** Encoding de imagens

5.4.3 Requisitos de Rede

- **Largura de Banda Mínima:** 3G (384 kbps)
- **Latência Máxima:** 500ms (tolerável)
- **Latência Ideal:** < 200ms
- **Modo Offline:** Suporte limitado (visualização de histórico)

6. REQUISITOS NÃO-FUNCIONAIS

6.1 Requisitos de Desempenho

ID	Requisito	Métrica	Prioridade
RNF01	Tempo de resposta de APIs	95% das requisições < 200ms	Alta
RNF02	Tempo de carregamento de tela	< 2 segundos	Alta
RNF03	Usuários simultâneos	Suportar 10.000 usuários ativos	Alta
RNF04	Throughput de transações	100 pedidos/minuto	Média
RNF05	Tempo de inicialização do app	< 3 segundos	Média
RNF06	Consulta ao catálogo	< 1 segundo	Alta
RNF07	Processamento de pagamento	< 5 segundos	Alta
RNF08	Upload de imagens	< 3 segundos (por imagem 2MB)	Baixa

6.2 Requisitos de Disponibilidade e Confiabilidade

ID	Requisito	Métrica	Prioridade
RNF09	Disponibilidade do sistema	99,5% (uptime mensal)	Alta
RNF10	Tempo de recuperação (MTTR)	< 1 hora	Alta
RNF11	Backup de dados	Diário automatizado, retenção 30 dias	Alta
RNF12	Recuperação de desastres (RPO)	< 1 hora de perda de dados	Média
RNF13	Recuperação de desastres (RTO)	< 4 horas para restauração completa	Média
RNF14	Taxa de erro de API	< 0,1%	Alta

6.3 Requisitos de Escalabilidade

ID	Requisito	Descrição	Prioridade
RNF15	Escalabilidade horizontal	Auto Scaling baseado em CPU (70%) e memória (80%)	Alta
RNF16	Crescimento de usuários	Suportar crescimento de 50% em 6 meses sem degradação	Média
RNF17	Armazenamento de dados	Capacidade para 1 milhão de produtos	Média
RNF18	Cache distribuído	Redis com replicação para reduzir carga no BD	Alta
RNF19	CDN para assets estáticos	CloudFront para imagens e assets	Média

6.4 Requisitos de Segurança

ID	Requisito	Descrição	Prioridade
RNF20	Criptografia de dados em trânsito	TLS 1.2+ obrigatório	Alta
RNF21	Criptografia de dados em repouso	AES-256 para dados sensíveis	Alta
RNF22	Autenticação	OAuth2/JWT com expiração 24h	Alta
RNF23	Autorização	RBAC (Role-Based Access Control)	Alta
RNF24	Senha	Bcrypt com salt, mínimo 8 caracteres	Alta
RNF25	Proteção contra injeção SQL	Prepared statements, ORM SQLAlchemy	Alta
RNF26	Proteção contra XSS	Sanitização de inputs	Alta
RNF27	Rate limiting	100 requisições/minuto por IP	Média
RNF28	Auditoria	Logs imutáveis de ações críticas	Alta
RNF29	Conformidade PCI-DSS	Tokenização de cartões, sem armazenamento	Alta
RNF30	Conformidade LGPD	Consentimento, portabilidade, direito ao esquecimento	Alta

6.5 Requisitos de Usabilidade

ID	Requisito	Descrição	Prioridade
RNF31	Facilidade de aprendizado	Novo usuário conclui primeiro pedido em < 5 minutos	Alta
RNF32	Taxa de erro do usuário	< 5% em fluxos principais	Média

RNF33	Satisfação do usuário (NPS)	NPS > 70	Média
RNF34	Acessibilidade	WCAG 2.1 Nível AA	Alta
RNF35	Internacionalização	Suporte a PT-BR (futuro: EN, ES)	Baixa
RNF36	Modo escuro	Disponível em todo o app	Média
RNF37	Feedback visual	Resposta imediata a todas as ações	Alta

6.6 Requisitos de Manutenibilidade

ID	Requisito	Descrição	Prioridade
RNF38	Cobertura de testes	Mínimo 85% de cobertura de código	Alta
RNF39	Documentação de código	Docstrings em todas as funções públicas	Média
RNF40	Complexidade ciclomática	Máximo 10 por função	Média
RNF41	Code review	Obrigatório via Pull Request	Alta
RNF42	Versionamento semântico	SemVer 2.0.0	Média
RNF43	Logs estruturados	JSON format com níveis (DEBUG, INFO, WARN, ERROR)	Alta

6.7 Requisitos de Portabilidade

ID	Requisito	Descrição	Prioridade
RNF44	Multiplataforma mobile	Android e iOS com código compartilhado (Ionic/RN)	Alta
RNF45	Independência de cloud	Infraestrutura via IaC (Terraform) para migração	Média
RNF46	Containerização	Docker para todos os serviços	Alta
RNF47	Orquestração	Kubernetes para deploy	Alta

