



CURSO DE SISTEMAS DE INFORMAÇÃO

Júlio César Barcelo Monteiro

IMPLEMENTAÇÃO DE UMA FERRAMENTA DE PERSISTÊNCIA DE  
DADOS MULTIPLATAFORMA EM TYPESCRIPT PARA  
APLICATIVO WEB E MOBILE

Palmas, TO

2023

Júlio César Barcelo Monteiro

# IMPLEMENTAÇÃO DE UMA FERRAMENTA DE PERSISTÊNCIA DE DADOS MULTIPLATAFORMA EM TYPESCRIPT PARA APLICATIVO WEB E MOBILE

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação da Universidade Estadual do Tocantins - UNITINS, como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação, sob orientação da professora Me. Tamirys Virgulino Ribeiro Prado.

Palmas, TO


2023

**JÚLIO CÉSAR BARCELO MONTEIRO**

**IMPLEMENTAÇÃO DE UMA FERRAMENTA DE PERSISTÊNCIA DE DADOS  
MULTIPLATAFORMA EM TYPESCRIPT PARA APLICATIVO WEB E MOBILE**

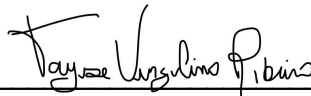
Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação da Universidade Estadual do Tocantins - UNITINS, como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação, sob orientação da professora Me. Tamirys Virgulino Ribeiro Prado.

Aprovado pela Banca Examinadora em 12 de junho de 2023

Documento assinado digitalmente  
 TAMIRYS VIRGULINO RIBEIRO PRADO  
Data: 19/07/2023 20:01:37-0300  
Verifique em <https://validar.it.gov.br>


---

Prof. Orientador



---

Prof. Membro da Banca

Documento assinado digitalmente  
 CARLOS HENRIQUE CORREA TOLENTINO  
Data: 24/07/2023 22:11:20-0300  
Verifique em <https://validar.it.gov.br>

---

Prof. Membro da Banca

Palmas, 2023

**Dados Internacionais de Catalogação na Publicação (CIP)**  
**Sistema de Bibliotecas da Universidade Estadual do**  
**Tocantins**

---

B242i	<p>BARCELO MONTEIRO, Júlio César</p> <p>Implementação de uma ferramenta de persistência de dados multiplataforma em Typescript para aplicativo Web e Mobile. Júlio César Barcelo Monteiro. - Palmas, TO, 2023</p> <p>Monografia Graduação - Universidade Estadual do Tocantins – Câmpus Universitário de Palmas - Curso de Sistemas de Informação, 2023.</p> <p>Orientadora: Tamirys Virgulino Ribeiro Prado</p> <p>1. Persistência de dados. 2. Javascript. 3. Desenvolvimento híbrido. 4. multiplataforma.</p>
-------	--

**CDD 610.7**

---

TODOS OS DIREITOS RESERVADOS – A reprodução total ou parcial, de qualquer forma ou por qualquer meio deste documento é autorizado desde que citada a fonte. A violação dos direitos do autor (Lei nº 9.610/98) é crime estabelecido pelo artigo 184 do Código Penal.

Elaborado pelo sistema de geração automática de ficha catalográfica da UNITINS com os dados fornecidos pelo(a) autor(a).

## ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO DO CURSO DE SISTEMAS DE INFORMAÇÃO DA FUNDAÇÃO UNIVERSIDADE ESTADUAL DO TOCANTINS - UNITINS

Aos 12 dias do mês de **Junho** de **2023**, reuniu-se na Fundação Universidade Estadual do Tocantins, Câmpus Palmas, Bloco B, às **11 horas**, sob a Coordenação do Professora **Tamirys Virgulino Ribeiro Prado**, a banca examinadora de Trabalho de Conclusão de Curso em Sistemas de Informação, composta pelos examinadores Professora **Tamirys Virgulino Ribeiro Prado** (Orientadora), Professora **Tayse Virgulino Ribeiro** e Professor **Carlos Henrique Correia Tolentino**, para avaliação da defesa do trabalho intitulado “**Implementação de uma Ferramenta de Persistência de Dados Multiplataforma em Typescript para Aplicativo Web e Mobile**” do acadêmico **Júlio César Barcelo Monteiro** como requisito para aprovação na disciplina Trabalho de Conclusão de Curso II (TCC II). Após exposição do trabalho realizado pelo acadêmico e arguição pelos Examinadores da banca, em conformidade com o disposto no Regulamento de Trabalho de Conclusão de Curso em Sistemas de Informação, a banca atribuiu a pontuação: 8.5 .

Sendo, portanto, o Acadêmico: (X) Aprovado ( ) Reprovado

Assinam esta Ata:

Professor Orientador: **Tamirys Virgulino Ribeiro Prado**

Examinador: **Carlos Henrique Corrêa Tolentino**

Examinador: **Tayse Virgulino Ribeiro**



Documento assinado digitalmente  
TAMIRYS VIRGULINO RIBEIRO PRADO  
Data: 19/07/2023 20:01:37-0300  
Verifique em <https://validar.iti.gov.br>



Documento assinado digitalmente  
CARLOS HENRIQUE CORREA TOLENTINO  
Data: 24/07/2023 22:11:20-0300  
Verifique em <https://validar.iti.gov.br>

**Tamirys Virgulino Ribeiro Prado**  
**Presidente da Banca Examinadora**

Coordenação do Curso de Sistemas de Informação

---

## Resumo

Neste trabalho, foi abordada a questão da persistência de dados em aplicativos, com foco especial nos desafios encontrados em ambientes híbridos. Foi proposta e desenvolvida uma ferramenta multiplataforma, que passou por uma análise detalhada utilizando uma metodologia de pesquisa aplicada. Os resultados obtidos comprovaram a eficácia da ferramenta, demonstrando suas vantagens e benefícios para o desenvolvimento de software. Além disso, a publicação do pacote "persistor-node" no npmjs foi um resultado significativo, trazendo visibilidade, reputação e promovendo a colaboração da comunidade, enriquecendo ainda mais a solução desenvolvida.

**Palavras-chave:** persistência, multiplataforma, desenvolvimento híbrido.

## **Lista de Tabelas**

Tabela 1 - Objetos de pesquisa relacionados

20

## Lista de Ilustrações

Figura 1. O que é typescript	14
Figura 2 - React Native Android e iOS	18
Algoritmo 1 - Requisição simples e sem cacheamento	23
Figura 3 - Diagrama de sequência da requisição sem cache	23
Algoritmo 2 - Implementação de cache numa requisição	24
Figura 4 - Diagrama de sequência do uso de cache para requisições	25
Figura 5 - Diagrama de Containers do funcionamento do pacote em cada ambiente	27
Figura 6 - Gráfico dos resultados obtidos no primeiro teste via API no navegador	32
Figura 7 - Gráfico dos resultados obtidos no primeira teste via cache no navegador	33
Figura 8 - Gráfico dos resultados obtidos no primeiro teste via API no mobile	34
Figura 9 - Gráfico dos resultados obtidos no primeiro teste via cache no mobile	35
Figura 10 - Gráfico dos resultados obtidos no segundo teste via API no navegador	36
Figura 11 - Gráfico dos resultados obtidos no segundo teste via cache no navegador	37
Figura 12 - Gráfico dos resultados obtidos no segundo teste via API no mobile	38
Figura 13 - Gráfico dos resultados obtidos no segundo teste via cache no mobile	39
Figura 14 - Gráfico dos resultados obtidos no terceiro teste via API no navegador	41
Figura 15 - Gráfico dos resultados obtidos no terceiro teste via cache no navegador	42
Figura 16 - Gráfico dos resultados obtidos no terceiro teste via API no mobile	43
Figura 17 - Gráfico dos resultados obtidos no terceiro teste via cache no mobile	44
Figura 18 - Resultado dos testes realizados utilizando o SonarQube	46



## **Lista de Abreviaturas e Siglas**

- SO: Sistema Operacional
- IoT: Internet of Thing ou Internet das coisas
- SSD: Solid State Drive ou Dispositivo de Estado Sólido
- HTTP: Hyper Text Transfer Protocol ou Protocolo de transferência de hipertexto
- API: Application Programing Interface ou Interface de Programação de Aplicativos
- RAM: Random Access Memory ou Memória de Acesso Aleatório
- Gb: Gigabyte
- Tb: Terabyte
- ADB: Android Debug Bridge ou Ponte de Depuração Android
- NVMe: Non Volatile Memory Express ou Memória Não Volátil Expressa
- IDE: Integrated Development Environment ou Ambiente de Desenvolvimento integrado
- iOS: iPhone Operating System
- ms: Milissegundos
- $\mu$ s: Microssegundos

# Sumário

<b>Resumo</b>	<b>5</b>
<b>Lista de Tabelas</b>	<b>6</b>
<b>Lista de Ilustrações</b>	<b>7</b>
<b>Lista de Abreviaturas e Siglas</b>	<b>8</b>
<b>Sumário</b>	<b>9</b>
<b>1. Introdução</b>	<b>10</b>
1.1. Problema	11
1.2. Justificativa	12
<b>2. Objetivos e Hipóteses</b>	<b>13</b>
2.1. Objetivo Geral	13
2.2. Objetivos Específicos	13
2.3. Hipóteses	13
<b>3. Referencial Teórico</b>	<b>14</b>
3.1. Typescript	14
3.2. Javascript	15
3.3. Interfaces	15
3.4. Arquitetura	15
3.5. Estrutura de dados	16
3.6. Padrões de projeto	16
3.7. Persistência de dados	16
3.8. Multiplataforma	17
3.9. React Native	18
Quadro comparativo	20
<b>4. Metodologia</b>	<b>22</b>
4.1. Métodos	22
4.2. Materiais	29
<b>5. Resultados</b>	<b>30</b>
5.1. Publicação da Ferramenta	30
5.2. Testes Realizados	30
Primeiro Teste	31
Segundo Teste	36
Terceiro Teste	40
Testes de qualidade de código	45
<b>6. Conclusões</b>	<b>48</b>
<b>Referências</b>	<b>49</b>

## 1. Introdução

Este trabalho aborda a implementação de uma ferramenta de persistência de dados em um ambiente multiplataforma, visando aprimorar o desempenho e a eficiência dos sistemas desenvolvidos. A utilização de tecnologias híbridas, como o React e o React Native, tem se mostrado promissora para o desenvolvimento de aplicações que funcionam em diferentes dispositivos. Essas tecnologias permitem a criação de interfaces modernas e responsivas, além de compartilhar código entre os ambientes, aumentando a produtividade e facilitando a manutenção.

A proposta deste trabalho é explorar a implementação de uma ferramenta que padronize a persistência de dados em aplicações multiplataforma, abrangendo o cacheamento de respostas da API e a otimização das chamadas de dados. O objetivo é melhorar o desempenho e reduzir o consumo de recursos. Foi realizado um estudo de viabilidade e eficácia dessa abordagem em um projeto de conclusão de curso, analisando os resultados obtidos em termos de desempenho e eficiência.

Um dos resultados significativos obtidos ao longo deste trabalho foi a publicação de um pacote JavaScript chamado "persistor-node" no npmjs, um repositório amplamente utilizado pela comunidade. Essa publicação é um marco importante para a disseminação da solução desenvolvida, pois promove visibilidade, reputação e facilita a colaboração da comunidade com o código criado. Além disso, a ferramenta incorpora nativamente o controle de tempo de duração das informações armazenadas, simplificando o gerenciamento e a manutenção dos dados.

Dessa forma, este trabalho se propõe a contribuir para o avanço do desenvolvimento de aplicativos web e mobile, fornecendo uma solução eficaz e abrangente para a persistência de dados em ambientes híbridos. Através da utilização da ferramenta multiplataforma desenvolvida, espera-se melhorar as condições de trabalho dos desenvolvedores, proporcionar uma experiência mais fluida para os usuários e permitir a disseminação e colaboração da comunidade em torno dessa solução.

## 1.1. Problema

Segundo o estudo realizado por (KOKOLIS et al., 2021), o armazenamento em nuvem visa garantir a segurança e disponibilidade dos dados para o usuário. No entanto, um dos principais desafios enfrentados por esses serviços é a velocidade da rede, mesmo em redes internas. Com os avanços tecnológicos em dispositivos de armazenamento sólido, como SSDs, a transmissão de dados pela rede tornou-se mais lenta em comparação à leitura dos dados em si. Isso pode levar à insatisfação do cliente final, uma vez que muitos desses dados poderiam estar disponíveis localmente, evitando a latência causada pela rede.

