Gabriela Vieira dos Santos Vaz – BP3044513
Jonas Ribeiro da Rosa – BP304422X
Vinícius Pereira Costa – BP3044289
Iago Felipe Steigleder Bacci – BP3044246

Projeto Back-end & Projeto de Componentes

Trabalho apresentado ao Curso de Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo Campus Bragança Paulista, como requisito das disciplinas de Desenvolvimento de Componentes e Desenvolvimento Web Back-End sob orientação do Prof. Luiz Gustavo Diniz de Oliveira Véras.

Novembro, 2025

Projeto Back-end

Documento de Especificação da API

1. INFORMAÇÕES GERAIS

1.1 Nome da Aplicação

Sistema de Gerenciamento de Livraria (Bookstore Platform)

1.2 Integrantes do Grupo

- Gabriela Vieira dos Santos Vaz – BP3044513
- Jonas Ribeiro da Rosa – BP304422X
- Vinícius Pereira Costa – BP3044289
- Iago Felipe Steigleder Bacci – BP3044246

2. TEMA DO PROJETO

2.1 Descrição do Tema

O projeto consiste no desenvolvimento de uma API REST para gerenciamento de uma livraria, permitindo o controle e administração de livros, editoras, clientes e pedidos realizados pelos usuários.

O sistema foi planejado para simular um ambiente real de e-commerce de livraria, sendo capaz de:

- Cadastrar clientes
- Cadastrar editoras e livros
- Associar livros a autores e categorias
- Criar pedidos de compra
- Gerenciar itens dentro de pedidos
- Consultar informações por meio de endpoints REST
- Aplicar arquitetura baseada em componentes e boas práticas de projeto

3. ARQUITETURA DO SISTEMA

3.1 Tipo de Arquitetura

O sistema foi construído seguindo os princípios de Clean Architecture e Domain-Driven Design (DDD), dividido em módulos independentes (Componentes) que são integrados pela aplicação principal.

```
bookstore-platform/
├── book-domain/          # Componente de Gestão de Livros (Catálogo/Estoque)
├── customer-domain/      # Componente de Gestão de Clientes
├── order-domain/         # Componente de Gestão de Pedidos (Core Business)
├── common-domain/        # Componente de Infraestrutura e Serviços Compartilhados (Frete/Email)
├── api-rest/             # Aplicação Spring Boot (API Gateway/Controller)
└── pom.xml               # Parent POM (Gerenciamento de Dependências)
```

3.2 Tecnologias Utilizadas

- **Linguagem:** Java 17
- **Framework:** Spring Boot 3.1.5
- **Persistência:** Spring Data JPA / Hibernate
- **Banco de Dados:**
    - Produção: MySQL 8.0 (Driver 8.4.0)
    - Testes: H2 Database (Em memória)
- **Serviços e Integrações Externas:**
    - ViaCEP: Utilizado para consulta de endereços e cálculo lógico de frete por região.
    - MailHog: Ambiente local para captura e inspeção de e-mails enviados.
    - Spring Mail: Estrutura configurada para envio de notificações, como alertas de estoque.
- **Testes:** JUnit 5 (Jupiter), Mockito, AssertJ
- **Build:** Maven
- **Documentação:** SpringDoc OpenAPI (Swagger UI)

4. ENTIDADES DO SISTEMA

4.1 Book-Domain (Catálogo)

Responsável pelas regras de negócio dos produtos.

- **Entidades:** Livro (Abstrata), Autor, Editora, Categoria, Funcionário.
- **Padrões Aplicados:**
    - **Polimorfismo/Template Method (RN02):** Cálculo de preço dinâmico nas subclasses LivroCapaDura, LivroBrochura e LivroDigital.
    - **Rich Domain Model:** Lógica de validação de estoque (decrementarEstoque, verificarEstoqueMinimo) encapsulada na entidade.
- **Funcionalidades:** CRUD de Livros, Baixa de Estoque.

4.1.1 Livro

Representa os livros disponíveis para venda.
Atributos:

- ID
- Título
- Preço
- ISBN
- Status
- Quantidade
- Resumo
- Ano Publicação
- Número de Páginas
- Editora
- Categoria
- Autores

4.1.2 Editora

Empresa responsável pela publicação dos livros.
Atributos:

- ID
- Nome
- CNPJ

- Telefone
- Email

4.1.3 Funcionário

Atributos:

- ID
- Nome
- Matrícula

4.1.4 Categoria

Atributos:

- ID
- Nome

4.1.5 Autor

Atributos:

- ID
- Nome
- Data de Nascimento
- Nacionalidade

5.1 Customer-Domain (Cliente)

Responsável pela gestão dos usuários.

- **Entidades:** Cliente (Aggregate Root), Endereço.
- **Funcionalidades:** Cadastro com validação de unicidade (CPF/Email), Busca por Email.

5.1.1 Cliente

Representa o usuário que efetua compras na livraria.
Atributos principais:

- ID
- Nome
- CPF
- Data Nascimento
- Email

- Telefone
- Endereço
- Senha

5.1.2 Endereço

Atributos principais:

- ID
- CEP
- Rua
- Número
- Complemento
- Bairro
- Cidade
- Estado

6.1 Order-Domain (Pedidos)

Responsável pelas compras efetuadas pelos clientes.

- **Entidades:** Pedido, ItemPedido.
- **Padrões Aplicados:**
    - **Strategy Pattern (RN04):** Hierarquia Pagamento → PagamentoPix (8% desconto) e PagamentoCartao (3% à vista).
    - **Factory Pattern:** Classe PagamentoFactory para decidir qual estratégia de pagamento instanciar.
    - **Snapshot Pattern:** ItemPedido congela o preço do livro no momento da compra.
- **Funcionalidades:** Efetuar Pedido (Transacional), Cálculo de Total, Histórico.

6.1.1 Pedido

Representa uma compra realizada por um cliente.
Atributos principais:

- ID
- Data do Pedido
- Status
- Valor Total
- Valor Frete
- Prazo Entrega

- Endereço Entrega

6.1.2 Item Pedido

Atributos principais:

- ID
- Quantidade
- Preço Unitário

5. RELACIONAMENTO ENTRE ENTIDADES

O sistema foi modelado com base em relacionamentos entre entidades que representam com fidelidade o funcionamento real de uma livraria:

Entidade A	Relacionamento	Entidade B
Cliente	1:N	Pedido
Pedido	1:N	ItemPedido
Livro	N:M	Autor
Livro	N:M	Categoria
Livro	1:N	Editora

- **Cliente 1:N Pedido**
    - Indica que um cliente pode realizar vários pedidos, porém cada pedido pertence a apenas um cliente.
- **Pedido 1:N ItemPedido**
    - Indica que um pedido pode conter vários itens, e cada item está associado a um único pedido.
- **Livro M:N Autor**
    - Indica que um livro pode ter vários autores, e vários autores podem ter escrito vários livros.
- **Livro M:N Categoria**
    - Indica que um livro pode estar em várias categorias, e uma categoria pode agrupar vários livros.

- **Livro 1:N Editora**
    - Indica que um livro pode pertencer a apenas uma editora, e uma editora pode estar associada a vários livros.

6. DOCUMENTAÇÃO DA API

- **Cliente-Controller**

Verbo HTTP	Path	Body de Requisição	Body de Retorno	Status Sucesso	Status Erro	Observações
GET	/api/clientes	—	Lista de clientes	200 OK	500	Retorna todos os clientes
GET	/api/clientes/{id}	—	Cliente	200 OK	404	Busca por ID
GET	/api/clientes/email/{email}	—	Cliente	200 OK	404	Busca por e-mail
POST	/api/clientes	JSON com dados do cliente	Cliente criado	201 Created	400	Criação
PUT	/api/clientes/{id}	JSON atualizado	Cliente	200 OK	404	Atualização
DELETE	/api/clientes/{id}	—	—	204 No Content	404	Exclusão

- **Livro-Controller**

Verbo HTTP	Path	Body de Requisição	Body de Retorno	Status Sucesso	Status Erro
GET	/api/livros	—	Lista de livros	200	500
GET	/api/livros/{id}	—	Livro detalhado	200	404
POST	/api/livros	JSON	Livro	201	400
PUT	/api/livros/{id}	JSON	Livro	200	404
DELETE	/api/livros/{id}	—	—	204	404

- **Editora-Controller**

Verbo HTTP	Path	Body de Requisição	Body de Retorno	Status Sucesso	Status Erro
GET	/api/editoras	—	Lista de editoras	200	500
GET	/api/editoras/{id}	—	Editora	200	404
POST	/api/editoras	JSON	Editora	201	400
PUT	/api/editoras/{id}	JSON	Editora	200	404
DELETE	/api/editoras/{id}	—	—	204	404

- **Pedido-Controller**

Verbo HTTP	Path	Body de Requisição	Body de Retorno	Status HTTP de Sucesso	Status HTTP de Erro	Observações
GET	/api/pedidos	—	Lista de pedidos	200 OK	500	Retorna todos os pedidos
GET	/api/pedidos/{id}	—	Pedido	200 OK	404	Busca por ID
GET	/api/pedidos/cliente/{clienteId}	—	Lista	200 OK	404	Pedidos de um cliente
POST	/api/pedidos	JSON do pedido	Pedido criado	201 Created	400	Cria pedido
PUT	/api/pedidos/{id}	JSON atualizado	Pedido	200 OK	404	Atualiza
DELETE	/api/pedidos/{id}	—	—	204 No Content	404	Remove

7. Implantação da API com NGINX

A implantação da API foi realizada utilizando duas instâncias do projeto Spring Boot e um servidor NGINX configurado como reverse proxy para habilitar o balanceamento de carga.

