**Você disse:**

estou fazendo um programacao de uma busca por campo id, usando cypress, quero que vc analise se o codigo esta correto

**O ChatGPT disse:**

Claro, pode me enviar o código que você fez para a busca por campo ID no Cypress que eu analiso linha por linha e aponto possíveis melhorias ou erros.

**Você disse:**

it("Buscar usuario por ID ", () => { cy.request({ method: 'GET', url: 'https://serverest.dev/usuarios/${Cypress.env(id)}' }).then((response) => { expect(response.status).to.eq(200)

**O ChatGPT disse:**

Olhei seu código e tem um detalhe que vai impedir a interpolação do valor do **ID**: você está usando aspas simples ' ' na URL, e dentro delas a sintaxe ${...} não funciona.

No JavaScript, para interpolar variáveis dentro de strings, você precisa usar **template literals** com **crases** `.  
Além disso, o Cypress.env() deve receber o nome da variável como string normal, sem crase extra.

O ajuste ficaria assim:

it("Buscar usuário por ID", () => {

cy.request({

method: 'GET',

url: `https://serverest.dev/usuarios/${Cypress.env('id')}`

}).then((response) => {

expect(response.status).to.eq(200)

})

})

📌 **Pontos importantes:**

1. **Troque as aspas simples por crases** para usar ${}.
2. **Cypress.env('id')** não precisa de crase dentro do argumento.
3. Verifique se a variável id realmente está setada no cypress.env.json ou via comando (--env id=valor), senão a URL ficará inválida.
4. Caso queira validar também o corpo da resposta, você pode adicionar algo como:
5. expect(response.body).to.have.property('\_id', Cypress.env('id'))

Se quiser, posso te mostrar também **como garantir que o teste falhe caso o ID não exista**. Isso deixaria seu teste mais robusto. Quer que eu inclua essa verificação?

**Você disse:**

perfeito, deu certo mesmo, voce estava correto

**O ChatGPT disse:**

Ótimo! 😄  
Esse tipo de erro com crases vs. aspas é bem comum, principalmente quando estamos usando interpolação em Cypress ou JavaScript puro.

Se você quiser, posso te passar um **modelo de boas práticas para testes de API no Cypress**, incluindo:

* Tratamento de erros
* Validação de schema da resposta
* Reuso de variáveis de ambiente

Assim seu teste já fica mais organizado e pronto para crescer sem quebrar fácil.

Quer que eu monte esse modelo pra você?

**Você disse:**

quero sim

**O ChatGPT disse:**

Beleza 😄  
Vou te montar um modelo de **boas práticas para teste de API com Cypress**, usando busca por ID como exemplo.

**Modelo de teste de API no Cypress (com boas práticas)**

/// <reference types="cypress" />

describe('Testes de API - Usuários', () => {

const baseUrl = 'https://serverest.dev/usuarios';

it('Buscar usuário por ID', () => {

const userId = Cypress.env('id');

// 🔹 Validação se a variável está definida

expect(userId, 'ID do usuário não está definido no Cypress.env').to.not.be.undefined;

cy.request({

method: 'GET',

url: `${baseUrl}/${userId}`,

failOnStatusCode: false // Evita que o teste quebre antes de validar o erro

}).then((response) => {

// 🔹 Valida status code

expect(response.status, 'Status code inesperado').to.eq(200);

// 🔹 Valida formato e dados obrigatórios

expect(response.body).to.have.property('\_id', userId);

expect(response.body).to.have.property('nome').and.to.be.a('string');

expect(response.body).to.have.property('email').and.to.be.a('string');

});

});

});

**Boas práticas aplicadas aqui**

1. **Variáveis de ambiente**
   * O ID não fica hardcoded, é lido do cypress.env.json ou passado via CLI:
   * npx cypress run --env id=abc123
2. **Validação de variáveis antes do teste**
   * Se Cypress.env('id') não estiver setado, o teste já acusa erro com uma mensagem clara.
3. **failOnStatusCode: false**
   * Evita que o Cypress encerre o teste antes que você valide o motivo do erro (bom para testes negativos).
4. **Validação do body**
   * Garante que o retorno tenha os campos esperados, não apenas o status 200.
5. **Separação de constantes**
   * baseUrl definido no topo evita repetição e facilita manutenção.

Se quiser, posso te adicionar **um teste negativo** também, pra garantir que se o ID não existir, a API retorne status e mensagem corretos.  
Assim seu conjunto de testes já cobre **sucesso e erro**.

Quer que eu já inclua esse teste negativo?

