Report
Online Ephemeral Decentralized Chat

Romain Dulout   Pierre-Malo Gaudichau   Timothée Janvier
Achraf Maksi   Hamza Mouftih   Guillaume Obin   Anas Quazbary

May 2021

Contents

1 Introduction

Nowadays, the web is strongly centralized due to the **Tech Giants** (Google, Amazon, Facebook, Apple and Microsoft also known as GAFAM) dominating the global Internet. Thanks to their exceptional market capitalization which makes them the most valuable public companies globally, they eliminate smaller tech companies, avoid their contributory obligations and compete with states. Moreover, they have absolute control of their users. Over the past 20 years, these structures have penetrated billions of people's lives, knowing their habits, their families and friends, the way they consume and their tendency to be influenced. They currently own over one zettabyte of private user data (a thousand billion of billions of bytes). Even if the **Big Five** try to reduce their ecological impact in a few years, the environmental disaster they produce persists. Recently, they directly attacked democracy, censoring political opponents, activists and everyone who does not conform to their way of thinking.

The Internet, as a space where everyone has absolute freedom, is a concept that belongs to the past. Nevertheless, this trend seems to be reversed with the upcoming of new technologies based on decentralized models which place the user as a participant (client and server), compared to GAFAM models which allow the user to be merely a customer. For example, cryptocurrency based on blockchain algorithms is heard of more and more, with Elon Musk and his Dogecoin, to get around the traditional bank system. Peer-to-peer models, also known as *serverless* architectures, minimize maintenance and service deployment costs in comparison to the centralized model.

For all these reasons, this project focuses on the creation of a chat room, where censorship could not be applied, democracy is preserved, security and integrity of user data are guaranteed and environmental concerns are respected. The use of modern and innovative web technologies aside, one of the principal challenges of the decentralized chat application presents itself in the implementation of data updating user methods, to palliate to desynchronizations between users. CRDTs (conflict-free replicated data type), for example used in Google Docs, offer the possibility to replicate independently data structure for each user. Thanks to the implementation's properties and its properties, CRDTs are the perfect solution to create a peer-to-peer chat.

In accordance with customer requests, during this semester, a web based ephemeral and decentralised instant messaging application has been created, using CRDTs' technology. This application is called PEAR-CHAT in reference to peer-to-peer models (and Apple's design).

2 CRDT: Confilct-free replication data type

CRDT stands for Conflict-free Replicated Data Type. The CRDT approach allows synchronization in terms of the underlying data structure. A CRDT, at a high level, is a type of object that can be merged with any objects of the same type in arbitrary order, to produce an identical union object. CRDT merge must be associative ($a\times(b\times c) = (a\times b)\times c$), commutative ($a \times b = b \times a$), and idempotent ($a \times a = a$). The resulting CRDT for each mutation or merge must be *greater* than all its inputs. Mathematically, this flow is said to form a monotonic *semilattice*. As long as each connected peer eventually receives the updates of every other peer, the results will probably converge even if one peer happens to be a month behind. There are two major types of CRDTs at play: state-based CRDTs and operation-based CRDTs.

2.1 State-based CRDTs

Also known as convergent replicated data types (CvRDTs), they send their full local state to other replicas, where the states are merged by a merging function. There are two basic components of a CvRDT that need to be kept in mind. The first component being the state component which allows all the possible states to be represented as elements of a set. For a CvRDT, that set must be ordered by some binary relation. The whole point of CvRDTs is to replicate state across nodes. A merge function is needed in order to ultimately keep that state in sync.

The key point is that the states in a System converge toward the Value of the System as the merge occurs. This ensures the following properties. First, the order of merges does not matter. This is guaranteed by the associativity and commutativity of joins. Then, it does not matter how many times a particular merge is repeated. This is guaranteed by the idempotence of joins.

For example, to implement a simple CRDT 3 vectors of integers are used: $X = (3, 0, 0), Y = (0, 2, 0), Z = (0, 0, 1)$. Each element in the vector corresponds to a node. Each node gradually picks up the latest value for the other node, compares it to the value it already has and keeps the max. The possible operations are defined below.

