

# || PROVA DE CONCEITO

## 1. Título

Uso do padrão “Branch por abstração” para decomposição de arquitetura monolítica em microsserviços.

## 2. Objetivo

Realizar a decomposição de parte de nosso sistema monolítico em microsserviços usando o padrão de referência “Branch por abstração” sem que os serviços sejam interrompidos durante a migração de arquitetura.

## 3. Escopo

O protótipo criado nessa prova de conceito irá usar o padrão de decomposição com o objetivo migrar o módulo de vendas do sistema monolítico para microsserviços, onde estão inclusas as funções de pedidos, clientes e produtos.

## 4. Requisitos

### Funcionais:

- Realizar pedidos;
- Cadastrar clientes;
- Cadastrar produtos;

### Não funcionais:

- Os serviços devem permanecer em funcionamento durante a migração de arquitetura.
- Os contratos com os clientes não podem ser alterados.
- O desempenho deve ser similar ou melhor.

## 5. Metodologia

- Foi utilizado metodologia **ágil** para desenvolvimento da prova de conceito com sprints semanais e incrementais.
- Para a execução as seguintes ferramentas foram usadas:
  - **Trello** para gerenciamento das tarefas;
  - **Github** para versionamento de código fonte;
  - **.Net Framework 4.8, Framework .Net 8 e Visual Studio** para desenvolvimento.
  - **Swagger e Postman** para realização de testes e documentação.

## 6. Cronograma

As tarefas foram distribuídas em 3 sprints de duas semanas cada totalizando 45 dias. As atividades ficaram divididas da seguinte forma:

- **Sprint 01:** Planejamento e desenvolvimento do protótipo;
- **Sprint 02:** Desenvolvimento do protótipo, execução de testes e análise dos resultados;
- **Sprint 03:** Documentação da prova de conceito;

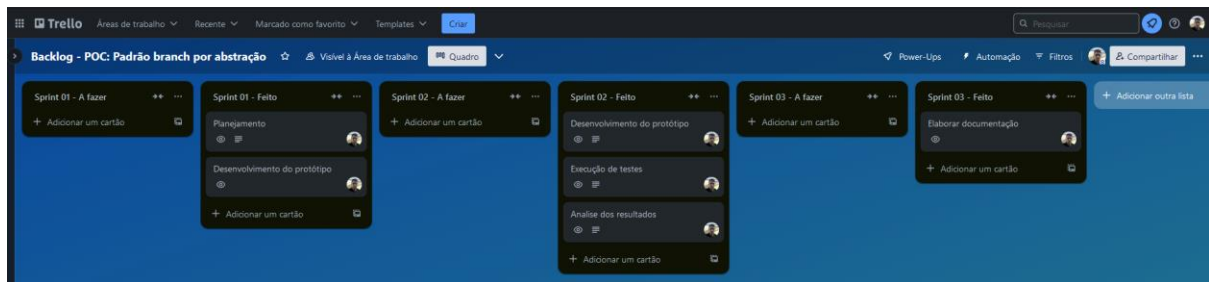


Figura 1 - Sprints catalogadas no Trello.

## 7. Recursos

- **Humanos:** 1 desenvolvedor backend;
- **Tecnológicos:** Ferramenta para desenvolvimento e versionamento, acesso ao código fonte monolítico.
- **Financeiro:** Horas de trabalho.

## 8. Critérios de sucesso

Os critérios abaixo balizam o sucesso da prova de conceito:


- As funcionalidades devem estar 100% migradas para arquitetura de microsserviços;
- Os serviços devem estar em pleno funcionamento (pedidos, clientes e produtos);
- Os contratos com os usuários devem ser mantidos;
- Deve ocorrer pouca ou nenhuma instabilidade/interferência durante a migração dos serviços;

## 9. Critérios x resultados

Após a implementação do padrão de decomposição “Branch por abstração” os resultados esperados foram obtidos. A seguir demonstro cada um deles:

### Funcionalidades migradas para arquitetura de microsserviços:

Inicialmente tínhamos um sistema monolítico onde os serviços de pedidos, clientes e produtos funcionavam juntos:




Explore

## APIs Monolito

### Cliente

Show/Hide | List Operations | Expand Operations

- GET /api/v1/cliente/{codigo}
- DELETE /api/v1/cliente
- GET /api/v1/cliente
- POST /api/v1/cliente
- PUT /api/v1/cliente

### Pedido

Show/Hide | List Operations | Expand Operations

- GET /api/v1/pedido/{codigo}
- DELETE /api/Pedido
- GET /api/Pedido
- POST /api/Pedido
- PUT /api/Pedido

### Produto


Show/Hide | List Operations | Expand Operations

- GET /api/v1/produto/{codigo}
- DELETE /api/v1/produto
- GET /api/v1/produto
- POST /api/v1/produto
- PUT /api/v1/produto

[ BASE URL: , API VERSION: v1 ]

Figura 2 - Sistema monolítico

Após a migração temos cada um dos serviços independentes:


Select a definition
XPe.PrjAplicado.Microservico.Clientes.API v1

## XPe.PrjAplicado.Microservico.Clientes.API 1.0 OA53

<https://localhost:7056/swagger/v1/swagger.json>

### Cliente

^

- GET /api/v1/cliente/{codigo} ▾
- GET /api/v1/cliente ▾
- POST /api/v1/cliente ▾
- PUT /api/v1/cliente ▾
- DELETE /api/v1/cliente ▾