Existem diversos trabalhos que implementam estratégias para persistência de dados local em diferentes níveis. Por exemplo, (BELTRÃO, 2014), propõe um modelo voltado para o ambiente Android chamado JPDroid, enquanto (AL-ROUSAN; AL-SHARGABI; ABUALESE, 2019), apresentam um modelo especializado em criptografia de dados no navegador e (CERQUEIRA, 2017) tem o compartilhamento e persistência de dados entre dispositivos geograficamente próximos. Cada um desses trabalhos utiliza estratégias específicas para a linguagem e o propósito de sua aplicação, ampliando a complexidade no desenvolvimento de aplicações para diferentes cenários e ambientes.

No entanto, com a popularização do desenvolvimento de aplicações híbridas, surge a necessidade de imaginar e projetar soluções viáveis que possam ser executadas em diferentes ambientes. Até o momento, não há uma ferramenta que atenda a todas as necessidades levantadas, com uma estratégia que permita sua execução em diversos cenários e ambientes. As ferramentas especializadas em um único ambiente impossibilitam a centralização da lógica em uma única solução, o que exige que os desenvolvedores criem soluções específicas para cada ambiente, em vez de se concentrarem nos problemas de suas aplicações.

## 1.2. Justificativa

Segundo (CHERNY, 2019) e em tradução livre “Typescript é a linguagem que vai dominar as próximas gerações de aplicações web, mobile, NodeJS e Internet das coisas (IoT)” ainda segundo Cherny, a linguagem possibilita programas mais seguros evitando erros comuns na hora de programar, simplificando futuras atualizações na lógica de funcionamento do programa.

Segundo (DRISCOLL et al., 1989), persistir os dados de uma aplicação exige uma estrutura que permita velocidade de armazenamento, economia de espaço e que possa ser acessada e modificada. Driscoll faz isso citando modelos que sustentam um ou mais dos pontos sugeridos. Complementando, o trabalho de (KOKOLIS et al., 2021) descreve como algumas dessas estratégias citadas são utilizadas também por serviços de armazenamento em nuvem para a persistência de seus dados de maneira eficiente, viabilizando velocidade na leitura e na escrita das informações.

No texto apresentado, é possível observar que a persistência de dados é uma necessidade já abordada por outros trabalhos e possui algumas implementações consolidadas. O objetivo desse trabalho é demonstrar a implementação da ferramenta desenvolvida em trabalho anterior complementado por esse e demonstrar os ganhos que se obteve com a adoção dessa ferramenta.

Portanto, esse trabalho de conclusão de curso se propõe a comprovar que a adoção de tal ferramenta trará benefícios para a rotina de desenvolvimento de softwares, demonstrando com um estudo de casos aplicado ao ambiente web e mobile como persistência de dados em diferentes plataformas melhora a experiência do usuário, mantendo a segurança, eficiência e padronização no código.

## 2. Objetivos e Hipóteses

### 2.1. Objetivo Geral

Este trabalho de conclusão de curso visa demonstrar, por meio de um estudo de casos, a aplicação prática de uma ferramenta multiplataforma para persistência de dados local. Essa ferramenta já teve viabilidade comprovada em trabalho anterior e agora busca evidenciar os benefícios que ela oferece ao desenvolvedor, tais como: desempenho na execução da aplicação, simplicidade no uso da funcionalidade e controle sobre os processos que serão executados.

### 2.2. Objetivos Específicos

Para avaliar os resultados, são traçados objetivos específicos para validar o cumprimento do objetivo geral do projeto, a respeito desses têm-se que a ferramenta deve:

- Reduzir o tempo de execução das aplicações web e mobile desenvolvidas quando utilizando cache;
- Facilitar integração e interoperabilidade entre aplicações que executam em diferentes plataformas e precisam persistir dados;
- Atingir os padrões de qualidade exigidos para um projeto de software segundo algumas métricas existentes

### 2.3. Hipóteses

As hipóteses levantadas a respeito do tema discutido nesse trabalho, para validar o uso da ferramenta ou rejeitá-lo, são:

- $H_0$  A ferramenta é capaz de melhorar a experiência do usuário das aplicações produzida em javascript;
- $H_1$  A ferramenta não oferece vantagens significativas;

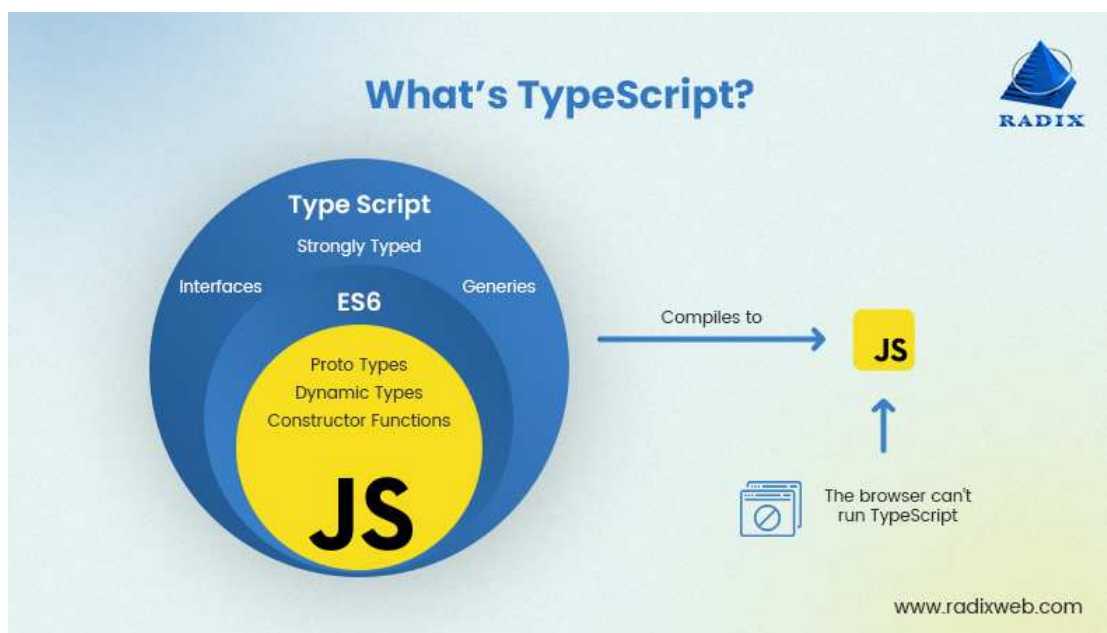
### 3. Referencial Teórico

Este tópico aborda os elementos teóricos e técnicos necessários para o desenvolvimento da pesquisa. Pois recorre a conceitos já existentes, portanto esse capítulo procura apresentar e descrever os principais conceitos que serão trabalhados.

#### 3.1. Typescript

Conforme a documentação oficial, typescript é um conjunto de ferramentas com todos os recursos presentes no javascript aliado a outras validações para benefício do desenvolvedor. De maneira resumida são adicionadas diversas validações ao código que são realizadas antes dele ser executado, aproximando o javascript das qualidades que outras linguagens, fortemente tipadas, possuem e aumentando a previsibilidade do código.

Figura 1. O que é typescript



Fonte: <https://radixweb.com/blog/typescript-vs-javascript>

A Figura 1 exemplifica a relação entre o Typescript e o Javascript, destacando como o primeiro adiciona recursos ao segundo, mas sem remover suas funcionalidades, e ao final, é traduzido em um código final, em javascript.

### 3.2. Javascript

Segundo a documentação do Typescript, Javascript ou ECMAScript como também é conhecido, é uma linguagem que começou sua vida como uma linguagem para validações simples que poderiam ser feitas do lado do cliente em seu próprio navegador, mas que se popularizou e ganhou espaço no desenvolvimento fora dos navegadores com a criação do NodeJS, tendo hoje espaço em todas as áreas de desenvolvimento, seja ela Mobile, Web, NodeJS ou IoT.

### 3.3. Interfaces

Segundo a documentação oficial do typescript, interfaces são uma maneira poderosa de definir contratos dentro de seu código, bem como contratos com código fora de seu projeto. Portanto, são criadas na aplicação para definir como as estruturas geradas deverão se comportar, especificando quais atributos existirão, bem como seus métodos e quais serão os tipos de dados retornados ao acessar cada propriedade ou executar métodos. Além disso, o uso de interfaces melhora a previsibilidade de código durante sua escrita, ajudando tanto o programador quanto a IDE a entender e prever como o código irá se comportar, além de servir como documentação em alguns casos. Também podem ser utilizadas para definir contratos que outros desenvolvedores devem seguir na hora de implementar algum recurso desenvolvido por você ou por sua equipe.

### 3.4. Arquitetura

A Arquitetura de software diz respeito à declaração de componentes do sistema e suas propriedades, como cada componente se relaciona um com o outro. Diz respeito à construção de toda a documentação necessária para que futuros desenvolvedores que precisem trabalhar com seu código ou dar manutenção no mesmo. A origem da arquitetura de software deriva do trabalho de (DIJKSTRA, 1976) e de (PARNAS, 1988) que demonstraram em seus trabalhos a importância de se descrever as estruturas de um sistema.



### 3.5. Estrutura de dados

Segundo (LIPSCHUTZ, 2014), dados são valores simples ou conjuntos de valores simples, que podem ser organizados de diferentes formas, e ele define que o modelo lógico ou matemático de uma organização de dados é então o que definimos como estrutura de dados. Uma estrutura de dados é uma forma de representar uma estrutura complexa de dados num computador e é necessária por que os computadores só conseguem compreender dados numericamente, então são definidas estruturas que seguem alguns padrões para que consigam simbolizar linguagem não numérica, e é dessa forma que é possível processar itens como textos, imagens, músicas e diversas outras estruturas em um computador, celular ou quaisquer outros dispositivos.

### 3.6. Padrões de projeto

Segundo (GAMMA et al., 1994), padrões de projeto são paradigmas criados para resolver problemas comuns que ocorrem durante o desenvolvimento de uma aplicação, por uma das abordagens descritas em seu livro. Conhecidos como “Gang of Four” ou “Gangue dos Quatro” em tradução livre, (GAMMA et al., 1994) descreveram uma série de problemas comuns durante o desenvolvimento, catalogando-os conforme sua abordagem atua para resolver o problema em questão, sendo as categorias: estruturais, comportamentais e criacionais.

### 3.7. Persistência de dados

De acordo com (KAPLAN, 2004), e em tradução livre, dado persistente, no campo de processamento de dados, descreve informações acessadas raramente e que provavelmente não serão modificadas. Deste modo, dados persistentes podem ser compreendidos como informações contextuais que a aplicação não tem a necessidade de validar a todo momento. Portanto, serve para guardar dados como, por exemplo: autenticação de usuário, para evitar que o mesmo precise criar uma sessão a cada acesso; carrinho de compras, para ser validado numa requisição única durante a finalização da compra ou os últimos termos utilizados em uma pesquisa, para acesso

rápido aos resultados encontrados anteriormente. Desse modo, a aplicação final melhora a experiência do usuário evitando requisições desnecessárias e demoradas.

### 3.8. Multiplataforma

Software Multiplataforma, também conhecido como Software Independente de Plataforma ou Software de Plataforma Cruzada, é um termo usado para exemplificar sistemas capazes de serem executados em mais de um ambiente, isso é, que pode ser executado tanto num celular, quanto num computador, e/ou videogame. Alguns exemplos de software multiplataforma mais comuns são jogos, programas, redes sociais e aplicativos móveis.

(ROY CHOUDHARY, 2014), afirma que é esperado que os softwares mais modernos consigam executar em uma diversidade de ambientes, seja ele um dispositivo móvel ou web, independentemente da capacidade de processamento desse dispositivo.

