Teste de performance em tecnologias de armazenamento de dados no *client-side Web* com abordagem ao *Offline First*.

Halyson Rezende Nery[1], Luiz Alexandre de Sousa Freitas[1], Mateus Augusto Sousa Pereira[1], Ana Maria Martins Carvalho[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia Goiano - Campus Morrinhos (IF Goiano) Caixa Postal 92 – 75650-000 - Morrinhos - GO – Brasil

{halyson97, mateusaugustodes.p}@gmail.com, luiz.alexandre@live.com, ana.carvalho@ifgoiano.edu.br

Abstract. *The present article is focused on test and the comparison of the performance of data storage technologies on the Web client side, in order to evaluate these technologies that follow the Offline First's tendency. The tests were performed to compare the time spent in the operations of insertion, retrieval and deletion of data using Session Storage, Local Storage, Web SQL and PouchDB.*

Resumo. *O presente artigo visa testar e comparar o desempenho de tecnologias de armazenamento de dados no client-side Web, com objetivo de avaliar essas tecnologias que seguem a tendência do Offline First. Foram realizados testes para comparar o tempo gasto nas operações de inserção, recuperação e remoção de dados utilizando Session Storage, Local Storage, Web SQL e PouchDB.*

1. Introdução

Em tempos em que a mobilidade é um requisito e não há garantia de conectividade de rede a todo momento[Alex 2013], espera-se garantir que a aplicação se mantenha operacional mesmo quando *off-line*, sendo essa uma característica chave das aplicações *Web* modernas que necessitam de alta disponibilidade [Offline 2017].

As aplicações *Web* tradicionais são projetadas para um cenário ideal onde se tem conexão estável e rápida à Internet, no qual a instabilidade ou falta de conexão é vista apenas como um problema temporário e sendo tratado como erro [Feyerke 2013]. Contrapondo o modelo tradicional, o movimento denominado *Offline First* defende que a aplicação deve estar pronta para cenários sem acesso à Internet [Sauble 2015].

Para oferecer uma experiência adequada, segundo o movimento *Offline First*, a aplicação deve ser projetada primeiro para o funcionamento *off-line* [Holt 2017], ou seja, armazenar e manter os dados das operações realizadas no *client-side* (parte da aplicação que roda a partir do navegador do usuário), sincronizando estes com o servidor quando *online*, dessa forma não tratando a falta de conexão como um problema [Feyerke 2013].

Considerando os conceitos do *Offline First*, que se torna cada vez mais relevante nos dias atuais [Ferraz 2015], este trabalho visa realizar a comparação de velocidade das principais tecnologias de armazenamento de dados disponíveis no *client-side*, realizando

um teste de *Benchmark* (avaliação de desempenho de um processo) com objetivo de avaliar o tempo da execução das operações de inserção, recuperação e remoção.

2. Tecnologias de armazenamento de dados em navegadores *Web*

Há diversos modelos de armazenamento de dados para utilizar no *client-side*, como chave-valor, relacional, indexado, orientado a documentos, entre outros. Modelos estes que podem ser implementados em memória ou com persistência de dados [Lawson 2015], ser nativos do navegador ou funcionar por meio de *adapters* (camada de abstração que realiza operações utilizando bancos de dados nativos) [PouchDB 2017].

Neste trabalho foram utilizadas as tecnologias de armazenamentos: *Session Storage*, *Local Storage*, *Web SQL* e *PouchDB*, este último utilizando o *IndexedDB* para a persistência de dados [PouchDB 2017].

2.1. *Session Storage*

O *Session Storage* é um banco de dados do tipo chave-valor que faz parte da especificação *Web Storage*, sendo sua principal característica manipular dados em memória disponibilizando-os apenas para a sessão atual. Por sugestão da *W3C* (*World Wide Web Consortium*) sua implementação nos navegadores é de 5MB de armazenamento por origem [Webstorage 2016].