### Schemas

^

- Cliente >

Figura 3 - Microserviço de clientes

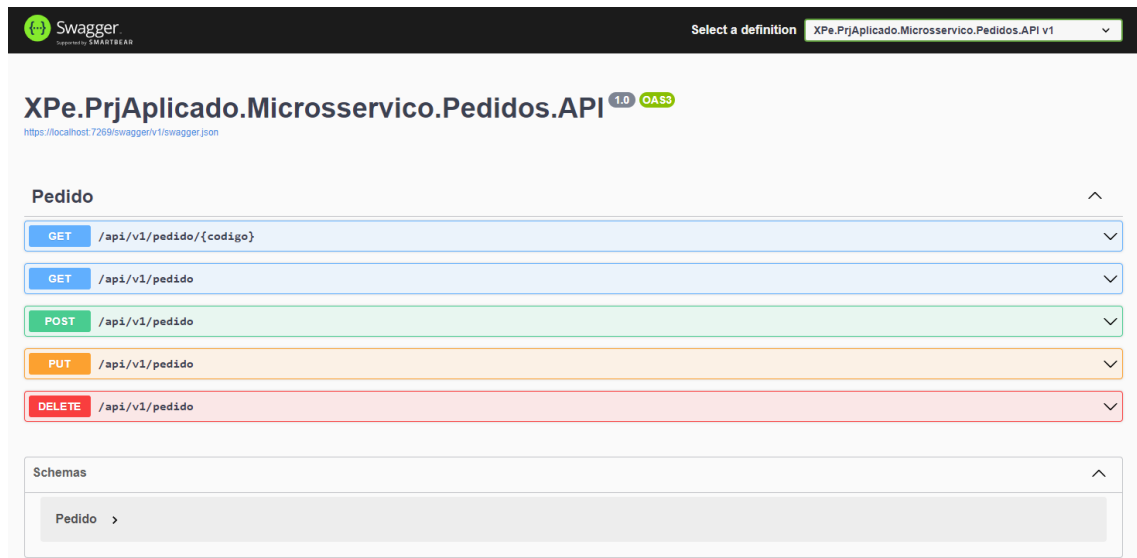


Figura 4 - Microserviço de pedidos

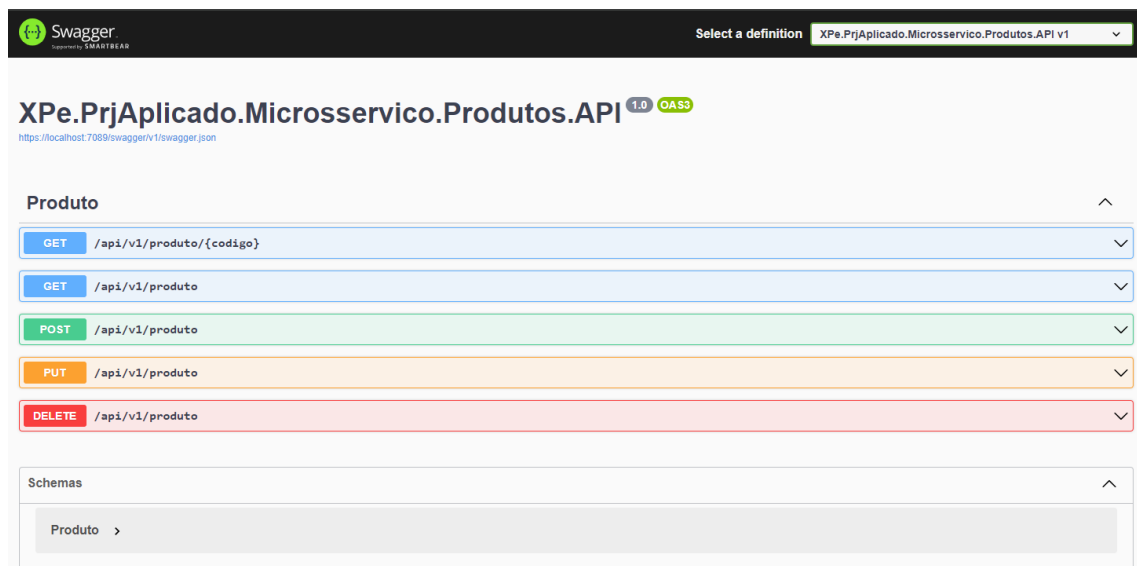
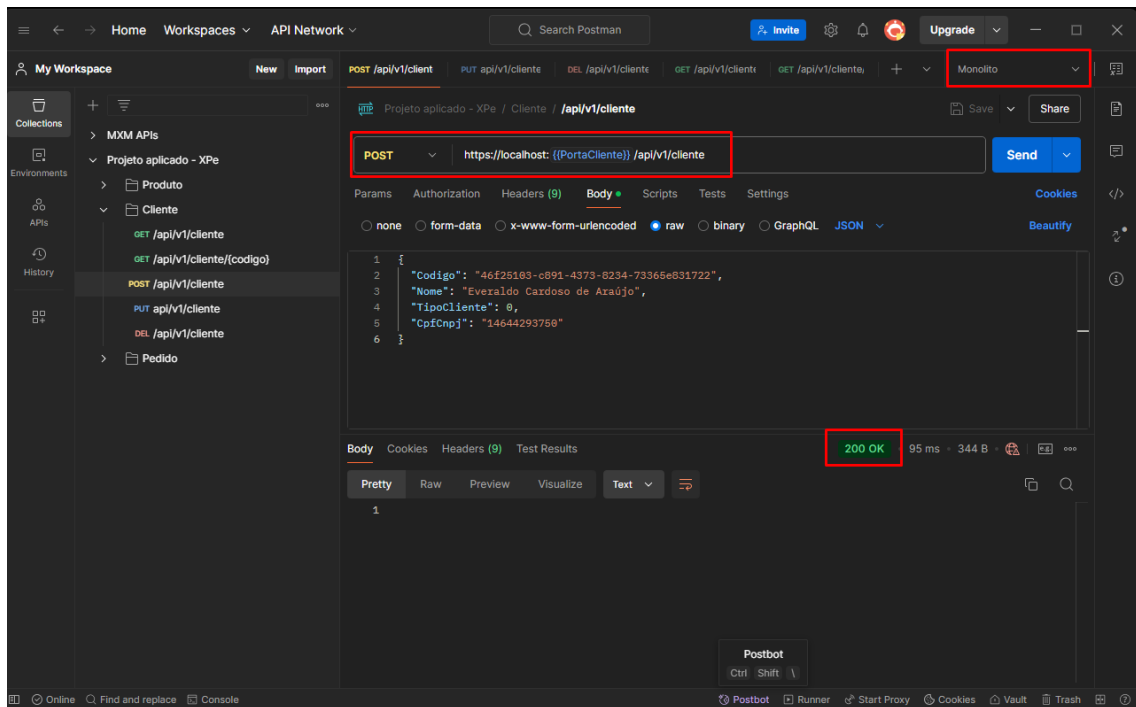


