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. Maria Christina Barretto

Assistant Professor

Dr. Vivek Jog

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.

This project aims to deliver a scalable web services framework which is easy to work with for both the developer and the end users. 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, the project will have a real-time video conferencing platform using an experimental lossy compression algorithm involving neural networks. Finally, various existing services like Google Calendar and Drive will be integrated, 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 Maria 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

List of Figures

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 [25].

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.This problem can be solved 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. Google Classroom couldn't fully become a replacement for education systems because of the following:

- **Difficult account management.** Suppose the user wants to upload an assignment onto the classroom for submission, which is currently on another google account, the user will need to logout of the current account, download the document and sign in with the required Google account and 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. the user 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 for beginners.

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, the project will have a real-time video conferencing platform using an experimental lossy compression algorithm involving neural networks. Finally, various existing services like Google Calendar and Drive will be integrated, to ease automation efforts, reducing the need to manually manage multiple tools.

1.2 Purpose of the Project

The purpose of the project is to offer a single integrated interface which the students can access and stay updated. Furthermore, the project will adopt an offline-first architecture, thus reducing the dependency on network infrastructure. Being an offline-first application would help the students access their work while being "disconnected", which can help the students focus when they need to.

The project will also prioritize optimizing for scalability, robustness and flexibility, allowing easy integration for new modules. One of the goals of this project is to improve communication between users of the product while reducing network bandwidth usage. Finally, the project aims to build interfaces and tools for anonymous data collection and analysis for measuring, managing, and possibly automating various patterns, while also offering insights to the faculty.

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

This chapter begins with reviews of popular and recent web standards like WebRTC, Matrix, GraphQL and REST. Further sections then review existing neural network architectures, best practices, and algorithms. Finally, the chapter ends with the review and comparison of various existing key technologies, packages and tools.

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. [22]

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.

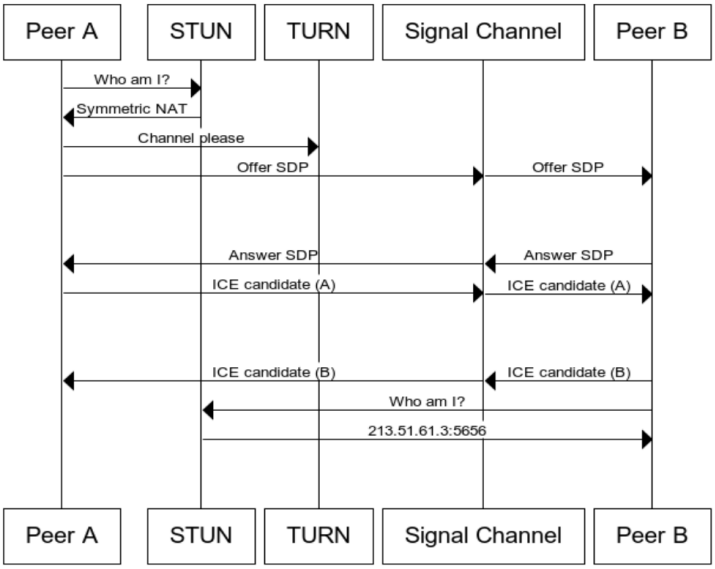

Figure 2.1: Call Service: webRTC Signalling Plane

- **Traversal Using Relays around NAT (TURN)** - The TURN server assists in the NAT traversal by helping the endpoints learn about the routers 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 on the current Internet infrastructure. Likewise, after receiving media over a peer connection, it has to be decompressed. For this, use 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. [19]

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, the need for using different apps to talk to different people is eliminated. Whatever Matrix client the user chooses, 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. [10]

By default, Matrix uses simple HTTPS+JSON APIs as its baseline transport, but also embraces more sophisticated transports such as WebSockets or ultra-low bandwidth Matric via CoAP+Noise. [11]

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. [29]

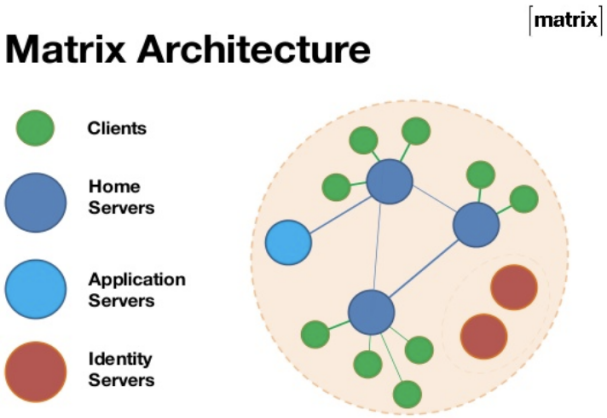

Figure 2.2: An example of a Matrix Architecture, illustrated very simply using abstract coloring.

2.2.1 Conclusion

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, the sender can freely communicate across a global network, without having to use specific services based on which ones the intended receiver uses.

Another major problem with online communication today is that most of the services are operated by commercial organizations. This means the user's data is only

protected by policy, and not by design. But because Matrix is federated, the user has control over their data storage and access. [30]

2.3 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. [6]

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. [21]

Figure 2.3: A simple illustration of a REST API request-response.

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. [7]

2.4 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 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 can be requested 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. [20]

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.

Figure 2.4: GraphQL can handle the tasks of multiple REST endpoints, neither overfetching, nor underfetching, offering the exact data format as requested by the client.

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. [15]

2.4.1 How does GraphQL excel REST API?

Data Fetching

With a REST API, data would be gathered by accessing multiple endpoints. The application would end up having to make multiple requests to different endpoints

to fetch the required data, and could lead to overfetching of data. In GraphQL on the other hand, a single query to the GraphQL server that includes the concrete data requirements would be sufficient. The server would then respond with a JSON object where these requirements are fulfilled. Therefore using GraphQL, the client can specify exactly the data it needs in a query.

Over-fetching and Under-fetching of Data

One of the most common problems with REST is that of over-fetching and under-fetching. This happens because the only way for a client to download data is by hitting multiple endpoints that return fixed data structures. GraphQL gives the clients the exact data they request for.

Benefits of a Schema and Type System

GraphQL uses a strong type system to define the capabilities of an API. All the types that are exposed in an API are written down in a schema using the GraphQL Schema Definition Language (SDL). This schema serves as the contract between the client and the server to define how a client can access the data. Once the schema is defined, the teams working on frontend and backends can do their work without further communication since they both are aware of the definite structure of the data that's sent over the network. Frontend teams can easily test their applications by mocking the required data structures. Once the server is ready, the switch can be flipped for the client apps to load the data from the actual API.

Rapid Product Iterations on the Front-end

A common pattern with REST APIs is to structure the endpoints according to the views inside the app in order for the client to get all required information for a particular view by simply accessing the corresponding endpoint. However, the major drawback of this approach is that it doesn't allow for rapid iterations on the

frontend. With every change that is made to the UI, there is a high risk that now there is more or less data required than before. Consequently, the backend needs to be adjusted as well to account for the new data needs. This notably slows down the ability to incorporate user feedback into a product. But owing to the flexible nature of GraphQL, changes on the client-side can be made without any extra work on the server. Since clients can specify their exact data requirements, no backend adjustments are required when the design and data needs on the frontend change.

Insightful Analytics on the Back-end

GraphQL allows having fine-grained insights about the data that's requested on the backend. As each client specifies exactly what information it's interested in, it is possible to gain a deep understanding of how the available data is being used. This can for example help in evolving an API and deprecating specific fields that are not requested by any clients any more. With GraphQL, the developer can also do low-level performance monitoring of the requests that are processed by the server. GraphQL uses the concept of resolver functions to collect the data that's requested by a client. Instrumenting and measuring performance of these resolvers provides crucial insights about bottlenecks in the system. [8]

2.5 Neural Discrete Representation Learning

Recent advances in generative modelling of images, audio and videos have yielded impressive samples and applications. At the same time the challenges that follow these tasks are few-shot learning, domain adaptation, or reinforcement learning heavily rely on learnt representations from raw data.

Learning representations with continuous features have been used in many previous work but discrete representations are potentially more natural fit for many applications. Discrete representations are a natural fit for complex reasoning, planning and predictive learning (e.g., if it rains, I will use an umbrella).

This concept of discrete representation gave rise to the Vector Quantization-Variational Autoencoder. This model relies on Vector Quantization with the combination of Variational Autoencoder framework with discrete latent representations through a novel parameterisation of the posterior distribution of (discrete) latents given an observation. [31]

2.5.1 Vector Quantization - Variational Autoencoder (VQ-VAE)

VAEs consist of the following parts: an encoder network which parameterised a posterior distribution $q(z|x)$ of discrete latent random variables z given the input data x, a prior distribution $p(z)$, and a decoder with a distribution $p(x|z)$ over input data.