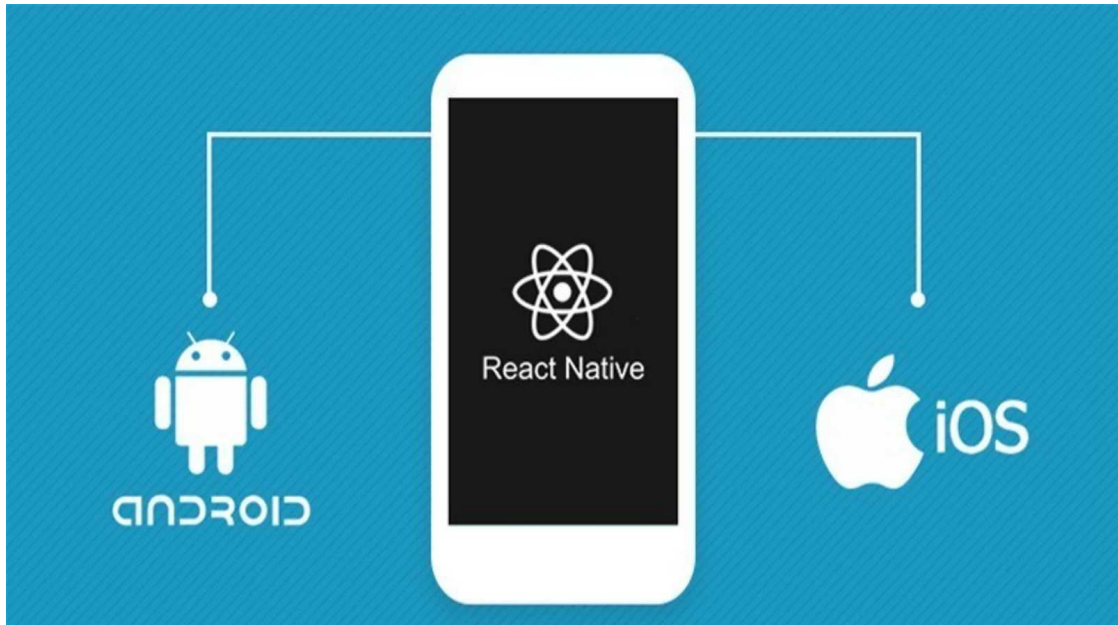
Um ramo do desenvolvimento multiplataforma focando em dispositivos móveis é o desenvolvimento híbrido, que implementa aplicações que funcionam em tanto ambiente Android quanto iOS e vem sendo adotado em força por diversas (micro) empresas e startups que atendam o público, uma vez que o número de usuários em qualquer das plataformas representa uma parte considerável de seus consumidores, portanto, estas empresas não podem ignorar nenhum dos ambientes.

O desenvolvimento específico para cada plataforma tem um custo muito alto, deste modo muitas dessas empresas optam por adotar uma tecnologia para desenvolvimento híbrido, reduzindo os custos com o desenvolvimento e com a manutenção de seu produto.

### 3.9. React Native

De acordo com (BODUCH; DERKS, 2020), React native é uma biblioteca para desenvolvimento mobile que foi desenvolvida pelo Facebook tendo em vista o sucesso de sua antecessora web, o React.

Figura 2 - React Native Android e iOS



Fonte: <https://dev.to/alexandrefreire/react-native-aprenda-o-basico-643>

A Figura 2 demonstra de maneira lúdica a capacidade do React Native de ser executado no ambiente android e iOS utilizando do mesmo código em javascript.

O React Native sintetiza o mesmo conceito do React para construção de telas, sendo que sua ideia é então aproveitar a facilidade de desenvolvimento de telas que o React proporciona aliado a baixa curva de aprendizado que um desenvolvedor que já trabalhou com a ferramenta na web teria para aprender então os novos conceitos que fossem necessários para trabalhar com a ferramenta mobile de desenvolvimento híbrido. Está presente hoje não só nos ambientes Android e iOS, mas também no Windows, MacOS, Linux e até mesmo nos navegadores.

Portanto, sua versatilidade permite que o desenvolvedor utilize todo o poder do Framework React, já consolidado no desenvolvimento web, para produzir telas de

aplicações mobile, com acesso a funcionalidades do SO que seriam impossíveis se a aplicação fosse acessada por um site

## Quadro comparativo

É exposto um quadro que analisa outras pesquisas realizadas na área de persistência de dados, neste, destacam-se algumas colunas para a elaboração do mesmo, sendo elas: padrões de projeto, mobile, web, servidor e multiplataforma. O primeiro, diz respeito ao uso de alguma boa prática levantada por (GAMMA et al., 1994) em seu livro “*Design Patterns: Elements of Reusable Object-Oriented Software*”, o que segundo o autor do livro, melhora a qualidade final do código produzido e aumenta a capacidade de manutenção no mesmo. Os três seguintes - mobile, web e servidor - tem relação a plataforma ou ambiente ao qual o projeto documentado se foca em atender. Por fim, a coluna multiplataforma sintetiza as três colunas anteriores numa única afirmação: “esse projeto roda em qualquer um dos ambientes elencados?”

A leitura da tabela é feita consoante o que se segue: (+) presente, (-) ausente.

Tabela 1 - Objetos de pesquisa relacionados

Trabalho / Ferramenta	Padrões de projeto	Mobile	Web	Servidor	Multiplataforma
(AL-ROUSAN; AL-SHARGABI; ABUALESE, 2019)	+	-	+	-	-
(BELTRÃO, 2014)	+	+	-	-	-
(LANGSARI; ROCHIMAH; AKBAR, 2018)	+	-	-	+	-
(CERQUEIRA, 2017)	+	+	-	-	-
(NETO et al., 2017)	+	+	-	-	-
(ROQUE, 2013)	+	-	+	-	-
(MACCHERONE, 2022)	+	-	-	+	-

Fonte: O Autor

Através do quadro é possível perceber duas dinâmicas que ocorrem nos trabalhos existentes: O primeiro é o fator comum de todos os projetos na adoção de padrões de projeto para o desenvolvimento. Isso pelo fato de que, segundo (GAMMA et al., 1994), são soluções para problemas comuns durante o desenvolvimento de qualquer projeto de software, e em

alguns casos garantem o máximo de eficiência para o desenvolvimento e futura manutenção do código; O Segundo é que nenhum dos trabalhos está disponível nos três ambientes considerados na tabela, portanto, nenhum é multiplataforma, comprovando uma lacuna existente cujo presente projeto se propõe a resolver caso se prove viável.

## 4. Metodologia

O trabalho usa a metodologia de pesquisa exploratória descritiva. Segundo (CARLOS GIL, 2002) A pesquisa exploratória descritiva é uma modalidade de pesquisa que busca explorar e descrever as características e propriedades de um fenômeno ou problema. Esta metodologia é utilizada quando o pesquisador não possui informações suficientes sobre o assunto em questão e busca conhecer os elementos que compõem o objeto de estudo, suas dimensões, variáveis e inter-relações.

### 4.1. Métodos

Para este trabalho de conclusão de curso, foi realizada a implementação da ferramenta de persistência de dados em dois ambientes distintos: uma aplicação web utilizando a biblioteca React e outra móvel utilizando o framework React Native, ambos consumindo uma API de seriados e filmes fornecida pelo serviço The Movie DB<sup>1</sup>. Então obteve-se os dados de tempo de duração das requisições realizadas, com e sem estratégia de cacheamento, e comparou-se os resultados obtidos em cada ambiente, observando os dados e comentando seus resultados a fim de compreender o que significam.

Dentre as principais vantagens que pode se obter da implementação dessa ferramenta, destaca-se o recurso de cacheamento das respostas da API na aplicação, que resulta em um desempenho aprimorado e na redução de chamadas desnecessárias à API. Adicionalmente, a ferramenta possibilita a reutilização de código entre os ambientes React e React Native, o que agiliza o processo de desenvolvimento e facilita a manutenção dos códigos em ambos os ambientes.

Ao armazenar as respostas da API em cache, evitamos repetidas chamadas à mesma requisição, reduzindo a carga no servidor e diminuindo o tempo de resposta para o usuário. Além disso, o cacheamento permite que a aplicação acesse os dados localmente, eliminando a necessidade de uma nova requisição ao servidor em casos em que os dados não foram alterados. Isso resulta em uma experiência mais ágil e responsiva para o usuário, principalmente em situações de acesso à internet instável ou lenta.

---

<sup>1</sup> <https://www.themoviedb.org/>

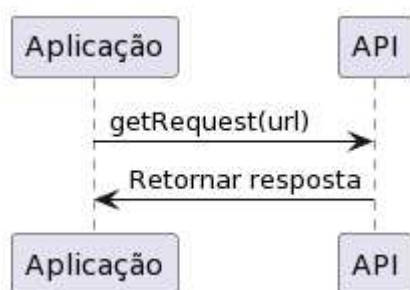
### Algoritmo 1 - Requisição simples e sem cacheamento

```
function fazerRequisicao(url) {  
  /* Realiza a requisição à URL e retorna a resposta */  
}  
  
function getRequest(url) {  
  var resposta = fazerRequisicao(url)  
  return resposta  
}
```

Fonte: O Autor

No exemplo fornecido pelo Algoritmo 1, é possível observar uma abstração simplificada de uma aplicação que realiza uma requisição a uma URL. Essa implementação simples permite que a aplicação tenha comportamentos dinâmicos com base na resposta obtida. No entanto, nota-se que toda vez que a função *getRequest()* for executada, será realizada uma nova requisição à URL. Outra forma de visualizar o que ocorre no código acima é observar o seguinte diagrama:

Figura 3 - Diagrama de sequência da requisição sem cache



Fonte: O Autor

Ao analisar a Figura 3, é possível notar que o diagrama exposto é linear e não apresenta condições de uso, portanto ao realizar uma requisição, a sua resposta é recebida pela aplicação, e posteriormente poderá ser processada pelo sistema. Implementar essa sequência pode gerar custos adicionais, em casos onde há um grande volume de consultas realizadas ou quando uma consulta precisa ser repetida diversas vezes com os mesmos parâmetros anteriores e sem mudanças na resposta recebida.



Para lidar com essa questão, é considerada a implementação de estratégias de otimização, como o cacheamento das respostas, que permite armazenar temporariamente as respostas das requisições e reutilizá-las quando necessário.

Dessa forma, no contexto da Figura 3, seria evitado o envio repetido de solicitações à URL, reduzindo tanto os custos quanto o tempo de resposta. A partir disso, é importante avaliar a viabilidade e a necessidade do cacheamento conforme os requisitos e características específicas do projeto. Portanto, com uma abordagem adequada, é possível equilibrar a dinamicidade da aplicação e os custos associados às requisições, proporcionando uma melhor experiência para os usuários e otimizando o desempenho do sistema.

Dito isto, a ferramenta de persistência desenvolvida nesse projeto foi utilizada para simplificar a implementação do cacheamento e observar seus benefícios gerais no projeto, comparando tempos de execução e tempo percebido pelo usuário com cada implementação. Uma forma de demonstrar a implementação utilizada em ambos os ambientes de maneira simplificada é a que segue:

#### Algoritmo 2 - Implementação de cache numa requisição

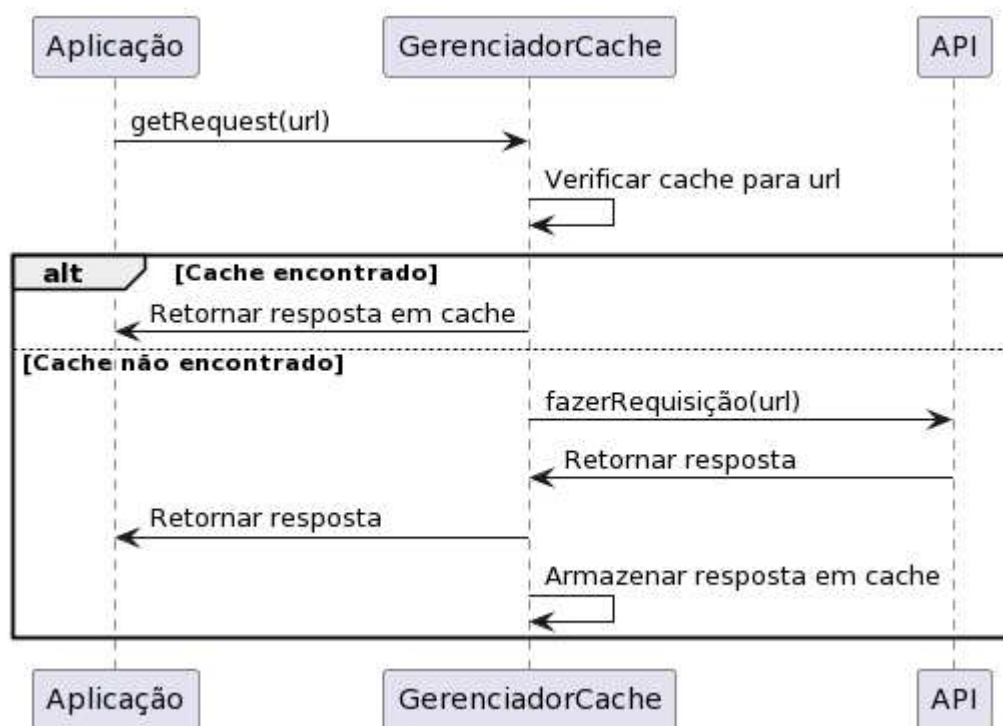
```
function fazerRequisicao(url) {  
  # Realiza a requisição à API e retorna a resposta  
}  
  
function calcularTempoExpiracao() {  
  # Calcula o tempo de expiração da resposta em cache  
}  
  
let cache = novo dicionario vazio;  
  
# valida se a requisição já possui alguma versão salva em cache e se ela ainda  
# possui tempo de validade, retorna do cache caso sim e faz uma nova requisiç  
# ão caso não  
function getRequest(url) {  
  if (cache contém a chave url e cache[url].tempoExpiracao > tempoAtual()) {  
    return cache[url].resposta  
  } else {  
    resposta = fazerRequisicao(url)  
    tempoExpiracao = calcularTempoExpiracao()  
    cache[url] = { resposta: resposta, tempoExpiracao: tempoExpiracao }  
    return resposta  
  }  
}
```

Fonte: O Autor

No pseudocódigo exibido no Algoritmo 2, nota-se na função *getRequest(url)*, responsável por verificar se a resposta para a URL solicitada está presente no cache e se ainda não foi expirada. Caso a resposta seja encontrada e esteja dentro do tempo de expiração definido, ela é retornada imediatamente. Caso contrário, é realizada uma nova requisição à API por meio da função *fazerRequisicao(url)*, obtendo assim a resposta atualizada.

