UC:TESTE DE BACK- END

Teste de Codificação server-side O que é ?

O teste do lado do servidor é um método de teste A/B em que as variações de um teste são renderizadas diretamente do servidor da Web e, em seguida, enviadas premeditadamente para o dispositivo dos visitantes. A implementação diretamente no servidor permite que você execute testes mais sofisticados que, de outra forma, poderiam dificultar a experiência do usuário se implementados no lado do cliente. Além disso, o teste do lado do servidor também é optado nos casos em que é simplesmente inviável experimentar ao lado do cliente. Por exemplo, testar dois algoritmos diferentes de recomendação de produtos para um site de e-commerce.

Em outras palavras, o teste do lado do servidor abre a porta para um mundo totalmente novo de possibilidades de experimentação, permitindo que você teste profundamente dentro de sua tech stack e execute testes avançados que estão além do escopo da interface do usuário ou de mudanças cosméticas. Como todo o trabalho pesado ocorre no nível do servidor, a experiência do usuário se torna significativamente suave e rápida. Mais importante ainda, o teste do lado do servidor permite que você veja além do seu site e otimize até mesmo a experiência do seu produto, aplicativo para dispositivos móveis e dispositivos IoT. Assim, capacitar as equipes de produto e desenvolvimento para criar produtos verdadeiramente revolucionários e otimizados para a experiência.

Definição de Server-side

O server-side, ou lado do servidor, é a parte de uma aplicação web que é executada no servidor. É responsável por processar as solicitações feitas pelos usuários e fornecer as respostas adequadas. Enquanto o client-side, ou lado do cliente, é executado no navegador do usuário, o server-side lida com a lógica de negócios, acesso a banco de dados, autenticação, entre outras funcionalidades.

Como funciona o Server-side

Quando um usuário acessa um website ou envia uma solicitação para uma aplicação web, essa solicitação é enviada para o servidor. O servidor, por sua vez, processa essa solicitação utilizando a lógica de negócios e os dados armazenados em um banco de dados, se necessário. Após processar a solicitação, o servidor envia uma resposta de volta para o cliente, que pode ser uma página HTML, um arquivo de imagem, um documento PDF, entre outros.

Linguagens de programação Server-side

Existem várias linguagens de programação que podem ser utilizadas para desenvolver o lado do servidor de uma aplicação web. Alguns exemplos populares incluem PHP, Python, Ruby, Java e C#. Cada linguagem tem suas próprias características e vantagens, e a escolha da linguagem depende das necessidades e preferências do desenvolvedor.

Vantagens do Server-side

O server-side oferece várias vantagens no desenvolvimento web. Uma das principais vantagens é a segurança. Como o código server-side é executado no servidor, ele não é visível para os usuários finais. Isso significa que informações sensíveis, como senhas e chaves de acesso a banco de dados, podem ser mantidas em segurança. Além disso, o server-side permite o controle total sobre a lógica de negócios e a manipulação de dados, o que facilita a criação de aplicações complexas e escaláveis.

Planejamento dos testes o que é?

É um documento modelo usado em projetos, ele delineia os planos de um projeto dedicado a testar um determinado software.Geralmente inclui detalhes como cronograma do projeto, estratégia, metas, prazos e estimativas sobre os resultados dos testes ou possíveis desafios.
Além de orientar o processo de teste, os planos de teste também contribuem para a priorização dos objetivos do próprio software, indo além do mero processo de teste.Isso auxilia na especificação das práticas e estratégias a serem adotadas pelos profissionais durante a fase de teste. Por exemplo, se um dos requisitos-chave do software é um recurso que salva arquivos automaticamente, um objetivo central do plano de teste seria garantir que este recurso funcione corretamente por meio de testes abrangentes em diferentes tipos de arquivos.

Por que um plano de teste é importante?

