

Department of Computer Science

University of Kaiserslautern

Master Thesis

Offline caching in web applications for AntidoteDB

Server Khalilov

red17electro@gmail.com

University of Kaiserslautern
Department of Computer Science
Software Engineering

Leader:
Prof. Dr. Arnd Poetzsch-Heffter

Supervisor:
Dr. rer. nat. Annette Bieniusa



Zusammenfassung

Zusammenfassung auf deutsch

Abstract

The purpose of this thesis is to explore the possibilities of developing an offline web application, which would serve as a client for the AntidoteDB database. We developed a prototype of the application and designed the architecture of the application in such a way that both offline and online functionalities are possible.

To model the data stored offline in the local database of a web-browser, Conflict-free Replicated Data Types (CRDTs). It lets to ease the task of merging the changes.

Apart from that, the client-server protocol was designed, in order to support the functionality of the application. The paper could be divided into two parts: firstly, the problem of designing mentioned solution is going to be discussed. Secondly, there is an implementation part and a description of how specific problems were tackled.

Acknowledgement

I would like to take this opportunity and express my biggest gratitude to everyone, who supported me through all the ups and downs I had during my time at the University of Kaiserslautern.

Firstly, I am very grateful to Prof. Dr. Arnd Poetzsch-Heffter and my thesis supervisor Dr. rer. nat. Annette Bieniusa for making it possible for me to work on the topic that suits my area of interest. Moreover, I want to thank Dr. rer. nat. Annette Bieniusa for having the door to her office always open for a discussion, whenever I needed it. I could not ask for more.

Next, I would like to thank the staff at Software Technology Group, who made me feel very welcome and patiently answered any questions I had. Especially, Mathias Weber, Peter Zeller, and Deepthi Akkoorath.

Finally, I want to mention my family and, particularly, my parents. They did their very best to encourage and motivate me throughout the whole period of my study here. Nothing of it would ever be possible without them. Thank you.

Ich versichere hiermit, dass ich die vorliegende Masterarbeit mit dem Thema „Offline caching in web applications for AntidoteDB“ selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Die Stellen, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich durch die Angabe der Quelle kenntlich gemacht.

Kaiserslautern, den 15. December 2018

Server Khalilov

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Motivation | 1 |
| 1.2 | Research questions | 1 |
| 1.3 | The structure | 2 |
| 2 | Background | 3 |
| 2.1 | Theoretical background | 3 |
| 2.1.1 | Operation-based replication approach | 3 |
| 2.1.2 | CRDTs | 3 |
| 2.2 | Related Work | 3 |
| 3 | Requirements | 5 |
| 3.1 | Problem Description | 5 |
| 4 | Design | 7 |
| 4.1 | Overview of the protocol | 7 |
| 4.1.1 | The way of exchanging the data | 7 |
| 4.1.2 | A client receives an update from the server | 8 |
| 4.1.3 | A client sends an update to the server | 8 |
| 4.1.4 | Two clients interact with a server. | 9 |
| 4.2 | System's main components | 9 |
| 4.2.1 | Web Application | 10 |
| 4.2.2 | Server | 11 |
| 4.2.3 | A database layer | 11 |
| 4.3 | Offline functionality | 11 |
| 4.4 | Online functionality | 12 |
| 4.5 | The transition between offline and online modes | 12 |
| 5 | Technologies | 15 |
| 5.1 | Service Workers | 15 |
| 5.2 | IndexedDB database | 16 |
| 5.3 | Cache API | 17 |
| 6 | Implementation | 19 |

Contents

| | | |
|----------|------------------------|-----------|
| 7 | Evaluation | 21 |
| 8 | Conclusion | 23 |
| 8.1 | Summary | 23 |
| 8.2 | Future Work | 23 |
| | List of Figures | 25 |
| | List of Tables | 27 |
| | List of Code | 29 |
| | Bibliography | 31 |

1 Introduction

In this chapter we are going to discuss the motivation, research questions and the scope of this thesis.

1.1 Motivation

AntidoteDB is a planet scale, highly available, transactional database. It accepts parallel updates that are geo-distributed near users. As well as that, it is available under partition, efficient and responsive. At the same time, AntidoteDB avoids anomalies, thanks to the following features: it merges concurrent operations according to their semantics, it groups related operations into atomic transactions, delivers updates in causal order. The main motivation of this thesis is to explore the possibilities of implementing a web-client for the AntidoteDB with a support of caching and by utilizing the main features of the database. As AntidoteDB is already known for its use in the development of applications, it would dramatically improve the user-experience if offline work with the database is going to be reached. I am going to explore the best ways to develop such an application, design it and implement. Here and from now on, we will call such an application by the name of WebCache.

