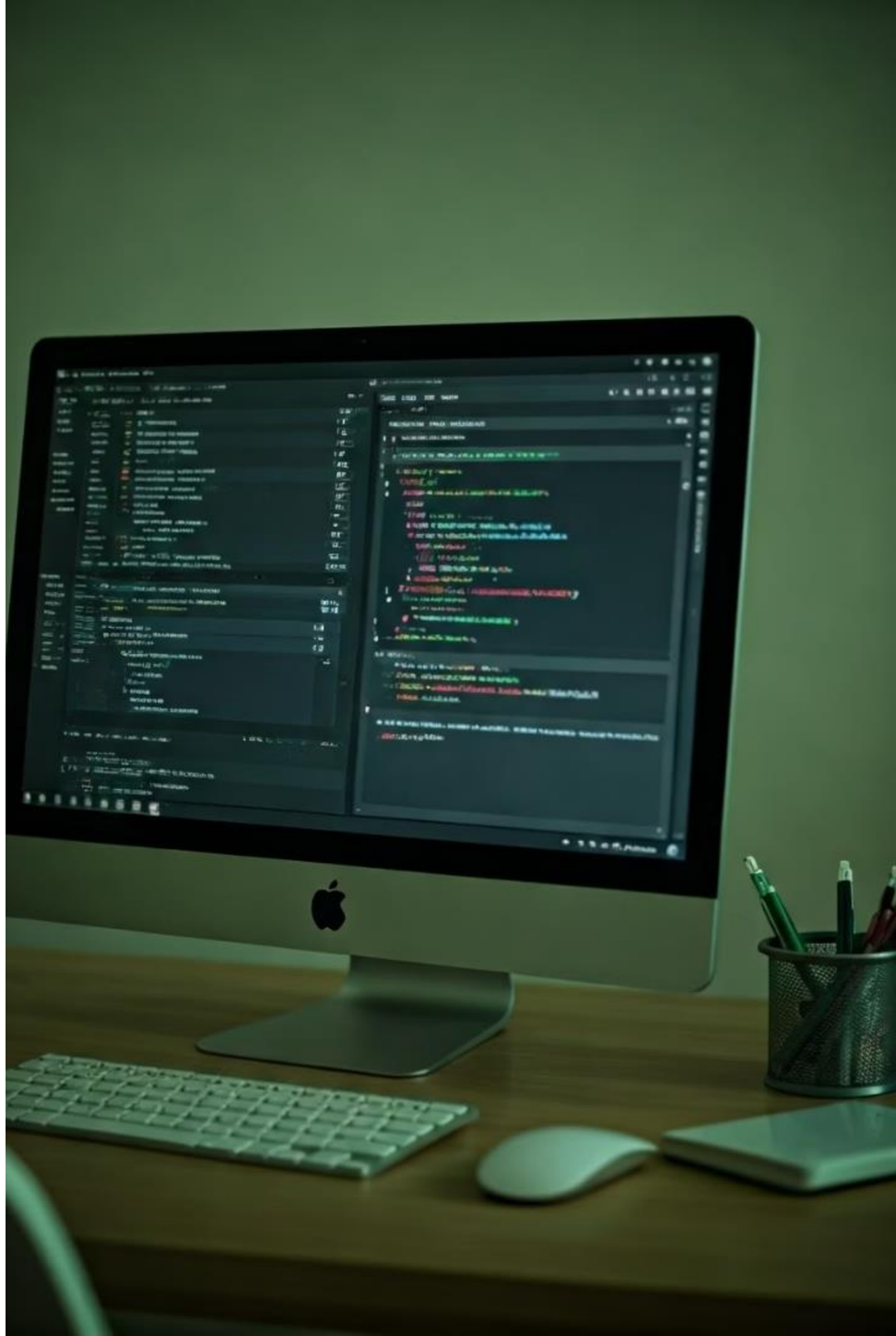
Teste-na-prática: API REST com Node.js e node:test



Arquitetura e Funcionalidades da API

Estrutura com Express

Configuração leve e organizada para gerenciamento eficiente.

