PizzaSystem: Desenvolvimento de um Sistema de Gerenciamento com Abordagem Dupla em Console e Web

Davi Gonçalves Silva

Gustavo Zaia Pastro

UniAnchieta

Curso Ciências da Computação

Av. Odila Azalim, 575 – Vila Nova Jundiainopolis

Jundiaí/SP, Brasil, 13210-795 {gupastro@gmail.com, davigonssilva@gmail.com}

Resumo – Este trabalho apresenta o desenvolvimento do PizzaSystem, um sistema de gerenciamento para pizzaria criado para fins estudantis. O projeto demonstra a aplicação de conceitos fundamentais em duas implementações distintas: (1) uma aplicação de console desenvolvida em TypeScript [3] puro executada em Node.js [2], e (2) uma aplicação *Web* composta por uma interface e uma *API REST* desenvolvida utilizando o *Express* [5], com o banco de dados *SQL SERVER* [6] via *Docker* [7] para a persistência. O programa conta com sistemas essenciais de uma Pizzaria, sendo eles: Cadastro de Clientes, Gerenciamento de Produtos, Gerenciamento de Pedidos e Relatórios [1].

I. Introdução

A gestão eficiente de operações é um pilar extremamente fundamental para pequenos e médios negócios, ou que estão começando agora, como restaurantes ou pizzarias. No cenário atual, um sistema de cadastro de clientes, gerenciamento de pedidos são cruciais para reduzir erros humanos e automatizar essa tarefa e fornecer dados através dos relatórios gerados. A modernização desses processos que vão de gerenciamento de cadastros até controle financeiro, é fundamental para manter e agilizar o fluxo de trabalho e melhorar a experiência do consumidor.

Embora no mercado de trabalho exista diversas opções de softwares mais robustas, completas e complexas, existe um campo para o desenvolvimento de sistemas com arquiteturas mais simples, com tecnologias mais acessíveis. A criação de protótipos funcionais, como aplicações de console ou interfaces web, permite confirmar conceitos fundamentais, como manipulação de dados, de forma rápida e de baixo custo.

Neste cenário, este artigo apresenta o desenvolvimento do *PizzaSystem*, um sistema simples de gerenciamento de pizzaria criado para fins de estudo. O projeto utiliza de duas formas de implementação de um mesmo conjunto de regras em duas plataformas distintas: (1) um console interativo, utilizando *TypeScript,* sendo executado em um ambiente do `Node.js`, e (2) uma aplicação *Web* completa, contendo um *frontend* interativo e um *backend* em `Node.js` [1].

O objetivo deste trabalho é detalhar o processo de construção, organização e as técnicas que foram utilizadas em ambas as versões do sistema. Uma análise comparativa das duas abordagens: o uso de arquivos de texto `(.txt)` na versão de console e a utilização de um banco de dados *SQL SERVER* [6] na versão *Web*. As próximas seções deste artigo explicarão a arquitetura, a metodologia de desenvolvimento, os resultados obtidos e as conclusões de ambas as versões utilizadas.

II. Desenvolvimento e Metodologia

O corpo deste projeto irá detalhar as técnicas, organização do código e as estratégias adotadas no projeto de gerenciamento de pizzaria *PizzaSystem*.

A. Arquitetura e Organização do Projeto

O programa foi constituído em um único repositório *Git,* separando-as em diretórios diferentes garantindo a independência na hora de executar as variadas versões.

1. *Versão Console (TypeScript):* Esta versão é uma aplicação de linha de comando sendo executada sobre o `Node.js`. O código-fonte foi escrito em *TypeScript* e reside dentro do diretório `src/`. O arquivo `tsconfig.json` [1]

é o diretório que define as regras de compilação do código, ele especifica que o código em *TypeScript*(`.ts`) deve ser transpilado para *JavaScript*(`.js`) dentro do diretório `dist/`. As dependências do sistema são dirigidas pelo `package.json`, que inclui o *TypeScript* como vínculo de desenvolvimento e a biblioteca `readline-sync` [4] para ler e sincronizar as entradas do usuário no console.

2. *Versão Web (FullStack):* Uma versão com interface interativa e uma arquitetura cliente-servidor [1]. O sistema dessa versão é composto por 3 camadas principais, sendo elas: *Frontend, Backend, Banco de Dados.*

Na camada de *Frontend* (Interface do site) foi utilizado *HTML*, *CSS* e *JavaScript*, atuando como cliente que consome os dados via solicitações *HTTP* não simultâneas *(fetch)* [1].

