

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Title of the Dissertation

Ângelo Miguel Tenreiro Teixeira

PREPARAÇÃO DA DISSERTAÇÃO



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. João Correia Lopes

January 23, 2021

Title of the Dissertation

Ângelo Miguel Tenreiro Teixeira

Mestrado Integrado em Engenharia Informática e Computação

January 23, 2021

Abstract

TODO abstract

Keywords: keyword1, Keyword2, keyword3

Acknowledgements

TODO ACKNOLEGDEMENTS

Author

“Ceci n’est pas une citation.”

Ângelo Teixeira

Contents

1	Introduction	1
1.1	Offline Availability	1
1.2	Conflict Resolution	2
1.3	Reputation System	2
1.4	Summary	2
2	Background and Literature Review	3
2.1	Offline Availability	4
2.2	Conflict resolution	8
2.3	Reputation System	8
2.4	Similar platforms	12
2.5	Similar work	12
3	Problem Statement	15
3.1	Problem Definition	15
3.2	Proposed Solution	15
3.2.1	Reputation System	15
3.3	Methodology	17
3.4	Planning	17
4	Conclusions and Future Work	19
4.1	Expected Results	19
A	Loren Ipsum	21
	References	23

List of Figures

2.1	HTTP protocol example interactions	4
-----	--	---

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
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 [9]. 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 This tool exists and is somewhat outdated, hence the opportunity to build something better.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.

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 and a proposed solution will be stated in [Chapter ??](#).

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 the proposed solution can be found in Chapter ??

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 Chapter ??.

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 [37]. 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 [28] [21] [34] [33]. 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 are some

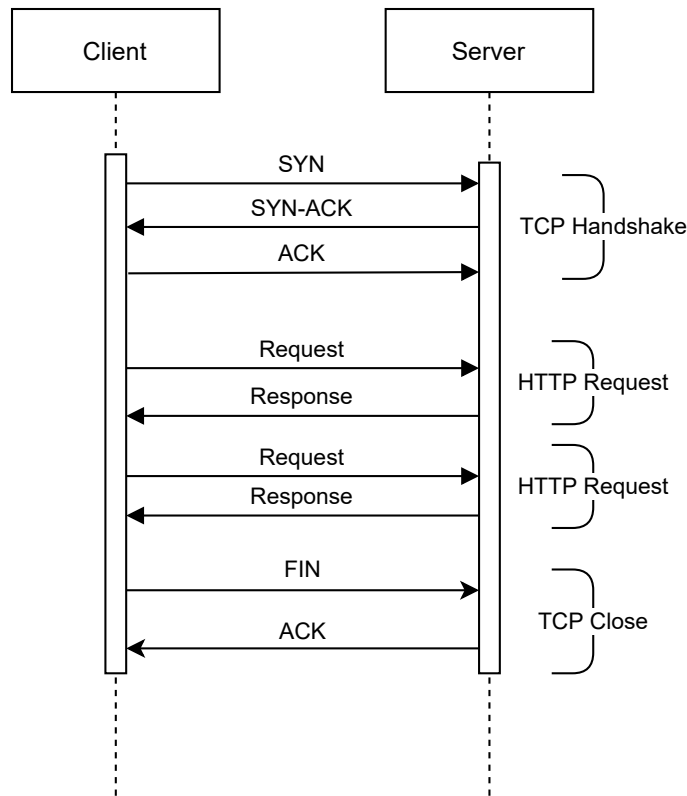


Figure 2.1: HTTP protocol example interactions

frameworks that allow this more recent architecture such as React [11], Vue [12] and Angular [10].

Since the web uses the HTTP Protocol [4] [7], it is quite hard to achieve some real-time behavior with good performance, due to the protocol's design [38]. 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 [5] 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 [?], 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 frus-

tration. 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, [31] 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 [29] 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 [14]. 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 [31] 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. [30], representing the web pages as nodes, with edges meaning navigation within 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 [40] 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. [25] 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 [?] - 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 [?] - with the same goal of CouchDB, but meant to run on the end user devices.

Also mentions CRDT (Conflict-Free Replicated Data Types) as a foundational technology to achieve the local-first ideals. CRDT are general-purpose data structures similar to the common lists and maps but built for multi-user environments from the beginning. CRDT merge changes

¹<https://www.google.com/docs/about/>

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 *Automerger*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. [42] 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. [24] 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 [27] 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 [6]). 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 [39] [26];

²<https://github.com/automerger/automerger>

- **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 [8];
- **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 [3] 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 [?]; For more space, **Indexed Database** is recommended.

