Centro Universitário do Sul de Minas – UNIS
Curso: Ciência da Computação – 4° Período
Disciplina: Mobile Technologies
Professor: Pedro Palmuti

RELATÓRIO TÉCNICO

Comparativo de Desenvolvimento do App *Paridade*

Equipe:

- Fernanda Laudares Silva
- Julia Ragi Beltrão
- Lucas Silva Ciacci
- Sara Souza Veríssimo

Comparativo de Desenvolvimento do App *Paridade*

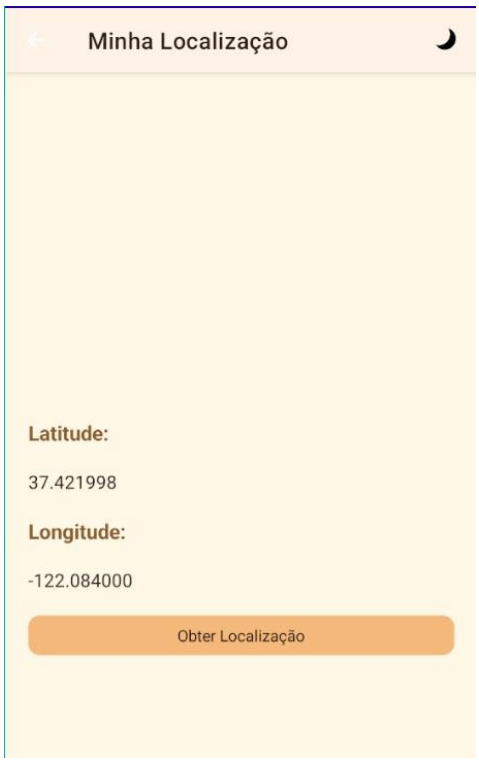
O presente trabalho tem como objetivo analisar comparativamente a experiência de desenvolvimento e o desempenho de um aplicativo simples, denominado *App Paridade*, construído em duas abordagens distintas: o framework .NET MAUI, utilizando C# no Visual Studio 2022, e o desenvolvimento nativo em Android, com Kotlin no Android Studio. A motivação central foi compreender as diferenças práticas entre essas tecnologias, tanto em aspectos de produtividade e facilidade de manutenção, quanto em métricas técnicas como tempo de inicialização, tamanho do pacote e consumo de memória.

A proposta do aplicativo foi a mesma para ambas as plataformas, a fim de reduzir vieses: uma navegação básica entre lista e detalhe, uma lista filtrável com vinte itens locais em JSON, integração com a API pública do IBGE, acesso a um recurso nativo (no caso, a geolocalização), alternância entre tema claro e escuro e a implementação de um teste unitário simples. Ao manter os mesmos assets, cores e layout, buscou-se garantir que a comparação se concentrasse na tecnologia e não nas escolhas de design.

- **Lista filtrável com vinte itens locais:**
    - **.NET MAUI :**

    - **ANDROID NATIVO :**

- **Geolocalização:**
  - **.NET MAUI :**

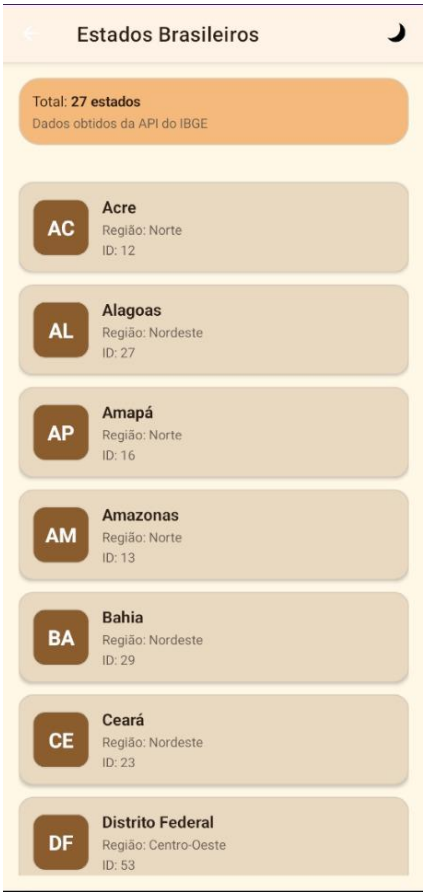


ANDROID NATIVO :

- **Tema Dark/Light :**
  - **.NET MAUI :**

ANDROID NATIVO :

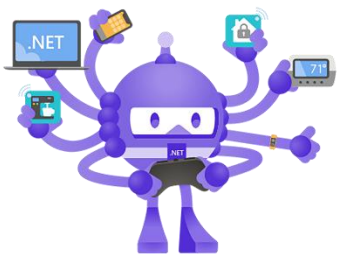
- **Requisição HTTP para API pública (IBGE)**
  - **NET MAUI :**