As instâncias da API foram iniciadas com o comando:

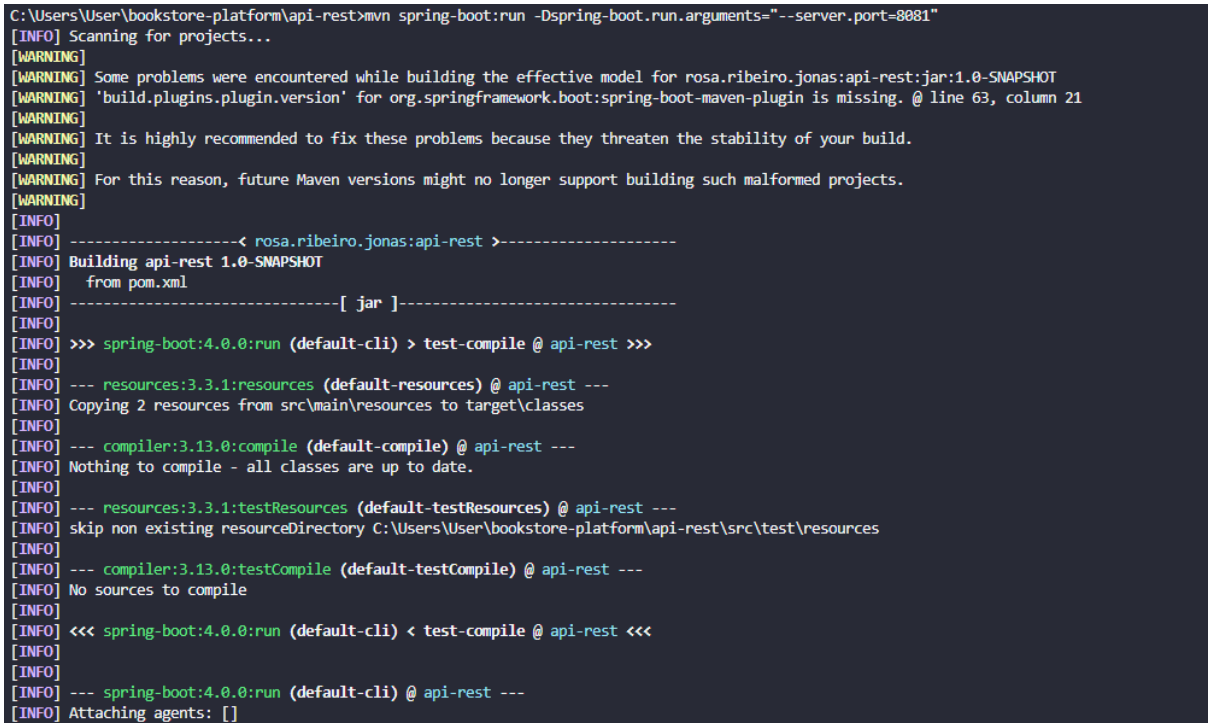
- mvn spring-boot:run

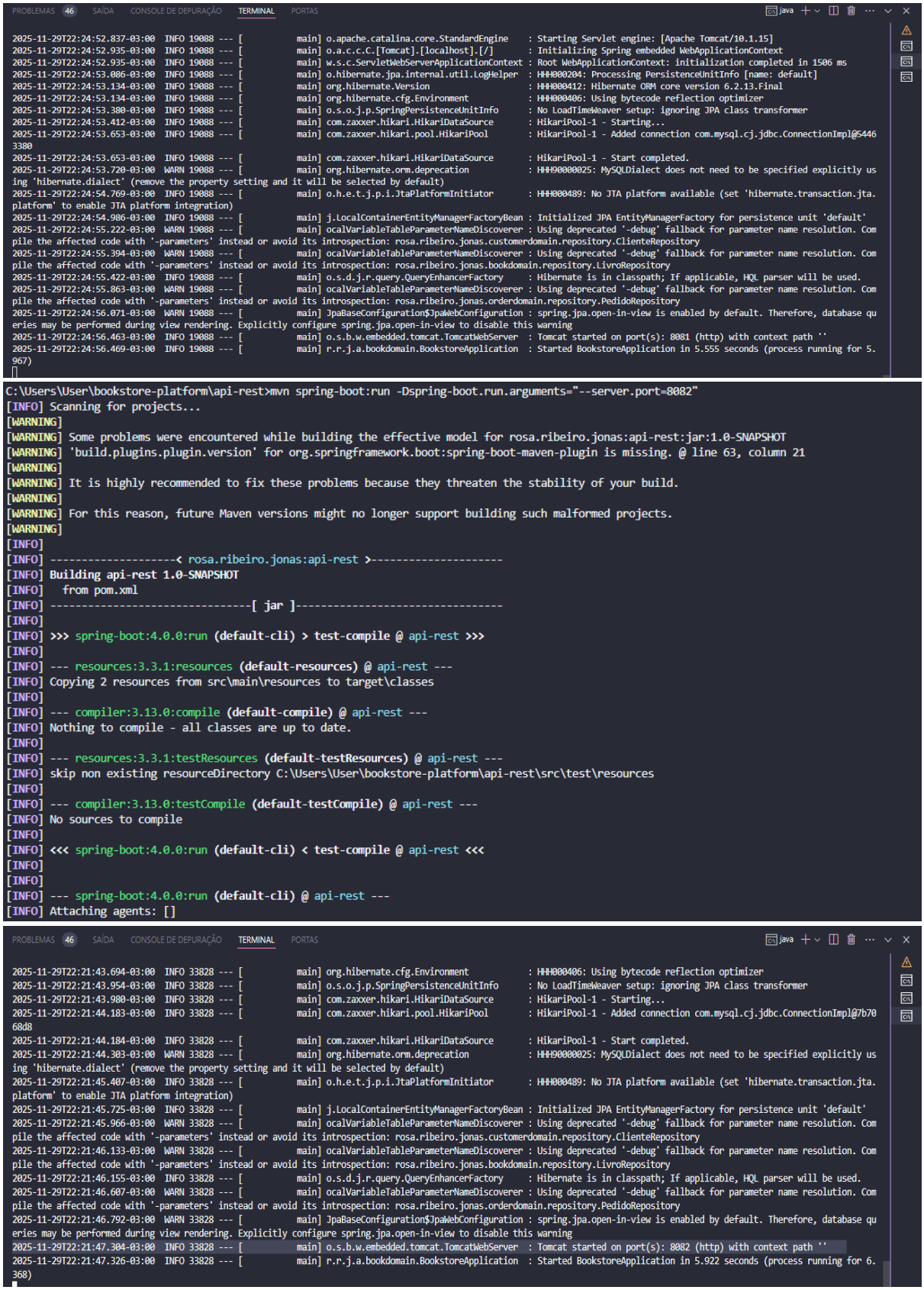
Executadas nos diretórios correspondentes:

- C:\Users\User\bookstore-platform\api-rest\

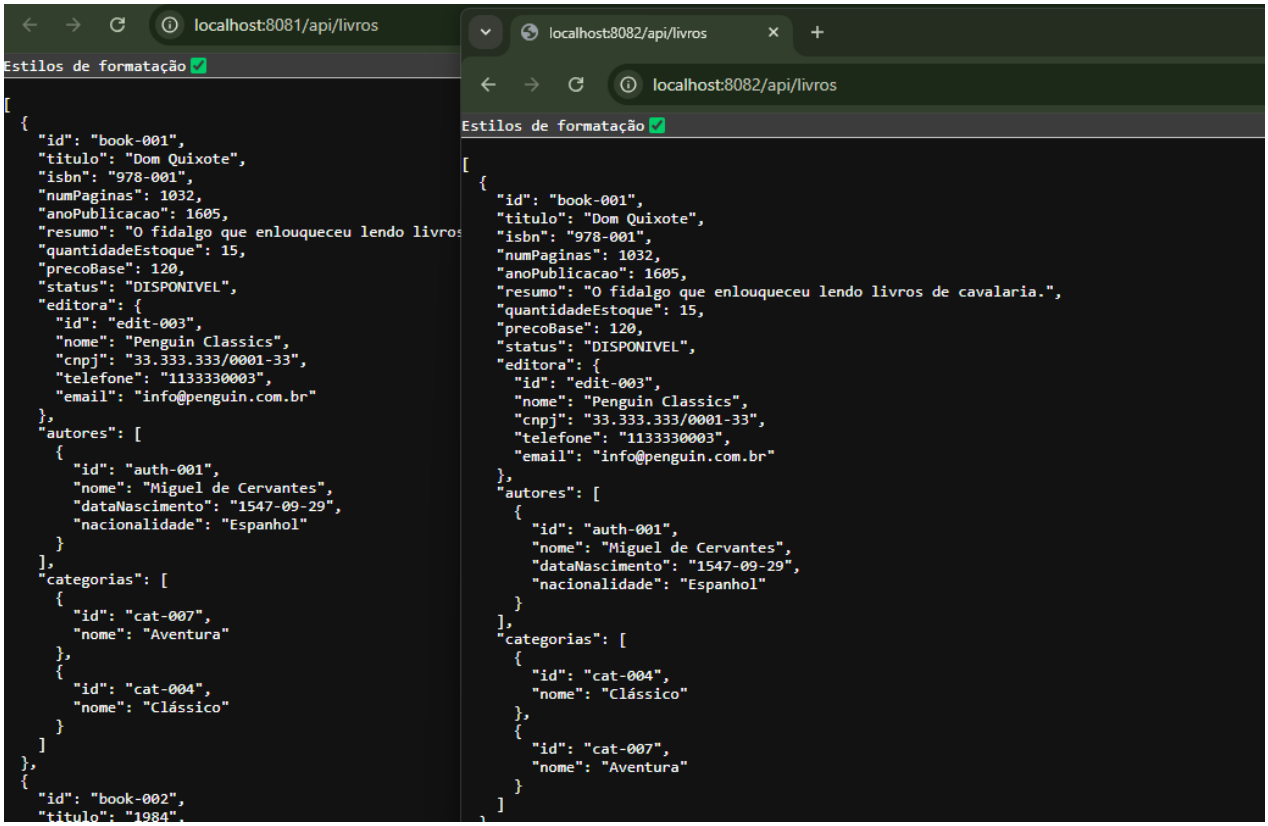
Duas imagens foram registradas mostrando:

- As portas **8081** e **8082** sendo iniciadas no terminal
- Ambas as instâncias acessíveis no navegador, confirmando o funcionamento





Dois backends Spring Boot executando nas portas 8081 e 8082 simultaneamente.



Ambos os serviços foram testados diretamente via navegador nas portas 8081 e 8082, retornando corretamente a lista de livros.

Na sequência, foi inserida a configuração utilizada no arquivo **nginx.conf**, acompanhada de imagens demonstrando:

- O upstream configurado para distribuir as requisições
- O NGINX em execução
- As portas 8081, 8082 e 8080 ativas simultaneamente (via netstat)
- O endpoint funcionando através da porta 8080 no navegador
- http://localhost:8080/api/livros

Com isso, o balanceamento de carga foi implementado com sucesso.