Figura 5 - Microserviço de produtos

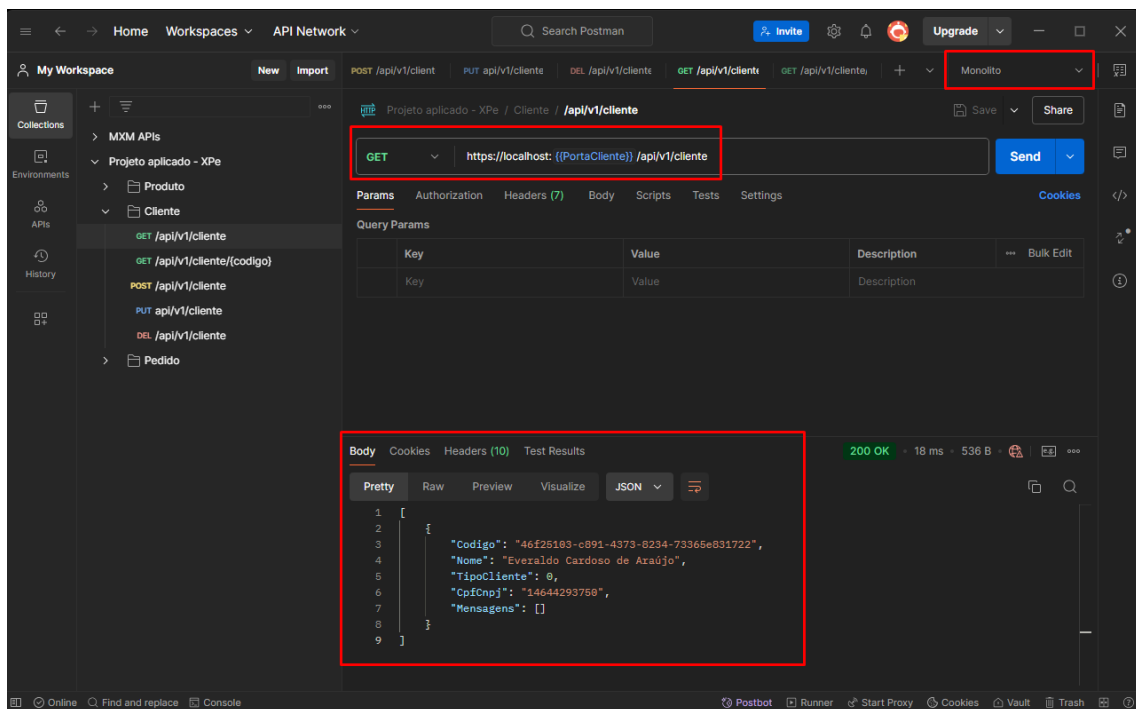
### Serviços em pleno funcionamento (pedidos, clientes e produtos):

Durante e após a migração os serviços se mantiveram em funcionamento sem interrupções. A seguir temos os testes de funcionamento:

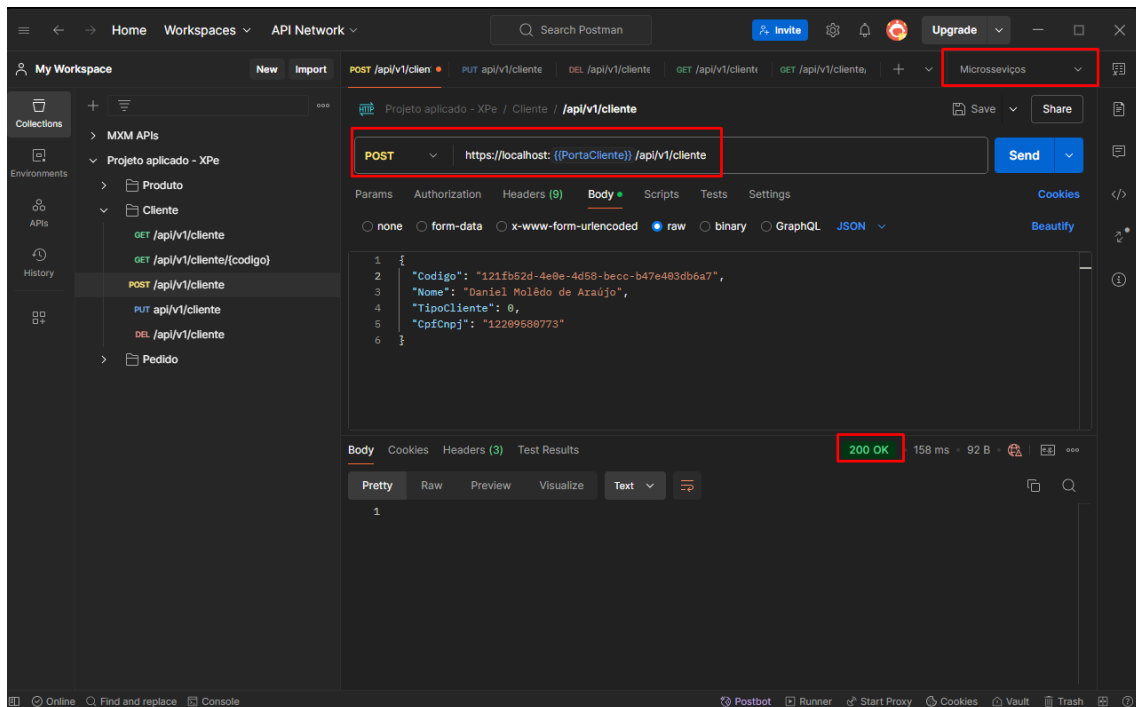
**Teste 01:** Criando um cliente com o método [POST] - [/api/cliente](#) apontando para o serviço monolítico:



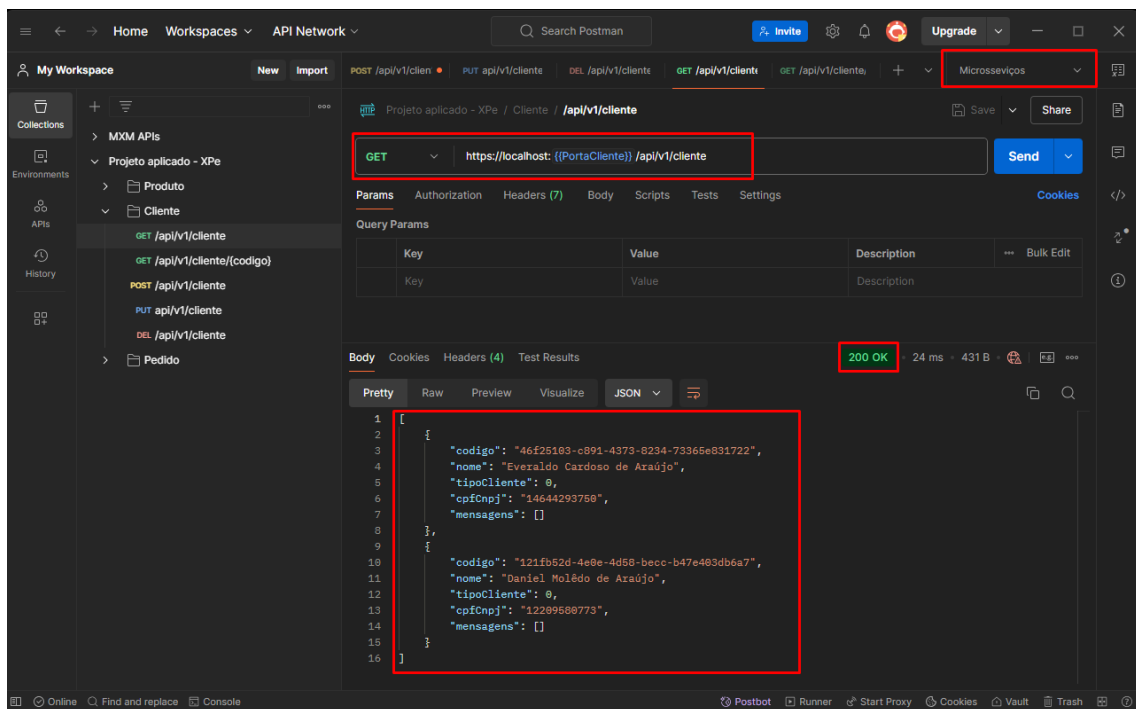
**Teste 02:** Obtendo a lista de todos os clientes com o método [GET] - [/api/cliente](#) apontando para o serviço monolítico:



**Teste 03:** Criando um cliente com o método [POST] - [/api/cliente](#) apontando para o microserviço de clientes:



**Teste 04:** Obtendo a lista de todos os clientes com o método [GET] - [/api/cliente](#) apontando para o microserviço de clientes:



Neste último teste é possível verificar que ao realizar a consulta é retornado dois clientes sendo que o primeiro foi cadastrado pelo serviço monolítico e o segundo pelo microserviço de clientes.