Os planos de teste desempenham um papel crucial ao direcionar as equipes de teste em direção a objetivos específicos, proporcionando vantagens significativas:

- Orientação para equipes de garantia de qualidade: O plano de teste serve como um guia para a equipe de garantia de qualidade, estabelecendo metas e criando um cronograma para o projeto. Facilita a distribuição eficiente de tarefas em equipes maiores.
- Explicação do processo de teste: Ao explicar o processo de teste de forma compreensível, os planos de teste se tornam ferramentas de comunicação eficazes, permitindo que profissionais de diferentes áreas compreendam o que está envolvido no teste do software.
- Organização de recursos e objetivos: Ao listar recursos e objetivos, os planos de teste ajudam a rastrear necessidades de suprimentos para o projeto. Além disso, ao definir funções, responsabilidades e informações de contato, facilitam a comunicação eficaz entre os membros da equipe.
- Estimativas de cronograma: Fornecer informações detalhadas sobre o cronograma do processo, incluindo prazos e estimativas, permite a identificação de metas de eficiência. Essas estimativas também beneficiam outros departamentos, como o de marketing, que pode planejar suas atividades com base no progresso do teste.

Planejamento dos Teste

O planejamento de testes em server-side envolve uma série de etapas e decisões para garantir que a lógica da aplicação que roda no servidor funcione corretamente e de forma confiável. O primeiro passo é entender bem o escopo do sistema: quais funcionalidades o servidor oferece, quais endpoints de API estão disponíveis, que regras de negócio são aplicadas, como é feito o acesso a banco de dados e quais integrações externas existem (como gateways de pagamento ou serviços de autenticação).

Depois disso, é importante decidir os tipos de testes que serão feitos. Os testes unitários verificam funções ou métodos isoladamente, sem depender de banco de dados ou rede. Os testes de integração avaliam se os diferentes módulos do sistema funcionam bem juntos — por

exemplo, se uma requisição POST cria um novo registro corretamente no banco. Já os testes de API verificam se os endpoints estão funcionando conforme esperado, com os dados corretos, retornando os status HTTP adequados.

Em alguns casos, também se aplicam os testes de contrato, que garantem que a comunicação entre serviços (por exemplo, entre microsserviços) esteja no formato esperado. E para garantir que o servidor suporta bem a carga de usuários, podem ser feitos testes de carga, simulando múltiplas requisições simultâneas.

Durante o planejamento, também é necessário escolher as ferramentas mais adequadas de acordo com a linguagem usada. Em Node.js, por exemplo, é comum usar Jest ou Mocha junto com Supertest. Em Python, usa-se Pytest. Em Java, JUnit e RestAssured são populares. Independentemente da linguagem, é fundamental automatizar os testes, integrar com a pipeline de CI/CD, e manter uma boa cobertura de testes, priorizando os fluxos mais críticos.

Implementação dos testes

A implementação dos planejamentos de testes é a fase em que tudo o que foi definido na etapa de planejamento começa a ser colocado em prática. Ela envolve transformar os objetivos e estratégias definidos em ações concretas que permitam verificar se o sistema atende aos requisitos especificados. Isso inclui a criação dos casos de teste, que são descrições detalhadas de cenários que precisam ser verificados, considerando pré-condições, passos a serem seguidos e os resultados esperados. Esses casos são criados com base nos requisitos do sistema, tanto funcionais quanto não funcionais.

Também é necessário preparar o ambiente de testes, garantindo que ele esteja configurado de forma adequada e semelhante ao ambiente de produção, com todos os sistemas, dados e conexões necessárias. Em paralelo, prepara-se os dados de teste, que podem ser fictícios, reais (desde que anonimizados) ou gerados automaticamente. Esse processo é fundamental para garantir que os testes sejam realizados de forma eficaz e representativa.

As ferramentas de apoio aos testes também são configuradas nesse momento. Isso inclui ferramentas de automação, de gerenciamento de testes e de controle de defeitos. Com tudo pronto, inicia-se a execução dos testes conforme o planejado. Durante essa execução, os

resultados são documentados cuidadosamente, e qualquer falha ou comportamento inesperado é registrado para análise e correção. Os defeitos encontrados são encaminhados para a equipe de desenvolvimento, e, conforme as correções são entregues, novos ciclos de testes podem ser executados.

Essa fase exige constante monitoramento e comunicação entre os envolvidos, para garantir que o progresso esteja de acordo com o cronograma e que eventuais problemas sejam resolvidos rapidamente. Em resumo, a implementação do planejamento de testes transforma a estratégia de qualidade em uma ação prática e sistemática, essencial para assegurar que o software atenda às expectativas dos usuários e esteja pronto para uso.