2.2. *Local Storage*

O *Local Storage*, assim como o *Session Storage*, é um banco de dados do tipo chave-valor que faz parte da especificação *Web Storage*, com a diferença que o *Local Storage* realiza persistência de dados. A *W3C* sugere que seja disponibilizado pela implementação do *Local Storage* nos navegadores o espaço de armazenamento de 5MB [Webstorage 2016].

2.3. *Web SQL*

O *Web SQL* é um banco de dados relacional baseado no *SQLite*, que trouxe a implementação real de banco de dados relacionais em navegadores [Kimak 2013]. O *Web SQL* tornar-se-ia uma especificação recomendada da *W3C*, porém o grupo de trabalho de aplicativos *Web* do *W3C* deixou de trabalhar na especificação em novembro de 2010 [WebSQL 2010]. Atualmente entre os navegadores com maior mercado [StatCounter 2017] somente o *Chrome*, *Opera* e *Safari* mantém suporte ao *Web SQL* [Caniuse 2017].

2.4. *PouchDB*

PouchDB é um banco de dados orientado a documentos para utilização no navegador *Web* inspirado no *Apache CouchDB*. O *PouchDB* utiliza tecnologias de armazenamento nativas do navegador para a persistência de dados por meio de *adapters* [PouchDB 2017]. Uma de suas principais vantagens é a sincronização com o *CouchDB* e servidores compatíveis, tendo como desvantagem a característica de não ser nativo dos navegadores *Web*, necessitando assim a sua importação ao projeto [PouchDB 2017].

3. Metodologia

3.1. Programa Teste

Para a realização do experimento foi criado um arquivo do tipo *JSON* (*JavaScript Object Notation*) denominado *dados.json*, contendo 28000 registros com tamanho total de 2.5MB. Cada registro possui um *id*, *nome*, *sobrenome* e *e-mail*, simulando dados reais de pessoas.

Para cada tipo de armazenamento definido no experimento foi criado um *script* na linguagem *JavaScript* que realiza a leitura do arquivo *dados.json* através de uma requisição *AJAX* (*Asynchronous Javascript and XML*), adicionando o retorno da requisição ao escopo da página, para assim realizar as operações estabelecidas no teste. A Figura 1 mostra a codificação base utilizada para criação dos *scripts*.

```
1  (function(){
2      var xhttp = new XMLHttpRequest();
3      xhttp.onreadystatechange = function() {
4          if (this.readyState == 4 && this.status == 200) {
5
6              window.dadosTeste = JSON.parse(this.responseText);
7
8              //Operações de testes no banco;
9
10         }
11     };
12     xhttp.open("GET", "./dados.json");
13     xhttp.send();
14 })();
15
```

Figura 1: Modelo base de *script* utilizado na criação dos testes

O cálculo de tempo das operações foi realizado utilizando como base a *API* (*Application Programming Interface*) *performance.now* que retorna o tempo em milissegundos, decorrido a partir do carregamento da página, com precisão de 5 milésimos de um segundo [MDN 2017]. No início de cada operação foi obtido o tempo inicial e ao término o tempo final. Vale ressaltar que, para as operações realizadas nas tecnologias de armazenamento *Web SQL* e o *PouchDB,* o experimento leva em consideração que suas operações são executadas de forma assíncrona, assim obtém-se o tempo final na execução do *call-back* (função passada como argumento para outra função). A diferença entre o valor final e o inicial é o tempo gasto para realizar a operação, sendo este o resultado da execução do teste. O resultado ignora o tempo de carregamento do arquivo, considerando somente o tempo de execução da operação.

3.2. Ambiente

Os testes foram executados em um computador com processador de 4 núcleos modelo *Intel Core i5-5200U 2.20Ghz, 8GB* de memória *RAM DDR3* de *1600MHz* e disco rígido de *1TB* de *5400 RPM.* O sistema operacional utilizado foi o *Ubuntu Gnome 16.10 64-bit* utilizando o *kernel 4.8.0-46*. Os navegadores adotados foram o *Google Chrome* na

versão *57.0 64-bit* e *Mozilla Firefox* na versão *52.0 64-bit*. O servidor utilizado para execução da aplicação de teste foi o *http-server* da versão *6.10.2* do *Node.js*.