The motivation of this thesis is to show you the functionalities of AntidoteDB. Therefore, it is not to demonstrate some brand new-idea in terms of application. But it is to show that it is possible to build an application based on Antidote-DB that works offline and online.

1.2 Research questions

The following research questions are going to be addressed in this thesis:

- **RQ1.** How efficient is it to use a web-client with cache rather than without it?

1 Introduction

- **RQ2.** What are the methods available to implement web-applications that would be able to work off-line and in the conditions of poor network connections?
- **RQ3.** What could be a scalable solution for transmitting CRDT data between a server and clients?

1.3 The structure

Here, I am going to briefly sketch the structure of this thesis and briefly explain the contents of each chapter.

- Description of the main requirements of the application;
- Design of the architecture
- Description of the implementation phase
- Evaluation
- Conclusion

2 Background

2.1 Theoretical background

For this thesis, one of the core parts in the architecture belongs to the database called AntidoteDB[1]. It is geo-distributed, which means that the datacenters of AntidoteDB could be spread across anywhere in the world. Moreover, due to its features, it is efficient and effective. AntidoteDB groups related operations into atomic transactions, delivers updates in a causal order and knows how to merge concurrent operations. The last is possible because of CRDT-datatype, which is used in Antidote. The main concept of CRDTs is to merge data without additional interaction of the programmer, which makes it possible to avoid the problems that are common for other databases.

2.1.1 Operation-based replication approach

Replication is a fundamental concept of distributed systems, well studied by the distributed algorithms community[2]. In this thesis, we are going to use the operation-based replication approach, which originally means that replicas converge by propagating operations to every other replica[3]. Once an operation is received in a replica, it is applied locally.

2.1.2 CRDTs

In this thesis, we are going to use

2.2 Related Work

This chapter is needed, if the thesis is somewhat research-oriented. (This should be usually the case.) In some cases though, the thesis may also be more application- or implementation-oriented, in which case a designated related work chapter may not be necessary, but instead citations are just used appropriately throughout the thesis, but specifically in the

2 Background

chapters Introduction and Background. Literature search could be based on DBLP.

Maybe you can mention SwiftCloud here.

<https://github.com/SyncFree/SwiftCloud>

3 Requirements

In this chapter, we will describe the requirements for the desired web-application.

3.1 Problem Description

The application should showcase the functionality of communicating with an AntidoteDB database while online. As well as that, it should be reliable, which means it should load instantly and work even in uncertain network conditions. The application should be able to respond quickly to user interactions, be highly available (possible to work with it even when there is no internet at all).

4 Design

As we specified the requirements, we can go further into the design part of the system, and that is what we are going to explain in this chapter. First of all, we will define the protocol of the system. Moreover, secondly, we will describe the client- and server- sides of it.

4.1 Overview of the protocol

The fundamental part of the WebCache will be its protocol design. We need to examine the following cases, in order to specify the protocol of the application:

- A client receives an update from the server
- A client sends an update to the server
- Two clients interact with a server

4.1.1 The way of exchanging the data

Because the AntidoteDB is using CRDT datatypes, the following options are possible to update the database: state-based and operation-based. Due to the time constraints and the amount of work, this thesis will consider only the operation-based approach, which was described in the theoretical background chapter. Therefore, whenever a client wants to update the database, it will send to the server a list of operations. However, whenever it wants to read the value, it will receive the current state of the object from the database.

For this thesis, we are going to use such datatypes as counters and sets. As was already mentioned in the Introduction chapter, the counter is a number datatype, which allows incrementing and decrementing the value. Reading a counter will return the aggregated value of all performed operations.

4.1.2 A client receives an update from the server

Let us say that a client requests an update from the server. In this case, if the request is successful, the server is going to respond with a value for the requested key and the timestamp of the last write – t_0 .