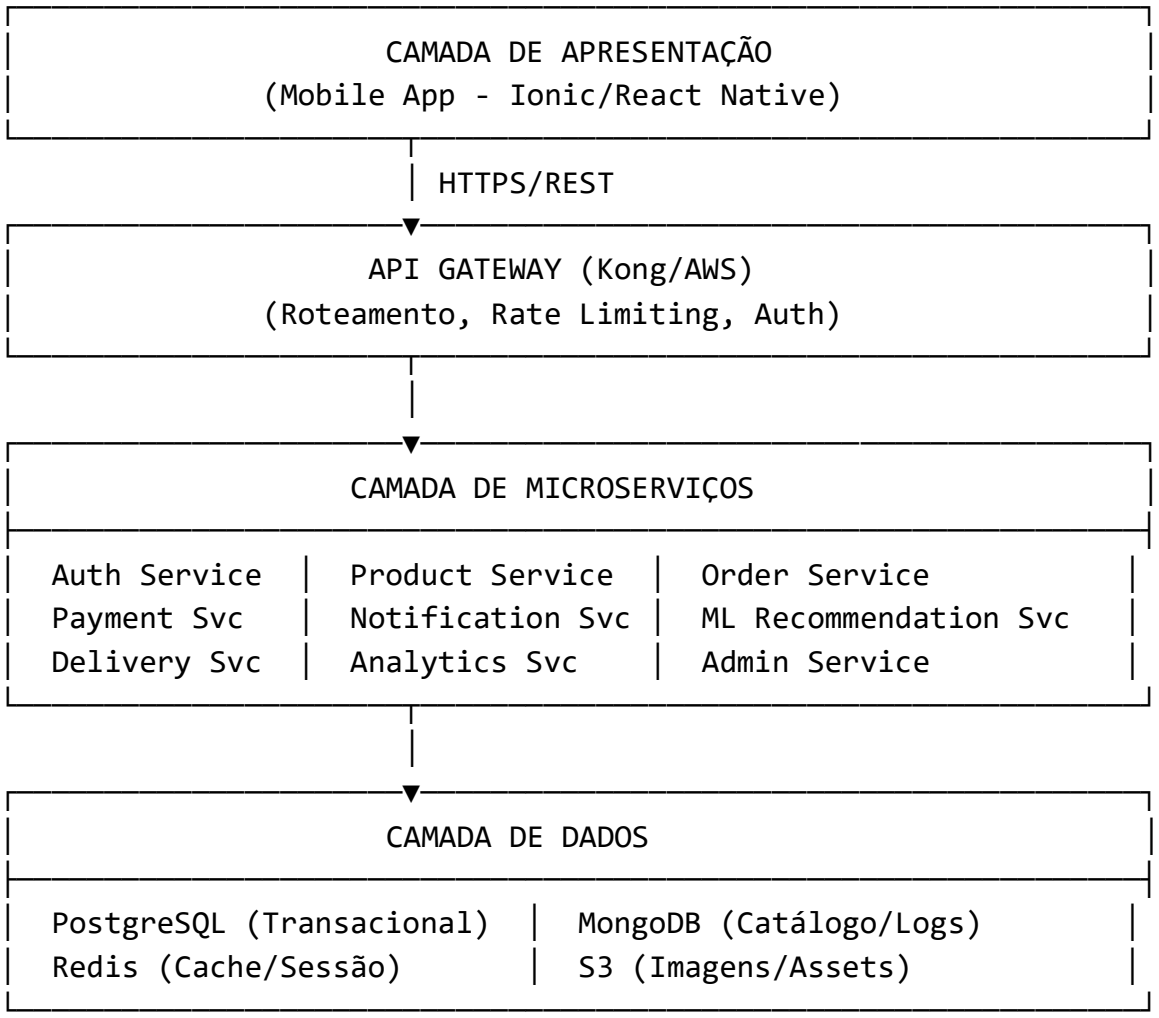
6.8 Requisitos de Qualidade de Código

ID	Requisito	Descrição	Prioridade
RNF48	Linting	Pylint/Flake8 com score mínimo 8.5	Média
RNF49	Formatação	Black e isort para Python	Média
RNF50	Type hints	Mypy para verificação de tipos	Média
RNF51	Testes automatizados	Pytest com TDD	Alta
RNF52	CI/CD	Pipeline automatizado (GitHub Actions)	Alta

7. ARQUITETURA E DESIGN PATTERNS

7.1 Arquitetura Geral

Padrão Arquitetural: Microserviços + MVC (Backend) + MVVM (Frontend Mobile)



7.2 Padrões de Criação (Creational Patterns)

7.2.1 Factory Method

Uso: Criação de objetos de recomendação baseados no perfil do usuário

```
from abc import ABC, abstractmethod

class RecommendationFactory(ABC):
```

```
    @abstractmethod
    def create_recommender(self):
        pass

class CollaborativeFilteringFactory(RecommendationFactory):
    def create_recommender(self):
        return CollaborativeFilteringRecommender()

class ContentBasedFactory(RecommendationFactory):
    def create_recommender(self):
        return ContentBasedRecommender()

# Uso
factory = CollaborativeFilteringFactory()
recommender = factory.create_recommender()
recommendations = recommender.get_recommendations(user_id)
```

7.2.2 Singleton

Uso: Conexão com banco de dados, configurações globais

```
class DatabaseConnection:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls._instance.connection = cls._create_connection()
        return cls._instance

    @staticmethod
    def _create_connection():
        return create_engine(DATABASE_URL)
```

7.3 Padrões Estruturais (Structural Patterns)

7.3.1 Adapter

Uso: Integração com Mercado Pago API

```
class PaymentGatewayAdapter(ABC):
    @abstractmethod
    def process_payment(self, amount, method):
        pass

class MercadoPagoAdapter(PaymentGatewayAdapter):
    def __init__(self):
        self.sdk = mercadopago.SDK(ACCESS_TOKEN)

    def process_payment(self, amount, method):
        # Adapta para formato Mercado Pago
        preference = self._create_preference(amount, method)
        return self.sdk.preference().create(preference)
```

7.3.2 Facade

Uso: Simplificar operações complexas de pedido

```
class OrderFacade:
    def __init__(self):
        self.cart_service = CartService()
        self.payment_service = PaymentService()
        self.notification_service = NotificationService()
        self.inventory_service = InventoryService()

    def place_order(self, user_id, payment_info):
        # Orquestra múltiplos serviços
        cart = self.cart_service.get_cart(user_id)
        order = self.cart_service.convert_to_order(cart)

        payment_result = self.payment_service.process(payment_info)
        if payment_result.success:
            self.inventory_service.reserve_items(order.items)
            self.notification_service.send_confirmation(user_id,
order)
            return order
        else:
            raise PaymentFailedException()
```

7.4 Padrões Comportamentais (Behavioral Patterns)

7.4.1 Observer

Uso: Sistema de notificações para mudanças de status de pedido

```
from typing import List

class OrderObserver(ABC):
    @abstractmethod
    def update(self, order):
        pass

class CustomerNotification(OrderObserver):
    def update(self, order):
        send_push_notification(order.customer_id, f"Pedido
{order.id} atualizado para {order.status}")

class Order:
    def __init__(self):
        self._observers: List[OrderObserver] = []
        self._status = None

    def attach(self, observer: OrderObserver):
        self._observers.append(observer)

    def set_status(self, status):
        self._status = status
        self._notify()

    def _notify(self):
        for observer in self._observers:
            observer.update(self)
```

7.4.2 Strategy

Uso: Cálculo de preço com diferentes estratégias de desconto

```
class PricingStrategy(ABC):
    @abstractmethod
    def calculate_price(self, base_price, quantity):
        pass
```

```
class RegularPricing(PricingStrategy):
    def calculate_price(self, base_price, quantity):
        return base_price * quantity

class BulkDiscountPricing(PricingStrategy):
    def calculate_price(self, base_price, quantity):
        if quantity >= 10:
            return base_price * quantity * 0.9  # 10% desconto
        return base_price * quantity

class Order:
    def __init__(self, pricing_strategy: PricingStrategy):
        self.pricing_strategy = pricing_strategy

    def calculate_total(self, items):
        total = 0
        for item in items:
            total +=
self.pricing_strategy.calculate_price(item.price, item.quantity)
        return total
```

7.5 Padrões de Arquitetura

7.5.1 MVC (Model-View-Controller) - Backend

```
# Model (models/product.py)
class Product(Base):
    __tablename__ = "products"
    id = Column(UUID, primary_key=True)
    name = Column(String)
    price = Column(Numeric)

# View (schemas/product.py) - DTO
class ProductResponse(BaseModel):
    id: UUID
    name: str
    price: Decimal

# Controller (controllers/product.py)
@router.get("/products", response_model=List[ProductResponse])
```

```
async def list_products(db: Session = Depends(get_db)):
    products = db.query(Product).filter(Product.active ==
True).all()
    return products
```

7.5.2 Repository Pattern

```
class ProductRepository:
    def __init__(self, db: Session):
        self.db = db

    def get_by_id(self, product_id: UUID) -> Product:
        return self.db.query(Product).filter(Product.id ==
product_id).first()

    def get_all_active(self) -> List[Product]:
        return self.db.query(Product).filter(Product.active ==
True).all()

    def create(self, product: Product) -> Product:
        self.db.add(product)
        self.db.commit()
        self.db.refresh(product)
        return product
```

7.5.3 DAO (Data Access Object)

```
class UserDAO:
    def __init__(self, db: Session):
        self.db = db

    def find_by_email(self, email: str) -> Optional[User]:
        return self.db.query(User).filter(User.email ==
email).first()

    def save(self, user: User):
        self.db.add(user)
        self.db.commit()
        return user
```

```
    def update(self, user: User):
        self.db.commit()
        self.db.refresh(user)
        return user
```

7.5.4 DTO (Data Transfer Object)

```
class CreateOrderDTO(BaseModel):
    items: List[OrderItemDTO]
    address_id: UUID
    payment_method: PaymentMethod
    observations: Optional[str]

class OrderResponseDTO(BaseModel):
    id: UUID
    total: Decimal
    status: OrderStatus
    created_at: datetime

    class Config:
        orm_mode = True
```

7.6 Microserviços

Serviços Identificados:

