FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Title of the Dissertation

Ângelo Miguel Tenreiro Teixeira

PREPARAÇÃO DA DISSERTAÇÃO

U. PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. João Correia Lopes

January 29, 2021

Title of the Dissertation

Ângelo Miguel Tenreiro Teixeira

Mestrado Integrado em Engenharia Informática e Computação

January 29, 2021

Abstract

TODO abstract

Keywords: keyword1, Keyword2, keyword3

Acknowledgements

TODO ACKNOLEGDEMENTS

Author

"Ceci n'est pas une citation."

Ângelo Teixeira

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
OT	Operational Transformation
CRDT	Conflict-free Replicated Data Type

Chapter 1

Introduction

As of today, millions of users follow their teams' games online to keep up-to-date regarding the events of a match [15]. Some of those had a special connection to their hometown team, but since they play in way lower leagues and without much exposure, oftentimes the users end up missing information and losing the passion they once had for the hometown team.

There is a specific group of users, however, that keeps following the games of the smaller teams, and most importantly: sharing updates about them. One platform that allows users to do that, as of today, is zerozero.pt, from ZOS. This enables the most passionate fans that still watch the smaller leagues to share what is going on in the game, reporting the events and building the game's history, totally community-driven. This tool exists and is somewhat outdated, hence the opportunity to build something better.

The goal is to allow multiple users to report the events that happen in a sporting event, which show up for everyone following that match in real-time. As internet connectivity is often poor inside stadiums, the tool must allow offline work, which is synced whenever possible. This can generate many data inconsistencies, which must be handled by the tool.

This project will provide an approach to this problem and the following sections provide more details on the key-objectives of the project. In Chapter 2, a comparison with a similar project is made, as well as a *State of the Art* exploration on the multiple scopes of this project. Chapters 3 and 4 define the problem and propose solutions for it, respectively. Conclusions are present in Chapter 5.

1.1 Offline Availability

As previously stated, internet connection in stadiums is poor most of the time. Thus, the users must have the option to interact with the application and synchronize once possible. This will obviously lead to data consistency issues (i.e. two users report a goal, changing the result to "1-0" for example, but one of them is offline, so when it finally synchronizes, the result is already “3-2” and it should not be overwritten.)

More information on this topic is presented in Section 2.1 and a proposed solution will be stated in Section 4.1.

1.2 Conflict Resolution

Another objective of the tool is to provide users with automatic conflict resolution when possible. Some strategies are depicted in the State of the Art section, in Chapter 2.2. Here, it is important to preserve the truth and the most up-to-date versions of data. In this scenario, there might not be a source of truth present to verify and validate all inputs, so other strategies must be used, such as an agreement-based implicit voting - if nobody questions a user’s input, it must be true until stated otherwise.

Additionally, different strategies can be used to solve conflicts automatically, thus improving the user experience. More on this topic is available in Section 2.2 and a preliminary proposed solution can be found in Section 4.2.

1.3 Reputation System

The third key-objective of the application will be the reputation system. Currently, there already exists a ranking concept, as well as a “trusted” user, which is the equivalent to the maximum reputation and should be considered as the source of truth in case of conflict.

But what about the cases where two “non-trusted” users’ inputs conflict, or even the case of two “trusted” users? Who should win? To resolve conflicts, an answer to these *conundrums* is fundamental. Ergo a new reputation system is required, and more details are available in Section 2.3. A preliminary proposed solution is presented in 4.3

1.4 Summary

TODO summary???

Chapter 2

Background and Literature Review

This section will dive deep on previously done work related to this project. First, a Background is provided, for the reader to have context on some relevant work and information that precedes the findings present in the following sections. Second, since the goal is to develop a complete application, there will be an analysis on the specific problems, and how they have been solved in the literature. Then, there will be a comparison between a similar work and similar existing applications.

Background

Since this project is intended for general use, the easiest and most common client available to users is the web client, also known as a web browser. In the early stages of the web, the applications followed a server-client architecture where the client had little to no work: it just rendered some previously compiled HTML and CSS on the page and the interaction with the server was made through HTML forms. Later on, JavaScript usage increased, and the pages started to be a bit more dynamic [61]. Still, it wasn't until more performant and portable devices such as the iPhone were available to the public, that web applications started to tilt their focus to the client side.

More recently, we start to see "Single Page Applications", which harness the client-side JavaScript capabilities to simulate legacy web interactions such as changing to another page or view without actually reloading the page, trading client-side work for network load [48] [33] [55] [54]. In this architecture, there still exists a server and clients (the web browsers), however, the server serves usually a REST API, and a bare-bones HTML document which serves as a base for the clients to render the rest of the document with JS, based on API calls results. There is also a mixed option, where the server handles some logic, usually related to session management or localization, and responds with an HTML document that already has some information to prevent the client-side to take too much time on work that will happen on every request. On the web, there

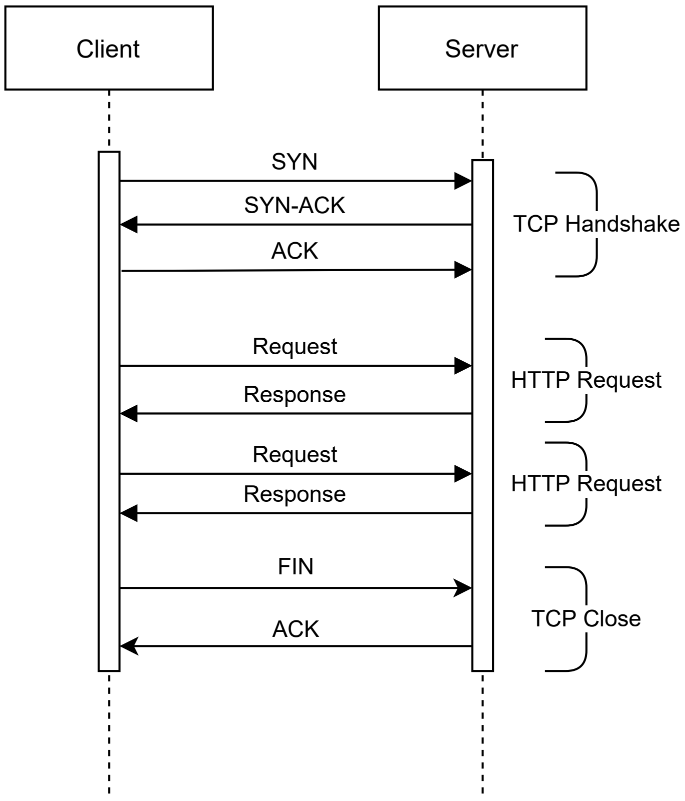

Figure 2.1: HTTP protocol example interactions

are some frameworks that allow this more recent architecture such as React [17], Vue [18] and Angular [16].

Since the web uses the HTTP Protocol [8] [11], it is quite hard to achieve some real-time behavior with good performance, due to the protocol's design [62]. As can be seen in Fig. 2.1, every data transfer starts with a request and ends with a response. It is therefore tied to this two-step process, which limits its potential. In order to allow a more performant way of data transmission between client and server, similar to the well-known data sockets available in Operating Systems used for bi-directional data transfer in real-time, there are Web Sockets [10] which serve the same purpose and allow Web Applications to send real-time updates to the clients without them needing to request them, which would only be possible otherwise using techniques such as polling (i.e. the client keeps asking the server for updates periodically).

