

# **Video Streaming Application – Vidhyapak**

**A Project Report**

***Submitted By:***

**Jainam Patel (1641006)**

**Het Jagani (1641007)**

**Hitarth Panchal (1641034)**

***in partial fulfillment for the award of the degree***

***of***

**BACHELOR OF TECHNOLOGY**

**IN**

**INFORMATION AND COMMUNICATION TECHNOLOGY (ICT)**

***at***



**Ahmedabad  
University**

**School of Engineering and Applied Science (SEAS)**

**Ahmedabad, Gujarat**

**APRIL 2020**

## DECLARATION

I hereby declare that the project entitled “**Video Streaming Application - Vidhyapak**” submitted for the B. Tech. (ICT) degree is my original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.



**Jainam Patel**



**Het Jagani**



**Hitarth Panchal**

**Place:** School of Engineering and Applied Science, Ahmedabad University

**Date:** 30/04/2020

## **CERTIFICATE**

This is to certify that the project titled “**Video Streaming Application - Vidhyapak**” is the bona fide work carried out by Het Jagani, Hitarth Panchal and Jainam Patel, students of B.Tech (ICT) of School of Engineering and Applied Sciences at Ahmedabad University during the academic year 2019-2020, in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (Information and Communication Technology) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

**Signature of the Guide**

**Place:** School of Engineering and Applied Science, Ahmedabad University

**Date:** 30/04/2020

## **Abstract**

This project is motivated by the current Gyapak system that is being used in the SEAS campus. The proposed system will help students to access the video lectures uploaded by the professors. We intend to develop a web application that can be accessed within the campus. Microservice based architecture will be utilized as a fundamental structure of our web application. Idea of such style is derived from Service Oriented Architecture (SOA). By following such system architecture can help build an efficient software, efficient in the aspects of cloud computing like availability, load balancing, platform independent, caching, fault tolerance, security, scalability, etc. The whole software is developed as a bunch of services, which works independently and communicates with each other to serve the users. We have developed such five services to handle various domain specific tasks. Implementation of them will be such that it imitates some of the cloud computing features. All these services will be wrapped inside the Docker Containers for the deployment. Each of them will run on different host machines, also multiple instances of the same service will be executed. Every implementation detail and the way the software is engineered, from software development to software deployment (delivery), everything boils down to implement above mentioned key features to build a performance-oriented software.

## Acknowledgement

We wish to thank **Dr. Sanjay Chaudhary** for his kind support and valuable guidance throughout the project. We would also like to thank our BTP coordinator **Dr. Vinay Vachharajani** for managing and helping to present our project in the time of COVID19 pandemic.

Without their support this project would not have been possible

**Jainam Patel** (1641006)

**Het Jagani** (1641007)

**Hitarth Panchal** (1641034)

## List of Figures

1. System Architecture Diagram	11
2. Use Case Diagram	12
3. Sequence diagram showing authentication done by user	13
4. Sequence Diagram for video upload workflow	14
5. Sequence Diagram for video streaming	15
6. Discovery Service interaction with Discovery Client	16
7. Login screen	19
8. Create Student Account Form	19
9. Home Page with professor folders	20
10. Courses Page with folder for each course	20
11. Videos in course	21
12. More Details for video	21
13. Added Courses List Page	22
14. Form to add course details	22
15. Uploaded Videos List Page	23
16. Form to add video details	24
17. Watch Video Page	24
18. Video Search Page	25
19. Database ER diagram	31

# Table of Contents

Title Page	1
Declaration of Student	2
Certificate of the Guide	3
Abstract	4
Acknowledgement	5
List of Figures	6
<b>1. Introduction</b>	<b>8</b>
<b>2. Literature Survey</b>	<b>9</b>
<b>3. System Architecture Diagram</b>	<b>11</b>
<b>4. Use Case Diagram</b>	<b>12</b>
<b>5. Sequence Diagram</b>	<b>13</b>
5.1. Authentication Sequence Diagram	13
5.2. Video Upload Sequence Diagram	14
5.3. Video Streaming Sequence Diagram	15
<b>6. Services</b>	<b>16</b>
6.1. Discovery Service	16
6.2. User Interface Service	17
6.3. Video Streaming Service	25
6.4. Video Upload Service	26
6.5. Video Processing Service	27
6.6. User Authentication Service	28
6.7. User Information Service	29
<b>7. Database</b>	<b>30</b>
<b>8. Deployment</b>	<b>32</b>
8.1. Docker Swarm	34
8.2. Kubernetes	37
<b>9. Future Directions</b>	<b>39</b>
<b>10. References</b>	<b>40</b>

## 1. Introduction

In SEAS we have a centralized platform to access the files that professors upload. It is completely internal in the college network and only a person with college email address can access those files. All the materials for reference or extra materials is put on Gyapak by the professors, so that students can access it and it could help them in studies. This present system requires to upload the file to the folder and then student can view it or download it. Professors generally put slides in the form of PDFs or some reference materials.

In this project an application similar to Gyapak will be developed. As Gyapak is used to access the materials related to courses, this application is intended to stream the video lectures for courses that are taught at the university. The application developed will be web based for easy access to the students. It will be designed and implemented to tackle cloud-based software challenges like security, reliability, scalability and maintainability.

The following features will be designed and implemented as a part of the project. Implementation of these features will lead to the development of the whole application.

- **Scalable Microservice based Architecture**
- **Service Discovery**
- **Fault Tolerance**
- **Security**
- **Availability**
- **Load Balancing**



## 2. Literature Survey

With the advent of Cloud Computing, the most fundamental question that one needs to answer before the development phase of any Software is “Which architecture is best suitable? And why? Back in the days the software used Monolithic Architecture which is an application with a single code base with multiple modules. Now, as the software started getting bigger and bigger, several problems had to be answered like coordination among the development teams, scalability, maintainability, availability, deployment, testing and so on. So, a new approach addressing these problems, called Microservices Architecture was defined. This approach structures the application as a collection of independent services which communication through HTTP resource API.

The main advantages with this approach, as mentioned by S. Newman in his book are the followings. First, microservices can rely on heterogeneity, each service can use different technology compared to other services to achieve desired goals and performance. Second, if one component of the system fails, it does not impact the other services. Third, Scaling is much more accessible and easier as only the services requiring scaling are scaled as compared to monolithic, where the application has to be scaled as a whole unit. Fourth, Deployment is much easier as the services are independent and it has no adverse effect on the performance of other services. Fifth, it will enable organizational alignment as the number of people working on a specific code base decrease [1]. Service discovery is an essential component of any microservices application, as the locations are not assigned at the development stages. Also, it may be deployed on cloud where services could relocate and replicate themselves.

Many researches have been conducted which tried to analyze the performance of Microservices Architecture and Monolithic Architecture. A research conducted by Singh and Peddoju, to analyze the performance of these architectures, by running tests comprising of 2000 threads, exhibited that microservices has a better performance in terms of throughput when it is used for a larger number of requests [2]. Villamizar et al. concluded that use of microservices can help reduce up to 13.42% of infrastructure costs based on the performance tests done by them. They also claimed that emerging

cloud services like AWS Lambda, which is exclusively designed for deploying microservices, allows companies to reduce the infrastructure costs by 77.08% [3].

Which environment to choose for Microservices based Architecture? Container-based or Virtual Machine-based environment. Virtual Machines provides virtualization of hardware as well as operating system while in Containers, only operating system is virtualized not the hardware. A research on performance comparison between Linux containers and Virtual Machines, authors showed that containers outperformed VMs in terms of performance and scalability. Containers outperformed VMs by 22x in terms of scalability and 5x in processing requests [4]. Many performance comparisons have been reported on virtual machines and containers. Services deployed using containers are expected to take less execution time (less latency) compared to VMs. When using VMs, significant performance overhead was reported which was highlighted in the CPU utilization and memory usage factors [5].

Salah et al. performed an experimental comparison of VMs and Containers on Amazon Cloud Environment and the research highlighted that VMs-based web services outperformed container-based web services on important metrics such as throughput, response time and CPU utilization. They concluded with a point that if the applications require certain performance criteria, the use of EC2 in AWS cloud should be recommended over ECS because Amazon Cloud runs containers on the top of EC2 VMs and not on bare metal physical hosts [6].