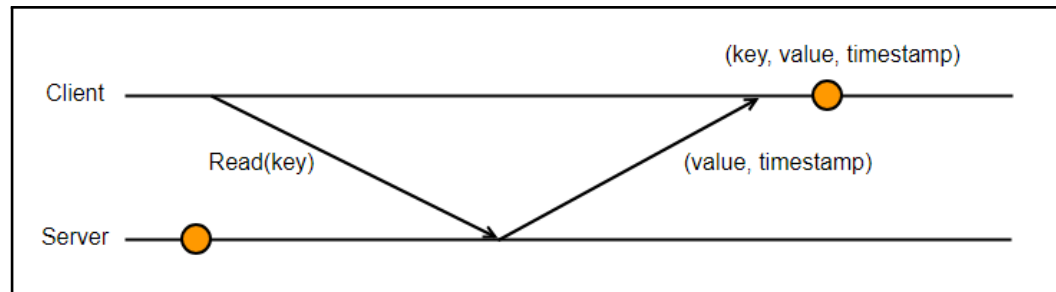


Figure 4.1: The way a client receives a key-value update from the server.

In case a client receives an update from the server, and that update has an earlier timestamp than the one, which is already stored in the client's cache, then a client skips it and can try later for fresher updates. The implementation of this will be explained later in the chapter of implementation.

4.1.3 A client sends an update to the server

In the case of writing the information to the server, a client has to send a key with an operation to the server. Then the acknowledgment with a timestamp is going to be sent back to the client.

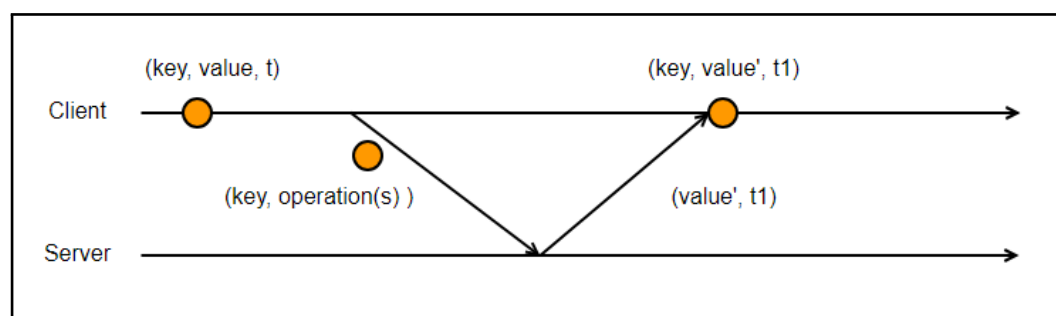


Figure 4.2: The way a client sends a write operation to the server.

Several updates on a client while offline

The client is capable of performing updates when offline. These updates will take effect immediately. However, in order for them to be applied to the server, the client has to be back online. Once the connection is established again, all the updates that were performed on the client-side in offline mode will be sent to the server.

4.1.4 Two clients interact with a server.

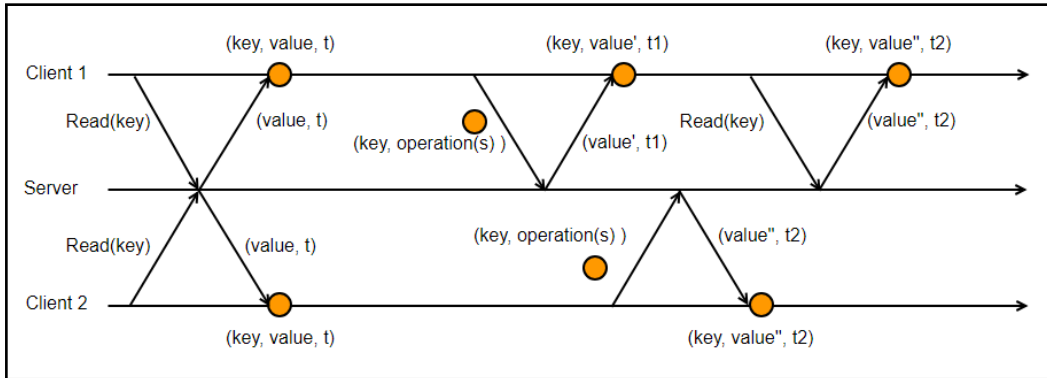


Figure 4.3: Graphical representation over a possible communication between two clients and a server.