Execução dos testes

A execução dos testes pode ser feita de forma manual ou automatizada, mas em ambientes server-side o mais comum (e recomendado) é a automação.

O processo geralmente segue os seguintes princípios:Primeiro, o ambiente de testes precisa estar configurado corretamente. Isso inclui variáveis de ambiente, banco de dados de testes (que deve ser separado do banco de produção), e a inicialização do servidor ou dos serviços que serão testados. Se estiver usando testes de integração ou de API, é comum criar dados mock ou fixtures antes da execução e limpá-los depois. Na prática, os testes são disparados por comandos de terminal, por exemplo: npm test no Node.js, pytest no Python ou mvn test no Java. Esses comandos executam todos os arquivos de teste configurados e geram relatórios com os resultados.

Durante a execução, cada teste verifica um comportamento específico do sistema. Se o comportamento for o esperado, o teste passa. Caso contrário, ele falha e geralmente informa qual foi a expectativa e o que ocorreu na prática, ajudando a identificar o problema.

Além da execução local por desenvolvedores, é essencial que os testes rodem automaticamente sempre que há uma nova alteração no código. Isso é feito através de pipelines de integração contínua (CI), usando ferramentas como GitHub Actions, GitLab CI, Jenkins, etc. A CI roda os testes toda vez que um novo commit é feito ou um pull request é aberto, garantindo que novas mudanças não quebrem funcionalidades existentes.

Após a execução, os resultados são analisados. Se todos os testes passarem, o sistema pode ser considerado estável naquele ponto. Se houver falhas, os desenvolvedores precisam investigar e corrigir os problemas antes de prosseguir com a entrega.

Também é comum verificar métricas como tempo total de execução e cobertura de testes (percentual do código que foi testado), para garantir qualidade e eficiência. Algumas ferramentas integram tudo isso e geram relatórios visuais para facilitar a análise.

Análise de resultado dos testes

A análise de resultados dos testes é uma etapa fundamental no processo de garantia de qualidade. É a partir dela que se entende se o sistema está funcionando corretamente, se há falhas a serem corrigidas e se os testes estão cobrindo bem o que foi planejado.

Quando os testes são executados, eles geram saídas que indicam quais testes passaram e quais falharam. A análise desses resultados envolve alguns pontos principais:

Primeiro, é importante identificar quais testes falharam. Cada falha geralmente vem acompanhada de uma mensagem explicando o motivo. Pode ser, por exemplo, que uma função retorna um valor diferente do esperado, que um endpoint retornou o status HTTP errado, ou que houve uma exceção não tratada. Esses detalhes ajudam o desenvolvedor a entender rapidamente a origem do problema.

Depois, é preciso classificar a gravidade das falhas. Nem todo erro tem o mesmo peso. Um bug em um recurso crítico, como autenticação ou pagamento, é mais urgente do que um erro em uma mensagem de interface. A análise dos testes deve levar isso em conta para priorizar as correções.

Outro aspecto importante é a cobertura de testes, que mostra quanto do código foi realmente exercitado pelos testes. Ferramentas de cobertura, como Istanbul (Node.js), Coverage.py (Python) ou JaCoCo (Java), geram relatórios indicando quais linhas, funções ou arquivos foram testados. Se a cobertura estiver baixa em partes críticas da aplicação, isso sinaliza a necessidade de escrever mais testes.

Além disso, a análise pode identificar falsos positivos ou falsos negativos. Um falso positivo é quando um teste passa, mas deveria falhar — isso pode indicar um teste mal escrito. Já um falso negativo é quando o teste falha, mas o código está correto — geralmente causado por problemas no próprio teste ou no ambiente de execução.

Outro ponto a observar é o tempo de execução dos testes. Testes muito lentos podem prejudicar o fluxo de desenvolvimento e a entrega contínua. Durante a análise, testes desnecessariamente pesados ou redundantes podem ser otimizados ou eliminados.

Por fim, os resultados da análise devem ser documentados e comunicados. Em times maiores, é comum gerar relatórios ou dashboards que mostram o status geral dos testes, os erros mais frequentes, a evolução da cobertura e a estabilidade das últimas versões.