1. **Auth Service:**
    a. Autenticação e autorização
    b. Gerenciamento de tokens JWT
    c. Controle de sessões
2. **Product Service:**
    a. CRUD de produtos
    b. Gerenciamento de catálogo
    c. Controle de estoque
3. **Order Service:**
    a. Criação de pedidos
    b. Atualização de status
    c. Histórico de pedidos
4. **Payment Service:**
    a. Processamento de pagamentos
    b. Integração Mercado Pago

c. Gestão de transações

5. **Notification Service:**
   a. Push notifications (Firebase)
   b. E-mail notifications
   c. SMS (futuro)
6. **ML Recommendation Service:**
   a. Recomendações personalizadas
   b. Treinamento de modelos
   c. Feature engineering
7. **Delivery Service:**
   a. Atribuição de entregadores
   b. Rastreamento GPS
   c. Gestão de entregas
8. **Analytics Service:**
   a. KPIs e métricas
   b. Relatórios gerenciais
   c. Data warehouse

8. SEGURANÇA E COMPLIANCE

8.1 Autenticação e Autorização

8.1.1 OAuth2/JWT

Implementação:

```
from jose import JWTError, jwt
from datetime import datetime, timedelta

SECRET_KEY = os.getenv("JWT_SECRET_KEY")
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 1440  # 24 horas

def create_access_token(data: dict):
    to_encode = data.copy()
    expire = datetime.utcnow() +
timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY,
algorithm=ALGORITHM)
    return encoded_jwt
```

```
def verify_token(token: str):
    try:
        payload = jwt.decode(token, SECRET_KEY,
algorithms=[ALGORITHM])
        return payload
    except JWTError:
        raise HTTPException(status_code=401, detail="Token
inválido")
```

Regras:

- Token expira em 24 horas
- Refresh token válido por 7 dias
- Algoritmo HS256 (mínimo RS256 para produção)
- Armazenamento seguro em Secrets Manager

8.1.2 RBAC (Role-Based Access Control)

Perfis:

```
class UserRole(str, Enum):
    CLIENTE = "cliente"
    # ESPECIFICAÇÃO DE REQUISITOS DE SOFTWARE (SRS)
## APLICATIVO MOBILE DE CUPCAKES GOURMET

---

**Versão:** 1.0
**Data:** 17/11/2025
**Autor:** Equipe de Engenharia de Software
**Status:** Aprovado

**Conformidade:**
- IEEE 830-1998 - Recommended Practice for Software Requirements
Specifications
- IEEE 29148-2018 - Systems and Software Engineering
- ABNT NBR ISO/IEC/IEEE 29148:2013 - Engenharia de requisitos
- ABNT NBR ISO/IEC 12207:2017 - Processos do ciclo de vida de
software
- ISO/IEC 25010:2011 - Software product quality model

---
```

ÍNDICE

1. [INTRODUÇÃO](#1-introdução)
2. [DESCRIÇÃO GERAL](#2-descrição-geral)
3. [REQUISITOS ESPECÍFICOS](#3-requisitos-específicos)
4. [MODELAGEM DO SISTEMA](#4-modelagem-do-sistema)
5. [REQUISITOS DE INTERFACE](#5-requisitos-de-interface)
6. [REQUISITOS NÃO-FUNCIONAIS](#6-requisitos-não-funcionais)
7. [ARQUITETURA E DESIGN PATTERNS](#7-arquitetura-e-design-patterns)
8. [SEGURANÇA E COMPLIANCE](#8-segurança-e-compliance)
9. [GESTÃO DO PROJETO](#9-gestão-do-projeto)
10. [ANEXOS](#10-anexos)

---

1. INTRODUÇÃO

1.1 Propósito do Documento

Este documento especifica todos os requisitos funcionais e não funcionais do **Aplicativo Mobile de Cupcakes Gourmet**, com o objetivo de orientar o desenvolvimento, validação e manutenção do sistema. Destina-se a desenvolvedores, testadores, gerentes de projeto, stakeholders e equipe de manutenção.

1.2 Escopo do Produto

O **Aplicativo Mobile de Cupcakes Gourmet** é uma plataforma integrada de e-commerce que visa revolucionar a experiência de compra de cupcakes gourmet, oferecendo:

Para Clientes:
- Catálogo digital com ≥40 sabores (doces e salgados)
- Pedidos personalizados com rastreamento em tempo real
- Pagamentos seguros via Pix e cartão de crédito
- Recomendações inteligentes baseadas em ML/LLMs
- Notificações push sobre status de pedidos
- Avaliação e feedback de produtos

Para Administradores:
- Gerenciamento completo de catálogo e estoque

- Controle de ingredientes e custos unitários
- Dashboards analíticos com KPIs de vendas
- Gestão de usuários e entregadores
- Relatórios gerenciais e financeiros
- Monitoramento de segurança e logs

Para Entregadores:
- Visualização de pedidos atribuídos
- Atualização de status em tempo real
- Navegação integrada
- Histórico de entregas

1.3 Definição do Problema

A loja de cupcakes gourmet enfrenta desafios operacionais críticos:

- **Gestão Manual Ineficiente:** Pedidos via telefone/WhatsApp causam erros e atrasos
- **Falta de Rastreabilidade:** Clientes sem visibilidade do status do pedido
- **Estoque Desatualizado:** Dificuldade em controlar ingredientes e custos
- **Ausência de Análise:** Sem dados para decisões estratégicas
- **Experiência Limitada:** Cliente sem personalização ou recomendações

Solução Proposta:
Plataforma mobile integrada que une compras, pagamentos, notificações, gestão operacional e recomendações personalizadas via ML/LLMs.

1.4 Público-Alvo

Perfil	Descrição	Características
Clientes Finais	Consumidores de cupcakes gourmet	Usuários de smartphones (Android/iOS), idade 18-55 anos, conectados digitalmente
Administradores	Gestores da loja	Conhecimento básico de gestão, familiaridade com sistemas web
Entregadores	Responsáveis pela logística	Motoristas com smartphones, familiaridade com apps de navegação

| **Equipe Técnica** | DevOps e desenvolvedores | Conhecimento em Python, Cloud, CI/CD, ML |

1.5 Definições, Acrônimos e Abreviações

Termo	Definição
API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration/Continuous Delivery
DTO	Data Transfer Object
IaC	Infrastructure as Code
JWT	JSON Web Token
KPI	Key Performance Indicator
LLM	Large Language Model
ML	Machine Learning
MVP	Minimum Viable Product
OAuth2	Open Authorization 2.0
ORM	Object-Relational Mapping
RBAC	Role-Based Access Control
REST	Representational State Transfer
SRS	Software Requirements Specification
TDD	Test-Driven Development
UML	Unified Modeling Language
WCAG	Web Content Accessibility Guidelines

1.6 Referências

- IEEE 830-1998: Recommended Practice for Software Requirements Specifications
- IEEE 29148-2018: Systems and Software Engineering
- ABNT NBR ISO/IEC/IEEE 29148:2013: Engenharia de requisitos
- ISO/IEC 25010:2011: Software product quality model
- LGPD (Lei Geral de Proteção de Dados) - Lei nº 13.709/2018
- PCI-DSS: Payment Card Industry Data Security Standard
- Documento de Modelagem UML - Versão 1.0
- Documento de Casos de Uso - Versão 1.0

1.7 Visão Geral do Documento

Este SRS está organizado conforme as melhores práticas IEEE e ABNT:

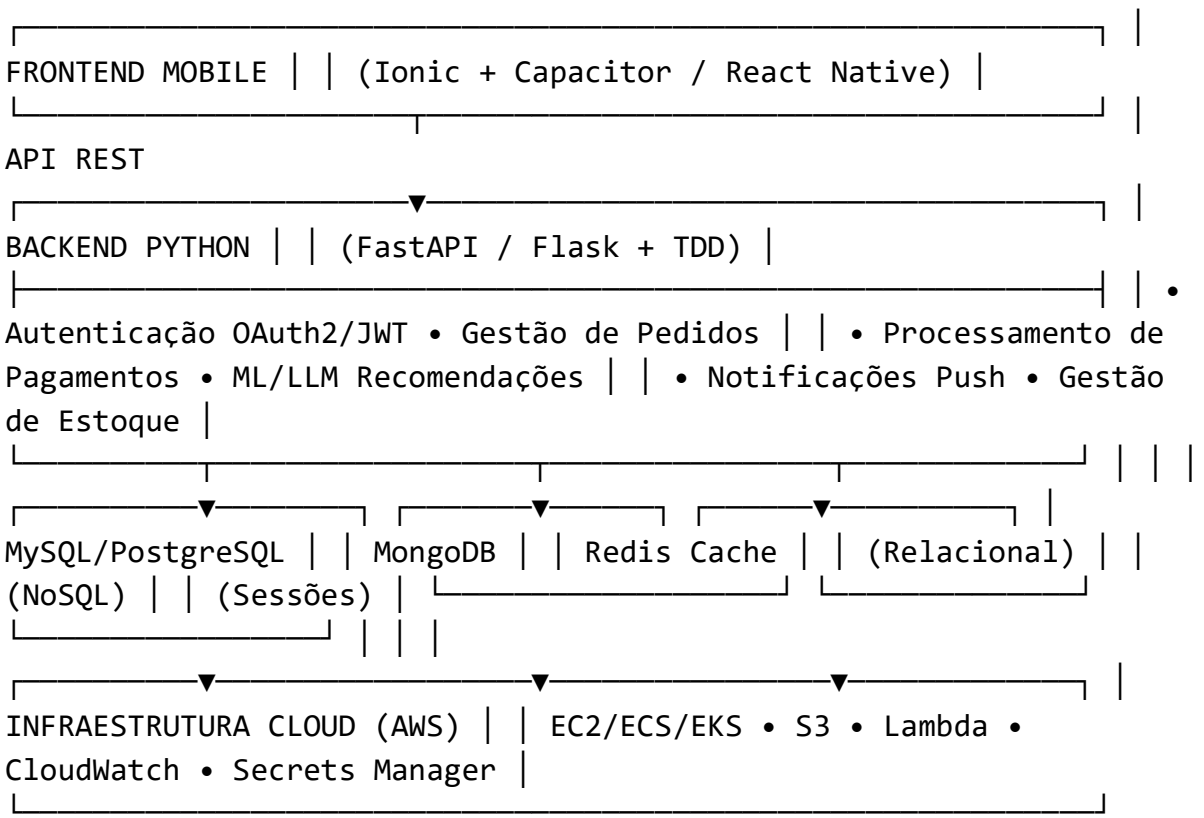
- **Seção 1:** Contextualização e propósito
- **Seção 2:** Descrição geral do sistema e perspectivas
- **Seção 3:** Requisitos funcionais detalhados
- **Seção 4:** Modelagem UML e diagramas
- **Seção 5:** Especificação de interfaces
- **Seção 6:** Requisitos não-funcionais (performance, segurança, usabilidade)
- **Seção 7:** Arquitetura e padrões de design
- **Seção 8:** Segurança, compliance e LGPD
- **Seção 9:** Metodologia, cronograma e gestão
- **Seção 10:** Anexos técnicos

---

2. DESCRIÇÃO GERAL

2.1 Perspectiva do Produto

O Aplicativo Mobile de Cupcakes Gourmet é um sistema independente que integra múltiplos componentes:



Integrações Externas:
- **Mercado Pago API:** Gateway de pagamento (Pix, cartões)
- **Firebase Cloud Messaging:** Notificações push
- **Sistema de Entregas:** Rastreamento logístico (API própria)
- **MLFlow/Kubeflow:** Pipelines de Machine Learning
- **Prometheus + Grafana:** Monitoramento e métricas

2.2 Funções Principais do Sistema

2.2.1 Módulo de Autenticação e Cadastro
- Registro de novos usuários com validação de e-mail
- Login seguro com OAuth2/JWT
- Recuperação de senha
- Gerenciamento de perfil

2.2.2 Módulo de Catálogo
- Exibição de ≥40 sabores de cupcakes (doces e salgados)
- Filtros por categoria, preço, popularidade
- Busca inteligente de produtos
- Detalhamento de ingredientes e informações nutricionais

2.2.3 Módulo de Pedidos
- Carrinho de compras com atualização dinâmica
- Criação de pedidos personalizados
- Rastreamento em tempo real
- Histórico de pedidos

2.2.4 Módulo de Pagamentos
- Integração com Mercado Pago
- Suporte a Pix e cartões de crédito/débito
- Processamento seguro (PCI-DSS)
- Confirmação automática e geração de recibos

2.2.5 Módulo de Notificações
- Push notifications via Firebase
- Alertas de status de pedido
- Promoções e ofertas personalizadas
- Lembretes de pagamento

2.2.6 Módulo Administrativo
- Dashboard com KPIs (vendas, engajamento, estoque)

- CRUD de produtos, estoque e preços
- Gestão de usuários e permissões
- Relatórios financeiros e operacionais
- Logs de auditoria

2.2.7 Módulo de Entregas
- Atribuição de pedidos a entregadores
- Atualização de status (Em Preparo → Saiu → A Caminho → Entregue)
- Geolocalização em tempo real
- Histórico de entregas

2.2.8 Módulo de ML/LLMs
- Recomendações personalizadas baseadas em histórico
- Sistema de sugestões inteligentes
- Análise preditiva de demanda
- Pipelines de retraining automatizado
- Monitoramento de concept/data drift

2.3 Características dos Usuários

Cliente Final
- **Expertise Técnica:** Baixa a média (familiaridade com apps mobile)
- **Idade:** 18-55 anos
- **Nível Educacional:** Variado
- **Expectativas:** Interface intuitiva, pagamento rápido, rastreamento claro

Administrador
- **Expertise Técnica:** Média (conhecimento de gestão)
- **Idade:** 25-60 anos
- **Nível Educacional:** Ensino superior desejável
- **Expectativas:** Dashboards claros, relatórios confiáveis, controle total

Entregador
- **Expertise Técnica:** Baixa (uso básico de smartphone)
- **Idade:** 18-50 anos
- **Nível Educacional:** Ensino fundamental/médio
- **Expectativas:** App simples, navegação integrada, atualizações fáceis

2.4 Restrições

2.4.1 Restrições Tecnológicas
- Aplicação deve funcionar em **Android ≥ 10** e **iOS ≥ 14**
- Backend obrigatoriamente em **Python 3.12+**
- Uso de **Docker** e **Kubernetes** para containerização
- Deploy em **AWS** como cloud principal (GCP/Azure como fallback)
- Bancos de dados: **MySQL/PostgreSQL** (relacional) e **MongoDB** (NoSQL)

2.4.2 Restrições de Negócio
- Catálogo mínimo de **40 sabores** (incluindo versões salgadas)
- Suporte obrigatório a **Pix** e **cartões de crédito/débito**
- Tempo máximo de resposta: **200ms** para 95% das requisições
- Disponibilidade mínima: **99,5%**

2.4.3 Restrições Regulatórias
- Conformidade com **LGPD** (Lei Geral de Proteção de Dados)
- Compliance **PCI-DSS** para processamento de pagamentos
- Acessibilidade conforme **WCAG 2.1 Nível AA**

2.4.4 Restrições de Recursos
- Equipe de desenvolvimento: 5-8 pessoas
- Prazo de entrega: 6 semanas (MVP)
- Orçamento limitado para infraestrutura inicial

2.5 Suposições e Dependências

2.5.1 Suposições
- Usuários possuem conectividade à internet (3G mínimo)
- Dispositivos com GPS habilitado para rastreamento
- Clientes possuem conta bancária ou cartão de crédito
- Administradores têm conhecimento básico de gestão

2.5.2 Dependências
- **Mercado Pago API:** Disponibilidade do gateway de pagamento
- **Firebase:** Serviço de notificações push operacional
- **AWS:** Infraestrutura cloud disponível e escalável
- **APIs de Mapas:** Google Maps ou OpenStreetMap para geolocalização
- **MLFlow/Kubeflow:** Plataformas de ML operacionais

2.6 Requisitos de Compatibilidade

2.6.1 Dispositivos Móveis
- **Android:** Versão 10.0 ou superior
- **iOS:** Versão 14.0 ou superior
- **Resoluções:** Suporte de 320x568px até 1440x3040px
- **Orientações:** Portrait (vertical) como padrão

2.6.2 Navegadores Web (Admin Dashboard)
- Google Chrome 90+
- Mozilla Firefox 88+
- Safari 14+
- Microsoft Edge 90+

---

3. REQUISITOS ESPECÍFICOS

3.1 Requisitos Funcionais

RF01 - Cadastro de Usuário
Prioridade: Alta
Caso de Uso: UC01

Descrição: O sistema deve permitir que novos usuários criem uma conta no aplicativo.

