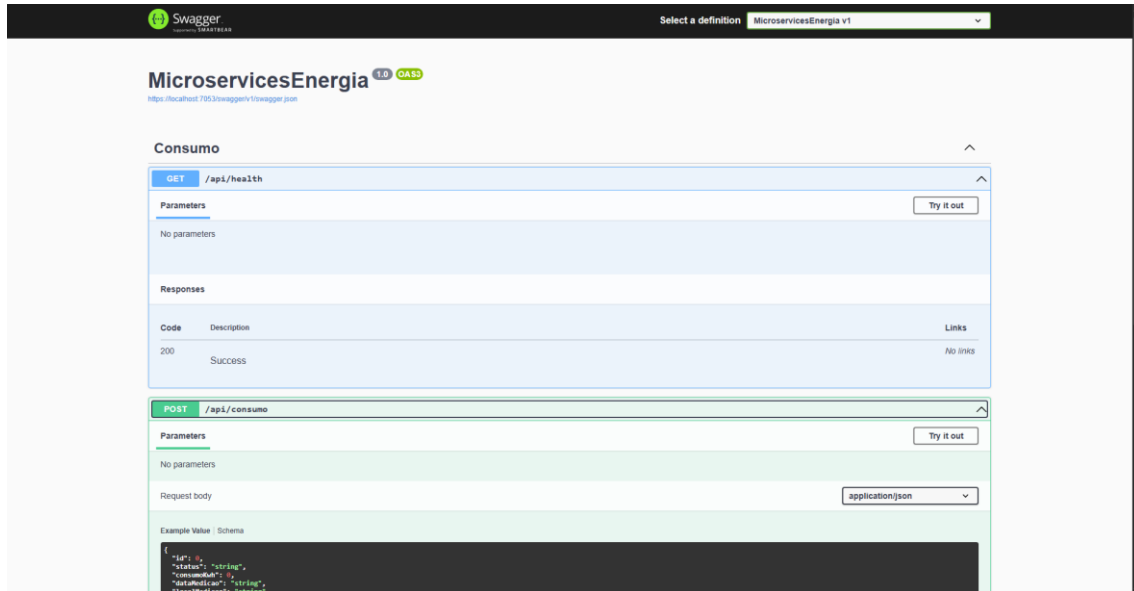


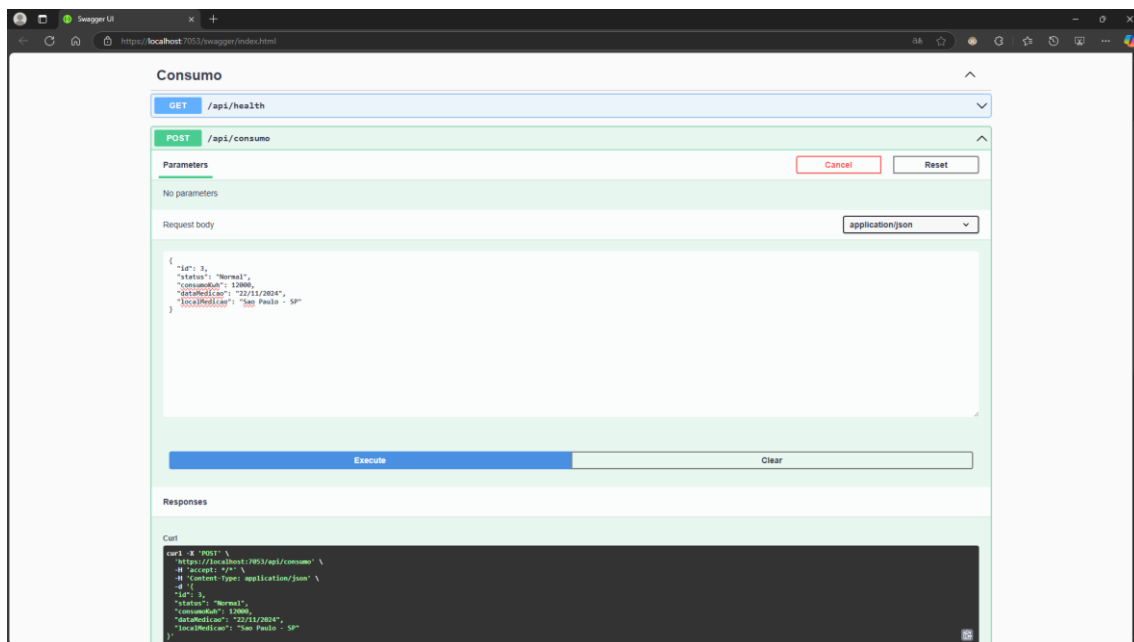
Documentação MicroServices

