Instituto Infnet

Sistema de Questionário

Arquitetura .NET

25E4

Luan Fernandes

https://github.com/souluanf/OnlineSurvey

Sumário

Introdução

Este documento apresenta a arquitetura do Sistema de Questionários Online, projetado para suportar pesquisas públicas em larga escala, como pesquisas eleitorais. A solução foi desenvolvida utilizando o ecossistema .NET, priorizando escalabilidade, manutenibilidade e time-to-market.

O código-fonte completo está disponível em: https://github.com/souluanf/OnlineSurvey

Diagrama de Contexto

O diagrama de contexto mostra o sistema como uma caixa preta e suas interações com usuários e sistemas externos.

Visão para Usuários de Negócio		
Ator	**Descrição**	**Interação**
Respondente	Cidadão que acessa a pesquisa através de anúncios em redes sociais	Responde perguntas de múltipla escolha
Administrador	Equipe da startup que gerencia as pesquisas	Cria pesquisas, ativa/encerra coleta, visualiza resultados

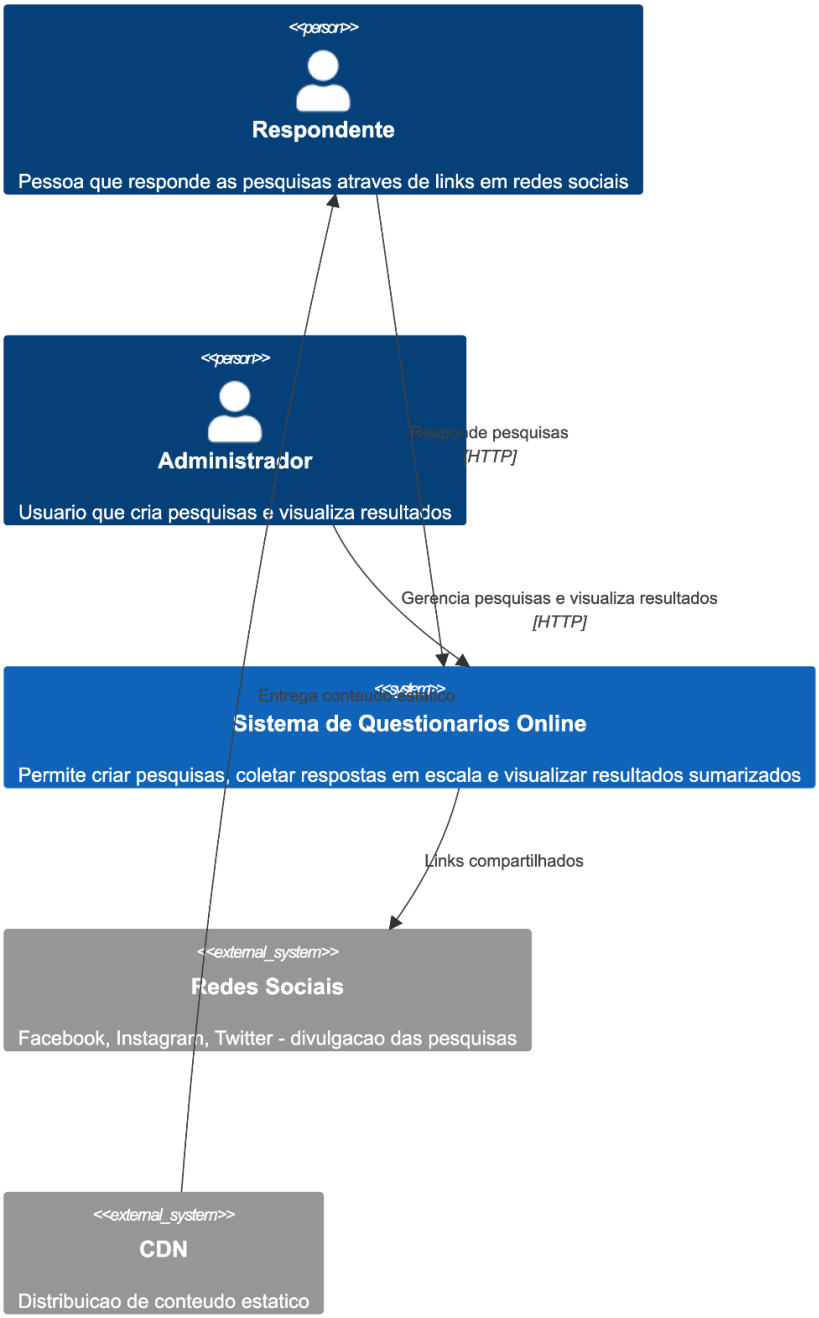
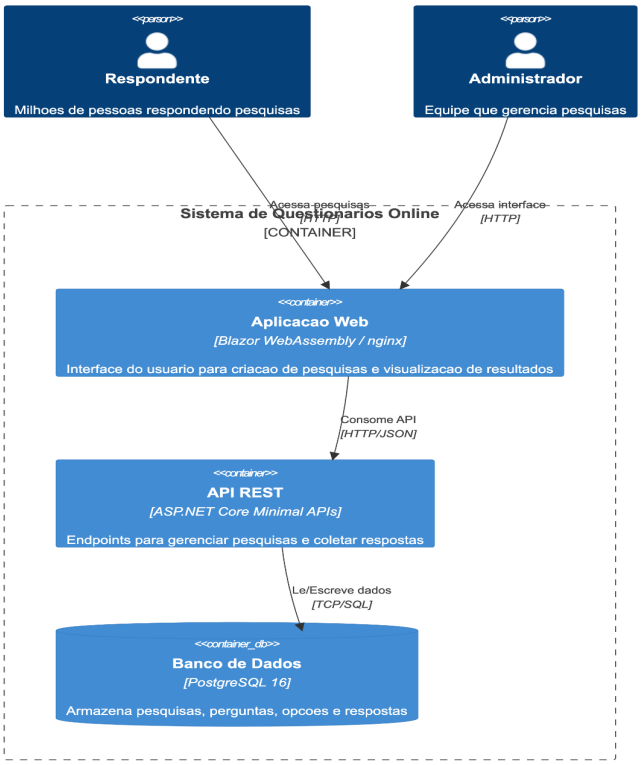