Entradas:
- Nome completo (obrigatório)
- E-mail válido (obrigatório, único)
- Telefone (obrigatório, formato brasileiro)
- Senha (obrigatório, mínimo 8 caracteres)
- CPF (opcional no cadastro inicial)

Processamento:
1. Validar formato de e-mail (regex)
2. Verificar unicidade do e-mail no banco
3. Validar força da senha (mínimo 8 caracteres, 1 maiúscula, 1 número)
4. Criptografar senha com bcrypt
5. Gerar token de verificação de e-mail
6. Criar registro de usuário com status "Pendente"

7. Enviar e-mail de confirmação

Saídas:
- Conta criada com status "Pendente"
- E-mail de verificação enviado
- Mensagem de sucesso exibida

Regras de Negócio:
- RN01: E-mail deve ser único no sistema
- RN02: Senha deve ter mínimo 8 caracteres, 1 letra maiúscula, 1 número
- RN03: Conta só é ativada após verificação de e-mail
- RN04: CPF/telefone podem ser adicionados posteriormente

RF02 - Autenticação de Usuário
Prioridade: Alta
Caso de Uso: UC02

Descrição: O sistema deve permitir que usuários registrados acessem suas contas.

Entradas:
- E-mail ou nome de usuário
- Senha

Processamento:
1. Validar existência do usuário
2. Verificar hash da senha com bcrypt
3. Verificar se conta está ativa
4. Gerar token JWT com expiração de 24h
5. Registrar log de acesso
6. Criar sessão do usuário

Saídas:
- Token JWT válido
- Dados básicos do usuário (nome, tipo, permissões)
- Redirecionamento para tela principal

Fluxos Alternativos:
- **FA01:** Credenciais incorretas → Exibir erro, incrementar

contador de tentativas
- **FA02:** Conta bloqueada (5 tentativas) → Exibir mensagem, solicitar recuperação
- **FA03:** E-mail não verificado → Exibir aviso, oferecer reenvio de e-mail

Regras de Negócio:
- RN05: Após 5 tentativas incorretas, conta bloqueada por 30 minutos
- RN06: Sessão JWT expira após 24 horas de inatividade
- RN07: Apenas usuários com e-mail verificado podem fazer login

---

RF03 - Visualização de Catálogo
Prioridade: Alta
Caso de Uso: UC03

Descrição: O sistema deve exibir o catálogo completo de cupcakes disponíveis.

Entradas:
- Filtros de categoria (opcional)
- Termo de busca (opcional)
- Ordenação (preço, popularidade, nome)

Processamento:
1. Consultar produtos ativos no banco de dados
2. Aplicar filtros selecionados
3. Ordenar conforme critério escolhido
4. Paginar resultados (20 itens por página)
5. Carregar imagens otimizadas (lazy loading)

Saídas:
- Lista de produtos com:
  - Imagem (thumbnail 300x300px)
  - Nome do produto
  - Descrição curta
  - Preço
  - Avaliação média (estrelas)
  - Indicação de disponibilidade

Regras de Negócio:

- RN08: Apenas produtos com status "Ativo" são exibidos
- RN09: Produtos sem estoque mostram "Indisponível" (não podem ser adicionados ao carrinho)
- RN10: Imagens carregadas progressivamente (lazy loading)
- RN11: Catálogo deve ter mínimo 40 sabores ativos

RF04 - Gerenciamento de Carrinho
Prioridade: Alta
Caso de Uso: UC04 (parcial)

Descrição: O sistema deve permitir adicionar, remover e modificar itens no carrinho.

Entradas:
- ID do produto
- Quantidade desejada
- Observações especiais (opcional)

Processamento:
1. Validar disponibilidade em estoque
2. Verificar quantidade máxima permitida (10 unidades/item)
3. Calcular subtotal do item
4. Adicionar ao carrinho da sessão
5. Recalcular total geral
6. Aplicar cupons de desconto (se houver)

Saídas:
- Carrinho atualizado com:
  - Lista de itens
  - Quantidade de cada item
  - Subtotais
  - Total geral
  - Descontos aplicados

Regras de Negócio:
- RN12: Validação de estoque em tempo real antes de adicionar
- RN13: Quantidade máxima de 10 unidades por item
- RN14: Carrinho expira após 30 minutos de inatividade
- RN15: Itens reservados temporariamente ao adicionar ao carrinho

---

RF05 - Efetuar Pedido
Prioridade: Alta
Caso de Uso: UC04

Descrição: O sistema deve permitir que o cliente finalize a compra dos itens no carrinho.

Entradas:
- Endereço de entrega
- Método de pagamento
- Observações do pedido (opcional)

Processamento:
1. Validar endereço de entrega
2. Verificar disponibilidade final de estoque
3. Calcular taxa de entrega baseada em distância
4. Criar registro de pedido com status "Aguardando Pagamento"
5. Reservar itens no estoque
6. Acionar módulo de pagamento (RF06)

Saídas:
- Pedido criado com número único
- Resumo do pedido exibido
- Redirecionamento para pagamento

Regras de Negócio:
- RN16: Valor mínimo de pedido: R$ 20,00
- RN17: Taxa de entrega calculada por geolocalização
- RN18: Itens reservados por 15 minutos para pagamento
- RN19: Pedido só é confirmado após pagamento aprovado

RF06 - Processar Pagamento
Prioridade: Alta (Crítico)
Caso de Uso: UC05

Descrição: O sistema deve processar pagamentos via Mercado Pago (Pix e cartões).

Entradas:
- Dados de pagamento (cartão ou Pix)
- Valor total do pedido
- ID do pedido

Processamento:
1. Validar dados de pagamento localmente
2. Criptografar informações sensíveis
3. Enviar requisição à API do Mercado Pago
4. Aguardar resposta (timeout 30s)
5. Processar retorno (aprovado/recusado)
6. Atualizar status do pedido
7. Registrar transação

Saídas:
- **Sucesso:**
  - Pedido com status "Pago"
  - Comprovante de pagamento
  - Notificação de confirmação
- **Falha:**
  - Mensagem de erro detalhada
  - Opção de tentar novamente
  - Liberação de itens reservados

Regras de Negócio:
- RN20: Dados de cartão nunca armazenados (tokenização via Mercado Pago)
- RN21: Validação básica: formato de cartão, validade, CVV
- RN22: Pix: QR Code válido por 10 minutos
- RN23: Máximo 3 tentativas de pagamento por pedido

RF07 - Acompanhar Pedido
Prioridade: Média
Caso de Uso: UC06

Descrição: O sistema deve permitir que o cliente visualize o status do pedido em tempo real.

Entradas:
- ID do pedido ou lista de pedidos do usuário

Processamento:
1. Consultar status atual do pedido
2. Buscar histórico de atualizações (timeline)
3. Obter localização do entregador (se disponível)
4. Calcular previsão de entrega

Saídas:
- Timeline com estados:
  - Aguardando Pagamento
  - Pago
  - Em Preparo
  - Saiu para Entrega
  - A Caminho
  - Entregue
- Horário de cada atualização
- Mapa com localização (se em trânsito)
- Previsão de chegada

Regras de Negócio:
- RN24: Atualização automática a cada 30 segundos
- RN25: Notificação push em cada mudança de status
- RN26: Geolocalização apenas quando status "A Caminho"

RF08 - Notificações Push
Prioridade: Média
Caso de Uso: UC07

Descrição: O sistema deve enviar notificações push para eventos relevantes.

