Introdução á API Rest

Conceitos de HTTP, API REST e JSON

HTTP (HyperText Transfer Protocol)

- Protocolo para comunicação entre cliente e servidor.

Verbos HTTP:

Verbo	Função
GET	Solicita dados
POST	Envia dados para criação
PUT	Atualiza ou cria recurso
DELETE	Remove recurso

Status Code:

- Indicam sucesso ou erro nas requisições (ex: `200`, `404`, `500`).

---

4. APIs

O que são APIs?

- Interfaces que permitem comunicação entre sistemas.
- Utilizam protocolos como HTTP e formatos como JSON.

Vantagens:

- Integração entre serviços
- Redução de custos
- Controle e segurança (via tokens e keys)

---

5. Arquitetura: Monolito vs Microserviços

Monolito:

- Sistema único e centralizado.

Vantagens:

- Simples de desenvolver e testar

Desvantagens:

- Dificuldade em escalar e manter

Microserviços:

- Componentes independentes e focados em funcionalidades específicas.

Vantagens:

- Escalabilidade e manutenibilidade

Desvantagens:

- Complexidade de arquitetura e operação

O que são Histórias de Usuário?

- Descrição breve da funcionalidade.
- Características:
  - Independente
  - Mensurável
  - Testável (com critérios de aceitação)

Example Mapping

- Técnica para esclarecer histórias usando 3 perspectivas:
  - Product Owner (cliente)
  - Desenvolvedor
  - Testador

Testes Estáticos com Swagger

O que é Swagger?

Swagger é uma ferramenta que documenta APIs REST. Ele exibe:

- Os **recursos disponíveis** na API,
- As **operações (métodos HTTP)**,
- E os **parâmetros necessários** para cada requisição.

Conceitos Básicos de API REST

- Um app faz uma **requisição HTTP ou HTTPS** para acessar funcionalidades da API.
- As requisições são feitas usando métodos como GET, POST, PUT, DELETE, etc.

Swagger na Prática

- Do lado esquerdo: Exibe o arquivo `.yml` (documento técnico da API).
- Do lado direito: Mostra uma **interface gráfica** interpretando esse `.yml`, facilitando a leitura e execução.

Swagger Petstore 1.0.0

- Exibe todos os métodos, recursos e operações que a API disponibiliza.
- Permite simular requisições e observar respostas diretamente na interface.

Tipos de Parâmetros

- **Path (Caminho)**: Parte da URL usada para identificar um recurso específico. Ex: `/user/{id}`
- **Query (Consulta)**: Usado para filtrar ou modificar a resposta. Ex: `/user?status=active`
- **Body**: Conteúdo enviado na requisição, geralmente em JSON, contendo os dados para criação ou atualização.

Resposta da API

- O Swagger exibe:
  - **Status da requisição** (200, 400, etc.)
  - **Tipo de conteúdo retornado** (JSON ou XML)

Validações em uma API – Resumo Prático

Como validar o status de uma API

- Utilize ferramentas como o **Postman** ou o navegador para executar endpoints e verificar o retorno em JSON.
- Exemplo: `https://simple-books-api.glitch.me/status` retorna o status da API.

Conceitos importantes

- **Header**: contém metadados da requisição, como autenticação (`Authorization: Bearer token`), tipo de conteúdo, etc.
- **Body**: contém os dados reais enviados ao servidor, geralmente em JSON (ex: login com `username` e `password`).

Boas práticas para testar APIs

1. **Entender o projeto**: saber o objetivo e os fluxos para definir ferramentas e testes.
2. **Documentação completa**: base para saber os métodos, endpoints e respostas esperadas.
3. **Planejamento de testes**: escolha dos testes funcionais, de segurança, etc.
4. **Segurança**: testar autenticação, permissões e protocolos seguros (HTTPS, tokens).

Testes Funcionais em API REST

- Validações feitas com ferramentas como Swagger e Insomnia.
- Foco: confirmar se a **função da API** está correta.
- **Regras de negócio** nem sempre estão documentadas no Swagger — podem vir de casos de uso, entrevistas ou anotações soltas.