<<person>>
Respondente
Pessoa que responde as pesquisas atraves de links em redes sociais
<<person>>
Administrador
Usuario que cria pesquisas e visualiza resultados
Responde pesquisas
[HTTP]
Gerencia pesquisas e visualiza resultados
[HTTP]
Entrega conteudo estatico
<<system>>
Sistema de Questionarios Online
Permite criar pesquisas, coletar respostas em escala e visualizar resultados sumarizados
Links compartilhados
<<external_system>>
Redes Sociais
Facebook, Instagram, Twitter - divulgacao das pesquisas
<<external_system>>
CDN
Distribuicao de conteudo estatico

Diagrama de Containers

O diagrama de containers mostra as aplicações e bancos de dados que compõem o sistema.

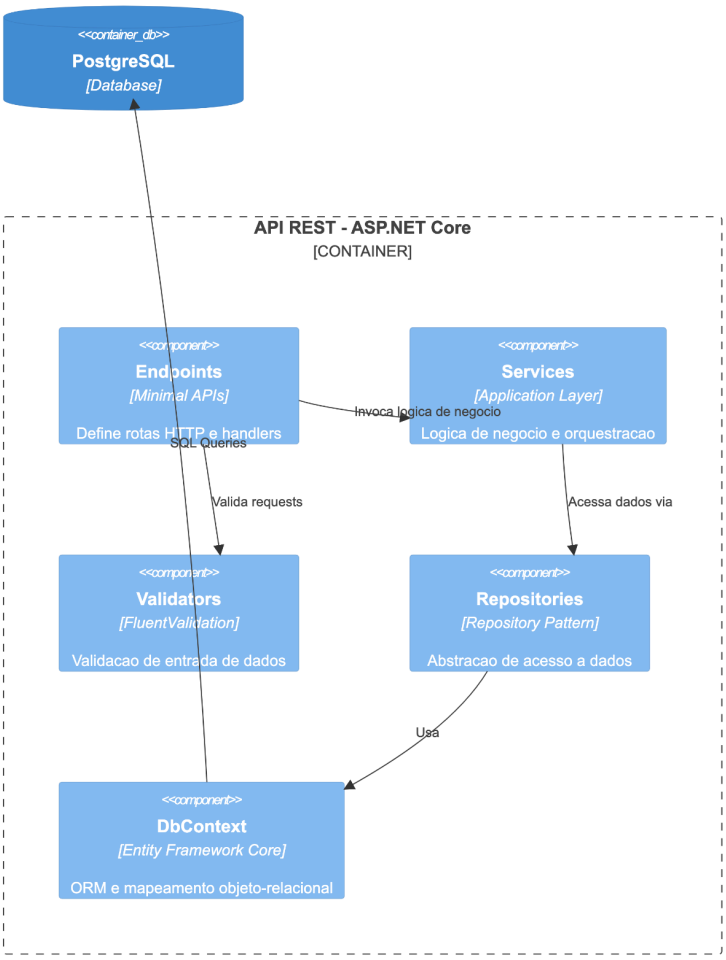


Detalhamento dos Containers		
Container	**Tecnologia**	**Justificativa**
Aplicação Web	Blazor WebAssembly	SPA moderna, executa no browser, permite desenvolvimento full-stack em C#

API REST	ASP.NET Core Minimal APIs	Alta performance, baixa latência, ideal para APIs de alto throughput
Banco de Dados	PostgreSQL	Open-source, robusto, excelente para cargas de escrita intensiva

