

# **Idealização e prototipagem de uma aplicação comercial escalável utilizando microsserviços**

Acadêmico MICHAEL KAWAN DA COSTA DE MELLO SILVA

Turma FLD209111CET

Professor mediador DIEGO MARTINS MOREIRA

Professor regente GUILHERME AUGUSTO REIS DOS SANTOS

## **RESUMO**

Este trabalho apresenta uma pesquisa bibliográfica aprofundada com o objetivo de analisar e integrar conceitos cruciais para o desenvolvimento de software moderno e de alta qualidade. Foram explorados temas como Domain-Driven Design (DDD), visando à modelagem de sistemas alinhados à estratégia de negócios; Microsserviços, como abordagem arquitetural para construir soluções escaláveis e modulares; Código limpo, focando em práticas para garantir legibilidade e manutenibilidade do código; e Programação funcional, como paradigma para lidar com a complexidade de forma eficaz. A metodologia adotada consistiu na revisão sistemática da literatura especializada, com base em obras de autores renomados nas respectivas áreas. Como conclusão, destaca-se que a sinergia entre essas disciplinas é fundamental para a criação de sistemas robustos, adaptáveis e sustentáveis, evidenciando a interdependência entre a arquitetura de software, as boas práticas de desenvolvimento e a compreensão do domínio.

**Palavras-chave:** MICROSERVIÇOS, DOMAIN-DRIVEN DESIGN(DDD) E ARQUITETURA.

## **INTRODUÇÃO**

No contexto atual da transformação digital, empresas realizam continuamente operações complexas e transferências de dados em grande escala. Nesse cenário, a escolha da arquitetura de software mais adequada torna-se um fator decisivo para garantir escalabilidade, flexibilidade e manutenção facilitada. Entre as abordagens arquitetônicas mais discutidas, destaca-se a arquitetura de microsserviços, que propõe a decomposição de sistemas em serviços independentes, cada um com responsabilidades bem definidas.

Durante a revisão de literatura, observou-se a forte relação entre a arquitetura de microsserviços e os princípios do Domain-Driven Design (DDD), os quais enfatizam a modelagem orientada ao domínio e a organização do sistema com base em regras de negócio. Os livros “Criando Microsserviços”, de Sam Newman (2022), e “Aprenda Domain-Driven Design”, de Vlad Khononov (2024), serviram como base teórica para o desenvolvimento do protótipo apresentado neste trabalho.

Diante disso, surgiu a seguinte pergunta de pesquisa: como aplicar conceitos de Domain-Driven Design em uma aplicação baseada em microsserviços, considerando um cenário com requisitos de negócio inicialmente indefinidos ou até indecisos?

O objetivo geral deste trabalho é prototipar uma aplicação distribuída, orientada por domínio, utilizando arquitetura de microsserviços, com base nas boas práticas apresentadas na literatura técnica.

Como objetivos específicos, destacam-se:

- Estudar os fundamentos do Domain-Driven Design e sua aplicabilidade em projetos de software;
- Compreender a arquitetura de microsserviços e seus principais benefícios e desafios;
- Definir um domínio fictício e suas fronteiras contextuais;
- Implementar um protótipo funcional baseado nos princípios de DDD e microsserviços;
- Avaliar a viabilidade técnica da abordagem em um ambiente com requisitos incertos.

Este trabalho, portanto, propõe-se a explorar como conceitos de modelagem e arquitetura podem ser aplicados desde as fases iniciais do desenvolvimento, mesmo diante de incertezas nos requisitos de negócio.

## **FUNDAMENTAÇÃO TEÓRICA**

Este capítulo apresenta a base teórica utilizada para o desenvolvimento do protótipo proposto, fundamentado nos livros Criando Microsserviços, de Sam Newman (2022), e Aprenda Domain-Driven Design, de Vlad Khononov (2024). O objetivo é demonstrar como os conceitos de microsserviços e Domain-Driven Design (DDD) guiaram a estruturação da aplicação.

Segundo o livro Aprenda Domain-Driven Design, de Vlad Khononov (2024, p.3-18) afirma que “O domínio do negócio define qual é a principal área de atividade da empresa”, “Um subdomínio da empresa é uma área de atividade comercial aprimorada.”

e ferramentas também conhecido como Subdomínios de suporte “Dão suporte aos negócios da empresa. No entanto, ao contrário dos subdomínios principais, [...] não fornecem vantagem competitiva.”. Um resumo no dia-a-dia dessas ferramentas são os métodos CRUD, que existe em várias camadas de um negócio, no entanto, não existe valor comercial nenhum em tentar reinventar funcionalidades, métodos e até ferramentas já consolidadas no mercado.