Erros comuns no back-end

- **Gravidade**: dano causado pelo bug.
- **Prioridade**: urgência para corrigir.
- **Risco**: impacto imprevisível no projeto.
- **Erros por documentação ruim**: Swagger incompleto causa falhas nos testes.
- **Erros por massa de dados**: sobrecarga ou dados incompletos causam falhas na aplicação.

Plano de Teste de Software

O que é

- Documento que **planeja** as atividades do processo de teste (não executa).
- Pode haver **vários planos** para diferentes níveis: usabilidade, sistema, funcional, etc.
- Define: funcionalidades a serem testadas, objetivos, responsáveis, riscos e ações de contingência.

Importância

- Ajuda a estimar o **esforço necessário** e a **validar a qualidade da aplicação**.
- Serve como guia para **organizar, comunicar e orientar** todos os envolvidos no processo.
- Permite acompanhar o progresso, ajustar o planejamento e melhorar processos futuros.

Estrutura comum de um Plano de Teste

- Nome do projeto
- Resumo do plano
- Pessoas envolvidas
- Funcionalidades/módulos a testar
- Local dos testes
- Recursos necessários
- Critérios de entrada e saída
- Riscos e ações de contingência
- Divulgação dos resultados
- Cronograma

Conteúdos essenciais do artigo

- **Introdução**: escopo e objetivos
- **Requisitos a testar**
- **Estratégias, tipos de teste e ferramentas**
- **Equipe e infraestrutura**
- **Cronograma de testes**
- **Documentos complementares**

Cobertura de Testes de APIs

Criação de Fluxo no Postman (Compass)

- A videoaula ensinou a criar **fluxos de teste no Postman**, cobrindo o backend sem seguir rotas específicas.
- **Exemplo de fluxo:**
  a. Criar usuário
  b. Adicionar item ao carrinho
  c. Finalizar compra

Como Medir a Cobertura de Testes de API REST

Cobertura = quanto da API está de fato **sendo testado**, com foco em:

Cobertura de Entrada (Input)

1. **Path Coverage:**
   - Testar todos os **endpoints** (URIs).
   - Ex: 6 endpoints testados / 13 existentes → **46% de cobertura**.
2. **Operator Coverage:**
   - Verifica se todos os **métodos HTTP (GET, POST, PUT, DELETE)** estão sendo testados.
   - Ex: 14/19 métodos testados → **74%**.
3. **Parameter Coverage:**
   - Verifica se todos os **parâmetros dos métodos** foram testados.
   - Ex: 5/5 parâmetros → **100%**.
4. **Parameter Value Coverage:**

- Testar **todos os valores possíveis** para parâmetros do tipo boolean ou enum.

5. **Content-type Coverage:**
   - Verifica se os testes cobrem **todos os tipos de conteúdo (Content-Type)** aceitos (ex: JSON, XML).
   - Ex: 2/4 tipos cobertos → **50%**.
6. **Operation Flow Coverage:**
   - Mede se os **fluxos de operação (sequência de chamadas)** foram testados.
   - Ex: 1 fluxo testado de 4 possíveis → **25%**.

Cobertura de Saída (Output)

1. **Response Properties Body Coverage:**
   - Verifica se todas as **propriedades da resposta (body)** estão cobertas.
2. **Status Code Coverage:**
   - Mede se todos os **status codes** possíveis estão testados.
   - Ex: 15/25 → **60% de cobertura**.

Testes Candidatos à Automação

- **Deve-se automatizar:**
  - Tarefas repetitivas
  - Funcionalidades críticas
  - Partes da aplicação com **alto risco**
  - Funcionalidades que o cliente **prioriza**
- **Importante:**
  - **Demandas novas não devem ser automatizadas antes de testes manuais.**
  - Primeiro se testa manualmente, depois se avalia se o cenário será automatizado.