Em seguida, o tempo de expiração é calculado pela função *calcularTempoExpiracao()*, e tanto a resposta quanto o tempo de expiração são armazenados no cache para uso futuro. Aqui está o diagrama correspondente ao fluxo do pseudocódigo:

Figura 4 - Diagrama de sequência do uso de cache para requisições



Fonte: O autor

Na Figura 4, é possível visualizar os benefícios que a implementação da ferramenta de persistência de dados pode trazer para um sistema, pois desempenha um papel fundamental, permitindo a implementação de um algoritmo eficiente e versátil de armazenamento de dados em diferentes ambientes.

Um dos principais benefícios que podem decorrer da implementação do diagrama demonstrado na Figura 3 é a reutilização do código entre os ambientes web em React e mobile em React Native. Isso é possível, pois o fluxo descrito resulta em uma economia significativa de tempo e esforço de desenvolvimento, além de garantir uma maior consistência e uniformidade no tratamento dos dados em ambas as plataformas.

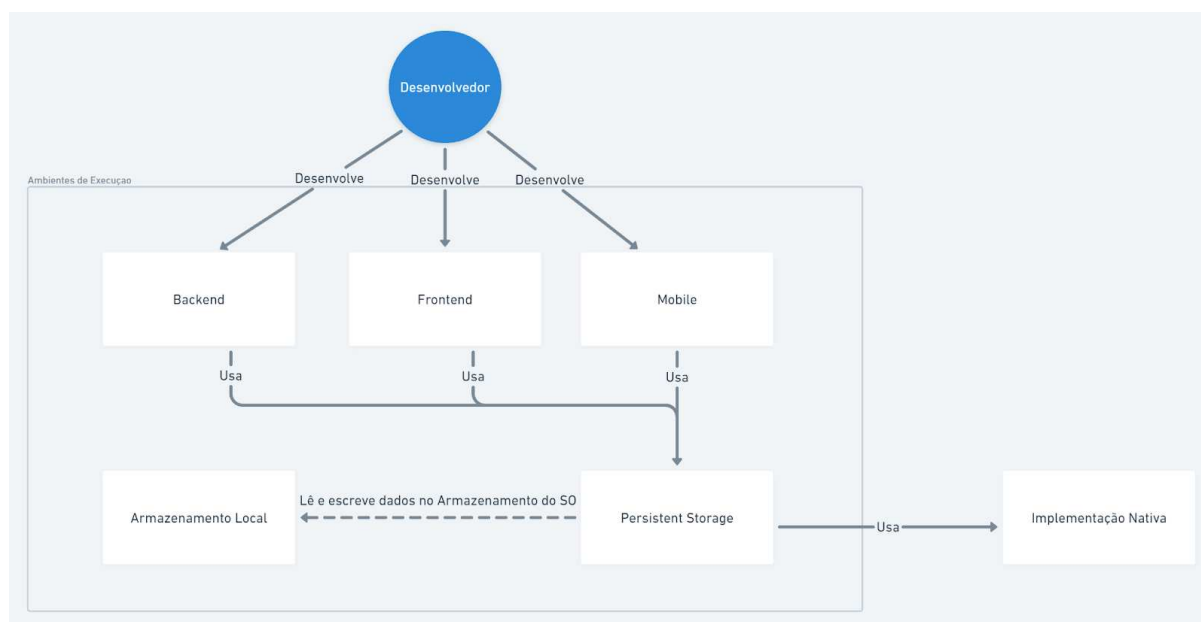
Deste modo, conforme as hipóteses levantadas, e seguindo a implementação, buscou-se explorar os resultados que seriam obtidos através da implementação da ferramenta de controle de tempo de duração das informações armazenadas. A expectativa era que essa funcionalidade proporcionasse uma melhor gestão dos dados, evitando o armazenamento de informações obsoletas e contribuindo para a atualização contínua do cache. Com essa abordagem, esperava-se obter melhorias significativas no desempenho do sistema, reduzindo o tempo de resposta e otimizando o consumo de recursos.

Além disso, a simplificação no gerenciamento dos dados proporcionada pela ferramenta também poderia impactar positivamente a manutenção do projeto, facilitando a identificação e correção de problemas relacionados ao armazenamento e acesso das informações.

O próximo passo do trabalho foi averiguar os resultados obtidos com a implementação da ferramenta em relação aos objetivos estabelecidos. Nesse sentido, foi avaliado o desempenho do sistema e o consumo de recursos que são indicadores relevantes que ajudam a entender a satisfação de um usuário ao utilizar o sistema. Os dados coletados foram analisados estatisticamente, a fim de verificar se a implementação da ferramenta alcançou os resultados esperados.

Para exibir uma visão macro do funcionamento da ferramenta foi construído um diagrama, ilustrado na Figura 5, que encapsula onde cada ação é realizada no desenvolvimento de uma aplicação que implementa a ferramenta proposta.

Figura 5 - Diagrama de Containers do funcionamento do pacote em cada ambiente



Fonte: O Autor

O diagrama na Figura 5, exibe uma visão simplificada do funcionamento esperado do pacote final, exibindo os relacionamentos entre o desenvolvedor e os diferentes ambientes e como todos podem recorrer à ferramenta desenvolvida, que vai ler e escrever os dados a serem persistidos.

Para o escopo deste trabalho considera-se o uso do Javascript como a linguagem aplicada para a realização dos experimentos exibidos na Figura 5. É importante salientar que os códigos produzidos são escritos em typescript, todavia isso não se caracteriza como um desvio do que foi proposto pelo trabalho, uma vez que:

1. Typescript é um conjunto de ferramentas construídas em cima do Javascript para auxiliar no desenvolvimento.
2. Todo o código produzido em Typescript sofre uma tradução/compilação para o Javascript antes de ser executado.

Para a solução do problema levantado, é então defendida a criação de uma ferramenta que, segundo um contrato comum e independente do ambiente em que está sendo executada, consiga persistir informações que lhe forem encaminhadas. Desse modo, o desenvolvedor obtém uma estrutura simples, mas que contempla as funcionalidades essenciais para o armazenamento das informações importantes para sua aplicação ou serviço.

Portanto, o trabalho sustenta-se nas hipóteses levantadas e visa satisfazer os objetivos propostos, para isso é imprescindível que a ferramenta provoque alguma redução de recursos nos cenários destacados, tendo foco em melhorar a experiência do usuário na aplicação. Portanto, durante a validação desse trabalho foi realizada uma comparação entre os resultados obtidos pelo uso da ferramenta contra requisições sem nenhuma estratégia de cache, tal como demonstrado no Algoritmo 1 e na Figura 3.

## 4.2. Materiais

Para a construção da ferramenta e dos ambientes de testes foi utilizada a linguagem Typescript, tendo em vista sua versatilidade no desenvolvimento de aplicações para diversas plataformas. Além disso, foram aplicados padrões de projeto para garantir a integridade (Multiton), segurança (Closure) e o desempenho (Half Sync Half Async) da ferramenta, além de seguir a arquitetura MVC durante o desenvolvimento dos ambientes web e mobile.

Como recursos computacionais, foi utilizado um Xiaomi Poco F4 GT, executando a versão 13 do sistema operacional e MIUI<sup>2</sup> versão 14.0.2 para executar os testes no ambiente mobile e o navegador Google Chrome<sup>3</sup> na versão 113.0.5672.127 (Versão oficial) 64 bits como ambiente para realização dos testes web, executando em um computador equipado com:

- AMD© Ryzen© 9 5900X
- 32Gb de memória RAM DDR4
- Disco Rígido de 1 Tb à 7200 RPM de taxa de rotação
- Placa de rede WIFI 6 CF-AX181 PRO/PLUS
- Windows 11 compilação 25381.1
- SSD de 1 Tb M.2 NVMe de terceira geração

Estando o Sistema Operacional instalado no SSD e o código-fonte do projeto testado no disco rígido. Outras informações relevantes quanto aos materiais utilizados para a realização dos testes da ferramenta são: foi utilizada a versão 18.15.0 do Node<sup>4</sup>, bem como a versão 11 do pacote OpenJDK<sup>5</sup> (build 11.0.11+10). Quanto ao Android Studio, tem-se a versão Android Studio<sup>6</sup> Flamingo | 2022.2.1 Patch 1 Build #AI-222.4459.24.2221.9971841, Sendo que a versão alvo do SDK escolhida para os testes foi a API 33 do Android. Além disso, nenhum dos testes foi realizado utilizando o Emulador Android disponível no Android Studio. Todos foram realizados no dispositivo físico, utilizando a Ponte de Depuração Android (Android Debug Bridge ou ADB) na versão 1.0.41, 32.0.0-9680074 e o SonarQube versão 10.0.0.68432.

---

<sup>2</sup> <https://www.mi.com/global/miui>

<sup>3</sup> <https://www.google.pt/intl/pt-BR/chrome/>

<sup>4</sup> <https://nodejs.org/pt-br>

<sup>5</sup> <https://openjdk.org/install/>

<sup>6</sup> <https://developer.android.com/studio>

## 5. Resultados

### 5.1. Publicação da Ferramenta

Como parte dos resultados obtidos na pesquisa, foi possível publicar o pacote "persistor-node" no repositório npmjs. O npmjs é um repositório online de pacotes de software para a linguagem de programação JavaScript. Este repositório atua como um sistema de gerenciamento de pacotes, permitindo que os desenvolvedores compartilhem, publiquem, instalem e gerenciem bibliotecas, frameworks e ferramentas JavaScript em seus projetos. O npmjs é amplamente reconhecido como o principal repositório utilizado pela comunidade de desenvolvedores JavaScript, e sua popularidade e adoção generalizada o tornam uma fonte confiável e central para a descoberta, distribuição e colaboração em projetos JavaScript.

Isso permite que a solução esteja acessível para outros desenvolvedores. Ao disponibilizar o pacote "persistor-node" no npmjs, a solução ganha visibilidade e se torna facilmente encontrada por outros profissionais que buscam uma ferramenta de persistência de dados multiplataforma. Além disso, ao estar presente nesse repositório, o pacote ganha reputação e credibilidade, o que contribui para sua adoção pela comunidade de desenvolvedores.

A disponibilidade no npmjs também simplifica o processo de instalação e incorporação do pacote em projetos existentes, uma vez que o gerenciamento de dependências é facilitado pelo gerenciador de pacotes do npm. Isso promove a colaboração e o compartilhamento de conhecimento, permitindo que outros desenvolvedores contribuam, forneçam feedback e até mesmo estendam as funcionalidades da solução. Em suma, a publicação do pacote impulsiona a disseminação da solução, tornando-a acessível, popular e pronta para ser adotada em diversos projetos e cenários de desenvolvimento.

### 5.2. Testes Realizados

Neste capítulo são detalhados os resultados obtidos pelos testes da ferramenta de persistência de dados descrita no Capítulo 3 (Metodologia). Inicialmente é apresentado os tipos de testes de desempenho, e posteriormente os resultados da implementação desses testes. Por fim, é realizado o teste de qualidade de código utilizando a ferramenta SonarQube.

