DON BOSCO COLLEGE OF ENGINEERING

FATORDA, MARGAO, GOA-403602

DEPARTMENT OF COMPUTER ENGINEERING

2020-2021

"FOCUSA: a comprehensive, scalable, and real-time e-Learning software"

By

Mr Yash Diniz

Mr Alston Dias

Mr Pranav Paranjape

Mr Imamsab Bagwan

Mr Rey Dias

Under the guidance of

Ms. Christina Barretto

Assistant Professor

ABSTRACT

Colleges, schools and other educational environments have a lot of existing tools which help students and teachers interact and share resources. These tools, however, are developed to serve very specific and limited purposes and are very difficult to manage manually. Most of them fail to provide a single comprehensive platform for all necessary management operations. Moreover, these existing tools also fail to cater to the management and analysis of data and thus are not used for generating insights and management optimisation.

Our project aims to deliver a scalable web services framework which is easy to work with for both the developer and the customers. It will allow connecting students and teachers through subscription portals, by letting faculty moderators post content which can be accessed by the students and synced whenever an internet connection is available. Furthermore, we aim to build a real-time video conferencing platform using a lossy compression algorithm involving neural networks. Finally, we plan on integrating various existing services like Google Calendar and Drive, to ease automation efforts, reducing the need to manually manage multiple tools.

ACKNOWLEDGMENT

First and foremost, we would like to thank our Director Fr. Kinley D' Cruz, Principal Dr. Neena S. P. Panandikar and Head of Department Dr. Gaurang Patkar for providing all the help we needed.

The team put in a lot of effort on the project. However, it would not have been possible to complete it without the help of our faculties.

We are highly indebted to Miss Christina Barretto for her guidance and constant supervision as well as for providing necessary information regarding the project and to our co-guides Dr. Vivek Jog and Miss Nisha Godhino for their guidance.

We would like to take this opportunity to thank our institution DON BOSCO COLLEGE OF ENGINEERING and our faculty members of the Computer Engineering Department who helped and supported us to go ahead with the project.

Contents

1 Introduction **1**

1.1 Problem Definition 1

1.1.1 Existing Systems 2

1.1.2 Proposed System 4

1.2 Purpose of the Project 4

1.3 Scope of the Project 5

1.4 Report Organization 6

2 Literature Survey **7**

2.1 WebRTC 7

2.1.1 RTCDataChannel 8

2.1.2 Establishing the connection 8

2.1.3 Signalling 9

2.1.4 NAT Traversal - ICE, TURN and STUN 9

2.1.5 Codecs 10

2.2 MATRIX 11

2.2.1 Why MATRIX? 12

2.3 PeerJS 13

2.3.1 PeerJS Server 13

2.4 The REST API 14

2.5 GraphQL 15

2.5.1 How does GraphQL excel REST API ? 17

3 Software Requirement Specification **18**

4 Design **19**

5 Implementation **20**

6 Conclusion **21**

List of Figures

2.1 Call Service: webRTC Signalling Plane 10

2.2 Matrix Architecture 12

2.3 GraphQL vs. REST 16

List of Tables

Chapter 1

Introduction

The essence of education is the ability for teachers to efficiently disseminate knowledge and obtain positive feedback from students, by using tools for a good teaching-learning experience. Educational environments have a lot of existing tools which help students and teachers interact and share resources. These tools, as will be explained further in this report, are developed to serve very specific and limited purposes, and are difficult to manage manually. They fail to provide a single comprehensive platform for all necessary operations. Moreover, these existing tools fail to cater to the management and analysis of data and thus are not used for generating insights and management optimisation. Furthermore, our worldwide internet infrastructure is lacking the reliability it needs to help people stay connected [2].

1.1 Problem Definition

Multiple e-learning platforms exist, each trying to solve a unique problem. Very few platforms function using an offline-first architecture, thus leading to really high dependency on network infrastructure. It gets difficult for both students and teachers to manage their work, since they have to use multiple platforms, dividing their attention among each of them. Current e-learning platforms also provide very little in the way of automation, and lack a common, integrable interface.

