Mocks em Apex

Garantindo Testes Confiáveis e Independentes

Agenda

01

Introdução

02

Ferramentas de mock

03

Mocks de integração

04

StubProvider

05

Boas práticas

06

Links úteis

01

Introdução

A importância dos mocks

Testes unitários

Permite que testes verifiquem apenas o comportamento da unidade em teste

Acomplamento

Minimiza as dependências concretas nos testes, aumentando a resiliencia contra mudanças

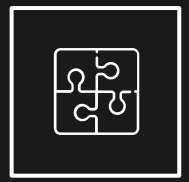
Deploy

Evita falhas de testes durante deploys devido à dependências externas ao teste

A importância dos mocks

Cobertura

Permite testar cenários que seriam difíceis de acionar com implementações reais



Design

Encoraja criação de interfaces bem definidas e melhor encapsulamento ao exigir que pense sobre as interações entre componentes

Velocidade

Aumenta a velocidade dos testes pois elimina a necessidade de chamar o banco de dados

02

Ferramentas de mock

Classes e Interfaces

Recurso	Finalidade	Particularidade
WebServiceMock	Integrações SOAP	Trabalha no nível de objeto de resposta gerado pelo WSDL2Apex
HttpCalloutMock	Integrações que utilizam o protocolo HTTP	Trabalha diretamente com HTTP, simulando corpo, status e cabeçalho
StaticResourceCalloutMock	Chamadas para 1 API	Implementação do HttpCalloutMock. É necessário 1 recurso estático como resposta
MultiStaticResourceCalloutMock	Chamadas para múltiplas API's	O mesmo que StaticResourceCalloutMock, mas necessita de vários recursos estáticos e definir endpoints diretamente
StubProvider	Mock de classes	Necessita criar implementação concreta da interface

03

Mocks de integração

WebServiceMock

WebServiceMock é uma interface que permite simular respostas de serviços SOAP (gerados a partir de um WSDL) durante testes.

- **Específico** : Apenas para classes geradas pelo WSDL2Apex;
- **Alto nível** : Trabalha com objetos de respostas tipados;
- **Resposta estruturada** : Você configura objetos de resposta específicos do serviço.

WebServiceMock

```
  private class NumbersInWordsSOAPRequestMock1 implements WebServiceMock {
    public void doInvoke(
      Object stub,
      Object request,
      Map<String, Object> response,
      String endpoint,
      String soapAction,
      String requestName,
      String responseNS,
      String responseName,
      String responseType
    ) {
      // Configuração do objeto de resposta específico do serviço
      wwwDataaccessComWebservicesserver.NumberToWordsResponse_element response_x = new
wwwDataaccessComWebservicesserver.NumberToWordsResponse_element();
      response_x.NumberToWordsResult = 'one hundred and twenty three ';
      response.put('response_x', response_x);
    }
  }

//Test.setMock(WebServiceMock.class, new NumbersInWordsSOAPRequestMock1());
```

HttpCalloutMock

HttpCalloutMock é uma interface que permite simular qualquer requisição HTTP durante os testes.

- **Universal** : Para qualquer chamada HTTP;
- **Baixo nível** : Trabalha com objetos HTTP;
- **Resposta manual** : Você configura a resposta HTTP manualmente.

HttpCalloutMock

```
   private class ManufacturerJSONMock implements HttpCalloutMock {
    public HttpResponse respond(HttpRequest req) {
      HttpResponse res = new HttpResponse();
      res.setStatusCode(200);
      res.setHeader('Content-Type', 'application/json');
      res.setBody(
        '{"Results":[' +
          '{"Mfr_ID":987,"Country":"UNITED STATES (USA)","Mfr_Name":"TESLA,
INC.","Mfr_CommonName":"TESLA"},' +
          '{"Mfr_ID":956,"Country":"GERMANY","Mfr_Name":"VOLKSWAGEN
AG","Mfr_CommonName":"VOLKSWAGEN"}' +
          '],"Count":2,"Message":"Results returned successfully"}'
      );
      return res;
    }
  }

//Test.setMock(HttpCalloutMock.class, new ManufacturerJSONMock());
```

StaticResourceCalloutMock

StaticResourceCalloutMock é uma implementação da interface HttpCalloutMock que permite usar recursos estáticos como respostas para as chamadas HTTP

- **Reutilização** : Usa arquivos armazenados como recursos estáticos para simular respostas;
- **Conveniente** : Ideal para respostas grandes ou complexas;
- **Manutenção simplificada** : Separa os dados de teste do código de teste.

StaticResourceCalloutMock

```
StaticResourceCalloutMock mock = new StaticResourceCalloutMock();
mock.setStaticResource('xml_numbers_in_words_response_example');
mock.setStatusCode(200);
mock.setHeader('Content-Type', 'text/xml; charset=utf-8');
Test.setMock(HttpCalloutMock.class, mock);
```

MultiStaticResourceCalloutMock

MultiStaticResourceCalloutMock amplia a funcionalidade do StaticResourceCalloutMock permitindo mapear múltiplos recursos estáticos para diferentes endpoints

- **Versatilidade** : Gerencia múltiplos endpoints em um único mock;
- **Específico para URI** : Associa diferentes recursos para diferentes URIs;
- **Organização** : Facilita testes que envolvem múltiplas chamadas HTTP.