Rotas principais

/GET - Lista livros

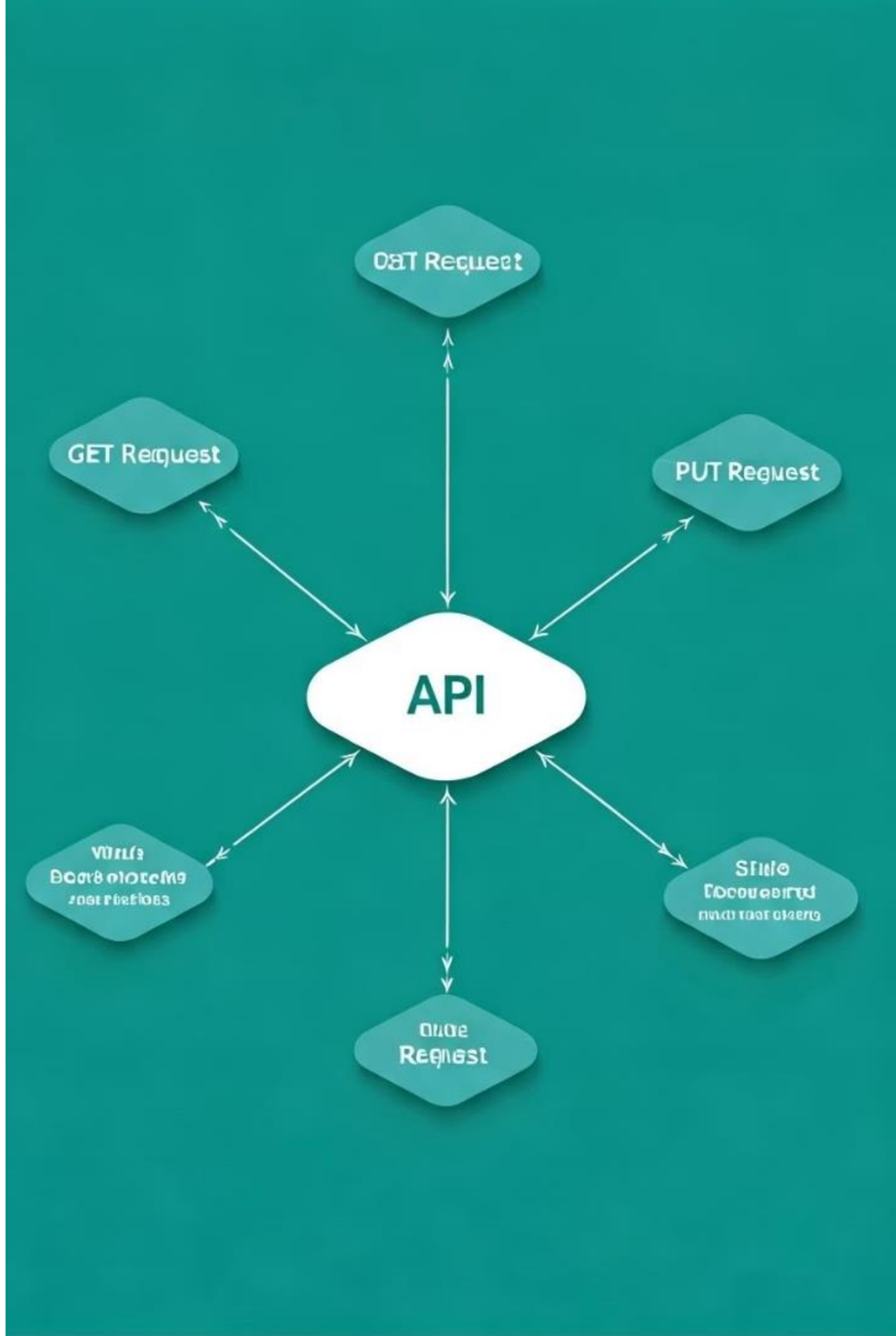
/POST - Cria livro

/PUT - Atualiza livro

/DELETE - Remove livro

Exemplo de payload

{"titulo": "Dom Quixote", "autor": "Miguel de Cervantes"}



Testando com node:test

Simplicidade nativa

node:test usa funções test() e assert para validar funcionalidades.

Validação de rotas

Exemplos práticos para GET, POST, PUT e DELETE com status checks.

Exemplos

```
test('SETUP - iniciar servidor', async (t) => {
  server = app.listen(3000);
  controller.resetBooks();
});
      You, 15 minutes ago • inital commit
test('Criar livro', async () => {
  const res = await makeRequest({
    method: 'POST',
    path: '/',
    data: { title: '1984', author: 'George Orwell' },
  });

  assert.strictEqual(res.status, 201);
  assert.strictEqual(res.body.title, '1984');
});
```

```
test('Editar livro', async () => {
  const res1 = await makeRequest({
    method: 'POST',
    path: '/',
    data: { title: 'Original', author: 'Autor' },
  });

  const bookId = res1.body.id;

  const res2 = await makeRequest({
    method: 'PUT',
    path: `/${bookId}`,
    data: { title: 'Atualizado', author: 'Autor X' },
  });

  assert.strictEqual(res2.status, 200);
  assert.strictEqual(res2.body.title, 'Atualizado');
});
```

Prós e contras do node:test

Vantagens

- Simplicidade e fácil integração
- Sem dependências externas
- Controle manual das configurações

Desvantagens

- Funcionalidades limitadas comparado a Jest e Mocha
- Setup mais manual para servidor e requisições
- Sem modo watch nativo para reexecutar testes
- Cobertura de testes (coverage)