Designing and Implementing Multi-user Distributed Text Editor

# Introduction

It is required to design and implement a multi-user distributed text editor, which will allow several users to collaborate in reading and editing text documents in a real-time environment where changes at any user will be broadcasted to all the other users at the same exact time of the changes. It is more like we need to design and implement a clone of Google Documents application.

# System Design

There are several functional and non-functional requirements that we must follow in order to start our system design plan.

## Functional Requirements

* Users can change a document at the same time without any conflict
* Allow sharing documents between users through a unique document ID and hyper-link

## Non-Functional Requirements

* The system must support multiple clients or autonomous agents like an API for sharing and updating data.
* The system should be distributed across multiple clients or server nodes.
* The system should be robust
* The system should be able to continue operation even if one of the participant nodes crashes
  + If three clients are collaborating on the same document and one client failed, the other two should continue collaboration on the system achieving reliability.
* It should be possible to recover the state of the node following a crash, to continue operation
  + Retrieving the final state of the document when the node goes back online
* The system should maintain multiple replicas for fault tolerance.

## Design Constraints

* Concurrency
  + Since several users are working on the same document
  + Operational Transformation
* Latency
  + Clients are working in different places, and the connection is established through the internet, so there is a latency between each and all clients when they are collaborating on the same document.
  + RESTful vs Publish/Subscribe Architectures.
* Security
* For each type of user, what type of data access restrictions are required.
* For each type of user, what type of update privileges are required.
* For each type of user are there any other Window behaviors which require specific privilege.

* Backup and Recovery Requirments
* *Acceptable down time for system.*
* *Acceptable data and user interface state loss due to system crash.*

## Operational Transformation

In order to provide real-time and collaborative environment in a text editor, we must consider any conflicts that may arise when more than two nodes are collaborating. For example, if a node inserted some text at position x, and another node deleted the text that exists at position x at the same time. Here, we present Operational Transformation, which is a technology that aims to solve conflicts in real-time collaborative editing environments. In order to do that, we must maintain consistency between local replicas of documents, since each client have its own local copy of the document.

A document will be stored as a sequence of operations in order of execution instead of plain text. So, we need a collaboration protocol to understand when to apply changes. We thought about identifying possible operations into three types:

* Insert Text
* Delete Text

Whenever we edit a document, all the changes are appended to the document saving these operations in one of those three types. In addition to saving operations by each user in a changelog database.

## WebSocket vs HTTP

1. Web Socket

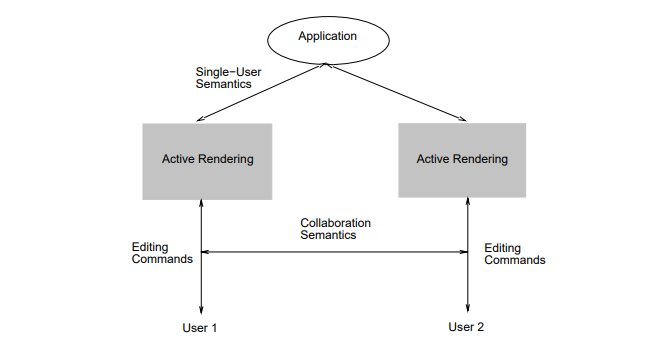
* WebSockets is a technology that enables bidirectional, full-duplex communication between client and server over a persistent, single-socket connection. This allows for low-latency, realtime updates, and the creation of richer communication and gaming applications. Previously, the web was dependent on requests and responses, which aren’t dynamic enough for those kinds of apps.
* WebSockets generally do not use XMLHttpRequest, and as such, headers are not sent every-time we need to get more information from the server. This, in turn, reduces the expensive data loads being sent to the server.
* WebSocket is an event-driven protocol, which means you can actually use it for truly realtime communication. Unlike HTTP, where you have to constantly request updates, with websockets, updates are sent immediately when they are available.

1. Web Transport

* The newer WebTransport offers secure, multiplexed, realtime transport and already has APIs for sending data both reliably and unreliably. In a reliable data transfer, the sender is notified of the success or failure of the data transmission, and failed transmissions are usually resent until they succeed, after which the next data packet is sent. In unreliable transfer, there’s no confirmation of transmission success, and packets that aren’t received simply don’t get delivered.
* Unreliable transfer is often used for things like streaming videos, where speed is a concern, and minor data loss, such as a few frames of video, is acceptable. Because WebTransport uses both of these methods, there are many use cases for it, such as bidirectional data streaming for multiplayer gaming, interactive live streams, and data transfer for sensors and internet of things devices.
* It avoids the head-of-line blocking delays that WebSockets suffers from, and is less resource intensive when creating connections
* It works with HTTP/3, the upcoming version of the transport protocol used by the World Wide Web. HTTP/3 uses the QUIC protocol for transport layer data exchange, which has several advantages. QUIC can prevent head-of-line blocking delays, improving network performance in many situations. This is a limitation of WebSockets.

