PLATFORM FOR IMPROVING SEARCHABILITY AND INTERACTIVITY OF RECORDED LECTURES

Project ID: 19-087

Software Requirement Specification

Karunaratne D. C.
IT16037434

Bachelor of Science (Honors) in Information Technology
Specializing in Software Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

May 2019

PLATFORM FOR IMPROVING SEARCHABILITY AND INTERACTIVITY OF RECORDED LECTURES

Project ID: 19-087

Software Requirement Specification

Bachelor of Science (Honors) in Information Technology
Specializing in Software Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology
Sri Lanka

May 2019

DECLARATION

I declare that this is my own work and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Name	Student ID	Signature
Karunaratne D. C.	IT 16037434	

The above candidate is carrying out research for the undergraduate Dissertation under my supervision.

Signature of the supervisor: Date:

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Purpose

This document is a Software Requirement Specification (SRS) which aims to provide a comprehensive description of the requirements of the software product which will be the main deliverable of the research project, *A Platform for Improving Searchability and Interactivity of Recorded Lectures (19-087).* The focus of this SRS will specifically be on the module of the team member D. C. Karunaratna (IT16037434), which will handle the matching of each line of code in a sample code file to its occurrence in a live-coding video. The document will address details such as the project scope, target audience, functional requirements and non-functional requirements of the project.

This document will act as a reference document throughout the software development lifecycle and will benefit the project supervisor, co-supervisor and members of the research team who will be carrying out development.

1.2 Scope

This document covers the requirements for one of the components of the proposed Platform for Improving Searchability and Interactivity of Recorded Lectures. The software product will follow a Software as a Service (SaaS) model. Current architectural styles and best practices in the field of cloud-computing will be considered when designing the application. The application will be hosted by an appropriate third party and delivered to the end users over the internet as a web application. There are four main components of the system,

- Code extraction and source code matching component.
- Topic Segmentation component
- Question Generation component
- Audio enhancement and lecture slide matching component

This SRS will focus mainly on the Code Matching component. This component will contain a frame extractor, an audio extractor, a trained neural network to identify frames containing code and a line matching algorithm to match the frames to the occurrences in the sample code.

1.3 Definitions, Acronyms and Abbreviations

ADSL	Asymmetric Digital Subscriber Line
FTTH	Fiber to The Home
HSDPA	High-Speed Downlink Packet Access
SaaS	Software as a Service
SAST	Static Application Security Testing
SRS	Software Requirements Specification

Table 1.1 – Definitions, Acronyms and Abbreviations

1.4 Overview

This SRS document intends to cover the functional and non-functional requirements of the proposed *Platform for Improving Searchability and Interactivity of Recorded Lectures* with specific focus on the code matching component of the product. Each of the requirements have been discussed clearly in detail under three main chapters. The level of technical detail increases as the chapters progress. The first chapter gives an overview of the whole system, along with a brief description about the purpose and the scope of this document.

The second chapter starts by comparing the software application to the existing systems in the market. It then moves on to explain several interfaces, constraints and operation of users to provide the reader with a better perspective of understanding the product, finally giving a summary of major functions, characteristics of users, constraints of system, assumptions and dependencies.

The third chapter of this SRS will focus on specific requirements of the product and is primarily written for individuals who require the technical details of the system. This

chapter will describe in depth the functionalities mentioned in chapter two. The latter part of this chapter provides details about the system attributes such as reliability, maintainability, availability and security.

The final additional chapter provides supporting information regarding the contents of the document.

2 OVERALL DESCRIPTION

The recorded lecture is a powerful tool for conveying lecture content. It has several advantages over conventional lectures. For example, recorded lectures do not require time or location constraints [1]. Recorded lectures can be further improved upon by corelating them with supporting lecture material such as lecture slides, questionnaires and source code samples. The main aim of this research is to create a product that will enable lecturers to create such content with minimum human intervention and enable students to gain a more interactive learning experience. Platforms such as LearnWorlds [1], Echo360 [2] and Techsmith Relay [3] allow videos to be edited to make them more interactive however the methods employed by these platforms are manual and therefore tedious and time consuming. Furthermore, at the time of writing this document there is no platform which automatically identifies relationships between source code files and software engineering lecture videos.