**Teste 05:** Obtendo cliente específico pelo método [GET] - [/api/cliente/{{codigo}}](#) em ambos os serviços:

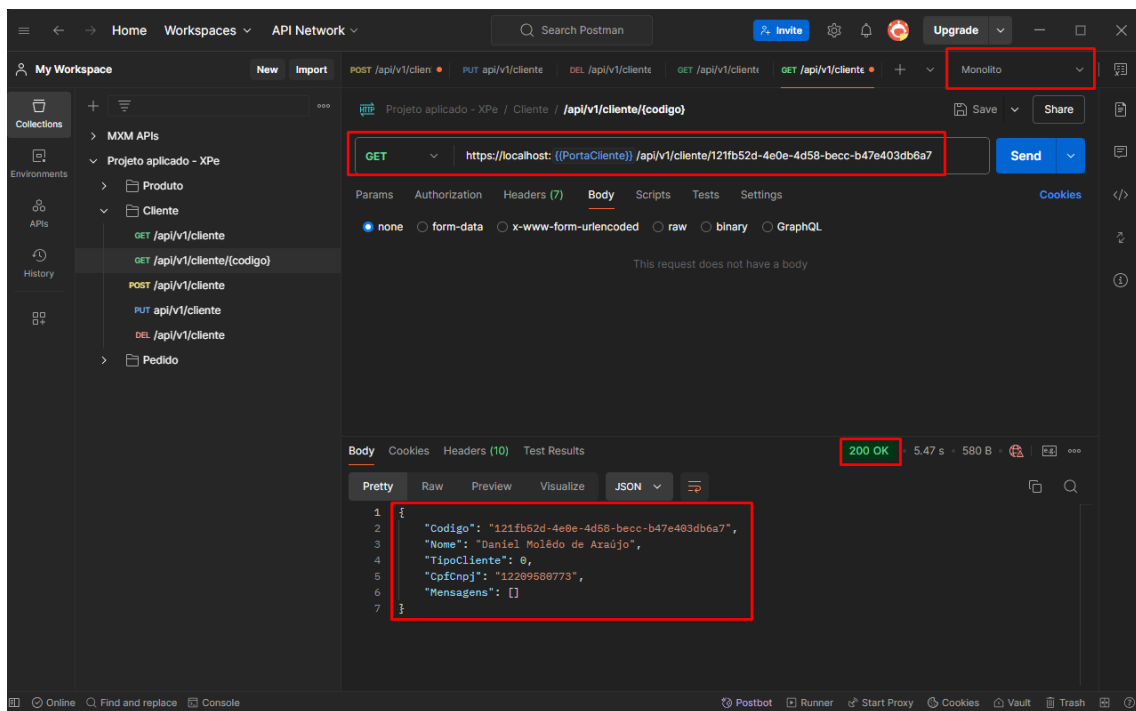


Figura 6 - Obtendo cliente pelo sistema monolítico

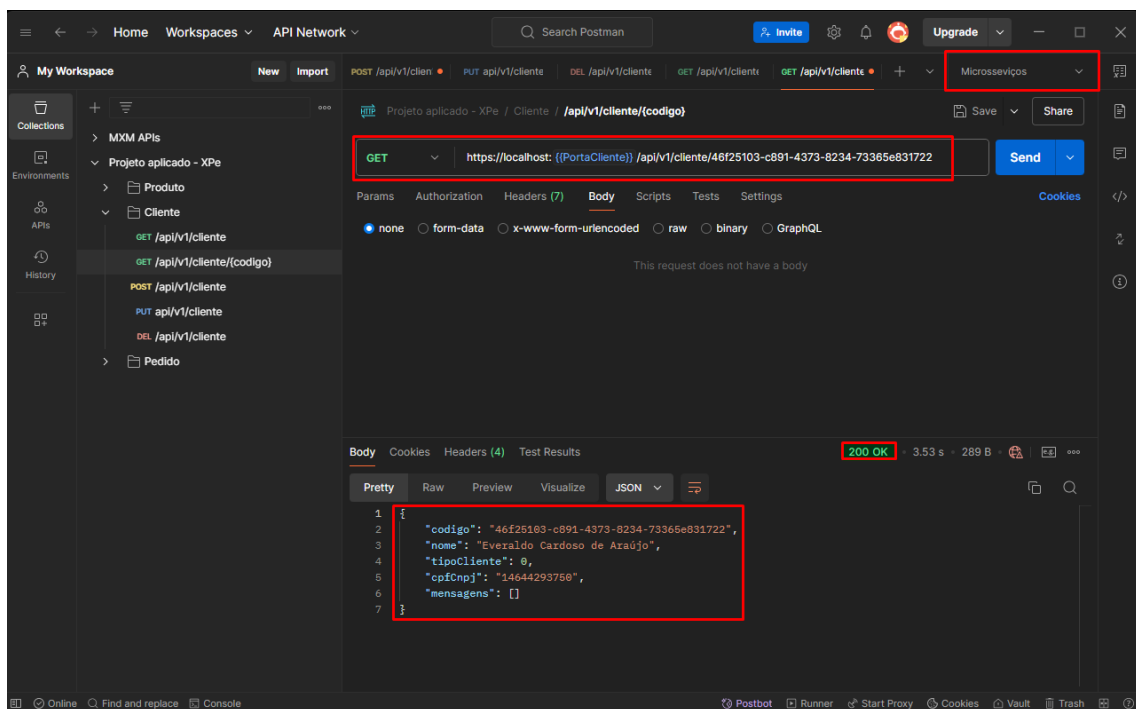


Figura 7 - Obtendo o cliente pelo microserviço de clientes

**Teste 06:** Atualizando cliente com o método [PUT] - [/api/cliente](#) apontando para o microserviço de clientes e consultando atualização pelo serviço monolítico:

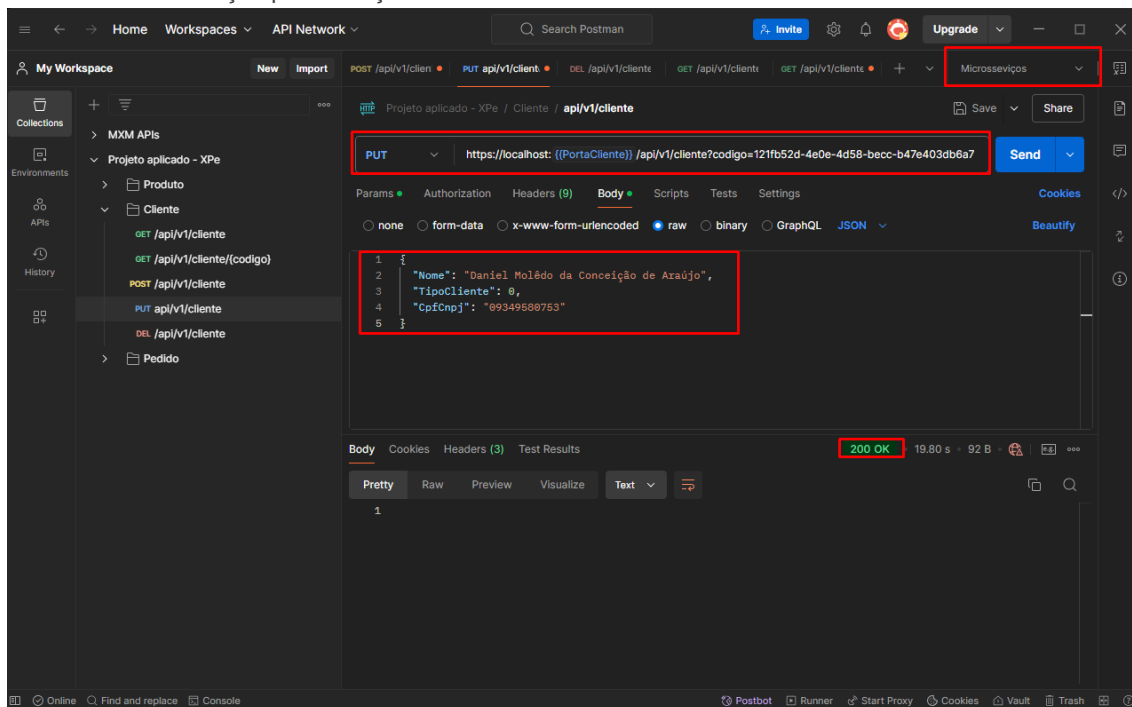


Figura 8 - Atualizando cliente

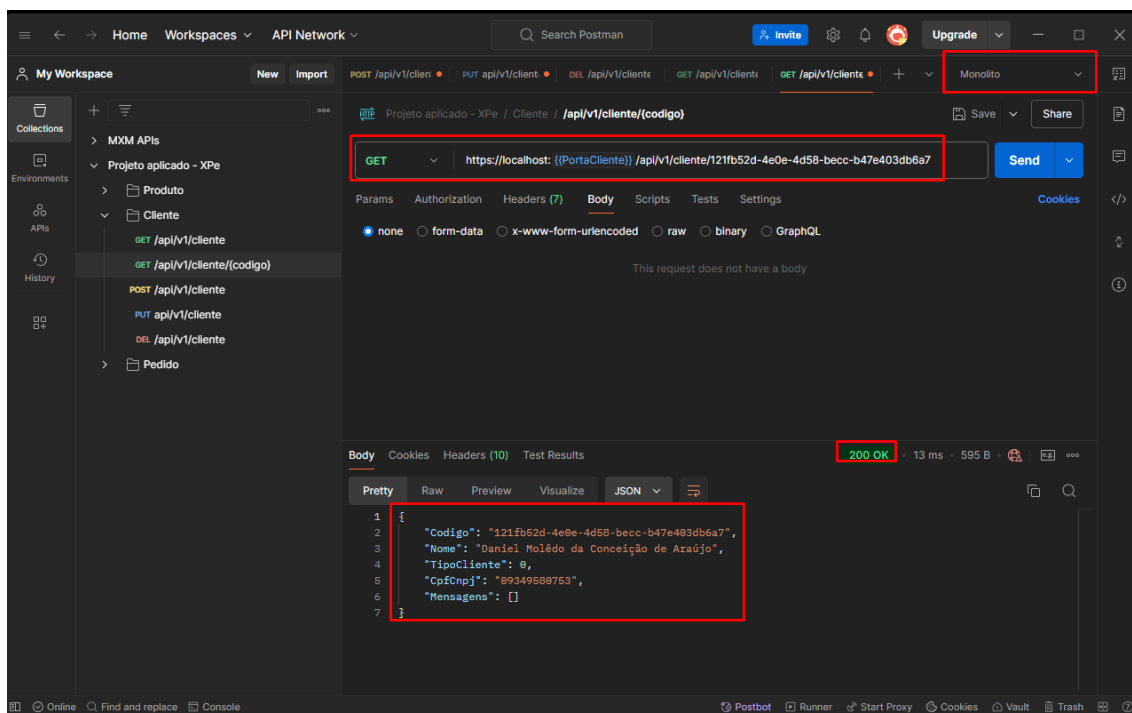
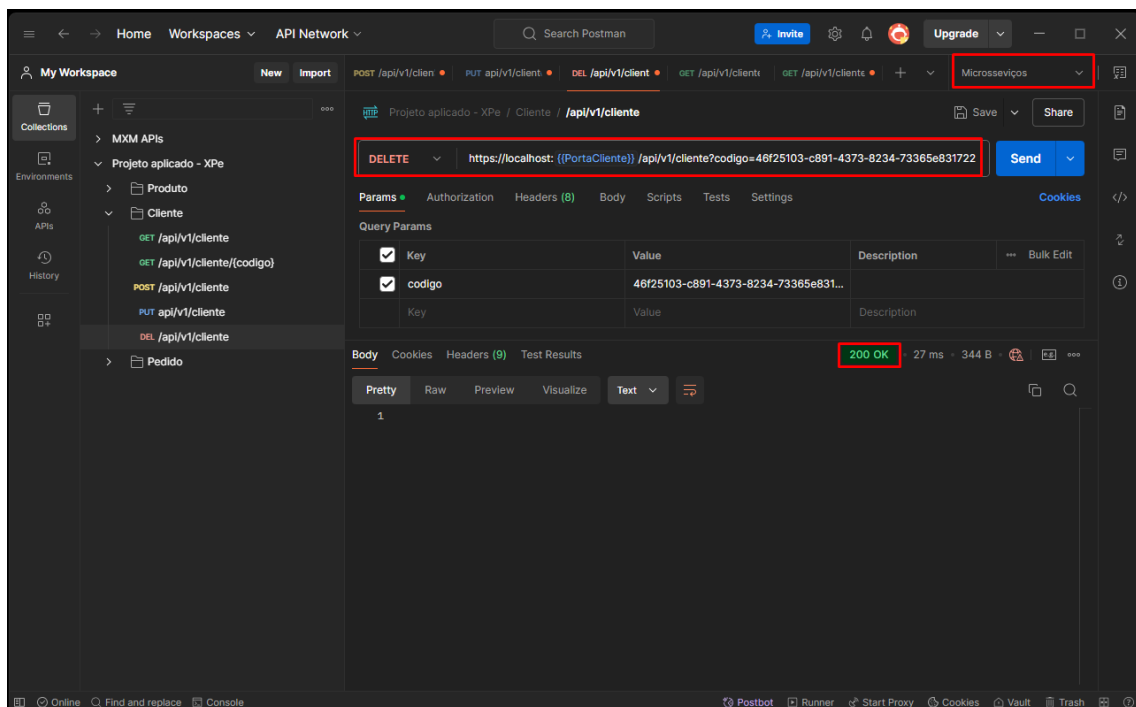


