Decentralized Timeline

Large Scale Distributed Systems
G14, FEUP, 2022

Emanuel Trigo 201605389
Fábio Huang 201806829
Sara Pereira 202204189
Valentina Wu 201907483

Table of contents

01

Requirements

Problem description and
project “Rules”

Requirements

Goal: Decentralized timeline service.

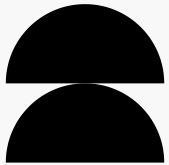
Each user:

a. Has an identity;

b. Publishes and deletes small text messages in their local machine;

c. Subscribes other user's timelines and accesses their messages;

d. Helps to store and forward subscribed users' timelines.



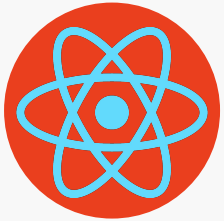
02 Technology

Frameworks and Packages

Technology used

Gun

Decentralized graph database

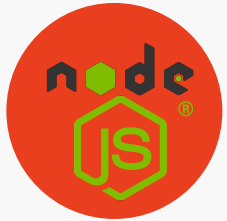


React

JavaScript library for building user interfaces

WebRTC

Real-time communication for the web

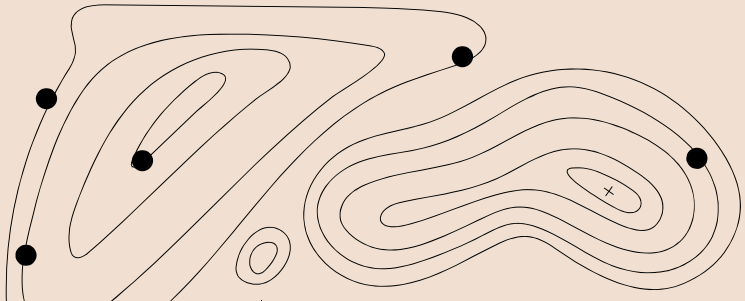


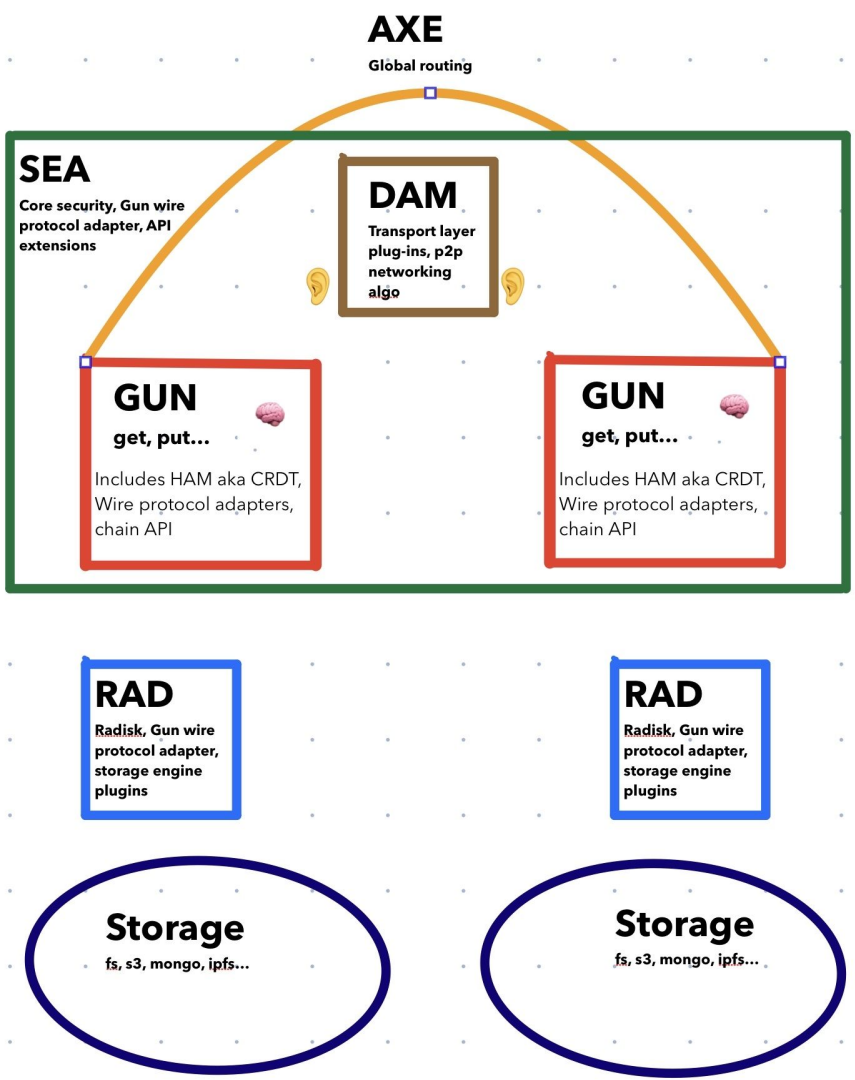
Node.js

Open source server environment

GUN 03

How Gun works and Graph Structure

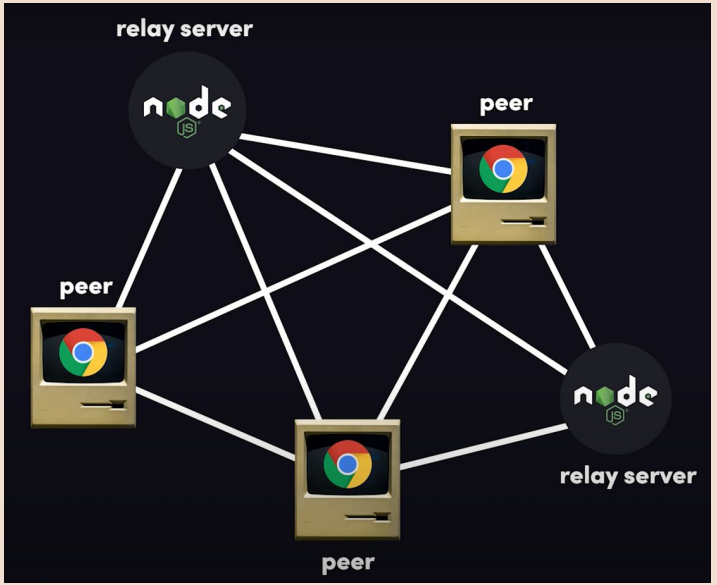




Gun Ecosystem

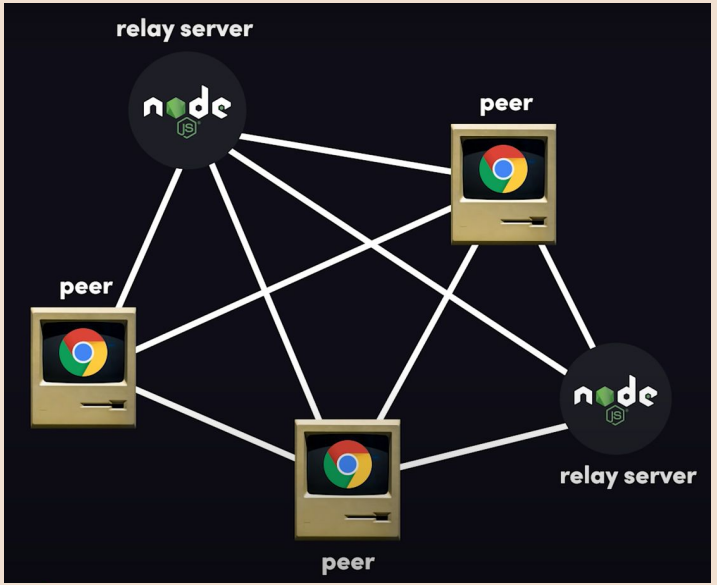
Gun

- **Decentralized graph** database.
- Data is distributed across **multiple peers** (or users) through WebRTC.
- Each peer stores the data it needs in the browser's **local storage.**



Gun

- And then **sync changes** with the other peers in the network.
- A peer can be a **user** or a **relay server** that makes the system more reliable.
- A relay server allows **persistent storage** and makes the network **more robust.**

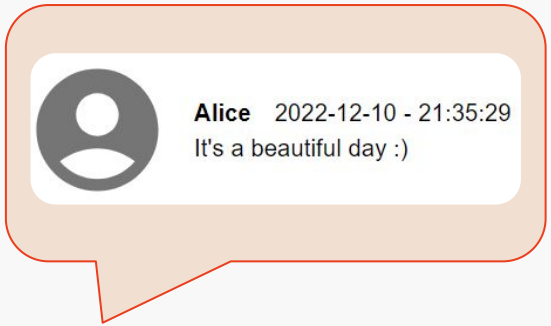


Gun - Graph Structure

Node

Each node can reference another node allowing circular dependencies.