Typically, the posteriors and priors in VAEs are assumed normally distributed with diagonal covariance. Extensions include autoregressive prior and posterior models, normalising flows, and inverse autoregressive posteriors . VQ-VAE uses discrete latent variables which is a new way of training, by using vector quantisation (VQ). The posterior and prior distributions are categorical, and the samples drawn from these distributions index an embedding table. These embeddings are then used as input into the decoder network.

2.5.2 Discrete Latent Variables

Let's define a latent embedding space $e \in R^{K \times D}$ where K is the size of the discrete latent space (i.e., a K-way categorical), and D is the dimensionality of each latent embedding vector e_i. Note that there are K embedding vectors $e_i \in R^D$, $i \in 1, 2, ..., K$.

The discrete latent variables z are then calculated by a nearest neighbour look-up using the shared embedding space e as shown in equation 2.1.

$$q(z = k|x) = \begin{cases} 1 & \text{for } k = \text{argmin}_j \|z_e(x) - e_j\|_2 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where $z_e(x)$ is the output of the encoder network.

The input to the decoder is the corresponding embedding vector e_k as given in equation 2.2

$$z_q(x) = e_k, \text{where } k = \text{argmin}_j \|z_e(x) - e_j\|_2 \tag{2.2}$$

2.5.3 Learning

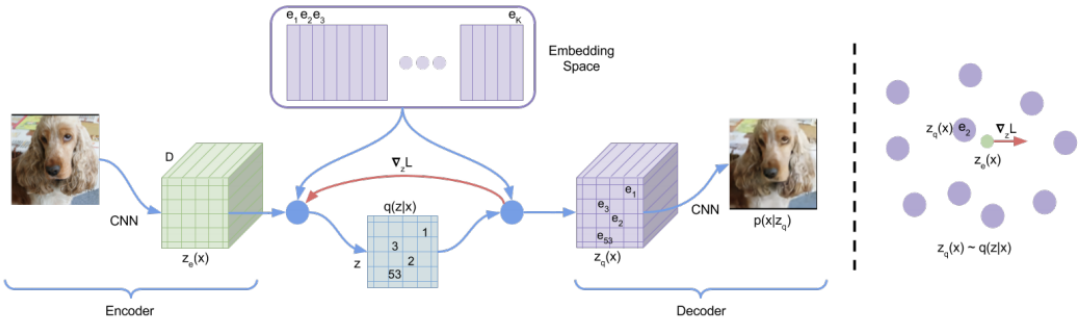

Figure 2.5: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2. The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

During forward computation the nearest embedding $z_q(x)$ is passed to the decoder, and during the backwards pass the gradient $\nabla_z L$ is passed unaltered to the encoder. Since the output representation of the encoder and the input to the decoder share the same D dimensional space, the gradients contain useful information for how the encoder has to change its output to lower the reconstruction loss.

As seen on figure 2.5 (right), the gradient can push the encoder's output to be discretized differently in the next forward pass, because the assignment in equation 2.1 will be different

The log-likelihood of the complete model $\log p(x)$ can be evaluated as follows:

$$\log p(x) = \log \sum_k p(x|z_k)p(z_k) \tag{2.3}$$

Because the decoder $p(x|z)$ is trained with $z = z_q(x)$ from MAP-inference, the decoder should not allocate any probability mass to $p(x|z)$ for $z \neq zq(x)$ once it has fully converged.

Thus, the authors concluded. [28, 31]

$$\log p(x) \approx \log p(x|z_q(x))p(z_q(x)) \tag{2.4}$$

2.5.4 Prior

The author also stated a prior distribution can be made autoregressive by depending on z in the feature map as it is a categorical distribution over the discrete latents $p(z)$. While training the VQ-VAE the prior is kept constant and uniform. After training, fit an autoregressive distribution over z, $p(z)$, so that it can generate x via ancestral sampling.

2.6 Efficient text-search using TF-IDF

TF-IDF stands for Term Frequency-Inverse Document Frequency. The TF-IDF weight is a weight often used in information retrieval and text mining. Variations of the TF-IDF weighting scheme are often used by search engines in scoring and ranking a document's relevance given a query. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

TF-IDF is a weighting scheme that assigns each term in a document a weight based on its Term Frequency (TF) and Inverse Document Frequency (IDF). The terms with higher weight scores are considered to be more important.

Typically, the TF-IDF weight is composed by two terms:

1. Normalized Term Frequency (TF)
2. Inverse Document Frequency (IDF)

Therefore, Term Frequency is a measure of how frequently a term occurs in a document.

$$tf(t, D) = \frac{\text{Number of times the term } t \text{ appears in document } D}{\text{Total number of terms in the document}} \tag{2.5}$$

Inverse Document Frequency is a measure of how important a term is. [18]

$$idf(t, D) = \log \frac{N}{n_t} \tag{2.6}$$

where, N = Total number of documents, n_t = Number of documents having the term t in it.

Now combine both to produce a composite weight for each term in each document.

$$tfidf(t, D) = tf(t, D) \times idf(t, D) \tag{2.7}$$

Also, The TF-IDF score/relevancy for a multi-term query $q = (t1, ..., t_m)$ is [32]:

$$score_{tfidf}(q, d, D) = \sum_{i=1}^{m} score_{tfidf}(t_i, d, D) \tag{2.8}$$

Use of TF-IDF text search in an example

Suppose a search engine contained the following three documents,

Document A: "the mouse played with the cat"

Document B: "the quick brown fox jumped over the lazy dog"

Document C: "dog 1 and dog 2 ate the hot dog"

The TF-IDF vector is now computed for each document like so [23]:

Document A: "the mouse played with the cat"

$$\text{Relevance(the, Doc}_A) = \frac{2}{6} \times \ln\left(\frac{3}{3}\right) = 0.0$$
$$\text{Relevance(mouse, Doc}_A) = \frac{1}{6} \times \ln\left(\frac{3}{1}\right) = 0.183102$$
$$\text{Relevance(played, Doc}_A) = \frac{1}{6} \times \ln\left(\frac{3}{1}\right) = 0.183102$$
$$\text{Relevance(with, Doc}_A) = \frac{1}{6} \times \ln\left(\frac{3}{1}\right) = 0.183102$$
$$\text{Relevance(cat, Doc}_A) = \frac{1}{6} \times \ln\left(\frac{3}{1}\right) = 0.183102$$

2.6.1 Compare query against a document

1. Find the TF-IDF vector for the document. This should be an easy, $O(1)$ lookup since the TF-IDF vector was already computed.

2. Compute the TF-IDF vector for the query. (Note: technically, the query is treated as if it were a new document. However, the IDF values need not be recomputed: just use the ones computed earlier.)

3. Compute the cosine similarity between the document vector and the query vector.

4. This can be done by comparing the TF-IDF vector for the query against all the document TF-IDF vectors and compute how similar the query vector is to each document by exploiting properties of the cosine function and the dot product.

The K documents of the collection with the highest vector space scores on the query q need to be determined. Typically, these K top documents ordered by score in decreasing order is what is needed; for instance many search engines use $K = 10$ to retrieve and rank-order the first page of the ten best results. [23]

Time Complexity

The time complexity of this algorithm is:

$$O(|Q||D||T|) = O(|D||T|) \tag{2.9}$$

where $|Q|$ is considered fixed, Q is the set of words in the query and D is the set of all documents.

Also, if the D is the set of only the matching documents, then since scores is a priority queue (at least in doc-at-time-scheme) built on heap, putting every d into takes $\log K$. Therefore, an estimate can be $O(|D| \log K)$. [16]

2.7 Comparison between Key Technologies

This section introduces key technologies, software packages and tools. It is important to note that NVIDIA Maxine will not be used while building this project. The subsection dedicated to NVIDIA Maxine contains a key observation which could possibly affect the project, hence it was added.

2.7.1 NVIDIA Maxine

NVIDIA Maxine is a fully accelerated platform SDK for developers of video conferencing services to build and deploy AI-powered features that use state-of-the-art models in their cloud. Video conferencing applications based on Maxine can reduce

video bandwidth usage down to one-tenth of H.264 using AI video compression, dramatically reducing costs.

Maxine includes APIs for the latest innovations from NVIDIA research such as face alignment, gaze correction, face re-lighting and real time translation in addition to capabilities such as super-resolution, noise removal, closed captioning and virtual assistants. These capabilities are fully accelerated on NVIDIA GPUs to run in real time video streaming applications in the cloud.