Let us assume that initially, the server has the key-value pair (k, v) at the timestamp t_0 . Therefore, after both clients receive updates from the server, they consist of the (k, v) pair at the timestamp t_0 . In case clients change the value under mentioned key to something else, they will have to get an acknowledgement from the server in order to receive a unique timestamp related to the change. At the representation above, a *Client 1* is acting first and getting an acknowledgement of its change at the time t_1 , while *Client 2* makes the change later at time t_2 . That makes *Client 1* receive a new value v'' , when it reads the information from the server again.

4.2 System's main components

As the protocol is explained, we can start designing the system. Let us have a look at the main components of it:

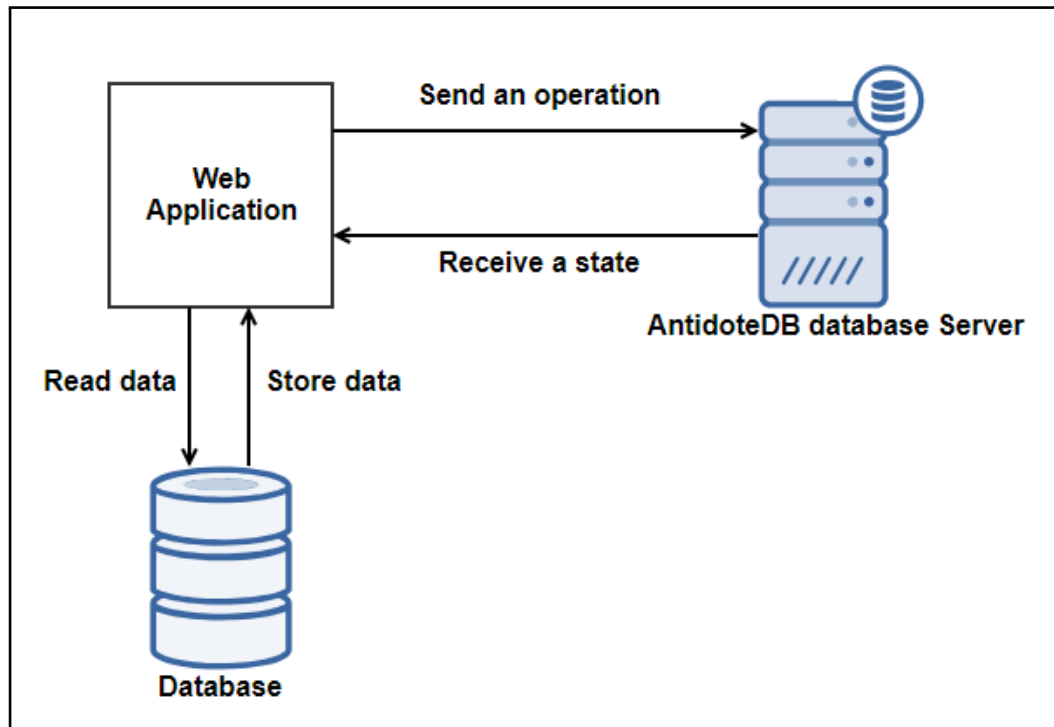


Figure 4.4: An overview of the system's design.

4.2.1 Web Application

First of all, in order for the user to be able to communicate with an AntidoteDB server, we are going to have a running web application that serves as a client. It runs in the web-browser, supports various commands from the user and sits on top of the local database layer.

These are the supported commands:

- *read(key)* – an asynchronous function that pulls database changes concerning the *key* that the user passed.
- *update(key, op)* – an asynchronous function that processes user-made updates.
 - *key*: a key, which is going to be updated;
 - *op*: an operation, which is going to be performed on the key;

4.2.2 Server

It is a configured AntidoteDB server that supports the following scenarios:

- receiving an operations or an array of operations performed on a CRDT-object, according to the key, and applying them on the server;
- sending back to the client the state of requested CRDT-object / objects according to their state on the server;

4.2.3 A database layer

This layer should consist of the database, which is going to store the actual states of CRDT-objects, as well as user-added operations performed on the states. When a user performs *read* by *key* operation from the cache, the following actions are taking place:

- Firstly, the state of the object O is going to be found by *key* in the database
- Next, operations o performed on the object O are selected
- Afterwards, selected operations o applied on the object O .
- Finally, the object from the previous step is returned back as a response to the application.

4.3 Offline functionality

In this section, we are going to describe the offline functionality of the system.

