Teste Hapvida .NET

> **Prova técnica — Dev Backend (.NET)**
>
> Objetivo: construir uma **HTTP API em .NET 8** que integra **APIs públicas** (CEP + clima), com foco em **boas práticas**, **resiliência**, **testes** e **documentação**.
>
> Personas nas US: **Usuário**, **Integrador**, **Operador** (SRE/DevOps), **Mantenedor** (Dev), **Avaliador**.

Escopo e Regras Gerais

- **Tempo sugerido:** 4–8 horas.
- **Stack sugerida:** .NET 8 (C#) com **Controllers**.
- **APIs públicas (sem chave):**
  - CEP primária: **BrasilAPI CEP v2** (`https://brasilapi.com.br/api/cep/v2/{cep}`)
  - CEP fallback: **ViaCEP** (`https://viacep.com.br/ws/{cep}/json/`)
  - Clima primária: **Open-Meteo Forecast** (`https://api.open-meteo.com/v1/forecast?...`)
  - Geocodificação (se necessário): **Open-Meteo Geocoding** (`https://geocoding-api.open-meteo.com/v1/search?...`)
- **Persistência para a prova:** usar **banco em memória**. Recomendado **SQLite in-memory** ou **MongoDB in-memory**.
- **Entrega:** Repositório público no **GitHub**.
- **O projeto rodando no Docker é OBRIGATÓRIO**.
- **Padrões de erro: RFC 7807 (Problem Details)**.

US01 — Consulta de CEP

Como um Usuário, quero informar um CEP válido e receber um endereço normalizado, para localizar a cidade/UF e dados correlatos.

CA:

- Rota `GET /cep/{zipCode}` (aceitar com/sem hífen; normalizar para **8 dígitos**).
- CEP inválido → **400** (Problem Details).
- Consultar **BrasilAPI CEP v2** como primária.
- Em falha/indisponibilidade, **fallback** para **ViaCEP**.
- Responder **200** com JSON normalizado contendo, quando disponível:

```
{
  "zipCode": "01001000",
  "street": "Praça da Sé",
  "district": "Sé",
  "city": "São Paulo",
  "state": "SP",
  "ibge": "3550308",
  "location": { "lat": -23.5507, "lon": -46.6334 },
  "provider": "brasilapi|viacep"
}
```

- Não encontrado em nenhum provedor → **404** (Problem Details).

NI:

- Mapear campos distintos dos provedores para **um DTO único**.
- Logar qual provedor atendeu (auditoria).

US02 — Persistir CEP (reaproveitando lógica da US01)

Como um Usuário, quero enviar um CEP para que o sistema consulte e salve o resultado em um banco em memória, para reutilizar depois sem precisar informar o CEP novamente.

CA:

- Rota `POST /cep`
- **Request (JSON):**

```
{ "zipCode": "01001000" }
```

- Validar CEP (8 dígitos). Inválido → **400** (Problem Details).
- **Reutilizar o serviço da US01** para consultar CEP.
- Persistir em tabela `ZipCodeLookups` (exemplo de colunas):
  - `Id` (GUID/INT), `ZipCode`, `Street`, `District`, `City`, `State`, `Ibge`, `Lat`, `Lon`, `Provider`, `CreatedAtUtc`.
- Responder **201 Created** com corpo do recurso salvo.
- Não permitir **entradas repetidas**.

Exemplo de resposta 201:

```
{
  "id": 42,
```

```
  "zipCode": "01001000",
  "street": "Praça da Sé",
  "district": "Sé",
  "city": "São Paulo",
  "state": "SP",
  "ibge": "3550308",
  "location": { "lat": -23.5507, "lon": -46.6334 },
  "provider": "brasilapi",
  "createdAtUtc": "2025-09-19T12:34:56Z"
}
```

NI:

- **Banco:** preferir **SQLite in-memory** ou **MongoDB in-memory**.

US03 — Clima atual e previsão usando os CEPs salvos

Como um Usuário, quero consultar o clima atual e a previsão para N dias sem informar CEP, para que o sistema use os CEPs salvos.

CA:

- Rota `GET /weather?days=3` (**sem parâmetro de CEP**).
- `days` padrão **3**; aceitar **1–7** (fora disso → **400**).
- Buscar no banco **todos os registros de CEP** (`CreatedAtUtc DESC`).
  - Se não houver nenhum CEP persistido → **404** com Problem Details (`no-saved-cep`).
- Usar `Lat/Lon` dos registros salvos.
  - Se não houver `Lat/Lon`, **geocodificar** por `city + state` (Open-Meteo Geocoding).
- Consultar **Open-Meteo Forecast** para **clima atual** e **diário** (mín/máx).
- **Cache (memória)** por `weather:{lat}:{lon}:{days}` ou `weather:{city}:{state}:{days}` com **TTL de 10 minutos**, para respostas rápidas e economia de recursos.
- Responder **200** com JSON:

```
[{
  "sourceZipCodeId": 42,
  "location": { "lat": -23.5507, "lon": -46.6334, "city": "São Paulo",
"state": "SP" },
  "current": {
    "temperatureC": 24.3,
    "humidity": 0.62,
    "apparentTemperatureC": 25.1,
    "observedAt": "2025-09-19T12:35:00Z"
  },
```

```
  "daily": [
    { "date": "2025-09-19", "tempMinC": 18.0, "tempMaxC": 27.0 },
    { "date": "2025-09-20", "tempMinC": 19.0, "tempMaxC": 28.0 },
    { "date": "2025-09-21", "tempMinC": 17.0, "tempMaxC": 26.0 }
  ],
  "provider": "open-meteo"
}]
```

NI:

- `IMemoryCache` com chave `weather:{lat}:{lon}:{days}` ou `weather:{cidade}:{uf}:{days}` (TTL 10 min).
- Manter datas em ISO 8601 (UTC). Converter unidades conforme necessário.