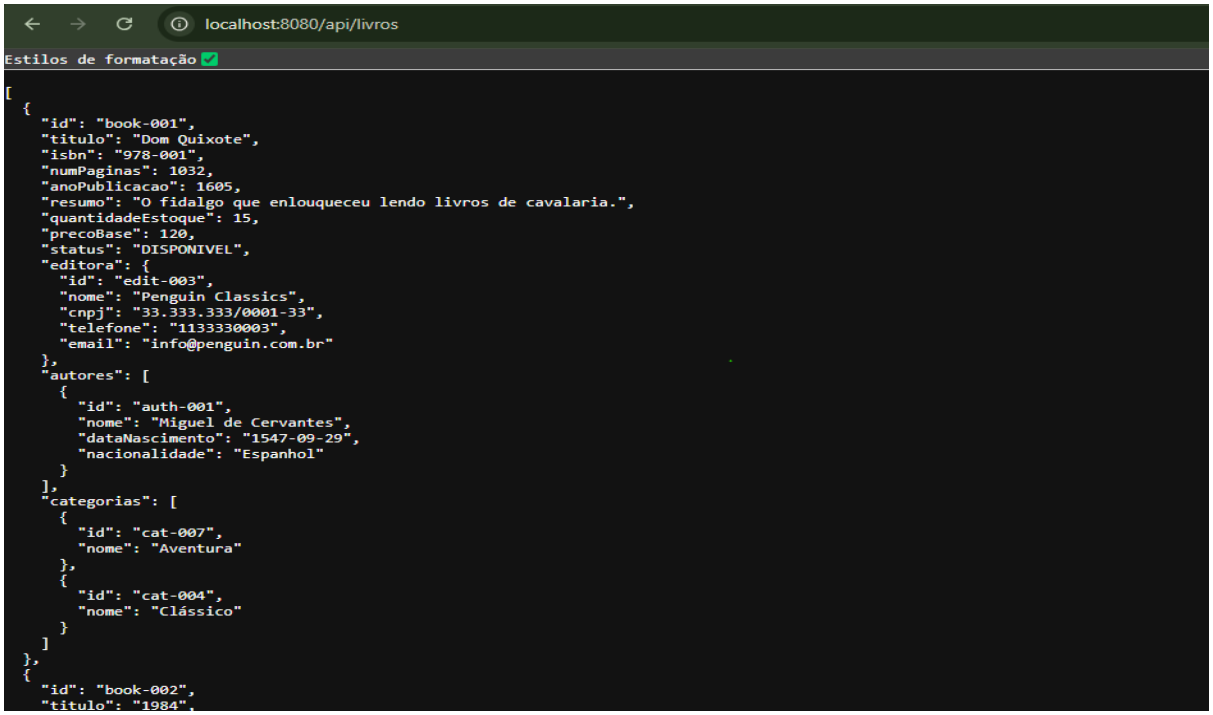
nginx.conf

conf > nginx.conf

```
worker_processes  1;

events {
    worker_connections  1024;
}

http {
    upstream backend {
        server 127.0.0.1:8081;
        server 127.0.0.1:8082;
    }

    server {
        listen 8080;

        location / {
            proxy_pass http://backend;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
    }
}
```

```
C:\nginx>netstat -ano | findstr ":8080"
  TCP    0.0.0.0:8080           0.0.0.0:0              LISTENING       20484

C:\nginx>netstat -ano | findstr ":8081"
  TCP    0.0.0.0:8081           0.0.0.0:0              LISTENING       19088
  TCP    [::]:8081              [::]:0                 LISTENING       19088
  TCP    [::1]:62474            [::1]:8081             TIME_WAIT       0
  TCP    [::1]:63829            [::1]:8081             TIME_WAIT       0

C:\nginx>netstat -ano | findstr ":8082"
  TCP    0.0.0.0:8082           0.0.0.0:0              LISTENING       33828
  TCP    [::]:8082              [::]:0                 LISTENING       33828
  TCP    [::1]:57842            [::1]:8082             TIME_WAIT       0
  TCP    [::1]:59517            [::1]:8082             TIME_WAIT       0
```

NGINX executando na porta 8080 e aplicações Spring Boot nas portas 8081 e 8082.



NGINX via navegador na porta 8080, retornando corretamente a lista de livros.

8. Relatório de testes de performance com JMeter

Os testes de desempenho foram realizados com o Apache JMeter, avaliando dois cenários:

- Sem balanceamento (acessando diretamente a porta 8081)
- Com balanceamento (acessando a API pela porta 8080 via NGINX)

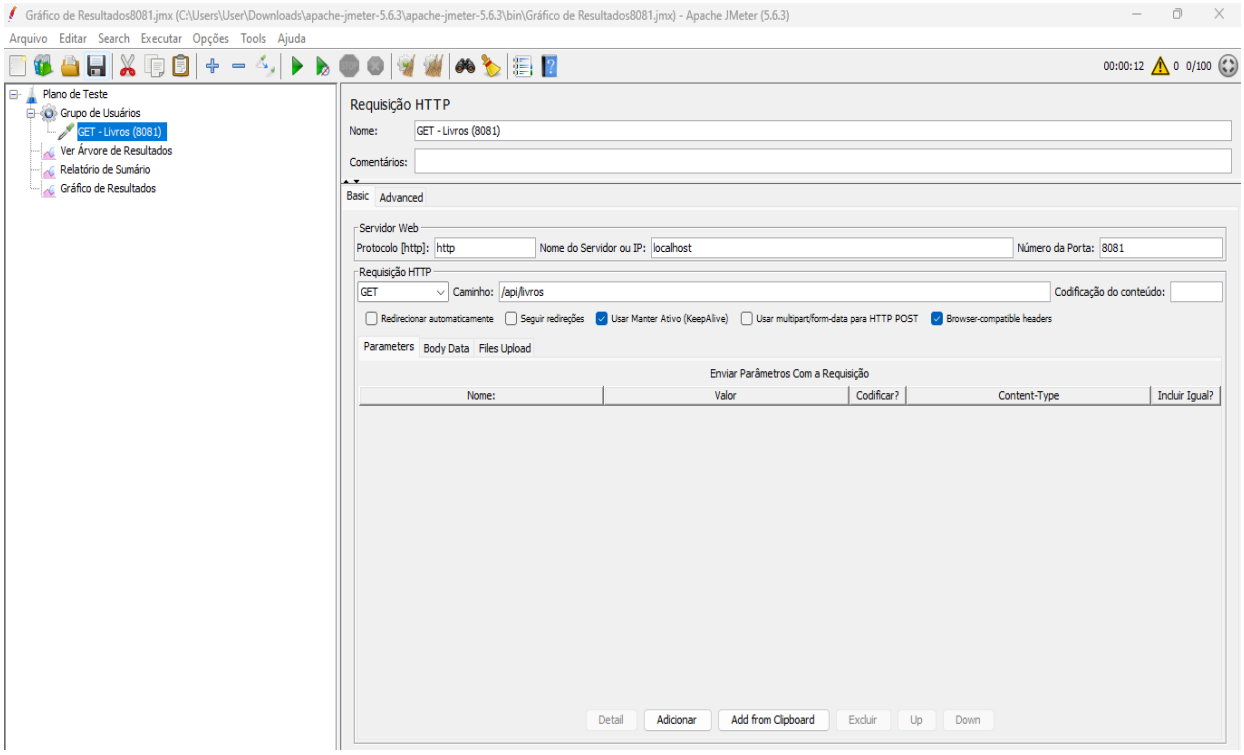
Cada teste foi configurado com 100 usuários virtuais, Ramp-Up de 10 segundos e 5 iterações, totalizando 500 requisições no primeiro cenário e 1000 no segundo.

- **Configurando o primeiro teste (SEM balanceamento)**

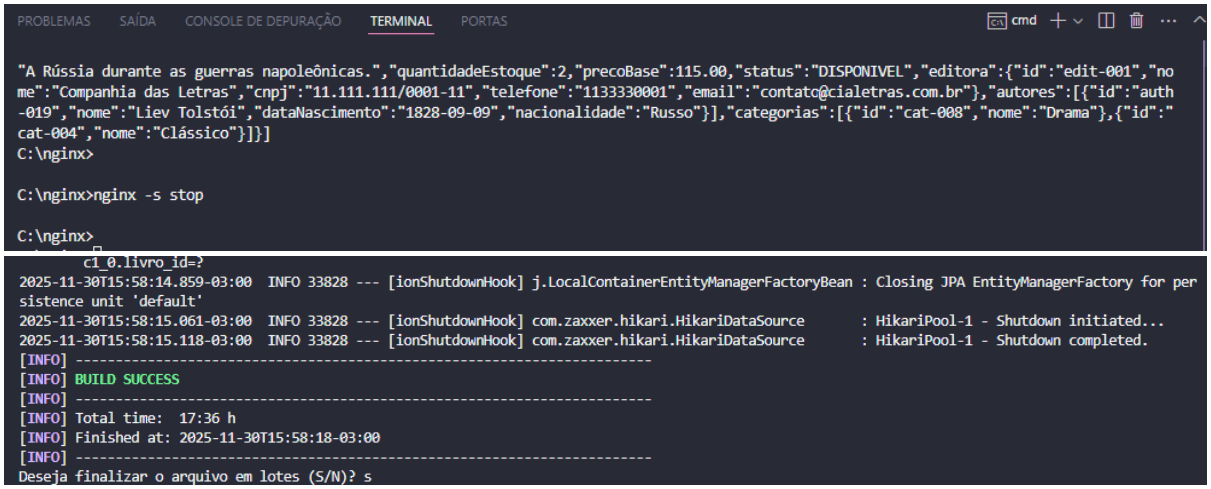
Para este cenário, apenas a instância da API na porta **8081** permaneceu ativa.

As imagens registram:

- A configuração da requisição HTTP no JMeter com:
  - Servidor: **localhost**
  - Porta: **8081**
  - Caminho: `/api/livros`



- Prova visual de que:
    - O **NGINX** estava desligado
    - A porta **8082** estava desligada
    - Apenas a porta **8081** estava em execução



- Resultados do teste:
    - Gráfico de resultados
    - Relatório de Sumário
    - Árvore de Resultados
    - Resultados em Tabela

Esses registros validam o comportamento da API com apenas uma instância ativa.