### 3. System Architecture Diagram

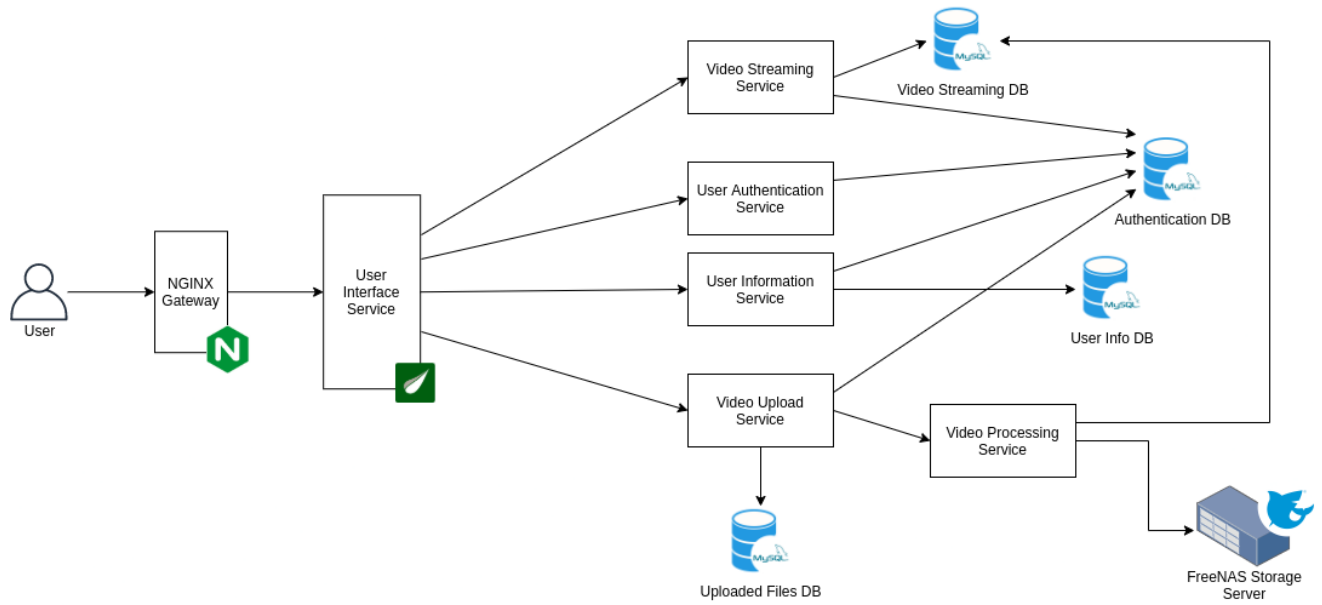


Figure 1: System Architecture Diagram

## 4. Use Case Diagram

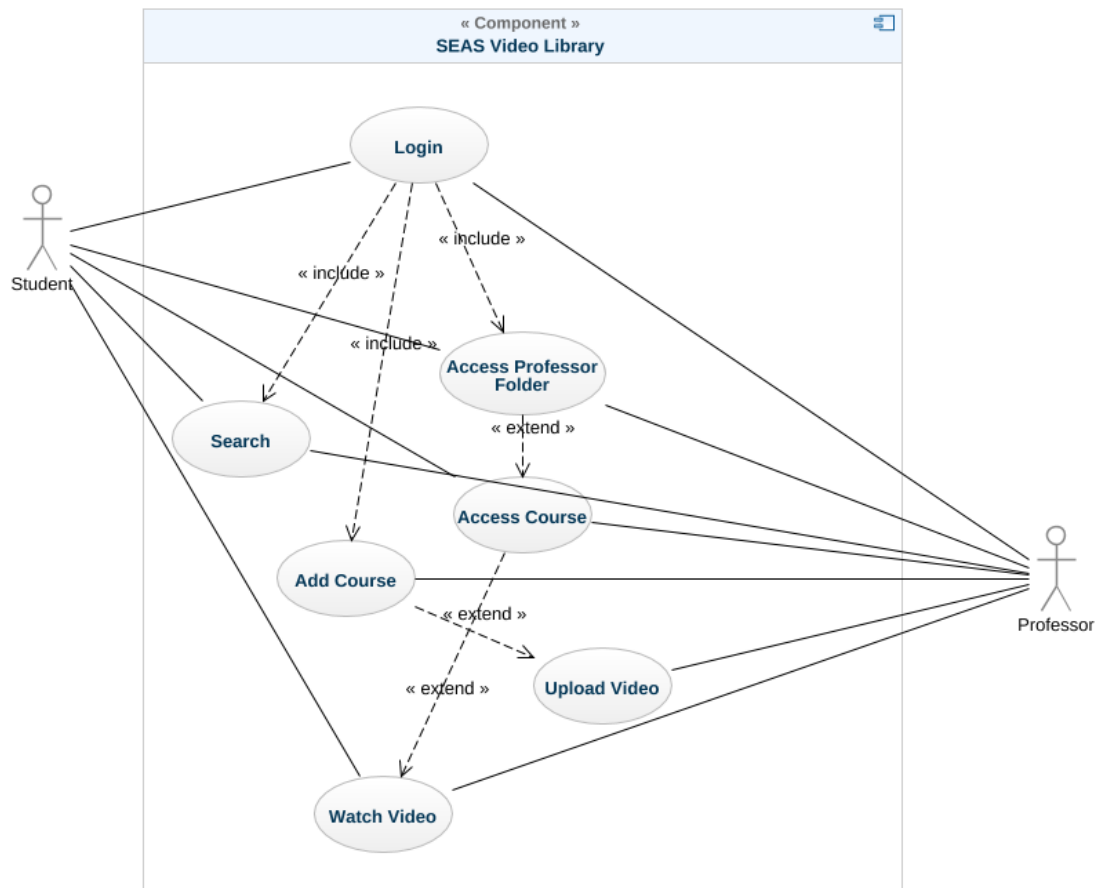


Figure 2: Use Case Diagram

## 5. Sequence Diagrams

### 5.1 Authentication Sequence Diagram

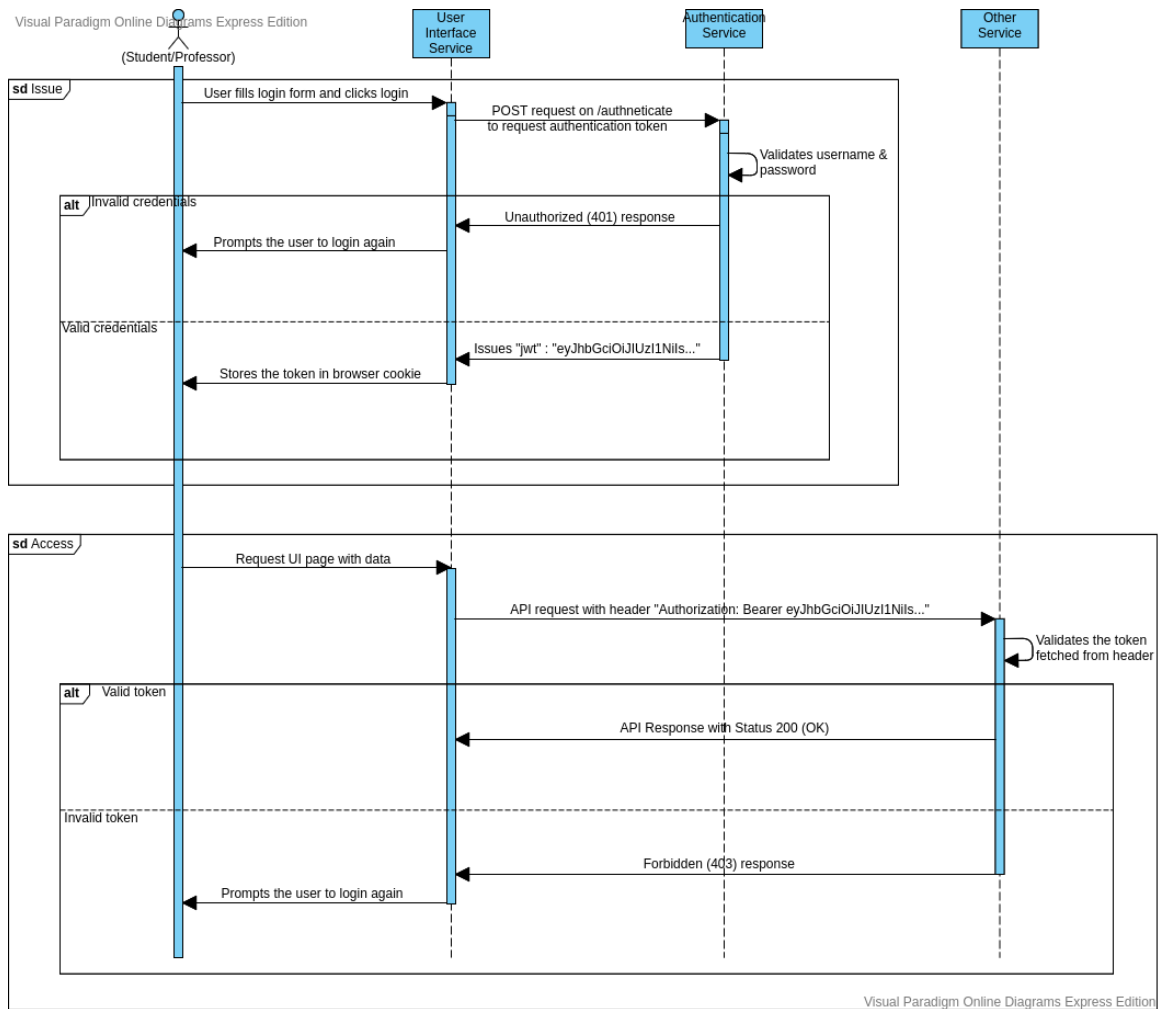


Figure 3: Sequence diagram showing authentication done by user

## 5.2 Video Upload Sequence Diagram

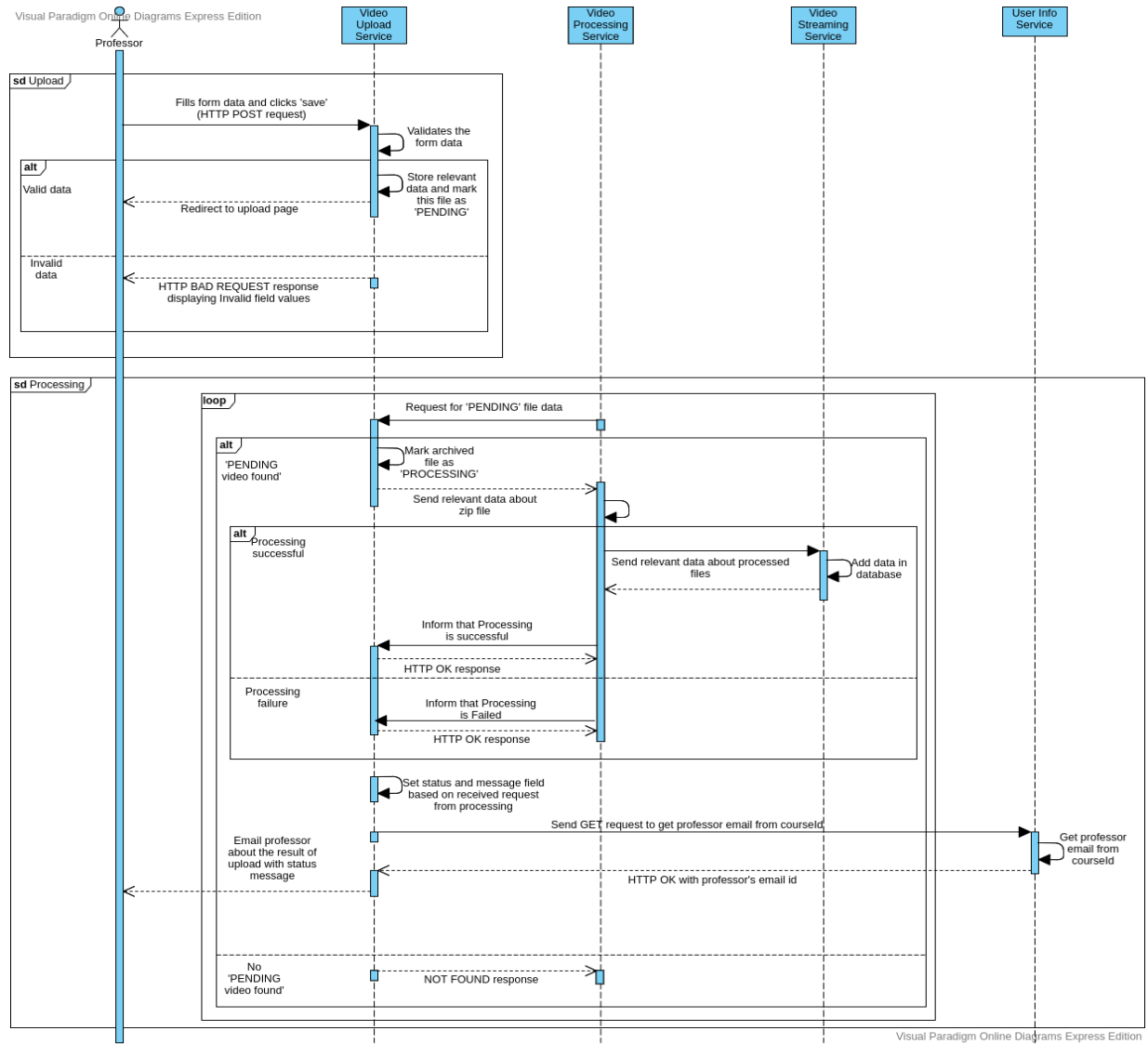


Figure 4: Sequence Diagram for video upload workflow