TODO * web (arch) * real time (sockets?) * relevant technologies * localstorage (vs cookies comparison?) for offline mode * pagerank, parallelism to reputation system

2.1 Offline Availability

As stated in [52], even though people complete interrupted tasks in less time with no quality drop, these conditions make users work faster in order to compensate for it, increasing stress and frustration. The lack of offline interaction would force users to interrupt their task of inputting the desired information. That would increase stress when adding it later once they connect again, as they would try to do it fast while that information still has relevance. Indeed, [51] tested a real-world application in both scenarios — with offline mode and without it — for which the basic version obtained a user satisfaction score of 66.5, whereas the offline-tolerant version had a score of 95.5, out of 100, proving that having an offline mode improves substantially the user experience in a web application.

Marco [49] exposes the interrupted internet connectivity problem applied to e-learning applications. It proposes the creation of a modeling tool to create interruption-resilient web applications, by defining the possible operations and how they should behave in case of interruption. This model is presented later in [20]. It presents the Offline Model, defining a taxonomy for interruptions on web applications. It further analyzes some properties of web applications in the presence of interruptions such as scheme and offline support, which depend on the application and the task being performed within it.

Marco [51] further specifies rules regarding the adaptation of basic web applications into offline-tolerant ones. On the *Application Domain Level*, the data model of the application should not be modified so as to support as many applications as possible; On the *Hypertext Level*, the navigation on the application could be slightly modified or adapted, regarding pages within the web application, depending on the connectivity status; On the *Presentation Level*, the user interface should be adapted according to policies specific to each page element as the interaction with those elements would change depending on the connectivity status.

It is, therefore, necessary to establish which components of the web application will be available in offline mode, and how. To specify the navigation, they propose the notation presented by Albertos et al. [50], representing the web pages as nodes, with edges meaning navigation withing the web application. Each node has attributes referring to policies about the behavior of the components in offline mode, as well as the mechanism to create the local copy. To specify the interface behavior in offline mode, the following policies are described:

- **Hide**: The element should be hidden, preventing user interaction
- **Disable**: The element should be shown, but interaction should not be possible
- **Save**: Save elements locally so that they can be accessed in offline mode (such as images or text documents)
- **Update**: For elements that can be interacted with when in offline mode, they should be updated — synchronized — when connectivity is regained

Yang [71] presents a mechanism to support offline collaborative work in a web application. Due to the age of the document, most of the used tools are now obsolete, but the general ideas still apply. It proposes the existence of local storage to every client, so that operations done in offline mode are kept. Then, once the user comes back online, the work saved in the local storage is synchronized with the online storage, where conflict-resolution algorithms take place in order to minimize said conflicts.

Kleppmann, Wiggins, Hardenberg et al. [41] proposes ideals for local-first software, local-first software being the idea that the data on users' local machines is the primary when compared to servers, not the other way around, as it often happens in web applications:

1. **"No spinners" / No loading**: Data is changed locally, and synchronization occurs quietly in the background;
    - A workaround pattern is mentioned — Optimistic UI — which consists of showing the changes immediately, while sending them to the server, which would need to be reverted in case of error, for example
2. **Multi-device support**: Users should be able to work everywhere;
3. **Optional network**: Since local-first applications store the primary copy of their data in each device's local file system, the user can read and write this data anytime, even while offline. It is then synchronized with other devices sometime later, when a network connection is available;
4. **Seamless Collaboration**: Real-Time collaboration should be possible with conflict resolution when necessary;
5. **Data Forever**: The data is independent of the company or service provider, since it is stored locally. It should therefore last forever;
6. **Secure and Private by default**: By having the data on each user's device locally, database tampering is not as harmful, as you'll have many backed up replicas. Moreover, by using end-to-end encryption, the server can only save encrypted data, thus making it private, as it can only be decrypted by the users who are allowed to do so;
7. **Ownership**: The user owns the data, the provider cannot block the access in any form since the data is local.

It further compares multiple applications in the way they handle the local-first ideals, such as Files and Email attachments, or Google Docs [1] as well as technologies to implement said ideals, such as Web Applications, Mobile Applications, CouchDB [2] — a multi-master database that allows each node to mutate the database and synchronize the changes with other nodes, meant to run on servers — and PouchDB [6] — with the same goal of CouchDB, but meant to run on the end user devices.

It also mentions CRDT (Conflict-Free Replicated Data Types) as a foundational technology to achieve the local-first ideals. As explained in the next section, CRDT are general-purpose data structures similar to the common lists and maps but built for multi-user environments from the

[1]https://www.google.com/docs/about/

beginning. CRDT merge changes from multiple users when possible, however it does not handle changes to the same element in the structure, in that case it keeps track of the conflicts for the application to deal with them, allowing for custom conflict resolution techniques. CRDT are generic enough that they can synchronize over any communication connection like server, peer-to-peer networks, Bluetooth, and others. The paper further presents *Automerge* [2] as a JavaScript CRDT implementation.

Furthermore, the authors built some prototypes using Electron, JavaScript and React using CRDT to verify that its usage was already viable in order to build local-first software for the web and desktop applications. The main conclusions were that CRDT worked reliably, while integrating easily with the other tools and seamlessly merging data. Also, they verified that the user experience was splendid, as it allowed for offline work and sync when possible, giving the feeling of "data ownership" to the user. Finally, they state that this technology combines well with Functional Reactive Programming (FRP), a paradigm which renders the view based on a function that receives data. If the data changes, it "reacts" and redraws. A popular framework using this model is React, but there are others such as Vue.js or Flutter. With such a tool, by syncing the data with a CRDT and reacting to the changes in terms of UI, a real-time experience is easily achieved.

Finally, it added that CRDT might have a performance problem if used to same many changes (since they essentially save all the history). That is something that must be taken into account when using that technology and designing the system.

Zawirski, Preguiça, Duarte et al. [73] proposes a way of supporting many client-side applications sharing a database of objects that they can read and update under a convergent causal consistency model, with support for application-specific conflict resolution. It relies on fast local writes and possibly stale data reads, in order to achieve better speeds. It allows updates to thousands of clients using only three data centers, leveraging client buffering and controlled staleness, absorbing the cost of scalability, availability and consistency.

Kao, Lin, Yang et al. [40] proposes a system to allow mobile applications to run in a web container, joining the benefits of both: being able to run offline and cross-platform.

In order to make it possible for users to work while they are not connected to the internet, there must be a way of storing their actions locally, so that they can be synchronized when they regain internet connectivity. Currently, HTML5 exposes multiple local storage APIs which can be used for this effect.

Liu [47] discusses multiple web client storage technologies, comparing their advantages and disadvantages:

- **Cookies**: Small piece of data that is included in every HTTP request (limit of 4 KB per cookie [9]). They can be managed by the server as well as the client and are mostly used to keep state in between HTTP requests, for use cases such as user sessions or keeping track of some user activity. They are very popular and widely used, but at the same time they have little capacity and can be insecure, due to their presence in the HTTP communication with the server [67] [42];

[2]https://github.com/automerge/automerge

- **localStorage**: Can be used to implement cross-page communication (in the same domain) by storing data and having the web application listen for storage events, all pages of the same domain will have the updated data in the respective section of *localStorage*. It can be faster than cookies since no interaction with the server is needed. The data is stored in key-value pairs and is partitioned by domain, with no risk of inter-domain data corruption. It can be read and modified using JavaScript and all values are strings.
- **sessionStorage**: Works in the exact same way as *localStorage*, but the data only lasts until the browser is closed, i.e. the session;
- **UserData**: Older technology, specific to Internet Explorer, and has since been deprecated, it had only 128 KB of capacity;
- **Web SQL Databases**: Uses SQL format to store the local data. In Google Chrome, it uses the SQLite DB. The interactions are made as it were a normal SQL database running, by using the $executeSql()$ or $transaction()$ APIs. Not supported by all browsers and not part of W3C standard [13];
- **Indexed Database**: Uses a NoSQL DB to store the data. Event-based so it is asynchronous by default. It might be helpful to use a Promise-based wrapper [7] such as idb [3].

