INSTITUTO DE INFORMÁTICA

CONSTRUÇÃO DE SOFTWARE

LEONARDO RIBEIRO OLIVEIRA PALMEIRA
VINÍCIUS SILVA BENEVIDES
THIAGO NASCENTE BORGES
MURILO MESQUITA CAROLINA
ANA LIZ BOMFIM GOMES
JOÃO PEDRO NERY

Goiânia
2024

Sumário

Análise de requisitos

Requisitos Funcionais

1. **Implementação de API FHIR**:
    - A aplicação deve expor APIs que sigam o padrão FHIR para recursos de saúde, como paciente, médico, id_paciente, entre outros.
    - A API deve permitir operações CRUD (Create, Read, Update, Delete) para os recursos FHIR.
    - A API deve ser capaz de retornar recursos FHIR em formatos compatíveis, como JSON ou XML, de acordo com o padrão FHIR.
2. **Testes de Conformidade com o Padrão FHIR**:
    - A aplicação deve garantir que todos os recursos FHIR implementados estejam de acordo com o padrão FHIR.
    - Deve haver testes automatizados para garantir que as respostas da API estejam no formato FHIR correto, conforme os recursos definidos.
    - A suite de testes deve validar se os dados retornados nas respostas da API estão corretos e completos.
3. **Testes de Endpoints FHIR**:
    - A aplicação deve incluir testes unitários para garantir que todos os endpoints FHIR (GET, POST, PUT, DELETE) estejam funcionando corretamente.
    - Os testes devem validar que as respostas da API estejam corretas em termos de status HTTP (ex: 200 OK, 400 Bad Request, 404 Not Found).
    - A aplicação deve garantir que a manipulação dos recursos FHIR (como paciente, id_paciente, etc.) seja realizada conforme esperado.
4. **Testes de Validação de Dados**:
    - A suite de testes deve incluir verificações para garantir que os dados nos recursos FHIR (como campos obrigatórios, tipos de dados, e formatos) sejam válidos.
    - A aplicação deve validar se os dados recebidos e enviados estão dentro dos limites e regras de validação do padrão FHIR, como tipos de dados, formatos de data e valores aceitáveis.

5. **Testes de Integração**:

    - A aplicação deve garantir que os recursos FHIR sejam armazenados e recuperados corretamente do banco de dados.

    - Deve haver testes para validar a integração da API com o banco de dados, garantindo que os recursos sejam persistidos e recuperados de acordo com o esperado.

6. **Resultado de testes**:

    - A aplicação deve garantir que as inconformidades retornadas sejam destacadas e exibidas ao usuário.

Requisitos Não Funcionais

1. **Desempenho**:

    - A aplicação deve ser capaz de processar e responder a requisições FHIR com desempenho adequado, mesmo em situações de alta carga.

    - O tempo de resposta para os testes de API FHIR deve ser medido e deve não exceder o tempo limite estipulado (ex: 1 segundo por requisição) para garantir a eficiência do sistema.

2. **Escalabilidade**:

    - A aplicação deve ser escalável para lidar com um número crescente de usuários ou requisições simultâneas sem afetar a performance da API FHIR.

    - A infraestrutura deve ser capaz de suportar o aumento da carga de trabalho, especialmente em ambientes de produção, e permitir a adição de mais instâncias de servidores, se necessário.
      protegida contra vulnerabilidades.

3. **Monitoramento e Logs**:

    - A aplicação deve incluir funcionalidades de monitoramento para garantir que todos os testes e interações com a API FHIR sejam monitorados.

    - Deve ser gerado e armazenado um log detalhado de todas as requisições e respostas da API, bem como os resultados da suite de testes, para diagnóstico e auditoria.

Arquitetura da aplicação

A arquitetura será modular e desacoplada, aproveitando as capacidades do ASP.NET Core e tecnologias auxiliares. A aplicação será dividida em três camadas principais: apresentação, aplicação e infraestrutura.

A camada de apresentação utilizará ASP.NET Core MVC e Blazor. O MVC fornecerá endpoints RESTful para integração com ferramentas externas, enquanto o Blazor criará uma interface web rica, exibindo relatórios, dashboards e gráficos interativos. O Blazor permitirá a visualização dinâmica dos resultados das execuções, facilitando a análise pelos usuários.

A camada de aplicação conterá a lógica de negócio para organização, execução e análise dos testes, com serviços que gerenciarão entidades como Casos de Teste, Suítes e Resultados. Ela coordenará a execução dos testes com os frameworks XUnit, NUnit ou MSTest, suportando execução isolada, garantindo eficiência e redução de tempo de processamento.

A camada de infraestrutura será responsável pela persistência de dados e comunicação com ferramentas externas. Utilizará o Entity Framework Core para acesso ao banco de dados (SQL Server, SQLite ou outros) e integrará o validador FHIR via CLI. Também realizará comparações de dados JSON com JsonDiffPatch.Net ou Newtonsoft.Json, gerando relatórios detalhados das divergências encontradas.

A comunicação entre as camadas será orientada a contratos, facilitando testes, manutenção e extensões. As entidades de domínio e a lógica de negócio ficarão isoladas da infraestrutura, garantindo flexibilidade.

No ciclo de execução, o usuário poderá cadastrar casos e suítes de teste pela interface Blazor ou APIs REST, acionar a execução das suítes, realizar validações FHIR, e ao final, visualizar relatórios interativos no Blazor com resultados e insights. Essa arquitetura garante flexibilidade, escalabilidade e manutenção facilitada.

Tecnologias

- ASP.NET Core API: Utilizado para gerenciar os casos de teste, suítes de teste e a execução dos testes. Serve como a base para a comunicação entre os componentes e para a criação de uma interface de usuário.

- ReactJS para a criação de relatórios e visualização de resultados dos testes.

- Entity Framework Core: Usado para persistir dados sobre os casos de teste, suítes e resultados em um banco de dados relacional (SQL Server, SQLite, etc.).

- XUnit/NUnit/MSTest: Frameworks de teste em .NET para organizar e executar os testes de forma estruturada.

- CLI (Command Line Interface): Para executar o validador FHIR, como validator_cli, a partir do código ASP.NET Core.

- JSON Diff (JsonDiffPatch.Net ou Newtonsoft.Json): Para comparar os resultados obtidos com os resultados esperados, especialmente se os dados forem em formato JSON.

Meio de comunicação

WhatsApp

Principalmente para comunicação rápida e informal, como discussões rápidas, esclarecimento de dúvidas simples e atualizações rápidas.

Discord

Ideal para conversas mais estruturadas, tanto por texto quanto por voz. Pode ser usado para reuniões de equipe, discussões sobre o projeto e integração com bots que ajudam a gerenciar tarefas e informações.

GitHub

Utilizado para o gerenciamento de código, controle de versão e colaboração no desenvolvimento de software. É o principal lugar onde o código-fonte do projeto é armazenado, revisado e compartilhado.

Google Drive

Armazenamento de arquivos compartilhados, como documentos de planejamento, documentação do projeto, e apresentações.