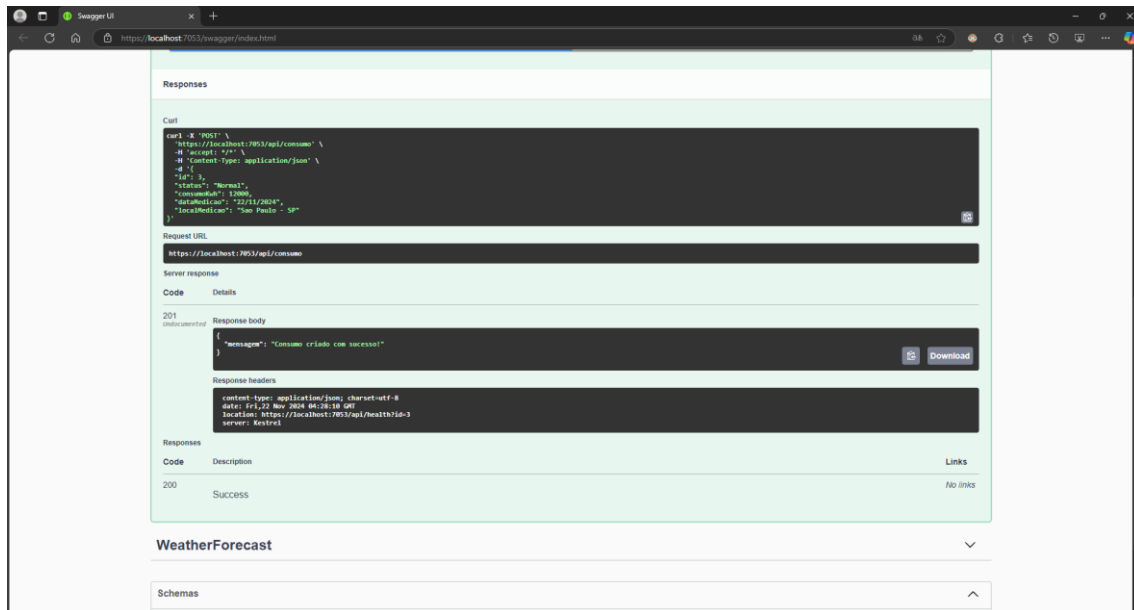
Planejamento e Estrutura do Microserviço