Para a validação das hipóteses, foram realizados testes de desempenho baseados no tempo de sistema, no tempo decorrido entre a requisição e o recebimento da resposta do

servidor, e para as requisições realizadas, foi comparado os tempos decorridos nos casos com e sem cache.

Os testes contaram com intervenção humana, e, portanto, a fim de minimizar possíveis divergências nos resultados, foi criado um ambiente automatizado para coletar os tempos das requisições, seguindo uma ordem sistemática de uma sequência de tarefas, que foram:

- Primeiro teste: Carregar a primeira pagina de séries
- Segundo teste: Carregar informações de um seriado específico
- Terceiro teste: Carregar a lista de episódios da primeira temporada do seriado;

É importante salientar que para o segundo teste foi escolhido o seriado “Flash”, que durante os testes realizados, estava presente na primeira página da API utilizada nos testes.

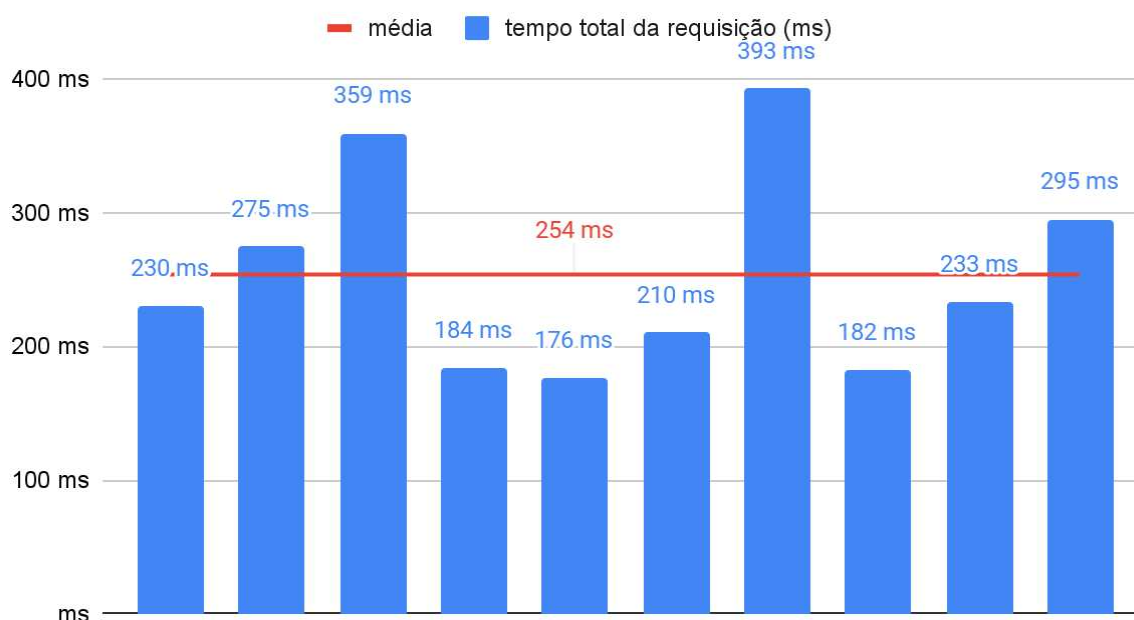
Cada um dos testes foi repetido dez vezes em cada ambiente. A razão para a repetição dos testes é a normalização dos tempos de requisição por meio de ferramenta estatística de média aritmética.

### Primeiro Teste

Para o primeiro teste, foi realizado o acesso inicial ao site e ao aplicativo, aguardando o carregamento da primeira página proveniente da API. Em seguida, o sistema procedeu com a coleta dos tempos de requisição, desde o início até a conclusão. Os resultados desses testes foram apresentados em um gráfico, no qual o eixo vertical representa o tempo decorrido para cada uma das execuções.



Figura 6 - Gráfico dos resultados obtidos no primeiro teste via API no navegador

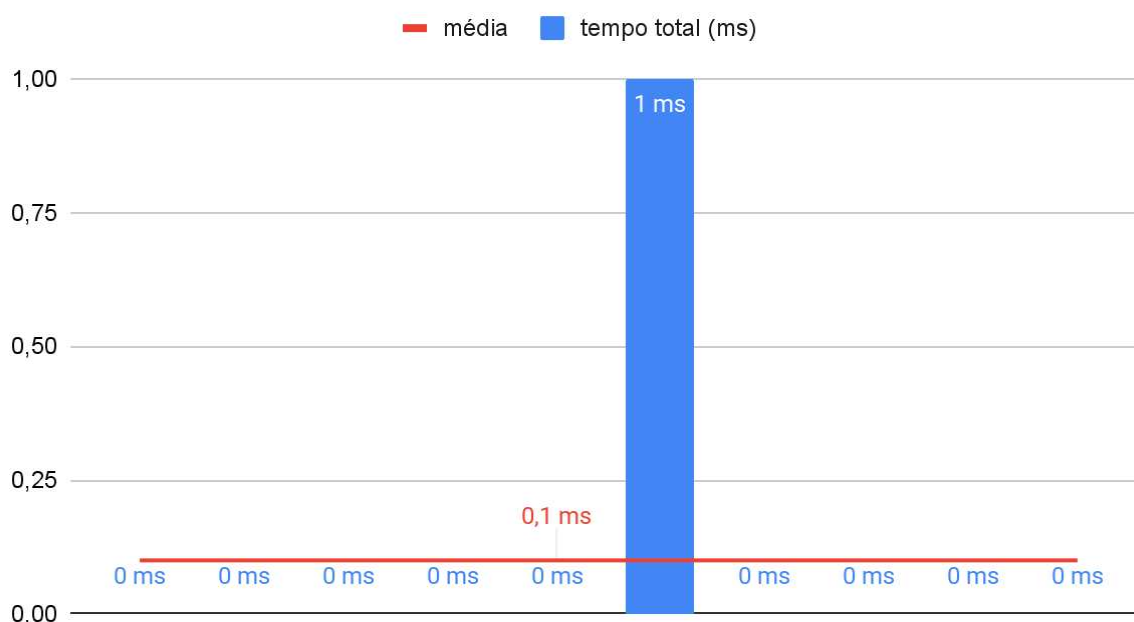


Fonte: O Autor

Os resultados dos testes de tempo de resposta da API, sem o uso de cache, estão apresentados no gráfico mostrado na Figura 6. No gráfico, as barras azuis representam os tempos de resposta de cada execução realizada, enquanto a linha vermelha indica a média aritmética dos tempos de resposta obtidos. Com base nos dados apresentados, observa-se que, em média, as requisições para a primeira página da API são concluídas em aproximadamente 254 milissegundos (ms). Essas informações são relevantes para compreender o desempenho do sistema e avaliar a eficiência das requisições feitas à API.

Os resultados do carregamento da primeira página com os dados em cache são apresentados na Figura 7.

Figura 7 - Gráfico dos resultados obtidos no primeira teste via cache no navegador



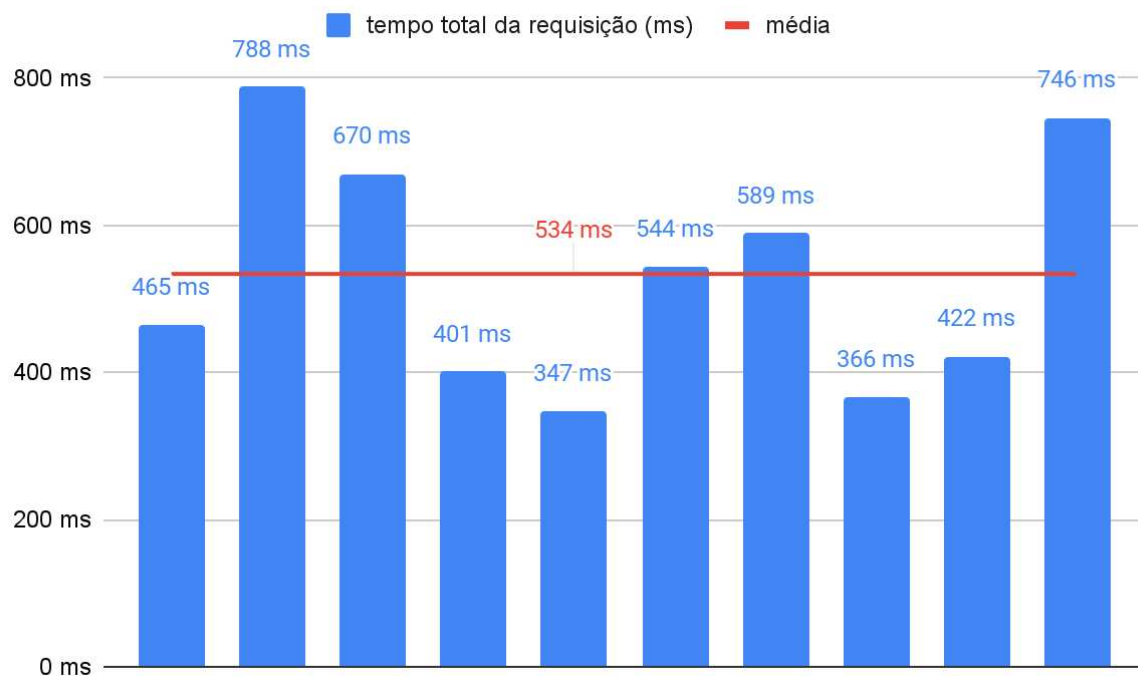
Fonte: O Autor

Para esse teste, é apresentado na Figura 7, que para os dados coletados no navegador, com a utilização de cache, o tempo de resposta foi fortemente reduzido e se tornou praticamente irrelevante.

É importante ressaltar que os dados estão apresentados em milissegundos(ms), mas é possível notar que o intervalo de tempo não é adequado para esse teste, o intervalo apropriado seria o de microssegundos ( $\mu$ s) visto que 1ms equivale a 1000 $\mu$ s, e isso proporciona dados mais precisos. No entanto, devido às limitações do JavaScript, não é possível medir intervalos de tempo inferiores a 1ms. Apesar disso, os dados obtidos ainda evidenciam de forma clara os benefícios da utilização do cache no aprimoramento do desempenho do sistema.

Já a Figura 8 exibe o resultado do mesmo teste, porém para o ambiente mobile. É importante notar que o desempenho nesse contexto é limitado devido à natureza portátil do dispositivo, o que resulta em uma redução do desempenho em prol da duração da bateria. Por esse motivo, otimizações nesse ambiente são valorizadas, uma vez que tornam o aplicativo mais responsivo e performático em um ambiente com recursos limitados.

Figura 8 - Gráfico dos resultados obtidos no primeiro teste via API no mobile

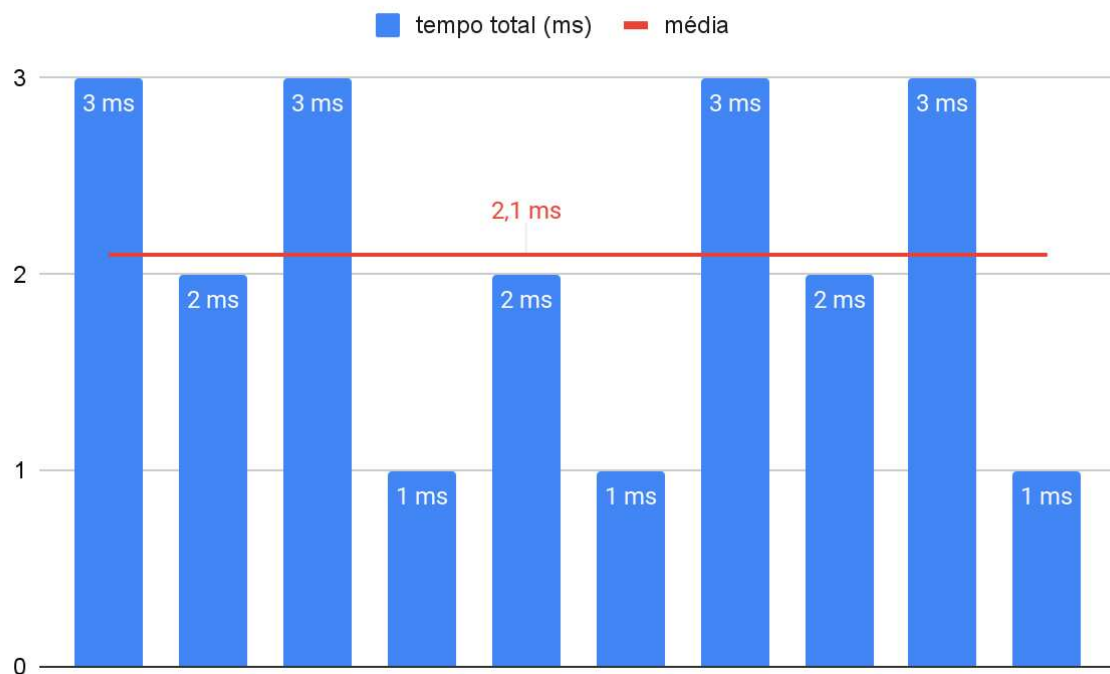


Fonte: O Autor

Na Figura 8, é perceptível que, em primeiro lugar, os valores são mais altos em comparação com os dados coletados no navegador. Algumas possíveis causas para isso incluem o desempenho do ambiente onde os testes foram executados, conforme mencionado anteriormente, e também a velocidade máxima suportada pelo setor responsável pelo acesso à internet no dispositivo. Apesar desses fatores, é notável que a média do tempo das requisições nesse caso é de 534ms.