The Platform for Improving Searchability and Interactivity of Recorded Lectures aims to address these drawbacks by introducing an automated process of identifying and combining the lecture material to create an enhanced user experience. The platform will be developed using a cloud microservices architecture which holds many benefits. Building the product as a collection of independent microservices not only facilitates an easier development process but also allows for a more robust system which has minimum downtime [2]. By utilizing the power of cloud computing, the application can be up-scaled and down-scaled with ease depending on the load.

2.1 Product Perspective

The background study and literature review carried out for the project proposal revealed several notable solutions which have a similar feature set and similar goals and objectives. Table 2.1 shows a comparison of features which are common across the platforms along with those that are improved in the proposed platform.

[1] https://www.learnworlds.com/
[2] https://www.echo360.com/about/
[3] https://www.techsmith.com

Features	LearnWorlds	Echo360	TechSmith Relay	Our Solution
Matching lines in code samples to occurrence in recorded lectures	✗	✗	✗	✓
Automated segmentation of lecture video into topic units	✗	✗	✗	✓
Matching slides with the lecture video	✗	✓	✗	✓✓
Automated noise removal from the video	✗	✗	✗	✓
Automatic question generation	✗	✗	✗	✓
Embedding questions in video playback	✓	✓	✓	✓✓
Automated question embedding in video playback	✗	✗	✗	✓

Table 2.1 – Comparison with existing products

✗: Not available

✓: Available

✓✓: Improved

From the literature survey and research activities carried out, we discovered that there is quite a research gap that needs to be addressed. Although similar existing products are available, the proposed system will improve on the features of these systems as well as introduce new features and functionality.

2.1.1 System Interfaces

- Python 3 sdk
- PyMongo Interface to connect MongoDB database and Python backend

2.1.2 User Interfaces

The user interfaces for the code matching component will consist of the following:

- Interface for uploading material: *this is the page seen by the lecturer or content creator.*
- Code finder interface: *source code is shown alongside the lecture video and clicking a line of code will highlight the timestamps where the code occurs in the video.*

2.1.3 Hardware Interfaces

No specialized hardware is required. All processing will be carried out in the cloud. A regular computer with an internet connection will be required to access the web application.

2.1.4 Software Interfaces

- Jupyter Notebook
- Google Tensorflow
- Ffmpeg – video and udio manipulation
- Amazon Web Services
- Keras – Deep Learning Library
- MongoDB
- Node.js
- Anglar.js
- Express.js

2.1.5 Communication Interfaces

Internet connection with a minimum data rate of 7Mbit/s

2.1.6 Memory Constraints

Minimum RAM of capacity 4GB

2.1.7 Operations

The following user operations can be seen in the code matching component

- Login to the site
- Upload lecture videos and source code file
- Search for a specific lecture.
- Use source code as an index to skip to the relevant frame in the lecture video

2.1.8 Site Adaptation Requirements

The product will be delivered across the web as a SaaS product. Since the end user interacts with a web-interface and all processing is carried out in the cloud, a JavaScript enabled web browser is all that is required

2.2 Product Functions

The product functions will be explained with the help of a use case diagram (Figure 2.1) and the relevant Use case scenarios (Table 2.2 and Table 2.3)