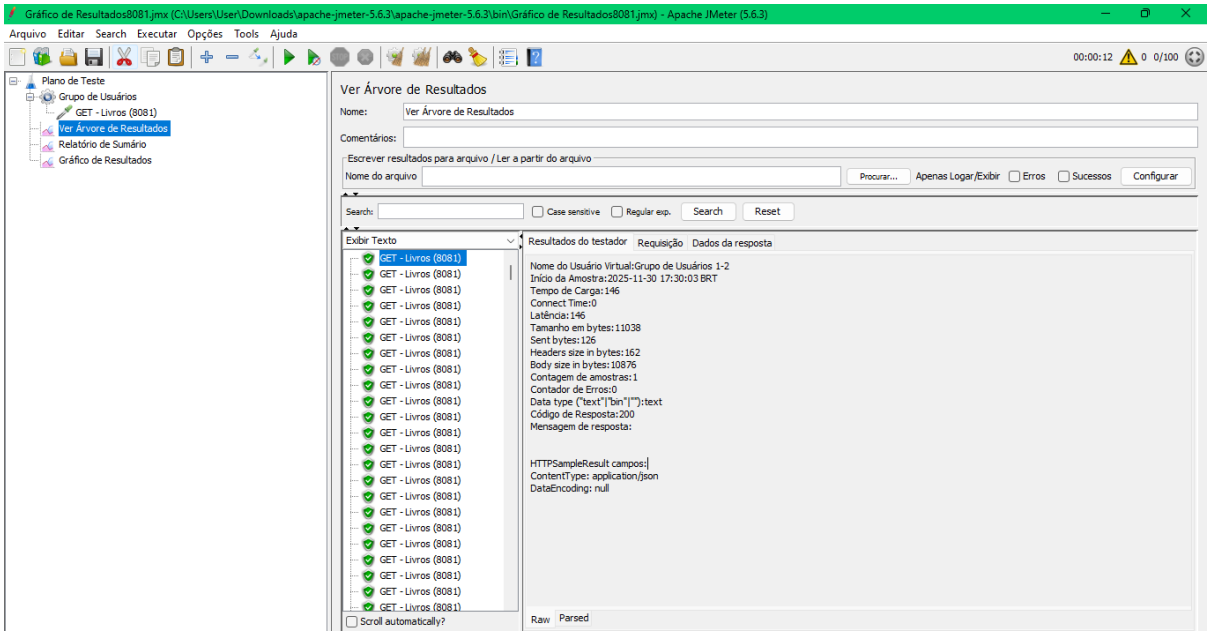


Gráfico de Resultados8081.jmx (C:\Users\User\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\Gráfico de Resultados8081.jmx) - Apache JMeter (5.6.3)
Arquivo Editar Search Executar Opções Tools Ajuda
00:00:12 0 0/100
Plano de Teste
Grupo de Usuários
GET - Livros (8081)
Ver Árvore de Resultados
Relatório de Sumário
Gráfico de Resultados
Ver Árvore de Resultados
Nome: Ver Árvore de Resultados
Comentários:
Escrever resultados para arquivo / Ler a partir do arquivo
Nome do arquivo
Procurar... Apenas Logar/Exibir Erros Sucessos Configurar
Search: Case sensitive Regular exp. Search Reset
Exibir Texto
GET - Livros (8081)
Scroll automatically?
Resultados do testador Requisição Dados da resposta
Nome do Usuário Virtual:Grupo de Usuários 1-2
Início da Amostra:2025-11-30 17:30:03 BRT
Tempo de Carga:146
Connect Time:0
Latência:146
Tamanho em bytes:11038
Sent bytes:126
Headers size in bytes:162
Body size in bytes:10876
Contagem de amostras:1
Contador de Erros:0
Data type ("text"|"bin"|""):text
Código de Resposta:200
Mensagem de resposta:
HTTPSampleResult campos:
ContentType: application/json
DataEncoding: null
Raw Parsed

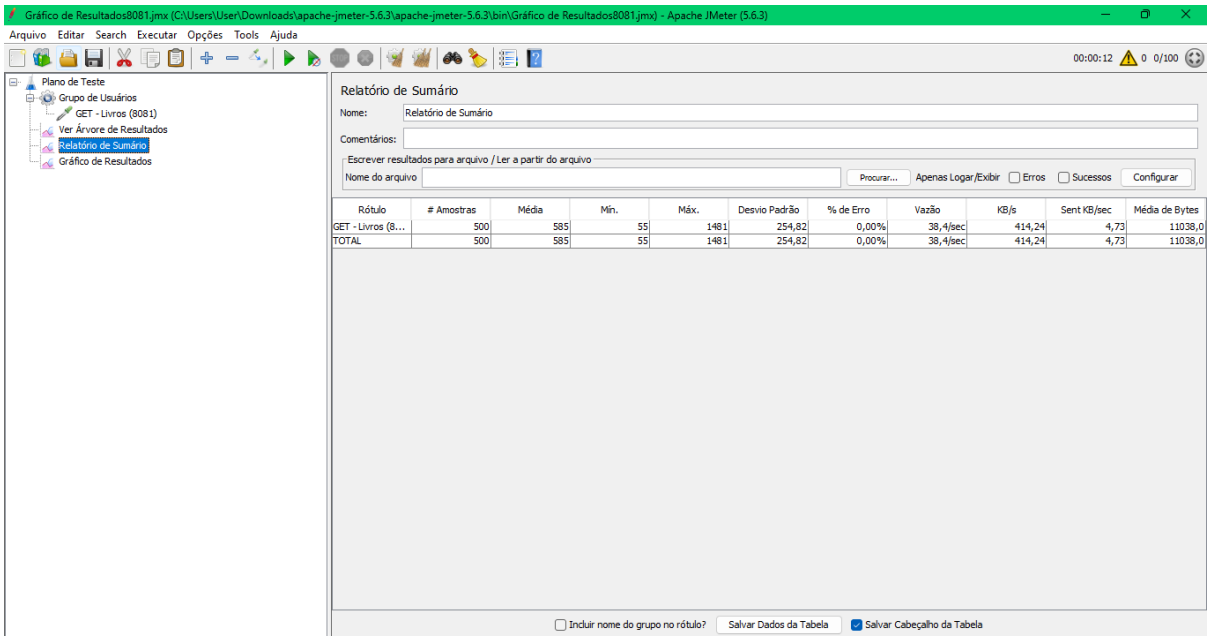
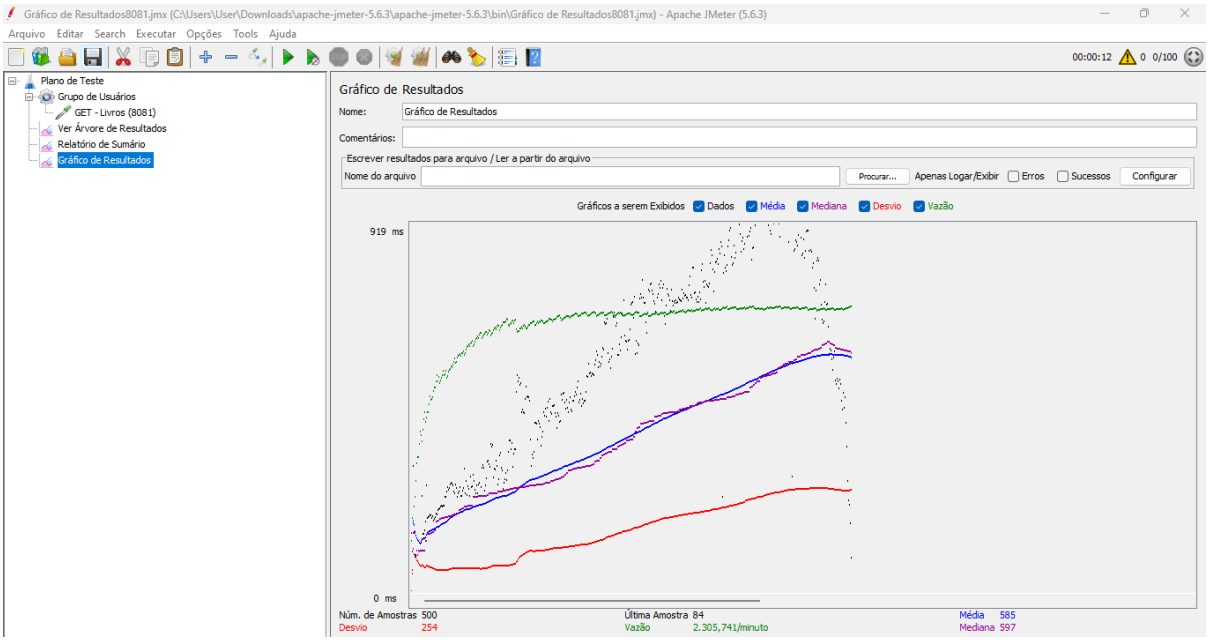


Gráfico de Resultados8081.jmx (C:\Users\User\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\Gráfico de Resultados8081.jmx) - Apache JMeter (5.6.3)
Arquivo Editar Search Executar Opções Tools Ajuda
00:00:12 0 0/100
Plano de Teste
Grupo de Usuários
GET - Livros (8081)
Ver Árvore de Resultados
Relatório de Sumário
Gráfico de Resultados
Relatório de Sumário
Nome: Relatório de Sumário
Comentários:
Escrever resultados para arquivo / Ler a partir do arquivo
Nome do arquivo
Procurar... Apenas Logar/Exibir Erros Sucessos Configurar
Rótulo # Amostras Média Mín. Máx. Desvio Padrão % de Erro Vazão KB/s Sent KB/sec Média de Bytes
GET - Livros (8... 500 585 55 1481 254,82 0,00% 38,4/sec 414,24 4,73 11038,0
TOTAL 500 585 55 1481 254,82 0,00% 38,4/sec 414,24 4,73 11038,0
Incluir nome do grupo no rótulo? Salvar Dados da Tabela Salvar Cabeçalho da Tabela



Ver Resultados em Tabela

Nome: Ver Resultados em Tabela

Comentários:

Escrever resultados para arquivo / Ler a partir do arquivo

Nome do arquivo | Procurar... | Apenas Logar/Exibir | Erros | Sucessos | Configurar

Amostra #	Tempo de início	Nome do Usuári...	Rótulo	Tempo da amos...	Estado	Bytes	Sent Bytes	Latency	Connect Time(ms)
1	17:52:44.685	Grupo de Usuári...	GET - Livros (808...	70		11038	126	70	1
2	17:52:44.757	Grupo de Usuári...	GET - Livros (808...	49		11038	126	49	0
3	17:52:44.786	Grupo de Usuári...	GET - Livros (808...	50		11038	126	50	1
4	17:52:44.806	Grupo de Usuári...	GET - Livros (808...	47		11038	126	47	0
5	17:52:44.836	Grupo de Usuári...	GET - Livros (808...	47		11038	126	47	0
6	17:52:44.853	Grupo de Usuári...	GET - Livros (808...	56		11038	126	56	0
7	17:52:44.883	Grupo de Usuári...	GET - Livros (808...	79		11038	126	79	0
8	17:52:44.886	Grupo de Usuári...	GET - Livros (808...	82		11038	126	82	0
9	17:52:44.909	Grupo de Usuári...	GET - Livros (808...	90		11038	126	90	0
10	17:52:44.962	Grupo de Usuári...	GET - Livros (808...	101		11038	126	101	0
11	17:52:44.968	Grupo de Usuári...	GET - Livros (808...	100		11038	126	100	0
12	17:52:44.986	Grupo de Usuári...	GET - Livros (808...	99		11038	126	99	1
13	17:52:45.063	Grupo de Usuári...	GET - Livros (808...	97		11038	126	97	0
14	17:52:45.069	Grupo de Usuári...	GET - Livros (808...	100		11038	126	99	0
15	17:52:45.085	Grupo de Usuári...	GET - Livros (808...	94		11038	126	94	0
16	17:52:45.087	Grupo de Usuári...	GET - Livros (808...	93		11038	126	93	0
17	17:52:45.168	Grupo de Usuári...	GET - Livros (808...	83		11038	126	83	0
18	17:52:45.179	Grupo de Usuári...	GET - Livros (808...	83		11038	126	83	0
19	17:52:45.180	Grupo de Usuári...	GET - Livros (808...	87		11038	126	87	0
20	17:52:45.185	Grupo de Usuári...	GET - Livros (808...	90		11038	126	90	1
21	17:52:45.251	Grupo de Usuári...	GET - Livros (808...	106		11038	126	106	0
22	17:52:45.263	Grupo de Usuári...	GET - Livros (808...	107		11038	126	107	0
23	17:52:45.268	Grupo de Usuári...	GET - Livros (808...	106		11038	126	106	0
24	17:52:45.275	Grupo de Usuári...	GET - Livros (808...	99		11038	126	99	0
25	17:52:45.285	Grupo de Usuári...	GET - Livros (808...	108		11038	126	108	1
26	17:52:45.370	Grupo de Usuári...	GET - Livros (808...	129		11038	126	129	0
27	17:52:45.375	Grupo de Usuári...	GET - Livros (808...	129		11038	126	129	0