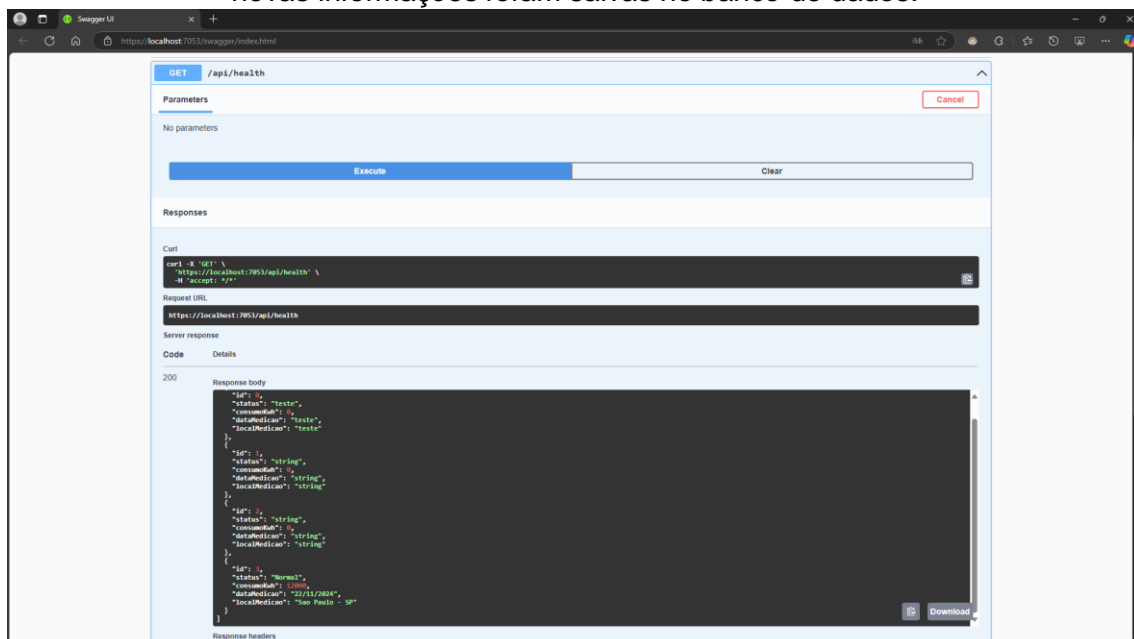
Utilizando o Swagger é possível identificar as rotas utilizadas na API, para o GET foi utilizado o “api/health” e para o POST “/api/consumo”.

Integração com MongoDB





Nas duas prints acima é capaz de observar a API retornando o código 201, indicando que o Consumo foi criado com sucesso e para comprovar que o MongoDB está corretamente interligado, pode-se testar o GET, como visto no print abaixo, e ver se as novas informações foram salvas no banco de dados.



Implementação de Cache com Redis

Para demonstrar a eficiência do Redis no sistema, foi criado duas novas rotas de GET, apenas para teste, uma utilizando cache e outra não e ao final do método é mostrado junto com as informações quantos milissegundos demorou a requisição.

GET /api/health-without-cache

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7053/api/health-without-cache' \
  -H 'accept: */*'
```

Request URL

https://localhost:7053/api/health-without-cache

Server response

Code Details

200

Response body

```
{
  "consumoKwh": 0,
  "dataMedicao": "teste",
  "localMedicao": "teste"
},
{
  "id": 1,
  "status": "string",
  "consumoKwh": 0,
  "dataMedicao": "string",
  "localMedicao": "string"
},
{
  "id": 2,
  "status": "string",
  "consumoKwh": 0,
  "dataMedicao": "string",
  "localMedicao": "string"
},
{
  "id": 3,
  "status": "Normal",
  "consumoKwh": 12000,
  "dataMedicao": "22/11/2024",
  "localMedicao": "Sao Paulo - SP"
}
},
"elapsedMilliseconds": 3
}
```

Response headers

No exemplo acima, sem cache, demorou 3 milissegundos para a requisição e a requisição com cache demorou cerca de 1 milissegundo

GET /api/health-with-cache

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7053/api/health-with-cache' \
  -H 'accept: */*'
```