Initially, the database is empty. Therefore, if the user is offline from the very beginning, he should be able to add the data into the database himself. The system represented in **Figure 4.4** will change by having only the Web Application and the database. However, whenever the connection is established with the server, the operations, which were stored in local database while offline, will be sent to the server.

At the **Figure 4.5**, the sequence of getting the data from the local database is shown. This case describes the scenario, when the server is unavailable and the application continues its work in an offline mode.

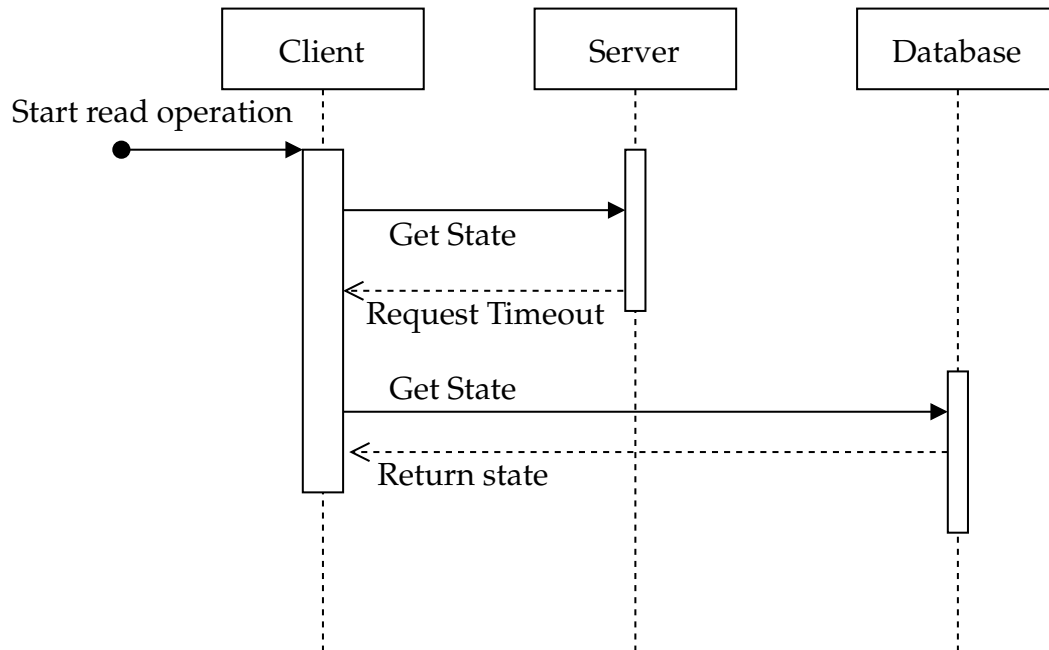


Figure 4.5: Successful request of state from the local database – Sequence diagram.

At the **Figure 4.6**, the sequence of storing the data locally is shown. This case describes the scenario, when the server is unavailable and the application continues its work in an offline mode.

4.4 Online functionality

In this section, we are going to describe the online functionality of the system.