- ## Resumo dos Resultados — (8081) Sem Balanceamento

- **Amostras:** 500
- **Média:** 585 ms
- **Min:** 55 ms
- **Max:** 1481 ms
- **Desvio Padrão:** 254,82
- **% Erro:** 0%
- **Vazão (Throughput):** 38,4 requisições/seg
- **KB/s:** 414,24
- **Média de bytes:** 11038 bytes

Esses valores indicam uma operação estável, com nenhuma taxa de erro e boa taxa de processamento, considerando que apenas uma instância estava respondendo as requisições.

- ## Configurando o segundo teste (COM balanceamento)

Nesse cenário, as três portas foram ativadas simultaneamente:

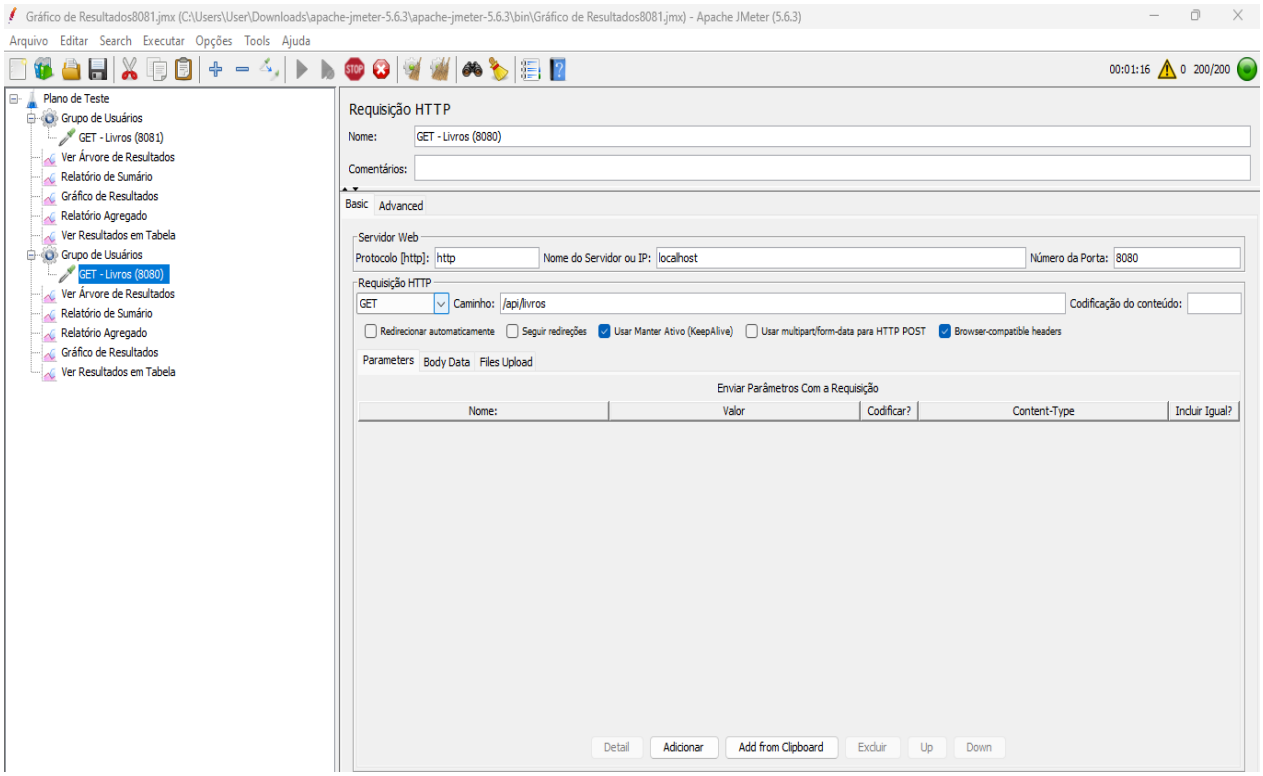
- 8081 e 8082: Instâncias da API
- 8080: NGINX realizando o balanceamento

As imagens utilizadas demonstram:

- O NGINX e ambas as instâncias da API ativos

```
C:\nginx>netstat -ano | findstr /R "8080 8081 8082"
  TCP    0.0.0.0:8080           0.0.0.0:0              LISTENING       22456
  TCP    0.0.0.0:8081           0.0.0.0:0              LISTENING       22608
  TCP    0.0.0.0:8082           0.0.0.0:0              LISTENING       34048
  TCP    [::]:8081              [::]:0                 LISTENING       22608
  TCP    [::]:8082              [::]:0                 LISTENING       34048
```

Configuração da requisição HTTP no JMeter apontando para a porta **8080**



- Resultados do teste:
  - Gráfico de resultados
  - Relatório de Sumário
  - Árvore de Resultados
  - Relatório Agregado
  - Resultados em Tabela

Esses dados comprovam o funcionamento correto do balanceamento de carga e permitem comparação com o cenário sem balanceamento.

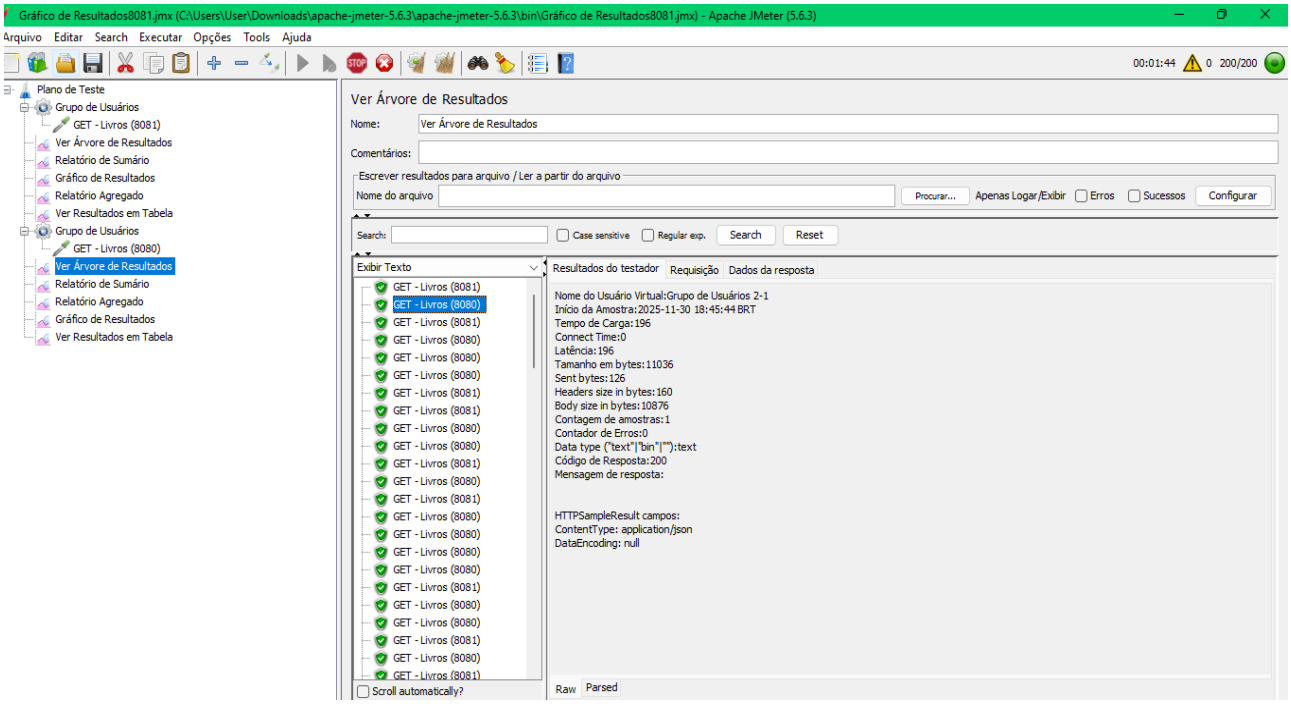


Gráfico de Resultados8081.jmx (C:\Users\User\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\Gráfico de Resultados8081.jmx) - Apache JMeter (5.6.3)
Arquivo Editar Search Executar Opções Tools Ajuda
00:01:44 0 200/200
Plano de Teste
Grupo de Usuários
GET - Livros (8081)
Ver Árvore de Resultados
Relatório de Sumário
Gráfico de Resultados
Relatório Agregado
Ver Resultados em Tabela
Grupo de Usuários
GET - Livros (8080)
Ver Árvore de Resultados
Relatório de Sumário
Relatório Agregado
Gráfico de Resultados
Ver Resultados em Tabela
Ver Árvore de Resultados
Nome: Ver Árvore de Resultados
Comentários:
Escrever resultados para arquivo / Ler a partir do arquivo
Nome do arquivo
Procurar...
Apenas Logar/Exibir
Erros
Sucessos
Configurar
Search:
Case sensitive
Regular exp.
Search
Reset
Exibir Texto
Resultados do testador
Requisição
Dados da resposta
Nome do Usuário Virtual:Grupo de Usuários 2-1
Início da Amostra:2025-11-30 18:45:44 BRT
Tempo de Carga:196
Connect Time:0
Latência:196
Tamanho em bytes:11036
Sent bytes:126
Headers size in bytes:160
Body size in bytes:10876
Contagem de amostras:1
Contador de Erros:0
Data type ("text"|"bin"|""):text
Código de Resposta:200
Mensagem de resposta:
HTTPSampleResult campos:
ContentType: application/json
DataEncoding: null
Scroll automatically?
Raw
Parsed