The “Data” field in this project consists in the **followed** users and the posts **timeline** of each user.



A post consists in a **message**, a **date** and an **author.**

Gun - Database

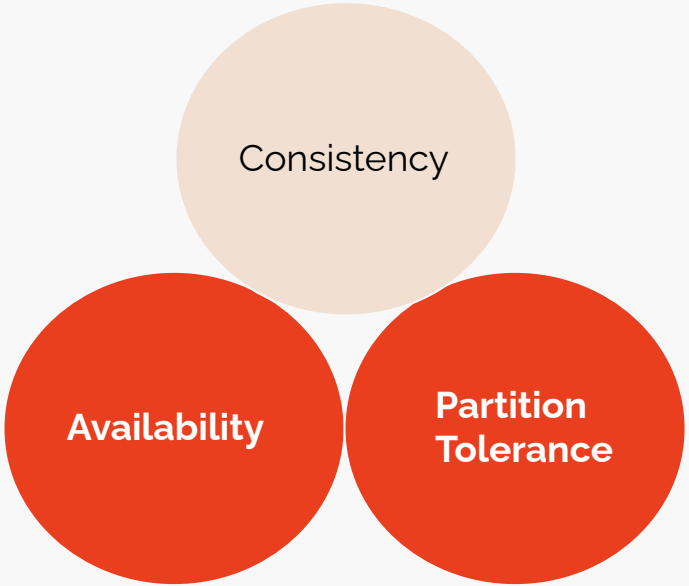
Users

User -> publicKey

```
Timeline: {
        key1: post1
        key2: post2
            ...
}
```

```
Follows: {
        userName1: publicKey1
        userName2: publicKey2
            ...
}
```

Gun - CAP Theorem

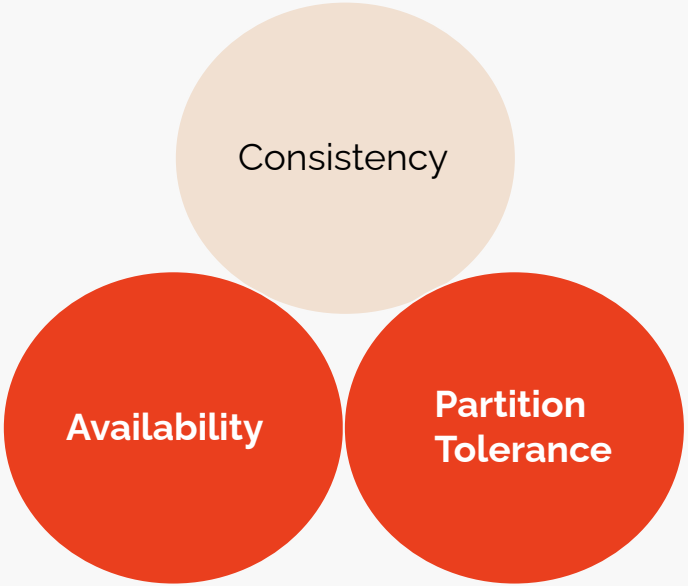


Users can find a **replica of data** even in case of partial node failures.

Peers can successfully **read and write** data even if other peers are **offline**.

GUN is **AP** with eventual consistency.

Gun - CAP Theorem



Regarding consistency, GUN opts for **eventual consistency** instead of strong consistency.

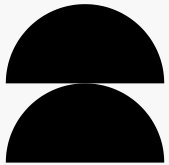
When connectivity becomes available again, GUN will synchronize data at **low latency** using at least once delivery to guarantee that all peers will deterministically **converge to the same data within a time frame**, without any extra coordination or gossip.

GUN is **AP** with eventual consistency.