- **increment**(): Increments the integer at the vector index corresponding to this node.
- **value**(): Sums all integers in the vector.
- **merge**(*incoming_state*): Replace the local state with the maximum between the local state and the incoming state.

So, with this structure, the result will eventually converge towards the state $(3, 2, 1)$. From a practical standpoint, this could be implemented using a *G-counter* which is a CRDT.

2.2 Operation-based CRDTs

Operation-based CRDTs are also called commutative replicated data types, or CmRDTs. Unlike the CvRDTs, CmRDT replicas propagate state by transmitting only the update operation and not the full local state. Once all replicas receive and execute all update operations, they eventually converge to a single state as long as concurrent update operations commute. The following algorithm describes how these data types function. When an update operation o is issued at some node i having state σ_i, a message m is prepared. This message m is then broadcast to all other j-node. A new state replica σ'_j is then created. Broadcast event, corresponding local delivery, and treatment made on local state are all executed atomically.

3 Global project architecture

The project is centered around the idea of *decentralization* and *ephemeral*. The latter is not particularly difficult to implement. Ephemeral means that nothing is kept and if data are to be processed, they are stored locally, not shared with anyone else and destroyed when it is done. For the decentralized aspect of the project, it was harder than expected. A decentralized architecture means that there is no central server and that every single user is a node on the network. The whole user group forms a mesh grid.

Due to the online nature of the project, a programming language dedicated to online interactions, lightweight enough to run on any computer on the market and portable is necessary. JavaScript is the best language in this case. To run JavaScript scripts on a computer, a *server* is needed (which serves as an *engine* for scripts). NodeJS is perfect for this usage.

The other requirement for this project is that it needs to run in a web browser. For this specific case, a library made by Google is available for decentralized architectures, it is called WebRTC. WebRTC is a protocol that allows *peer-to-peer* interactions between two (or more) end-users in a browser. Unlike in a regular JavaScript application, which can allow communications between peers with IP addresses and ports in a terminal session, in a web browser, WebRTC is the only protocol that can achieve this objective.

Secondly, the project needs to have a *foreground* user-interface. The front-end also needs to be run in a web browser. This means that JavaScript is still an adapted language for this use case. However, the engine that runs the scripts is totally different. React is the most adapted engine for user-interface development.

Unlike NodeJS, React has the ability to compile JavaScript scripts for them to be handled by web browsers as front-end content. Web browsers use a different language for front-end applications. It is called HTML (HyperText Markup Language) and it is very different from JavaScript. It is a markup language made to be displayed in a web browser. It can be assisted by CSS (Cascading Style Sheets) for styling the web page. React transforms JavaScript scripts into an exploitable HTML page.

React works in a different way than regular JavaScript. It is composed of components (or modules) that can be joined together to create a web page. These components are usually pre-written blocks of HTML code. There are many libraries that can either replace these components or complement them for styling purposes.

These two parts need to run on two separate servers. However, due to the nature of WebRTC protocol, it is impossible to make a 100 percent-decentralized architecture. There is always going to be a centralizing point at a moment (which means a third server).

Finally, the final application works the following way [1]. The end-user connects to the front-end server (React). Requests to the back-end (NodeJS) are handled by the front-end server and displayed on the web page. In this way, the end-user will never notice the interactions.

4 Back-end functionalities

The back-end works the following way: a server is executed in background and contains all the scripts that can run the application. NodeJS is the engine used for this purpose. A package dedicated to background services is added to the server. This package is Express.

Express is a *fast, unopinionated, minimalist web framework for NodeJS*[2]. It has many useful features. One of them is the ability to create routes. It defines the way the application needs to respond to a client request to a particular endpoint. These *hooks* can be triggered through a defined URL. In the project, a server browser is required for the end-user to find chat rooms to chat with other users.

To store information regarding the server browser, in this case the list of available chat rooms, a database needs to be set up. MongoDB has been chosen as the most suitable database engine for this project. It has the ability to store data under document-type format. It is easier to retrieve information with NodeJS.