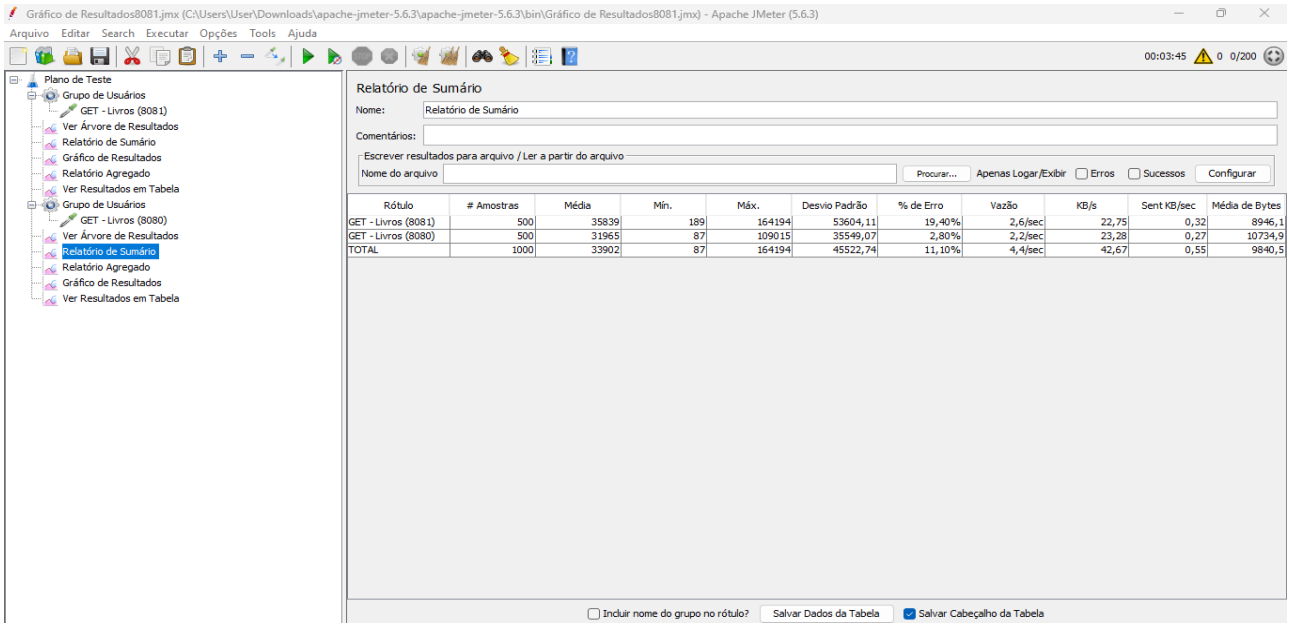


Gráfico de Resultados8081.jmx (C:\Users\User\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\Gráfico de Resultados8081.jmx) - Apache JMeter (5.6.3)
Arquivo Editar Search Executar Opções Tools Ajuda
00:03:45 0 0/200
Plano de Teste
Grupo de Usuários
GET - Livros (8081)
Ver Árvore de Resultados
Relatório de Sumário
Gráfico de Resultados
Relatório Agregado
Ver Resultados em Tabela
Grupo de Usuários
GET - Livros (8080)
Ver Árvore de Resultados
Relatório de Sumário
Relatório Agregado
Gráfico de Resultados
Ver Resultados em Tabela
Relatório de Sumário
Nome: Relatório de Sumário
Comentários:
Escrever resultados para arquivo / Ler a partir do arquivo
Nome do arquivo
Procurar...
Apenas Logar/Exibir
Erros
Sucessos
Configurar
Rótulo # Amostras Média Mín. Máx. Desvio Padrão % de Erro Vazão KB/s Sent KB/sec Média de Bytes
GET - Livros (8081) 500 35839 189 164194 53604,11 19,40% 2,6/sec 22,75 0,32 8946,1
GET - Livros (8080) 500 31965 87 109015 35549,07 2,80% 2,2/sec 23,28 0,27 10734,9
TOTAL 1000 33902 87 164194 45522,74 11,10% 4,4/sec 42,67 0,55 9840,5
Incluir nome do grupo no rótulo?
Salvar Dados da Tabela
Salvar Cabeçalho da Tabela

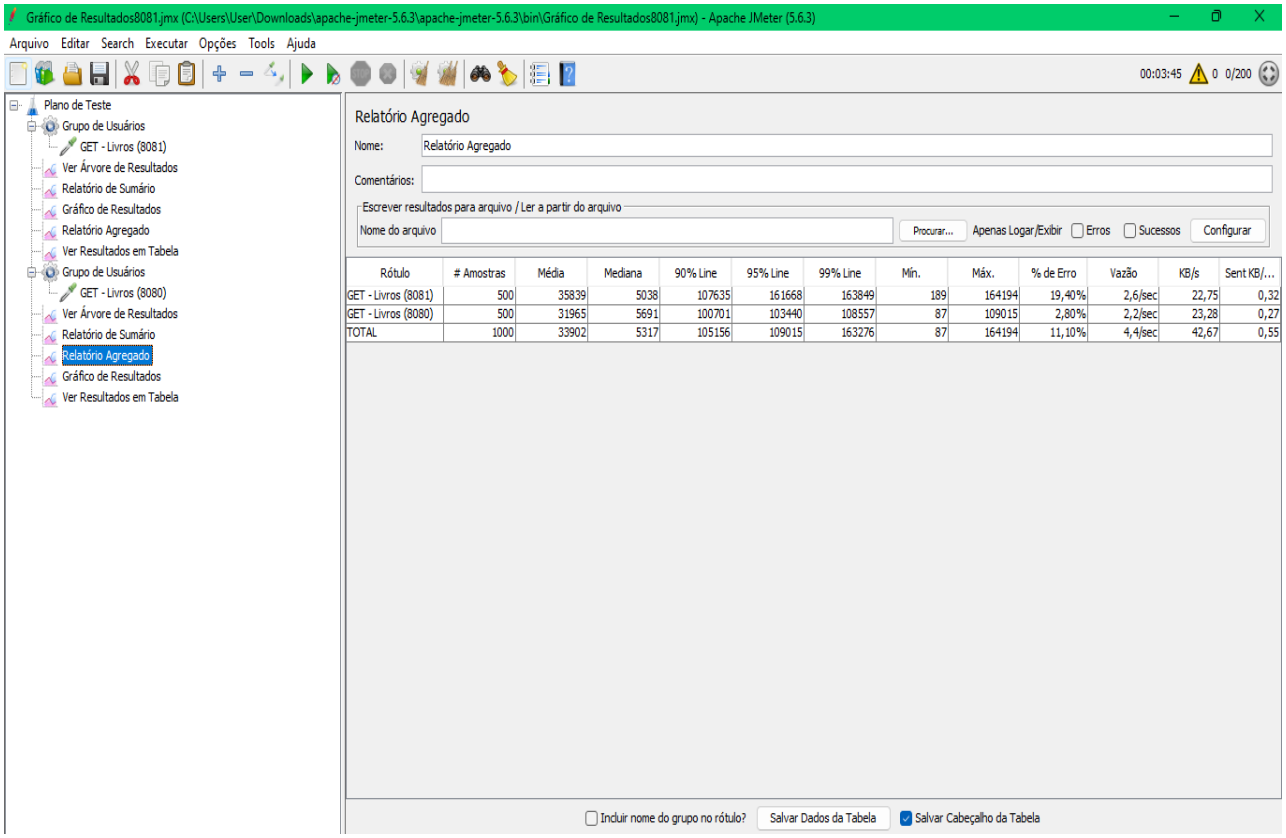


Gráfico de Resultados8081.jmx (C:\Users\User\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\Gráfico de Resultados8081.jmx) - Apache JMeter (5.6.3)
Arquivo Editar Search Executar Opções Tools Ajuda
00:03:45 0 0/200
Plano de Teste
Grupo de Usuários
GET - Livros (8081)
Ver Árvore de Resultados
Relatório de Sumário
Gráfico de Resultados
Relatório Agregado
Ver Resultados em Tabela
Grupo de Usuários
GET - Livros (8080)
Ver Árvore de Resultados
Relatório de Sumário
Relatório Agregado
Gráfico de Resultados
Ver Resultados em Tabela
Relatório Agregado
Nome: Relatório Agregado
Comentários:
Escrever resultados para arquivo / Ler a partir do arquivo
Nome do arquivo
Procurar...
Apenas Logar/Exibir
Erros
Sucessos
Configurar
Rótulo # Amostras Média Mediana 90% Line 95% Line 99% Line Mín. Máx. % de Erro Vazão KB/s Sent KB/...
GET - Livros (8081) 500 35839 5038 107635 161668 163849 189 164194 19,40% 2,6/sec 22,75 0,32
GET - Livros (8080) 500 31965 5691 100701 103440 108557 87 109015 2,80% 2,2/sec 23,28 0,27
TOTAL 1000 33902 5317 105156 109015 163276 87 164194 11,10% 4,4/sec 42,67 0,55
Incluir nome do grupo no rótulo?
Salvar Dados da Tabela
Salvar Cabeçalho da Tabela