Jemel and Serhrouchni [23] 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

TODO Creative conflict resolution in realtime collaborative editing systems
 TODO A Consensus-Driven Group Recommender System

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 [22] 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

³<https://www.npmjs.com/package/idb>

al. [15] 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 reachable 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 [20] 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 [16] 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 [19] 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 [13], 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 parallelisms 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. [36] 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. [32] present a dynamic interaction based reputation model (DIB-RM), which is further evaluated in [41]. 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 [18], 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 3.2.1.

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 [2]. 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 [35] 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

problem section intro here

3.1 Problem Definition

TODO US here

3.2 Proposed Solution

TODO arch + techs here mockups?

3.2.1 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 [18], 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. 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;

$$newInputRep = oldInputRep + (1 - oldInputRep) * maxRepReward * userRep * userRepInfluence \quad (3.1)$$

- 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;

$$newInputRep = oldInputRep * (1 - maxRepPunishment * userRep * userRepInfluence) \quad (3.2)$$

- 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;

$$newInputRep = oldInputRep + ((1 - oldInputRep) * repReward * userRep * userRepInfluence) * (1 + explicitAgreementBonus) \quad (3.3)$$

- 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;

$$newUserRep = oldUserRep + (1 - oldUserRep) * \frac{\sum inputRep}{numInputs} \quad (3.4)$$

- 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)

$$newUserRep = oldUserRep * decayCoefficient^{timeSinceLastUpdate} \quad (3.5)$$

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

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 [17]. The most common are Scrum, XP and Kanban.

3.4 Planning

TODO GANTT here

Chapter 4

Conclusions and Future Work

conclusions intro

4.1 Expected Results

something else?

Appendix A

Loren Ipsum

References

- [1] 365scores. <https://www.365scores.com/pt/pages/about>.
- [2] MyCujoo. <https://mycujoo.tv/en/about-us>.
- [3] Promise API JS MDN. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.
- [4] RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1. <https://tools.ietf.org/html/rfc2616>.
- [5] RFC 6455 - The WebSocket Protocol. <https://tools.ietf.org/html/rfc6455>.
- [6] RFC 6455 - The WebSocket Protocol. <https://tools.ietf.org/html/rfc6455>.
- [7] RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2). <https://tools.ietf.org/html/rfc7540>.
- [8] Web SQL Database W3C. <https://www.w3.org/TR/webdatabase/>.
- [9] 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.
- [10] Angular. <https://angular.io/>, 2021.
- [11] React. <https://reactjs.org/>, 2021.
- [12] Vue. <https://vuejs.org/>, 2021.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] Yannis Bakos and Chrysanthos Dellarocas. Cooperation Without Enforcement? A comparative analysis of litigation and online reputation as quality assurance mechanisms. Technical report, 2003.

- [17] 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.
- [18] 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.
- [19] Chrysanthos Dellarocas. How often should reputation mechanisms update a trader’s reputation profile? *Information Systems Research*, 17(3):271–285, 2006.
- [20] Chrysanthos Dellarocas and R. Smith. Reputation Mechanisms. 2005.
- [21] 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.
- [22] 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.
- [23] 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.
- [24] Yung Wei Kao, Chiafeng Lin, Kuei An Yang, and Shyan Ming Yuan. A web-based, offlineable, and personalized runtime environment for executing applications on mobile devices. *Computer Standards and Interfaces*, 34(1):212–224, jan 2012.
- [25] 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.
- [26] 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.
- [27] Wen Tao Liu. Research on offline storage of web page. In *Applied Mechanics and Materials*, volume 518, pages 305–309, 2014.
- [28] 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.
- [29] Félix Albertos Marco. Supporting offline interaction with web sites resilient to interruptions applied to E-learning environments. In *Lecture Notes in Computer Science (including sub-series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8295 LNCS, pages 310–314, 2013.

- [30] Félix Albertos Marco, José Gallud, Víctor 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.
- [31] 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.
- [32] 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.
- [33] Ali Mesbah. Ajaxifying classic web applications. In *Proceedings - International Conference on Software Engineering*, pages 81–82, 2007.
- [34] 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.
- [35] 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.
- [36] Paul Resnick, Richard Zeckhauser, Eric Friedman, and Ko Kuwabara. Reputation systems. *Communications of the ACM*, 43(12):45–48, dec 2000.
- [37] Leon Shklar and Rich Rosen. *Web Application Architecture: Principles, Protocols and Practices*. Wiley, Chichester, UK, 2 edition, 2009.
- [38] Spero and S. E. Analysis of HTTP Performance problems. <http://www.w3.org/Protocols/HTTP/1.0/HTTPPerformance.html>, 1994.
- [39] 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.
- [40] 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.
- [41] 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.
- [42] 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.