3.3. Experimento

O experimento consiste em executar um teste de *Benchmark* nas operações de inserção, recuperação e remoção de dados no *client-side*, com objetivo de avaliar o desempenho das principais formas de armazenamento de dados disponíveis nos navegadores *Web* atuais. Cada operação foi executada manipulando o arquivo *dados.json*, contendo 28000 registros. Para a execução dos testes foram utilizados os dois principais navegadores da *Web*, *Google Chrome* e *Mozilla Firefox*, que em março de 2017 obtiveram 77.78% do mercado mundial de navegadores *desktop* e 91.74% do mercado brasileiro de navegadores *Web desktop* [StatCounter 2017].

Todos os tratamentos realizados neste experimento, foram replicados 31 vezes com objetivo de reduzir a possibilidade de erro experimental, sendo que foi descartado o primeiro resultado obtido para evitar ruídos em decorrência da inicialização do navegador.

4. Resultados

4.1. Resultados obtidos no *Google Chrome*

Os resultados dos testes de armazenamento obtidos no *Google Chrome*, são representados na Tabela 1 e a média dos resultados é mostrada na Figura 2, onde é apresentado o resultado de tempo da execução das operações. Os valores inferiores a 1 foram arredondados para 1 milissegundo.

Tabela 1: Compilação dos resultados em milissegundos obtidos no *Google Chrome*

	Session Storage			*Local Storage*			*Web SQL*			*PouchDB*		
	Ins	Rec	Rem	Ins	Rec	Rem	Ins	Rec	Rem	Ins	Rec	Rem
Média	28,7	6,2	1,0	31,1	5,6	1,0	899,5	533,7	15,4	270,2	559,6	171,9
Mediana	29,5	7,0	1,0	32,0	5,0	1,0	928,0	440,0	5,5	256,0	554,0	172,5
DP	4,3	1,0	1,0	7,0	1,0	1,0	116,2	128,0	19,1	43,0	72,9	49,2
Mínimo	23,0	5,0	1,0	21,0	5,0	1,0	725,0	395,0	3,0	245,0	440,0	120,0
Máximo	35,0	7,0	1,0	48,0	7,0	1,0	1256,0	737,0	70,0	439,0	846,0	389,0

Ins = Inserção; Rec = Recuperação; Rem = Remoção; DP = Desvio Padrão

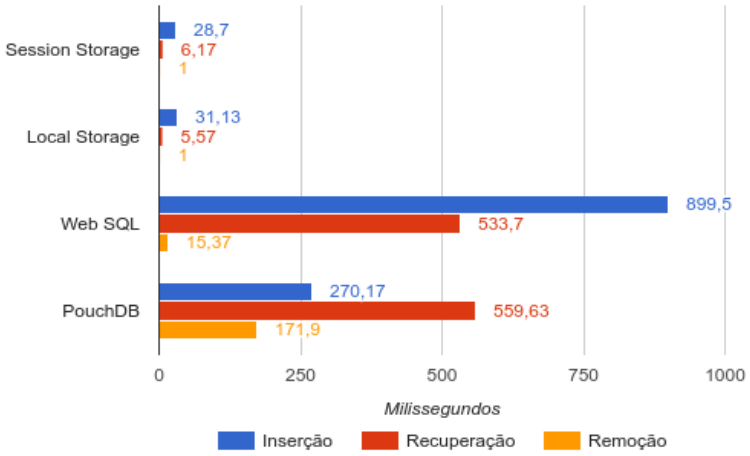

Figura 2: Médias obtidas nos resultados dos testes no *Google Chrome*

Analisando à média dos resultados obtidos entre as operações na Figura 2, o *Session Storage* e o *Local Storage* obtiveram resultados próximos, que não diferem estatisticamente [Barbetta 2010], considerando que a precisão do *performance.now* é de 5 milissegundos [MDN 2017].