Not all software platforms are built considering unreliable internet service. This dependency on a reliable internet by a majority of its users thus affects the entire network, and reduces quality of service for all internet users. We can solve this problem in many ways, like improving the network infrastructure, or building applications that responsibly use internet bandwidth. If unattended, issues of an unreliable system can include lesser productivity.

1.1.1 Existing Systems

There are a lot of e-learning systems and platforms available in the market but the most popular ones are Google Classroom and Moodle. The following shall be a brief overview of these popular e-learning platforms.

Google Classroom

To quite an extent google classroom managed to replace the traditional learning management system with no paper requirement, easy document sharing using google docs, easy assignment submission process and features to integrate with other Google services like YouTube, Drive and even Meet. We feel it still couldn't fully become the replacement for education system because of the following:

- **Difficult account management.** Suppose you want to upload an assignment onto the classroom for submission, which is currently on another google account, you will need to logout of the current account, download the doc and sign in with the required Google account and then upload; quite a hassle.
- **Doesn't have an offline first design**, which eliminates the need of always being online to access the documents, as the documents can be downloaded in the user's local storage for a limited amount of time.
- **No live post updates**, i.e. you need to keep refreshing to view the recent updates.

One biggest pros of google education services is modularity. The G-Suite offers all these loosely coupled services like Meet, Classroom, Hangouts, and many others.

Moodle

Yet another popular e-learning platform and the best alternative to Google Classroom. Its pros include limited offline use for certain features. It has impressive features for uploading and downloading lecture notes, creating quizzes and tests, supports push notifications for both students and teachers, generating reports and many others. It has a backup, restore and import features, which turn out to be really useful for teachers. The teachers can also manage learners' profiles and setting enrollment keys, with role-based restrictions.

One of it's biggest cons however include poor scalability, due to its tightly coupled nature. This also causes issues with robustness, since a single module crash can cause the whole system to crash. No inbuilt video-conferencing functionalities, and it also becomes difficult to integrate a third party service due to the same tightly coupled nature.

It also offers a relatively poor mobile offering, with limited integration into third-party modules. There is a major learning curve for building and taking Moodle administration courses if you've never done it before.

Active Document Platform

The Active Document Platform(AD) [1] is a very useful platform that solves the problem of unreliable internet connectivity, by serving as an offline-first and distributed way of sharing course documents. Not being dependent on a centralised server on the internet allows it to operate offline as well, and it synchronizes with its main node whenever internet connectivity gets established. The best part about AD is its document organization, and distributed collaborative editing. Furthermore, AD uses SCORM (Sharable Content Object Reference Model), which is a

collection of standards and APIs which help streamline connecting between other e-learning platforms which also use SCORM. One of its biggest cons however is limited functionality, and not very user-friendly.

1.1.2 Proposed System

Our project aims to deliver a scalable web services framework which is easy to work with for both the developer and the customers. It will allow connecting students and teachers through subscription portals, by letting faculty moderators post content which can be accessed by the students and synced whenever an internet connection is available. Furthermore, we aim to build a real-time video conferencing platform using a lossy compression algorithm involving neural networks. Finally, we plan on integrating various existing services like Google Calendar and Drive, to ease automation efforts, reducing the need to manually manage multiple tools.

1.2 Purpose of the Project

We plan on integrating various existing services so that the students need to use a single application interface, and are kept updated. We will work towards reducing network bandwidth usage, adopting an offline-first architecture wherever applicable. We will also add many optimizations, especially in video conferencing by intelligently encoding only the required features and intelligently transmitting it. We will also build interfaces to anonymously collect and analyze data which can be used to measure, manage various processes by finding out patterns that can help teachers improve their course material, and allowing us to focus on what matters, letting the computer automate various basic operations for us. We will also work to make our project scalable, robust, and flexible, allowing integrations to various third-party modules using our API.