It further concludes that *localStorage* is the most compatible among web browsers. In summary, for session id tracking or for small data storage, one should use *Cookies*; Up to 5 MB, *localStorage* delivers a solution across all common browsers [14]; For more space, **Indexed Database** is recommended.

Jemel and Serhrouchni [39] presents some vulnerabilities that can arise from using the HTML5 Local Storage API as is. It also introduces a way of securing data stored using the local storage API, based on the user, not only the domain the data belongs to.

2.2 Conflict resolution

Conflict resolution is a problem that has been studied extensively in the literature, with publications dating as far back as 1986 [37], but it was not until 1989 that the first algorithm was proposed to deal with this problem, by Ellis and Gibbs [34].

In that paper, they define groupware systems as multi-user (i.e. composed of two or more users) computer systems that allow development on a common task, providing an interface to a shared environment. Then, they propose an algorithm to solve the groupware real-time concurrency problem called **Operational Transformation (OT)** which allows concurrent editing without the need for locks, increasing responsiveness. *Response time* is defined as the time required for the user's action to be reflected on their screen. *Notification time* is defined as the time required for the user's action to be propagated to all other participants.

Real-time groupware systems have the following characteristics:

[3]https://www.npmjs.com/package/idb

- **Highly interactive**: Short response times;
- **Real-time**: Notification times should be close to the response times;
- **Distributed**: They should work even if the participants are connected in different machines and networks on the internet;
- **Volatile**: Participants may enter and leave the session at any time;
- ***Ad Hoc***: Participants don't follow a script, it is therefore impossible to know what is the information they are trying to access beforehand;
- **Focused**: Generally users will be trying to access the same data, generating a high degree of access conflicts;
- **External Channel**: Participants are often connected among them via an external channel such as an audio or video communication tool;

A groupware system model is then defined as being formed by a set of sites and operators. Sites consist of a site process (i.e. a user's unique session), a site object (i.e. the data being read and modified), and a unique site identifier. Operators are the set of operations available for users to apply to the site objects. The goal is to maintain consistency among all the site objects at all times. The site process performs three kinds of activities: **operation generation**, where the user generates an operation to be applied to the site objects. The site will then encapsulate the action in an operation request to be broadcasted to all other sites; **operation reception**, where an operation is received from another site; **operation execution**, where an operation is executed on the local site object. The model further assumes that the number of sites is constant, messages are received exactly once, without error, and that it is impossible to execute an action before it is generated.

The paper further specifies the following definitions regarding the groupware system:

- Given two operations a and b, generated at sites 1 and 2, respectively, a precedes b iff:
  - $a = b$ and the generation of a happened before the generation of b, or
  - $a \neq b$ and the execution of a on site 2 happened before the generation of b;
- The **Precedence Property** states that if an operation a precedes another operation b, then at every site the execution of a happens before the execution of b;
- A groupware session is **quiescent** iff all generated operations have been executed at all sites;
- The **Convergent Property** states that site objects are identical at all sites at quiescence;
- A groupware system is *correct* iff the **Convergent Property** and the **Precedence Property** are always satisfied.

The proposed algorithm uses four auxiliary data structures: State vector, Request, Request Queue, Request Log and a predefined, semantics-based matrix of transformations.

State Vectors are based on the partial ordering definition in [43] and the concept of vector clocks in [46] and [35], stores the amount of operations done per site, i.e.e the i'th component of the vector represents how many operations from site i have been executed in the current site. It is therefore possible to compare two state vectors s_i and s_j:

- $s_i = s_j$ if each component of s_i is equal to the corresponding component in s_j;
- $s_i < s_j$ if each component of s_i is less than equal to the corresponding component in s_j and at least one component of s_i is less than the corresponding component in s_j;
- $s_i > s_j$ if at least one component of s_i is greater than the corresponding component in s_j;

Requests are tuples in the form $< i, s, o, p >$ where i is the originating site's identifier, s the originating site's state vector, o is the operation and p is the priority associated with o. From the request state vector, a site can determine if the operation to execute can be executed immediately, or wait for needed updates from other sites, enforcing the precedence property. The *Request Queue* is a list of requests pending execution. Even thought the term "queue" is used, it does not imply first-in-first-out order. The *Request Log* stores at site i the executed requests at that site, in insertion order.

The Transformation Matrix defines for every operation type pair a function T, that transforms operations so that given two operations o_i and o_j, with priorities p_i and p_j, instances of operators O_u and O_v, respectively and

$$o'_j = T_{uv}(o_j, o_i, p_j, p_i)$$

$$o'_i = T_{vu}(o_i, o_j, p_i, p_j)$$

then T is such that

$$o'_j \circ o_i = o'_i \circ o_j$$

$\circ$ meaning composition of operations.

The algorithm has an initialization section, a generation section, a reception section, and an execution section. In the initialization section, the site's log and request queue are set to empty, and the state vector is initialized with all values being 0, since no operations have been done. The next section specifies that whenever a local operation is received, a request is formed, and it is added to the local queue and broadcasted to other sites. In the receiving section, when a request is received, it is simply added to the request queue. Finally, the execution section specifies how to apply the operations, handling conflicts. First, it checks the request queue to retrieve any request (with state s_j) that can be executed, s_i being the state in the local site i and there are three possibilities:

1. $s_j > s_i$: The request cannot be executed since there are changes done in site j that were not executed yet at site i, therefore the request must be left in the queue for later execution;

2. $s_j = s_i$: The two states are equal, therefore the request can be executed immediately without operation transformation

3. $s_j < s_i$: The request can be executed, but the operation must be transformed, since site i has executed requests that are preceded by request j, r_j. Site i's log L_i is examined for requests that were not accounted for by site j (i.e. the requests that were executed in i but not on j prior to the generation of r_j. Each such request is then used to transform o_j in o'_j, according to the Transformation Matrix. o'_j is then executed and the state vector is incremented.

Some changes were later proposed to the state vector technique [44] [22] to allow dynamic entries, instead of a constant number of concurrent participants. Ellis and Gibbs [34] mentioned this in the OT definition and addressed it noting that participants can enter and leave every time the system is quiescent, since the Request Logs can be reset, and it should function like a checkpoint on each site.

Nichols, Curtis et al. [56] build another algorithm on top of the existing OT, presented in [34] that uses a server managing the collaboration, instead of being peer-to-peer like the former. This reduces the need for the request priority fields in the requests for tie-breaking, since the server can use a different strategy such as a reputation system, which the next section will develop upon. By removing the need for multicasting, since the server orchestrates the process and the communication is done in server-client pairs only, there is no need for message reordering logic, since a message transport protocol such as TCP [12] can be used instead, ensuring message delivery in the correct order before reaching the application layer, reducing the clients' workload.

Wang, Bu and Chen [68] propose a multi-version approach to resolving conflicts in a real-time multi-user graphic designing system. It is a **Compatible-Precedence** approach since, unlike a **Conflict-Precedence** approach where the objects are locked for resolution by users in case of conflict, in this case the conflict resolution is done by the system. It proposes a classification of operations in one of two types: Non Multi-version Operations (NMO) and Multi-version Operations (MO). The first are operations that, when in conflict, can be resolved by choosing one of them. The latter are operations that are better suited to be decided by the users, hence keeping both versions of the operation history is necessary. To represent this model, it uses a left-subtree-child and right-subtree-sibling binary tree. The left-subtree represents the continuous conflict-resolved operations, and each node will have a right-child representing a conflicting operation, thus creating a different branch which can have its own history, by adding left children to it. Finally, it proposes a set of algorithms to insert operations depending on their category — NMO or MO — and undo operations.