To couple MongoDB with NodeJS, another library is required to make the process easier. This library is Mongoose. Mongoose has the ability to connect NodeJS to a premade MongoDB database, to create document models for the database and to make requests to the database.

For the project's server browser, HTTP request methods are necessary. Four methods are used in Express routes. GET allows the end-user to retrieve the information stored in the database. It can be a specific element or the whole collection. POST allows the end-user to store the information in the database. Whenever data is pushed, a new document is created with a random ID that allows it to be easily identified. PATCH allows the end-user to modify a document which is already present in the database.

[1]See **Figure 1** in annex for better explaination.

[2]`https://expressjs.com/`

DELETE allows the end-user to delete a specific document in the database.[3]

Express also allows a public repository to be created which can be accessed with standard URLs. This is used to store JavaScript scripts and embed them into a web page. For the chat service, two packages are used. The first one is called WebRTC-Swarm. It allows the interconnection between users and it defines the behavior regarding data transmission and reception. WebRTC-Swarm, as recommended by the WebRTC protocol, works with “signaling”. To complete this step, the second package used is SignalHub.

SignalHub works in a very specific way. It needs to be run on a separate server. WebRTC-Swarm then makes the calls by reaching SignalHub server. Signaling is a way to notify that someone wants to connect with someone else. This is done by the generation of a unique ID. SignalHub stores IDs of users that are ready to receive connection requests. IDs are then removed from the list once users are disconnected.

At this point, the code is still not usable in a web browser. The reason is that, when the script is loaded, it makes a series of imports from external modules. These modules are present in the server. However, this code is executed on the client-side, which means into the web browser. These modules are not accessible from the web browser. A treatment is required before going further.

This can be done with a small program called Browserify. This allows JavaScript scripts that are using external modules to be executed into a web browser. What it does is simple. It takes all the dependencies from a script, writes all the functions that are called in the script in the header of a new file and puts the script code at the bottom of this file. This new generated file can then be executed properly into a web browser.

Once generated, the script is dropped into Express’ public folder to make it visible and accessible directly from a web browser. The script is then called from the front-end server. There is a script markup in HTML language for this purpose. The script URL is simply referenced into this markup within the dedicated chat page. To dynamically change elements on the page directly from the script, dedicated markups are declared and tagged with specific unique IDs. The script then updates the information within these markups when it is necessary.

Finally, to implement CRDTs functionalities to the chat, a library called Automerge is used for this purpose. Automerge grants the ability to create a document which stores a message history locally for each user. Once all users are disconnected, the history is destroyed for all of them. In this case, messages are sent back by connected users when a user has a desynchronization or a disconnection. It uses an operation-based approach by only sending the updated state. A sequence number is also added to sort messages

[3]See **Figure 2** for an overview.

properly.

5 Front-end functionalities

JavaScript is the language used to code the interface with a framework called React. It is a well-known and useful JavaScript framework. It has great reactivity and intelligibility.

The app is dynamic. It is not an *on-demand* web page. The content can vary depending on the user's actions like text entry, date and more. Whereas in an *on-demand* web page, in order to modify the content of a web page, calls have to be made to the server. JavaScript combined with React allows information storing and ordering. The app requires a user to login. It then allows the user to create chat rooms, view all available chat rooms and join them.

A React web app works with components. Five main components have been designed.

- **App** is the central component that makes redirection to other components.
- **Login** allows the user to register to the app with a nickname.
- **NewChatRoom** enables the end-user to create a room, choose the privacy and the visibilty. A password can also be set up.
- **JoinChatRoom** grants the ability to browse for visible and available rooms and to join them.
- **Room** permits the user to navigate through joined rooms, view different conversations and send messages.

All these components are linked via page URLs with React Routers. Routers also store information like chat rooms parameters, user name, password and send them to the upper component (**App** in this case).

Once a user is registered, a new room can be created. The end-user can also join an existing room. To do this, a page that lists all existing rooms can be used. If the number of rooms is too large, a by-name filter is available.

6 Project management

The project is punctuated by a predefined organization and managed through a GANTT diagram and a user-story system to meet customer's expectations. Rules of conduct have been devised as well as monitoring indicators, task distribution and division into sub-teams.