No cenário da prototipação apresentada, existe um cenário empresarial, que foi completamente possível se encaixar nos Domínios do DDD. Uma empresa fictícia, com seu domínio sendo as vendas e manufatura. Seus subdomínios principais envolvem métodos que fazem análise de dados extraindo diretamente dos maquinários. Já os subdomínios de suporte, são pequenas aplicações já consolidadas de terceiros que fazem todas as comunicações e apresentação dos dados.

No livro de Vlad Khononov (2024, p.33-46), o autor afirma “Enquanto os subdomínios são descobertos, os contextos delimitados são projetados. A divisão de um domínio em contextos delimitados é uma decisão de design estratégico.” Um contexto delimitado de uma atividade, pode ser implementado e mantido por uma equipe, duas equipes não podem trabalhar no mesmo contexto, embora devam existir maneiras de comunicação entre as mesmas. Essa lógica beneficia também a arquitetura de microsserviços, como será discutido posteriormente.

Na aplicação protótipo, tais contextos delimitados se alinham perfeitamente com a arquitetura de microsserviços. Sendo uma aplicação dividida em vários serviços independentes, com cada serviço tendo seu contexto bem definido. Cada uma delas tem sua lógica própria e não pode compartilhar responsabilidade, promovendo alta coesão e baixo acoplamento. Além disso, os métodos de comunicação favorecem essa separação, mantendo a integridade de cada serviço com seu respectivo contexto.

Segundo Sam Newman em Criando Microserviços(2022, p.24) “Microsserviços são serviços que podem ser lançados de forma independente e são modelados com base em um domínio de negócios. Um serviço encapsula uma funcionalidade e a torna acessível a outros serviços através de redes[...]”. Uma aplicação baseada nessa arquitetura é comumente implementada em nuvem e visando os domínios da empresa, mas nada impede de implementar localmente e focar nos subdomínios, mantendo os mesmos princípios da arquitetura.

Sobre as vantagens da arquitetura de microsserviços, Sam Newman afirma "Ao combinar os conceitos de ocultação de informações e um design orientado a domínios com a eficácia dos sistemas distribuídos, os microsserviços podem ajudar a proporcionar ganhos significativos em comparação com outras formas de arquiteturas distribuídas." (2022, p.45).

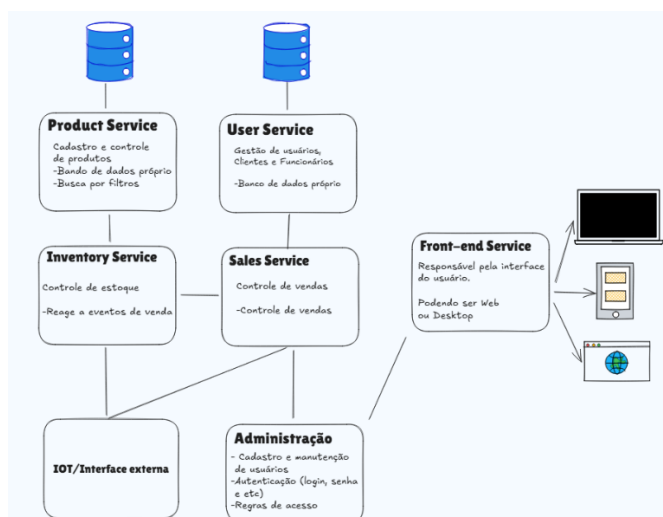


Figura 1 Exemplo arquitetura de autoria própria, desenhada através do Excalidraw – Cada bloco representando um serviço independente. autoria própria.

microsserviços que colaboram entre si, podemos optar por empregar diferentes tecnologias em cada microsserviço. Isso permite escolher a ferramenta correta para cada tarefa, em vez de selecionarmos uma única abordagem padronizada, que sirva para qualquer propósito, a qual muitas vezes acaba sendo o menor dominador comum." (2022, p.45 – Heterogeneidade de tecnologias).

Na aplicação protótipo, um Design de arquitetura foi desenhado, levando em consideração os princípios dos microsserviços e DDD já mencionados. Cada serviço é uma aplicação independente, com suas lógicas e comunicações bem definidas, além disso, isso nos favorece a heterogeneidade, como Sam Newman descreve "Com um sistema composto de vários

Sam Newman afirma "Uma startup com cinco pessoas provavelmente achará que uma arquitetura de microsserviços é um peso. Uma empresa em rápido crescimento, com uma centena de pessoas, provavelmente achará que seu crescimento será muito mais fácil se ela tiver uma arquitetura de microsserviços devidamente alinhada com seus esforços de desenvolvimento de produtos." (2022, p.58 – Quando microsserviços funcionam bem).

Isso reforça a escolha da arquitetura no cenário fictício que foi proposto, em que a empresa necessita de um protótipo escalável e alinhado com seu crescimento e complexidade organizacional.