Maxine-based applications let service providers offer the same features to every user on any device, including computers, tablets, and phones. Applications built with Maxine can easily be deployed as microservices that scale to hundreds of thousands of streams in a Kubernetes environment. [12]

Features

- **Easy to use SDK**: Includes libraries, tools and example pipelines for developers to quickly add AI features to their applications.
- **Ultra-low Bandwidth**: AI Video Compression uses one-tenth the bandwidth of H.264 video compression standard.
- **State-of-the-art AI model**: Includes pre-trained models with thousands of hours of training on NVIDIA DGX™A100.
- **Fully GPU Accelerated**: Optimizes end-to-end pipelines for the highest performance on NVIDIA Tensor Cores GPUs.

Key Technologies

- **Reduce Video Bandwidth compared to H.264**: With AI-based video compression technology running on **NVIDIA GPUs**, developers can reduce bandwidth use down to one-tenth of the bandwidth needed for the H.264 video compression standard. This cuts costs for providers and delivers a smoother

video conferencing experience for end users, who can enjoy more AI-powered services while streaming less data on their computers, tablets, and phones.

- **Face Re-Animation**: Using new AI research, the key facial points of each person on a video call can be identified and then used to reanimate a person's face on the other side using a still image of the call with Generative Adversarial Networks (GANs).

  These key points can be used for face alignment, where faces are rotated so that people appear to be facing each other during a call, as well as gaze correction to help simulate eye contact, even if a person's camera isn't aligned with their screen.

  Developers can also add features that allow call participants to choose their own avatars that are realistically animated in real time by their voice and emotional tone.

- **Video and Audio Effects**: AI-based super-resolution and artifact reduction can convert lower resolutions to higher resolution videos in real time which helps to lower the bandwidth requirements for video conference providers, as well as improves the call experience for users with lower bandwidth. Developers can add features to filter out common background noise and frame the camera on a user's face for a more personal and engaging conversation.

  Additional AI models can help remove noise from low-light conditions creating a more appealing picture.

- **Conversational AI**: Maxine-based applications can use NVIDIA Jarvis, a fully accelerated conversational AI framework with state-of-the-art models optimized for real time performance. Using Jarvis, developers can integrate virtual assistants to take notes, set action items, and answer questions in human-like voices.

  Additional conversational AI services such as translations, closed captioning and transcriptions help ensure everyone can understand what's being discussed on the call.

Observation

NVIDIA Maxine needs all of its devices to have NVIDIA hardware or access to the NVIDIA cloud. This reduces the customer base because of its cost and huge unavailability of GPUs.

It is therefore important to understand that dependence on compute-intensive hardware could currently be the only barrier to building effective applications using Neural Networks.

2.7.2 Comfort Noise

Comfort noise is an audibly soft, synthetic background noise that was introduced in wireless communications mainly to overcome the issues of artificial silence which occurs in voice activity detection systems. The issues of receiving prolonged periods of artificial silence would include:

- The listener disconnecting prematurely believing transmission has been lost.
- The speech does not sound smooth, since the voice activity detector would abruptly cut the signal as an attempt to save bandwidth, but this would give the listener a perception that the voice quality has worsened.
- The sudden changes in voice amplitude, punctuated by repeated gaps of silence would be jarring to the listener.

As can be observed, the above issues are not technical. Engineers arrived at a simple solution which involved adding comfort noise to the communication. This noise is added at the receiving end of the communication, and not transmitted. [4] Such a setup helps save bandwidth while also circumventing the issues described above.

2.7.3 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. [13]

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. [17]

2.7.4 CouchDB for Offline first design

What is RxDB?

RxDB (Reactive Database) is a NoSQL database for JavaScript Applications like websites, hybrid Apps, Electron-Apps, Progressive Web Apps and NodeJs.

RxDB allows querying the current state but subscribe to all state changes. Therefore it is useful in UI-based real time applications and makes it easy to develop and adds great performance benefits. RxDB provides modules for realtime replication with any CouchDB compliant endpoint and also with custom GraphQL endpoints to

replicate data between clients and server. [26]

What is PouchDB?

PouchDB is an open-source JavaScript database inspired by Apache CouchDB which runs well within the browser. PouchDB helps build applications that work as well offline as they do online. It allows applications to store data locally while offline, then synchronize it with CouchDB and compatible servers when the application is back online, keeping the user's data in sync no matter where they next login. [14]

What is CouchDB?

CouchDB is (categorised as) a NoSQL database.A CouchDB database does not have a schema, or pre-defined data structures such as tables. Data is stored as JSON document. The structure of the data, or document, can change dynamically to accommodate evolving needs. CouchDB is a peer based distributed database system. Any number of CouchDB hosts (servers and offline-clients) can have independent "replica copies" of the same database, where applications have full database interactivity (query, add, edit, delete). When back online or on a schedule, database changes can be replicated bi-directionally. It is schema-free. [24]

Offline First Design

Offline first is an application development paradigm where developers ensure that the functionality of an app is unaffected by intermittent lack of a network connection. In addition offline first usually implies the ability to sync data between multiple devices For an app to be offline first, both the code and the data required for it to function should not be dependent on the presence of the network. Mobile applications do need to do something extra to make code available offline, Web Applications however can include service workers to achieve the same. With service workers the concern shifts to finding the safest time to update the code. Refer to figure 2.6.

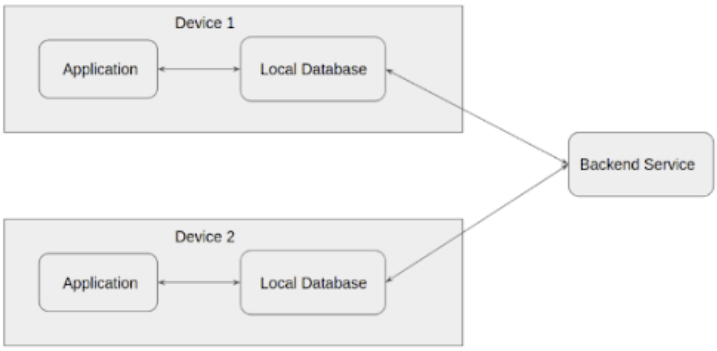

Figure 2.6: Simple architecture of the offline first model

A simple solution to making data available offline is to have a local database to read and write to. The database can then be replicated to and from a server whenever a network is available. [9]

Push-pull replication

RxDB can do a two-way replication(push-pull replication) with a GraphQL endpoint. This allows data replication from the server into the client-side database and then realtime query and modification. When the user is offline, they still can use the data and later sync it with the server when the client is online again.

Pros

1. GraphQL-replication is faster and needs less resources.
2. Does not need a couchdb-compliant endpoint, only a graphql-endpoint.

Cons

1. Cannot replicate multiple databases with each other.
2. It is assumed that the graphql-server is the single source of truth.

3. GraphQL server has to be setup, while with couchdb-sync, the replication only requires to start a server. [27]

Conflict resolution

If the app can be used from multiple devices, it is possible for the user to make conflicting changes on different devices.

- Simplest way to handle conflicts is to assume that they don't matter and users will simply correct the data later. This approach is also known as last-write-wins.

Since conflict detection and resolution is a must, there are many broad approaches, one of them which pouchDB uses is:

Version the objects:

Every time a change is made a new version is created. In addition the parent version for a given version is also kept track. A conflict can be identified by the fact that two versions have the same parent. Two generic auto merge strategies are:

1. **Last write wins**: Here the last revision is considered to reach the server as the final value.
2. **Merge by fields**: If the two devices modified different fields of the object, it might be possible to auto merge by taking the new fields from both objects.

In the above approach merging is done by the server and clients simply fetch the merged value. However if user intervention is needed to resolve a conflict, then there is a feature to merge on the client. This will require all clients to store the version history of the document as well. A problem with this is that clients can

independently merge the document creating new multiple merge revisions. Clients need to handle this (For example by examining the history and ignoring one of the merge versions).

PouchDB and CouchDB handle this by using a hash of the document contents as the object version. So if two different devices resolve the conflict the exact same way they will end up with the exact same version id.

How are documents deleted?

A simple way to delete objects is to have a "deleted" flag in the object. Merge function can then decide what to do if a deleted object has been modified. On the client, any object with the deleted flag set can be immediately purged as there will be no more updates to it. The server would need to make sure that every client has purged the object locally. The not so ideal but practical alternative is to periodically purge old objects.

Who uses this currently?