### 5.3 Video Streaming Sequence Diagram

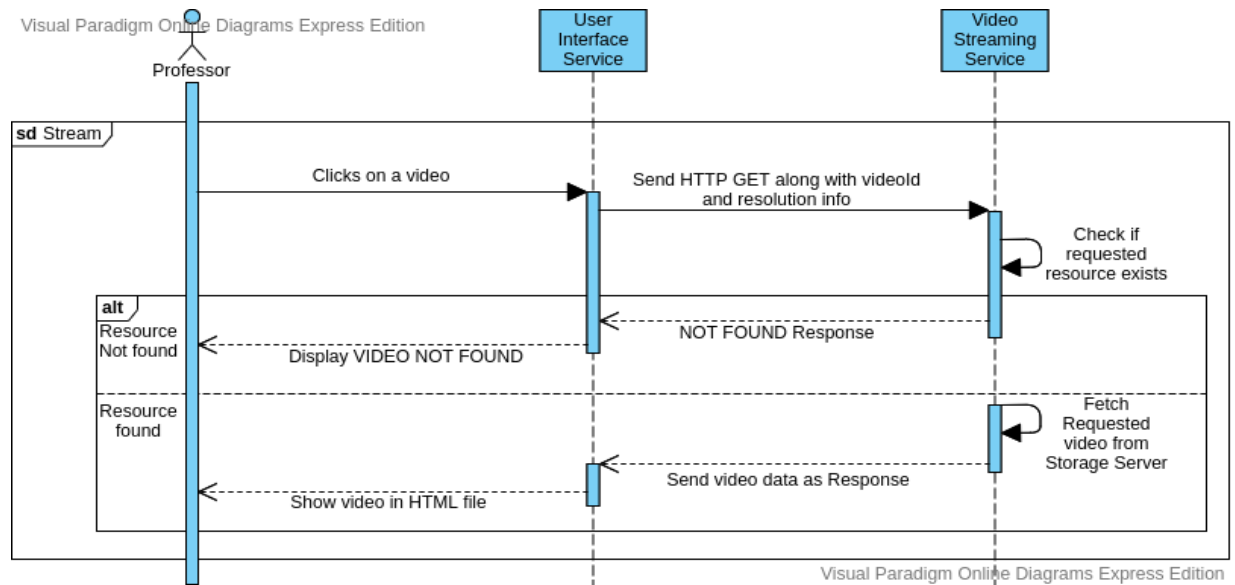


Figure 5: Sequence Diagram for video streaming

## 6. Services

### 6.1 Discovery Service

For the scalability purpose and the service discovery between all the microservices, a discovery service is developed. Whenever a microservice is started it gets registered by discovery service. This is done by periodically sending heartbeat to the discovery service. If the discovery service does not receive the heartbeat from a service then it removes that service from registered list.