Entradas:
- Evento disparador (mudança de status, promoção, lembrete)
- ID do usuário destinatário
- Conteúdo da mensagem

Processamento:
1. Verificar se usuário tem notificações habilitadas
2. Consultar token FCM do dispositivo
3. Montar payload da notificação

4. Enviar via Firebase Cloud Messaging
5. Registrar envio em log

Saídas:
- Notificação entregue ao dispositivo
- Badge no ícone do app (iOS)
- Som/vibração conforme configuração

Eventos que Disparam Notificações:
- Pedido confirmado (pagamento aprovado)
- Status de entrega alterado
- Pedido saiu para entrega
- Pedido entregue
- Promoções personalizadas
- Cupons de desconto disponíveis

Regras de Negócio:
- RN27: Usuário pode desabilitar notificações nas configurações
- RN28: Notificações de marketing apenas com consentimento explícito
- RN29: Máximo 5 notificações por dia (exceto status de pedido)

RF09 - Gerenciar Catálogo (Admin)
Prioridade: Alta
Caso de Uso: UC09

Descrição: Administradores devem poder criar, editar, ativar/desativar e remover produtos.

Entradas:
- Dados do produto:
  - Nome (obrigatório)
  - Descrição (obrigatório)
  - Categoria (doce/salgado)
  - Preço (obrigatório)
  - Ingredientes
  - Informações nutricionais
  - Imagens (mínimo 1, máximo 5)
  - Status (Ativo/Inativo)

Processamento:

1. Validar permissões do usuário (RBAC)
2. Validar unicidade do nome do produto
3. Validar formato e tamanho das imagens (max 2MB cada)
4. Fazer upload de imagens para S3
5. Criar/atualizar registro no banco
6. Atualizar cache (Redis)
7. Registrar ação em log de auditoria

Saídas:
- Produto criado/atualizado
- Mensagem de confirmação
- Cache atualizado

Regras de Negócio:
- RN30: Apenas administradores podem gerenciar catálogo
- RN31: Produtos inativos não aparecem para clientes
- RN32: Exclusão física só permitida se produto nunca foi vendido
- RN33: Imagens redimensionadas automaticamente (300x300px, 800x800px)

RF10 - Gerenciar Estoque e Custos (Admin)
Prioridade: Alta
Caso de Uso: UC10

Descrição: Administradores devem controlar quantidade em estoque e custos unitários.

Entradas:
- ID do produto
- Quantidade em estoque
- Custo unitário
- Margem de lucro desejada

Processamento:
1. Validar permissões
2. Atualizar quantidade em estoque
3. Registrar custo unitário
4. Calcular preço de venda sugerido
5. Registrar histórico de alterações
6. Criar alerta se estoque baixo (< 10 unidades)

Saídas:
- Estoque atualizado
- Relatório de variação de custos
- Alertas de estoque baixo

Regras de Negócio:
- RN34: Não permitir estoque negativo
- RN35: Histórico de custos mantido por 12 meses
- RN36: Alerta automático quando estoque < 10 unidades
- RN37: Sugestão de preço: custo unitário × (1 + margem)

RF11 - Gerar Relatórios e Dashboards (Admin)
Prioridade: Alta
Caso de Uso: UC11

Descrição: Sistema deve fornecer relatórios gerenciais e KPIs visuais.

Entradas:
- Período desejado (data inicial e final)
- Tipo de relatório (vendas, estoque, financeiro, clientes)
- Filtros adicionais (categoria, produto, entregador)

Processamento:
1. Consultar dados históricos
2. Aplicar filtros e agregações
3. Calcular KPIs:
   - Total de vendas
   - Ticket médio
   - Produtos mais vendidos
   - Taxa de conversão
   - Churn rate
4. Gerar gráficos (Recharts/Chart.js)
5. Exportar (PDF/Excel opcional)

Saídas:
- Dashboard interativo com:
  - Gráficos de tendência de vendas
  - Top 10 produtos

  - Receita por período
  - Análise de estoque
  - Performance de entregadores
- Opção de exportação

Regras de Negócio:
- RN38: Apenas administradores têm acesso
- RN39: Dados atualizados em tempo real
- RN40: Histórico mantido por 24 meses
- RN41: Exportação limitada a 10.000 registros por vez

RF12 - Atribuir Pedido a Entregador (Admin)
Prioridade: Alta
Caso de Uso: UC08

Descrição: Administrador atribui pedidos pagos a entregadores disponíveis.

Entradas:
- ID do pedido
- ID do entregador

Processamento:
1. Verificar se pedido está pago
2. Verificar disponibilidade do entregador
3. Criar atribuição no sistema
4. Notificar entregador via push
5. Atualizar status do pedido para "Atribuído"
6. Enviar dados ao Sistema de Entregas (UC20)
7. Registrar log de atribuição

Saídas:
- Pedido atribuído
- Entregador notificado
- Status atualizado

Regras de Negócio:
- RN42: Apenas pedidos pagos podem ser atribuídos
- RN43: Entregador pode ter máximo 3 pedidos simultâneos
- RN44: Priorizar entregadores com menos entregas no dia

RF13 - Visualizar Pedidos Atribuídos (Entregador)
Prioridade: Média
Caso de Uso: UC14

Descrição: Entregador visualiza pedidos que lhe foram atribuídos.

Entradas:
- ID do entregador (sessão autenticada)

Processamento:
1. Consultar pedidos atribuídos ao entregador
2. Filtrar por status (Pendente, Em Andamento, Concluído)
3. Ordenar por horário de atribuição
4. Calcular distância até endereço de entrega

Saídas:
- Lista de pedidos com:
  - Número do pedido
  - Endereço de entrega
  - Itens do pedido
  - Valor total
  - Status atual
  - Distância estimada

Regras de Negócio:
- RN45: Apenas pedidos do entregador logado são exibidos
- RN46: Pedidos concluídos aparecem no histórico (últimos 30 dias)

RF14 - Aceitar/Recusar Entrega (Entregador)
Prioridade: Média
Caso de Uso: UC15

Descrição: Entregador pode aceitar ou recusar pedido atribuído.

Entradas:
- ID do pedido

- Ação (Aceitar/Recusar)
- Motivo da recusa (se aplicável)

Processamento:
1. Verificar se pedido ainda está disponível
2. **Se aceitar:**
   - Atualizar status para "Aceito"
   - Iniciar cronômetro de preparo
   - Notificar cliente
3. **Se recusar:**
   - Marcar como recusado
   - Registrar motivo
   - Notificar administrador para reatribuição

Saídas:
- Status atualizado
- Notificações enviadas

Regras de Negócio:
- RN47: Tempo máximo para aceitar/recusar: 5 minutos
- RN48: Após 3 recusas, entregador recebe advertência
- RN49: Recusa automática após timeout

RF15 - Atualizar Status da Entrega (Entregador)
Prioridade: Alta
Caso de Uso: UC16

Descrição: Entregador atualiza progresso da entrega em tempo real.

Entradas:
- ID do pedido
- Novo status
- Localização GPS (automática)
- Observações (opcional)

Processamento:
1. Validar sequência lógica de status (RN55)
2. Capturar coordenadas GPS
3. Registrar timestamp

4. Atualizar banco de dados
5. Enviar ao Sistema de Entregas (UC21)
6. Acionar notificação push ao cliente (UC07)

Sequência de Status:

Em Preparo → Saiu para Entrega → A Caminho → Entregue

Saídas:
- Status atualizado
- Cliente notificado
- Timeline atualizada

Regras de Negócio:
- RN55: Status deve seguir ordem lógica obrigatória
- RN56: Cada mudança registra timestamp e GPS
- RN57: Captura de GPS quando disponível (não bloqueante)
- RN58: Cliente notificado em cada mudança

RF16 - Confirmar Entrega (Entregador)
Prioridade: Alta
Caso de Uso: UC17

Descrição: Entregador confirma entrega concluída ao cliente.