US04 — Execução local e Docker (OBRIGATÓRIO)

Como um Mantenedor, quero rodar a API localmente e em container, para padronizar o ambiente.

CA:

- **Dockerfile** multi-stage com imagem final reduzida (ex.: `mcr.microsoft.com/dotnet/aspnet:8.0`).

NI:

- Documentar portas e variáveis de ambiente no `README.md`.

US05 — Validação & Mensagens de erro

Como um Integrador, quero receber mensagens de erro consistentes e legíveis, para tratar falhas de forma padronizada.

CA:

- Usar **RFC 7807** em todos os erros: `type`, `title`, `status`, `detail`, `traceId`.
- Exemplos:
  - **400** CEP inválido:

```
{
  "type": "https://errors.suaapi/invalid-cep",
  "title": "CEP inválido",
  "status": 400,
  "detail": "CEP deve conter 8 dígitos.",
  "traceId": "00-...-..."
}
```

  - **404** CEP não encontrado (US01) ou **nenhum CEP salvo** (US04).

- **504** timeout de provedor externo após política de resiliência.

NI:

- Middleware global para exceções + normalização dos Problem Details.

US06 — Resiliência a falhas externas

Como um Operador, quero que chamadas a provedores externos sejam resilientes, para evitar quedas por intermitências.

CA:

- **Timeouts** explícitos por `HttpClient` (ex.: 2–5s por tentativa).
- **Retry** com backoff + jitter (ex.: 3 tentativas para erros transitórios).
- **Circuit Breaker** (abre após X falhas/Y s; half-open e fecha se OK).
- Logar tentativas e estado do circuito.

NI:

- `Microsoft.Extensions.Http.Resilience` (recomendado) ou `Polly`.

US07 — Observabilidade (Logs, Correlação, Métricas)

Como um Operador, quero logs estruturados, correlação de requisições e métricas básicas, para monitorar a API em produção.

CA:

- Logs estruturados (JSON) com `traceId` em cada request.
- Métricas mínimas (requisições por rota, latência).

US08 — Documentação via OpenAPI/Swagger

Como um Integrador, quero um catálogo OpenAPI com exemplos, para consumir a API sem ambiguidade.

CA:

- **Swagger UI** em `/swagger`.
- Schemas de request/response com exemplos e códigos de status.

NI:

- `Swashbuckle.AspNetCore` + exemplos inline nos DTOs.

US9 — Testes (Unitários e Integração)

Como um Mantenedor, quero testes automatizados cobrindo regras críticas, para evoluir com segurança.

CA:

- **Unitários** cobrindo:
  - Normalização/mapeamento de CEP.
  - Fallback de provedores.
  - Serviço de **persistência** (US03).
  - Mapeamento de **clima** (current + diário).
  - Validações (CEP e `days`).

NI:

- **Faker*** para mocks.
- **xUnit** para testes unitários.

Contratos de API (Resumo)

`GET /cep/{cep}` *(US01)*

200

```
{
  "zipCode": "01001000",
  "street": "Praça da Sé",
  "district": "Sé",
  "city": "São Paulo",
  "state": "SP",
  "ibge": "3550308",
  "location": { "lat": -23.5507, "lon": -46.6334 },
  "provider": "brasilapi"
}
```

400/404 → Problem Details (RFC 7807)

`POST /cep` *(US02)*

Request

```
{ "zipCode": "01001000" }
```

201 Created

```
{
  "id": 42,
```

```
  "zipCode": "01001000",
  "street": "Praça da Sé",
  "district": "Sé",
  "city": "São Paulo",
  "state": "SP",
  "ibge": "3550308",
  "location": { "lat": -23.5507, "lon": -46.6334 },
  "provider": "brasilapi",
  "createdAtUtc": "2025-09-19T12:34:56Z"
}
```

400/404 → Problem Details (RFC 7807)

`GET /weather?days=3` *(US03 — sem CEP)*

200

```
[{
  "sourceCepId": 42,
  "location": { "lat": -23.5507, "lon": -46.6334, "city": "São Paulo", "state":
"SP" },
  "current": {
    "temperatureC": 24.3,
    "humidity": 0.62,
    "apparentTemperatureC": 25.1,
    "observedAt": "2025-09-19T12:35:00Z"
  },
  "daily": [
    { "date": "2025-09-19", "tempMinC": 18.0, "tempMaxC": 27.0 },
    { "date": "2025-09-20", "tempMinC": 19.0, "tempMaxC": 28.0 },
    { "date": "2025-09-21", "tempMinC": 17.0, "tempMaxC": 26.0 }
  ],
  "provider": "open-meteo"
}]
```

404 → nenhum CEP salvo. **400/504** → validações/timeout (Problem Details).

Exemplos de uso (curl)

```
# CEP (consulta simples, sem persistir)
curl -s http://localhost:5000/cep/01001000

# Persistir CEP (usa a mesma lógica da US02 e grava no banco em memória)
curl -s -X POST http://localhost:5000/cep \
  -H "Content-Type: application/json" \
  -d '{"zipCode":"01001000"}'
```

```
# Clima usando os CEPs salvos (sem passar CEP)
curl -s "http://localhost:5000/weather?days=3"
```