2.2.1 Use Case Diagram

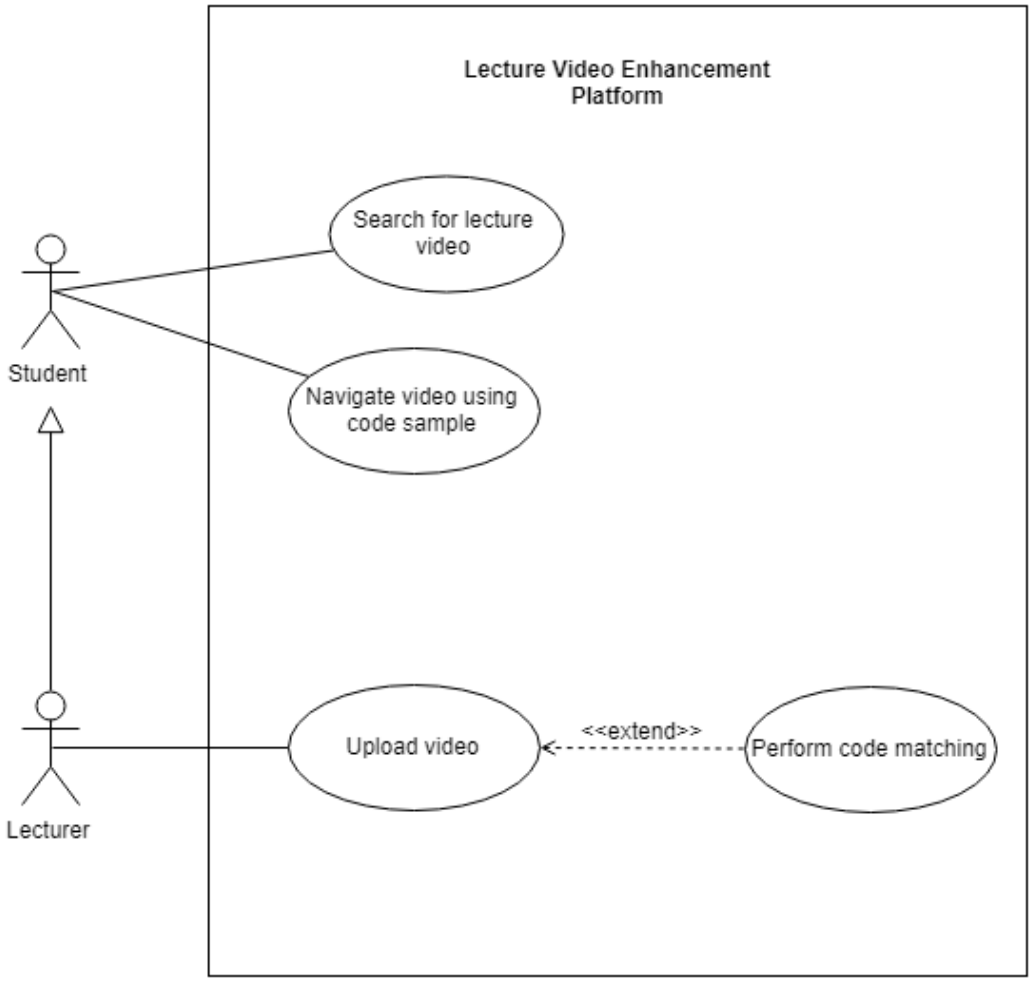

Figure 2.1 – Use case diagram for code matching component

2.2.2 Use Case Scenarios

Use case name	Upload video
Pre-Conditions	User must be logged into the system using valid credentials.
Post-Conditions	Video and lecture materials should be uploaded to the platform
Actor	Lecturer/ course creators
Main success scenarios	1. Use case starts with user logging into the system. 2. User navigates to the upload content page. 3. User clicks ‘upload video’ button 4. User selects video files from the computer 5. System displays the uploading progress. 6. User clicks the ‘upload content’ button. 7. User selects lecture slides and/or code file from the computer. 8. System displays the uploading progress. 9. System displays uploaded video and lecture slides.
Extensions	5a. Upload fails and system displays error message. 8a. Upload fails and system displays error message.

Table 2.2 – Upload video use case scenario

Use case name	Navigate video using code sample
Pre-Conditions	User must be logged into the system using valid credentials.
Post-Conditions	-
Actor	Lecturer/ course creators
Main success scenarios	1. Use case starts with user logging into the system.

	2. User navigates to the search lecture videos page. 3. User selects the desired lecture video to watch 4. User goes to the ‘code finder’ tab on the video details page. 5. System displays the source code alongside the lecture video. 6. User clicks on a line in the source code which is displayed. 7. System displays the list of timestamps where the line of code is found in the video. 8. User clicks on a timestamp from the list. 9. System goes to the specific timestamp in the lecture video.
Extensions	5a. System displays “No source code file uploaded” in the source code section. 6a. System displays “Line not found in video” message

Table 2.3 - Navigate video using code sample Use Case Scenario

2.2.3 Activity Diagrams

The diagrams in this section are limited to those required by the code matching module. Figure 2.2 shows the process of uploading the lecture video and code file.