6.1 WBS / OBS: explanation of tasks

First, a list of the tasks to be accomplished is made during a *brainstorming* session. Then, these are divided according to the WBS (Work Breakdown Structure) which is a deliverable-oriented break down of a project into smaller components. This simplifies the project by offering a better role organization to achieve OBS (Organizational Breakdown Structure) which is used to define the responsibilities for project management, cost reporting, billing, budgeting and project control. Following this step, it is possible to produce a PERT diagram (Program Evaluation Review Technique), which allows an estimation of duration for the deliverable and the history of completion. Thanks to this, the critical path, which represents the optimal way to carry out the project, has been defined. These different preliminary steps are necessary for the construction of the GANTT diagram which illustrates the overall project schedule. It is used for daily monitoring purposes.

6.2 Planning: progress of tasks and milestones

In order to follow the progress of the project, a GANTT diagram was built to maintain transparency and ensure it is up-to-date. This allowed, among other things, to ensure deadline completion and mobilization if a task is likely to fall behind.

Following the separation of deliverables at the start of the project, a division of the project into two sub-teams of two members and one of three was planned. One takes care of the design of back-end functions, the other group handles the front-end functions. Finally, the last team is responsible for linking the work carried out by the two teams to deliver a functional web application. There is now an important framework upon which the various tasks are carried out, but there was still a need to layout specific steps.

6.3 Communication with the customer

The AGILE project management focuses on delivering maximum value against business priorities in the time and budget allowed, especially when the drive to deliver is greater than the risk. It is founded on 4 principles: breaking requirements into smaller pieces, collaborative working, customer-oriented approach and the integration of planning and execution. In the spirit of this management approach, the client is involved from start to finish in the project through the use of a repeated incremental process. It takes into account the changes of needs. It allows, through exchanges, to consider setbacks and possible difficulties in the planning. In order to facilitate this process, a user-story system was created to illustrate key steps. User-stories correspond to use cases from the customer's point of view for which the team must find suitable technical solutions. These are divided into two main aspects, the back-end and the front-end (detailed previously). In addition, at each major breakthrough, a meeting with the client is set up to check that the team is moving in the right direction.

7 Conclusion

To answer the initial problem "creating an online ephemeral and decentralized chat using CRDT", the project was divided into several objectives. First, a phase of discovery of modern web technologies such as WebRTC, Javascript, NodeJs, React and Browserify has been put into place as a first step to develop the decentralized web application. Then, a study of the CRDTs algorithm led to choosing Automerge as the framework for the data chat structure. Thereafter, the implementation phase of the project took place. Finally, the main focus was on cryptographic techniques, bug fixes and application deployment.

Learning new programming languages was not a problem for the team, however selecting the technologies which fit best with requirements was quite difficult. For instance, the backend library was switched a few times in order to facilitate the use of peer-to-peer protocol. Moreover, the complete decentralization of the app was a difficult chore. How is it possible to share and access the chat in a fully decentralized manner? Should the application only be transmitted by email? So as to make the application accessible to as many people as possible, the decision to centralize the access and distribution of the application was made, all while leaving the chat operation completely decentralized.

Finally, in addition to the progress in the team's technical engineering skills, this project was an opportunity for the members to work on their interpersonal skills and on group dynamics, which are essential to the project management, such as team spirit, coordination and communication. A particular attention has been paid to carrying, listening and respecting values so that each member of the team may be at the heart of the project. This kind setting allowed serene work on this project. Individually, this project allowed the team members to apply their respective strengths such as curiosity, autonomy and creativity, to play their part in the PEAR-CHAT project.

