Full-stack FHIR-based MBaaS with Server- and Client-side Caching Capable WebDAO

Zoltán Richárd Jánki, Vilmos Bilicki

Abstract: In healthcare systems, it is essential to have applications that are robust responsive and have a good performance. It is also advisable to store data in some standardized way so it can be integrated with other systems. However, in the 21st century an application may be doubtful of use if the user-experience is at a low level. Several studies inform us about how tolerant the users are when they visit a website or wait to retrieve some data. Based on these studies, we will construct a system that is capable of working offline and can also unburden the server-side. This will be achieved by establishing a so-called Web Data Access Object (WebDAO), which has a maintainable offline capability and also performs better in most given circumstances. Our measurements were evaluated in the context of how users tolerate a delay and slow responses.

Keywords: full-stack, caching, FHIR, WebDAO

Introduction

In the healthcare sector, more and more information about patients has to be processed quickly and efficiently. As a consequence, the healthcare records must be handled in a digitized format. These electronic healthcare records (EHR) must be available, discoverable and understandable. For these requirements a standardized and structured storage is essential. The Health Level Seven (HL7) organization [6] solved these challenges by creating the so-called Fast Healthcare Interoperability Resources (FHIR) standards. With its well-defined resources, this standard is suitable for modelling any areas of healthcare field.

Using the FHIR in Web applications, it is not obvious which implementation would be the best in terms of availability and performance. Full-stack development is one of the most popular approaches today. In a full-stack environment the code base can be shared between the client and the server because they are written in the same programming language. There are open-source implementations of FHIR, but none of them supports client-side object modelling in JavaScript. Hence, we had to create our own Mobile Backend as a Service (MBaaS) following the Representational State Transfer (REST) paradigm [2]. This paradigm asserts that neither the server, nor the client knows anything about the other's state. This stateless system allows both participants to handle the messages received from the other one without seeing any messages that arrived earlier.

To improve the performance, we decided to pursue the idea of polyglot persistence where we use different database systems, but each of them is used for what they are best at. In order to achieve the highest availability we have to exploit the offline capabilities of the client-side and the server-side and create offline-first applications. The framework called Hibernate applies the Data Access Object (DAO) Pattern to separate low-level operations from high-level business services. It uses its own caching model where the queries are forwarded to the different levels of the cache first instead of communicating with the backend. The Hibernate Query Language (HQL) is first forwarded to the Second-Level (L2) Cache and it searches for the queried objects in it. If the required objects are not in the L2 Cache, the request is forwarded to the database. By doing this, the waiting time for an application can be significantly reduced [3][4].

There are several studies available concerning the correlation betweeen users' patience and speed of the Web. As Daniel An from Google said in his article, mobile pages can display content to users within 3 seconds on average and the average time to first byte is under 1.3 seconds [5].

Xiaming Chen and et al. showed that if the time between the emission time of the request and the arrival of the response is more than 4 seconds, the interruption ratio rises rapidly from 18%. If a waiting time of 10 seconds is reached, the ratio of interruption is already at 40% [6]. We use these thresholds to provide a solution that is quite effective.

Below, we are going to review the different offline-first solutions in a full-stack environment and then we would like to present our solution. We will describe the novel features of our system, present the results of our measurements, then draw some conclusions about our prototype systems.

State of the art

FHIR and full-stack development

Over time, FHIR has become widely acknowledged. Several organizations use this standard and some of them are so big that they are present in many countries. For instance, the InterSystems HealthShare [7] is a healthcare informatics platform for hospitals, integrated delivery networks and regional and national health information exchange. It is now present in more than 20 countries. Another well-known project is the so-called Substitutable Medical Applications and Reusable Technologies (SMART) [9]. Since it uses the FHIR standard, the platform was renamed to SMART on FHIR. These systems are widely spread, but none of them uses a full-stack environment.

However, today JavaScript is the only programming language that is supported by every major browser. As we review the history of JavaScript, we see that over time it underwent numerous changes and we can say that JavaScript is a language that is suitable for any general-purpose computing task. [9].