Sendo avaliados somente os bancos que persistem dados, o *Local Storage,* no quesito inserção, foi aproximadamente 9 e 29 vezes mais rápido do que o *PouchDB* e o *Web SQL,* respectivamente. Na recuperação de dados, o *Local Storage* foi cerca de 100 e 96 vezes mais rápido do que o *PouchDB* e o *WebSQL,* nessa ordem. No quesito remoção, o *Local Storage* foi 171 vezes mais rápido do que o *PouchDB* e 15 vezes mais do que o *Web SQL*.

Comparando o *PouchDB* com o *Web SQL* pode se observar que o *PouchDB* na operação de inserção de dados foi cerca de 3 vezes mais rápido, todavia, inferior nas demais operações, sendo que na recuperação de dados obtiveram resultados próximos e na exclusão ele é aproximadamente 11 vezes mais lento do que o *Web SQL*.

4.2. Resultados obtidos no *Mozilla Firefox*

Os resultados obtidos nas execuções dos experimentos utilizando *Mozilla Firefox* são apresentados na Tabela 2 e as médias são apresentadas na Figura 3. Vale ressaltar que o *Mozilla Firefox* não oferece suporte ao *Web SQL*, por esse motivo ele não é apresentado nos testes.

Tabela 2: Compilação dos resultados em milissegundos obtidos no *Mozilla Firefox*

	Session Storage			*Local Storage*			*PouchDB*		
	Ins	Rec	Rem	Ins	Rec	Rem	Ins	Rec	Rem
Média	38,4	1,0	1,0	37,9	1,0	1,0	342,2	222,7	168,5
Mediana	37,0	1,0	1,0	36,5	1,0	1,0	342,5	211,0	167,5
DP	6,1	1,0	1,0	5,7	1,0	1,0	37,3	31,6	36,7
Mínimo	35,0	1,0	1,0	35,0	1,0	1,0	293,0	193,0	136,0
Máximo	66,0	1,0	1,0	64,0	1,0	1,0	428,0	317,0	292,0

Ins = Inserção; Rec = Recuperação; Rem = Remoção; DP = Desvio Padrão

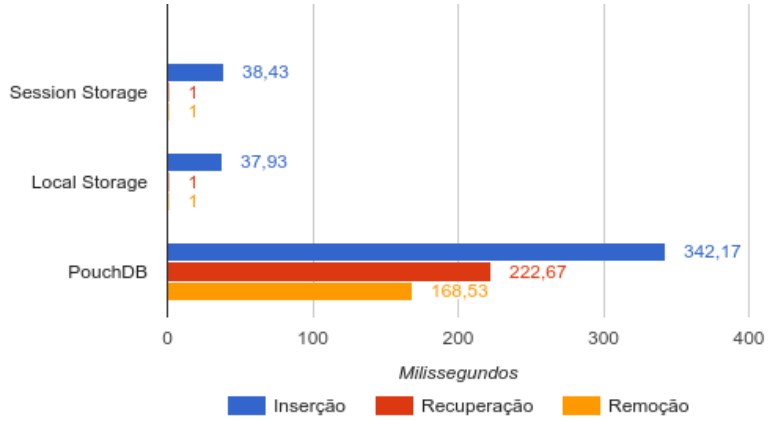

Figura 3: Média obtidas nos resultados dos testes no *Mozilla Firefox*

Comparando os resultados das médias obtidas, o *Session Storage* e o *Local Storage*, assim como no *Google Chrome*, não se diferem estatisticamente. Na comparação dos bancos que fazem persistência de dados, pode observar que o *Local Storage* em relação ao *PouchDB* foi aproximadamente 9 vezes mais rápido na inserção, enquanto a recuperação e remoção foi 222 e 168 vezes mais rápido respectivamente.

