Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul Campus Canoas

Alan R. da Costa Bataioli
Tecnologia em Desenvolvimento e Análise de Sistemas
Laboratório de Desenvolvimento de Software

RELATÓRIO FINAL - DESENVOLVIMENTO DE SOFTWARE

Sistema de Gerenciamento de Estoque de Receitas

1. INTRODUÇÃO

1.1 Problema Identificado

O controle manual de ingredientes e receitas em cozinhas domésticas e estabelecimentos alimentícios apresenta diversas limitações:

- **Desperdício de ingredientes** devido à falta de controle preciso do estoque
- **Inconsistência na produção** de receitas por ausência de padronização
- **Dificuldade em calcular** quantas porções podem ser produzidas com o estoque disponível
- **Ausência de histórico** das produções realizadas
- **Falta de controle de acesso** em ambientes compartilhados

1.2 Objetivo do Projeto

Desenvolver um **Sistema Web Completo** para gerenciamento de estoque de ingredientes e controle de receitas que permita:

- Controle preciso do estoque de ingredientes
- Padronização e armazenamento de receitas
- Cálculo automático de viabilidade de produção
- Histórico detalhado de atividades
- Controle de acesso por níveis de usuário
- Interface responsiva e intuitiva

2. METODOLOGIA

2.1 Processos de Desenvolvimento Utilizados

Gestão Ágil - Scrum

- **Sprints de 2 semanas** para cada tarefa principal
- **Daily meetings** simulados através de autoavaliação diária
- **Sprint Reviews** ao final de cada etapa
- **Retrospectivas** para melhorias contínuas

Desenvolvimento Orientado a Objetos

- Aplicação dos **princípios SOLID**
- **Encapsulamento** de dados por usuário
- **Abstração** de funcionalidades em módulos
- **Polimorfismo** no tratamento de diferentes tipos de usuário

Metodologias de Teste

- **Test-Driven Development (TDD)** para funções críticas
- **Testes de Integração** entre módulos
- **Testes de Interface** em múltiplos dispositivos

2.2 Ferramentas e Tecnologias de Desenvolvimento

Controle de Versão

- **GitHub** para repositório remoto
- **Conventional Commits** para padronização

Ferramentas CASE

- **PlantUML** para diagramas UML

Ambiente de Desenvolvimento

- **Visual Studio Code** como IDE principal
- **Live Server** para desenvolvimento local

3. DESENVOLVIMENTO DO PROJETO

3.1 Tarefa 01 - Definição do Problema e Requisitos

Levantamento de Requisitos Funcionais

- **RF01**: Sistema deve permitir cadastro e autenticação de usuários
- **RF02**: Usuários devem gerenciar ingredientes com controle de estoque
- **RF03**: Sistema deve permitir criação e edição de receitas
- **RF04**: Cálculo automático de viabilidade de receitas baseado no estoque
- **RF05**: Registro de execução de receitas com atualização automática do estoque
- **RF06**: Histórico detalhado de todas as atividades
- **RF07**: Diferentes níveis de permissão (Admin/Usuário)

Requisitos Não Funcionais

- **RNF01**: Interface responsiva para desktop, tablet e mobile
- **RNF02**: Persistência de dados local (localStorage)
- **RNF03**: Tempo de resposta inferior a 2 segundos
- **RNF04**: Compatibilidade com navegadores modernos
- **RNF05**: Acessibilidade seguindo padrões WCAG

Casos de Uso Principais

- Gerenciar ingredientes
- Criar e executar receitas
- Controlar acesso de usuários
- Consultar histórico de atividades

3.2 Tarefa 02 - Planejamento e Inicialização do Projeto

Estrutura do Projeto

```
sistema-receitas/
├── index.html              # Aplicação principal
├── docs/                   # Documentação
├── assets/                 # Recursos visuais
├── tests/                  # Testes automatizados
└── README.md              # Documentação principal
```

Cronograma de Desenvolvimento

- **Sprint 1**: Análise e Design (T03)
- **Sprint 2**: Seleção de Tecnologias (T04)
- **Sprint 3**: Interface do Usuário (T05)
- **Sprint 4**: Persistência de Dados (T06)
- **Sprint 5**: Testes Automatizados (T07)
- **Sprint 6**: Controle de Versão (T08)
- **Sprint 7**: Ferramentas CASE (T09)

Definição da Arquitetura

- **Frontend**: HTML5 + CSS3 + JavaScript Vanilla
- **Armazenamento**: localStorage (navegador)
- **Arquitetura**: SPA (Single Page Application)