Whenever a microservice needs to access any other microservice, it first needs to call the discovery service. The discovery service then redirects the request to the running instance of the requested service. If more than one instances of particular service is running then discovery service redirects the incoming requests in load balanced way. Following diagram shows how the discovery server and client (every microservice) interact:

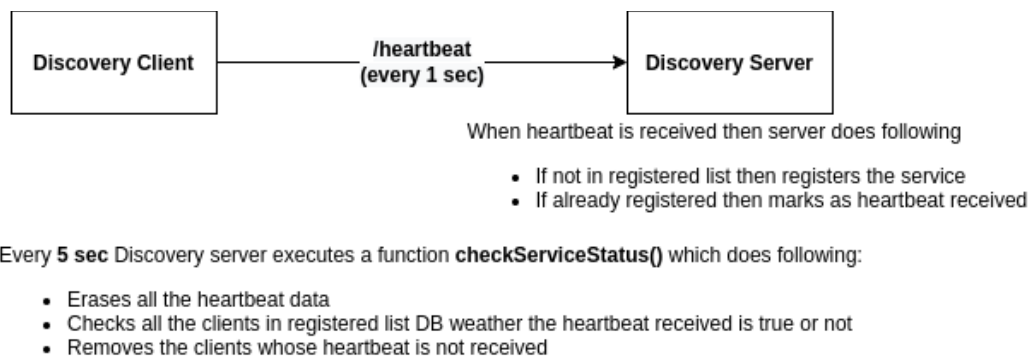


Figure 6: Discovery Service interaction with Discovery Client

As seen in the diagram the discovery client has to send the heartbeat continuously in order to be registered with the discovery server. This heartbeat contains object which carries the client IP address, port number and application name, which will be registered in the database. Also, some parameters such as avg load, number of connections etc. are sent in heartbeat in order to take load balancing decisions by discovery server.

#### Endpoints:



- **POST /discovery/heartbeat** : Endpoint where client service will make post request with all the parameters to register itself.
- **GET /discovery/registeredservices** : Returns list of all the services that registered by discovery server.
- **GET /discovery/serviceinfo/{service name}** : Returns object of service whose name is provided in the parameter. If more than one instances of the service is available then the IP addresses are returned in load balanced way.

## 6.2 User Interface Service

User Interface for the application is implemented as a service. Bootstrap framework is used for styling of web pages. Thymleaf is used as template engine to serve the web pages loaded with the data fetched from backend microservices. The UI service will fetch JWT token from the browser cookies on each page to authenticate the user (See 4.6). Since the UI service will call the backend services to fetch data, hence fault tolerance mechanism needs to be implemented in the UI service, in order to prevent cascading failures. For fault tolerance Hystrix framework is used. Hystrix implements circuit breaking mechanism to serve data from the cache if any backend service is down.

### Fault Tolerance

When one component stops working due to some reason it should not take the whole application down. That's a benefit of following the microservice architecture. While in monolith architecture, it isn't true because all the code is packed into one application. In our case, primary job the software is to stream the videos to users, no matter what it should not stop doing that. In our web interface to watch a video user has to first select a professor, then navigate through the courses of that particular professor and then at last user can get the list of videos of selected course. First two parts of previous statement require to perform following two tasks by the UI service.

- Get the list of all the professors from the User Information Service.

- Upon selection of the professor, get all the course taught by that professor from User Information Service.

Hence when the User Information Service is down, UI service will get **timeout** response. Due to this reason, user won't be able to watch any videos on the software. This should not happen and to avoid such scenarios we have used Hystrix framework, Redis and Jedis. Hystrix is helpful to implement fault tolerance code, Redis is in-memory data structure store to cache important data, Jedis is a Redis client to manipulate data inside the Redis server. Above explained problem is solved using these three tools. Now above-mentioned steps changes to following steps.

- Get the List of all the professors/List of courses for a selected professor from the User Information Service.
- If the response is **timeout (Indicates that User Info Service is in outage)**, read the data from cache (Redis server) and return the data. If required data is not in the cache then show Not Found on the screen.
- When user info service returns a **200** response, UI will add the entry inside the cache if it's not present or if its present in cache it will update the data in the cache then show the result on the screen.

By doing so guarantees the availability of our software to some extent, serving continuously. Showing the software is fault tolerant, resilient and capable to handle some failures. Always delivering its core services to its users.

Following are the webpages in the User Interface service

- **Login Page**

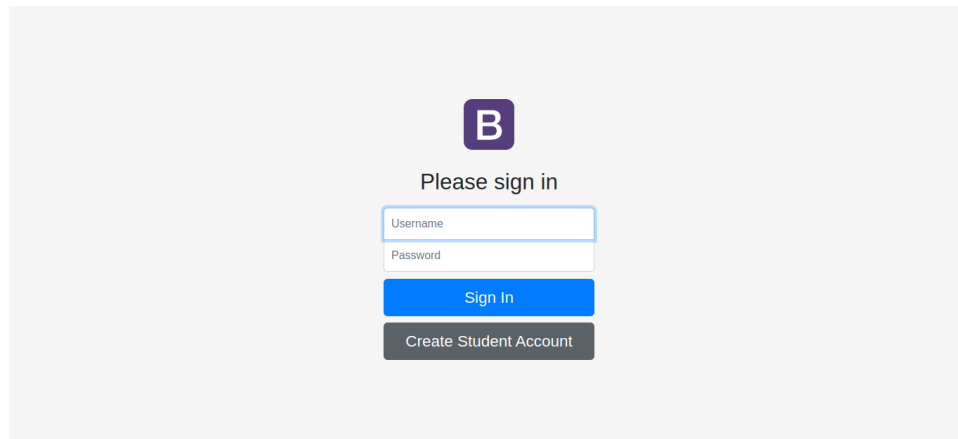
The login screen features a light gray background. At the top center is a purple square icon with a white letter 'B'. Below the icon, the text 'Please sign in' is centered. Underneath this text are two input fields: 'Username' and 'Password'. Below the 'Password' field is a blue button labeled 'Sign In'. At the bottom of the form is a dark gray button labeled 'Create Student Account'.

Figure 7: Login screen

When user clicks “Sign In” the UI service sends the login credentials to authentication service and saves the authentication token provided in browser cookies, if the credentials are valid.

- **Create Student Account**

The form is titled 'Enter Student Details'. It contains several input fields: 'First Name' (placeholder: Enter First Name), 'Middle Name' (placeholder: Enter Middle Name), 'Last Name' (placeholder: Enter Last Name), 'User Name' (placeholder: Enter User Name), 'Password' (placeholder: Enter Password), and 'Date of Birth' (placeholder: dd/mm/yyyy). Below the 'User Name' field, there is a small note: 'You will use this username for login.'

Figure 8: Create Student Account Form

When any user wants to access the application, he/she can create a student account by clicking on create student account on login screen. This will take user to page to fill all the information regarding the account.