PouchDB and CouchDB follow the above model. In this setup, versions are stored both on the client and the server. When there is a conflict a deterministic algorithm auto picks a winning version. [9]

2.7.5 Eventual consistency via CAP theorem

A distributed system is a system that operates robustly over a wide network. A particular feature of network computing is that network links can potentially disappear, and there are plenty of strategies for managing this type of network segmentation. CouchDB differs from others by accepting eventual consistency, as opposed to putting absolute consistency ahead of raw availability like RDBMS or Paxos. What these systems have in common is an awareness that data acts differently when many

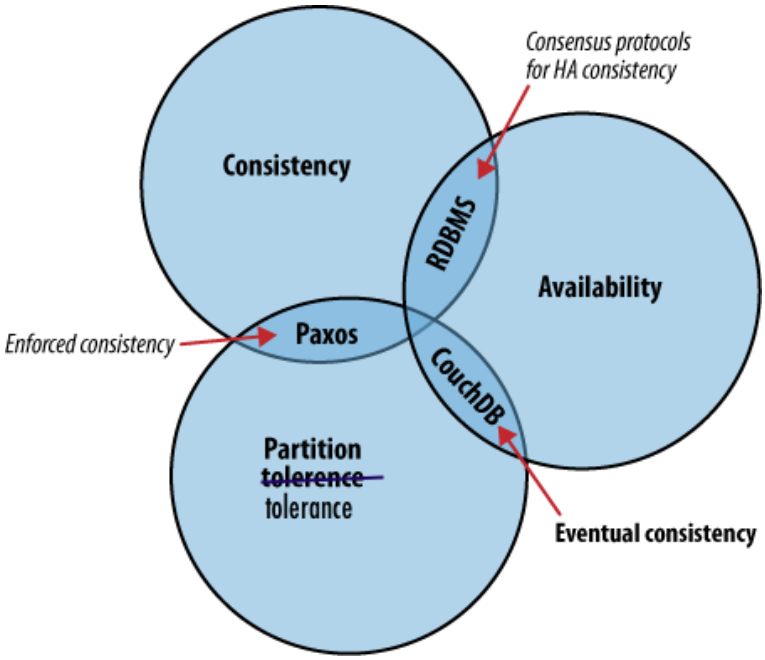

Figure 2.7: CAP Theorem. No database can offer the perfect solution for CAP.

people are accessing it simultaneously. Their approaches differ when it comes to which aspects of consistency, availability, or partition tolerance they prioritize. Refer to figure 2.7.

The CAP theorem describes a few different strategies for distributing application logic across networks. CouchDB's solution uses replication to propagate application changes across participating nodes. This is a fundamentally different approach from consensus algorithms and relational databases, which operate at different intersections of consistency, availability, and partition tolerance.

- **Consistency**: All database clients see the same data, even with concurrent updates.
- **Availability**: All database clients are able to access some version of the data.
- **Partition tolerance**: The database can be split over multiple servers.

If availability is a priority, the clients can write data to one node of the database without waiting for other nodes to come into agreement. If the database knows

how to take care of reconciling these operations between nodes, a sort of "eventual consistency" is achieved in exchange for high availability.

MapReduce algorithm

CouchDB uses MapReduce to compute the results of a view. MapReduce makes use of two functions, "map" and "reduce," which are applied to each document in isolation. Being able to isolate these operations means that view computation lends itself to parallel and incremental computation. More important, because these functions produce key/value pairs, CouchDB is able to insert them into the B-tree storage engine, sorted by key. Lookups by key, or key range, are extremely efficient operations with a B-tree, described in big O notation as $O(logN)$ and $O(logN + K)$, respectively. [5]

Chapter 3

Software Requirement Specification

3.1 Introduction

3.1.1 Background

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. They fail to provide a single comprehensive platform for all necessary management operations. Moreover, these existing tools fail to cater to the management and analysis of data and thus are not used for generating useful insights.

3.1.2 Project Overview

This 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, the project will have a real-time video conferencing platform using an experimental lossy compression algorithm involving neural networks. Finally, various existing services like Google Calendar and Drive will be integrated, to ease automation efforts, reducing the need to manually manage multiple tools.

3.1.3 Hardware Requirements

Development: Minimum 8 GB RAM for native app development.

Production: As stated in the constraints the server should have at least 1 GB RAM. The client should be able to run the minimal version of the neural network encoder. (final production requirements will be obtained through experimentation)

3.1.4 Software Requirements

Recommended: Linux OS 4.19 kernel or later in production. Windows 10 1803 (Build 17134 or later) for development. NGINX server 1.17 or later as reverse proxy and load balancer. Any smartphone or PC supporting smooth operation of the most recent Google Chrome or Mozilla Firefox available at client.

3.1.5 Constraints

Needs to run in 1GB, with 1 vCPU core at production environment. Clients will have limited network bandwidth, mostly using mobile web browsers, but have persistent cache for offline storage (which may be cleared often)

3.1.6 Assumptions

- Clients will mostly operate offline, with syncing times usually after working hours.
- Server will be operating in a scalable and distributed architecture, initially using college or cloud infrastructure.
- Clients will operate offline-first using persistent browser cache to download and store filtered versions of the database on themselves, effectively reducing server load.

3.1.7 Dependencies

NodeJS, GraphQL, CouchDB, ReactJS

3.2 Functional Requirements

- Connecting students and teachers through subscription portals (topics), which can be made private by virtue of an authentication code if needed.
- Allowing faculty and students to post content to specific groups, which are moderated by limited people.
- A real-time video conferencing platform using an Experimental lossy compression algorithm involving neural networks

3.3 Non Functional Requirements

3.3.1 Scalability

- Using CouchDB as a distributed database allows for simplifying horizontal scalability using eventual consistency.

- Using containerized microservices further improve the scope for horizontal scalability.
- Simple since GraphQL is used as the Web Services Framework and also using the ReactJS and CouchDB technology stack.

3.3.2 Portability

- GraphQL is a platform-independent querying API, which can serve as a Web Services Framework for improved portability and integration.
- ReactJS is a cross-platform framework for building reactive websites. It supports any platform that can run a modern graphical web-browser.

3.3.3 Security

- SSL will be used for encrypting the traffic.
- Servers will have firewalls, and SSH connections will be protected by password and certificates.
- Each service will also be containerized, running in complete isolation, interacting strictly via loosely coupled message passing.

3.3.4 Maintainability

- Microservice architecture will be used on the server-side. The loosely-coupled modularity offered by this architecture simplifies maintenance tasks.
- The frontend will be built with reusable ReactJS components. The goal is to maximize code reuse, and simplify maintenance.
- Third-party npm packages will also be incorporated wherever necessary.

3.3.5 Performance

- Distillation techniques for the neural networks will also be incorporated, and an attampt will be made to reduce the CPU and memory footprints of the same as much as possible.

3.4 Interface Requirements

3.4.1 User Interfaces

- Simple, minimal, responsive and easy to navigate User Interface.
- User Interface elements must be consistent.
- Strategic use of themes and colors to suit the purpose.
- Component to navigate to and display the content pertaining to video conferencing module.
- Components to navigate to course subscription, profile and post, login, logout.
- Components dealing with posting, collection and visual display of data analysis

3.4.2 Hardware Interfaces

- SSH protocol for the server-side interface.
- Device hardware, like the camera, microphone, and others. The WebRTC negotiation interface will be used to obtain secure access to the hardware.

3.4.3 Communication Interfaces

- WebRTC (Real-Time Communication) standards and protocols to enable real time, peer to peer audio and video communication.

- Request/Response protocol, HTTP/1.1 (as inherent to GraphQL)

3.5 Technology Used

Frontend: ReactJS (For Webapp), React Native (for Mobile App)

Middleware: NodeJS, GraphQL

Backend: CouchDB, PouchDB databases, with promises.

3.6 Definitions, Acronyms and Abbreviations

- **Web Service** - a service offered by an electronic device to another electronic device, communicating with each other via the World Wide Web, or a server running on a computer device.
- **Native App** - An app to be installed and run on a device without using any form of emulation.
- **Neural Network Encoder** - A neural network encoder converts input into a feature vector which represents the input but taking lesser space than the original input.
- **Reverse Proxy** - ensures smooth flow of traffic between the client and the server and routes the client requests to appropriate backend server. Popularly used for load balancing and security.
- **Load Balancing** - methodical and efficient distribution of network or application traffic across multiple servers.
- **Microservices** - The application is broken down into multiple, isolated services, each performing a small part of the entire application. Microservices are easy to manage and operate in parallel.