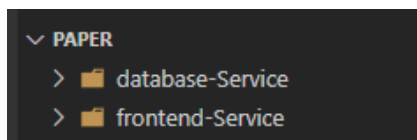
A arquitetura se comunica bem com o conceito de contextos delimitados apresentado anteriormente com DDD. Sam Newman menciona "A ocultação de informações descreve o desejo de ocultar o máximo possível de detalhes atrás da fronteira de um

módulo (ou, em nosso caso, de um microserviço).” (2022, p.61-62 – Ocultação de informação). Tal lógica é muito implementada na arquitetura de microserviços, onde cada serviço é responsável por sua lógica e dados, expondo apenas o necessário para interação com os demais.

Sam Newman(Criando Microserviços, 2022 - p.116-148) menciona que existem diferentes tipos de comunicação que podem ser utilizados, como o consolidado protocolo HTTP, que será utilizado na prototipação. Mas o principal critério em torno do método escolhido são as ações compensatórias alinhada que são aplicadas, como tratamento de erros em cada serviço, que favorece um baixo acoplamento e alta coesão entre os módulos. Além disso, os serviços precisam implementar padrões de sincronidade entre os mesmos, mas isso é um aspecto que precisa ser levado em consideração em aplicações mais robustas e com regras específicas sobre o uso da mesma.

Como mencionado anteriormente, o uso de ferramentas já consolidadas para os métodos comuns em processos que não geram valor é algo comum. Existem diversas no mercado que atendem bem esse propósito, como o Java com o ecossistema Spring Boot, que traz diversas ferramentas que podem ser utilizadas para tal.

A ferramenta escolhida para o desenvolvimento do protótipo foi o NestJS, uma tecnologia robusta e moderna para desenvolvimento back-end com JavaScript/TypeScript, construída sobre a biblioteca Express. O NestJS permite estruturar aplicações de forma modular e escalável, facilitando a organização do código. A biblioteca Express traz diversas ferramentas para utilização dos métodos HTTP de maneira simples, eficiente e já bastante documentada. Além disso, foi utilizada a biblioteca pg-promise, ela realiza conexão e interação com banco de dados PostgreSQL de maneira mais abstraída e ausenta a implementação de lógicas complexas para utilização do mesmo.



*Figura 2 Visão das pastas do projeto.  
autoria própria.*

Essa aplicação foi reduzida em apenas dois serviços, mas que implementam todos os conceitos aqui apresentados.

O database-Service é o serviço responsável pela comunicação com o banco de dados. Esse serviço apenas implementa a lógica de conexão e manipulação do mesmo. Expõe uma API para os possíveis demais serviços realizar as comunicações, junto com

tratamento de erros e exceções do mesmo, isso significa que mesmo se algum tipo de dado mal informado for enviado ela não será finalizada, apenas retornará um erro para o serviço solicitante, isso se alinha com o princípio de baixo acoplamento.

O frontend-Service é um serviço de apresentação das informações e que se comunica com database-Service enviando-o requisições HTTP com um corpo JSON e os dados. Esse serviço se divide em duas partes: Back-end e Front-end. O código inserido no back-end é responsável por enviar os arquivos estáticos usando o Express, como HTML, CSS e Javascript para o navegador do usuário, assim como resgatar as solicitações do usuário, processa-los no back-end ou fazer algum tipo de verificação e então realizar as requisições HTTP com os dados usando a biblioteca do Javascript chamada Axios, também retornando algum possível erro.

É importante ressaltar que na aplicação protótipo não existe nenhum tipo de segurança implementada entre cada serviço, o que é indispensável num sistema real e especialmente em serviços fornecidos através de nuvem ou WEB pública. A ausência de sistema de segurança entre os serviços permite que agentes maliciosos conectados na rede possam se comunicar através dos endpoints de cada serviço de maneira muito fácil e descontrolada. A ausência dessa implementação se deve a alguns fatores, como diferentes tipos de lógicas no sistema que precisam ser levadas em considerações para então definir o tipo e segurança entre os serviços. Podem existir implementações de segurança que percorrem todos os microsserviços ou até mais centralizado e de maneira previamente definida entre determinados serviços, a definição disso resultaria em mais tempo necessário para a prototipação.

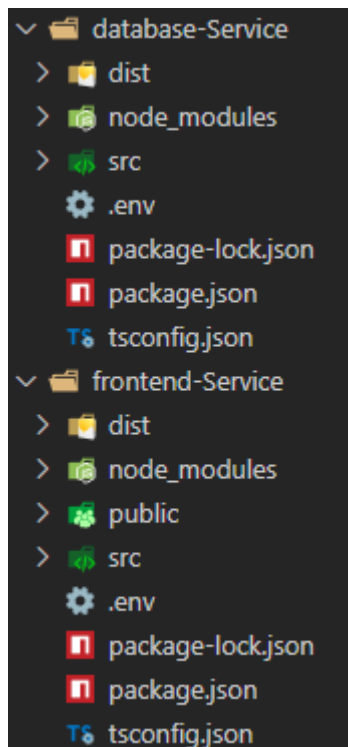


Figura 3 Visão inicial de ambos serviços. autoria própria.