- **Home Page**

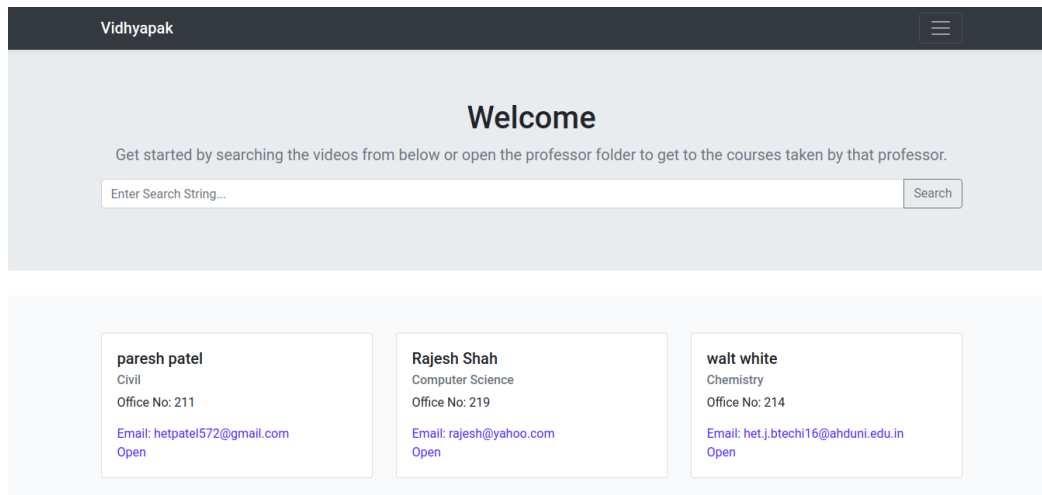


Figure 9: Home Page with professor folders

Homepage consist of the professor's folders. User can access all the courses of particular professor by clicking "Open" link in professor card.

- **Courses Page**

This page displays all the courses of a particular professor. When a professor adds a course, it is displayed on this page to every user.

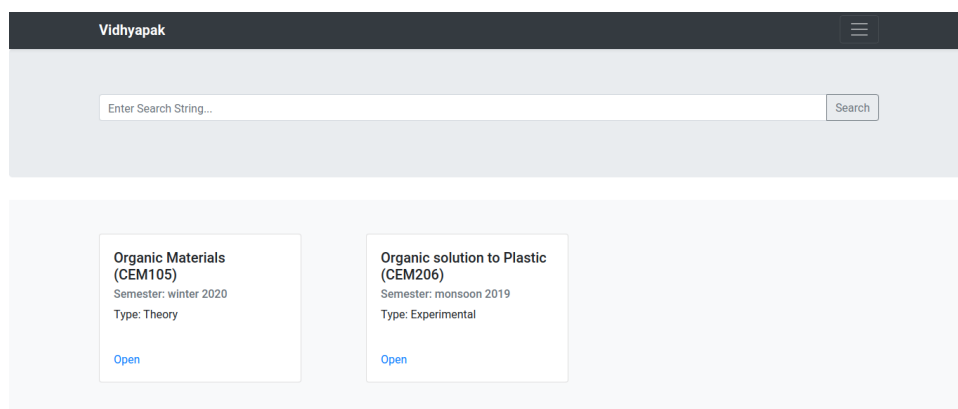


Figure 10: Courses Page with folder for each course

- **Videos Page**

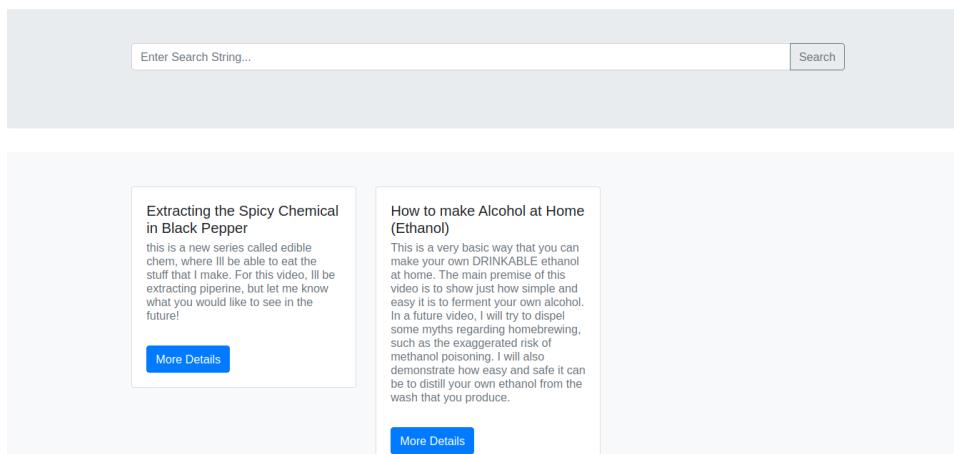


Figure 11: Videos in course

When user clicks a course folder then all the videos inside the course is displayed as shown in the above image. When user clicks on more details a popup is shown which contains video title, description, link to watch that video and attached lecture note materials that user can download.

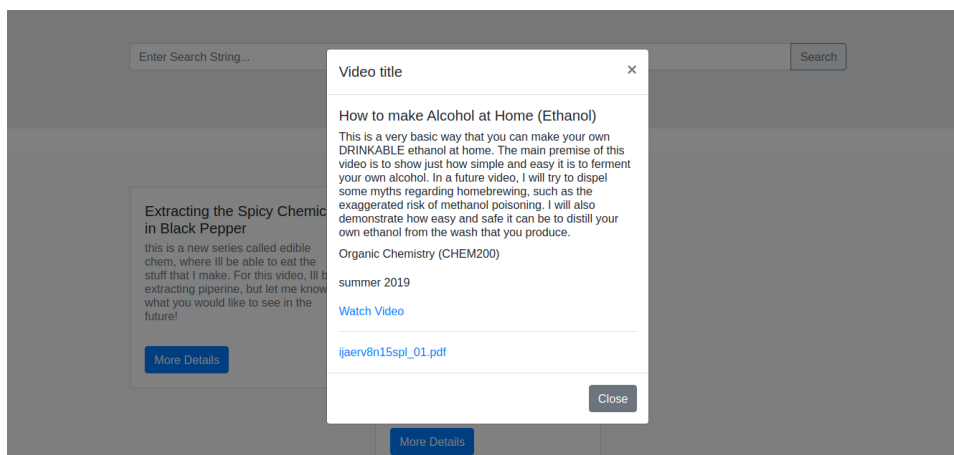


Figure 12: More Details for video

- **Add Course Page**

This page is only available for the professors. Professor can add new course in his/her folder.

Vidhyapak							
List of all the courses							
Id	Course Code	Course Name	Course Year	Course Semester	Course Type	Edit	Delete
3	CEM105	Organic Materials	2020	winter	Theory	<button>Edit</button>	<button>Delete</button>
4	CEM206	Organic solution to Plastic	2019	monsoon	Experimental	<button>Edit</button>	<button>Delete</button>
<button>Add New</button>							

Figure 13: Added Courses List Page

This page displays all the courses that added by the professor who has logged in. Professor can edit course details (Course Code, Name, Year, Semester, Type) by clicking on 'Edit' button. Also, professor can delete the course and add new course by clicking the buttons provided.

If the professor wants to add new course in his/her folder then he/she can do by clicking on "Add New" button. This will open a popup form in which professor can add all the course information and click "Save" to save the course in his/her folder.

Vidhyapak
List of all the courses

Id	Name	Department
3	CEM105	Organic Materials
4	CEM206	Organic solution t

New Course

Course Code:

Course Name:

Course Year:

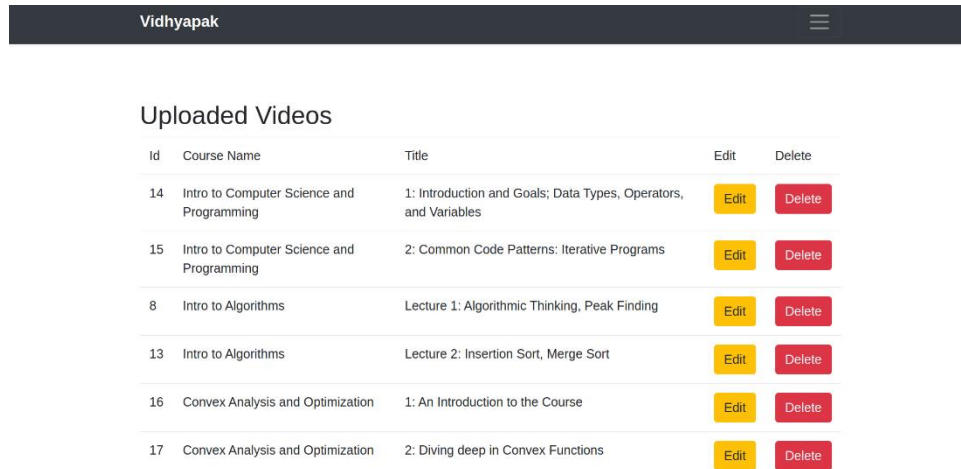
Course Semester:

Course Type:

Close Save

Figure 14: Form to add course details

- **Upload Video Page**



Id	Course Name	Title	Edit	Delete
14	Intro to Computer Science and Programming	1: Introduction and Goals; Data Types, Operators, and Variables	Edit	Delete
15	Intro to Computer Science and Programming	2: Common Code Patterns: Iterative Programs	Edit	Delete
8	Intro to Algorithms	Lecture 1: Algorithmic Thinking, Peak Finding	Edit	Delete
13	Intro to Algorithms	Lecture 2: Insertion Sort, Merge Sort	Edit	Delete
16	Convex Analysis and Optimization	1: An Introduction to the Course	Edit	Delete
17	Convex Analysis and Optimization	2: Diving deep in Convex Functions	Edit	Delete

Figure 15: Uploaded Videos List Page

This page lists all the videos the user (Logged in) previously uploaded. User can change video title and/or description of the video he uploaded by clicking on the 'Edit' button, also he/she can delete that particular video by clicking on the 'Delete' button, as shown in the image. All the videos the user uploaded is part of a course.

If user wants to upload a video, he/she can do so by clicking on 'Add New' button. In that for, user has to enter video title, description, course ID (Id of course to which this video will belong) and attach an archived (ZIP file) file (Which will content a video file and/or document files like PDF, DOC, PPT, etc.). After pressing "Save" button, archived will be uploaded to server and will be queued for further processing.

**Upload Video**

Video Title:

Video Description:

Course ID:

Select file:

Choose file No file chosen

Close Save

Id	Course Name	Edit	Delete
14	Intro to Computer Science Programming	Edit	Delete
15	Intro to Computer Science Programming	Edit	Delete
8	Intro to Algorithms	Edit	Delete
13	Intro to Algorithms	Edit	Delete
16	Convex Analysis and Optim	Edit	Delete
17	Convex Analysis and Optim	Edit	Delete

Add New

Figure 16: Form to add video details

- **Watch Video Page**

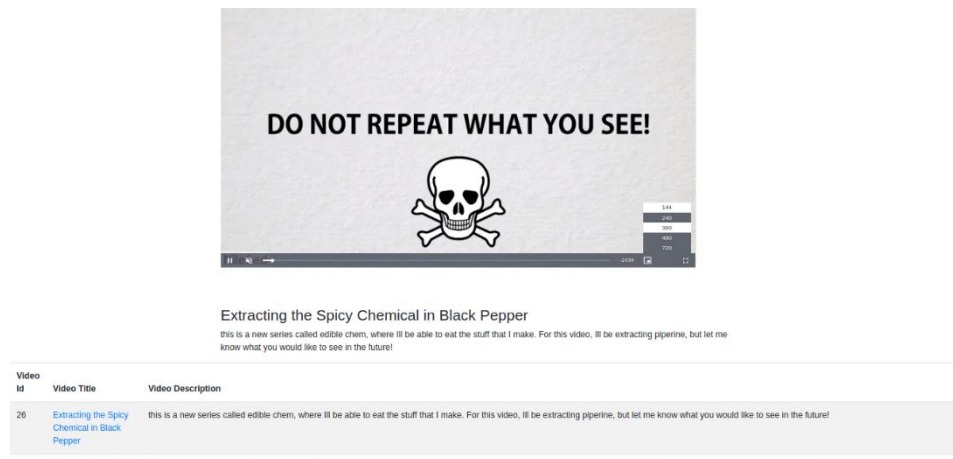


Figure 17: Watch Video Page

To watch video user clicks on the “Watch Video” link given in the “More Detail” popup. This will open a watch video page as shown above. User can select video quality from the bottom left menu as shown above. Also, below video player video details are shown and below that all the videos associated with the course is listed. User can go to any video by clicking on the video title from that list.



- **Search Page**

In every page a search bar is provided. When user provides a search string, this search string is matched with video title and video description. The relevant search results are shown as above. Here list of videos with the course to which they are associated is shown. User can directly watch video by clicking on the video title or he/she can go to the course by clicking on the course name.

Vidhyapak				
Video Id	Video Title	Video Description	Course Name	Course Year
24	<a href="#">001 - Environmental Science</a>	In this video Paul Andersen outlines the AP Environmental Science course. He explains how environmental science studies the interaction between earth and human systems. A planetary boundary model is used to explain the importance of sustainability. The importance of science practices and knowledge of the APES format is also included.	<a href="#">Environmental Science (CIV105)</a>	Semester: winter 2020

Figure 18: Video Search Page

## 6.3 Video Streaming Service

Video streaming service handles the actual streaming of the videos. First of all, it registers all the videos in its database sent by video processing service. There are multiple resolutions available for one video. One video also contains many other document files. All of these details are sent by the processing service (via **HTTP POST** request) and are stored in the database of streaming service.

After receiving the client request to stream video, it first retrieves all the data from database. Client request mentions details like video ID, video byte-range and resolution. Streaming server will only serve that particular byte-range and resolution specified by the client. Same way user can also download (endpoint is mentioned below) any document attached to a particular video by clicking on a link provided in UI. While searching for a video, user query is attached in URI as a request parameter, then at streaming service it's extracted and all the relevant videos are returned.

### Endpoints:

- **GET /stream/video/{id}/res/{res}**: Streams the video with provided id ({id}) in path parameter and the quality of streamed video is also provided as path parameter ({res}).
- **GET /stream/lectureNote/{id}/download**: Downloads the lecture note with id provided in path parameter.
- **GET /stream/video/{id}/details**: Returns the object which contains metadata of video such as height, width, duration etc. Also, it contains the details about the Lecture Notes that are associated with video.
- **GET /stream/course/{courseId}/videos**: Returns the list of video objects which are associated with the course whose id is provided in path parameter.
- **GET /stream/video/search**: Returns the video objects whose title or description matches the given search string as request parameter search.
- **POST /stream/video/register**: Registers the video details of provided video object as body.
- **DELETE /stream/video/{id}**: Deletes the details of video from the database whose id is provided in path parameter.

## 6.4 Video Upload Service

Upload service is responsible for handling upload of zip file. This file will have a video file and all other relevant documents about that particular class (lecture note PDFs, DOCs, PPTs, programming language files, etc.). Upload service stores this zip file to the storage server at some location. And will register all required details about it into the database. All the uploaded zip files have to be processed. So, while registering the details of the uploaded file, upload service marks it 'PENDING' task. 'PENDING' status means that file is yet to be processed. This whole process is performed when a user uploads a video from UI page. All corner cases are successfully handled by this service while performing this task, i.e. When user doesn't attach a file or leaves some fields empty. It will inform the user when the data is empty.

Upload service also listens for incoming HTTP requests from the video processing service for processing of the 'PENDING' videos. When it receives requests, it fetches the earliest entry from its database and returns that entry to the processing service for that zip file to process. And also changes the status of that entry to 'PROCESSING'. Due to various reasons processing might fail, for that the upload service has to be responsible. So, it also listens for PUT request from the processing service. Upon failed or successful processing of the file, processing service sets some fields to let upload service know why it has failed. After receiving that, firstly it changes status (set by the processing service) of that file in the database, secondly it uses course Id to get the email of the professor from User Information Service. And emails him/her about their upload and why it has failed. Also, upon successful processing it also informs the professor that the file has been successfully processed and video is uploaded on the application server. Some of the key endpoints are mentioned below.

#### **Endpoints:**

- **POST upload/video/submit:** Gets the zip file and other data about that from the user.
- **GET upload/pending/path:** Returns the earliest 'PENDING' entry from its database.
- **PUT upload/status/{id}:** Changes the status of a particular (id provided in the URI) entry and informs the corresponding user about the result.

### **6.5 Video Processing Service**

Primary task of this service is to process all the videos and register all the videos to video streaming service. It will execute a function every 5 seconds. This function performs series of operations stated below.