Além disso, os dados dos testes com cache no ambiente mobile seguem abaixo:

Figura 9 - Gráfico dos resultados obtidos no primeiro teste via cache no mobile



Fonte: O Autor

Através da Figura 9, fica evidente que, mesmo com as limitações do ambiente móvel, o uso do cache teve um impacto significativo no desempenho do carregamento dessa requisição. Isso implica que a navegação pela página inicial do aplicativo será extremamente confortável para o usuário, contanto que dados em cache não expirados estejam disponíveis. Em média, espera-se que as informações sejam exibidas na tela em aproximadamente 2ms após a solicitação.

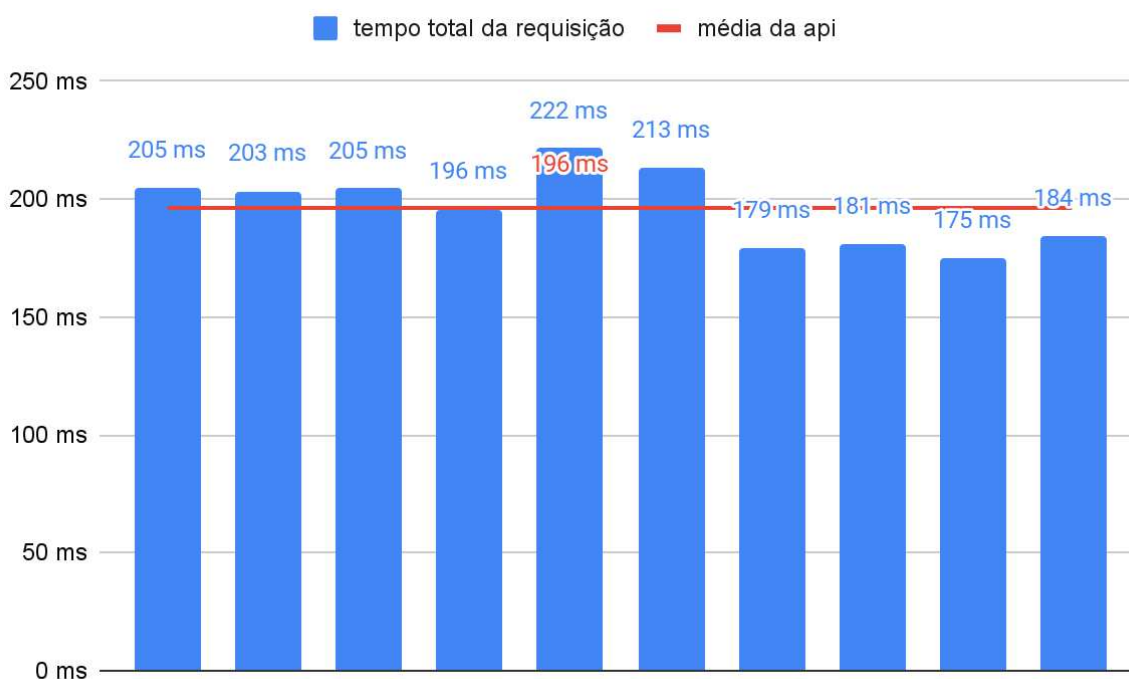
No entanto, é importante ressaltar que esses números representam uma média e podem variar dependendo de vários fatores, como a velocidade da memória do dispositivo. Além disso, é fundamental atualizar adequadamente os dados em cache para evitar que o usuário receba informações desatualizadas.

## Segundo Teste

Para o segundo teste, foi realizado o carregamento de informações de um seriado, a fim de obter o tempo necessário para exibir essas informações. Esse teste pretende compreender como o cache interage com o aplicativo em cenários em que as requisições à API são menos frequentes. Em teoria, nesse contexto o cache fornece menos benefícios perceptíveis aos tempos de requisição para o usuário final.

Entretanto, é importante destacar que, mesmo em cenários com requisições menos frequentes, o cache ainda desempenha um papel relevante. Embora os tempos de carregamento possam não ser reduzidos de forma tão significativa quanto em requisições mais frequentes, o cache ainda pode contribuir para a melhoria do desempenho geral do aplicativo.

Figura 10 - Gráfico dos resultados obtidos no segundo teste via API no navegador



Fonte: O Autor

Os dados apresentados na Figura 10, assim como os da Figura 6, representam a duração das requisições feitas à API. No entanto, uma diferença notável observada nos testes executados é a maior consistência dos dados na Figura 10 em comparação com a Figura 6.

Isso significa que os valores de tempo obtidos na Figura 10 apresentam um intervalo mais próximo entre si, indicando uma maior uniformidade nos tempos de resposta.

Figura 11 - Gráfico dos resultados obtidos no segundo teste via cache no navegador

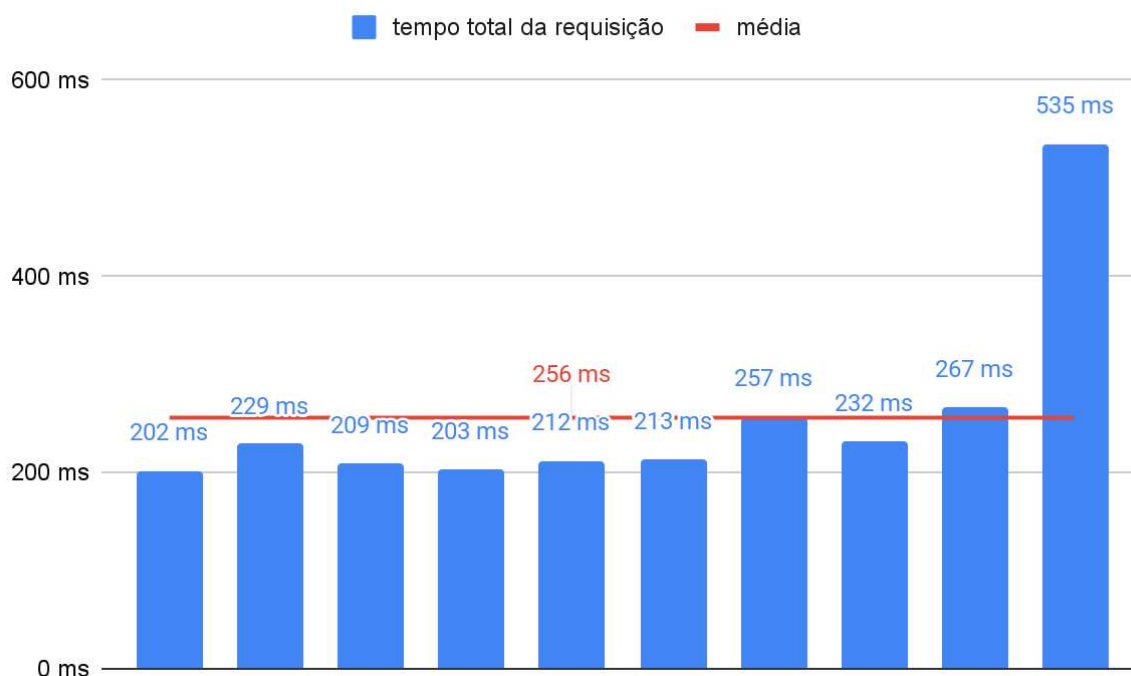


Fonte: O Autor

Conforme apresentado na Figura 11, é destacado a redução nos tempos de requisição via API, o que era esperado e reforça os resultados obtidos na Figura 7. A implementação de cache resulta em uma melhoria significativa nos tempos de carregamento da página no navegador, permitindo que os dados a serem exibidos sejam carregados praticamente instantaneamente. Ainda segundo a Figura 11, há a queda nos tempos de requisição, confirmando os benefícios proporcionados pelo uso do cache.

Para complementar os resultados, os testes realizados no ambiente mobile também revelaram melhorias nos tempos de requisição ao comparar os testes com e sem cache. A fim de ilustrar essas diferenças de maneira mais clara, é apresentado novamente os gráficos com os resultados obtidos através da Figura 12.

Figura 12 - Gráfico dos resultados obtidos no segundo teste via API no mobile



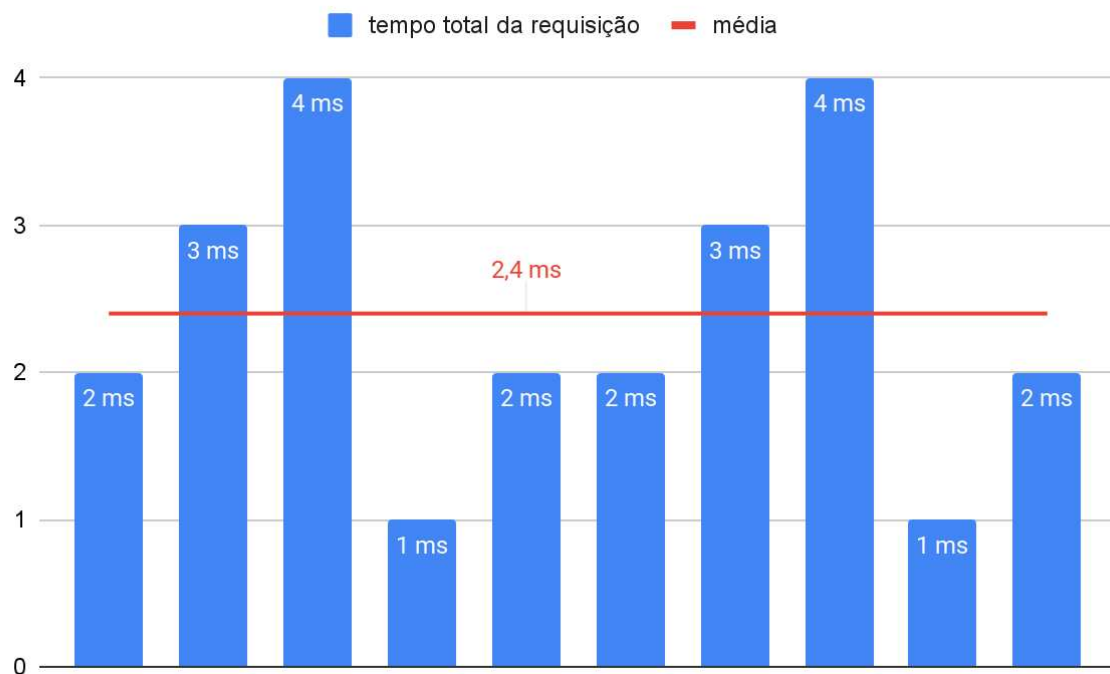
Fonte: O Autor

Na Figura 12, é possível observar novamente os tempos de carregamento da tela de detalhes do seriado. Desta vez, os dados estão mais próximos daqueles obtidos no navegador, com tempos significativamente menores em comparação ao carregamento da primeira página da lista de seriados, cujos valores podem ser observados na Figura 8. Essa informação demonstra os diferentes resultados que uma requisição à mesma API pode obter, dependendo do ambiente de execução. Ambientes com especificações mais modestas tendem a ser mais sensíveis a quaisquer variações que possam existir.

Para uma análise mais detalhada desse fato, é importante observar os dados coletados no ambiente mobile, porém, desta vez, considerando a utilização de cache para obter os dados persistidos. Isso permitirá verificar como o cache afeta o desempenho e a consistência dos tempos de requisição nesse contexto específico.

Portanto, conforme apresentado na Figura 13, os números apresentados são novamente inferiores em relação aos tempos obtidos por meio da API.