1.3 Scope of the Project

Must be implemented

- Scalable and loosely coupled Web Services Framework.
- Connecting students and teachers through subscription portals (courses), which can be made private by virtue of an authentication code if needed.
- Allowing faculty and students to have specific roles, which are moderated by limited people.

Should be implemented

- Integrating various existing services, like Google Calendar, Drive, classroom, etc. by building web-hooks, which allow for easy automation. (by virtue of bots)
- A real-time video conferencing platform using an experimental lossy compression algorithm involving neural networks.

Could be implemented

- Having a marketplace for modules, where it is as easy as integrating by pressing install.
- Creating a simple student planner application that can help the student manage submissions, projects, and time in general (integrated with Calendar for ease of use)
- A system that can perform collection, monitoring, reporting, and long-term benefit analysis of student and employee attendance at the college. It can help students set their priorities.
- An online test tool that allows faculty to create tests and students can answer tests.

1.4 Report Organization

The current introductory section provides a brief introduction to each chapter.

Chapter 1: Introduction

This section focuses on the purpose and scope of the proposed system of FOCUSA.

Chapter 2: Literature Survey

This section describes the concepts and technologies used to develop the project.

Chapter 3: Software Requirement Specification

This section provides information about the specific requirement of the proposed system.

Chapter 4: Design

This section describes the software lifecycle model, which will be used in developing the software. It also includes system design and detailed design.

Chapter 5: Implementation

This section deals with the implementation of the project where in the snapshots of each execution steps are shown.

Chapter 6: Conclusion

This section deals with the conclusion that can be derived after implementing the final System.

Chapter 2

Literature Survey

2.1 WebRTC

WebRTC (Web Real-Time Communication) is a free, open-source project providing web browsers and mobile applications with real-time communication via simple application programming interfaces (APIs). It allows audio and video communication to work inside web pages by allowing direct peer-to-peer communication, eliminating the need to install plugins or download native apps.

It supports video, voice, and generic data to be sent between peers, allowing developers to build powerful voice- and video-communication solutions.

There are 3 primary components of the WebRTC API and each plays a unique role in WebRTC specification:

MediaStream (GetUserMedia)

The MediaStream API provides a way to access device cameras and microphones using JavaScript. It controls where multimedia stream data is consumed, and provides some control over the devices that produce the media. It also exposes information about devices able to capture and render media.

RTCPeerConnection

The Peer Connection is the core of the WebRTC standard. It provides a way for participants to create direct connections with their peers without the need for an intermediary server (beyond signalling). Each participant takes the media acquired from the media stream API and plugs it into the peer connection to create an audio or video feed. The PeerConnection API has a lot going on behind the scenes. It handles SDP negotiation, codec implementations, NAT Traversal, packet loss, bandwidth management, and media transfer.

2.1.1 RTCDataChannel

The RTCDataChannel API was setup to allow bi-directional data transfer of any type of data - media or otherwise - directly between peers. It was designed to mimic the WebSocket API, but rather than relying on a TCP connection which although reliable is high in latency and prone to bottlenecks, data channels use UDP-based streams with the configurability of the Stream Control Transmission Protocol (SCTP) protocol. This design allows the best of both worlds: reliable delivery like in TCP but with reduced congestion on the network like in UDP.

2.1.2 Establishing the connection

Before a peer-to-peer video call can begin, a connection between the two clients needs to be established. This is accomplished through signalling. Signalling falls outside of the realm of the WebRTC specification but is the vital first step in establishing an audio/video connection.

2.1.3 Signalling

Signalling allows two endpoints (senders, receivers, or both) to exchange metadata to coordinate communication in order to set up a call. This call-and-response message flow contains critical details about the streaming that will take place, that is, the number and types of streams, how the media will be encoded, etc.

This is needed for two reasons: because the communicating peers do not know each other's capabilities, and the peers do not know each other's network addresses.