1. Send GET request to the upload service to fetch 'PENDING' file entry to process.
2. Unzip the zip file at a location if file type is not zip set the status field to 'FAILED' and inform the upload service.
3. Go through all the files and count the video files. If the count is 0 or more than 1, set status to 'FAILED' and inform the upload service.

4. Process the video file. Processing includes the converting the input video file to different video resolutions and also changing the video codec. All the video resolutions and video codec used, are following standards.
5. Upon successful execution of all the above steps it will send POST request to video streaming service with all the details about the processed file. This file includes one video file and other document files (PDFs, DOCXs, etc.) with it, If the response from streaming service is **200 OK** then set status to 'ARCHIVED' and inform the upload service.

A key detail is when its processing a file the timer is paused. Reason being video processing utilizes all the CPU cores up to 99%, so in that state it can't handle another file. Basically, processing service will fetch path of pending video, process the file, send relevant data to other services, wait for 5 seconds and repeat. This service's sole purpose is to just process the videos so no endpoints are exposed by it. Another key detail is, there will be multiple instances of this service will be running to handle processing of all the queued videos quickly.

When it comes to streaming videos in the browser, selection of appropriate video codec and audio codec becomes necessary. Because [not all browsers support all the video and audio codecs](#). So here we have selected [H.264](#) video codec, [AAC](#) audio codec and MP4 file format. This combination is supported in all widely used browsers including Chrome, Firefox, Safari, Opera. Including both mobile and desktop versions of them. Also, conversion to different video resolutions is done and [this](#) standards are followed for that sake. Max quality for a video is Full HD (1920x1080) depending on the source video resolution. Source video could be in any video codec, audio codec and file format and the output video should follow above mentioned standards. So, for conversion [FFMPEG \(Command line utility\)](#) is used, this process is called video transcoding (encoding + decoding), doing so is out of our scope.

## 6.6 User Authentication Service

The authentication of users in the application is done by distributing JWT (Json Web Tokens). Whenever a user wants to access the application than first it needs to authenticate from the authentication service. When user performs login, the user credentials are passed to the authentication service which checks the database and validates the credentials. If the credentials are valid then authentication service issues a JWT which is stored in the UI service's database. This JWT will have expiry

time of 24 hours after which it is expired and user needs to login again. All the services will have a http filter which will check for JWT in incoming request's "Authorization" header. If valid JWT is found then only the service will be accessed otherwise the user will be redirected to login again.

Also, besides the authentication the service database also contains the information about the authorization roles. Hence every microservice is configured with particular access to these roles. There are total three roles for authorization: ROLE STUDENT, ROLE PROFESSOR and ROLE ADMIN. These roles are used to authorize the service endpoints.

#### Endpoints:

- **POST /authentication/authenticate:** Returns the JWT in response if the username and password provided in the body are valid.

### 6.7 User Information Service

All the user data is stored in the UserInfo database. All the details about the students, professor, courses etc. are stored in the User Info database. To access the database all the services need to call the User Information Service. This service accesses the database and returns the requested data.

#### Endpoints:

- **GET /person/all:** Returns list of all the person objects.
- **GET /person/{username}:** Returns the person object whose username is provided in path parameter.
- **GET /person/usernameExist:** Returns true/false based on whether username exists in database or not.
- **GET /professor/all:** Returns list of all professor objects.
- **GET /professor/{username}:** Returns the professor object whose username is provided in path parameter.
- **POST /professor/save:** Saves the professor object provided in body.
- **PUT /professor/update:** Updates the professor object with the object provided in body.

- **DELETE /professor/remove:** Removes the details of professor whose username is provided as request parameter.
- **GET /student/all:** Returns list of all student objects.
- **GET /student/{username}:** Returns the student object whose username is provided in path parameter.
- **POST /student/save:** Saves the student object provided in body.
- **PUT /student/update:** Updates the student object with the object provided in body.
- **GET /course/all:** Returns list of all course objects.
- **GET /course/code/{courseCode}:** Returns the course object whose course code is provided as path parameter.
- **GET /course/professor/{username}:** Returns list of courses that are added by professor whose username is provided in path parameter.
- **POST /course/save:** Saves the course object provided in body.
- **PUT /course/update:** Updates the course object with the object provided in body.
- **DELETE /course/remove:** Removes the details of course whose id is provided as request parameter.

## 7. Database

As we are developing the application with microservice architecture, the database cannot be shared between the services. A monolith database must be broken down to facilitate the individual development of every service. Each microservice encapsulates a specific business capability and its own data. The monolithic database decomposes into a distributed data model with many smaller databases, each aligning with a microservice. Below image shows the whole database ER diagram, broken down for different microservices.

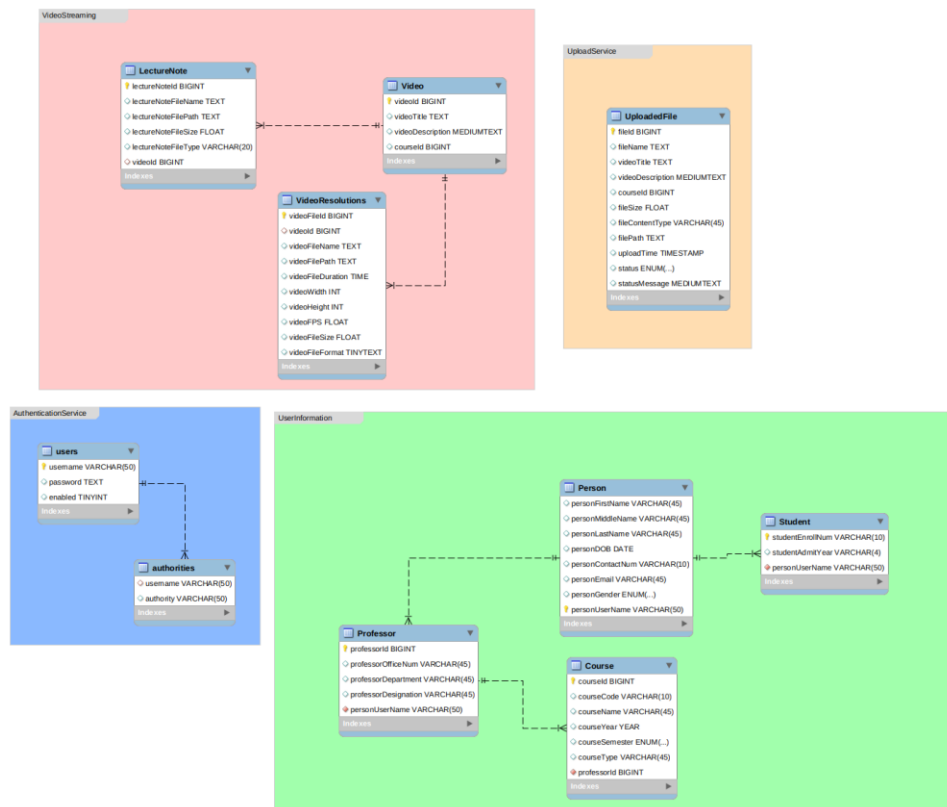


Figure 19: Database ER diagram

A MySQL database server is setup in one machine which hosts all the databases for the microservices. Here we are using schema-per-service model for our architecture. Because different schema for different services make ownership of the databases clear and simple. This means each service has a database schema that's private to that service. But all these schemas are hosted in one database server.

Some of the services such as user-information might cause congestion on the database because of its high-throughput data operations. To solve the problem of congestion a cache server is user for faster retrieval of data when number of users accessing application scales.

For the purpose of caching a Redis server is used. Redis is an open source, in-memory data 18 structure store, used as a database cache. Redis stores the cache in form of key-value pairs. The keys are

user for fast retrieval of the data from cache. Following are the configurations that are made for cache server:

- The maximum memory used for the cache is 1GB.
- Eviction policy when memory is full: LFU (Least Frequently Used)

## 8. Deployment

All the internal microservice APIs are developed using Spring Boot framework. Out of the box, Spring Boot uses a `public static void main` entry-point that launches an embedded web server for you. When the jar is built using `mvn clean install` command, we get the archive packed with embedded web server and all the other dependencies that are required for the application to run. Hence, to run the whole application only this jar file is required. When the application is started, Spring Boot will detect that you have a Spring MVC controller and start up an embedded Apache Tomcat 7 instance.