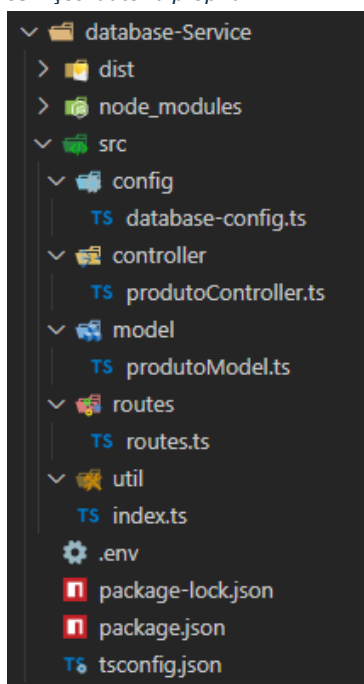


Figura 4 visão do serviço database. autoria própria.

Ambos serviços compartilham estruturas parecidas, mas com focos distintos. Ambos utilizam Typescript, um Javascript aprimorado, que necessita ser compilado antes de executado e um arquivo tsconfig.json de configuração. O arquivo .env é responsável por manter configurações a ser utilizadas no serviço, e que podem ser utilizadas em qualquer parte da aplicação, necessita a importação da biblioteca dotenv. Package.json são as instruções para que o Node compile para dentro da pasta dist os arquivos finais. Nota-se que no frontend-Service existe a pasta Public, ela é responsável por armazenar os arquivos estáticos, como HTML, CSS ou até Javascript que serão enviados para o navegador do usuário através da lógica dentro do src. Ambas também possuem node\_modules, responsável por armazenar todos os arquivos de bibliotecas instaladas.

Toda a lógica principal fica armazenada na pasta src. Dentro da mesma existem uma hierarquia de pastas para uma melhor organização.

Config é responsável por armazenar as configurações de acesso ao banco de dados, além de armazenar uma instância dessa conexão para ser usada posteriormente, evitando que cada nova requisição crie um novo acesso ao banco, evitando problemas no desempenho.

Controller é responsável pela camada que faz a conexão com e configuração com o acesso externo (API).

Model é onde armazenamos classes que servem de modelos para as entidades a serem inseridas ou manipuladas no banco de dados.

Routes é onde armazenamos a lógica de roteamento das endpoints. Uma aplicação localhost definida na porta 8080

roda em "localhost:8080" os endpoints são acessos além dela, como "localhost:8080/login", o "/login" é definido e tem sua lógica em routes.

Util é para lógicas auxiliares da aplicação, nesse caso não houve necessidade de uso e está vazia. Como essa estrutura segue um padrão de projeto, resolvi deixa-la para exemplo.

O index.ts está dentro da pasta src, junto com as demais pastas mencionadas acima, esse arquivo é o main da aplicação, o ponto inicial, nele é onde se aplica a lógica de iniciação da aplicação.

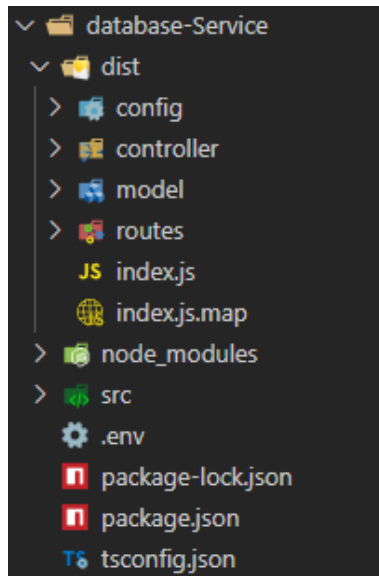


Figura 5 visão da pasta dist. autoria própria.

É importante notar que a pasta dist é onde é armazenado os arquivos convertidos/compilados pelo Node. Todos os arquivos Typescript passam ser Javascript. O Typescript apenas auxilia com várias logicas da programação não presente no Javascript padrão, como a tipagem, interfaces e outros, depois de processado passa a ser um Javascript padrão mas com todas as assertividades antes desenvolvidas com Typescript.

O comando padrão definido no package.json para fazer essa transformação é “npm run build”, dentro do diretório onde está package.json e pode ser modificado nesse mesmo arquivo.

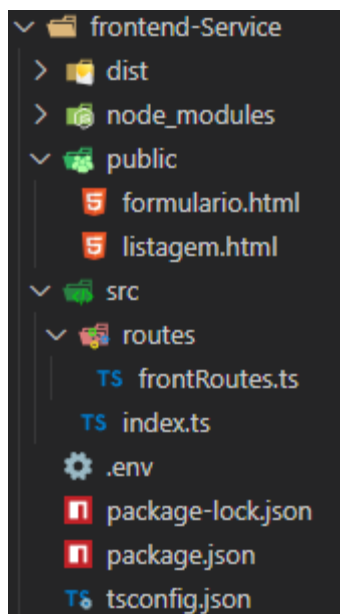


Figura 6 visão do serviço front-end autoria própria

No serviço front-end a lógica da organização das pastas é similar, com pequenas alterações e menor. Public, como mencionado antes, armazena todos os arquivos estáticos que podem ser usados e retornados ao usuário que acessa pela lógica em src, todos os scripts e css estão dentro do HTML por uma questão de facilidade, mas poderiam estar em um arquivo separado e específico.