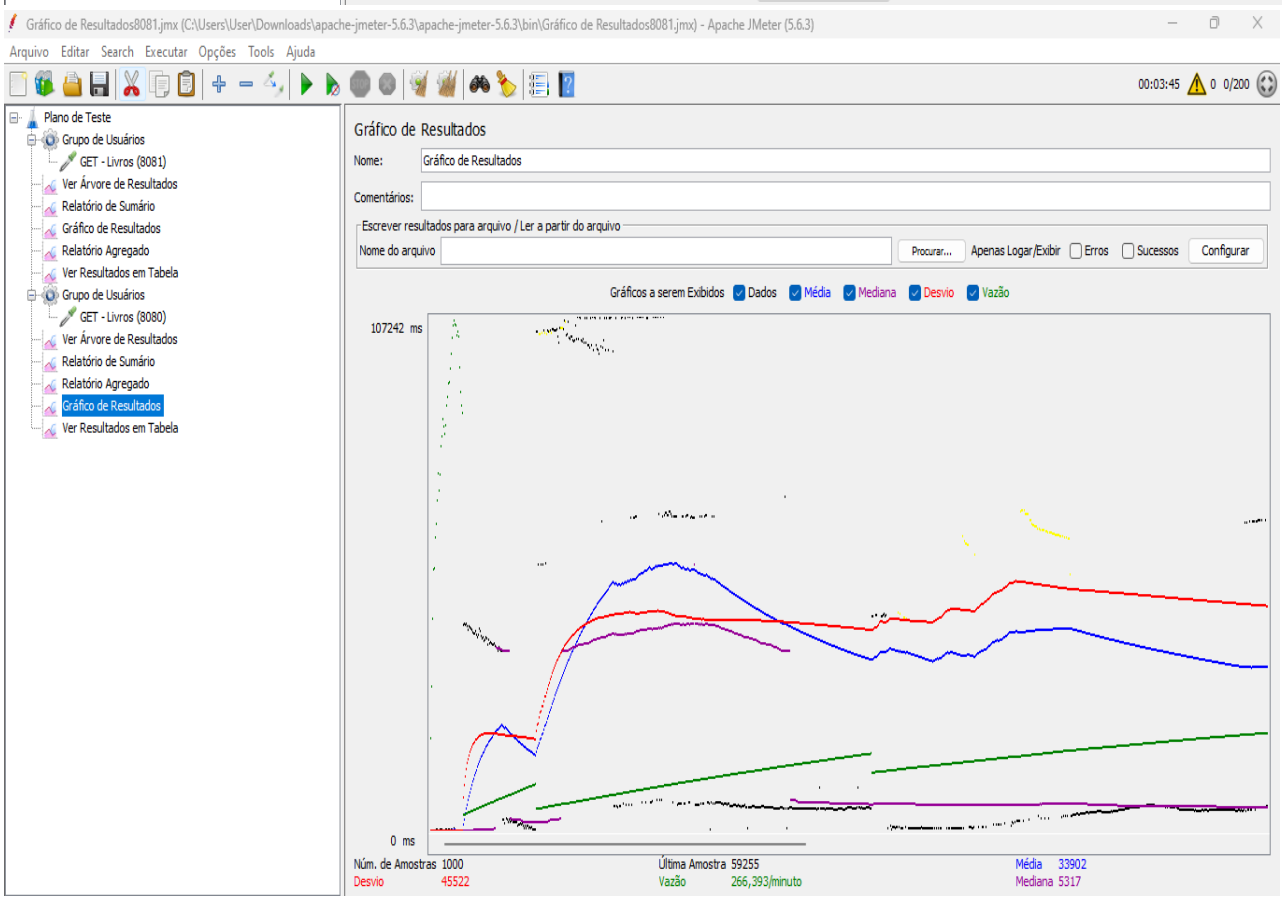
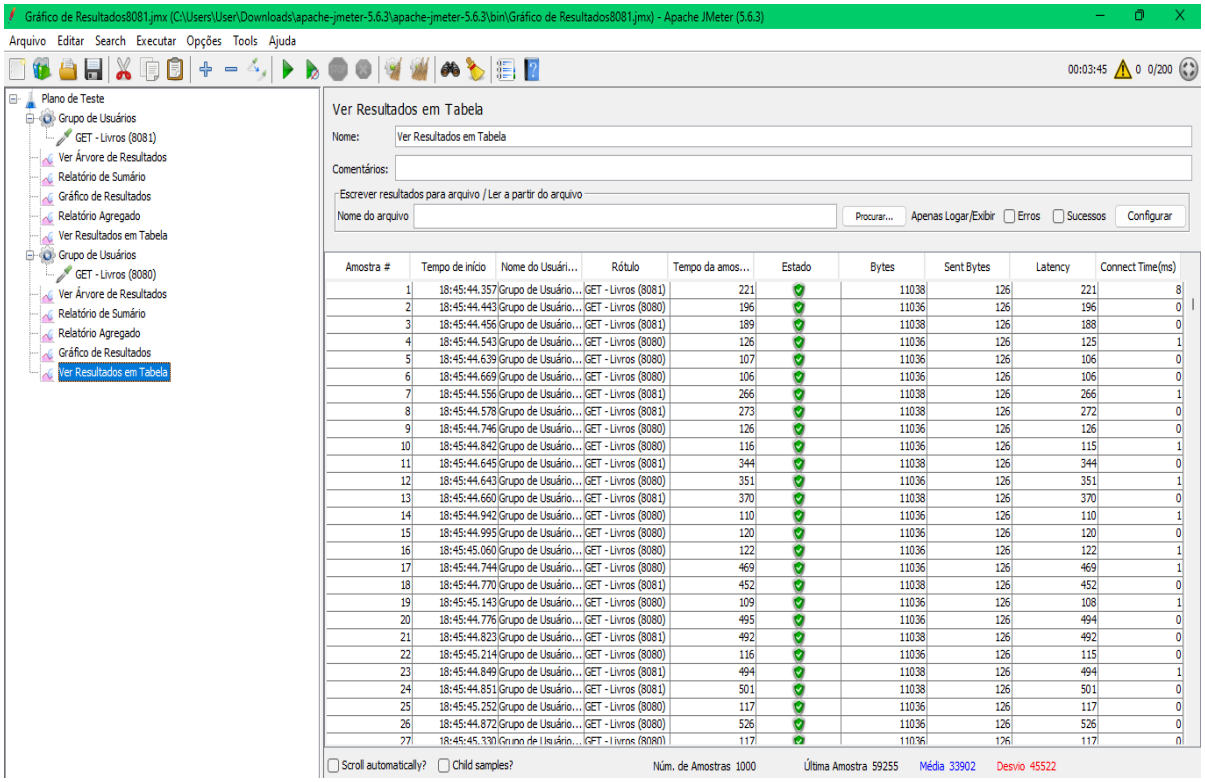


Gráfico de Resultados8081.jmx (C:\Users\User\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\Gráfico de Resultados8081.jmx) - Apache JMeter (5.6.3)
Arquivo Editar Search Executar Opções Tools Ajuda
00:03:45 0 0/200
Plano de Teste
Grupo de Usuários
GET - Livros (8081)
Ver Árvore de Resultados
Relatório de Sumário
Gráfico de Resultados
Relatório Agregado
Ver Resultados em Tabela
Grupo de Usuários
GET - Livros (8080)
Ver Árvore de Resultados
Relatório de Sumário
Relatório Agregado
Gráfico de Resultados
Ver Resultados em Tabela
Gráfico de Resultados
Nome: Gráfico de Resultados
Comentários:
Escrever resultados para arquivo / Ler a partir do arquivo
Nome do arquivo
Procurar...
Apenas Logar/Exibir
Erros
Sucessos
Configurar
Gráficos a serem Exibidos
Dados
Média
Mediana
Desvio
Vazão
107242 ms
0 ms
Núm. de Amostras 1000
Última Amostra 59255
Média 33902
Desvio 45522
Vazão 266,393/minuto
Mediana 5317



- **Resumo dos Resultados — (8080) Com Balanceamento**

- **Amostras:** 1000
- **Média:** 33.902 ms
- **Min:** 87 ms
- **Max:** 164.194 ms
- **Desvio Padrão:** 45.522,74
- **% Erro:** 11,10%
- **Vazão (Throughput):** 4,4 requisições/seg
- **KB/s:** 42,67
- **Bytes médios:** 9840,5

Este cenário apresentou uma taxa de erro maior e um tempo médio elevado, indicando sobrecarga durante a execução do teste com distribuição de carga. Isso pode ocorrer devido a configuração local, concorrência alta em duas instâncias e processamento simultâneo pelo NGINX.

- ## Comparação entre os Cenários

Métrica	Sem Balanceamento (8081)	Com Balanceamento (8080)
Amostras	500	1000
Média (ms)	585 ms	33.902 ms
Máximo (ms)	1481 ms	164.194 ms
% Erro	0%	11,10%
Throughput	38,4 req/s	4,4 req/s

- ## Análise

- O cenário **sem balanceamento** apresentou desempenho melhor, com throughput alto, zero erros e baixa latência média.
- Já o cenário **com balanceamento** obteve latência muito alta e porcentagem de erros significativa. Isso é comum em ambientes *locais*, onde múltiplas instâncias competem pelos mesmos recursos (CPU, RAM, disco).
- Apesar disso, o teste **comprova o funcionamento do balanceamento**, pois o NGINX distribuiu requisições entre 8081 e 8082 corretamente.

Projeto Componentes

1. Geração dos Artefatos (.JAR) dos Componentes

Conforme exigido pela especificação de componentes (BRADECO), cada módulo do projeto foi empacotado individualmente como um arquivo .jar.

A geração dos artefatos foi realizada através do Maven com o comando:

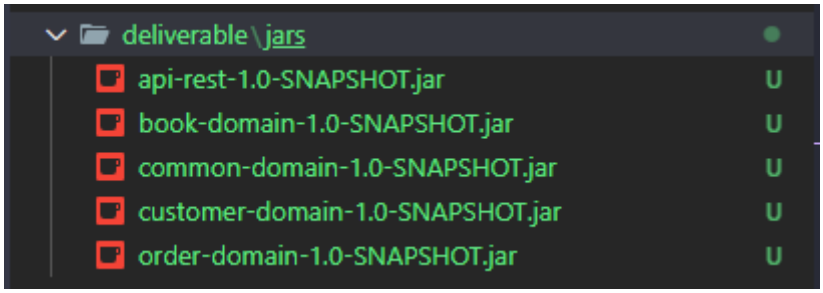
- mvn clean package –DskipTests

Após o build, os arquivos .jar gerados foram organizados na pasta:

- deliverable/jars/

Os componentes empacotados são:

- **book-domain.jar** – Regras de negócio relacionadas ao catálogo de livros
- **customer-domain.jar** – Gestão de clientes e seus endereços
- **order-domain.jar** – Processamento de pedidos
- **common-domain.jar** – Utilitários e elementos compartilhados
- **api-rest.jar** – Camada de apresentação (controllers REST) que integra os componentes



2. Aplicação dos Princípios SOLID

Aluno	RA	Componente
Vinícius Pereira Costa	BP3044289	book-domain
Jonas Ribeiro da Rosa	BP304422X	order-domain
Gabriela Vieira dos Santos Vaz	BP3044513	customer-domain
Iago Felipe Steigleder Bacci	BP3044246	common-domain

Parte Individual – Vinícius

Componente: book-domain

O componente book-domain é responsável por toda a lógica de negócio relacionada ao catálogo de livros da plataforma.

1. **Princípio da Responsabilidade Única**
   Definição: Uma classe deve ter apenas um motivo para mudar, ou seja, deve ter apenas uma responsabilidade.

Livro (Classe Abstrata) A classe Livro é responsável apenas por manter os dados e a lógica de estoque (decrementar/incrementar). A responsabilidade de cálculo de preço é delegada aos seus subtipos (SRP aplicado por delegação de responsabilidade).

2. **Princípio do Aberto/Fechado**
   Definição: Entidades de software (classes, módulos, funções, etc.) devem ser abertas para extensão, mas fechadas para modificação.

Livro e suas subclasses (LivroCapaDura, LivroBrochura, LivroDigital) O cálculo de preço está aberto para extensão. Para adicionar um novo formato de livro com uma regra de preço diferente, basta criar uma nova subclasse de Livro e implementar o método calcularPreco(). O código da classe base Livro permanece fechado para modificação.