Entradas:
- ID do pedido
- Foto do comprovante (opcional)
- Assinatura digital ou código de confirmação

Processamento:
1. Verificar status atual (deve estar "A Caminho")
2. Capturar localização final
3. Registrar horário de entrega
4. Fazer upload da foto (se houver)
5. Atualizar status para "Entregue"
6. Calcular tempo total de entrega
7. Notificar cliente e administrador

Saídas:
- Pedido marcado como "Entregue"
- Comprovante armazenado
- Notificações enviadas

Regras de Negócio:
- RN59: Entrega só pode ser confirmada se status for "A Caminho"
- RN60: Foto de comprovante obrigatória para pedidos > R$ 200
- RN61: Cliente pode avaliar após confirmação de entrega

RF17 - Sistema de Recomendações (ML/LLM)
Prioridade: Alta
Caso de Uso: RF09 (Novo)

Descrição: Sistema gera recomendações personalizadas usando Machine Learning.

Entradas:
- Histórico de pedidos do usuário
- Preferências declaradas
- Comportamento de navegação
- Contexto (dia da semana, horário, ocasião)

Processamento:
1. Coletar dados do usuário (anonimizados)
2. Aplicar modelo de ML treinado (Collaborative Filtering)
3. Considerar fatores contextuais
4. Ranquear produtos por relevância
5. Aplicar regras de negócio (disponibilidade, estoque)
6. Registrar recomendação para feedback loop

Saídas:
- Lista de até 10 produtos recomendados
- Score de relevância
- Razão da recomendação (ex: "Baseado em suas compras anteriores")

Regras de Negócio:
- RN62: Recomendações apenas de produtos disponíveis em estoque
- RN63: Modelo retreinado semanalmente com novos dados

- RN64: Monitoramento de concept drift automatizado
- RN65: Dados anonimizados conforme LGPD

RF18 - Avaliação de Produtos
Prioridade: Baixa
Caso de Uso: Novo

Descrição: Clientes podem avaliar produtos após entrega.

Entradas:
- ID do pedido
- ID do produto
- Nota (1 a 5 estrelas)
- Comentário (opcional, max 500 caracteres)
- Fotos (opcional, max 3)

Processamento:
1. Verificar se usuário recebeu o pedido
2. Verificar se já avaliou o produto
3. Validar nota e comentário
4. Armazenar avaliação
5. Recalcular média de avaliações do produto
6. Atualizar cache

Saídas:
- Avaliação registrada
- Média do produto atualizada
- Agradecimento ao cliente

Regras de Negócio:
- RN66: Apenas clientes que compraram podem avaliar
- RN67: Avaliação disponível até 30 dias após entrega
- RN68: Usuário pode avaliar apenas uma vez por produto/pedido
- RN69: Comentários passam por moderação básica (filtro de palavrões)

3.2 Matriz de Rastreabilidade de Requisitos Funcionais

ID	Requisito	UC Relacionado	Prioridade	Status
RF01	Cadastro de Usuário	UC01	Alta	Aprovado
RF02	Autenticação	UC02	Alta	Aprovado
RF03	Visualizar Catálogo	UC03	Alta	Aprovado
RF04	Gerenciar Carrinho	UC04	Alta	Aprovado
RF05	Efetuar Pedido	UC04	Alta	Aprovado
RF06	Processar Pagamento	UC05	Alta	Aprovado
RF07	Acompanhar Pedido	UC06	Média	Aprovado
RF08	Notificações Push	UC07	Média	Aprovado
RF09	Gerenciar Catálogo	UC09	Alta	Aprovado
RF10	Gerenciar Estoque	UC10	Alta	Aprovado
RF11	Relatórios e Dashboards	UC11	Alta	Aprovado
RF12	Atribuir Pedido	UC08	Alta	Aprovado
RF13	Visualizar Pedidos (Entregador)	UC14	Média	Aprovado
RF14	Aceitar/Recusar Entrega	UC15	Média	Aprovado
RF15	Atualizar Status	UC16	Alta	Aprovado
RF16	Confirmar Entrega	UC17	Alta	Aprovado
RF17	Recomendações ML	-	Alta	Aprovado
RF18	Avaliação de Produtos	-	Baixa	Aprovado

---

4. MODELAGEM DO SISTEMA

4.1 Diagrama de Casos de Uso

Conforme documentado no **Documento de Caso de Uso Geral**, o sistema contempla 21 casos de uso principais distribuídos entre os atores:

Atores:
- Cliente
- Administrador
- Entregador
- Sistema de Pagamento
- Sistema de Entregas
- Sistema de Notificações

Casos de Uso Principais:
- UC01 a UC07: Funcionalidades do Cliente
- UC08 a UC13: Funcionalidades do Administrador

- UC14 a UC17: Funcionalidades do Entregador
- UC18 a UC21: Integrações com Sistemas Externos

Referência: Consultar Documento de Caso de Uso Geral para diagramas completos.

4.2 Diagrama de Classes

Classes Principais:

Usuário

Classe: Usuario Atributos:

- id_usuario: UUID (PK)
- nome: String
- email: String (unique)
- senha_hash: String
- telefone: String
- cpf: String (nullable)
- tipo_usuario: Enum (Cliente, Admin, Entregador)
- ativo: Boolean
- data_criacao: DateTime
- ultimo_acesso: DateTime

Métodos:

- autenticar(senha): Boolean
- atualizarPerfil(dados): Boolean
- validarCPF(): Boolean
- alterarSenha(novaSenha): Boolean

Produto

Classe: Produto Atributos:

- id_produto: UUID (PK)
- nome: String
- descricao: Text
- categoria: Enum (Doce, Salgado)

- preco: Decimal
- custo_unitario: Decimal
- quantidade_estoque: Integer
- imagem_url: String[]
- ingredientes: Text
- info_nutricional: JSON
- ativo: Boolean
- avaliacao_media: Float

Métodos:

- atualizarEstoque(quantidade): Boolean
- validarDisponibilidade(): Boolean
- calcularPrecoTotal(quantidade): Decimal
- calcularMargemLucro(): Float

Pedido

Classe: Pedido Atributos:

- id_pedido: UUID (PK)
- id_usuario: UUID (FK)
- id_endereco: UUID (FK)
- id_entregador: UUID (FK, nullable)
- data_pedido: DateTime
- valor_total: Decimal
- taxa_entrega: Decimal
- status: Enum (AguardandoPagamento, Pago, EmPreparo, SaiuEntrega, ACaminho, Entregue, Cancelado)
- data_entrega_prevista: DateTime
- observacoes: Text

Métodos:

- calcularTotal(): Decimal
- atualizarStatus(novoStatus): Boolean
- validarPedido(): Boolean
- cancelar(motivo): Boolean

ItemPedido

Classe: ItemPedido Atributos:

- id_item: UUID (PK)
- id_pedido: UUID (FK)
- id_produto: UUID (FK)
- quantidade: Integer
- preco_unitario: Decimal
- subtotal: Decimal
- observacoes: Text

Métodos:

- calcularSubtotal(): Decimal
- validarQuantidade(): Boolean

Pagamento

Classe: Pagamento Atributos:

- id_pagamento: UUID (PK)
- id_pedido: UUID (FK)
- metodo_pagamento: Enum (Pix, CartaoCredito, CartaoDebito)
- valor: Decimal
- status_transacao: Enum (Pendente, Aprovado, Recusado, Estornado)
- data_pagamento: DateTime
- codigo_transacao: String
- gateway_response: JSON