ANDROID NATIVO :

- **Desenvolvimento:**

O ambiente de desenvolvimento em .NET MAUI contou com o Visual Studio 2022 (versão mínima 17.8), rodando no Windows 11 Pro 64 bits, com o Android SDK 34 configurado para emuladores Google APIs. Já no desenvolvimento nativo, utilizou-se o Android Studio Narwhal 3, com Kotlin 1.9.20 e o mesmo SDK (minSdk 24, targetSdk 34), em uma máquina com Windows 10. Para padronização, foram escolhidos emuladores semelhantes (Pixel 5 e Pixel 7) e desativados recursos como Hot Reload e Instant Run durante a coleta de métricas.



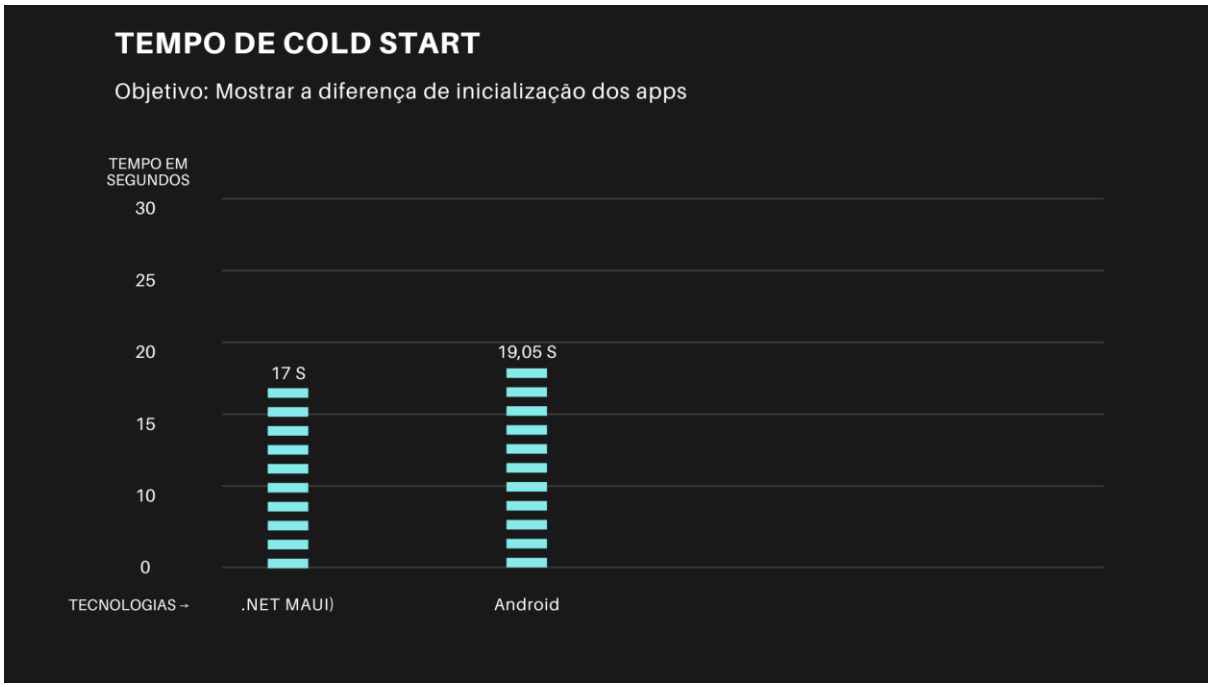
.NET MAUI

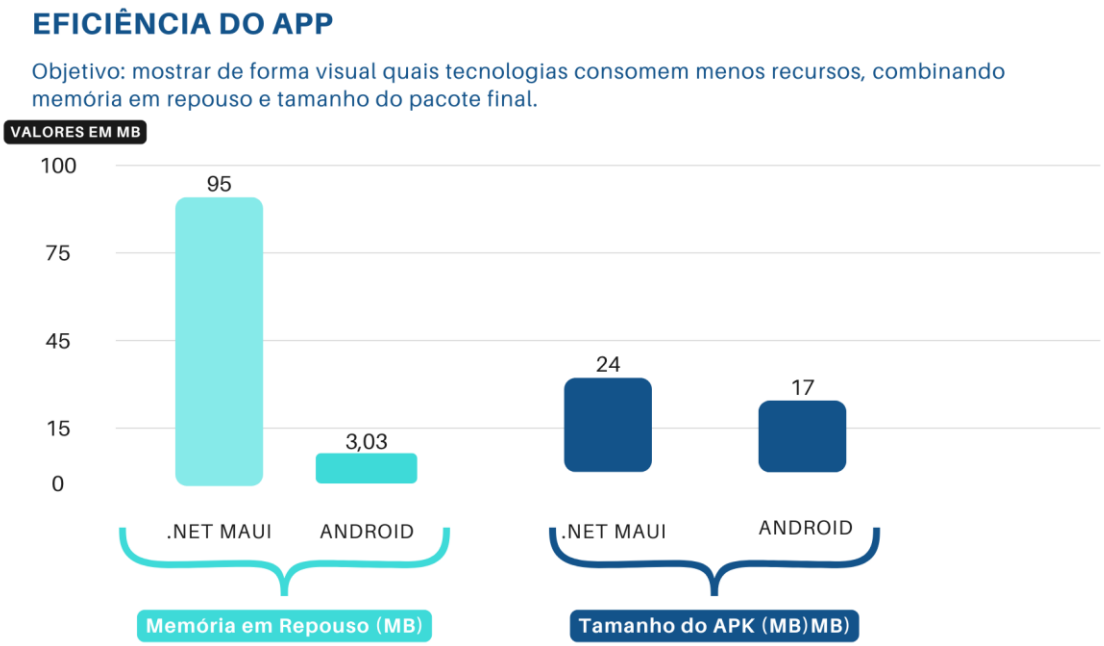
ANDROID STUDIO

Os resultados obtidos mostraram diferenças relevantes. No .NET MAUI, o aplicativo apresentou tempo de inicialização de aproximadamente 17 segundos, tamanho de pacote em torno de 24 MB e consumo de memória de 95 MB em repouso, após 30 segundos na tela de lista. Já no Android nativo, os resultados foram mais enxutos em desempenho: o cold start foi de 19,05 segundos, o tamanho final do APK ficou em 17 MB e o consumo de memória, significativamente menor, em torno de 3 MB. Em relação ao esforço de desenvolvimento, ambos os projetos demandaram cerca de seis horas de trabalho, divididas em blocos de 30 minutos.

RESUMO GERAL DAS MÉTRICAS

Métrica	.NET MAUI(C#)	ANDROID NATIVO (Kotlin)
Cold Start (s)	17,0	19,05
Tamanho do APK (MB)	24	17
Memória em repouso (MB)	95	3,03
Tempo de desenvolvimento (h)	6	6





Apesar de ambos os ambientes terem permitido a implementação de todas as funcionalidades previstas, a experiência de desenvolvimento se mostrou distinta. No .NET MAUI, a produtividade foi beneficiada pelo Hot Reload e pela familiaridade com C#, permitindo ajustes rápidos de interface. Entretanto, o consumo elevado de memória e o tamanho maior do