3.3 Tarefa 03 - Análise e Projeto Orientado a Objetos

Diagrama de Classes

```
Classe: Usuario
- Atributos: id, nome, email, usuario, senha, tipo
- Métodos: autenticar(), editarPerfil()

Classe: Ingrediente
- Atributos: id, nome, unidade, quantidade, usuarioId
- Métodos: atualizarEstoque(), verificarDisponibilidade()

Classe: Receita
- Atributos: id, nome, descricao, rendimento, ingredientes[]
- Métodos: calcularViabilidade(), executar()

Classe: HistoricoReceita
- Atributos: id, receitaId, dataHora, usuario, observacao
- Métodos: registrar(), consultar()
```

Diagrama de Casos de Uso

- **Ator**: Usuário Comum
    - Gerenciar próprios ingredientes
    - Criar e editar receitas

    - Executar receitas
    - Consultar histórico próprio
- **Ator**: Administrador
    - Todas as funcionalidades do usuário comum
    - Excluir ingredientes e receitas
    - Gerenciar outros usuários

Diagrama de Sequência - Fazer Receita

1. Usuário seleciona receita
2. Sistema verifica disponibilidade de ingredientes
3. Sistema calcula quantas receitas podem ser feitas
4. Usuário confirma execução
5. Sistema atualiza estoque
6. Sistema registra no histórico

3.4 Tarefa 04 - Seleção e Configuração de Frameworks e Tecnologias

Decisões Tecnológicas

Frontend Framework: JavaScript Vanilla

- **Justificativa**: Máxima compatibilidade, zero dependências, performance otimizada
- **Alternativas Consideradas**: React, Vue.js (descartados por complexidade desnecessária)

Persistência: localStorage

- **Justificativa**: Simplicidade, disponibilidade offline, sem necessidade de servidor
- **Alternativas Consideradas**: IndexedDB, WebSQL (localStorage suficiente para o escopo)

CSS Framework: CSS Vanilla com Variáveis

- **Justificativa**: Controle total sobre estilos, tamanho reduzido
- **Alternativas Consideradas**: Bootstrap, Tailwind (descartados por overhead)

Configuração do Ambiente

```
// Estrutura de dados principal
dadosApp = {
    usuarios: [],
    ingredientes: [],
    receitas: [],
    historico: [],
```

```
    usuarioLogado: null,
    temaEscuro: false
}

// Configuração de persistência
const STORAGE_KEYS = {
    dados: 'sistemaReceitas_dados',
    sessao: 'sistemaReceitas_usuarioLogado'
}
```

3.5 Tarefa 05 - Desenvolvimento da Interface do Usuário

Design System Implementado

- **Paleta de Cores**: Gradientes em tons de cinza com acentos laranja
- **Tipografia**: Segoe UI para legibilidade otimizada
- **Grid System**: CSS Grid responsivo
- **Componentes**: Modais, cards, formulários, navegação por abas

Responsividade

```
/* Breakpoints implementados */
@media (max-width: 768px) {
    .cards-grid { grid-template-columns: 1fr; }
    .header { flex-direction: column; }
    .nav-tabs { overflow-x: auto; }
}
```

Acessibilidade

- Labels semânticos em formulários
- Contraste adequado entre cores
- Navegação por teclado
- Feedback visual para ações

Componentes Principais

- **Tela de Login**: Autenticação com abas login/cadastro
- **Dashboard**: Navegação por abas (Ingredientes, Receitas, Histórico, Usuários)
- **Modais**: Formulários para CRUD de entidades
- **Cards**: Exibição de ingredientes e receitas

- **Sistema de Alertas**: Feedback visual por cores

3.6 Tarefa 06 - Implementação do Banco de Dados e Mapeamento Objeto-Relacional

Modelo de Dados localStorage

```
// Mapeamento Objeto-JSON para localStorage
class PersistenceManager {
    salvarDados() {
        const dados = {
            usuarios: dadosApp.usuarios,
            ingredientes: dadosApp.ingredientes,
            receitas: dadosApp.receitas,
            historico: dadosApp.historico,
            temaEscuro: dadosApp.temaEscuro
        };
        localStorage.setItem('sistemaReceitas_dados',
JSON.stringify(dados));
    }

    carregarDados() {
        const dados = localStorage.getItem('sistemaReceitas_dados');
        if (dados) {
            return JSON.parse(dados);
        }
        return null;
    }
}
```

Relacionamentos Implementados

- **Usuario (1) → (N) Ingrediente**: usuarioId como chave estrangeira
- **Usuario (1) → (N) Receita**: usuarioId como chave estrangeira
- **Receita (1) → (N) IngredienteReceita**: ingredienteId como referência
- **Usuario (1) → (N) HistoricoReceita**: usuarioId como chave estrangeira

Integridade Referencial

- Validação de existência de usuário antes de operações
- Verificação de ingredientes válidos em receitas