4.5 The transition between offline and online modes

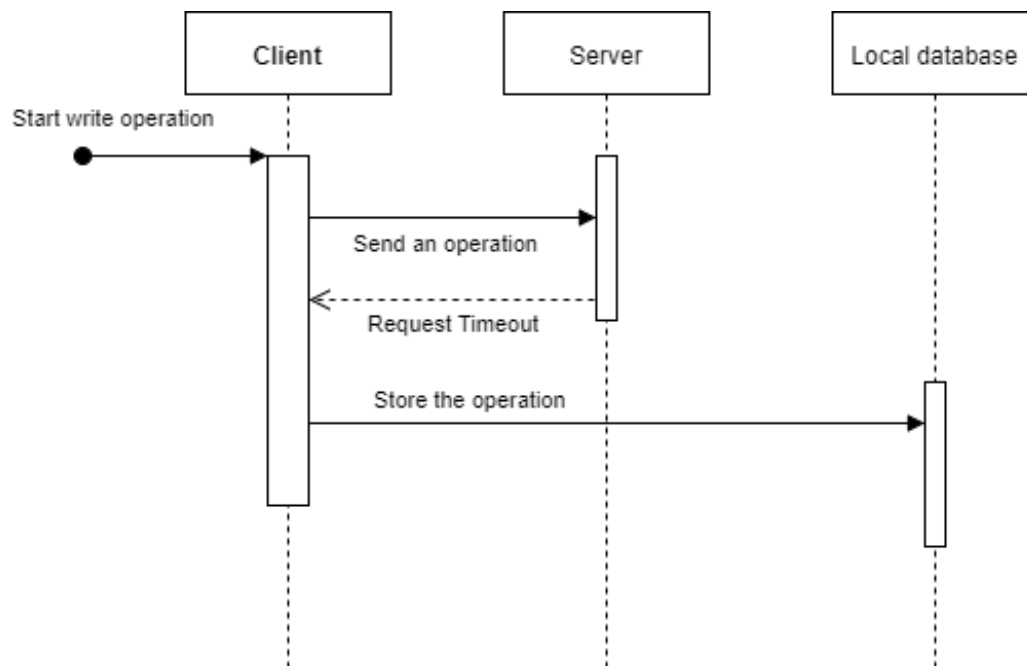


Figure 4.6: Successful storing of an operation in the local database – Sequence diagram.

5 Technologies

This chapter consists of detailed description of used technologies to accomplish thesis goal. I believe that the implementation of the web-client for the AntidoteDB with a cache support would require the following components: ‘

- A service worker, in order to control requests coming from the client.
- A Cache API for storing assets and make the application available offline.
- A database to store the data locally.

5.1 Service Workers

A service worker[4] is a JavaScript file that sits between the application and network requests. As well as that, it runs separately from the page. It has the following properties:

- It is not visible to the user.
- It cannot access the DOM, but it does control pages.
- It can intercept requests as the browser makes them.
 - it can just let requests go to the network as usual.
 - it can skip the network and redirect the request to a cache.
 - or it is possible to perform any combination of these things.

A service worker will be used in order to intercept the network traffic. In situations, when there are problems with network, we are going to get content from cache (which consists of the last content, which we were able to get from the network). However, we will have to wait for the network to fail, before showing the cache-content. And here the problem arises: if the connection is slow, the user would still get frustrating hard-time of

waiting for the response. Therefore, we will have to introduce offline-first approach: it means that we will try to get as much information from the cache as possible.

We might still go to the network, but we are not going to wait for it by updating the information simultaneously from the cache. Afterwards, if we get some new content from the network, we can update the data again. Once we get this new information from the network, we can update the view and, as well as that, we can store that information into the cache for the next time. If we can't get the data from the network, then we will stick with what we've got. Taking such approach makes the user satisfied offline, online and even on slow-connections. It will make the user care less about the connectivity.

5.2 IndexedDB database

When the user opens an application, we want to show the latest data the device received. Then we make the web-socket connection (a web-socket bypasses both the service worker and the HTTP cache), and we start receiving new data records. When we receive it, we want to update the application state, taking new data into account. But apart from that, we would like to add new data to already stored one in the cache. We might also think about removing the data, which is too old to keep (depends on the user case).

The database API we can use in this case is IndexedDB. It allows to create multiple databases with a custom name. Each database contains multiple object stores – one for each kind of thing we want to store. An object store contains multiple values - JS objects, strings, numbers, dates, arrays. Items in the object stores can have a separate primary key, which should be unique in the particular store, to identify an object. There are multiple operations that can be done to items in object stores: get, set, add, remove, iterate. All read or write operations in IndexedDB must be a part of a transaction: this means that if we create a transaction for a series of steps and one of those fails – none of them are going to be applied. The browser support of IndexedDB is good, because every major browser supports it.

In case the user wants to make a change to the data, there are two cases to handle.

- The request to the server is successful: in this case, the request would end up changing the data on the server's side, while the client will

just update its own cache once it gets a response from the server.

- The request to the server is not successful: here, it is going to be a bit more interesting. We will have to wait for an error message and store the data in a temporary database for the updates that are not sent to the server's side just yet. Afterwards, I think it would be logical to have a timer, which will check the connection with the server. Once it is back, every transaction again is going to be sent to the server. After all the data is sent, this temporary database can be cleaned.

Several problems could arise, however, which we will address in the architecture chapter.