Gun - CRDT

- The **conflict resolution algorithm** is at the center of everything gun does: it's how peers eventually arrive at the same state, and how offline edits are merged.
- Because graphs only contain nodes, and nodes only contain key-value pairs our goal is to merge **key-value pairs**.
- Choosing which key-value to keep requires an extra bit of metadata, called **state**.
- State is used to determine ordering of updates, and is always relative to the machine which receives it.

```
{
    "name": {
        "value": "Alice",
        "state": 10
    }
}
```



```
{
    "name": {
        "value": "Allison",
        "state": 8
    }
}
```

Gun - CRDT

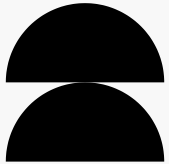
And if we get an update with the **same state** as us, but with a conflicting value?

```
{
    "name": {
        "value": "Alice",
        "state": 10
    }
}
```

```
{
    "name": {
        "value": "Allison",
        "state": 10
    }
}
```

This conflict is resolved by comparing their **string** values with JSON.stringify, choosing the **greater** of the two.

Lexical sort is only used if there is a conflict on the exact **same value** at the exact **same time**.



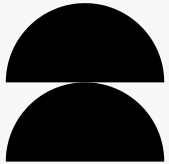
Gun - CRDT

How do we prevent a devious user from submitting an update with a state of 10 zillion?

```
{
    "name": {
        "value": "Alice",
        "state": 10
    }
}
```

```
{
    "name": {
        "value": "Allison",
        "state": 10000…
    }
}
```

- The solution is to simply wait until your machine reaches the state of 10 zillion before acknowledging the update.
- If an update isn't acknowledged, it never escapes volatile memory onto disk.



Gun - DAM

DAM is Gun's default **transport layer** abstraction and **P2P networking** algorithm.

How it works:

-> GET requests out to all peers it is connected

-> each peer receives that request, checks the message ID and deduplicates against it (if it has seen it before, it ignores it)

-> if it hasn't seen it before:

- it rebroadcasts out the message to all of its peers
- processes the message by seeing if it has the requested data that it can ACK back

DAM also adds the IP/url/ID of the peers the current peer is connected to to each message

Functionalities

Authentication, Posting and Following

04

Authentication

Regarding the authentication, we need to instantiate ***Gun*** and then reference ***user()***

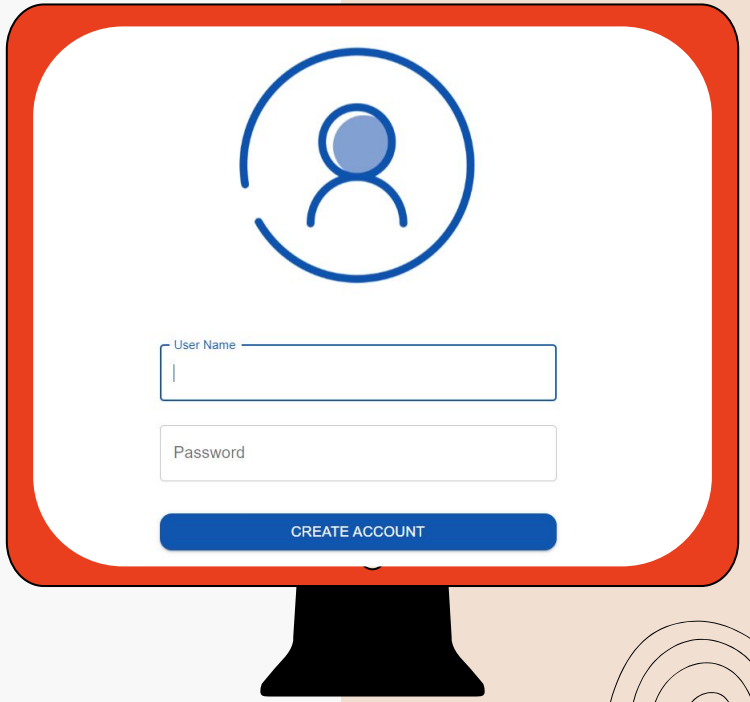
```
const gun = Gun();
const user = gun.user().recall({ sessionStorage: true });
```