Citro, McGovern and Ryan [29] present an algorithm to handle both exclusive and non-exclusive conflicts. Exclusive conflicts are those in which operations leading to it neither can be realized at the same time nor can be executed in a specific order since the operations leading to the conflict overwrite each other, so it is impossible to define an order for them to execute maintaining the intention of both users. Non-Exclusive conflicts are those in which the operations can be realized at the same time, being handled using conventional consistency management techniques such as operational transformation [34].

The paper builds on top of existing techniques such as operational transformation and multi-versioning in order to more effectively handle both types of conflicts. Additionally, it does not suffer from a "partial intention" problem, since it allows delayed post-locking, based on the post-locking of objects technique proposed by Xue et al. [70] so that users can fully express their intentions on the object, allowing for a more informed decision when manually resolving the conflict; it also does not require a group leader or other conflict resolution roles in order to resolve conflicts.

Over the years, more implementations were published each with their own advantages and compromises, but for the sake of keeping the focus of this document in a more general view of existing alternatives, I redirect the reader to OT Wikipedia article which has a good summary table [4] of most of the alternatives and their characteristics. Additionally, Sun, Chengzheng provides a FAQ page [5] serving as a knowledge base about the topic. This being a popular algorithm, there are many tools that implement it and can be included in an application, such as TogetherJS [6], based around what they call the hub: a server that everyone in the session connects to which echoes messages to all the participants using WebSockets, or ShareDB [7], a real-time database backend supporting OT. The default implementation of ShareDB is an in-memory, non-persistent database with no queries. It is possible, however, to create connectors for other databases and connectors are provided for MongoDB[8] and PostgreSQL [9].

ShareDB is horizontally scalable using a publish and subscribe mechanism, making it relatively future-proof. Since it uses Express for the server it is possible to use middleware to hook into the server pipeline and modify objects as they pass through ShareDB, adding flexibility.

Conflict-free Replicated Data Types (CRDT) are a different approach to the real-time coordination of inputs problem in a groupware system [60]. They can be operation-based [45] [25], similar to OT, or state-based [26] [21].

Operation-based CRDT are also called commutative replicated data types, or CmRDTs. CmRDT replicas propagate state by transmitting only the update operation, similarly to OT. Replicas receive the updates and apply them locally. The operations are commutative. However, they are not necessarily idempotent. The communications' infrastructure must therefore ensure that all operations on a replica are delivered to the other replicas, without duplication, but in any order.

State-based CRDT are called convergent replicated data types, or CvRDTs. In contrast to CmRDTs, CvRDTs send their full local state to other replicas, where the states are merged by a function that must be commutative, associative, and idempotent. The merge function provides a

[4]https://en.wikipedia.org/wiki/Operational_transformation#OT_control_(integration)_algorithms
[5]https://www3.ntu.edu.sg/scse/staff/czsun/projects/otfaq/#_Toc321146200
[6]https://togetherjs.com
[7]https://github.com/share/sharedb
[8]https://www.mongodb.com/
[9]https://www.postgresql.org/

join for any pair of replica states, so the set of all states forms a *semilattice*. The update function must monotonically increase the internal state, according to the same partial order rules as the *semilattice*. Kleppmann, Martin provides good presentations[10] [11] on the CRDT topic. Similarly to OT, CRDT also has available libraries and tools to be used in applications. Riak[12] is a distributed NoSQL key-value data store based on CRDT. Bet365 is an example of a large system using Riak[13]. Another example of CRDT implementation is Facebook's Apollo database[14], showing that CRDT can be used at scale as well.

A third and more recent alternative to the conflict resolution problem is Differential Synchronization [36] [5]. It provides an alternative to OT, being an algorithm that is symmetrical, as it has nearly identical code in both client and server; state-based, thus not requiring a history of edits to be kept by clients; asynchronous, since it does not block user input while waiting for the response over the network; network resilient, convergent, suitable for any content for which semantic diff and patch algorithms exist; and highly scalable. A working example of this algorithm can be seen in MobWrite[15].

As it can be noted, real-time applications usually apply a replicated architecture, where each client has its own replica, a user may directly edit its own version, realizing its effect immediately. The effects are then propagated to other clients. Propagation can be operation-based [57] [64] [63] [69] or state-based [36]. Chengzheng, David, Agustina et al. [28] state that most real-time co-editors, including those based on OT and CRDT, have adopted the operation propagation approach for communication efficiency, among others. It continues by comparing both solutions. Firstly, CRDT solutions have significantly higher time and space complexities than OT solutions, as revealed in [65]. Secondly, CRDT has a higher cost of initialization, since it needs to represent initial characters on the document, whereas OT's data structures start empty. This can have a big impact in session management and handling users that enter mid-session. Thirdly, OT does not have any additional time or space cost for non-concurrent operations, as the operations buffer can be emptied with garbage collection — as was previously mentioned, the system can reset some data structures every time it is quiescent. CRDT, on the other hand, will have similar time and space costs regardless of handling concurrent or sequential operations, as all operations will be applied in the internal object sequence, which can never be emptied unless the document itself is emptied. Finally, OT does not have any additional time or space cost for local operations, as they will never be concurrent with any operation in the Requests Log. CRDT, on the other hand, will have similar time and space costs regardless of handling local or remote operations, as all operations will be applied in the internal object sequence. The longer time the local operation processing takes, the less responsive the co-editor is to the local user. Herron [3] also compares OT to CRDT, looking at the implementation in a final application, revealing some examples and

[10] https://www.infoq.com/presentations/crdt-distributed-consistency/
[11] https://gotocon.com/berlin-2016/presentations/show_talk.jsp?oid=7910
[12] https://riak.com/introducing-riak-2-0/
[13] https://bet365techblog.com/riak-update
[14] https://dzone.com/articles/facebook-announces-apollo-qcon
[15] https://code.google.com/archive/p/google-mobwrite/

problems that can arise, such as the unwanted cursor movement.

In a more general way, Sun et al. [66] present different types of conflict resolution in a creative editing system. The **preventive resolution** prohibits concurrent work two same objects, avoiding conflicts altogether; **eliminative resolution** eliminates both operations when in conflict, eliminating the operations' history; **arbitrative resolution** elects one of the operations to be kept; **preservative resolution** keeps both options in different versions, so that the users can choose an alternative later; **creative resolution** produces a new operation, combining the two conflicting operations.

It further elaborates on the basic structure of a creative conflict resolution. There are three main components layered on top of each other. On the base, there is a conflict detector, that simply detects if there are conflicts between operations. Next, there is a conflict synthesis component, that creates the new operations based on the conflicts. Finally, there is the conflict management UI, that is responsible for notifying users about conflicts and allowing them to select the desired conflict resolution strategy.

2.3 Reputation System

There are multiple examples of how reputation can be used in multi-user systems and how it can affect the group dynamics. Many refer to it as a solution to "Group Recommendations", which are based in **trust** among participants whereas others mention its ability to induce cooperation. Haveliwala [38] shows how the PageRank algorithm can be personalized so that each link among nodes has a different weight, in order to express a dynamic preference among nodes. Andersen et al. [23] demonstrates multiple trust-based recommendation systems and how they comply with a set of relevant axioms. Most importantly, it shows how the aforementioned personalized PageRank (PPR) algorithm can be used to simulate a trust network among peers, by linking users with differently weighted connections. The greater the weight, the more a user trusts another, and the most likely it is for the Random Walk algorithm to choose that "path of trust". The latter also shows that PPR satisfies three out of five relevant axioms: **Symmetry**, **Positive Response**, **Transitivity**, but not Independence of Irrelevant Stuff and **Neighborhood Consensus**.