Para o *Backend*, um servidor utilizando o *framework Express* [5] foi implementado por cima do `Node.js`. Este servidor tem como função expor as rotas (*endpoints*) *RESTful*, como o `/api/clientes` e `/api/pedidos`, que são responsáveis por processar as regras e gerenciar as solicitações vindas do *Frontend* [1].

O salvamento de dados ocorre em um sistema gerenciador de banco de dados relacional *Microsoft SQL Server* [6]. Para garantir uma maior portabilidade e facilidade de configuração do ambiente em que foi desenvolvido, foi utilizado um container *Docker* [7], orquestrado através do arquivo `docker-compose.yml` [1].

B. Técnicas de Construção e Lógica do Negócio

A lógica central (cadastros, pedidos, relatórios) foi implementada respeitando as particularidades de cada plataforma. Na versão de *TypeScript*, a lógica é rígida e executada localmente, processando os dados em memória e salvando-os diretamente no disco.

Já na versão *Web*, a lógica foi segregada em um modelo diferente. O *frontend* é responsável apenas pela apresentação e a interação com o usuário, o *JavaScript* é utilizado para manipular o *Document Object Model (DOM)* e gerenciar a exibição das telas (áreas de funcionário e do cliente). A comunicação com o servidor é feita utilizando a *API fetch*, que envia os dados no formato `JSON`, garantindo uma experiência melhor sem as páginas ficarem recarregando (*SPA – Single Page Application*) [1].

O aspecto que mais diferencia ambas as versões é em como os dados são salvos:

1. Versão Console (TypeScript): Os dados são armazenados em arquivos de textos sequenciais (`.txt`). Cada entidade possui seu próprio arquivo, manipulado através da importação da biblioteca `fs` (*File System*) do `Node.js`.

2. Versão Web: Busca como solução utilizar um banco de dados relacional *SQL SERVER* [6]. O *backend* faz uso da biblioteca `mssql` para se conectar ao banco e executar comandos *SQL* (`INSERT`, `UPDATE`, `SELECT`), garantindo uma integridade e que os dados sejam salvos e gerenciados independentemente do navegador ou do dispositivo em que o usuário escolheu para acessar o sistema.

C. Controle de Versionamento e Documentação

O controle de versões do projeto foi gerenciado pelo *Git*. O arquivo `.gitignore` [1] foi configurado para deletar arquivos de dependência, o código compilado e, o mais importante, os arquivos de texto gerados pela versão de console, com essa prática, apenas o código-fonte será rastreado pelo repositório *Git*.

A documentação principal está em um arquivo chamado `README.md` [1], descrevendo a funcionalidade do projeto, e contendo todo o passo a passo e requisitos para executar ambas as aplicações, junto ao fluxograma do projeto, detalhando suas operações.

III. Resultados e Discussão

A. Apresentação dos Resultados Funcionais

O principal resultado do projeto realizado é a capacidade de realizar as operações essenciais de um sistema de pizzaria. Ambas as versões realizam com sucesso as principais funcionalidades de um sistema de pizzaria, com um diferencial na versão *Web*. A aplicação na versão *HTML* foi dividida em duas áreas principais: Área do Cliente e Área do Funcionário [1]. As funcionalidades inclusas são:

1. *Área do Cliente:* Cadastro do Cliente e Acesso ao Cardápio para realizar pedidos [1].

2. *Área do Funcionário:* Painel administrativo do sistema da pizzaria, protegido por um login do usuário e senha de um administrador, que permite: Gerenciamento de Clientes, Gerenciamento de Produtos, Gerenciamento de Pedidos, Emissão de Relatórios e Gerenciamento de Status de Pedidos [1].

O fluxograma operacional do sistema, que aborda ambas as versões, está ilustrado na Fig. 1.

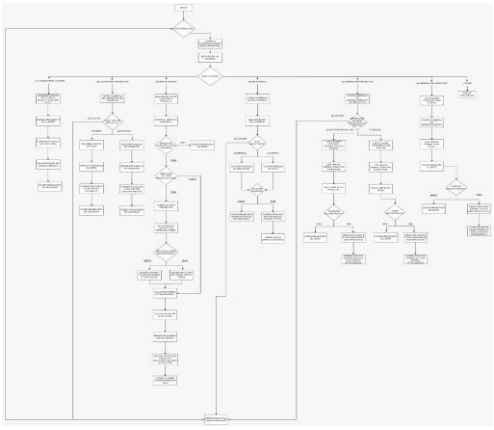

Fig. 1 – Fluxograma Operacional do PizzaSystem [1]

Adicionalmente, a versão *Web* possui uma interface de usuário (*UI*), que foi desenvolvida utilizando *HTML, CSS e JavaScript*. As diferentes abas da aplicação, como o Cadastro de Clientes (Fig. 2), demonstram na prática a aplicação da lógica *showView* e a manipulação do *DOM* para a exibição dos dados.