Offline Capabilities

When we talk about Web applications, everybody thinks of online applications but what if there is simply no Internet connection or it is Lie-Fi. Such situations can also occur in mobile applications that require a network connection. In all of the applications, the developers have to take into account network issues. In healthcare systems, availability and performance are two key factors. Can we have an observation that is interrupted because of a bad network connection or is it necessary to query all the records of a patient again after a refresh? The answer is of course no. This is why the offline capabilities of Web applications are very important for us. The basic methodology of caching in Web applications follows the idea of Hypertext Transfer Protocol (HTTP) Caching, but using a framework, this layer is totally hidden. Nowadays, on the client-side the recommended solutions are the use of Service Workers and IndexedDB. These tools guarantee convenient solutions for using the cache storage of a browser.

HTTP Cache

Every major browser carries an implementation of a HTTP Cache. This methodology is based on a proxy-server - also called a Web cache - that meets HTTP requests on the behalf of an origin Web server. The cache storage has its own disk storage that includes the copies of recently requested objects. The HTTP headers are extended with a Cache-Control field and an ETag field [10][11]. These fields are responsible for defining the caching policy and validating the response of the server. Service Workers and IndexedDB use the notion of a HTTP Cache and are involved in developing progressive Web applications.

Service Workers and IndexedDB

A Service Worker is a Web Worker object that is a JavaScript file running in a worker thread. Hence, it is separated from the browser's main thread and it can be executed asynchronously. Before Service Workers, the recommended storing mechanism was the Application Cache – also known as AppCache. AppCache provides a high-level API and every browser has support for it. Since it was not sufficiently flexible, AppCache was replaced by Service Worker, with its low-level Cache API. It is more adaptable because it hands over the "moving parts" to the developers and the configurations are left to the developers [12].

IndexedDB also provides a low-level API for caching data in a NoSQL storage system. It creates a sterling database in the browser and supports transactions as well. The objects can be stored and searched by key-value pairs. It works along similar lines to the local storage of the applications, but IndexedDB works asynchronously. It runs in the background and the background synchronization is also supported [13].

Our solution

A telemedicine application must be able to handle thousands of patients and hundreds of doctors with their data simultaneously. It should be responsive and have a high performance if the architecture is well defined and robust. Nowadays, full-stack development can achieve these requirements. On the client-side we use Angular 2+ frameworks and the backend consists of a LoopBack Server and several databases that realize the polyglot persistence. The Hadoop HDFS is used for storing huge files like videos and images, and Apache Cassandra is responsible for the patients' big data.

On the client-side it seems clear that Service Workers and IndexedDB have to be used. The offline capabilities of LoopBack Server are quite limited and there are no best practices for caching in LoopBack Framework, so we made some extensions to the basic architecture.

Firstly, we created a WebDAO that follows the classic DAO analogue. In our WebDAO layer we defined the models of our resources and the basic methods of the models. With the help of this layer, we were able to write queries on both the client-side and the server-side. Hence, the Cache Storage is not just a plain cache. The programmers can maintain the offline activities and also the response time may be better in the given circumstances. The main advantage of WebDAO is that we do not have to create our own data structure for storage, but we can use the HTTP Cache on the client-side.

IndexedDB can be actuated by another JavaScript database called Apache PouchDB. PouchDB is an open-source database that can synchronize with CouchDB and compatible servers. PouchDB also has a browser version that can be readily integrated into our Angular clients. It provides the toolset for creating an IndexedDB instance and also for modifying and synchronizing its content. With this technology we do not have to manipulate IndexedDB directly. Hence, our idea is to store the static objects - like images and HTML parts - in the Cache Storage by using Service Workers, and store the dynamic parts - like records from the Cassandra database - in an IndexedDB instance. The contents of PouchDB will be synchronized with the CouchDB instance that is also synchronized with LoopBack. These frameworks can improve the availability and the performance as well because if our application goes offline, the data is still available from the cache and using an offline-first strategy, we will not turn to the server for each requests, except if the data is unavailable in the cache. With the help of PouchDB, we can also use filters in our cached dataset, so a new request with filters will not take another request to the server. Another good thing about PouchDB is that it can communicate with an IndexedDB instance in the development mode, while native Service Workers and IndexedDB require a production mode for caching the responses.

Measurements and evaluation

In our system we created a so-called Web Data Access Object (WebDAO) which defines the model of FHIR resources, and these models have their own standardized REST endpoints. By following the REST paradigm, our WebDAO became general and in order to unburden the backend, we entrusted the caching to the browser. Since the REST endpoints are more general, the response is also broader, hence more specific queries can be handled on the client-side by retrieving objects from the Cache Storage. The exact time values can be calculated by the browser, since it can differentiate a query result received from cache or from a server.