- **Symmetry.** Isomorphic graphs result in corresponding isomorphic recommendations (anonymity), and the system is also symmetric

- **Positive response.** If a node's recommendation is 0 and an edge is added to a + voter, then the former's recommendation becomes +.

- **Transitivity.** For any graph (N, E) and disjoint sets $A, B, C \subseteq N$, for any source s, if s trusts A more than B, and s trusts B more than C, then s trusts A more than C.

- **Independence of Irrelevant Stuff (IIS).** A node's recommendation is independent of agents not reach- able from that node. Recommendations are also independent of edges leaving voters.

- **Neighborhood consensus.** If a nonvoter's neighbors unanimously vote +, then the recommendation of other nodes will remain unchanged if that node instead becomes a + voter.

Dellarocas [32] shows examples of how multiple platforms handle their user reputations mechanisms. It also states that reputation systems can prevent badly intended users and deter moral hazard by acting as sanctioning devices. If the community punishes users that behave poorly and if the punishment compensates the "cheating" profit, then the threat of public revelation of a user's cheating behavior is an incentive for users to cooperate instead. It further elaborates on the reputation dynamics of a multi-user application:

- **Initial Phase**: Reputation effects begin to work immediately and in fact are strongest during the initial phase, as users try and work hard to build a reputation on themselves. Reputation effects may fail, however, when short-run users are "too cautious" when compared to the long-run ones and therefore update their beliefs too slowly in order for the long-run user to find it profitable to try to build a reputation;

- **Steady state** (or lack thereof): In their simplest form, reputation systems are characterized by an equilibrium in which the long-run user repeatedly executes the safe action, also known as the Stackelberg action, and the user's reputation converges to the Stackelberg type (always collaborating and no cheating).

These dynamics have important repercussions for reputation systems. Dellarocas goes on to say that if the entire feedback history of a user is made available to everyone and if a collaborator stays on the system long enough, once he establishes an initial reputation for honesty will be tempted to cheat other users sometimes. In the long term, this behavior will lead to an eventual collapse of his reputation and therefore of cooperative behavior.

Bakos and Dellarocas [24] present a model for a reputation system which explores the ability of online reputation mechanisms to efficiently induce cooperation, when compared to contractual arrangements relying on the threat of litigation. It concludes that the effectiveness of a reputation mechanism in inducing cooperative behavior depends on the frequency of transactions that are affected by this mechanism, reminding that a minimum degree of participation is required before reputation can induce a significant level of cooperation. After this threshold is reached, however, the power of reputation manifests itself and high levels of cooperation can be supported.

Dellarocas [31] concludes that reputation mechanisms can induce higher cooperation and efficiency if, instead of publishing updated ratings as soon as they are available, they only update a user's public reputation every *n* transactions, meaning a summary statistic of a user's last ratings. In settings with noise, infrequent updating increases efficiency because it decreases the adverse consequence of artificial negative ratings. At the same time, however, infrequent updating increases a user's short-term gains from bad behavior and thus the minimum future punishment threat that can sustain cooperation.

In [19], tests were made in order to understand the reputation issues for users. These were made in Waze, a GPS-like driving assistant with crowd collaboration for road events. Even though this and zerozero.live are somewhat different, some paralelisms can be made and some gathered information still applies. They concluded that it was hard for users to recognize where the information came from, and if it was reliable at all. Furthermore, users did not care much about their reputation when submitting information (i.e. if they heard about some road event, they would publish it without verifying it), maybe this is somewhat different from our use-case of sporting events, as users are either actually watching the game, or following it from a reliable source. Additionally, when users knew the source of data, they tended to trust people in their close circle (e.g. family and friends) and the main conclusion is that the app needed to better convey the reputation of the source to let the consumers know how much they can or should trust the source.

Resnick et al. [59] elaborate about reputation systems and their generic importance on the web. It is more geared towards e-commerce examples where people investigate the reputation before interacting with each other. It mentions three important properties reputation systems should have:

- Long-lived entities that inspire an expectation of future interaction. If the entities are short-lived, their reputation matters little;
- Capture and distribution of feedback about current interactions (such information must be visible in the future);
- Use of feedback to guide trust decisions;

In the zerozero.live case, it might be hard to get expressive feedback from users regarding other users. Therefore, it is important to have some kind of implicit voting in place. Additionally, users are more inclined to express feedback when they disagree than when they agree, which means that the lack of negative feedback must be considered as some sort of positive feedback in order to balance the system. Besides, users won't see the reputation of other users beforehand in order to decide to interact or not, as they simply enter the event without knowing who is also there, so it is important that they can see the reputation, or a variant of it (i.e. some relative reputation based on the current group of users) while they are at the event (e.g. Showing it next to the user's name).

Melnikov, Lee, Rivera et al. [53] present a dynamic interaction based reputation model (DIB-RM), which is further evaluated in [72]. It presents a method to measure reputation as a function of user interaction frequency, also contemplating a reputation decay if the users stop contributing to the platform.

The aforementioned method is also present in [30], where the authors present a way to harness the "wisdom of the crowds", very much in line with what is required in zerozero.live, since there is no express authority during the event. It presents an example of a document sharing system and the approach to rank the documents based on the amount of readers, the reputation of the author, the time dynamics of reader consumption, and the time dynamics of documents contributed by the user. This last one manifests indirectly, but is still relevant: it means that if a user has less frequent readers on their documents, their reputation will decrease, so the contribution to the main document's reputation — the one they are reading now — will be smaller. Reputation values scale between 0 and 1, and it sticks to the following rules:

1. Every time a user consumes a document from an author, the author gains reputation according to:

   $$newRep = oldRep + (1 - oldRep) * repReward$$

   repReward is a constant between 0 and 1 and should consider the number of entities in the system. As the paper states: "If the number of expected consumers is in the order of hundreds or thousands, then an overly high value of *repReward* will potentially cause popular content to quickly converge towards 1 making it difficult to differentiate between similarly popular content.".

2. Every time a user consumes a document, the document gains "reputation" — meaning popularity in this case — according to the same formula of (1):

   $$newRep = oldRep + (1 - oldRep) * repReward$$

3. In order to take time dynamics into account, reputation should decrease over time, so that a "rich-get-richer" paradigm can be avoided. This is achieved by the following equation (both for users and for documents):

   $$newRep = oldRep * decayCoeff^k$$

   decayCoeff represents how much the reputation will change, and k is the amount of time units that have passed since the last reputation update, i.e. for a time unit of "days", k will be 0 in the first 24h, 1 in the next day, 7 in a week, and so on. This decouples the algorithm from the logistics, since the algorithm can now run in a fixed frequency, independently of the time units, and every time it re-calculates, it will give an accurate value. However, if for example the time unit is "day", and the algorithm updates every week only, there will be an offset of 6 days in which the value will be outdated.

4. Users with higher reputation matter more when calculating the document reputation changes:

   $$newRep = oldRep * repConsumer * B$$

   B is a constant within [0, 1] representing to what extent the user reputation *repConsumer* will influence the document's reputation.

   This system can be adapted and applied in zerozero.live if we map user inputs in an event as documents. However, we will be ranking users instead of inputs — "documents" in the analogy — even though they will also have reputation values. This will be explained in more detail in Section 4.3.

2.4 Similar platforms

On a basic level, this is a sporting-event following app. A similar platform would be 365scores.com [1], which offers the following of the same events in real-time, however it does not offer the community-input feature of this proposed work.

Another platform that enables live viewing of sporting events is mycujoo.tv [4]. This one enables the teams themselves to livestream the game with video, and mark specific events as they happen, so that the viewers can revisit those moments in the video. It, too, lacks the community input feature when inserting the events; it is more geared towards the clubs sharing ability, rather than the fans'.