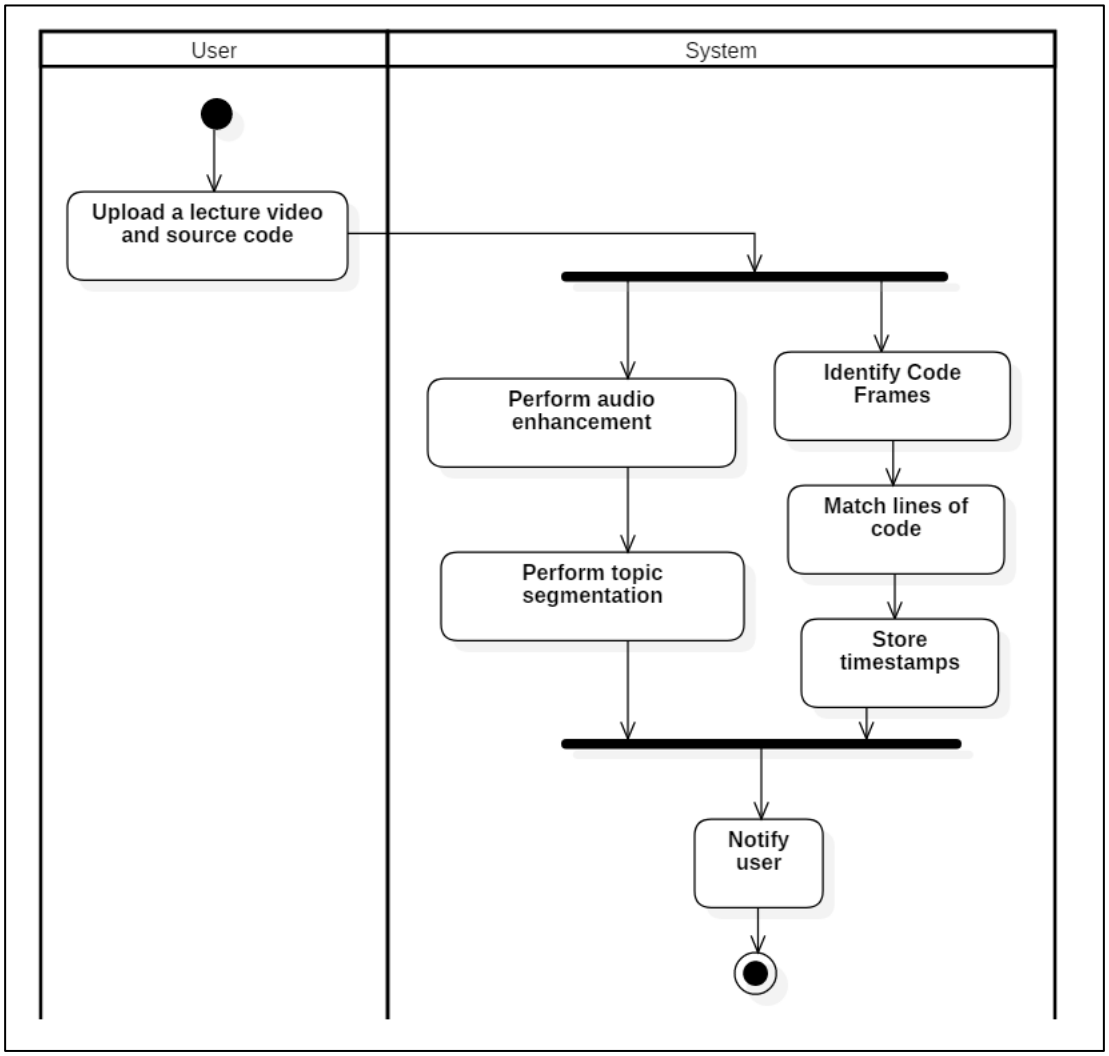

Figure 2.2 – Upload lecture video activity diagram

Figure 2.3 shows the process of navigating a lecture video using the code file as an index