Diagrama de Componentes da API

Componentes da API



Componentes do Frontend

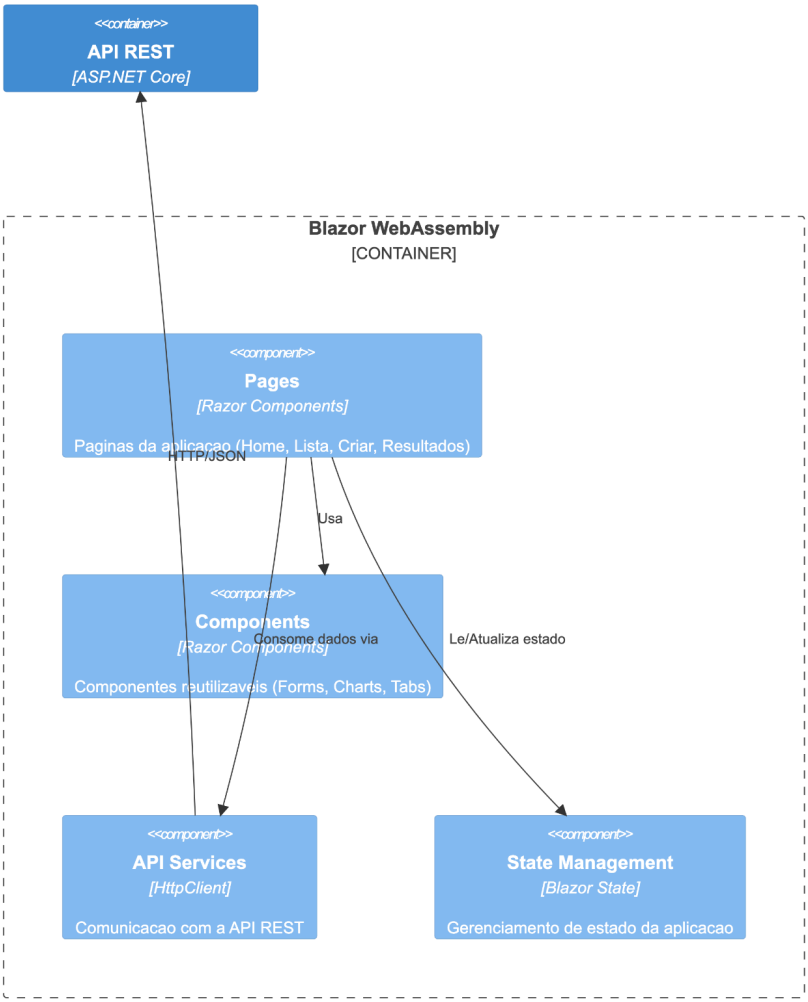
