Certified Tech Developer
The Ultimate Degree

Implantar API no Docker

Após assistir ao vídeo podemos citar que os **testes de integração** validam a interação entre as diferentes partes do sistema para verificar se **os componentes** de uma aplicação **funcionam corretamente em conjunto**.

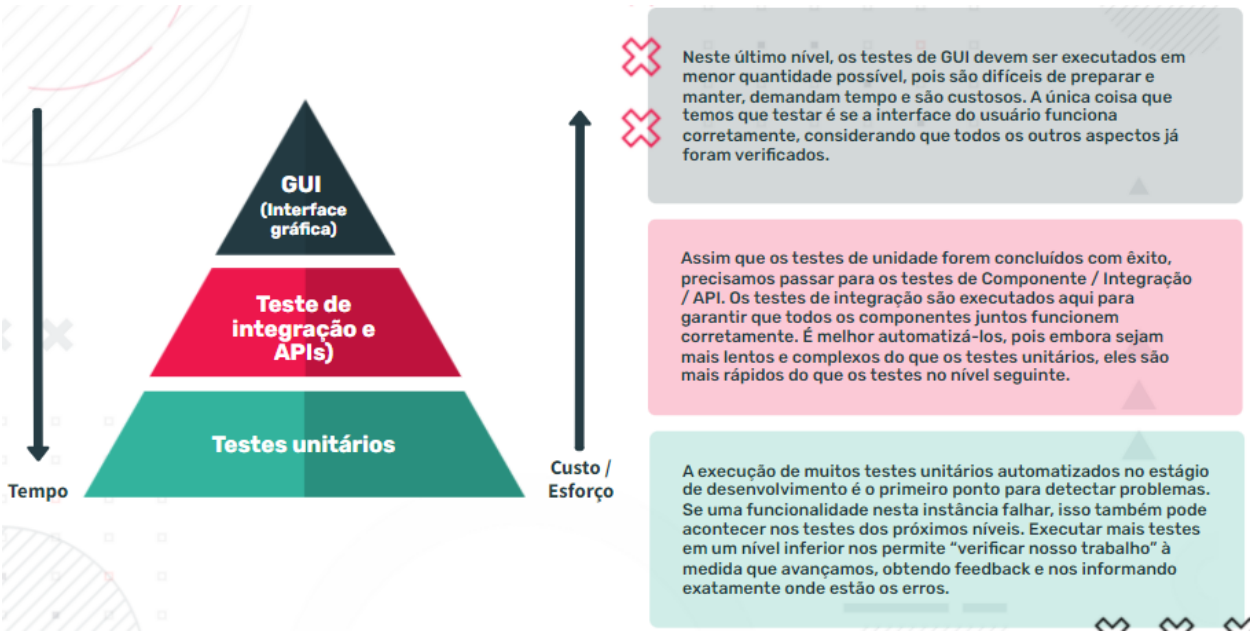
Esses testes cobrem uma área maior de código, da qual às vezes não temos controle sobre as bibliotecas de terceiros, e verificam, por exemplo, se um e-mail foi enviado, a conexão com o banco de dados, a conexão com outros serviços da web, entre outros.

Por exemplo, em uma arquitetura de microsserviços, eles são normalmente usados para verificar as integrações entre as diferentes camadas de código e componentes externos com os quais está sendo integrado, outros microsserviços, bancos de dados, caches, e etc.

Os testes de integração devem ter como objetivo cobrir os caminhos básicos de sucesso e falha através dos módulos de integração.