MultiStaticResourceCalloutMock

```
MultiStaticResourceCalloutMock multiMock = new MultiStaticResourceCalloutMock();
multiMock.setStaticResource('https://endpoint1.com', 'Recurso_Estatico_1');
multiMock.setStaticResource('https://endpoint2.com', 'Recurso_Estatico_2');
multiMock.setStaticResource('https://endpoint3.com', 'Recurso_Estatico_3');
multiMock.setStatusCode(200);
multiMock.setHeader('Content-Type', 'text/xml');
Test.setMock(HttpCalloutMock.class, multiMock);
```

04

StubProvider

StubProvider

StubProvider é uma interface genérica pertencente à Apex Stub API que permite criar uma biblioteca de mocking para classes Apex

- **Universal** : Pode criar mocks para qualquer classe;
- **Flexibilidade total** : Controle detalhado sobre métodos e comportamentos;
- **Programático** : Define comportamentos através de código em vez de recursos estáticos;
- **Avançado** : Mais poderoso, porém requer mais configuração manual.

StubProvider

É necessário atentar-se às limitações do Apex Stub API.

Não é possível criar mocks nos seguintes cenários:

- Métodos estáticos e/ou privados;
- Propriedades (getters e setters);
- Triggers;
- Classes internas (sub-classes);
- Tipos de sistema;
- Classes que implementam a interface *Batchable;*
- Classes que tem apenas construtores privados.

StubProvider

Outros pontos de atenção:

- Os objetos sendo mockados precisam estar no mesmo namespace da chamada de Test.createStub, mas as implementações da *StubProvider* podem estar em outro;
- Não é possível definir um *iterator* como retorno ou tipo de parâmetro.

StubProvider

```apex
global interface StubProvider {
  Object handleMethodCall(
    Object param0, // stubbedObject
    String param1, // stubbedMethodName
    System.Type param2, // returnType
    List<System.Type> param3, // listOfParamTypes
    List<String> param4, // listOfParamNames
    List<Object> param5 // listOfArgs
  );
}
```

Alguns exemplos sólidos de implementação da interface

- **Stub**: Uma implementação mais simples, porém bastante funcional e de fácil manutenção;
- **Apex Mockery**: Implementação mais robusta inspirada em Mockito, chai.js, sinon.js e jest.

05

Boas práticas

Boas práticas

Sobre Design de código

- Evite classes puramente estáticas;
- Utilize injeção de dependência;
- Crie interfaces para serviços externos;
- Separe as responsabilidades.

Boas práticas

Sobre Mocking

- Crie mocks reutilizáveis entre testes;
- Teste os limites do sistema;
- Verifique o comportamento dos mocks.

06

Links úteis

Links úteis

StubProvider

SALESFORCE. **Salesforce Developers** . Disponível em: <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_testing_stub_api.htm>. Acesso em: 11 mar. 2025.

SALESFORCE. **Salesforce Developers** . Disponível em: <https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_interface_System_StubProvider.htm>. Acesso em: 11 mar. 2025.

Links úteis

SOAP

SALESFORCE. **Salesforce Developers** . Disponível em: <https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_interface_webservicemock.htm>. Acesso em: 11 mar. 2025.

SALESFORCE. **Salesforce Developers** . Disponível em: <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_callouts_wsdl2apex_testing.htm>. Acesso em: 11 mar. 2025.

Links úteis

HTTP

SALESFORCE. **Salesforce Developers** . Disponível em: <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes_restful_http.htm>. Acesso em: 11 mar. 2025.

SALESFORCE. **Salesforce Developers** . Disponível em: <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_classes_restful_http_testing_httpcalloutmock.htm>. Acesso em: 11 mar. 2025.

SALESFORCE. **Salesforce Developers** . Disponível em: <https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_methods_system_staticresourcecalloutmock.htm>. Acesso em: 11 mar. 2025.

SALESFORCE. **Salesforce Developers** . Disponível em: <https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_methods_system_multistaticresourcecalloutmock.htm>. Acesso em: 11 mar. 2025.

Links úteis

Bibliotecas

DINICOLANTONIO, S. **Easier mocking in Apex (because the Stub API sucks)** . Disponível em: <https://blog.restlesslabs.com/spence/apex-stubbing>.

SALESFORCE. **GitHub - salesforce/apex-mockery: Lightweight mocking library in Apex** . Disponível em: <https://github.com/salesforce/apex-mockery>. Acesso em: 11 mar. 2025.

JPMONETTE. **GitHub - jpmonette/q: A Dynamic SOQL Query Builder for the Force.com Platform** ☁️. Disponível em: <https://github.com/jpmonette/q>. Acesso em: 11 mar. 2025.

Obrigado!