- **Message Passing** - Applications can share data with each other by passing messages over the network. These messages will be passed using a standard protocol like HTTP.
- **Persistent Cache** - intended for intermediate term storage of documents or data objects.
- **Lossy Compression** - method of data compression in which the size of the file is reduced by eliminating data in the file.
- **SSH** - Secure Shell Protocol is a method for secure remote login from one computer to another.
- **WebRTC** - Web Real-Time Communication comprises of protocols, standards and JavaScript API which enable real time P2P communication.
- **Peer to Peer** - Application to Application communication over the internet, where data packets always attempt to find the shortest path between devices, thus possibly reducing latency.
- **HTTP** - Hypertext Transfer Protocol gives users a way to interact with web resources such as HTML files by transmitting hypertext messages between clients and servers.

Chapter 4

Design

4.1 Agile SDLC

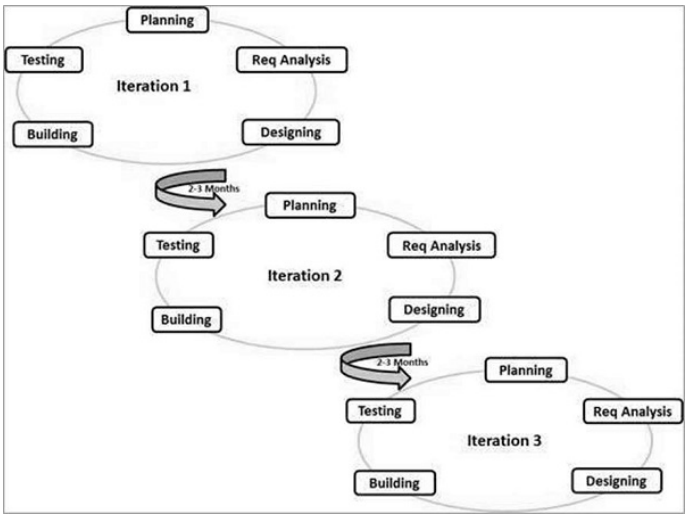

Figure 4.1: Agile SDLC model

The philosophy of Agile is that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. To deliver specific features for a release the tasks are divided into time boxes (small time frames).

Agile uses an iterative approach and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the

features required by the customer. Refer to figure 4.1.

Following are the Agile Manifesto principles:

- **Working software:** The best means of communication with the customers to understand their requirements is by using demo working software, instead of just depending on documentation.
- **Customer collaboration:** Continuous customer interaction is very important to get proper requirements, as the requirements cannot be gathered completely in the beginning of the project due to various factors.
- **Responding to change:** The focus is on quick response to changes and continuous development in agile development.

4.1.1 Agile v/s traditional SDLC Models

Traditional SDLC models are based on a predictive approach. Teams in traditional SDLC models usually work with detailed planning and have a complete view of the exact tasks and features to be delivered during the product life cycle. These models entirely depend on the requirement analysis and planning done in the beginning of the product life cycle. Any changes to be done go through a strict change control management and prioritization. [2]

On the other hand Agile uses an adaptive approach where there is no detailed planning and clarity of what features need to be developed in future. The team adapts to the changing product requirements dynamically. Agile has minimum failure as the product is tested frequently through the release iterations.

The backbone of agile methodology is customer interaction and open communication with minimum documentation are the typical features of agile. The teams in agile methodology are in close collaboration with each other and are often based in the same geographical location.

Pros

- Agile is a very realistic approach to software development.
- Functionality can be developed, rapidly demonstrated and promotes teamwork and cross training.
- Resource requirements are minimum.
- It delivers early partial working solutions and is suitable for fixed or changing requirements.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed and little or no planning required.
- Gives flexibility to developers.

Cons

- Not suitable for handling complex dependencies and more risk of sustainability, maintainability and extensibility.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if the customer is not clear, the team can be driven in the wrong direction.
- Has a very high individual dependency, since there is minimum documentation generated and transfer of technology to new team members may be quite challenging due to lack of documentation.

4.1.2 Why Agile?

Flexibility and time management are important factors to consider when working on projects. Working iteratively, as Agile necessitates, allows a project team to

deliver operational solutions in incremental builds. Each incremental change made to the code repository is thus maintained, and product releases are thus ensured to have minimal errors. Furthermore, tests are also performed prior to every product release, which help find bugs and mitigate errors which could have crept into the product during development.

It is also important to note that the JAMstack, which is the technology stack used to implement the project, has features which include loose-coupling between modules, and a very wide community of developers. Each module, called a microservice, can be easily maintained by individual developers. Since each microservice delivers a limited set of functions, documentation for each module can be minimised. JAMstack approaches project development in a similar way to Agile, and thus can be a perfect fit.

4.2 CI/CD pipeline

A CI/CD pipeline automates the software delivery process. The pipeline builds codes, runs test (CI), and safely deploys a new version of the application (CD).

4.2.1 CI and CD

CI stands for Continuous Integration. It is a development process in which the developers merge their code changes multiple times a day in a central repository. With CI, each change in code triggers an automated build and test sequence.

CD stands for Continuous Delivery which on top of continuous integration adds the practice of automating the entire software release process. CD includes infrastructure provisioning and deployment which may be manual and consists of multiple stages. [3]

4.2.2 Elements of CI/CD

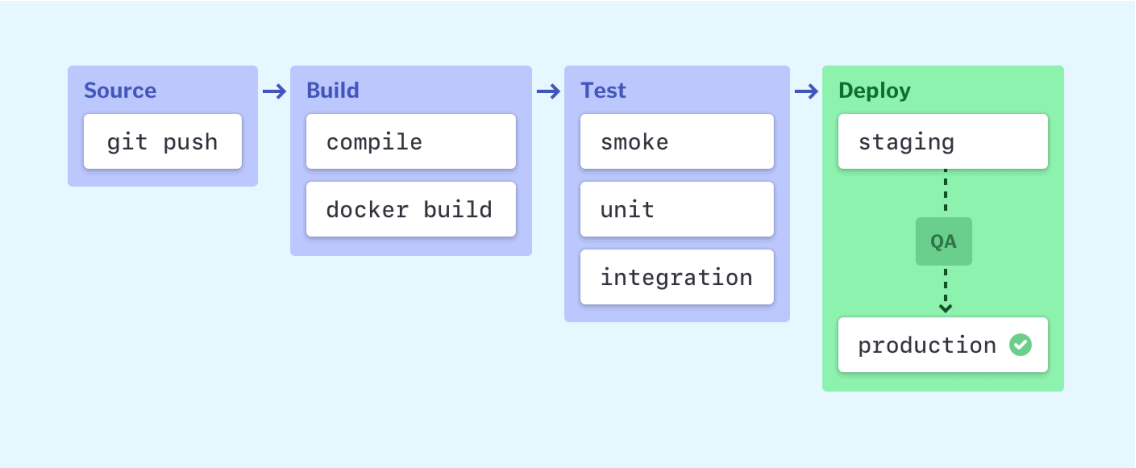

Figure 4.2: Elements of CI/CD

Source Stage

Source repository triggers a pipeline run. A notification is triggered to the CI/CD tool which runs the corresponding pipeline if there is a change in code.

Build Stage

To build a runnable instance of the product that which can potentially be shipped to end users the source code and its dependencies will need to undergo a build stage. Programs written in languages such as Java, C/C++, or Go need to be compiled, whereas Ruby, Python and JavaScript programs work without this step.

Regardless of the language, cloud-native software is typically deployed with Docker, in which case this stage of the CI/CD pipeline builds the Docker containers.

There are fundamental problems in the project's configuration if the project fails to pass the build stage, and it's best to address the problems immediately.

Test Stage

Automated tests can be run to validate the code and the behaviour of the product. This stage acts as a safety net that does not allow the bugs to reach the end-users.

This stage can last from seconds to hours depending on the complexity of the project.

Failure in this stage shows the errors in the code that developers didn't foresee when writing the code.

Deploy Stage

Once the runnable instance of the code has passed all the test stages, it is ready to deploy. There are multiple deploy environments for example "beta" environment for the product team and "production" environment for end-users.

Teams that have embraced the Agile model of development guided by tests and real-time monitoring usually deploy work-in-progress manually to a beta environment for additional manual testing and review, and automatically deploy approved changes from the master branch to production.

4.3 JAMstack

4.3.1 What is JAMstack?

Jamstack is an architecture designed to make the web faster, more secure, and easier to scale. It builds on many of the tools and workflows which bring maximum productivity.

It adheres to the following components : Javascript, APIs and Markup.

The core principles of pre-rendering and decoupling, enable sites and applications

to be delivered with greater confidence and resilience than ever before.