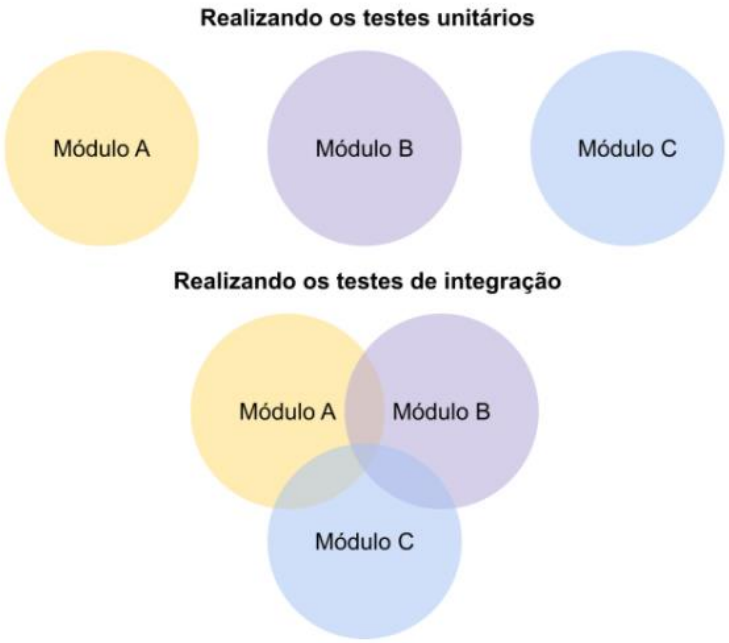
Pirâmide de testes ideal

A seguir, apresentamos a pirâmide de testes ideal, apresentada por Mike Cohn. Ela propõe quais testes devemos realizar de acordo com o momento do nosso projeto para economizar tempo, custos e esforços. O objetivo é ser capaz de se antecipar aos erros que podem ocorrer no momento indicado sem que se espalhem para cima.



Por que fazer testes de integração?

Uma vez que todos os componentes ou módulos funcionam corretamente de forma independente, devemos verificar o fluxo correto de dados entre os módulos dependentes.



No entanto, mesmo se todos os módulos em sua aplicação tenham passado pelos testes de unitários, ainda podem existir erros por vários motivos. Aqui estão alguns exemplos que demonstram a importância de realizar os testes de integração.

- Cada módulo é geralmente projetado por um desenvolvedor de software cuja lógica de programação pode ser diferente dos outros módulos. Portanto, esses testes tornam-se essenciais para determinar o funcionamento dos módulos como um todo;
- Verificar se a integração dos módulos com o banco de dados está correta ou não;
- Para verificar incompatibilidade entre módulos que podem gerar erros;
- Para testar a compatibilidade do hardware com o software;
- Para evitar erros de tratamento de exceções inadequadas entre os
- módulos.

Para encerrar esse assunto, veremos porque os testes unitários e os testes de integração não são iguais. A seguir conheceremos as principais diferenças.

Teste de integração com o Spring Boot: Mock MVC

O Spring oferece uma ótima ferramenta para testar aplicações Spring Boot: **MockMVC**. Essa estrutura fornece uma maneira fácil de implementar testes de integração para aplicações web. Como alternativa ao **MockMVC**, podemos utilizar os frameworks **RestTemplate** ou **Rest-Assured**. Para começar a testar endpoints declarados em um controller utilizando o MockMVC, precisamos adicionar as dependências **spring-boot-starter-web** e **spring-boot-starter-test** no arquivo **pom.xml**.

Vamos adicionar os testes de integração no projeto da clínica odontológica

Nosso líder técnico nos solicita para:

- Realizar testes de integração para o controller de pacientes;
- Utilizar o MockMVC, testar a busca de paciente pelo id;
- O teste deve verificar se o código de resposta é 200 e que no body retorna os dados do paciente.

Lembre-se de cadastrar um paciente antes de executar o teste, para assegurar que temos informações no banco de dados. Podemos inserir os pacientes da mesma forma que fizemos nos testes unitários.

Ajuda extra:

Para executar os testes sem ser impedido pela segurança da API, adicione o parâmetro addFilters = false em @AutoConfigureMockMvc.

Exemplo:

```
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc(addFilters = false)
public class IntegrationPacienteTest {
//test1
//test2

}
```