Dentro de src, o diretório principal e da lógica da aplicação, podem existir outras com funções de organização e que segue boas práticas já consolidadas por desenvolvedores, como visto no database-Service. Nesse caso apenas routes está presente, já que é uma lógica simples.



```

frontend-Service > src > TS index.ts > ...
1  import express from "express";
2  import frontRoutes from "../routes/frontRoutes.js";
3  import cors from "cors";
4  import { fileURLToPath } from "url";
5  import { dirname, join } from "path";
6
7  const __filename = fileURLToPath(import.meta.url);
8  const __dirname = dirname(__filename);
9
10 const app = express();
11
12 // --- Ordem dos Middlewares (MUITO IMPORTANTE!) ---
13 app.use(cors()); // permite requisições de outras origens
14 app.use(express.json()); // para passar JSON no corpo da requisição POST
15 app.use(express.static(join(__dirname, "..", "public"))); //para servir
16 // arquivos estáticos (CSS, JS do frontend, HTML estático)
17
18 // roteadores personalizados (frontRoutes neste caso)
19 app.use("/", frontRoutes);
20
21 //define a rota
22 const PORT = process.env.PORT || 3333;
23 app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
24 // app.listen usa o express na porta definida
25

```

Figura 7 index.ts do frontend-Service. autoria própria

num mesmo URL “localhost”, e por política padrão os navegadores implementam uma série de segurança que bloqueiam isso.

As rotas são definidas por outro arquivo chamado frontRoutes, como mostrado no import no topo do código, ele é utilizado no app.use que vem do Express e define e redirecionam a lógica dos endpoint, como “/formulário” e “/listagem”.

```

database-Service > src > TS index.ts > ...
1  import express from "express";
2  import db from "../config/database-config.js";
3  import routes from "../routes/routes.js";
4  import cors from "cors";
5
6  const app = express();
7  export const appPort = 3332;
8
9  app.use(
10    cors({
11      origin: "http://localhost:3333",
12    })
13  );
14
15  app.use(express.json());
16
17  app.use(routes);
18
19  app.listen(appPort, () => {
20    console.log(`Servidor rodando em http://localhost:${appPort}`);
21  });
22
23  export const appInstance = app;
24

```

Figura 8 index.ts do database-Service. autoria própria

No index.ts, o arquivo principal do frontend-service tem uma estrutura simples. Ela inicia o servidor Express, define a rota e também executa os middlewares, que são de extrema importância, eles fazem uma série de comunicações e processos entre navegador e servidor. O middleware Cors é necessário necessário nesse caso, pois estamos fazendo requisições

O index.ts do backend-Service segue a mesma lógica, importando e usando Express. Nota-se que ambos utilizam o mesmo URL “localhost” com a distinção da porta, enquanto o front-end roda na porta 3333 o back-end na porta 3332, isso pode ser alterado para qualquer porta preferível.