# RESTful vs Event-Driven Architectures

# General architecture for collaborative editing applications



# 1st: RESTful Architecture (representational state transfer)

## Characteristics

REST principles are defined by four interface controls, including identifying resources, managing resources through representations, self-descriptive communications, and hypermedia as the engine of the application state.

View the distributed system as a collection of resources, individually managed by components and these resources can be added, removed, retrieved or modified by remote applications while keeping that these resources provide the same interface and are identified by the same naming scheme.

The messages sent to or from a specific service are fully described and after executing an operation at a service that component totally forgets about the caller.

Has greater **stability** because it restrains component performance. so that each component can’t see further than the immediate layer with which it is intermingling.

REST uses less bandwidth, simple and more flexible making it more useful for internet usage.

Uses http operations (GET, POST, PUT, DELETE, UPDATE, PATCH)

## Guiding principles for REST (constraints)

1. Its **layered system** allows generating a more scalable and flexible application. An application has better security due to its layered system, as components in each layer can’t interact outside the successive layer. Also, it balances loads and offers shared caches for stimulating scalability.
2. **Code on demand (optional):**

A REST API definition permits extending client functionality by downloading and implementing coding in the form of applets or scripts. This restructures clients by decreasing the number of features important to be pre-implemented.

This REST principle allows for applets to be communicated through the API used within the application.

1. **Uniform Interface**

By applying the principle of generality to the components interface, we can simplify the overall system architecture and improve the visibility of interactions.

**Constraints for applying the principle of Uniform Interface:**

* Identification of resources
* **Manipulation of resources through representations 🡪** The resources should have uniform representations in the server response. API consumers should use these representations to modify the resources state in the server.
* **Self-descriptive messages**
* **Hypermedia as the engine of application state🡪 The client should have only the initial URI of the application. The client application should dynamically drive all other resources and interactions with the use of hyperlinks.**

1. **Client-Server**

separation of concerns, By separating the user interface concerns (client) from the data storage concerns (server), we improve the **portability** of the user interface across multiple platforms and improve **scalability** by simplifying the server components.

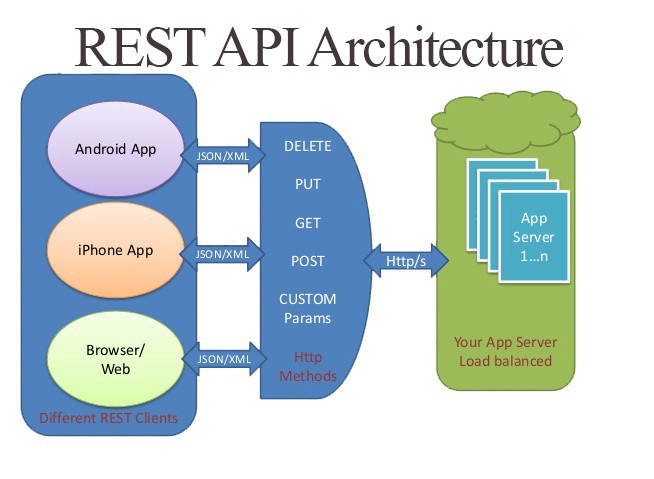
1. **Stateless**

The server cannot take advantage of any previously stored context information on the server. For this reason, the client application must entirely keep the session state

1. **Cacheable**

The cacheable constraint requires that a response should implicitly or explicitly label itself as cacheable or non-cacheable. And If the response is cacheable, the client application gets the right to reuse the response data later for equivalent requests and a specified period.

## How it works



A request is sent from client to server in the form of web URL as HTTP GET or POST or PUT or DELETE request. After that, a response comes back from server in the form of a resource which can be anything like HTML, XML, Image or JSON. But now JSON is the most popular format being used in Web Services.

## Using REST in implementation

# 2nd: Event-Driven architecture

## Load Balancer

Load balancing refers to the act of distributing network traffic across multiple services.

This makes sure that there's not too much load on a single server which could cause it to crash.

A load balancer acts as a ‘reverse-proxy’ to represent the application servers to the client through a virtual IP address (VIP). This technology is known as server load balancing (SLB). SLB is designed for pools of application servers within a single site or local area network (LAN).

Load balancers health check the application on the server to determine its availability. If the health check fails, the load balancer takes that instance of the application out of its pool of available servers. When the application comes back online, the health check validates its availability and the server is put back into the availability pool.

Because the load balancer is sitting in between the client and application server and managing the connection, it has the ability to perform other functions. The load balancer can perform content switching, provide content-based security like web application firewalls (WAF), and authentication enhancements like two factor authentication (2FA).

This is the primary function of the load balancer, server load balancing (SLB). The agent can provide additional functionality based on their role in the conversation. They can decide to allow and/or deny certain details (security). They may want to validate that the person they are talking to (authentication).

* Benefits of using Load Balancer
* it helps improve the responsiveness of your application.
* It also limits the chances of servers crashing as they're not being subjected to loads beyond what they're capable of withstanding.
* This is done to ensure maximum speed and capacity utilization

# High-Level System Design

# References