This leaves zerozero.live as a singular app that will allow fans to contribute with the games' events in real-time, increasing engagement, which can be complemented with the enormous football-related database which can provide real-time statistics about the game.

2.5 Similar work

Castro, João [58] has developed an application with the same goal, as a Master's Thesis as well. This work, however, will not be a continuation of Castro's work or use any of its code. It will benefit solely from the insights it can give, being a work with the same goal, with high importance in terms of literature review.

Castro's work focused mainly on the reputation system as a conflict resolution strategy (i.e. the user with the most reputation wins an argument over the user with less reputation). While this is a valid approach to start with, in the real world it has a lot of limitations such as highly-reputed users abusing their power. Further discussion about reputation systems in the literature is shown in Section 2.3. This work, however, intends to apply a different technique that, while harnessing the advantages of a reputation system, aims to prevent the problems that could arise when used by real users. One of them would be using different conflict resolution strategies, depending on the conflict strategy (i.e. a conflict in the game score is way more important and thus cannot be solved by blindly applying a reputation comparison than, say, a mistake on the player substitution). The way of solving conflicts in terms of User Experience is also a matter of study, as we don't want to fact-check every user input and disturb every other user experience with it, will at the same time guaranteeing the most true story possible. Finally, this work will have an "Offline Availability" goal as well, which is of great relevance in the real world, as the connectivity is not always the best, and many consistency problems result from it thus, it's only fair that it is included in the areas of study regarding this application.

Chapter 3

Problem Statement

This chapter describes the problem tackled in this dissertation, including the planned features, the expected result and the planning to achieve it. Section 3.1 describes the features to be developed. Section 3.2 addresses the planning of the intended tasks. The planned development methodology is defined in Section 3.3. Some preliminary ideas on the resolution of parts of the problem will be presented in the next chapter (Chapter 4).

3.1 Problem Definition

As mentioned in the Introduction (Chapter 1), the goal of the project is to develop a web application that allows users to follow a sport event in a real-time chat-like environment where everyone can input game events. For users that are just following, it would work like a live coverage of the event, for contributors it should be resilient to network failures, due to the Wi-Fi limitations of stadiums.

Due to the above goal, some necessary features start to surface such as real-time conflict resolution and the inherent reputation system for tie-breaking when necessary.

A prototype that allows event submission is already available, and since it is using React, and it is an appropriate technology for this task, it will be kept. Some features still need to be polished, but most of the UI is already done, which will allow a bigger focus on the actual real-time conflict resolution problem. The features are described as follows, in User Story format[1]:

US01 As a user, I want to be able to join a sport event channel, where I can see details about the event in real-time (either pre-filled or contributed by other users)

US02 As a user, I want to be able to be able to post event updates while on an event channel, in a chat-like interaction

[1]In software development and product management, a user story is an informal, natural language description of one or more features of a software system. User stories are often written from the perspective of an end user or user of a system.

  - Event updates include: Starting players, Goals, Fouls, Set-Pieces, Cards, Substitutions

US03 As a user, I want to be able to use the application while in offline mode, and have it synchronize once the connection is resumed

US04 As a user, I want to see a value representing the reputation of other users in a given event channel

US05 As a user, I want to be able to report inputted events as false

US06 As a user, I want to be able to see if there are any pending conflicts to be resolved

US07 As a user, I want to be able to resolve any pending conflicts

US08 As a user, I want to be able to join an event channel mid-session, being able to see the previous information

3.2 Planning

TODO GANTT here

3.3 Methodology

TODO mention scrum like stuff, in order to be able to gather feedback with ready products in between sprints (vs kanban which would be more continuous)

This work is intended to be tested on real users during its development. Thus, an agile methodology seems to be more fitting than a waterfall approach [27]. The most common are Scrum, XP and Kanban.

Chapter 4

Problem Solution

In this chapter, a preliminary approach for the problem defined in the previous chapter is presented. In some cases, it is still not possible to make a clear decision, as it requires implementation and further testing to validate that it actually works, and really fits the need. These proposed approaches are based on the research done in the Chapter 2, projected into the application domain and specific needs. This chapter is divided in three sections, which will more precisely elaborate on their respective topic, presenting one or multiple proposed solutions for that specific area.

TODO mention API

4.1 Offline Availability

According to the research done in Chapter 2, there needs to be a way of storing state locally, in case the user is offline. Since, this application is a web application, from the alternatives presented, **localStorage** seems to be the most available among browsers[1], while fitting the project's needs.

Currently, in the already existing "Proof-of-Concept", there is an already implemented solution, which uses a local queue of requests stored in *localStorage*, so as to not overwhelm the server with changes every second. This queue is "dumped" every 10 seconds, that is, every 10 seconds, the operations batched in the queue are sent to the server. While this allows for prevention of operations loss in case of sudden network connectivity, it might not be the best approach in terms of real-time and user experience. For example, let's say that a user generates a conflicting operation, but that operation is only sent 10 seconds later. Only then the user will see that effect on is device, when it could have happened sooner.

My proposal is to reduce the waiting time to a more reasonable 2 seconds, which, while preventing the overwhelming of messages to the server, reduces notification time and still lets the user cancel the operation before it being sent.

[1]https://caniuse.com/?search=localstorage

4.2 Conflict Resolution

This is a topic on which there is no clear solution. As it was seen in the research, the most well-know and used approaches are OT and CRDT. The community cannot, however, elect a clear winner[2], they are just *different*. On the OT side, a common approach is to use ShareDB[3], together with the JSON OT type definition[4].

Another alternative would be to use CouchDB[5] with its web browser counterpart, PouchDB[6]. It allows for replication of state among all users, with complete control on conflict resolution: When CouchDB encounters a conflict scenario, it arbitrarily chooses a winner, deterministically, however, it keeps the conflicting version as well, which can be used to solve the conflict with custom application logic, for example, based on a reputation system, or on merging the two inputs — Creative Resolution (Section 2.2)

On the CRDT side, there are two options: Automerge[7] and GUN[8]. Since the latter's documentation seems to be lacking, I am removing it from the options pool. Automerge is flexible enough, allowing for server-client network architecture, as well as peer-to-peer, and it works like JSON CRDT were described to work: each user has a local copy of the JSON state, which can be locally mutated, even when offline, and it will sync automatically with other nodes. It works similarly to the CouchDB/PouchDB conflict handling, in being as much automatic as possible, but letting the application know about existing conflicts and handle them.

At this point, the ShareDB option seems to work on a lower level than the other two, and it might be unfeasible to have good results in the expected timeframe. Between CouchDB/PouchDB and Automerge, there is no clear winner, the only distinctive charateristic being that CouchDB is more mature and has been used in production for much longer. Nevertheless, the wise decision here would be to try both in a small "Proof-of-Concept" and verify their usability.