3. **Princípio da Substituição de Liskov**
   Definição: Se S é um subtipo de T, então objetos do tipo T podem ser substituídos por objetos do tipo S sem alterar as propriedades corretas do programa.

Qualquer objeto do tipo LivroCapaDura, LivroBrochura ou LivroDigital pode ser usado no lugar de um Livro sem quebrar o comportamento esperado.

4. **Princípio da Segregação de Interface**
   Definição: “Clientes não devem ser forçados a depender de interfaces que não utilizam.”

O módulo book-domain lida com múltiplas entidades: Livro, Autor, Editora e Categoria. Em vez de criar uma única interface de persistência genérica (ex: CatalogoRepository) que conteria métodos para salvar, buscar e deletar todas essas entidades, o projeto utiliza interfaces de repositório separadas, dessa forma, cada cliente depende apenas das interfaces que realmente utiliza.

5. **Princípio da Inversão de Dependência**
   Definição:Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações.

O *ManterLivroService* e o *ManterEditoraService* utilizam apenas as interfaces *LivroRepository* e *EditoraRepository*, sem conhecer detalhes de persistência.As implementações reais desses repositórios são definidas externamente (na camada api-rest), fazendo com que os detalhes dependam das abstrações, e não o contrário.

Parte Individual – Jonas

Componente: Order-domain

É responsável por toda a lógica de negócio relacionada a pedidos, incluindo cálculo de totais, atualização de estoque, mudanças de status e aplicação das regras de pagamento.

1. **Princípio da Responsabilidade Única**
   Definição: Uma classe deve ter apenas um motivo para mudar, ou seja, deve ter apenas uma responsabilidade.

A classe *EfetuarPedidoService* tem a responsabilidade única de orquestrar a transação de criação de pedido (validar, calcular total, persistir). A lógica de cálculo de desconto é isolada nas classes de pagamento (*PagamentoPix*, *PagamentoCartao*), e a lógica de persistência é isolada nos repositórios.

2. **Princípio do Aberto/Fechado**
   Definição: Entidades de software (classes, módulos, funções, etc.) devem ser abertas para extensão, mas fechadas para modificação.

A classe abstrata *Pagamento* define o contrato (*processarDesconto*()), e as subclasses (*PagamentoPix*, *PagamentoCartao*) implementam a regra. Para adicionar uma nova forma de pagamento (ex: *PagamentoBoleto*), basta criar uma nova subclasse de Pagamento, sem modificar o código de criação de pedido.

3. **Princípio da Substituição de Liskov**
   Definição: Se S é um subtipo de T, então objetos do tipo T podem ser substituídos por objetos do tipo S sem alterar as propriedades corretas do programa.

Pagamento (Classe Abstrata) e suas subclasses *PagamentoFactory* pode retornar qualquer subtipo de Pagamento (*PagamentoPix* ou *PagamentoCartao*). O código cliente pode chamar *processarDesconto*() sem se preocupar com o tipo concreto, pois o comportamento é garantido pelo contrato da classe base.

4. **Princípio da Segregação de Interface**
   Definição: “Clientes não devem ser forçados a depender de interfaces que não utilizam.”

O serviço que precisa apenas manipular pedidos não é forçado a depender de métodos de persistência de pagamento, mantendo as interfaces pequenas e focadas.

5. **Princípio da Inversão de Dependência**
   Definição:Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações.

EfetuarPedidoService (módulo de alto nível) utiliza apenas as interfaces de repositório (*PedidoRepository*, *PagamentoRepository*), sem conhecer os detalhes de persistência. As implementações reais desses repositórios são definidas externamente (pelo Spring Data JPA), fazendo com que os detalhes dependam das abstrações, e não o contrário.

Parte Individual – Gabriela

Componente: Custumer-domain

O componente customer-domain é responsável por toda a lógica de negócio relacionada à manutenção e validação dos dados dos clientes.

1. **Princípio da Responsabilidade Única**
   Definição: Uma classe deve ter apenas um motivo para mudar, ou seja, deve ter apenas uma responsabilidade.

A classe *ManterClienteService* tem a responsabilidade única de aplicar as regras de negócio e organizar o CRUD de clientes. A entidade *Cliente* foca apenas em seus dados e regras internas. A responsabilidade de persistência é delegada à interface *ClienteRepository*.

2. **Princípio do Aberto/Fechado**
   Definição: Entidades de software (classes, módulos, funções, etc.) devem ser abertas para extensão, mas fechadas para modificação.

Para expandir o cadastro com novas informações (ex: RG, histórico, preferências), basta estender o modelo com novas classes ou novos campos sem alterar o funcionamento interno do *Cliente*.

3. **Princípio da Substituição de Liskov**
   Definição: Se S é um subtipo de T, então objetos do tipo T podem ser substituídos por objetos do tipo S sem alterar as propriedades corretas do programa.

Qualquer implementação de: *ClienteRepository* e *EnderecoRepository* pode ser substituída por uma implementação alternativa sem quebrar a lógica de cadastro do cliente.

4. **Princípio da Segregação de Interface**
   Definição: “Clientes não devem ser forçados a depender de interfaces que não utilizam.”

O módulo define interfaces de repositório coesas e específicas (*ClienteRepository*). O serviço que precisa apenas manipular clientes não é forçado a depender de interfaces de outros domínios.

5. **Princípio da Inversão de Dependência**
   Definição:Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações.

O *ManterClienteService* (módulo de alto nível) utiliza apenas a interface *ClienteRepository*, sem conhecer os detalhes de persistência.

Parte Individual – Iago

Componente: Common-domain

O componente common-domain é um módulo de baixo nível que contém classes e interfaces compartilhadas por outros domínios, como classes base e utilitários.

1. **Princípio da Responsabilidade Única**
   Definição: Uma classe deve ter apenas um motivo para mudar, ou seja, deve ter apenas uma responsabilidade.

O módulo é responsável apenas por definir estruturas de dados e utilitários que são comuns a todos os domínios (ex: classes base para entidades). Ele não contém lógica de negócio específica de Livro, Pedido ou Cliente.

2. **Princípio do Aberto/Fechado**
   Definição: Entidades de software (classes, módulos, funções, etc.) devem ser abertas para extensão, mas fechadas para modificação.

O módulo é aberto para a adição de novas classes utilitárias que podem ser usados por outros domínios, mas fechado para modificação em suas classes base, garantindo a estabilidade e a não quebra dos módulos que o utilizam.

3. **Princípio da Substituição de Liskov**
   Definição: Se S é um subtipo de T, então objetos do tipo T podem ser substituídos por objetos do tipo S sem alterar as propriedades corretas do programa.

Quaisquer classes base ou interfaces definidas neste módulo devem garantir que seus subtipos sejam substituíveis sem quebrar o comportamento dos módulos que os utilizam.

4. **Princípio da Segregação de Interface**
   Definição: “Clientes não devem ser forçados a depender de interfaces que não utilizam.”

Cada classe utilitária representa uma responsabilidade específica, e nenhuma interface força dependências desnecessárias.

5. **Princípio da Inversão de Dependência**
   Definição:Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações.

Atua como a camada de abstração (baixo nível) para a qual os módulos de negócio (alto nível) dependem. Ao depender de classes e interfaces definidas no common-domain, os módulos de negócio evitam depender uns dos outros, promovendo a inversão de dependência e o baixo acoplamento entre os domínios.

3. Integração entre Componentes

- ## 3.1 Princípios de Coesão

Princípio REP: O projeto organiza o código em quatro módulos independentes (*book-domain, order-domain, customer-domain, common-domain*), garantindo que cada componente represente uma unidade de liberação e reutilização. Cada módulo possui lógica própria, bem delimitada, e pode ser versionado ou distribuído como um .jar independentemente dos demais.

Princípio CRP: Cada módulo só contém classes que devem ser reutilizadas juntas.
Exemplos:

- *book-domain* reúne apenas classes relacionadas ao catálogo.
- *order-domain* contém somente classes referentes a pedidos e seu ciclo de vida.
- *customer-domain* agrupa todas as classes relacionadas ao cliente.

Nenhum domínio importa código que não utiliza, evitando dependências desnecessárias.

Princípio CCP: Classes que mudam pelos mesmos motivos estão agrupadas no mesmo módulo.
Exemplos:

- Alterações nas regras de preço ou estoque afetam apenas o *book-domain*.
- Alterações nas regras de pedido afetam apenas o *order-domain*.
- Alterações cadastrais afetam apenas o *customer-domain*.

- ### 3.2 Princípios de Acoplamento

Princípio ADP: A arquitetura modular do projeto é estritamente hierárquica e unidirecional: *api-rest* depende dos módulos de domínio, e os módulos de domínio dependem do *common-domain*. Não há dependências circulares (ex: *book-domain* dependendo de *customer-domain* e *customer-domain* dependendo de *book-domain*), o que garante que o projeto possa ser construído e testado em ordem.

Princípio SDP: O módulo *common-domain* é o mais estável (poucas razões para mudar e alta dependência de outros módulos). Os módulos de domínio (*book-domain*, etc.) dependem dele. O módulo *api-rest* (o mais volátil, pois lida com a interface externa) depende dos módulos de domínio. A dependência flui do volátil para o estável.

Princípio SAP: O módulo mais estável, o *common-domain*, é composto principalmente por abstrações (classes base, interfaces) que são estendidas pelos módulos de domínio. Isso garante que, embora o *common-domain* seja difícil de mudar (por ser estável), ele é fácil de estender, pois os módulos clientes podem implementar suas abstrações.

4. Implementação Funcional dos Casos de Uso

CSU01 – Pesquisar Livro
Implementado no *book-domain* + exposto no *api-rest*.
Permite:

- listar livros

- buscar por ID
- consultar dados completos do livro

Funciona via *ManterLivroService* e *LivroController*.

CSU02 – Efetuar Pedido
Implementado no *order-domain*.
O pedido:

- recebe os itens
- calcula total
- valida estoque (via book-domain)
- atualiza estoque
- muda status para “EM PROCESSAMENTO, PAGAMENTO PENDENTE e CONFIRMADO”
- finaliza caso pagamento aprovado

Toda lógica central está em *Pedido*, *ItemPedido* e *ManterPedidoService*.

CSU03 – Manter Cliente
Implementado no *customer-domain*.
Permite:

- cadastrar cliente
- atualizar dados
- consultar informações
- manter endereço

Expõe operações via *ManterClienteService*.

CSU04 – Visualizar Pedido
No *order-domain,* permite consultar:

- pedido por ID
- status atual
- itens associados

Expõe as operações por *PedidoController* na API.

CSU05 – Manter Livro
Implementado no *book-domain*.
Permite:

- cadastrar livro
- atualizar livro
- excluir livro
- atualizar estoque

Tudo feito via *ManterLivroService*.

5. Diagrama de Componentes e Diagrama de Arquitetura de Software