Figura 13 - Gráfico dos resultados obtidos no segundo teste via cache no mobile



Fonte: O Autor

Essa redução nos tempos de execução verificada na Figura 13 provavelmente ocorre devido às condições do sinal recebido pelo dispositivo móvel. Os smartphones possuem antenas pequenas e precisam economizar energia para manter a bateria por mais tempo. Como resultado, a qualidade da conexão à internet é geralmente inferior, e o dispositivo se beneficia significativamente da estratégia de persistência local das requisições, por meio do uso de cache.



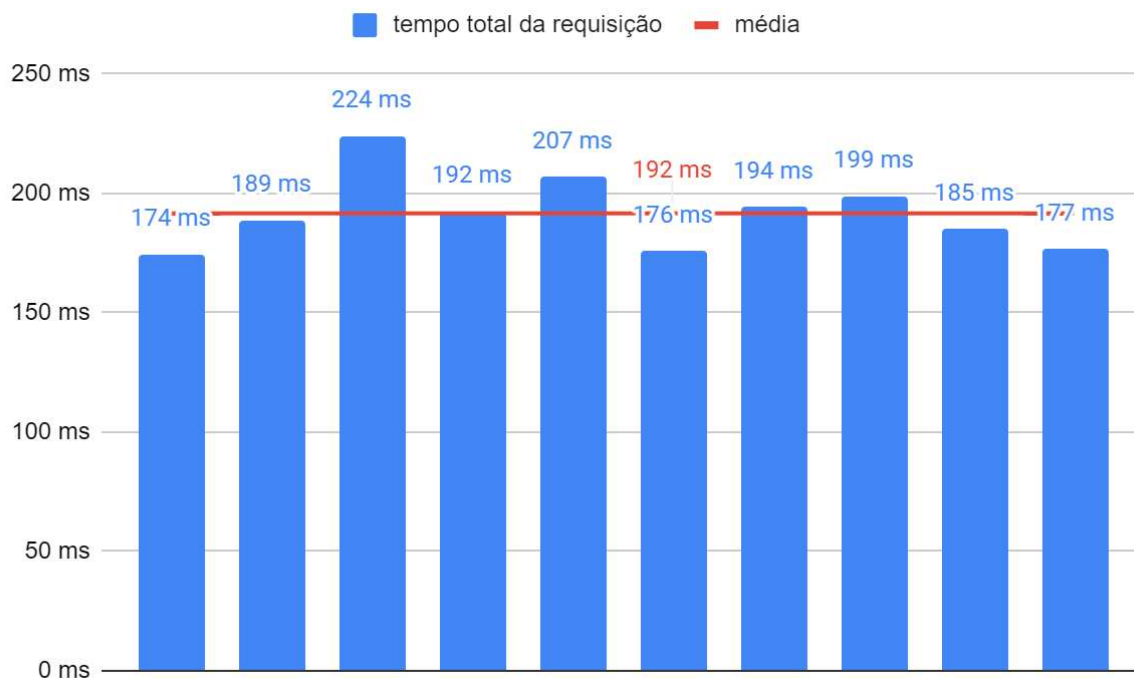
### Terceiro Teste

Uma explicação para o terceiro teste, que pode aparentar reproduzir os mesmos resultados dos dois testes anteriores, é que ele ilustra bem o cenário de requisições que podem se repetir diversas vezes, com poucas opções para o desenvolvedor intervir. Nesse caso, há uma informação específica que precisa ser exibida na tela, e é o usuário quem determina quantas vezes essa informação será solicitada e quando ela será necessária. Essa informação está relacionada aos dados de uma temporada de um programa, por exemplo.

Nesse cenário, o usuário tem o controle sobre quando e quantas vezes essa informação será exibida. Em casos em que essa informação não precisa ser atualizada com tanta frequência, a utilização de cache se torna uma solução interessante para otimizar o desempenho geral do sistema. O cache permite agilizar o acesso aos dados, reduzindo a necessidade de novas requisições à API e melhorando a experiência do usuário.

Portanto, ao considerarmos o navegador, é evidente o impacto do uso do cache no desempenho da tela em questão e como isso afeta diretamente a usabilidade para o usuário. O cache proporciona melhorias significativas, permitindo um carregamento mais rápido e suave dos dados necessários, resultando em uma experiência mais fluida e agradável ao navegar na aplicação.

Figura 14 - Gráfico dos resultados obtidos no terceiro teste via API no navegador



Fonte: O Autor

Os resultados apresentados na Figura 14 reforçam o que foi afirmado anteriormente na Figura 6 e na Figura 10. Os tempos de resposta estão alinhados com o esperado para uma requisição à API, sendo ligeiramente inferiores aos demais testes, mas ainda em um intervalo comum. Isso mostra que os resultados das requisições a diferentes endpoints da mesma API geralmente possuem tempos similares entre si.

Além disso, é possível observar que a implementação do cache nesse cenário específico provoca resultados consistentes no navegador. Para validar essa afirmação, pode-se analisar os resultados mais recentes no navegador, considerando o uso de cache através da Figura 15.

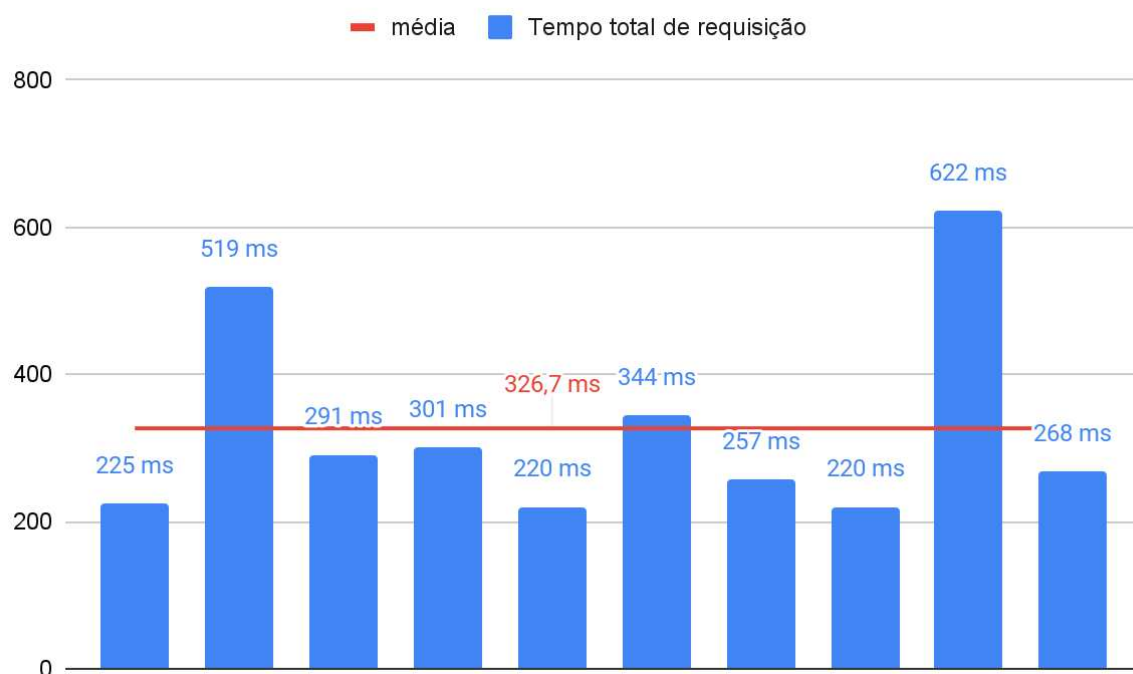
Figura 15 - Gráfico dos resultados obtidos no terceiro teste via cache no navegador



Fonte: O Autor

Conforme apresentado na Figura 15, é possível observar novamente os resultados obtidos nos testes anteriores com cache no navegador. Os resultados mostram uma aproximação dos tempos de carregamento ao imediato, o que sugere que, em todos os cenários testados, o uso de cache é benéfico para o desempenho do sistema no contexto do navegador. Essa consistência nos resultados reforça a importância e eficácia do cache na otimização da experiência do usuário ao tornar o carregamento das informações praticamente instantâneo.

Figura 16 - Gráfico dos resultados obtidos no terceiro teste via API no mobile

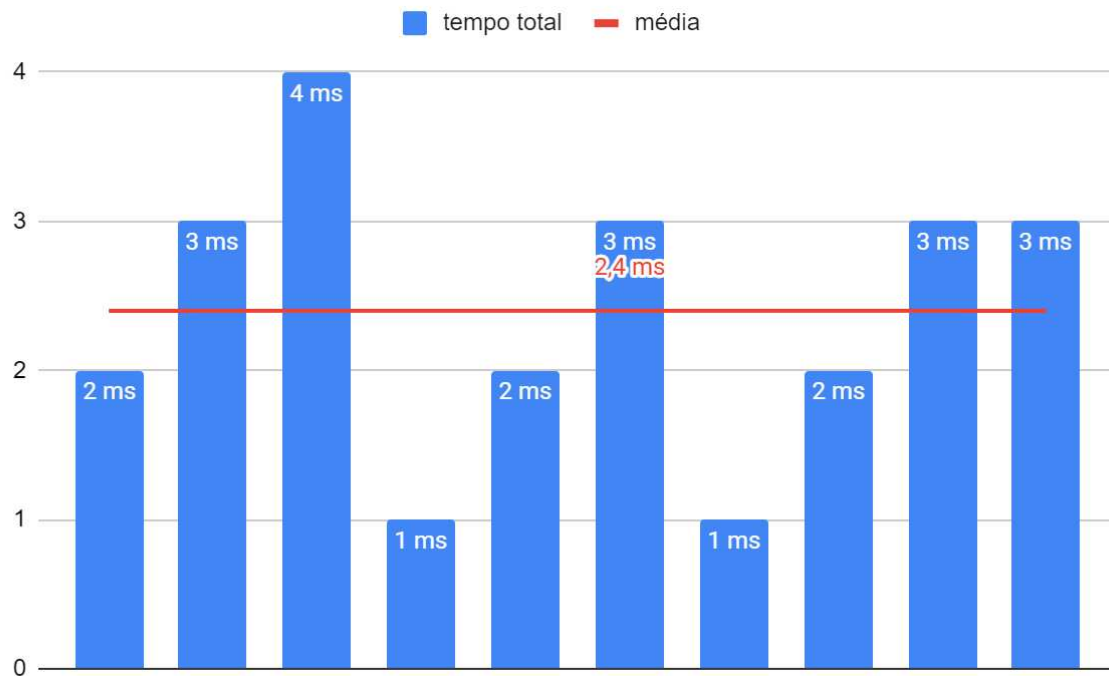


Fonte: O Autor

Os resultados obtidos no ambiente mobile são apresentados na Figura 16, no qual é possível visualizar o comportamento do aplicativo durante a execução do terceiro teste, assim como se verifica que o aplicativo alinha-se aos resultados anteriores ao manter os tempos de requisição próximos. Isso reforça a conclusão mencionada anteriormente, em relação ao navegador, de que os tempos de requisição a diferentes endpoints de uma mesma API, tendem a ser similares, independentemente do ambiente.

Agora, no ambiente mobile, isso indica que os tempos de resposta que o usuário irá observar no sistema, em condições ideais de disponibilidade, serão consistentes em média. Essa consistência nos tempos de requisição contribui para uma experiência mais previsível e confiável para o usuário no aplicativo móvel.

Figura 17 - Gráfico dos resultados obtidos no terceiro teste via cache no mobile



Fonte: O Autor

Por fim, conforme apresentado na Figura 17, os resultados do terceiro teste, executado em dispositivos móveis utilizando cache, revelaram um desempenho significativamente melhor em comparação aos tempos de execução anteriores. Ao analisar o tempo médio obtido, que foi de 2.4ms, fica evidente um comportamento notável.

Concluindo, os resultados demonstram que a utilização de cache tem um impacto significativo na melhoria do tempo de execução das requisições, tanto no ambiente móvel quanto no navegador. Essa constatação reforça a importância do cache como uma estratégia eficaz para otimizar o desempenho do sistema, proporcionando tempos de resposta mais rápidos e uma melhor experiência do usuário em ambos os ambientes.

## Testes de qualidade de código

Adicionalmente, a fim de validar a qualidade da ferramenta trabalhada, foi executado um teste da geral no código da ferramenta utilizando o SonarQube, que é uma ferramenta independente para validar qualidade de projetos. O uso do SonarQube é de extrema importância no desenvolvimento de software, pois desempenha um papel fundamental na garantia da qualidade do código.