aplicativo são pontos de atenção para cenários que exigem eficiência. Já no Android nativo, a curva de aprendizado foi um pouco mais acentuada, mas o controle direto sobre APIs e permissões garantiu maior estabilidade e desempenho, ainda que o processo de ajustes e testes fosse menos ágil.

Do ponto de vista acadêmico, este estudo revela um clássico dilema em desenvolvimento mobile: frameworks multiplataforma, como o MAUI, tendem a acelerar o desenvolvimento e facilitar a manutenção de projetos que miram diferentes plataformas, mas ainda apresentam perdas de desempenho quando comparados a soluções nativas. O Android nativo, por sua vez, se mantém como a opção mais eficiente em termos de recursos, ainda que exija maior dedicação e conhecimento específico.

Conclui-se que ambas as abordagens possuem pontos fortes e limitações. O .NET MAUI pode ser recomendado em cenários em que o objetivo principal seja alcançar múltiplas plataformas com rapidez e manutenção unificada. Já o Android nativo deve ser a escolha preferencial quando o desempenho e a eficiência são fatores críticos, como em aplicações que precisam rodar em dispositivos mais modestos ou lidar com processamento intensivo.

Referências

- MICROSOFT. **.NET Multi-platform App UI (.NET MAUI) Documentation**. Disponível em: https://learn.microsoft.com/dotnet/maui/. Acesso em: 17 set. 2025.

- GOOGLE. **Android Developers – Official Documentation**. Disponível em: https://developer.android.com/docs. Acesso em: 17 set. 2025.

- INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA (IBGE). **Serviços de Dados Abertos – API de Localidades**. Disponível em: https://servicodados.ibge.gov.br/api/docs/. Acesso em: 17 set. 2025.

- VILLAR, Miguel de Icaza; VERMEER, Gerald. **Cross-platform mobile development with .NET MAUI**. Packt Publishing, 2022.