Métodos:

- processarPagamento(): Boolean
- validarTransacao(): Boolean
- gerarComprovante(): PDF
- estornar(): Boolean

Relacionamentos:
- Usuario 1:N Pedido
- Usuario 1:N Endereco
- Pedido 1:N ItemPedido
- Produto 1:N ItemPedido
- Pedido 1:1 Pagamento

- Pedido N:1 Endereco
- Pedido N:1 Usuario (Entregador)

4.3 Diagrama de Sequência - Realizar Pedido com Pagamento

Cliente → Interface: login() Interface → Controller: validarCredenciais() Controller → BD: consultarUsuario() BD → Controller: dadosUsuario Controller → Interface: autenticacaoOK

Cliente → Interface: selecionarProdutos() Interface → Controller: consultarProdutos() Controller → BD: listarProdutosAtivos() BD → Controller: listaProdutos Controller → Interface: exibirCatalogo()

Cliente → Interface: adicionarAoCarrinho(produto, qtd) Interface → Controller: validarEstoque(produto, qtd) Controller → BD: consultarEstoque() BD → Controller: estoqueDisponivel Controller → Interface: itemAdicionado()

Cliente → Interface: finalizarPedido() Interface → Controller: criarPedido(dados) Controller → BD: inserirPedido() BD → Controller: pedidoCriado

Controller → ControllerPagamento: iniciarPagamento(pedido) ControllerPagamento → MercadoPago: enviarTransacao(dados) MercadoPago → ControllerPagamento: transacaoAprovada ControllerPagamento → BD: atualizarStatus("Pago") ControllerPagamento → Interface: exibirConfirmacao() Interface → Cliente: pedidoConfirmado

4.4 Diagrama de Estados - Ciclo de Vida do Pedido

[Criado] → AguardandoPagamento ↓ AguardandoPagamento → Pago (pagamento aprovado) AguardandoPagamento → Cancelado (timeout 15min ou falha pagamento) ↓ Pago → EmPreparo (atribuído ao entregador) ↓ EmPreparo → SaiuParaEntrega ↓ SaiuParaEntrega → ACaminho ↓ ACaminho → Entregue ↓ Entregue → [Finalizado]

Qualquer estado → Cancelado (por solicitação ou problema)

4.5 Diagrama Entidade-Relacionamento (ER)

Entidades Principais:

USUARIOS ├── id_usuario (PK) ├── nome ├── email (UNIQUE) ├── senha_hash ├── telefone ├── cpf ├── tipo_usuario ├── ativo └── data_criacao

ENDERECOS ├── id_endereco (PK) ├── id_usuario (FK) ├── cidade ├── rua ├── numero ├── bairro ├── cep ├── complemento ├── referencia └── principal (boolean)

PRODUTOS ├── id_produto (PK) ├── nome ├── descricao ├── categoria ├── preco ├── custo_unitario ├── quantidade_estoque ├── imagem_url ├── ingredientes ├── info_nutricional ├── ativo └── avaliacao_media

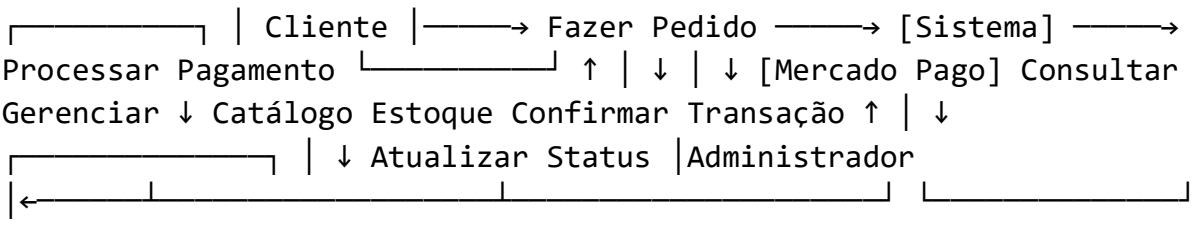
PEDIDOS ├── id_pedido (PK) ├── id_usuario (FK) ├── id_endereco (FK) ├── id_entregador (FK, nullable) ├── data_pedido ├── valor_total ├── taxa_entrega ├── status ├── data_entrega_prevista └── observacoes

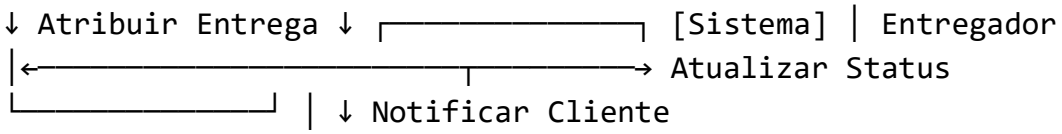
ITENS_PEDIDO ├── id_item (PK) ├── id_pedido (FK) ├── id_produto (FK) ├── quantidade ├── preco_unitario ├── subtotal └── observacoes

PAGAMENTOS ├── id_pagamento (PK) ├── id_pedido (FK) ├── metodo_pagamento ├── valor ├── status_transacao ├── data_pagamento ├── codigo_transacao └── gateway_response

AVALIACOES ├── id_avaliacao (PK) ├── id_produto (FK) ├── id_usuario (FK) ├── id_pedido (FK) ├── nota (1-5) ├── comentario └── data_avaliacao

4.6 Diagrama de Fluxo de Dados (DFD) - Nível 0





4.7 Modelo de SDLC Adotado

Metodologia: Scrum (Ágil) + TDD (Test-Driven Development)

Ciclo de Desenvolvimento:

1. PLANEJAMENTO DO SPRINT (2 dias) ├── Refinamento do backlog ├── Estimativa de story points └── Definição de sprint goal
2. DESENVOLVIMENTO (7 dias) ├── TDD: Escrever testes primeiro ├── Implementação de features ├── Code review (pull requests) └── Daily meetings (15min)
3. TESTES E QA (2 dias) ├── Testes de integração ├── Testes de aceitação ├── Testes de segurança └── Correção de bugs
4. DEPLOY (1 dia) ├── CI/CD automatizado ├── Deploy em homologação ├── Smoke tests └── Deploy em produção (blue/green)
5. RETROSPECTIVA (meio dia) ├── Análise do sprint ├── Lições aprendidas └── Plano de melhorias

Total: 2 semanas por sprint

---

5. REQUISITOS DE INTERFACE

5.1 Interface do Usuário (UI)

5.1.1 Princípios de Design

Filosofia: Material Design 3 (Android) / Human Interface Guidelines (iOS)

Diretrizes:
- **Simplicidade:** Máximo 3 cliques para qualquer função principal
- **Consistência:** Padrões visuais uniformes em todas as telas

- **Feedback Visual:** Resposta imediata a todas as ações do usuário
- **Acessibilidade:** Conformidade WCAG 2.1 Nível AA

5.1.2 Paleta de Cores

Primária: #E91E63 (Rosa vibrante - identidade cupcakes) Secundária: #FFC107 (Âmbar - call-to-actions) Fundo: #FFFFFF (Branco puro - clareza) Texto: #212121 (Cinza escuro - legibilidade) Erro: #F44336 (Vermelho - alertas) Sucesso: #4CAF50 (Verde - confirmações) Modo Escuro:#121212 (Preto suave - conforto visual noturno)

5.1.3 Tipografia

- **Fonte Principal:** Roboto (Android) / SF Pro (iOS)
- **Títulos:** 24sp (peso 700 - Bold)
- **Subtítulos:** 18sp (peso 600 - Semi-bold)
- **Corpo:** 16sp (peso 400 - Regular)
- **Legendas:** 14sp (peso 400 - Regular)

5.1.4 Telas Principais

TELA001 - Splash Screen
- Logo animado (2 segundos)
- Verificação de autenticação
- Transição suave