**Você disse:**

sim, por favor

**O ChatGPT disse:**

Perfeito 😄  
Aqui vai a versão **com teste positivo e negativo** para busca por ID no Cypress:

/// <reference types="cypress" />

describe('Testes de API - Usuários', () => {

const baseUrl = 'https://serverest.dev/usuarios';

it('Buscar usuário por ID existente', () => {

const userId = Cypress.env('id');

// 🔹 Validação de variável

expect(userId, 'ID do usuário não está definido no Cypress.env').to.not.be.undefined;

cy.request({

method: 'GET',

url: `${baseUrl}/${userId}`,

failOnStatusCode: false

}).then((response) => {

// ✅ Validação positiva

expect(response.status, 'Status code inesperado').to.eq(200);

expect(response.body).to.have.property('\_id', userId);

expect(response.body).to.have.property('nome').and.to.be.a('string');

expect(response.body).to.have.property('email').and.to.be.a('string');

});

});

it('Buscar usuário por ID inexistente', () => {

const fakeId = '999999999999999999999999'; // ID que não existe no banco

cy.request({

method: 'GET',

url: `${baseUrl}/${fakeId}`,

failOnStatusCode: false // 🔹 Permite testar o retorno de erro

}).then((response) => {

// ❌ Validação negativa

expect(response.status, 'Status esperado para recurso não encontrado').to.eq(400);

expect(response.body).to.have.property('message').and.to.be.a('string');

expect(response.body.message).to.match(/Usuário não encontrado/i);

});

});

});

**O que esse teste cobre agora**

1. **Caso de sucesso** (ID válido) → Garante que retorna 200 e os dados esperados.
2. **Caso de falha** (ID inexistente) → Garante que retorna status 400 e mensagem adequada.
3. **Resiliência** → Não quebra no meio, mesmo que o servidor retorne erro.
4. **Mensagens de erro claras** → Quando falha, o motivo aparece no próprio log do teste.

Se quiser, eu também posso montar **uma versão com schema validation usando AJV** para garantir que a estrutura da resposta da API nunca mude sem você perceber. Isso é ótimo pra QA de API.

Quer que eu já faça essa versão com validação de schema?

**Você disse:**

por favor

**O ChatGPT disse:**

Então vamos deixar seu teste **à prova de mudanças de contrato de API** usando **validação de schema com AJV** 😄

**Exemplo com Schema Validation no Cypress**

Aqui vamos validar não só os valores, mas também a **estrutura da resposta**.  
Isso detecta mudanças inesperadas no formato da API.

/// <reference types="cypress" />

import Ajv from 'ajv';

describe('Testes de API - Usuários com Validação de Schema', () => {

const baseUrl = 'https://serverest.dev/usuarios';

const ajv = new Ajv({ allErrors: true, strict: false }); // strict:false evita falha com campos extras

// 🔹 Schema esperado para um usuário

const userSchema = {

type: 'object',

required: ['\_id', 'nome', 'email', 'password', 'administrador'],

properties: {

\_id: { type: 'string' },

nome: { type: 'string' },

email: { type: 'string', format: 'email' },

password: { type: 'string' },

administrador: { type: 'string' }

},

additionalProperties: true // permite campos extras sem falhar

};

it('Buscar usuário por ID existente - Validação de Schema', () => {

const userId = Cypress.env('id');

expect(userId, 'ID do usuário não está definido no Cypress.env').to.not.be.undefined;

cy.request({

method: 'GET',

url: `${baseUrl}/${userId}`,

failOnStatusCode: false

}).then((response) => {

expect(response.status).to.eq(200);

// 🔹 Valida o schema

const validate = ajv.compile(userSchema);

const valid = validate(response.body);

if (!valid) {

throw new Error(`Erros de schema: ${JSON.stringify(validate.errors, null, 2)}`);

}

});

});

it('Buscar usuário por ID inexistente - Validação de Schema de erro', () => {

const fakeId = '999999999999999999999999';

// 🔹 Schema esperado para erro

const errorSchema = {

type: 'object',

required: ['message'],

properties: {

message: { type: 'string' }

},

additionalProperties: true

};

cy.request({

method: 'GET',

url: `${baseUrl}/${fakeId}`,

failOnStatusCode: false

}).then((response) => {

expect(response.status).to.eq(400);

// 🔹 Valida o schema do erro

const validate = ajv.compile(errorSchema);

const valid = validate(response.body);

if (!valid) {

throw new Error(`Erros de schema: ${JSON.stringify(validate.errors, null, 2)}`);

}

});

});

});

**O que melhoramos aqui**

✅ **Proteção contra mudanças no contrato da API** — Se a estrutura mudar, o teste falha imediatamente.  
✅ **Uso de Ajv** — Biblioteca rápida e confiável para validação JSON Schema.  
✅ **Testes positivos e negativos** com validação de formato e status code.  
✅ **Erros claros** — Se o schema falhar, a mensagem no log do Cypress explica exatamente qual campo está errado.

Se você quiser, eu posso também te preparar **um userSchema.json separado** e importar no teste, para evitar código repetido e deixar a manutenção mais fácil.

Quer que eu já te faça essa versão separando o schema num arquivo JSON?

**Você disse:**

muito obrigado, por enquanto nao

**O ChatGPT disse:**

Tranquilo 😄  
O importante é que agora seu teste está muito mais robusto, cobrindo:

* Busca por ID válido
* Busca por ID inválido
* Validação de **status**, **mensagem** e **estrutura da resposta**

Se no futuro você quiser separar o schema num arquivo .json ou criar testes dinâmicos para vários IDs, é só me chamar.

Boa sorte com seus testes de API 🚀