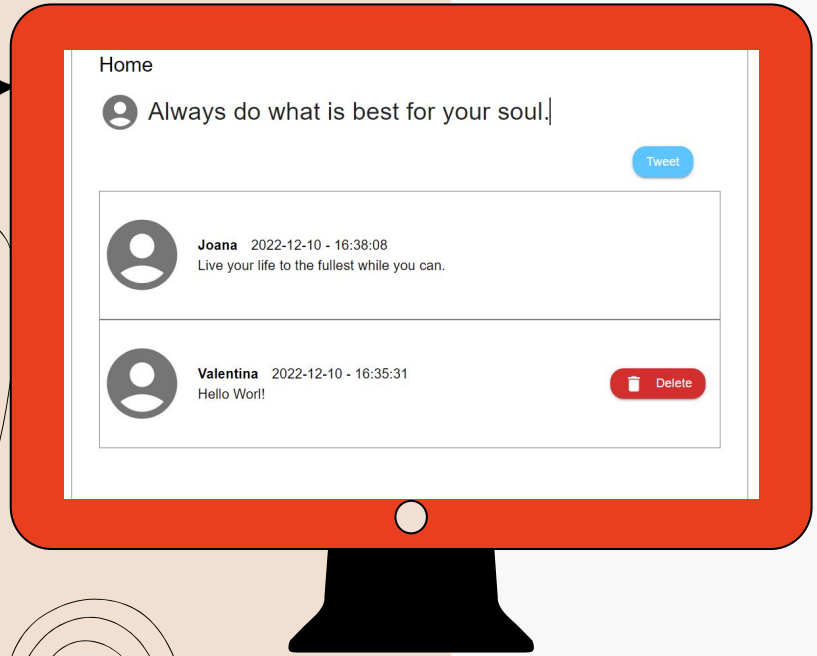
Create Account

When you create a new user, a cryptographic secure key pair is generated

The **username** is associated with the **public key**, so messages can be found

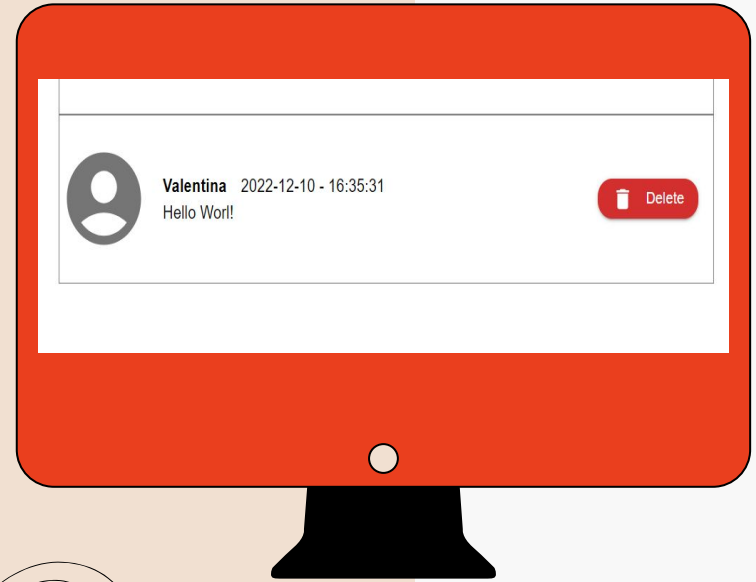
The **password** is a proof of work seed used to decrypt access to the account's **private key**





Post

When a user posts a message, the data is **saved** into gun and **synchronized** with the connected peers.