4. Conclusão

Este trabalho teve como objetivo o teste de performance das principais tecnologias utilizadas para armazenamentos no *client-side*. Por meio dos experimentos foi verificado que a melhor performance nos testes foi obtida pelo *Local Storage* e o *Session Storage*, obtendo resultados próximos em ambos os navegadores. Vale ressaltar que o *Session Storage* é somente um banco de dados em memória.

Conclui-se que entre as opções apresentadas para o desenvolvimento de aplicações web modernas que seguem os conceitos de *Offline First*, o *Local Storage* possui uma melhor performance em relação aos seus concorrentes, sendo indicado para criação de aplicações que armazenem menos de 5MB (cinco megabytes) de dados [Webstorage 2016]. Em aplicações que necessitam de um armazenamento superior, o PouchDB torna-se a melhor opção levando vantagem sobre o *WebSQL* por oferecer suporte a ambos navegadores testados.

Os resultados obtidos são referentes ao tempo de resposta das operações, porém vale salientar que esse não é o único requisito a ser analisado ao criar uma aplicação *Web* moderna. Antes da escolha da tecnologia de armazenamento a ser utilizada deve-se analisar cada caso de uso e suas peculiaridades.

Como trabalhos futuros será testado e comparado outras tecnologias de armazenamento de dados, abrangendo, além de tecnologias *cliente-side Web*, tecnologias de armazenamento nativas de dispositivos móveis.

Referências

Alex, H. (2013) “Say Hello to Offiline First”, http://hood.ie/blog/say-hello-to-offline-first.html, Acesso em: janeiro de 2017.

Barbetta, P. A., Bornia, A. C. e Reis, M.M. (2010), Estatística para Cursos de Engenharia e Informática, 3º edição.

Caniuse (2017) “Can i use Web SQL Database?”, http://caniuse.com/sql-storage. Acesso em: fevereiro de 2017.

Ferraz, R. (2015) “A importância de desenvolver considerando *off-line* first”, http://tableless.com.br/a-importancia-do-offline-first, Acesso em: janeiro de 2017.

Feyerke, A. (1995) “Designing Offline-First Web Apps”, https://alistapart.com/article/offline-first, Acesso em: fevereiro de 2017 .

Holt. B. (2017) “Offline-first web and mobile apps: Top frameworks and components”, http://techbeacon.com/offline-first-web-mobile-apps-top-frameworks-components, Acesso em: janeiro de 2017.

Kimak, S., Ellman, J. (2013) “Performance Testing and Comparison of Client Side Databases Versus Server Side”, Consecutive Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking & Broadcasting, PGNET 2013.

Lawson, N. (2015) “IndexedDB, WebSQL, LocalStorage – what blocks the DOM?”, https://nolanlawson.com/2015/09/29/indexeddb-websql-localstorage-what-blocks-the-dom. Acesso em: fevereiro de 2017.

MDN (2017) “Performance”, https://developer.mozilla.org/en-US/docs/Web/API/Performance. Acesso em: fevereiro de 2017.

Sauble, D. (2015), Offline First Web Development, Packt Publishing, 1th edition.

OfflineFirst. (1995) “Case Studies and Offline First Success Stories”, http://offlinefirst.org/casestudies/, Acesso em: março de 2017.

PouchDB (2017) “API Reference”, https://pouchdb.com/api.html. Acesso em: fevereiro de 2017.

StatCounter (2017) “Desktop Browser Market Share Worldwide”, http://gs.statcounter.com/browser-market-share/desktop. Acesso em: março de 2017.

WebSQL (2010) “Web SQL Database”, http://www.w3.org/TR/webdatabase/. Acesso em: fevereiro de 2017.

Webstorage (2016) “Web Storage (Second Edition)”, http://www.w3.org/TR/webstorage/ . Acesso em: fevereiro de 2017.