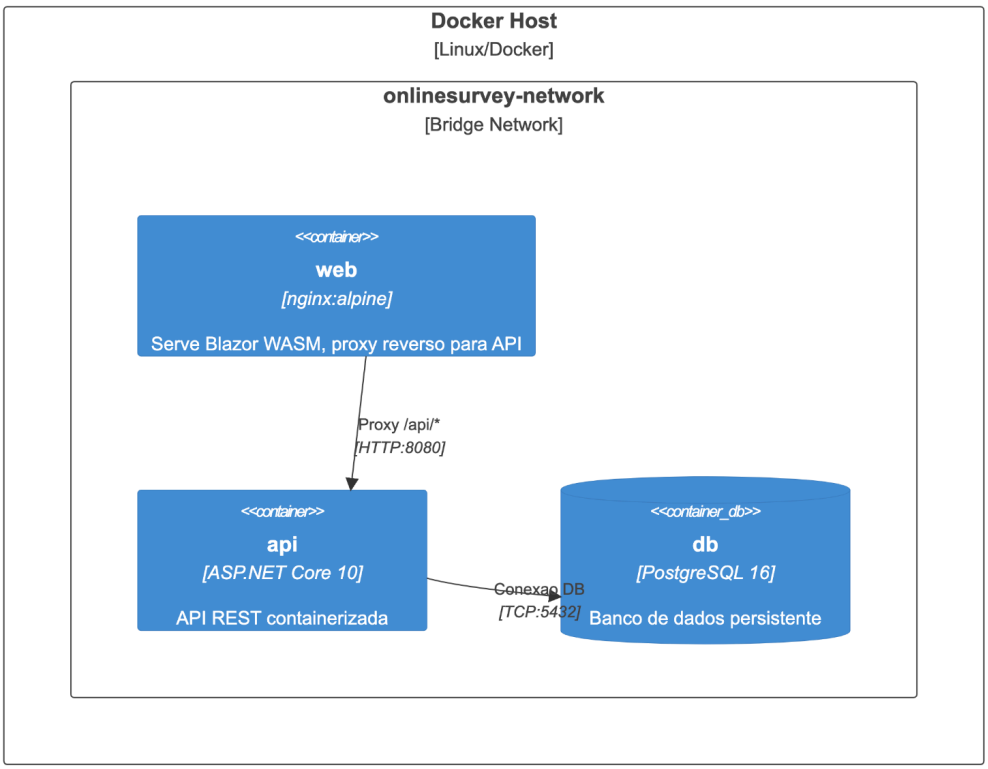
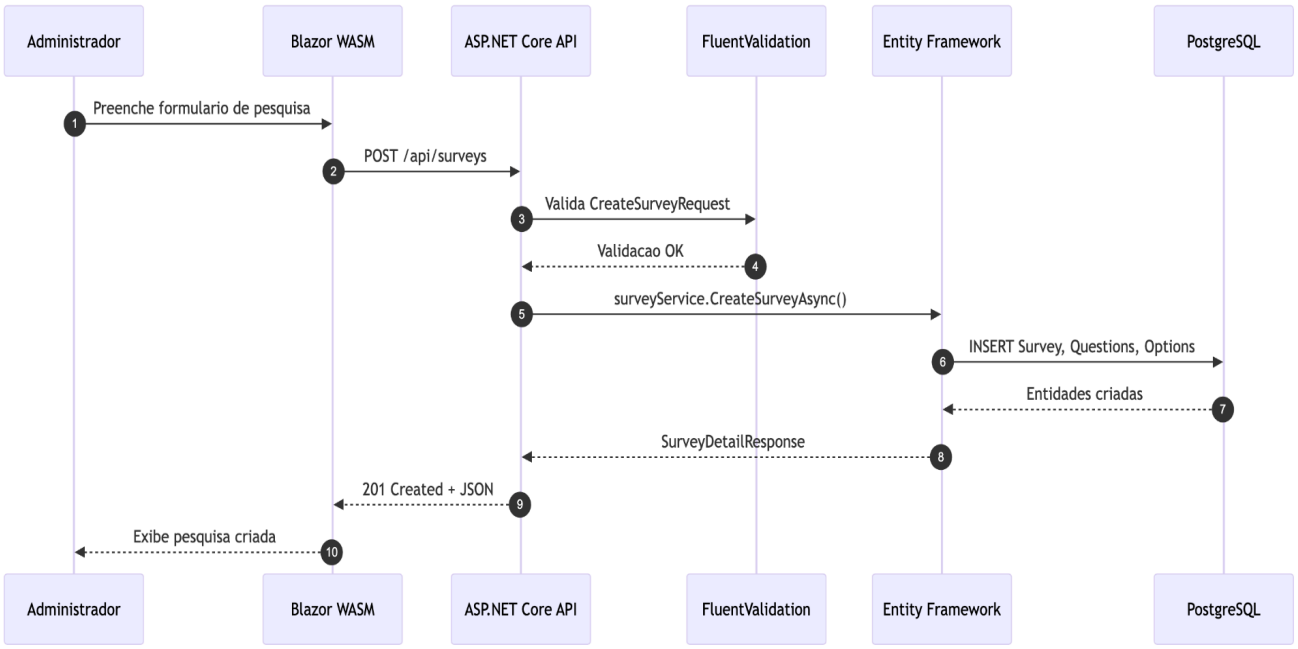