4.3.2 Why use JAMstack?

These are the core concepts that makes JAMstack deliver fast, secure, high-performing, resilient, and efficient infrastructure:

Speed

Serving JAMstack apps as static files directly from a Content Delivery Network makes it likely apps will load faster. Therefore the server need not spend time building the page before responding; all as a result of pre-rendering.

Cost

More often than not, JAMstack sites are going to run cheaper than their server side counterparts. Hosting static assets is cheap and now the pages are being served at the same rate.

Scalability

Since the files are being served off of static hosting, likely a CDN, that pretty much automatically gives infinite and inexpensive scalability.

Maintenance

The static site isn't hosted on a server, meaning hardware maintenance is offloaded. Static HTML, CSS, and JS are maintained headache-free on an automated, distributed CDN system.

Security

With JAMstack, there is no need to lockdown all the services as a central organisation, instead, each service can be locked in separate containerised services, leading to easier management and isolation.

4.3.3 A Comparison with the Traditional Web Architecture

Jamstack is the new standard architecture for the web. Using Git workflows and modern build tools, pre-rendered content is served to a CDN and made dynamic through APIs and serverless functions. Technologies in the stack include JavaScript frameworks, Static Site Generators, Headless CMSs, and CDNs. Refer to figure 4.3

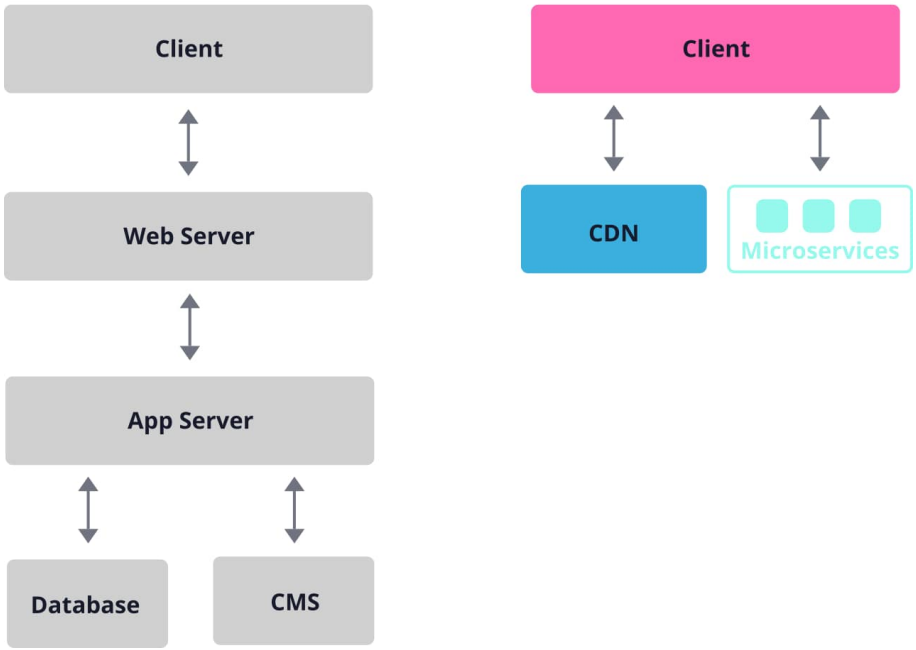

Figure 4.3: Traditional web v/s JAMstack

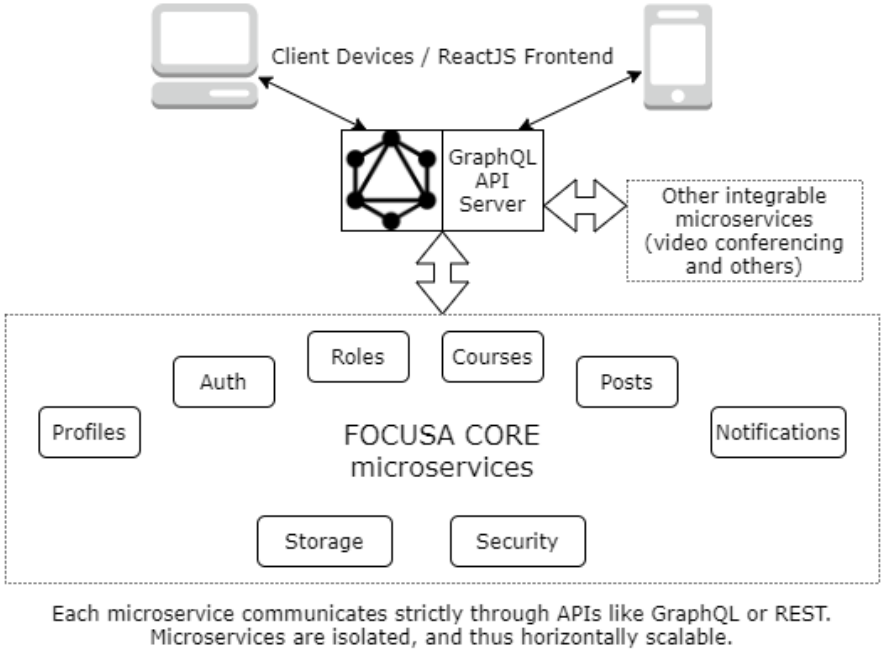

Figure 4.4: High Level Diagram illustrating all microservices of the project.

4.4 High Level Systems

Microservices, also known as the microservice architecture, is an architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. The benefits of decomposing an application into different, smaller services are:

- **Modularity:** This makes the application easier to understand, develop, test, and become more resilient to architecture erosion.
- **Scalability:** Since microservices are implemented and deployed independently

of each other, i.e. they run within independent processes, they can be monitored and scaled independently.

- **Integration of heterogeneous and legacy systems:** Microservices are considered as a viable mean for modernizing existing monolithic software application.

- **Distributed development:** It parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently. It also allows the architecture of an individual service to emerge through continuous refactoring.

The project, FOCUSA includes the following microservices. Refer to figure 4.4:

- **Auth:** performs the authentication process based on a username and password, and returns refresh cookies with a JWT.

- **Roles:** returns the roles that a user belongs to. It depicts the different roles that different admins perform.

- **Profiles:** Each user has a unique profile, containing details like the user display picture, username, and the various courses the user has subscribed to.

- **Courses:** returns the details of the courses which includes the course title and the different users subscribed to the course, as well as a provision for new users to subscribe to the course. Only users with certain roles can moderate their respective courses.

- **Posts:** refers to posts posted by the course moderators. The moderators are able to perform CRUD operations on these posts

- **Notifications:** pushes notifications to the user devices and also monitors the database for changes.

4.4.1 Using GraphQL with the Microservices

The projext will make use of a single GraphQL Schema as an API Gateway to all the microservices, integrating it under a single application. This enables easy integration of data from the different services.

One of the main benefits of having everything behind a single endpoint, is that data can be routed more effectively than if each request had its own service.

The microservices handle the business logic themselves, while the GraphQL platform interacts with the clients to process the queries by consulting the services using REST API, thus allowing for easier isolation and horizontal scaling.

4.5 Activity Diagram

Refer to figure 4.5. The user logs in to the app by putting in the valid credentials. The credentials put in by the user are validated by the system through the authentication mechanism on the server. If the credentials are found to be incorrect, the system runs the error handler to address the error occurred. If the user still is not able to produce the correct credentials, the process ends and the user is not allowed to use the app.

If the user has been validated and is found to be genuine, is directed to the home/main page of the app where he can choose to request for the News Feed, User Profile, Courses or launch video conferencing. All these requests can be made through the navigation panel provided in the app and are accessed as parallel activities.

Once the user requests for a service, the request is sent to the system as a query which is then resolved by the GraphQL to extract the specific requested data from the backend and the response returned is then displayed to the user in structured format.