8 Annex

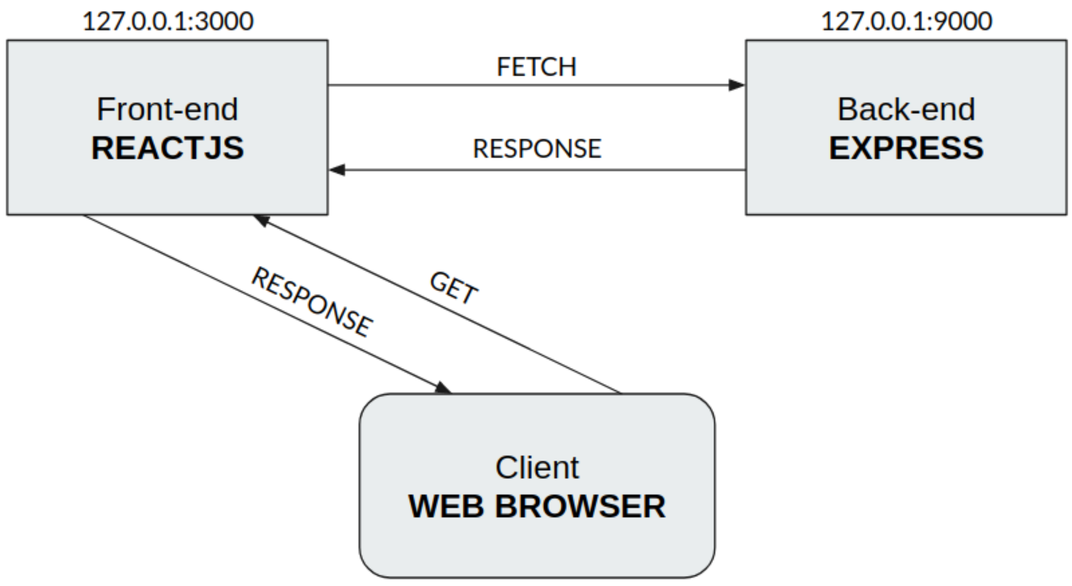

Figure 1: Global project architecure

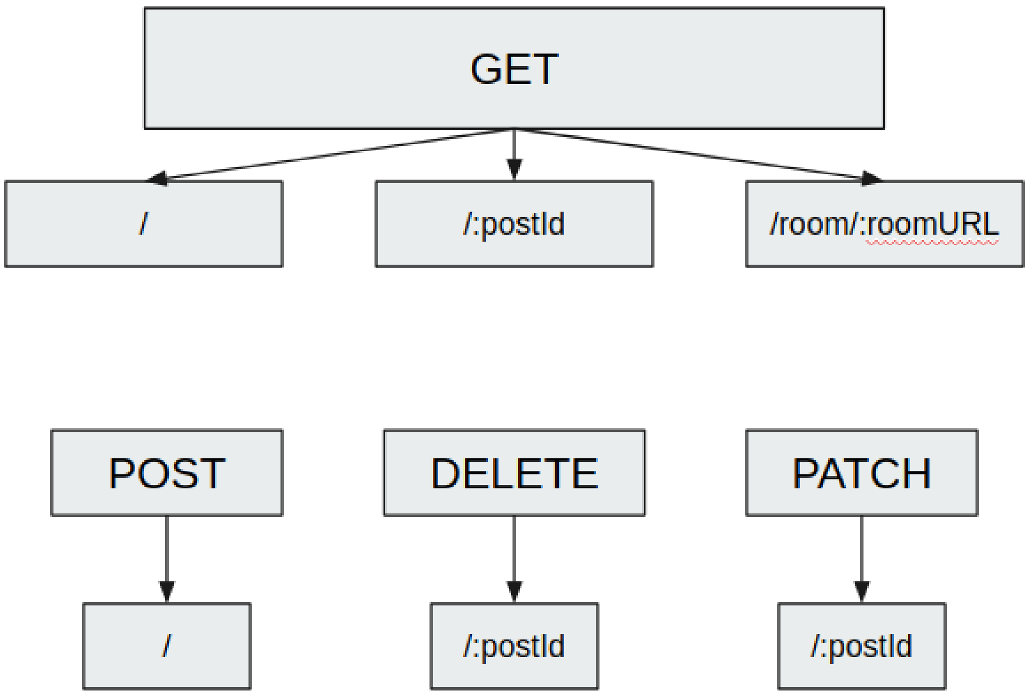

Figure 2: Back-end functionalities