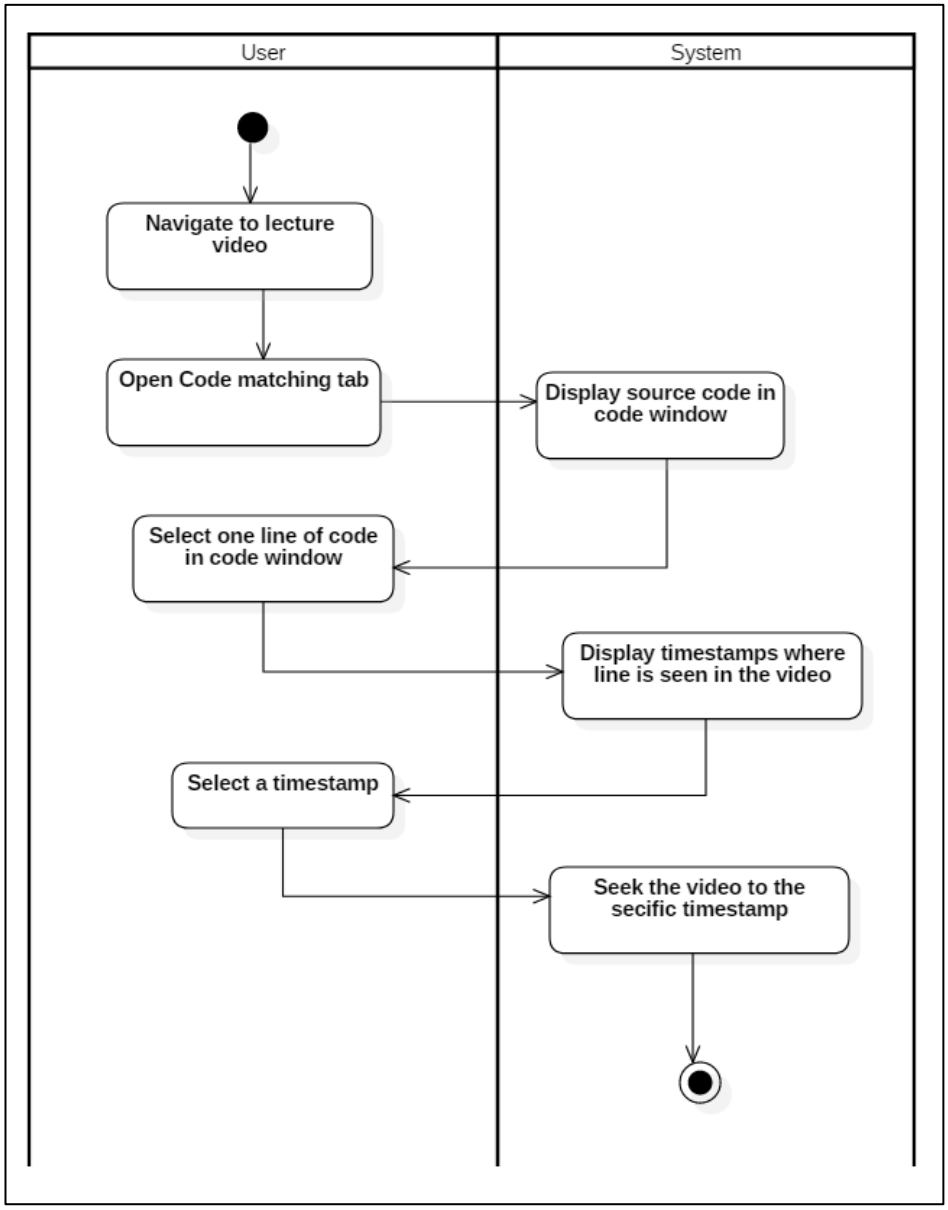

Figure 2.3 – Navigate video using code activity diagram

2.3 User Characteristics

The target audience of our system is universities and higher educational institutes who wish to provide course materials for their students online. Therefore, our users fall into 2 main categories:

- Lecturers
- Students

Lecturers will use this system to upload course materials, and make it more interactive for the students, whereas students will use the system for learning and studying purposes. Both the categories of users do not require any prior technical knowledge besides basic interactions with a web application.