All the services are individually developed, so each service is built in the way mentioned above. At the end of the build we have all the services' .jar files which can be run individually in order to get the whole application running. Following is the bash script used to package all the services into individual .jar files.

```
#!/bin/bash
if ! [ -x "$(which mvn)" ]
then
    echo "mvn doesn't exist... INSTALLING MAVEN"
    sudo apt-get install maven
else
    echo "mvn exist..."
fi
test -d Services || mkdir Services;
if [ $# -eq 1 ]
then
    mvn install -DskipTests --file $1/pom.xml;
    mkdir -p Services/$1;
```



```

cp $1/target/*.jar Services/$1/;
echo "-----PACKAGED $s-----"
exit 0;
fi

services=( discoveryserver UIService user-authentication-service user-information
VideoProcessing VideoStreamingService Video-Upload-Service )
for s in ${services[@]}
do
    mvn install -DskipTests --file $s/pom.xml;
    mkdir -p Services/$s;
    cp $s/target/*.jar Services/$s/;
    echo "-----PACKAGED $s-----"
done

```

After all the .jar files are generated we containerize the jars for further isolation of services. Also, the containers are easy to orchestrate and scale, the deployment is done by containerizing the services. We use docker and docker-compose for container orchestration. All the containers for individual services are built based on the alpine image with openjdk-11 installed. An example Dockerfile is as follows, used to build the user-information service. All the services are containerized in similar way.

```

FROM alpine:latest
ARG dataIP
ARG redIP
ENV databaseIP $dataIP
ENV redisIP $redIP
ENV discoveryIP "172.17.0.2"
RUN apk --no-cache add openjdk11 --repository=http://dl-
cdn.alpinelinux.org/alpine/edge/communitys
ADD . /usr/src/info/
EXPOSE 80
CMD java -jar /usr/src/info/user-information-0.0.1-SNAPSHOT.jar 80 $discoveryIP
$databaseIP $redisIP

```

## 8.1 Docker Swarm

After all the containers are generated, the deployment of the containers is done by docker-compose. Docker compose is used for orchestration of the services. By deploying all the services using docker-compose the services can be scaled individually. For example, by running `docker-compose scale userinterface=3` command the running instances of userinterface services can be scaled to 3 units. Also, multiple hosts in one network can be connected with each other using docker swarm to run these containers on the cluster of more than one host. Following is the docker-compose.yml file used for the deployment.

```
version: "3"
Services:
  discovery:
    image: discoveryserver:latest
  authentication:
    image: authentication:latest
    depends_on:
      - discovery
    environment:
      - discoveryIP=discovery
  deploy:
    replicas: 1
  userinfo:
    image: userinformation:latest
    depends_on:
      - discovery
    environment:
      - discoveryIP=discovery
    deploy:
      replicas: 1
  userinterface:
```

```
image: userinterface:latest
depends_on:
  - discovery
environment:
  - discoveryIP=discovery
deploy:
  replicas: 1
streaming:
image: streaming:latest
depends_on:
  - discovery
volumes:
  - "/mnt/shared:/mnt/shared"
environment:
  - discoveryIP=discovery
deploy:
  replicas: 1
videoupload:
image: videoupload:latest
depends_on:
  - discovery
volumes:
  - "/mnt/shared:/mnt/shared"
environment:
  - discoveryIP=discovery
deploy:
  replicas: 1
videoprocess:
image: videoprocessing:latest
depends_on:
  - discovery
```

```

volumes:
  - "/mnt/shared:/mnt/shared"
environment:
  - discoveryIP=discovery
deploy:
  replicas: 1
nginx:
  image: nginx:latest
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf:ro
  depends_on:
    - discovery
    - userinterface
  ports:
    - "80:80"

```

Here, only the port for UI service needs to be exposed to access the application. But for scalability purpose of UI service, we need to deploy a gateway which distributes the requests coming for UI Service, between multiple UI service container instances running. Hence, nginx server is used as the gateway which distributed the requests in round robin manner.

As due to the unavailability of the physical resources (due to COVID19 pandemic) to deploy the application in cloud computing lab we are using Amazon Web Services platform to deploy our application in virtual cloud. For deployment of the application three Ubuntu 18.04 Server VM with tire t2.medium are instantiated which would host the containers of all the microservices. Two Ubuntu 18.04 Server VM with tire t2.tiny are instantiated to host the MySQL database server and Redis server for caching. For storage purpose we have created Elastic File System (EFS) that is mounted in all servers. Docker swarm cluster is created of three VMs to run the containers of all services. Same config file as docker-compose.yml is used to deploy the services on docker swarm cluster.

## 8.2 Kubernetes

After all the container are built, those containers are pushed to public repository of docker hub. All the containers will be deployed on a Kubernetes cluster. All of the configurations for the deployment of each of our services is done by creating **YAML config file**, which is then consumed by K8s to deploy them. A sample YAML config file is shown below.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: streaming
spec:
  replicas: 2
  selector:
    matchLabels:
      app: streaming
  template:
    metadata:
      labels:
        app: streaming
    spec:
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
            - weight: 100
              podAffinityTerm:
                labelSelector:
                  matchExpressions:
                    - key: app
                      operator: In
                      values:
                        - streaming
```

```

        topologyKey: kubernetes.io/hostname

containers:
  - env:
      - name: databaseIP
        value: 172.31.35.190
      - name: redisIP
        value: 172.31.89.220
    image: hp2304/vidhyapak:streaming
    imagePullPolicy: Always
    name: streaming
    volumeMounts:
      - mountPath: /mnt
        name: nfs-mount
volumes:
  - name: nfs-mount
    hostPath:
      path: /mnt
      type: Directory
---
apiVersion: v1
kind: Service
metadata:
  name: streaming-service
spec:
  selector:
    app: streaming
  ports:
    - protocol: TCP
      port: 80
      nodePort: 32001
  type: NodePort

```

Kubernetes offers many ways to access a service, both from inside the cluster and to the outside world (internet). Here we have used **NodePort** service type in Kubernetes to expose our user interface, video upload and video streaming services. Since these three services are directly used by our users. Also, authentication and user information services are using **ClusterIP** service, this service type just expose them to inside the cluster, one can't access these services being outside the K8s cluster. In k8s container images are fetched directly from docker hub by providing it's repo path. Also, by mentioning number of replicas and pod affinity rules, service replicas will be scheduled in different nodes to improve service availability.

Our Kubernetes cluster consists of one master and three worker nodes. All the nodes are **AWS EC2 t2.medium** instances. Two Ubuntu 18.04 Server VM with tire **t2.micro** are instantiated to host the MySQL database server and Redis server for caching. For storage purpose we have created Elastic File System (EFS) that is mounted in all worker nodes.

## 9. Future Directions

The basic framework and architecture for the development of video streaming application is setup in this project. In future, this project can be extended to add more features in the application such as live transcripts of video, using machine learning for personalized user suggestions etc. Furthermore, as the application is built with microservices architecture, we can use existing gyapak student database as backend to automatically register students. Also, this whole project can be integrated with the AURIS system to provide video lectures of courses in which students register.

## 10. References

- [1] S. Newman, Building microservices: designing fine-grained systems, First Edition. Beijing Sebastopol, CA: O'Reilly Media, 2015
- [2] V. Singh and S. K. Peddoju, "Container-based microservice architecture for cloud applications," in 2017 International Conference on Computing, Communication and Automation (ICCCA), 2017, pp. 847–852.
- [3] M. Villamizar et al., "Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures," *Serv. Oriented Comput. Appl.*, vol. 11, no. 2, pp. 233–247, Jun. 2017.
- [4] A. M. Joy, "Performance comparison between linux containers and virtual machines," in *Computer Engineering and Applications (ICACEA)*, 2015 International Conference on Advances in. IEEE, 2015, pp. 342–346.
- [5] A. M. Joy, "Performance comparison between Linux containers and Virtual machines," *Computer Engineering and Applications (ICACEA)*, 2015 International Conference on Advances in, Ghaziabad, 2015, pp. 342- 346.
- [6] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "Performance comparison between container-based and VM-based services," *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, 2017.