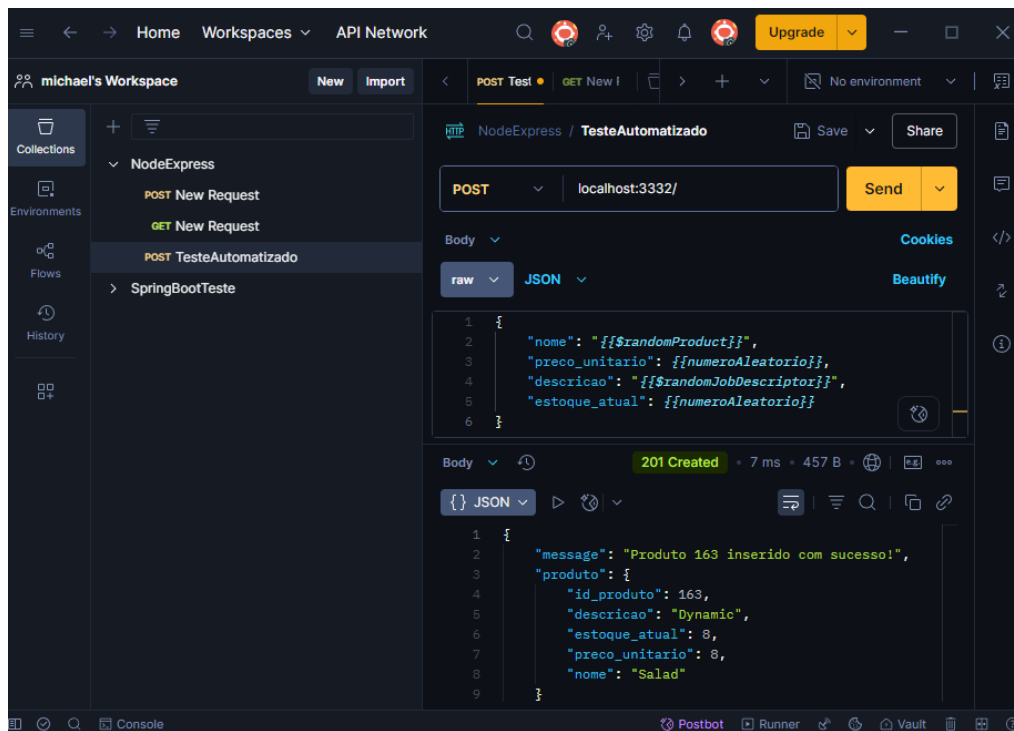


Figura 9 automação de testes em endpoints com postman. autoria própria.

Graças as arquiteturas independentes de ambos serviços, é possível realizar testes diretamente no serviço que faz a comunicação com o banco de dados de maneira isolada. Esse tipo de desacoplamento mostra que outros sistemas ou interfaces servidas para os usuários podem se comunicar com o back-end de forma independente. Nesse caso, foi realizado usando o POSTMAN, um aplicativo para testes de métodos HTTP, passando um corpo em JSON com parâmetros aleatórios gerado a cada requisição.

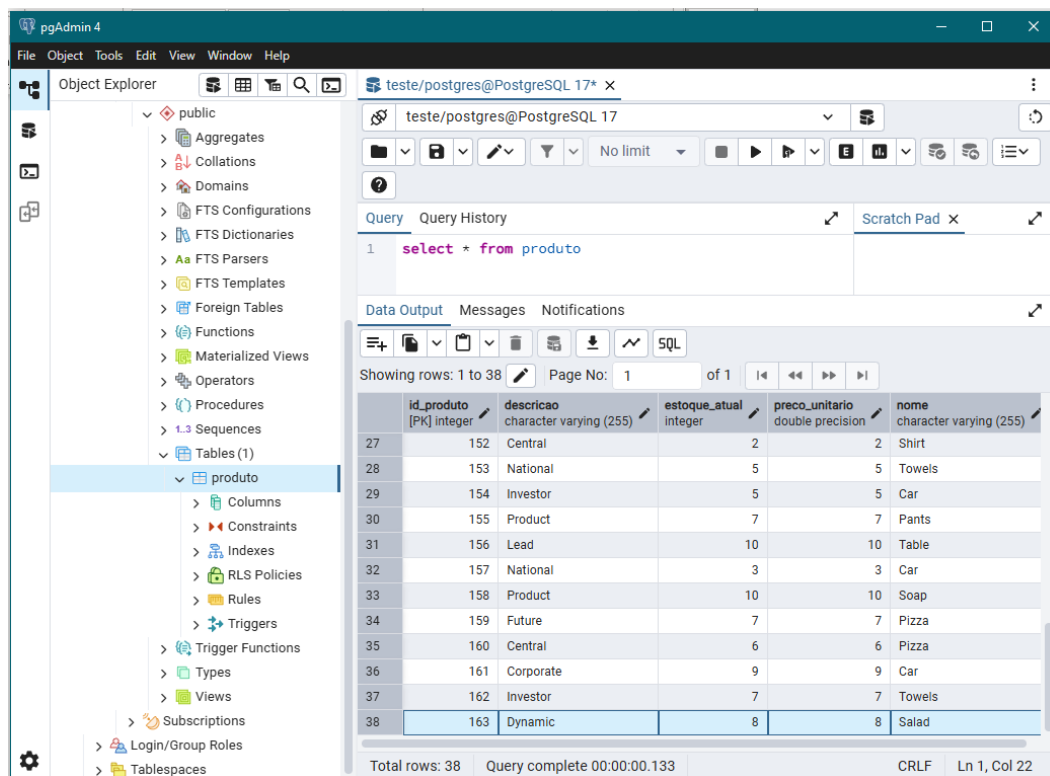


Figura 10 consulta ao banco de dados – nota-se o item inserido na linha 38 através do PostMan. autoria própria.

The screenshot shows a web browser window with the URL 'localhost:3333/formulário'. The page title is 'Cadastro de Produto'. The form contains four input fields: 'Nome do Produto:', 'Descrição:', 'Preço:', and 'Quantidade:'. Below these fields is a green button labeled 'Salvar Produto'.