2.1.4 NAT Traversal - ICE, TURN and STUN

Once the initial signalling for a streaming connection has taken place, the two endpoints need to begin the process of NAT (Network Address Translation) traversal. This assigns a public address to a computer inside a private network for setting up a real-time connection. In a WebRTC-enabled communication, unless the two endpoints are on the same local network, there will be one or more intermediary network devices (routers/gateways) between the two. There are three key specifications that are used in WebRTC to overcome these hurdles:

- **Interactive Connectivity Establishment (ICE)** - ICE is used to find all the ways for two computers to "talk to each other". It has two main roles, gathering candidates and checking connectivity. It guarantees that if there is a path for two clients to communicate, it will find it and ensure it is the most efficient. It makes use of two protocols - STUN and TURN.
- **Session Traversal Utilities for NAT (STUN)** – It is a lightweight and simple method for NAT Traversal. STUN allows WebRTC clients to find out their own public IP address by making a request to a STUN server.
- **Traversal Using Relays around NAT (TURN)** - The TURN server assists in the NAT traversal by helping the endpoints learn about the routers

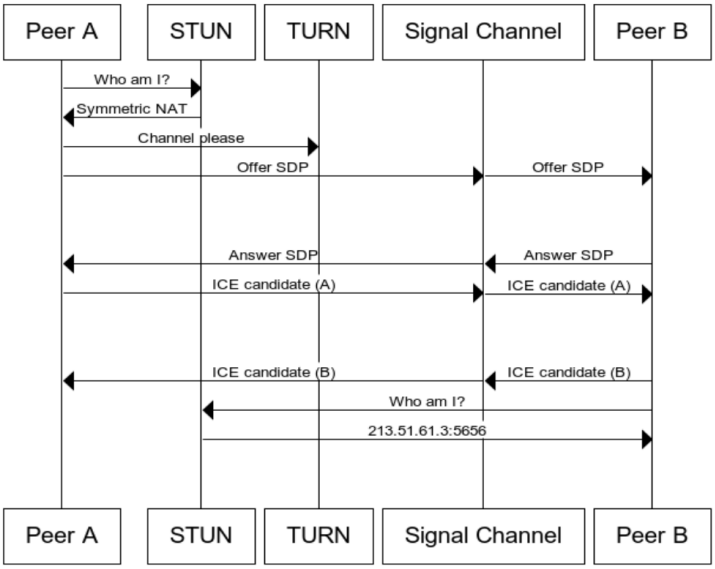

Figure 2.1: Call Service: webRTC Signalling Plane

on their local networks, as well as blindly relaying data for one of the endpoints where a direct connection is not possible due to firewall restrictions.

2.1.5 Codecs

Before sending the media over a peer connection, it has to be compressed. Raw audio and video is simply too large to send efficiently in our current Internet infrastructure. Likewise, after receiving media over a peer connection, it has to be decompressed. For this, we make use of a media codec.

WebRTC has mandated three audio codecs and two video codecs:

1. Audio - PCMU (G.711μ) running at 8,000Hz with a single channel (mono).
2. Audio - PCMA (G.711a) running at 8,000Hz with a single channel (mono).
3. Audio - Opus running at 48,000Hz with two channels (stereo).
4. Video - VP8.
5. Video - H.264/AVC using Constrained Baseline Profile Level 1.2.

The technology is available on all modern browsers as well as on native clients for all major platforms. The technologies behind WebRTC are implemented as an open web standard and available as regular JavaScript APIs in all major browsers. For native clients, like Android and iOS applications, a library is available that provides the same functionality.

2.2 MATRIX

Matrix is an open protocol for decentralised communication. It aims to make communication platforms interoperable and federated.

The main idea is to make real-time communication work seamlessly between different chat service providers, allowing users with accounts at one communications service provider to easily communicate with users of a different service provider.

Users have the privilege to communicate with people outside the Matrix network through bridges, which connect previously established communication networks, such as Slack and IRC, to the Matrix network. With bridges, you do not have to use different apps to talk to different people. Whatever Matrix client you choose, you can talk to anyone inside or outside the Matrix network.