Segundo a descrição oficial da ferramenta:

O SonarQube<sup>7</sup> é uma ferramenta de revisão de código automática e auto-gerenciada que ajuda sistematicamente a entregar um código limpo. Como um elemento central de nossa solução Sonar, o SonarQube se integra ao seu fluxo de trabalho existente e detecta problemas em seu código, auxiliando na realização de inspeções contínuas do código de seus projetos. A ferramenta analisa mais de 30 linguagens de programação diferentes e se integra à sua pipeline de CI e plataforma de DevOps para garantir que seu código atenda aos padrões de alta qualidade.

Essa ferramenta oferece uma abordagem sistemática e automatizada para a revisão de código, identificando potenciais problemas e violações de boas práticas de programação. Ao realizar análises estáticas, o SonarQube é capaz de detectar bugs, vulnerabilidades de segurança, duplicações de código, complexidade excessiva e outros padrões indesejáveis. Isso permite que os desenvolvedores identifiquem e corrijam problemas antes mesmo de serem implantados em produção, evitando retrabalho, melhorando a manutenibilidade do código e reduzindo riscos futuros.

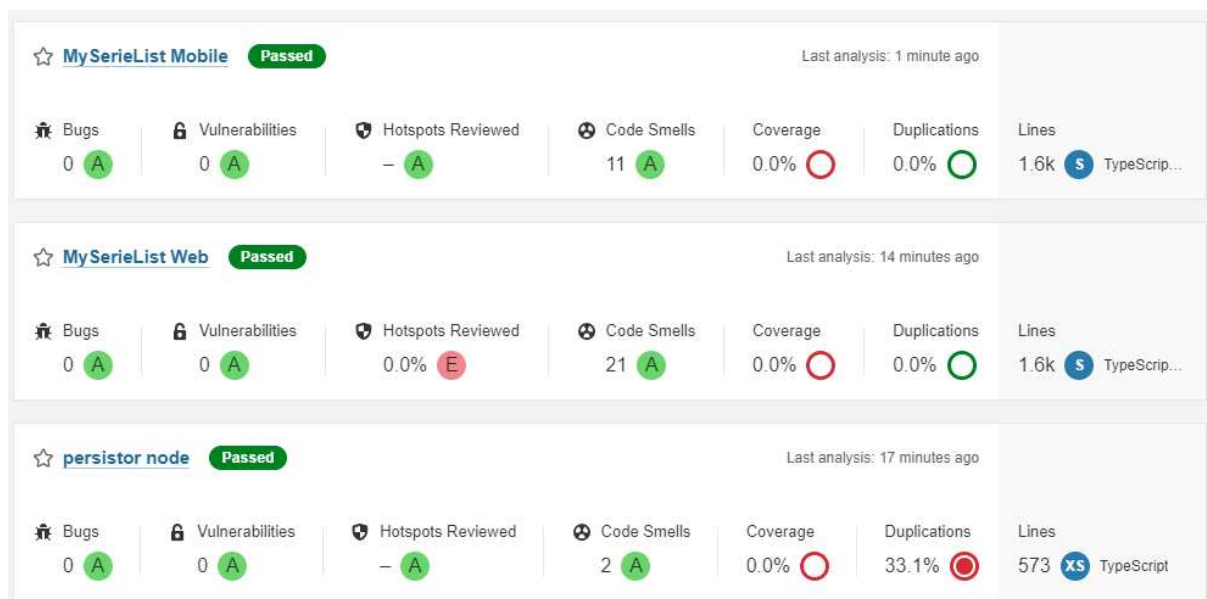
Além disso, o SonarQube oferece suporte a várias linguagens de programação, o que o torna uma ferramenta versátil e amplamente utilizada pela comunidade de desenvolvimento. Ao adotar o SonarQube em um projeto, é possível elevar o nível de qualidade do código, garantir a conformidade com as melhores práticas e promover a construção de um software mais confiável e eficiente.

Abaixo são exibidos os resultados dos testes realizados na ferramenta, e nos dois ambientes em que a mesma foi implementada:

---

<sup>7</sup> <https://docs.sonarqube.org/latest/>

Figura 18 - Resultado dos testes realizados utilizando o SonarQube



Fonte: O Autor

Na Figura 18 é possível notar a ausência de bugs em todos os ambientes testados, além disso, todos os sistemas desenvolvidos passaram nos testes de qualidade que a ferramenta realiza, apesar de ter encontrado alguns pontos que merecem atenção a respeito dos sistemas.

Primeiramente comentando as duplicações de código encontradas na ferramenta, *persistor-node*, que foi utilizada nos outros dois projetos, trata-se de um débito de código que deve ser corrigido e publicado em uma próxima versão da ferramenta, no entanto, como os resultados exibidos no documento foram colhidos utilizando uma versão da ferramenta que possui tais duplicações, foram mantidos para fins didáticos.

Quanto aos resultados no ambiente web, todos os testes realizados receberam notas A, considerada uma nota alta, com exceção dos testes realizados em *Hotspots Reviewed* que faz referência a pontos críticos do sistema em questão, que podem conter problemas de desempenho ou de segurança que afetam a funcionalidade da aplicação, dito isto, o problema encontrado teve origem devido ao uso de chaves atreladas ao índice de uma lista no React, de maneira resumida, o React usa chaves que o desenvolvedor atribui quando monta uma lista de componentes, essas chaves garantem que o React possa fazer manipulações na posição de cada elemento dessa lista sem realizar comparações lentas para a aplicação, no entanto, o uso em questão foi em componentes temporários exibidos na tela enquanto uma requisição a API

é feita, para demonstrar para o usuário que há um carregamento ocorrendo, conhecidas como *Skeletons*, desse modo, não representam nenhum problema sério ao sistema.

Além disso, ainda no ambiente web, existem “*cheiros de código*” destacados pelo sonarqube e referem-se a algumas variáveis e importações não utilizadas, porém, mantidas no sistema, isso pode ser considerada uma má prática quanto a organização do código trabalhado, porém não afeta o desempenho final da aplicação, pois durante a etapa de compilação da aplicação todo esse código é removido automaticamente por não ter nenhuma referência na aplicação e até por isso, não impede o sistema de obter uma nota alta e passar nas verificações da ferramenta, servindo como uma sugestão de melhoria da mesma.

Por fim, a Figura 18 exhibe que o ambiente mobile também passou em todos os testes realizados, porém só houve problemas, assim como no ambiente web, quanto aos “*cheiros de código*” onde, novamente, o sonarqube reclamou de variáveis não utilizadas no sistema. Os pontos levantados pela ferramenta são válidos e devem ser corrigidos em futuras versões dos sistemas trabalhados, porém foram mantidos nesse teste de qualidade, pois todos estavam presentes durante a realização dos testes realizados e exibidos nesse trabalho.



## 6. Conclusões

Considerando essas informações, é possível concluir que o uso de cache é uma estratégia eficiente para melhorar o desempenho e a experiência do usuário tanto em ambientes móveis quanto em navegadores web, proporcionando tempos de resposta rápidos e uma navegação fluida. Mesmo considerando que os resultados devem variar a depender das especificações do usuário, os dados continuamente entregaram melhorias que giram em torno de 50 (cinquenta) a 200 (duzentas) vezes mais rápido, é fácil concluir que a implementação atingiu os objetivos de melhoras a experiência do usuário proporcionando uma experiência mais fluída durante o uso da aplicação.

É importante notar, no entanto, que nem todos os cenários permitem a adoção da estratégia de cacheamento demonstrada nesse trabalho de conclusão, uma vez que existem cenários que exigem entregar ao usuário sempre as informações mais atualizadas para o cliente, e algo que fuja disso pode ocasionar em prejuízos tanto para o usuário quanto para o dono do site ou aplicativo em questão, como é o caso, por exemplo, de aplicativos que monitoram o valor de ações ou de criptomoedas, e, nesses cenários, outras estratégias precisam ser implementadas para atingir o fim de reduzir os tempos ou custos de operações no servidor.

Adicionalmente aos resultados obtidos, a publicação da ferramenta "persistor-node" no npmjs garante pontos positivos significativos. A disponibilidade do pacote nesse repositório proporciona visibilidade à solução desenvolvida, tornando-a facilmente acessível e encontrada por outros desenvolvedores em busca de uma ferramenta de persistência de dados multiplataforma. Além disso, o pacote ganha reputação e credibilidade, fortalecendo sua adoção e utilização na comunidade. A publicação no npmjs também traz benefícios práticos, como a simplificação da instalação e gestão de versões, uma vez que o gerenciador de pacotes do npm facilita o processo de incorporação do pacote em projetos existentes. E além de tudo o supracitado, promove a colaboração da comunidade, permitindo que outros desenvolvedores contribuam, forneçam feedback e ampliem as funcionalidades da ferramenta. Dessa forma, a publicação do pacote possibilita não apenas resultados técnicos positivos, como otimização de desempenho e reutilização de código, mas também benefícios estratégicos e de colaboração, fortalecendo o impacto e a disseminação da solução desenvolvida.

## Referências

AL-ROUSAN, T.; AL-SHARGABI, B.; ABUALESE, H. A New Security Model for Web Browser Local Storage. **International Journal of Advanced Computer Science and Applications**, v. 10, n. 8, 2019.

BELTRÃO, F. **JPDROID: JAVA Persistence For Android**. [s.l: s.n.].

BODUCH, A.; DERKS, R. **React and React Native**. 3. ed. Birmingham, England ; Mumbai: Packt, 2020.

BUNGE, M. Technology as Applied Science. **Contributions to a Philosophy of Technology**, v. 5, p. 19–39, 1966.

CARLOS GIL, A. **Como Elaborar Projetos De Pesquisa**. [s.l: s.n.]. Disponível em: <[https://files.cercomp.ufg.br/weby/up/150/o/Anexo\\_C1\\_como\\_elaborar\\_projeto\\_de\\_pesquisa\\_-\\_antonio\\_carlos\\_gil.pdf](https://files.cercomp.ufg.br/weby/up/150/o/Anexo_C1_como_elaborar_projeto_de_pesquisa_-_antonio_carlos_gil.pdf)>.

CERQUEIRA, F. A. B. **Um Sistema Publicador/Subscritor Com Persistência De Dados Para Redes De Dispositivos Móveis**. [s.l: s.n.].

CHERNY, B. **Programming TypeScript : Making Your JavaScript Applications Scale**. Sebastopol, Calif.: O'reilly Media, Inc. C, 2019.

DIJKSTRA, E. W. **A Discipline of Programming**. [s.l.] Prentice Hall, 1976.

DRISCOLL, J. R. et al. Making Data Structures Persistent. **Journal of Computer and System Sciences**, v. 38, n. 1, p. 86–124, fev. 1989.

GAMMA, E. et al. **Design patterns : elements of reusable object-oriented software**. Boston: Addison-Wesley, 1994.

KAPLAN, H. Persistent Data Structures. **Handbook of Data Structures and Applications**, v. 1, n. 1, p. 31–131–26, 28 out. 2004.

KOKOLIS, A. et al. Distributed Data Persistency. **MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture**, 17 out. 2021.

LANGSARI, K.; ROCHIMAH, S.; AKBAR, R. J. Measuring Performance Efficiency of Application applying Design Patterns and Refactoring Method. **IPTEK Journal of Proceedings Series**, v. 4, n. 1, p. 149, 29 jan. 2018.

LIPSCHUTZ, S. **Data structures**. New Delhi: Mcgraw Hill Education (India) Private Limited, 2014.

MACCHERONE, L. **node-localstorage**. Disponível em: <<https://github.com/lmaccherone/node-localstorage>>. Acesso em: 24 nov. 2022.

NETO, J. D. DE O. et al. **Análise comparativa de desempenho de aplicação Android com persistência em Banco de Dados Relacional e Banco de Dados Orientado a Objetos**. [s.l: s.n.].

PARNAS, D. L. SDI: A Violation of Professional Responsibility. **A Computer Science Reader**, p. 351–363, 1988.

**React Native · A framework for building native apps using React**. Disponível em: <<https://reactnative.dev/>>.

ROQUE, J. I. **Um Estudo De Performance De Uma Ferramenta De Object/Relational Mapping**. [s.l: s.n.].

ROY CHOUDHARY, S. Cross-platform testing and maintenance of web and mobile applications. **Companion Proceedings of the 36th International Conference on Software Engineering**, v. 1, n. 1, 31 maio 2014.

TYPESCRIPT. **Documentation - TypeScript for the New Programmer**. Disponível em: <<https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>>. Acesso em: 30 out. 2022.