2.4 Constraints

- The frontend client application which will be developed using JavaScript.
- The backend of the system will follow a microservices architecture where several components of the backend will be developed independent of each other.
- The developers of each microservice will be free to choose whatever technology stack suits their purpose.
- Internet connectivity is a must to interact between the frontend and backend processes.
- Backend system must have minimum of 2GB RAM.
- Angular.js will be used to develop frontend client application while the backend processes are developed using Python and machine learning frameworks.
- After performing the necessary analysis on the uploaded video, the final optimized video shall not be stored on the system internally. All temporary files will be removed after the final video is hosted on a suitable video hosting platform.

2.5 Assumptions and Dependencies

- All browsers are assumed to be capable of running the frontend application.
- It is assumed that internet connectivity can be obtained and maintained when required.

2.6 Apportioning Requirements

The following requirements are mainly focused on the code matching component and are categorized as follows,

- Essential Requirements:
    1. Allow learner to navigate the lecture video using the lines of code in a source code file as the reference.
    2. Allow the content creator to upload a lecture video and source code file so that it can be processed by the system.
- Desirable Requirements:
    1. Generate data on popularity of lecture videos.
    2. Have a feedback system where students can reach the lecturer who created the videos.
- Optional Requirements:
    1. Create a search function to search through the indexed video file for code occurrences.

3 SPECIFIC REQUIREMENTS

3.1 External Interface Requirements

3.1.1 User Interfaces

The interfaces mentioned under subheading 3.1.1 are those required by the code matching module only. This section provides a more detailed representation of the required interfaces and are intended for the developers of the system. Figure 3.1 shows the code finder view which displays the lecture video alongside the source code.

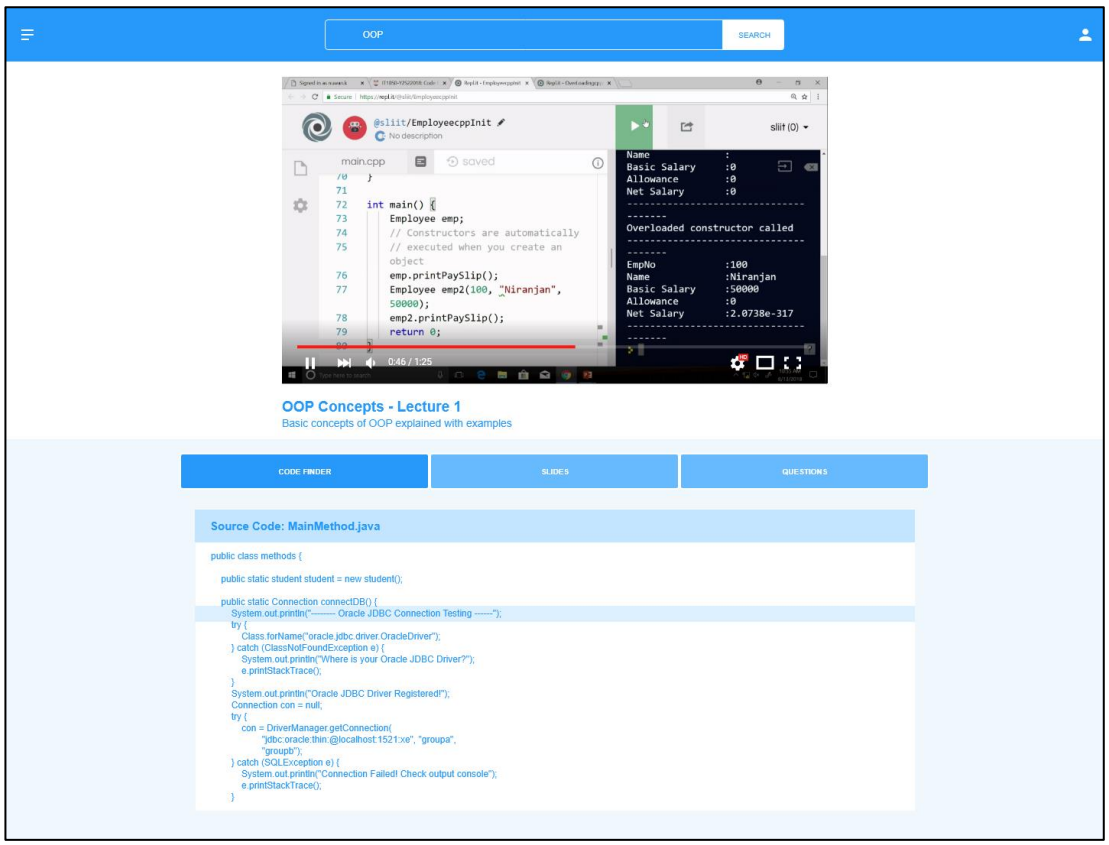

Figure 3.1 – Code Finder interface

3.1.2 Hardware Interfaces

No specialized hardware is required for the backend. A regular computer with an internet connection and web browser capable of running JavaScript will be required to access the web application.