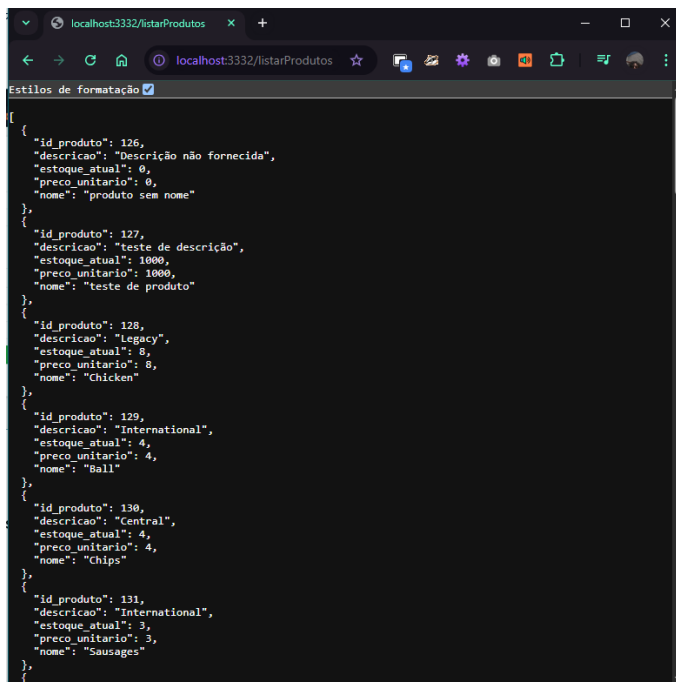
Figura 12 "localhost:3333/formulário" autoria própria.

The screenshot shows a web browser window with the URL 'localhost:3333/listagem'. The page title is 'Lista de Produtos'. The table displays the following data:

Nome	Descrição	Preço Unitário	Estoque Atual
produto sem nome	Descrição não fornecida	R\$ 0.00	0
teste de produto	teste de descrição	R\$ 1000.00	1000
Chicken	Legacy	R\$ 8.00	8
Ball	International	R\$ 4.00	4
Chips	Central	R\$ 4.00	4
Sausages	International	R\$ 3.00	3
Bacon	Internal	R\$ 5.00	5
Soap	Direct	R\$ 9.00	9
Chicken	Forward	R\$ 7.00	7
Bacon	Principal	R\$ 6.00	6
Ball	Central	R\$ 7.00	7
Fish	Central	R\$ 4.00	4
Tuna	National	R\$ 8.00	8
Hat	Dynamic	R\$ 1.00	1
Car	Dynamic	R\$ 8.00	8
Tuna	Principal	R\$ 7.00	7
Hat	Product	R\$ 10.00	10
Ball	Forward	R\$ 3.00	3
Car	International	R\$ 8.00	8
Bike	Corporate	R\$ 4.00	4
Cheese	Legacy	R\$ 2.00	2
Towels	Central	R\$ 8.00	8
Shoes	Corporate	R\$ 10.00	10
Soap	Legacy	R\$ 1.00	1
Computer	Legacy	R\$ 3.00	3
Computer	District	R\$ 3.00	3
Shirt	Central	R\$ 2.00	2

Figura 11 "localhost:3333/listagem" autoria própria.

Ambas páginas possuem lógicas e visual simples, cadastro e listagem. A página de cadastro também informa o status da inserção, caso sucesso ou erro.

A screenshot of a web browser window with the address bar showing 'localhost:3332/listarProdutos'. The browser's developer tools are open, displaying a JSON array of product objects. The JSON is formatted with syntax highlighting. The array contains seven objects, each with fields for 'id\_produto', 'descricao', 'estoque\_atual', 'preco\_unitario', and 'nome'.

```
[{"id_produto": 126, "descricao": "Descrição não fornecida", "estoque_atual": 0, "preco_unitario": 0, "nome": "produto sem nome"}, {"id_produto": 127, "descricao": "teste de descrição", "estoque_atual": 1000, "preco_unitario": 1000, "nome": "teste de produto"}, {"id_produto": 128, "descricao": "Legacy", "estoque_atual": 8, "preco_unitario": 8, "nome": "Chicken"}, {"id_produto": 129, "descricao": "International", "estoque_atual": 4, "preco_unitario": 4, "nome": "Ball"}, {"id_produto": 130, "descricao": "Central", "estoque_atual": 4, "preco_unitario": 4, "nome": "Chips"}, {"id_produto": 131, "descricao": "International", "estoque_atual": 3, "preco_unitario": 3, "nome": "Sausages"}]
```

Figura 13 "localhost:3332/listarProdutos" 'autoria própria

O back-end fornece uma endpoint para retorno dos produtos, nota-se que ela roda em "localhost:3332/listarProdutos", retornando um corpo em JSON que também pode ser visto na maioria dos navegadores.

Outros métodos também podem ser desenvolvidos e implantados nessa arquitetura, como métodos DELETE e UPDATE, que estão ausentes apenas com o PUT e GET.