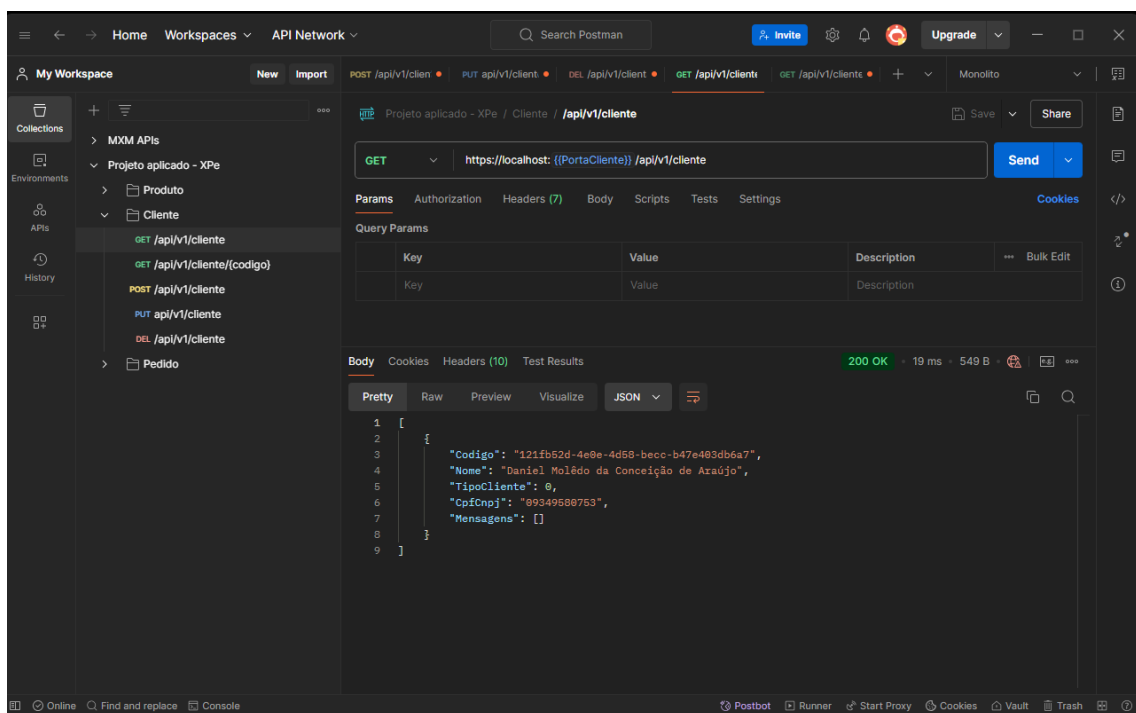
Figura 9 - Cliente atualizado

**Teste 07:** Apagando dados de um dos clientes pelo método [DELETE] - [/api/cliente](#) apontando para o microserviço de cliente:





Teste 08: Consultando novamente a lista de clientes para verificar se a exclusão funcionou, usando o método [GET] - [/api/cliente](#) do sistema monolítico:



Ao fim de todos os testes é possível perceber que apesar dos ambientes distintos o comportamento dos contratos é o mesmo, conforme esperado pelo uso do padrão. Dessa forma as aplicações consumidoras não sentem o impacto da mudança.