If the user requests to launch the video conferencing, the system estblishes a peer

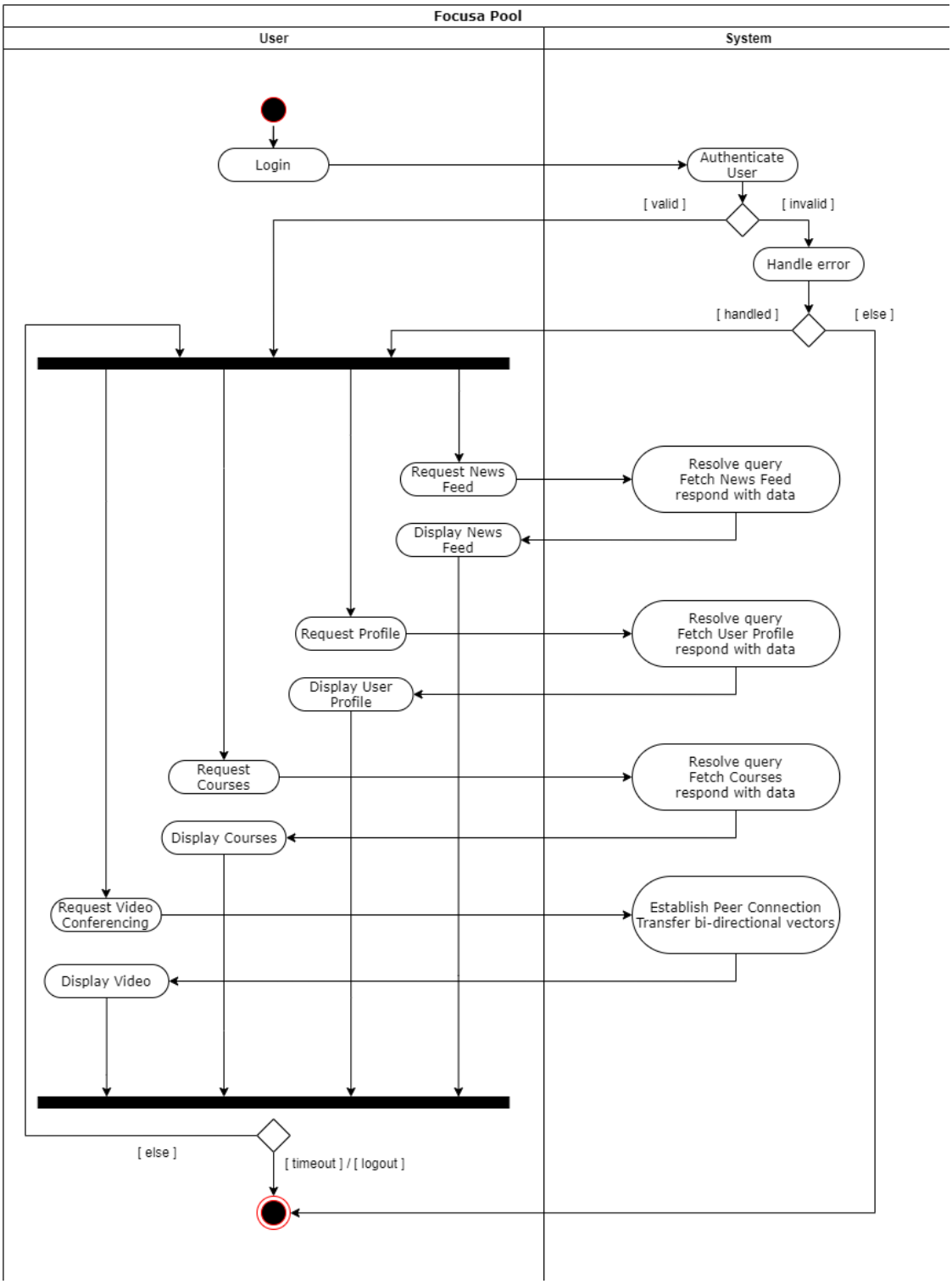

Figure 4.5: Activity Diagram

to peer connection with the requested user system, asks the user to grant access to the camera and audio and starts transfering bi-directional vectors as the secure connection is established. These bi-directional vectors are worked upon by the algo-

rithm used to reduce the bandwidth usage and the output is displayed in the form of video.

If the user decides to log out of the app, it can be accomplished via the corresponding UI component and as a result all the parallel activities combine to close and the user is logged out of the app safely . The user is also logged out of the app the session times out.

4.6 Database Schema Diagram

A database schema represents a **blueprint** or **architecture** of how the data will look. It describes the shape of the data and how it might relate to other tables or models. Refer to 4.6.

The **Collections** and their respective **Document Attributes** are:

1. **User**, which describes user details needed for authentication.
2. **Posts**, which desribes attributes like text, course, timestamp, URL attachment and flags for reported and approved.
3. **Courses**, which includes moderator roles, and describes information which will be displayed to the user, like the course name and description.
4. **Roles**, which is collection of roles which any user can have.
5. **Profile**, which describes all the extra user details like a full name, display picture, about paragraph, and a list of interests.

The core **relationships** between the attributes are:

1. Each user publishes many posts. One-to-many relationship.
2. Many users have many roles. Many-to-many relationship.

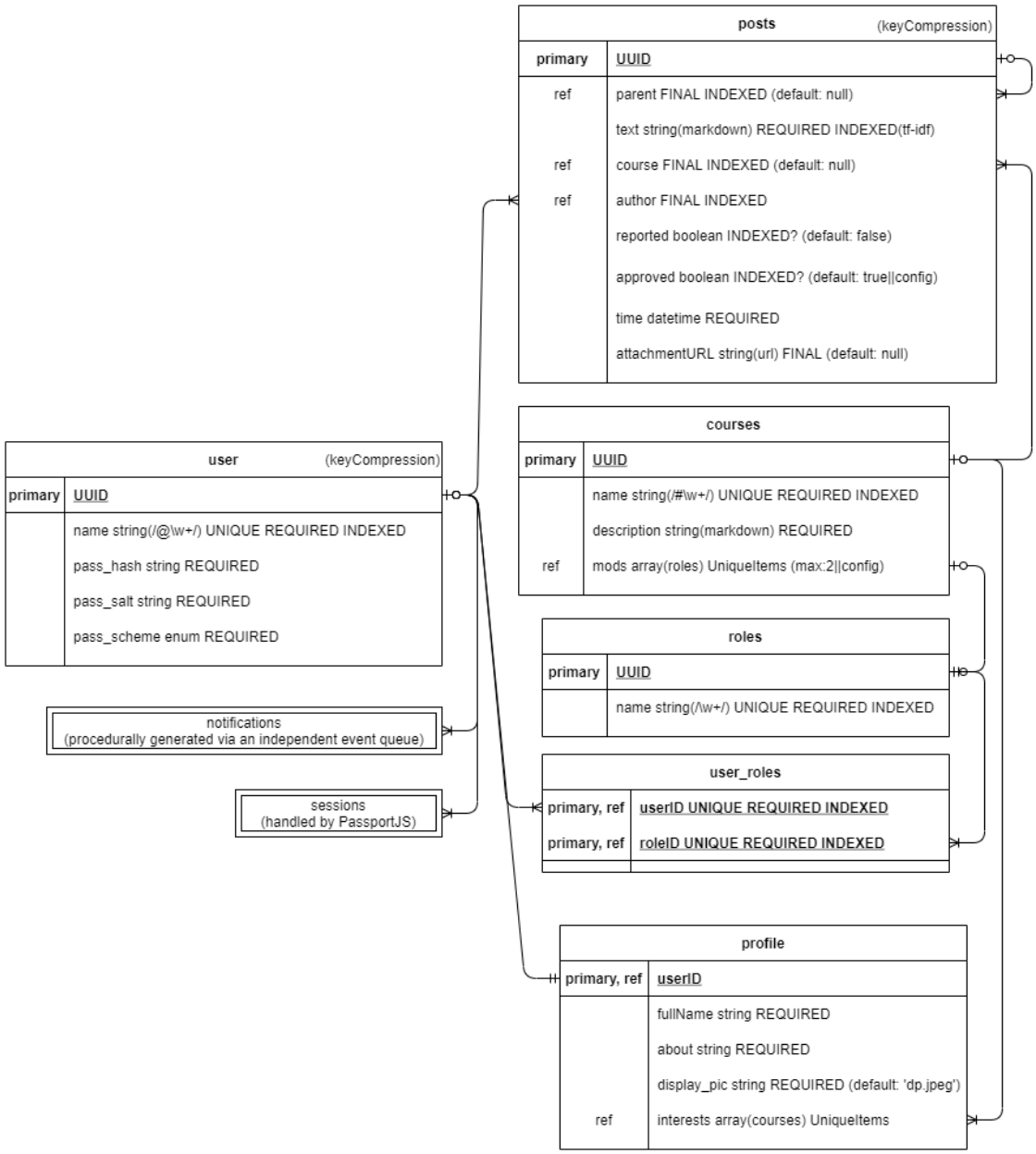

Figure 4.6: The Database Schema Diagram illustrates the Collections and document schemas, along with relationships between them.

3. Each user has a profile. One-to-one relationship with total cardinality.

4. Many comment posts have one parent post. Many-to-one relationship.

5. Each post belongs to one course. Many-to-one relationship.

6. Each course is moderated by many roles. One-to-many relationship.

7. Each profile subscribes to many courses. One-to-many relationship.

The Posts collection functionally stores all posts and comment documents as simple trees. This data structure can easily support and manage a hierarchy similar to tweets in Twitter. However, the moderators will be able to setup a limit to the depth of comments.