4.3 Reputation System

As was mentioned in the Literature Review (Section 2.3), an effective method to achieve a fair reputation system, which takes into account the time dynamics of user interactions as well as their current reputation, is to implement a personalized PageRank algorithm, which takes into account the reputation of users when calculating vouching or invalidation, in order to achieve a weighted voting system so as to provide long-term reputable users with a prize for their good behavior. Recalling the system present in [30], there are 4 rules involved in adapting the system to our use case:

1. Every time a user consumes a document from an author, the author gains reputation;

[2]https://news.ycombinator.com/item?id=22039950
[3]https://github.com/share/sharedb
[4]https://github.com/ottypes/json0
[5]https://couchdb.apache.org/
[6]https://pouchdb.com/
[7]https://github.com/automerge/automerge
[8]https://github.com/amark/gun

2. Every time a user consumes a document, the document gains "reputation" (i.e. popularity);
3. In order to take time dynamics into account, reputation should decrease over time, so that a "rich-get-richer" paradigm can be avoided (both for users and for documents);
4. Users with higher reputation matter more when calculating the document reputation changes;

With this in mind, I propose the following rules to adapt this to our scenario:

- Every time a user agrees with an input, he will improve the input's reputation according to rules 2 and 4;

$$\begin{aligned} newInputRep = oldInputRep + (1 - oldInputRep) * maxRepReward * userRep \\ * userRepInfluence \end{aligned} \quad (4.1)$$

- Every time a user disagrees with an input (either by inputting a real-conflicting input or reporting as false/inaccurate) he will worsen the input's rep according to rules (2's reverse) and 4;

$$newInputRep = oldInputRep * (1 - maxRepPunishment * userRep * userRepInfluence) \quad (4.2)$$

- Every time a user submits a falsely-conflicting input, meaning that both users submitted the same information resulting in duplicated information, it should act as an explicit agreement with the other user's input, so it should count more, according to an *explicitAgreementBonus* constant, which must be greater than zero to achieve the bonus effect;

$$\begin{aligned} newInputRep = oldInputRep \\ + ((1 - oldInputRep) * repReward * userRep * userRepInfluence) \\ * (1 + explicitAgreementBonus) \end{aligned} \quad (4.3)$$

- The user gains reputation according to the average of its inputs' reputations. Only takes into account the latest inputs, referring to the last event which will trigger the reputation update;

$$newUserRep = oldUserRep + (1 - oldUserRep) * \frac{\sum inputRep}{numInputs} \quad (4.4)$$

- Each user has a reputation decay according to rule 3, the time unit should be 1 week since there's at least one relevant game per week. This prevents users that generate a lot of inputs in a single game to enjoy their reputation boost for many more games, since they need to be consistent every week: it matters more if they make an input every week than 20 inputs once every 2 or more weeks. This decay is on a higher level than the events, creating 20 inputs in an event is roughly the same as 1 input in an event (since the football events last around 90 minutes)

$$newUserRep = oldUserRep * decayCoefficient^{timeSinceLastUpdate} \tag{4.5}$$

- The reputation values are updated at the end of each event, according to the event’s history.

Chapter 5

Conclusions

conclusions intro

5.1 Expected Results

something else?

Appendix A

Loren Ipsum

References

[1] 365scores. https://www.365scores.com/pt/pages/about.

[2] Apache CouchDB. https://couchdb.apache.org.

[3] Building real-time collaboration applications: OT vs CRDT. https://www.tiny.cloud/blog/real-time-collaboration-ot-vs-crdt/.

[4] MyCujoo. https://mycujoo.tv/en/about-us.

[5] Neil Fraser: Writing: Differential Synchronization. https://neil.fraser.name/writing/sync/.

[6] PouchDB. https://pouchdb.com/.

[7] Promise API JS MDN. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.

[8] RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1. https://tools.ietf.org/html/rfc2616.

[9] RFC 6265 - HTTP State Management Mechanism. https://tools.ietf.org/html/rfc6265.

[10] RFC 6455 - The WebSocket Protocol. https://tools.ietf.org/html/rfc6455.

[11] RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2). https://tools.ietf.org/html/rfc7540.

[12] RFC 793 - Transmission Control Protocol. https://tools.ietf.org/html/rfc793.

[13] Web SQL Database W3C. https://www.w3.org/TR/webdatabase/.

[14] Web Storage W3C. https://www.w3.org/TR/webstorage/.

[15] The Changing Profile of Sports Fans Around The World. https://www.facebook.com/business/news/insights/the-changing-profile-of-sports-fans-around-the-world, 2019.

[16] Angular. https://angular.io/, 2021.

[17] React. https://reactjs.org/, 2021.

[18] Vue. https://vuejs.org/, 2021.

[19] Orlando P. Afonso, Luciana C.de C. Salgado, and José Viterbo. User's understanding of reputation issues in a community based mobile app. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9742, pages 93–103. Springer Verlag, 2016.

[20] Felix Albertos-Marco, Victor M.R. Penichet, Jose A. Gallud, and Marco Winckler. A model-based approach for describing offline navigation of web applications. *Journal of Web Engineering*, 16(1-2):1–38, mar 2017.

[21] Paulo Almeida, Ali Shoker, and Carlos Baquero. Efficient State-based CRDTs by Delta-Mutation. 2014.

[22] Paulo Sérgio Almeida, Carlos Baquero, and Victor Fonte. Interval Tree Clocks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5401 LNCS, pages 259–274. Springer, 2008.

[23] Reid Andersen, Christian Borgs, Jennifer Chayes, Uriel Feige, Abraham Flaxman, Adam Kalai, Vahab Mirrokni, and Moshe Tennenholtz. Trust-based recommendation systems: An axiomatic approach. In *Proceeding of the 17th International Conference on World Wide Web 2008, WWW'08*, pages 199–208, 2008.

[24] Yannis Bakos and Chrysanthos Dellarocas. Cooperation Without Enforcement? A comparative analysis of litigation and online reputation as quality assurance mechanisms. Technical report, 2003.

[25] Carlos Baquero, Paulo Sérgio Almeida, and Ali Shoker. Making Operation-Based CRDTs Operation-Based. In *Proceedings of the First Workshop on Principles and Practice of Eventual Consistency*, PaPEC '14, New York, NY, USA, 2014. Association for Computing Machinery.

[26] Carlos Baquero and Francisco Moura. Using structural characteristics for autonomous operation. *Operating Systems Review (ACM)*, 33(4):90–96, 1999.

[27] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for Agile Software Development, 2001.

[28] S. U.N. Chengzheng, S. U.N. David, N. G. Agustina, C. A.I. Weiwei, and C. H.O. Bryden. Real differences between OT and CRDT under a general transformation framework for consistency maintenance in co-editors. *Proceedings of the ACM on Human-Computer Interaction*, 4(GROUP):1–26, jan 2020.

[29] Sandy Citro, Jim McGovern, and Caspar Ryan. Conflict management for real-time collaborative editing in mobile replicated architectures. In *Conferences in Research and Practice in Information Technology Series*, volume 62, pages 115–124, 2007.

[30] Elizabeth M. Daly. Harnessing wisdom of the crowds dynamics for time-dependent reputation and ranking. In *Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining, ASONAM 2009*, pages 267–272, 2009.

[31] Chrysanthos Dellarocas. How often should reputation mechanisms update a trader's reputation profile? *Information Systems Research*, 17(3):271–285, 2006.

[32] Chrysanthos Dellarocas and R. Smith. Reputation Mechanisms. 2005.

[33] Anna Derezińska and Krzysztof Kwaśnik. Evaluation and Improvement of Web Application Quality – A Case Study. In *Advances in Intelligent Systems and Computing*, volume 1173 AISC, pages 187–196. Springer, 2020.

[34] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume Part F1301, pages 399–407, New York, New York, USA, jun 1989. Association for Computing Machinery.

[35] C Fidge. Timestamps in Message-Passing Systems That Preserve the Partial Ordering. 1988.

[36] Neil Fraser. Differential Synchronization. In *DocEng'09, Proceedings of the 2009 ACM Symposium on Document Engineering*, pages 13–20, 2 Penn Plaza, Suite 701, New York, New York 10121-0701, 2009.

[37] Irene Greif, Robert Seliger, and William E Weihl. Atomic Data Abstractions in a Distributed Collaborative Editing System. In *Proceedings of the 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '86, pages 160–172, New York, NY, USA, 1986. Association for Computing Machinery.

[38] Taher H. Haveliwala. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):784–796, jul 2003.