Todos os códigos foram também implementados pensando em um código mais funcional, segundo Jack Widman(Aprenda Programação Funcional, 2024) afirma em seu livro "A programação funcional (PF) é um paradigma ou uma abordagem à programação, um modo de fracionar o mundo e montá-lo novamente em forma de códigos. Ela envolve a maneira como organizamos essa peça do mundo que estamos modelando e como organizamos e estruturamos o código.". Tal ideia foi implementada organizando um código mais conciso, direto e funcional, como mostrado nos principais arquivos das aplicações.

Além disso, aspectos de limpeza e organização foram levadas em consideração, Robert C. Martin(Código Limpo, 2009) em seu livro menciona "[...] desejamos que a leitura do código seja fácil, mesmo se sua criação for árdua. É claro que não há como escrever um código sem lê-lo, portando torná-lo de fácil leitura realmente facilita a escrita.". Outro aspecto que pode ser favorecido e também é mencionado no livro são as manutenções futuras, como também mencionado "Conforme a confusão aumenta, a produtividade da equipe diminui, assintoticamente aproximando-se do zero.". Tais afirmações como essas precisam ser levadas em consideração em um cenário real.

## **METODOLOGIA**

A presente pesquisa/desenvolvimento caracteriza-se pela abordagem qualitativa, por buscar aprofundar a compreensão sobre conceitos e aplicações que foram abordados, sem o objeto de quantificar dados ou realizar generalizações estatísticas.

Quando ao tipo de pesquisa, este trabalho configura-se como uma pesquisa bibliográfica. O levantamento e a análise dos dados foram realizados a partir do estudo aprofundado de obras publicadas por autores reconhecidos e especialistas nas áreas de Domain-Drive Design, Microserviços, Código Limpo e Programação Funcional, conforma detalhado na seção de Referências Bibliográficas.

Os procedimentos metodológicos envolveram a leitura exploratória, seletiva e analítica do material apresentado e usado como base, com o objetivo de identificar conceitos fundamentais, princípios e aplicações práticas relevantes para o desenvolvimento do trabalho. Além da compressão teoria, foram realizadas análises de exemplo e implementações descritos na literatura, buscando correlacionar os conhecimentos adquiridos e avaliar sua aplicabilidade em contexto de desenvolvimento de software.

Dessa forma, a construção dos resultados se deu pela síntese e interpretação crítica das informações coletadas, permitindo a estruturação de uma visão consolidada sobre os temas propostos.

## **CONSIDERAÇÕES**

Lidar com aplicações extensas num cenário real vai muito além de apenas aprender uma determinada linguagem de programação, também envolve entender a arquitetura do projeto, limpeza e clareza do mesmo, além das regras de negócios que precisam se encaixar e ser de uma linguagem clara e ubíqua, para que não haja erros e eventuais gargalos no desenvolvimento.

O trabalho me fez relacionar temas diferentes, como Microserviços e DDD, que estão muito pertos de casos de aplicações reais. Tais temas ajudaram não só pensar de maneira mais robusta, mas também de maneira moderna e escalável. Através dessa pesquisa foi possível consolidar o entendimento sobre a relevância das tais abordagens.

Em suma, esta experiência de pesquisa não só ampliou o repertório técnico e teórico, mas também solidificou a compreensão da interdependência entre arquitetura, design de código e estratégia de negócios. O aprendizado adquirido é de valor inestimável e de grande valor para meu aspecto profissional, reforço a importância da busca contínua por excelência e inovação utilizando casos reais e grandes fontes e especialistas no campo do desenvolvimento de software.

## REFERÊNCIAS BIBLIOGRÁFICAS

KHONONOV, Vlad. **Aprenda Domain-Driven Design: Alinhando Arquitetura de Software e Estratégia de Negócios**. [S.l.]: O'Reilly; Alta Books, 2024.

MARTIN, Robert C. **Código Limpo: Habilidades práticas do Agile Software**. [S.l.]: Prentice Hall; Alta Books, 2009.

MELLO, Michael. **Aplicação Comercial com microsserviços**. [S.l.]: GitHub, 2025. Versão 1.0. Disponível em:  
<https://github.com/DrakoMichael/AplicacaoComercialComMicroservicosUniasselv>;  
[https://drive.google.com/drive/folders/1rwyLWBBkv2nOMCAVIBNnlfMsVY4T7JRp?usp=drive\\_link](https://drive.google.com/drive/folders/1rwyLWBBkv2nOMCAVIBNnlfMsVY4T7JRp?usp=drive_link). Acesso em: 21 jun. 2025.

NEWMAN, Sam. **Criando Microsserviços: Projetando sistemas com componentes menores e mais especializados**. [S.l.]: O'Reilly; Novatec, 2022.

WIDMAN, Jack. **Aprenda Programação Funcional: Como pensar funcionalmente para trabalhar com códigos complexos**. [S.l.]: O'Reilly; Alta Books, 2024.