4.7 Sequence Diagram

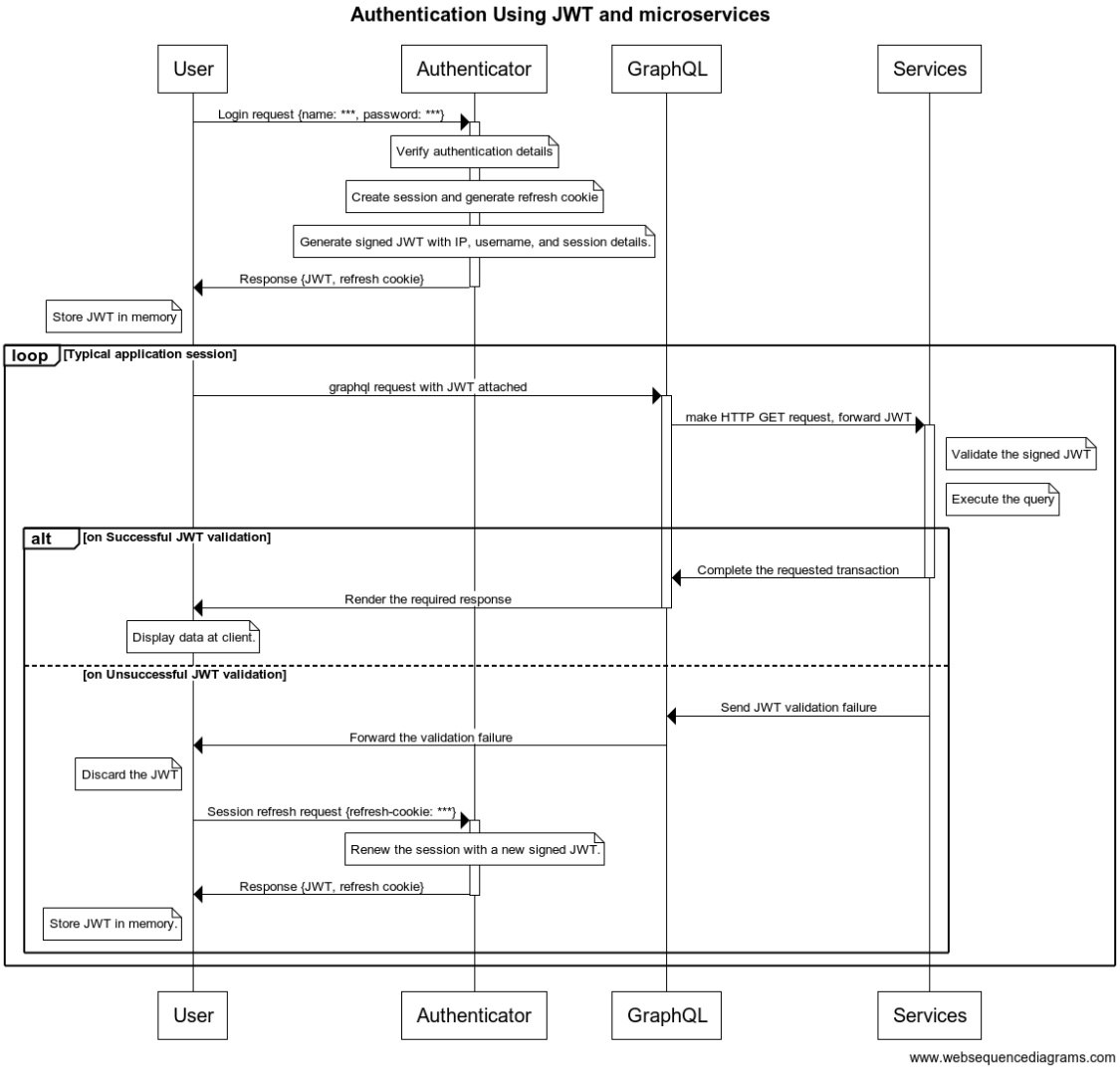

Figure 4.7: Authentication Sequence Diagram. This diagram illustrates four actors: client, auth, GraphQL, microservice.

Refer to figure 4.7. The User initiates with a request to login to the Authenticator along with their credentials ie; name and password. On receiving the credentials, the Authenticator verifies them, creates a session and generates a refresh cookie.

Auth also generates a signed JWT(JSON Web Token) conatining the IP, username and session details of the User.

This JWT is stored in the User's device memory. This JWT is synonymous to an ID card which can be used by the User to directly work with the GraphQL and Services without needing to disturb the Authenticator repeatedly.

The User can now communicate with GraphQL by sharing the JWT. GraphQL makes a request and forwards JWT to the Services. Services here refer to all the microservices that form the system. The Services validate the signed JWT received from the GraphQL (by checking its digital signature) and if found valid execute the query and return response data to GraphQL. GraphQL then converts the recevied response into the format requested by the User, and sends it to the User.

However, if the Services find the JWT to be invalid, a failure response is sent to GraphQL which then returns a failure response to the User. The User now needs to request the Authenticator for a session refresh upon which the Authenticator returns a new signed JWT along with the refresh cookie.

If the User accidentally refreshes their browser, the JWT will get deleted from memory. The User will then request for a session refresh, and a new signed JWT will be generated.

On logout the JWT is simply discarded by deleting the object from memory.

Bibliography

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

[2] Agile SDLC model, https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm.

[3] CI/CD pipeline, https://semaphoreci.com/blog/cicd-pipeline.

[4] Comfort noise, https://en.wikipedia.org/wiki/Comfort_noise.

[5] CouchDB guide, https://guide.couchdb.org/draft/consistency.html.

[6] Developers REST, https://www.sitepoint.com/developers-rest-api/.

[7] Geeks for geeks REST, https://www.geeksforgeeks.org/rest-api-introduction/#:~:text=REST%20API%20is%20a%20way,way%20without%20having%20any%20processing.&text=All%20communication%20done%20via%20REST,POST%20or%20PUT%20or%20DELETE.

[8] GraphQL vs REST, https://www.howtographql.com/basics/1-graphql-is-the-better-rest/.

[9] Hasura offline-first guide, https://hasura.io/blog/design-guide-to-offline-first-apps/#conflicts.

[10] Matrix orf the open protocol, https://www.reddit.com/r/privacy/comments/da219t/im_project_lead_for_matrixorg_the_open_protocol/.

[11] Matrix, https://matrix.org.

[12] NVIDIA Maxine, https://developer.nvidia.com/maxine.

[13] PeerJS simplifies WebRTC, https://peerjs.com/#:~:text=PeerJS%20simplifies%20WebRTC%20peer%2Dto,connection%20to%20a%20remote%20peer.

[14] PouchDB guide, https://pouchDB.com/learn.html.

[15] Rubrik GraphQL, https://www.rubrik.com/en/blog/technology/19/11/graphql-vs-rest-apis.

[16] StackOverflow complexity of Apache Lucene search. https://stackoverflow.com/questions/12107527/complexity-of-a-lucenes-search/12213375#12213375.

[17] Taming WebRTC with PeerJS, https://www.toptal.com/webrtc/taming-webrtc-with-peerjs.

[18] TF-IDF ML tutorial. https://sci2lab.github.io/ml_tutorial/tfidf/.

[19] Ultimate guide WebRTC, https://www.frozenmountain.com/ultimate-guide-to-webrtc.

[20] What is GraphQL, https://www.redhat.com/en/topics/api/what-is-graphql.

[21] What is REST, https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[22] Wikipidea WebRTC, https://en.wikipedia.org/wiki/WebRTC.

[23] TF-IDF example, https://courses.cs.washington.edu/courses/cse373/17au/project3/project3-2.html, 2017.

[24] Confluence. CouchDB introduction, https://cwiki.apache.org/confluence/display/COUCHDB/Introduction.

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

[26] Pubkey. RxDB github repository, https://github.com/pubkey/rxdb.

[27] Pubkey. RxDB replication, https://rxdb.info/replication-graphql.html.

[28] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2, 2019.

[29] Ruma. Matrix, https://www.ruma.io/docs/matrix/.

[30] Ruma. Why Matrix?, https://www.ruma.io/docs/matrix/why/.

[31] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018.

[32] Liang Wang, Paul Grubbs, Jiahui Lu, Vincent Bindschaedler, David Cash, and Thomas Ristenpart. Side-channel attacks on shared search indexes, https://www.cs.princeton.edu/~lw19/pub/sp2017.pdf, 2017.