Diagrama de Implantação (Docker)

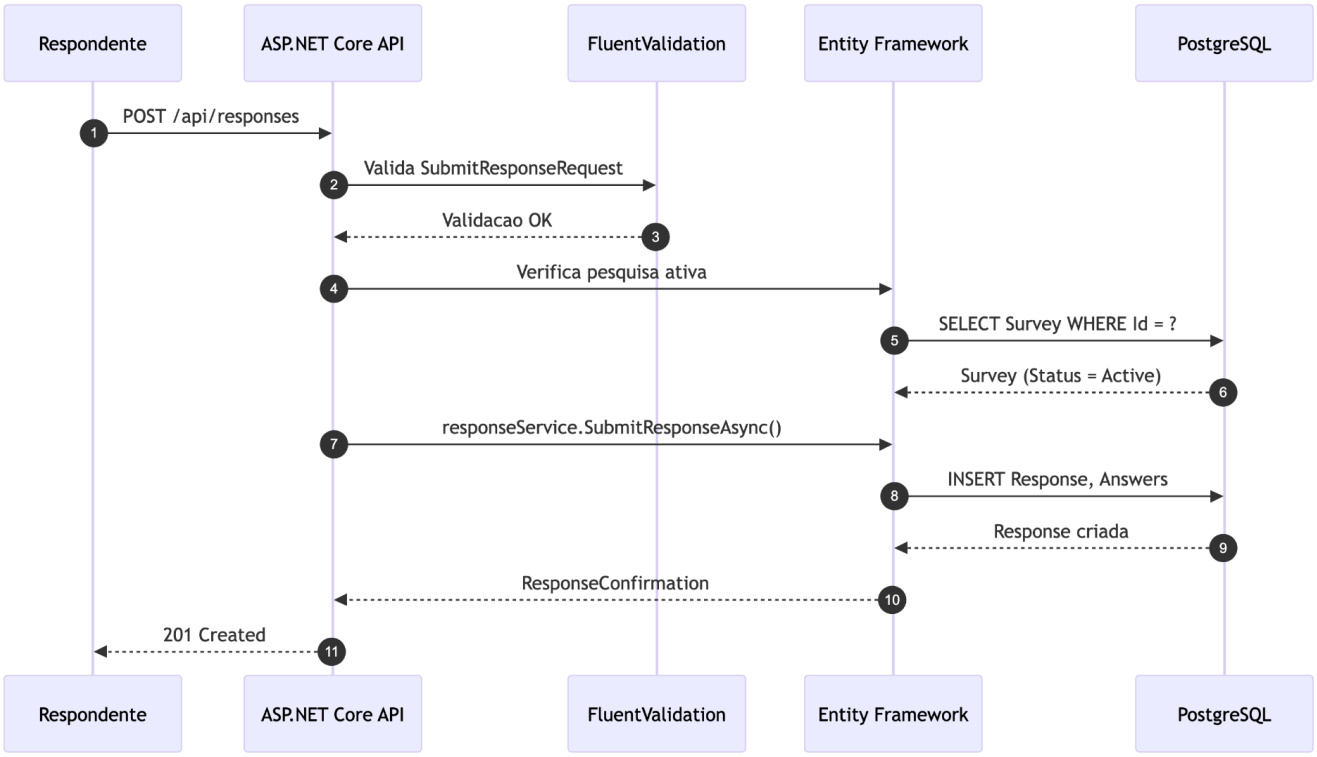


Fluxo de Dados

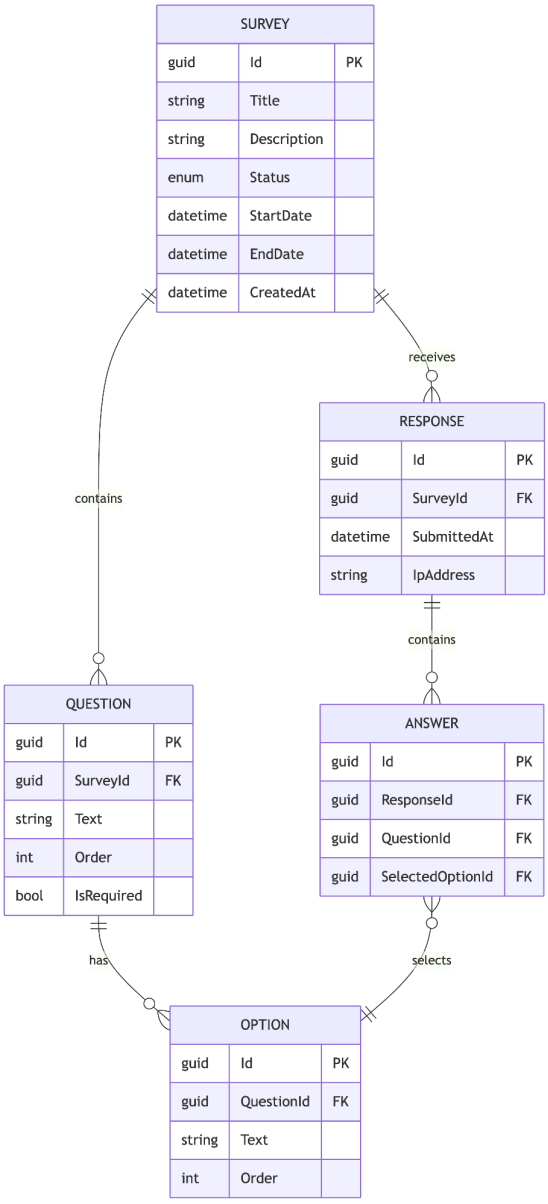
Fluxo de Criação de Pesquisa



Fluxo de Resposta (Alta Escala)



Modelo de Dados



Justificativas Arquiteturais

Para Desenvolvedores

ASP.NET Core Minimal APIs	
Performance	Menor overhead que controllers MVC, ideal para APIs de alto throughput
Simplicidade	Código mais enxuto, menos boilerplate
Produtividade	Time-to-market reduzido (crucial para prazo das eleições)
Documentação automática	Integração nativa com Swagger/OpenAPI