Matrix gives users total control over their communication by letting them run or select their own server while still participating in a global network, rather than being locked in silos like Signal, WhatsApp, Telegram, Slack etc.

The key feature of Matrix is that no single server hosts or controls a given conversation - instead, as one user communicates with another, the conversation gets replicated equally across the servers - meaning all the participants equally share ownership over the conversation and its history. There is never a central point of control or authority, unless everyone decides to use the same server.

By default, Matrix uses simple HTTPS+JSON APIs as its baseline transport, but also embraces more sophisticated transports such as WebSockets or ultra-low band-

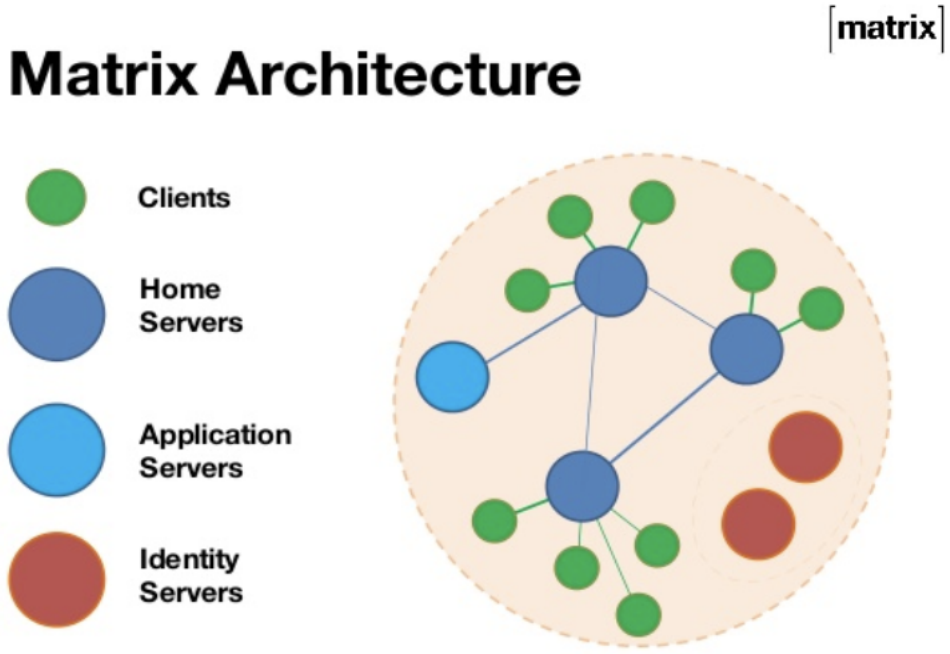

Figure 2.2: Matrix Architecture

width Matric via CoAP+Noise.

Applications using the Matrix protocol, called Matrix clients, have all the features one would want and expect from a modern chat app: instant messaging, group chats, audio and video calls, searchable message history, synchronization across all devices, as well as end to end encryption. Element is the best known Matrix client. Via Matrix, Element is able to bridge communications like IRC, Slack and Telegram into the app.

2.2.1 Why MATRIX?

As a result of several chat services not being interoperable with each other, people are forced to use multiple services to communicate. However, since Matrix is federated, just like email, you can freely communicate across a global network, without having to use specific services based on which ones your friends, family, and colleagues use.

Another major problem with online communication today is that most of the services

you use are operated by commercial organizations, forcing you to trust them to manage your data. But because Matrix is federated, you have control over where your data is stored and who has access to it.

2.3 PeerJS

The PeerJS library is aimed to simplify the peer-to-peer connection management. PeerJS wraps the browser's WebRTC implementation to provide a complete, configurable, and easy-to-use peer-to-peer connection API. With PeerJS, peers are identified by simply using an ID, a string that the peer can choose itself, or have a server generate one. Equipped with an ID, a peer can create a P2P data or media stream connection to a remote peer.