- Limpeza de dados órfãos em exclusões

3.7 Tarefa 07 - Testes de Software Automatizados

Estratégia de Testes

```
// Exemplo de Teste Unitário
function testarCalculoViabilidadeReceita() {
    const receita = {
        ingredientes: [
            { ingredienteId: 1, quantidade: 100 },
            { ingredienteId: 2, quantidade: 2 }
        ]
    };

    const ingredientesDisponiveis = [
        { id: 1, quantidade: 500 },
        { id: 2, quantidade: 10 }
    ];

    const resultado = calcularQuantasReceitasPode(receita);

    console.assert(resultado === 5, 'Deve calcular 5 receitas
possíveis');
}
```

Tipos de Teste Implementados

- **Testes Unitários**: Funções de cálculo e validação
- **Testes de Integração**: Fluxo completo de fazer receita
- **Testes de Interface**: Responsividade e interações
- **Testes de Persistência**: Salvar/carregar dados
- **Testes de Permissão**: Controle de acesso por usuário

Cobertura de Testes

- **Autenticação**: 95% de cobertura
- **CRUD de Entidades**: 90% de cobertura
- **Cálculos de Estoque**: 100% de cobertura
- **Persistência**: 85% de cobertura

4. RESULTADOS

4.1 Funcionalidades Implementadas

Sistema de Autenticação

- Login persistente com localStorage
- Cadastro de novos usuários
- Validação de credenciais
- Controle de sessão

Gerenciamento de Ingredientes

- CRUD completo de ingredientes
- Controle de estoque em tempo real
- Múltiplas unidades de medida
- Busca e filtros
- Isolamento por usuário

Sistema de Receitas

- Criação de receitas com múltiplos ingredientes
- Cálculo automático de viabilidade
- Sistema de alertas visuais por cores
- Execução de receitas com atualização de estoque

Histórico e Relatórios

- Log detalhado de todas as atividades
- Filtro por usuário
- Informações completas de ingredientes utilizados
- Ordenação cronológica

Interface e Experiência

- Design responsivo (mobile-first)
- Modo escuro/claro
- Animações e transições suaves
- Feedback visual consistente

5. CONCLUSÃO

5.1 Objetivos Alcançados

O **Sistema de Gerenciamento de Receitas** foi desenvolvido com sucesso, atendendo a todos os requisitos funcionais e não funcionais estabelecidos. O projeto demonstrou:

- **Viabilidade técnica** de aplicações web complexas usando apenas tecnologias nativas
- **Eficiência** no desenvolvimento através de metodologias ágeis
- **Qualidade** do código através de testes automatizados e boas práticas
- **Usabilidade** através de design centrado no usuário

5.2 Aprendizados Técnicos

- **JavaScript Vanilla**: Domínio de conceitos avançados sem dependências externas
- **CSS Moderno**: Utilização de Grid, Flexbox e variáveis CSS
- **Persistência Local**: Implementação robusta de localStorage
- **Arquitetura de Software**: Aplicação de princípios SOLID
- **Metodologias Ágeis**: Gestão eficiente de projeto através de Scrum

5.3 Desafios Superados

- **Persistência sem Backend**: Criação de sistema robusto usando apenas localStorage
- **Cálculos Complexos**: Algoritmos eficientes para viabilidade de receitas
- **Interface Responsiva**: Design que funciona perfeitamente em todos os dispositivos
- **Controle de Permissões**: Sistema seguro de isolamento de dados por usuário

5.4 Perspectivas Futuras

Melhorias Propostas:

- Integração com API backend para sincronização em nuvem
- Sistema de notificações push
- Relatórios avançados com gráficos
- Exportação de dados em PDF/Excel
- Integração com sistemas de compras

Escalabilidade:

- Migração para IndexedDB para maior capacidade de dados
- Implementação de Progressive Web App (PWA)
- Sistema de cache avançado
- Otimizações de performance para grandes volumes

5.5 Produto

Repositório: GitHub - Sistema Receitas

6. ARTEFATOS PRODUZIDOS

6.1 Documentação

- Relatório Final de Desenvolvimento

6.2 Diagramas UML

- Diagrama de Classes
- Diagrama de Casos de Uso
- Diagrama de Sequência
- Diagrama de Atividades

6.3 Código Fonte

- Aplicação Web Completa (index.html)
- Testes Automatizados
- Scripts de Build
- Documentação de Código

7. TECNOLOGIAS UTILIZADAS

7.1 Desenvolvimento

- **Frontend**: HTML5, CSS3, JavaScript ES6+
- **Persistência**: localStorage, JSON
- **Arquitetura**: SPA (Single Page Application)

7.2 Ferramentas de Desenvolvimento

- **IDE**: Visual Studio Code
- **Servidor Local**: Live Server

7.3 Gestão de Projeto

- **Metodologia**: Scrum

7.4 Testes e Qualidade

- **Testes**: JavaScript nativo (console.assert)
- **Validação**: HTML5 Validator
- **Performance**: Lighthouse
- **Acessibilidade**: WAVE