### Os contratos com os usuários devem ser mantidos:

Todos os contratos foram mantidos, assegurando que nenhuma modificação fosse percebida pelos usuários:

- Contratos do sistema em arquitetura monolítica x microsserviços respectivamente:
  - Clientes:

## Cliente

Show/Hide | List Operations | Expand Operations

GET	/api/v1/cliente/{codigo}
DELETE	/api/v1/cliente
GET	/api/v1/cliente
POST	/api/v1/cliente
PUT	/api/v1/cliente

Cliente ^	
GET	/api/v1/cliente/{codigo} v
GET	/api/v1/cliente v
POST	/api/v1/cliente v
PUT	/api/v1/cliente v
DELETE	/api/v1/cliente v

○ Pedido:

## Pedido

Show/Hide | List Operations | Expand Operations

GET	/api/v1/pedido/{codigo}
DELETE	/api/v1/pedido
GET	/api/v1/pedido
POST	/api/v1/pedido
PUT	/api/v1/pedido

Pedido ^	
GET	/api/v1/pedido/{codigo} v
GET	/api/v1/pedido v
POST	/api/v1/pedido v
PUT	/api/v1/pedido v
DELETE	/api/v1/pedido v

○ Produto:

## Produto

Show/Hide | List Operations | Expand Operations

GET	/api/v1/produto/{codigo}
DELETE	/api/v1/produto
GET	/api/v1/produto
POST	/api/v1/produto
PUT	/api/v1/produto

Produto		^
GET	/api/v1/produto/{codigo}	▼
GET	/api/v1/produto	▼
POST	/api/v1/produto	▼
PUT	/api/v1/produto	▼
DELETE	/api/v1/produto	▼

Para tornar esse processo possível foi necessário criar uma interface comum entre os serviços como garantia da manutenção do padrão:

```

namespace XPe.PrjAplicado.Interop.Controllers
{
    public interface IMigrationController<TipoRetorno, TipoEntidade, TipoChave>
    {
        TipoRetorno Get(TipoChave codigo);

        TipoRetorno Get();

        6 references | 0 changes | 0 authors, 0 changes
        TipoRetorno Post(TipoEntidade entidade);

        6 references | 0 changes | 0 authors, 0 changes
        TipoRetorno Put(TipoChave codigo, TipoEntidade entidade);

        6 references | 0 changes | 0 authors, 0 changes
        TipoRetorno Delete(TipoChave codigo);
    }
}

```

Figura 10 - Interface que garante o contrato com os clientes.

```

namespace XPe.PrjAplicado.Monolito.API.Controllers
{
    [Route("api/v1/cliente")]
    public class ClienteController : ApiControllerBase, IMigrationController<IHttpActionResult, Cliente, Guid>
    {
        // ...
    }
}

```

Figura 11 - Exemplo de implementação da interface no sistema monolítico.

```

namespace XPe.PrjAplicado.Microservico.Clientes.API.Controllers
{
    [Route("api/v1/cliente")]
    public class ClienteController : ApiControllerBase, IMigrationController<IActionResult, Cliente, Guid>
    {
        // ...
    }
}

```

Figura 12 - Exemplo de implementação da interface no sistema em microserviço.

### Deve ocorrer pouca ou nenhuma instabilidade/interferência durante a migração dos serviços:

Nenhuma instabilidade ou interferência dos serviços foi relatada pelos usuários da aplicação durante e posteriormente a implementação da migração dos serviços de pedidos, clientes e produtos.

## 10. Conclusão

Após todo o processo de análise, prototipação, testes e documentação desta POC, a conclusão a que se chega é que o padrão de decomposição “branch por abstração” cumpriu muito bem a necessidade imposta de realizar a migração de arquitetura de artefatos pré-existentes em nosso sistema monolítico para a arquitetura de microsserviços, sem que ficasse perceptível aos usuários as mudanças em andamento.

É claro que a migração não é simples pois tem percalços no caminho. Para que o resultado fosse atingido foi necessário criamos, por exemplo, um projeto de compartilhamento de recursos em .Net Standard 2.0, que provê a compatibilidade entre parte do software em .Net Framework 4.8 e .Net 8. Fora isso foi necessário realizar modificações no sistema legado para garantir a eficácia do padrão, tornando os controllers das APIs compatíveis com a interface que garante o contrato com os usuários.

Além da migração inicial muito a de se pensar no futuro. Questões como banco de dados compartilhado entre os serviços ainda precisam ser levantadas e estudadas, algo que não é objetivo deste documento.

A conclusão a que se chega é que o padrão pode ser adotado pela equipe de desenvolvedores e arquitetos como base para a migração de serviços futuros, pois atende a necessidade da equipe e norteia o desenvolvimento do produto com base nas diretrizes impostas pela empresa.

## 11. Referencias

Newman, Sam. *Migrando sistemas monolíticos para microsserviços: Padrões evolutivos para transformar seu sistema monolítico*. 1. ed. São Paulo: Novatec Editora Ltda, 2020.