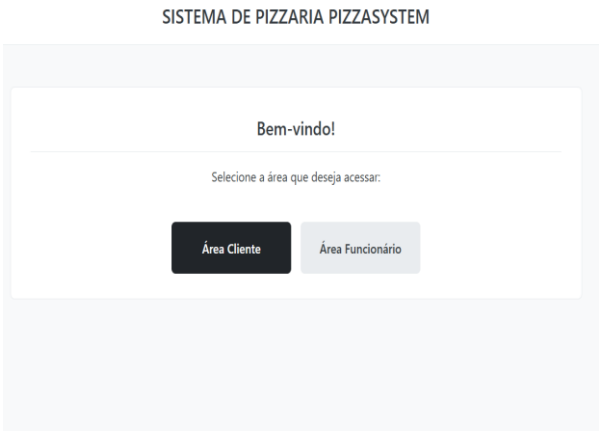

Fig. 2 – Página Inicial da Versão Web, demonstrando separação de áreas [1]

B. Discussão das Abordagens e Persistência

A análise comparativa das duas estratégias utilizadas na persistência de dados revela uma clara troca entre simplicidade e robustez.

O gerenciamento de dados baseado em arquivos de textos que foi utilizado na versão de console, é uma forma simples e bem eficiente de se realizar essa tarefa, não requerendo nenhuma instalação externa. No entanto, ela poderá enfrentar problemas se houver múltiplos acessos simultâneos e poderá ser ineficiente para tarefas mais complexas, exigindo a leitura e o *parsing* dos arquivos inteiros na memória.

Em contrapartida, a utilização do *SQL SERVER* na versão *Web* oferece robustez, integridade e escalabilidade. A arquitetura baseada em *API* e banco de dados resolve o problema de concorrência, permitindo múltiplos acessos simultâneos e consultas mais complexas que são otimizadas pelo motor do banco de dados. A principal limitação de utilizar essa abordagem é a complexidade, exigindo a configuração de um ambiente com `Node.js` e Docker. No entanto, o uso do `docker-compose` mitigou essa dificuldade, automatizando a criação do banco de dados. Dessa forma, a versão *Web* demonstra uma arquitetura viável para aplicações comerciais, superando as limitações da versão de *Console*.

IV. Conclusão

Este trabalho detalhou o processo de organização, técnicas e construção do projeto *PizzaSystem*, sistema de gerenciamento de uma pizzaria construído em duas versões com arquiteturas distintas. Os objetivos foram alcançados com sucesso, resultando em protótipos funcionais que implementam as operações necessárias.

A principal contribuição do estudo foi a análise entre as duas versões com diferentes gerenciamentos de dados: o uso de arquivos de texto (`.txt`) e o uso de um banco de dados relacional *SQL SERVER*. A implementação demonstrou que embora o uso de `.txt` seja simples, pode acabar apresentando problemas de eficiência, já a arquitetura baseada em *API* e banco de dados na versão *Web* se provou robusta, garantindo a integridade dos dados.

Conclui-se que a versão *Web* do projeto *PizzaSystem* supera os limites de um mero trabalho acadêmico, apresentando uma estrutura viável para aplicações comerciais, podendo ser utilizada como base para futuras expansões em ambientes semelhantes.

REFERÊNCIAS

[1] D. G. Silva e G. Z. Pastro. *PizzaSystem*. Repositório GitHub, 2025. Disponível em https://github.com/DaviGons/PizzaSystem. Acessado em: 11/11/2025.

[2] OpenJS Foundation. *Node.js.* https://nodejs.org/. Acessado em 11/11/2025.

[3] Microsoft. *TypeScript*. https://www.typescriptlang.org/. Acessado em 11/11/2025.

[4] A. K. T. *et al. readline-sync*. npm, inc. https://www.npmjs.com/package/readline-sync. Acessado em 11/11/2025

[5] StrongLoop, IBM, et al. *Express – Fast, unopinionated, minimalist web framework for Node.js* https://expressjs.com/. Acessado em 11/11/2025.

[6] Microsoft. *SQL Server Technical Documentation.* https://learn.microsoft.com/en-us/sql/sql-server/. Acessado em 11/11/2025.

[7] Docker Inc. *Docker Documentation.* https://docs.docker.com/. Acessado em 11/11/2025.