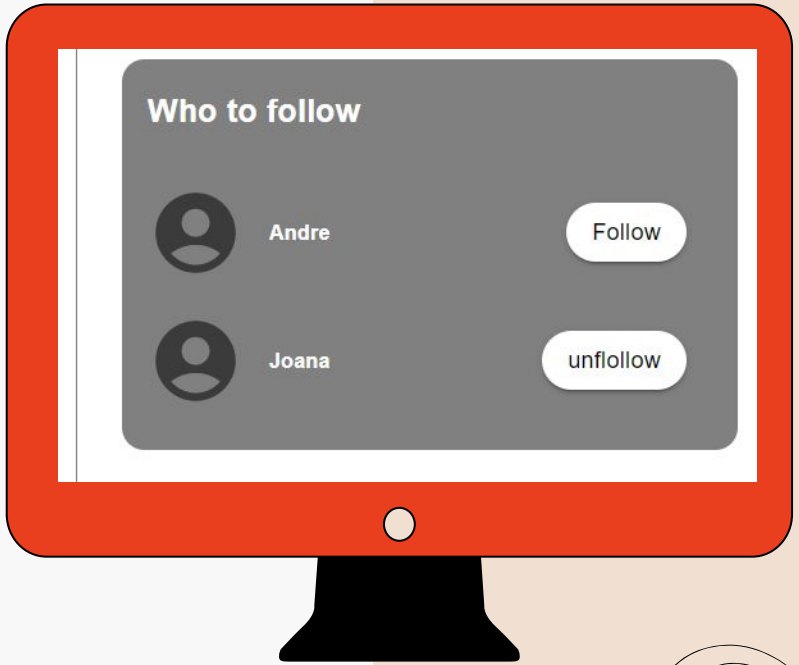


Delete Post

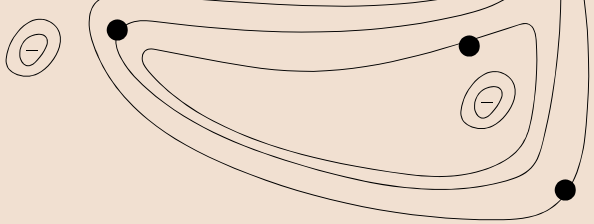
When a user deletes a post, the **reference** to the node's message becomes **null**.

Follow and Unfollow

Following an user, consists in adding a new entry (with **userName**) to the node's **Follows reference**.
To unfollow, that reference becomes **null**.

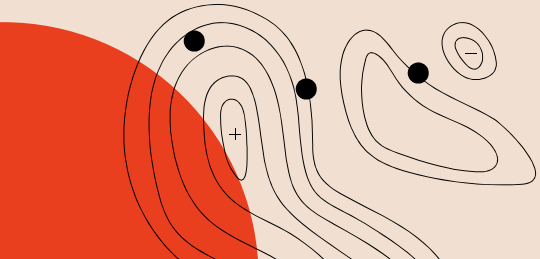
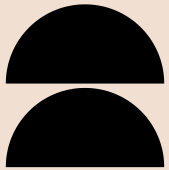
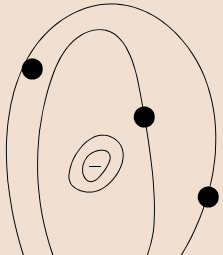






The messages from subscribed sources are **ephemeral** and only stored and forwarded for a given **time period**.

This is a consequence of the **browser's local storage** (up to 5MB) and GUN's work.

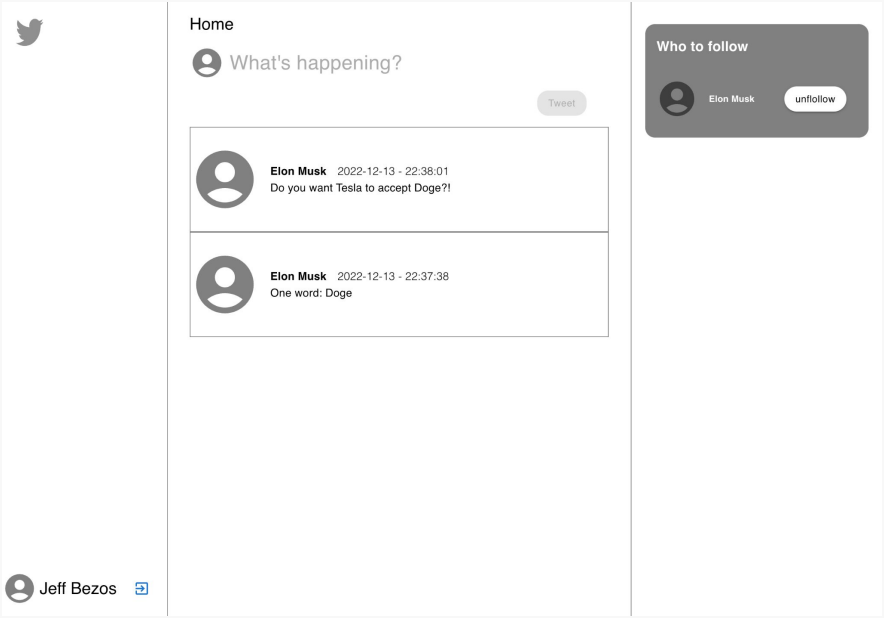




05

Demo

Project demonstration and interfaces

Demo



https://www.youtube.com/watch?v=vnDk2BjTk3Y

06

Limitations

Limitations and Future Work

Limitations and Future Work

- Avoid relay servers, and only use “normal” peers.
- Storage of posts and timelines in a persistent way (in the disc) in all the peers.
- Use SEA - Security, Encryption, & Authorization - API to encrypt data.

Thanks!

Do you have any questions?