2.3.1 PeerJS Server

Although WebRTC promises peer-to-peer communication, a server is still required to act as a connection broker, and handle signalling.

PeerJS provides an open source implementation of this connection broker server named PeerJS Server, written in Node.js.Users can run their own Server, or are at leisure to opt for the cloud-hosted version of PeerServer provided for free. To broker connections, PeerJS connects to a PeerServer. The PeerJS Server acts ONLY as a connection broker, and it has to be noted that no peer-to-peer data goes through the server.

To make the P2P connection work seamlessly, we will build the neural network encoder and integrate it into WebRTC possibly using the P2P library.

2.4 The REST API

A REST API is an application programming interface that conforms to the constraints of the REST architectural style and allows for interaction with RESTful web services.

REST is not a standard, but rather a set of recommendations and constraints for RESTful web services. These include:

1. **Client-Server.** System 'A' makes an HTTP request to a URL hosted by System 'B', which returns a response. It is identical to how a browser works: The application first makes a request for a specific URL, the request is then routed to a web server that returns an HTML page. This page may contain references to images, style sheets, and JavaScript, which incur further requests and responses.

2. **Stateless.** REST is stateless: the client request should contain all the information necessary to respond to a request. In other words, it should be possible to make two or more HTTP requests in any order and the same responses will be received.

3. **Cacheable.** A response should be defined as cacheable or not.

4. **Layered.** The requesting client need not know whether it's communicating with the actual server, a proxy, or any other intermediary.

The working first involves sending a request from the client to the server in the form of web URLs, using the HTTP GET, POST, PUT or DELETE methods. After that a response is retrieved from the server in the form of a resource which can be anything like HTML, XML, Image or JSON.

In HTTP there are five methods which are commonly used in a REST based Architecture i.e. POST, GET, PUT, PATCH, and DELETE. These methods correspond to create, read, update, and delete (or CRUD) operations respectively.

2.5 GraphQL

GraphQL is a query language and server-side runtime for application programming interfaces (APIs) that prioritizes giving clients exactly the data they request and nothing more. GraphQL is designed to make APIs fast, flexible, and developer-friendly. It is not tied to any specific database or storage engine and is instead backed by your existing code and data.

API developers use GraphQL to create a schema to describe all the possible data that clients can query through that service. This schema is made up of object types, which define which kind of object you can request and what fields it has. As queries come in, GraphQL validates the queries against the schema. GraphQL then executes the validated queries.

The API developer attaches each field in a schema to a function called a resolver. During execution,the resolver is called to produce the value.

GraphQL follows the same set of constraints as REST APIs, but it organizes data into a graph using one interface. Objects are represented by nodes (defined using the GraphQL schema), and the relationship between nodes is represented by edges in the graph. Each object is then backed by a resolver that accesses the server's data.

When a GraphQL server responds to an end user's request, it begins with the query root, and the resolver executes every field on the requested object. A key-value map houses each field's values, and some return another object selecting another set of fields. This continues until only a string or a number is returned. The server then responds with a nested set of objects, as requested by the end user.

Figure 2.3: GraphQL vs. REST

2.5.1 How does GraphQL excel REST API ?

Data Fetching

With a REST API, you would typically gather the data by accessing multiple endpoints. You basically end up having to make multiple requests to different endpoints to fetch the required data. In GraphQL on the other hand, you'd simply send a single query to the GraphQL server that includes the concrete data requirements. The server then responds with a JSON object where t hese requirements are fulfilled. Therefore using GraphQL, the client can specify exactly the data it needs in a query.

Chapter 3

Software Requirement Specification

Chapter 4

Design

Chapter 5

Implementation

Chapter 6

Conclusion

Bibliography

[1] Active Document, http://sensei.lsi.uned.es/ActiveDocument/.

[2] Carlos Iglesias. As internet access proves critical, we are missing targets to connect everyone. https://webfoundation.org/2020/04/covid-missed-targets/, April 2020.