Entity Framework Core	
Produtividade	Migrations automáticas, LINQ queries type-safe
Testabilidade	Suporte a InMemory provider para testes de integração
Maturidade	ORM mais utilizado no ecossistema .NET
Flexibilidade	Suporta múltiplos providers (PostgreSQL, SQL Server, SQLite)

FluentValidation	
Separação de responsabilidades	Validação desacoplada dos endpoints
Testabilidade	Validators podem ser testados unitariamente
Mensagens customizáveis	Feedback claro para usuários da API

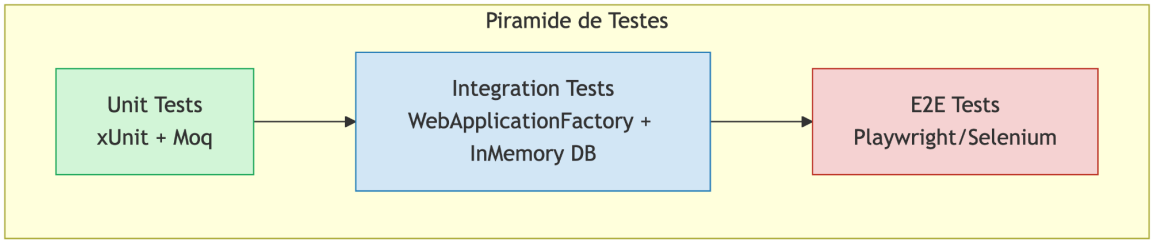
Blazor WebAssembly	
Full-stack C#	A equipe de devs já conhece .NET/C#
Sem JavaScript	Reduz complexidade e curva de aprendizado
SPA moderna	Experiência de usuário fluida
Componentes reutilizáveis	Razor Components para UI consistente

Para Usuários de Negócio

Requisito de Negócio	Solução Técnica	Benefício
Milhões de respostas	PostgreSQL + Connection Pooling	Banco robusto para alta carga de escrita
Disponibilidade 24/7	Docker containers + Health checks	Fácil deploy e monitoramento
Resultados em tempo real	API REST eficiente	Agregação rápida de resultados
Prazo curto	.NET stack unificado	Equipe já capacitada, sem curva de aprendizado
Custos controlados	PostgreSQL open-source	Sem custos de licenciamento

Estratégia de Testes

Pirâmide de Testes



Testes de Integração da API

```
public class SurveyEndpointsTests : IClassFixture<WebApplicationFactory<Program>>
{
    [Fact]
    public async Task CreateSurvey_WithValidData_ReturnsCreated()
    {
        var request = new CreateSurveyRequest
        {
            Title = "Pesquisa Eleitoral 2026",
            Questions = [new CreateQuestionRequest { Text = "Em quem você votaria?" }]
        };
        var response = await _client.PostAsJsonAsync("/api/surveys", request);
        response.StatusCode.Should().Be(HttpStatusCode.Created);
    }
}
```

Cobertura de testes implementada:

- Testes com banco InMemory (isolamento)

- Validação de endpoints

- Cenários de erro

Testes do Entity Framework

```
// Configuração para testes com InMemory Database
services.AddDbContext<ApplicationDbContext>(options =>
    options.UseInMemoryDatabase("TestDb"));
```

Por que InMemory para testes

- Execução rápida (sem I/O de disco)

- Isolamento entre testes

- Não requer container PostgreSQL

Conclusão

A arquitetura proposta atende aos requisitos do sistema de questionários online:

Requisito	Atendimento
Escala	PostgreSQL + APIs stateless + Docker
Prazo	Stack .NET unificado (equipe já capacitada)
Manutenibilidade	Clean Architecture + Testes automatizados
Custo	Tecnologias open-source
NET Framework	ASP.NET Core + EF Core + Blazor

A solução é pragmática, focando em entregar valor no prazo das eleições, utilizando tecnologias maduras que a equipe de desenvolvedores já domina.