Request URL

https://localhost:7053/api/health-with-cache

Server response

Code Details

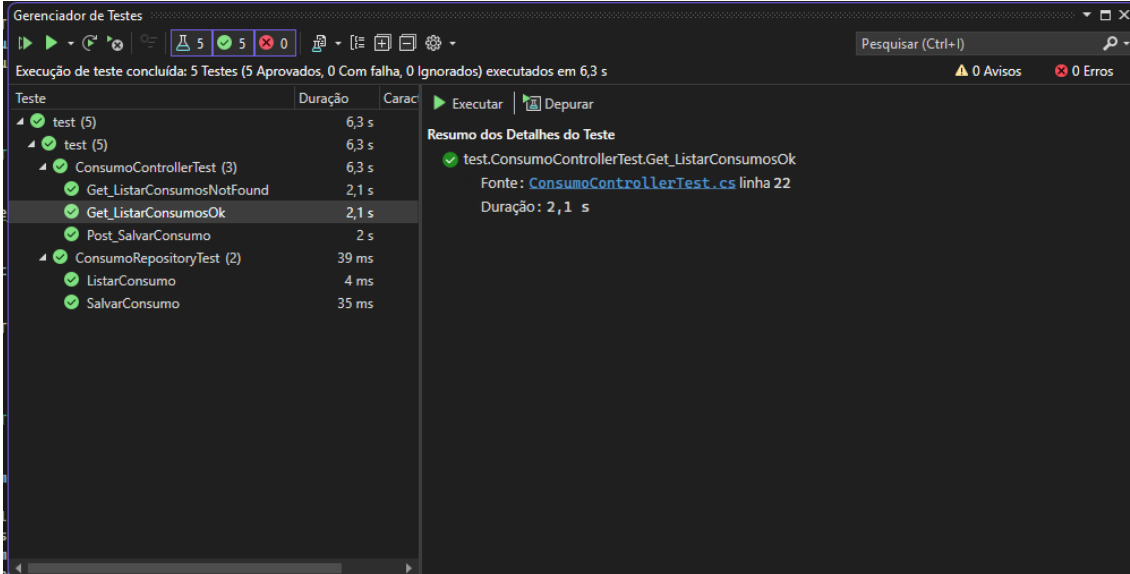
200

Response body

```
{
  "source": "cache",
  "data": [
    {
      "id": 0,
      "status": "teste",
      "consumoKwh": 0,
      "dataMedicao": "teste",
      "localMedicao": "teste"
    },
    {
      "id": 1,
      "status": "string",
      "consumoKwh": 0,
      "dataMedicao": "string",
      "localMedicao": "string"
    },
    {
      "id": 2,
      "status": "string",
      "consumoKwh": 0,
      "dataMedicao": "string",
      "localMedicao": "string"
    },
    {
      "id": 3,
      "status": "Normal",
      "consumoKwh": 12000,
      "dataMedicao": "22/11/2024",
      "localMedicao": "Sao Paulo - SP"
    }
  ]
},
"elapsedMilliseconds": 1
}
```

Testes Unitários com XUnit

Para comprovar que os 5 testes foram feitos, segue aprovação dos mesmos



The screenshot shows the 'Gerenciador de Testes' (Test Explorer) window in Visual Studio. The top status bar indicates 'Execução de teste concluída: 5 Testes (5 Aprovados, 0 Com falha, 0 Ignorados) executados em 6,3 s'. The main pane displays a tree view of tests, all of which are passed (green checkmarks). The right pane shows the 'Resumo dos Detalhes do Teste' (Test Details Summary) for the selected test, 'test.ConsumoControllerTest.Get_ListarConsumosOk', with a duration of 2,1 s.

Teste	Duração	Carac
test (5)	6,3 s	
test (5)	6,3 s	
ConsumoControllerTest (3)	6,3 s	
Get_ListarConsumosNotFound	2,1 s	
Get_ListarConsumosOk	2,1 s	
Post_SalvarConsumo	2 s	
ConsumoRepositoryTest (2)	39 ms	
ListarConsumo	4 ms	
SalvarConsumo	35 ms	

Resumo dos Detalhes do Teste

- test.ConsumoControllerTest.Get_ListarConsumosOk
Fonte: [ConsumoControllerTest.cs](#) linha 22
Duração: 2,1 s