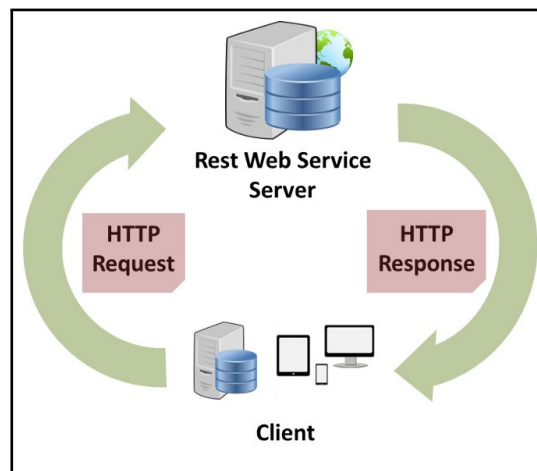
5.3 Cache API

If we want to store somewhere the HTML, the CSS and the JavaScript, as well as such assets as images, web fonts, then there is a place for all of it – Cache API. It allows conveniently manage the content for the offline use.

6 Implementation

In this chapter, we are going to explain the major problems that we encountered during the implementation phase.

Addressing the client, we will name the frameworks and techniques that we are going to use in the implementation phase. Concerning the server, which is going to provide the RESTful interface, though, we will cover the use cases.



Source: https://cdn-images-1.medium.com/max/660/1*EbBD6IXvf3o-YegUvRB_IA.jpeg

Figure 6.1: A look at the server-client architecture with a RESTful interface.

7 Evaluation

Evaluation

This chapter is needed, when results from the previous chapters need to be systematically discussed. This is the case, when an implementation needs to be assessed, or the results of a case study or an experiment need to be interpreted.

8 Conclusion

Conclusion

This is always the first chapter of the thesis. The chapter should be short (up to 5 pages). The chapter should feature sections as follows (where applicable):

- o Summary (Summarize this work in an insightful manner, assuming that the reader has seen the rest.)
- o Limitations or threats to validity (Point out the limitations of this work. In the case of empirical research, discuss threats to validity in a systematic manner.)
- o Future work (Provide insightful advice on where this research should be taken next.)

8.1 Summary

8.2 Future Work

- Automatic synchronization of updates;

Idempotence of updates

- wrote about this part in the notes to the architecture pdf

In case a client is sending an update to the server and does not receive an acknowledgement, what could have happened?

There are possibilities:

- The update was applied on the server, but the connection failed when the acknowledgement was about to be sent;
- The update was not applied on the server and the client did not receive an acknowledgement;

However, the client does not know which of these situations happened. Therefore, we have to find a general solution for such kind of behaviour.

One of the solutions could be the following: it does not matter, whether the update was applied or not. A client will just send the update again, till it does not receive an acknowledgement, regardless of what happens on the server's side.

8 Conclusion

The other one is: the client sends an update to the server, which should send a message back that it received an update. Afterwards, a client removes this update from the temporary database and sends back a message to the server that it is possible to apply the update.

However, the implementation of the solution for this problem is beyond the scope of this Master thesis. Therefore, these thoughts are going to be included in a section, where future improvements will be discussed.

List of Figures

| | | |
|-----|--|----|
| 4.1 | The way a client receives a key-value update from the server. | 8 |
| 4.2 | The way a client sends a write operation to the server. . . . | 8 |
| 4.3 | Graphical representation over a possible communicaton between two clients and a server. | 9 |
| 4.4 | An overview of the system's design. | 10 |
| 4.5 | Successful request of state from the local database – Se- quence diagram. | 12 |
| 4.6 | Successful storing of an operation in the local database – Sequence diagram. | 13 |
| 6.1 | A look at the server-client architecture with a RESTful in- terface. | 19 |

List of Tables

List of Code

Bibliography

- [1] Antidotedb. <https://www.antidotedb.eu/>, 2018 (accessed September 25, 2018).
- [2] Marc Shapiro, Nuno Pregui, Carlos Baquero, and Marek Zawirski. Replicated Data Types: A comprehensive study of Convergent and Commutative Replicated Data Types. (RR-7506):50, 2011. URL <http://hal.inria.fr/inria-00555588>.
- [3] Nuno Preguiça, Carlos Baquero, and Marc Shapiro. *Conflict-Free Replicated Data Types CRDTs*, pages 1–10. Springer International Publishing, Cham, 2018. ISBN 978-3-319-63962-8. doi: 10.1007/978-3-319-63962-8_185-1. URL https://doi.org/10.1007/978-3-319-63962-8_185-1.
- [4] Matt Gaunt. Service workers: an introduction. <https://developers.google.com/web/fundamentals/primers/service-workers/>, 2018 (accessed September 14, 2018).