Since the whole infrastructure takes place in our local network, the server response time can be much higher in production where we can have different network facilities and overloads. Our dataset contains 1 patient and 1,000,000 different observations belonging to this patient, but the Apache Cassandra retrieves at most 5,000 records in a query because of its driver setup. The first query that we executed was very simple. We queried all of the observations that belonged to our single user. The time that elapsed between the emission of the request and the arrival of the response can be seen in Table 1. It is observed that pulling 5,000 records through the network takes more than 1 second and we can say that it is worthwhile storing the objects in the cache and using an offline-first strategy. If we use more than one filter, the Cassandra database is still fast and the set of objects is much smaller, so a response can be obtained in 300 ms. PouchDB with IndexedDB seems to be weaker here because PouchDB is optimized for synching first. The queries can be speeded up by using indices. If we do not use any indices, the filtering costs more than 5 seconds. This is too much, so using indices is recommended. With indices the request takes about 500 ms. We also tried out filtering with our own indices by creating references from the proper keys in an array. The result was very similar and the difference was about plus or minus 10

ms.

Table 1: COMPARISON OF RESPONSES RECEIVED FROM SERVER AND CACHE

Request	*Query*	*Source of response*	*Number of retrieved records*	*Average time to get response*
List Patient1's observations	user: Patient1, date1: null, date2: null	LoopBack and Cassandra	5,000	1159 ms
		IndexedDB using PouchDB (no-filtering)		521.75 ms
List Patient1's observations that occured between date1 and date2	user: Patient1, date1: 2005-09-07T01:00:00.000Z, date2: 2006-09-07T01:00:00.000Z	LoopBack and Cassandra	8	291 ms
		IndexedDB using PouchDB filtering (without indices)		5446 ms
		IndexedDB using PouchDB filtering (with indices)		519.25 ms

Comparing these results with those that were presented in the above-mentioned studies about the general patience of users, we see that, our system still works within the thresholds and the applications integrated into the system can work offline as well.

Conclusions

Here, we presented a system that uses an offline-first strategy in order to achieve a higher performance and enhance availability. We created a WebDAO that generalizes the client-server communication and makes the tool available to the programmer to maintain the Cache Storage. With polyglot persistence our backend is now more powerful and the whole system works with a similar efficiency in the online state as in the offline state. In the future, we intend to improve the performance of the Cache Storage by using other indexing schemes like indices on patient data and other observation properties.

Acknowledgement

This study was supported by the EU-funded Hungarian grant EFOP-3.6.1-16-2016-00008.

References

[1] *FHIR Overview*, Available: https://www.hl7.org/fhir/overview.html, Accessed: 21 March 2018

[2] B. Mulloy, *Web API Design: Crafting Interfaces that Developers Love*, Available: https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf, Accessed: 28 March 2018

[3] J. P. Ottinger, D. Minter and J. Linwood *Beginning Hibernate*, 3rd Edition, 2014, Apress, United States

[4] J. P. Ottinger, S. Guruzu and G. Mak *Hibernate Recipes: A Problem-Solution Approach*, 2nd Edition, 2015, Apress, United States

[5] D. An, *Find out how you stack up to new industry benchmarks for mobile page speed*, Available: https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks, Accessed: 28 March 2018

[6] X. Chen and et al., *Passive profiling of mobile engaging behaviours via user-end application performance assessment*, Pervasive and Mobile Computing, vol. 29, 2016, pp. 95–112

[7] *InterSystems Health Informatics Platform*, Available: www.intersystems.com/products/healthshare/intersystems-health-informatics-platform, Accessed: 21 March 2018

[8] J. C. Mandel and et al. *SMART on FHIR: a standards-based, interoperable apps platform for electronic health records*, Journal of the American Medical Informatics Association, vol. 23, no. 5, 1 September 2016, pp. 899–908

[9] A. Bretz and C. J. Ihrig, *Full Stack JavaScript Development with MEAN*, 2014, SitePoint Pty. Ltd., 48 Cambridge Street Collingwood VIC Australia 3066

[10] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 7th Edition, 2017, Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, England

[11] I. Grigorik, *HTTP Caching*, Available: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching, Accessed: 22 March 2018

[12] *HTML Living Standard*, Available: https://html.spec.whatwg.org/multipage/offline.html, Accessed: 26 March 2018

[13] P. Walton, *Best Practices for Using IndexedDB*, Available: https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/indexeddb-best-practices, Accessed: 26 March 2018