* **DEFINIÇÃO DE TESTES**

**Teste de segurança**

Como nosso modelo de negócio é baseado em backups e armazenamento de arquivos fisicos e digitais dos nossos clientes, é de extrema importância nossa que esses arquivos não fiquem vulneráveis a ataques ou falhas na autenticação e autorização do sistema. Sendo assim, será realizado testes de vulnerabilidade para não permitir que usuários não autorizados acessem os recursos e funcionalidades do nosso sistema. Os testes incluem:

* **Análise de fragilidade da Senha**
* **Análise de configurações incorretas** nas permissões de acessos e falhas na validação da sessão.
* **Análise de atualizações e Patches de segurança** para evitar que possíveis invasores explorem as vulnerabilidades não corrigidas.
* **Teste de penetração** visando simular ataques de hackers e identificar as vulnerabilidades do nosso sistema.
* **Análise de Segurança Estática,** revisando o código-fonte em busca de vulnerabilidades de segurança, como a injeção de SQL
* **Teste de Recuperação de Desastres**, que verifica como o sistema se comporta ao se recuperar de um ataque ou perca dos arquivos, testando procedimentos de backup e restauração.

**Teste de Conformidade**

Esse teste é importante para que tenhamos a garantia que nosso sistema esteja em conformidade com os regulamentos e padrões relevantes, como a RGPD (Regulamento Geral de Proteção de Dados). Avaliando se o sistema protege adequadamente a privacidade e confidencialidade dos dados e arquivos dos usuários. É importante avaliar também se as instalações físicas que hospedam nosso sistema estão protegidas contra acesso não autorizado, incêndios, inundações e outros desastres naturais. Bem como, revisar a documentação do sistema e garantir que todas as políticas de segurança e POs estejam documentadas.

**Teste Unitário com Mockito**

Vamos considerar um exemplo simplificado onde temos uma interface ‘FileStorage’ que define métodos para salvar e recuperar arquivos. Temos uma implementação, ‘FileStorageImpl’, que interage com o nosso sistema de armazenamento de arquivos fisicos e digitais.

1. **Configuração do Teste (setUp)**: Usamos MockitoAnnotations.openMocks para inicializar os mocks. Criamos uma instância de FileStorageImpl passando esses mocks.
2. **Testes de Salvamento (testSaveFile**): Verificamos se **saveFile** chama o método correto (save) no armazenamento físico ou digital, dependendo da extensão do arquivo. Usamos **verify** para garantir que o método correto seja chamado e never para garantir que o método incorreto não seja chamado.
3. **Testes de Recuperação (testRetrieveFile)**: Verificamos se **retrieveFile** chama o método correto (retrieve) no armazenamento físico ou digital. Usamos **when** para definir o comportamento dos mocks e **assertArrayEquals** para verificar se os dados retornados são os esperados.

Com esses testes, garantimos que FileStorageImpl interaja corretamente com PhysicalStorage e DigitalStorage, validando o comportamento do sistema de forma isolada.