[39] Mayssa Jemel and Ahmed Serhrouchni. Content protection and secure synchronization of HTML5 local storage data. In *2014 IEEE 11th Consumer Communications and Networking Conference, CCNC 2014*, pages 539–540. IEEE Computer Society, 2014.

[40] Yung Wei Kao, Chiafeng Lin, Kuei An Yang, and Shyan Ming Yuan. A web-based, offline-able, and personalized runtime environment for executing applications on mobile devices. *Computer Standards and Interfaces*, 34(1):212–224, jan 2012.

[41] Martin Kleppmann, Adam Wiggins, Peter Van Hardenberg, and Mark McGranaghan. Local-first software: You Own Your Data, in spite of the Cloud. In *Onward! 2019 - Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, co-located with SPLASH 2019*, pages 154–178. Association for Computing Machinery, Inc, oct 2019.

[42] Hyunsoo Kwon, Hyunjae Nam, Sangtae Lee, Changhee Hahn, and Junbeom Hur. (In-)Security of Cookies in HTTPS: Cookie Theft by Removing Cookie Flags. *IEEE Transactions on Information Forensics and Security*, 15:1204–1215, 2020.

[43] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7):558–565, 1978.

[44] Tobias Landes. Dynamic Vector Clocks for Consistent Ordering of Events in Dynamic Distributed Applications. pages 31–37, 2006.

[45] Mihai Leţia, Nuno Preguiça, and Marc Shapiro. Consistency without concurrency control in large, dynamic systems. In *Operating Systems Review (ACM)*, volume 44, pages 29–34, apr 2010.

[46] Barbara Liskov and Rivka Ladin. Highly Available Distributed Services and Fault-Tolerant Distributed Garbage Collection. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '86, pages 29–39, New York, NY, USA, 1986. Association for Computing Machinery.

[47] Wen Tao Liu. Research on offline storage of web page. In *Applied Mechanics and Materials*, volume 518, pages 305–309, 2014.

[48] Hector M. Lugo-Cordero and Ratan K. Guha. A secured distribution of server loads to clients. In *Proceedings - IEEE Military Communications Conference MILCOM*, volume 2015-Decem, pages 372–377. Institute of Electrical and Electronics Engineers Inc., dec 2015.

[49] Félix Albertos Marco. Supporting offline interaction with web sites resilient to interruptions applied to E-learning environments. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8295 LNCS, pages 310–314, 2013.

[50] Félix Albertos Marco, José Gallud, Victor M.R. Penichet, and Marco Winckler. A model-based approach for supporting offline interaction with web sites resilient to interruptions. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8295 LNCS, pages 156–171, 2013.

[51] Félix Albertos Marco, Víctor M.R. Penichet, and José A. Gallud. User interaction with offline web applications: A case study. In *ACM International Conference Proceeding Series*, volume 07-09-Sept. Association for Computing Machinery, sep 2015.

[52] Gloria Mark, Daniela Gudith, and Ulrich Klocke. The cost of interrupted work: More speed and stress. In *Conference on Human Factors in Computing Systems - Proceedings*, pages 107–110, 2008.

[53] Almaz Melnikov, Jooyoung Lee, Victor Rivera, Manuel Mazzara, and Luca Longo. Towards Dynamic Interaction-Based Reputation Models. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, volume 2018-May, pages 422–428. IEEE, may 2018.

[54] Ali Mesbah. Ajaxifying classic web applications. In *Proceedings - International Conference on Software Engineering*, pages 81–82, 2007.

[55] Ali Mesbah and Arie Van Deursen. Migrating multi-page web applications to single-page AJAX interfaces. In *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*, pages 181–190, 2007.

[56] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping. High-latency, low-bandwidth windowing in the Jupiter collaboration system. In *UIST (User Interface Software and Technology): Proceedings of the ACM Symposium*, pages 111–120, New York, New York, USA, 1995. ACM.

[57] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. Data consistency for P2P collaborative editing. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, pages 259–268, New York, New York, USA, 2006. ACM Press.

[58] João Pedro Sousa Castro. *Mobile application for online monitoring and collaboration of a sporting event.* PhD thesis, Faculdade de Engenharia da Universidade do Porto, sep 2020.

[59] Paul Resnick, Richard Zeckhauser, Eric Friedman, and Ko Kuwabara. Reputation systems. *Communications of the ACM*, 43(12):45–48, dec 2000.

[60] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6976 LNCS, pages 386–400. Springer, Berlin, Heidelberg, oct 2011.

[61] Leon Shklar and Rich Rosen. *Web Application Architecture: Principles, Protocols and Practices*. Wiley, Chichester, UK, 2 edition, 2009.

[62] Spero and S. E. Analysis of HTTP Performance problems. *http://www.w3.org/Protocols/HTTP/1.0/HTTPPerformance.html*, 1994.

[63] Chengzheng Sun, David Chen, Xiaohua Jia, Yanchun Zhang, and Yun Yang. Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems. *ACM Transactions on Computer-Human Interaction*, 5(1):63–108, mar 1998.

[64] Chengzheng Sun and Clarence Ellis. Operational transformation in real-time group editors. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work - CSCW '98*, pages 59–68, New York, New York, USA, 1998. Association for Computing Machinery (ACM).

[65] David Sun, Chengzheng Sun, Agustina Ng, and Weiwei Cai. Real Differences between OT and CRDT in Correctness and Complexity for Consistency Maintenance in Co-Editors. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW1):1–30, may 2020.

[66] David Sun, Chengzheng Sun, Steven Xia, and Haifeng Shen. Creative conflict resolution in realtime collaborative editing systems. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, pages 1411–1420, New York, New York, USA, 2012. ACM Press.

[67] Sai Lahari Velagapudi and Himanshu Gupta. Privacy, Security of Cookies in HTTP Transmission. In *2019 4th International Conference on Information Systems and Computer Networks, ISCON 2019*, pages 22–25. Institute of Electrical and Electronics Engineers Inc., nov 2019.

[68] Xueyi Wang, Jiajun Bu, and Chun Chen. Research on Conflict Resolution and Operation Consistency in Real-Time Collaborative Graphic Designing System. In *Proceedings of the International Conference on Computer Supported Cooperative Work in Design*, volume 7, pages 145–150, 2002.

[69] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot: A scalable optimistic replication algorithm for collaborative editing on P2P networks. In *Proceedings - International Conference on Distributed Computing Systems*, pages 404–412, 2009.

[70] Liyin Xue, Kang Zhang, and Chengzheng Sun. An integrated post-locking, multi-versioning, and transformation scheme for consistency maintenance in real-time group editors. In *Proceedings - 5th International Symposium on Autonomous Decentralized Systems, ISADS 2001*, pages 57–64. Institute of Electrical and Electronics Engineers Inc., 2001.

[71] Y. Yang. Supporting online Web-based teamwork in offline mobile mode too. In *Proceedings of the 1st International Conference on Web Information Systems Engineering, WISE 2000*, volume 1, pages 486–490. Institute of Electrical and Electronics Engineers Inc., 2000.

[72] Ekaterina Yashkina, Arseny Pinigin, Joo Young Lee, Manuel Mazzara, Akinlolu Solomon Adekotujo, Adam Zubair, and Luca Longo. Expressing Trust with Temporal Frequency of User Interaction in Online Communities. In *Advances in Intelligent Systems and Computing*, volume 926, pages 1133–1146. Springer Verlag, 2020.

[73] Marek Zawirski, Nuno Preguiça, Sérgio Duarte, Annette Bieniusa, Valter Balegas, and Marc Shapiro. Write fast, read in the past: Causal consistency for client-side applications. In *Middleware 2015 - Proceedings of the 16th Annual Middleware Conference*, pages 75–87. Association for Computing Machinery, Inc, nov 2015.