3.1.3 Software Interfaces

- Jupyter Notebook
- Google Tensorflow
- Ffmpeg – video and udio manipulation
- Amazon Web Services
- Keras – Deep Learning Library
- MongoDB
- Node.js
- Anglar.js
- Express.js

3.1.4 Communication Interfaces

A stable high-speed internet connection is required.

3.2 Classes

- Essential
    - MediaController – To handle upload and streaming of video and other media content.
    - Authenticator – To generate authentication tokens for valid user credentials.
    - VideoPreProcessor – Handle initial processing of uploaded videos
    - ServiceController – Apply the required algorithm or service when called.
- Optional
    - DataAnalyzer – To Generate analysis reports

3.3 Performance Requirements

Considering the requirements of the code matching component in particular, the following performance requirements must be met,

- Loading a webpage should not take more than 4000 milliseconds.
- Once a page is loaded, each user interaction should not have a latency of more than 500 milliseconds.
- Video quality should be at least 480p.

3.4 Design Constraints

- The proposed system is mainly focused on playing videos to the users. User interfaces of the platform should be focused on enabling the interactivity and searchability provided by backend processes.
- Architectural design must emphasis on parallel processing since computational heavy tasks of the platform will take significant amount of time to complete.

.

3.5 Software System Attributes

3.5.1 Reliability

The system should be implemented using tried and tested libraries wherever possible to reduce the risk of failure. A failure will be defined as a software defect which will cause the entire system to be unavailable to end users. Each microservice should have at least 200 hours between each failure.

3.5.2 Availability

Availability is one of the major system attributes that is considered in the modern world. It can be simply defined as the probability that the system is functioning properly when requested for use. Availability is also a key factor in the proposed system because the system needs to be available for use whenever requested by the lecturers and students. As the backend will be deployed in a cloud environment, the availability of the system will depend

on the cloud provider's services and performance. However, we expect the system to be available about 90% of the time.

3.5.3 Security

Suitable encryption methods should be used to encrypt all data generated by the system. Special attention should be given to to maintain the confidentiality of the Users' personal information if it is stored. The user should be authenticated using a suitable framework before utilizing the system. All passwords must be stored as hashed values. For internal system communication each call which is transmitted to a service over the internet must use token-based authentication based on the OAuth or OAuth2 standard [3]. Furthermore, Static Application Security Testing (SAST) methods [4] should be employed in the continuous integration pipeline to identify security vulnerabilities in the source code.

3.5.4 Maintainability

Maintainability can be defined as the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. The system is divided into functionally cohesive modules to improve maintainability. Also, we practice software engineering best practices throughout this research, which will improve the overall maintainability of the system. The architecture of the system will be deployed in a manner that will support high maintainability in the cloud environment in which it is deployed.

4 Supporting information

4.1 References

[1] "Importance and Effectiveness of E-learning," 1 December 2015. [Online]. Available: https://higheredrevolution.com/importance-and-effectiveness-of-e-learning-9513046ed46c. [Accessed 6 March 2019].

[2] "Benefits of Microservices - Microservices on AWS," Docs.aws.amazon.com, 2019. [Online]. Available: https://docs.aws.amazon.com/aws-technical-content/latest/microservices-on-aws/benefits-of-microservices.html. [Accessed 20 April 2019].

[3] "OAuth 2.0 — OAuth," Oauth.net, 2019. [Online]. Available: https://oauth.net/2/. [Accessed 13 May 2019].

[4] "Static Application Security Testing (SAST) - Gartner IT Glossary," Gartner IT Glossary, 2019. [Online]. Available: https://www.gartner.com/it-glossary/static-application-security-testing-sast. [Accessed 13 May 2019].

4.2 Appendix A: High level architecture diagrams

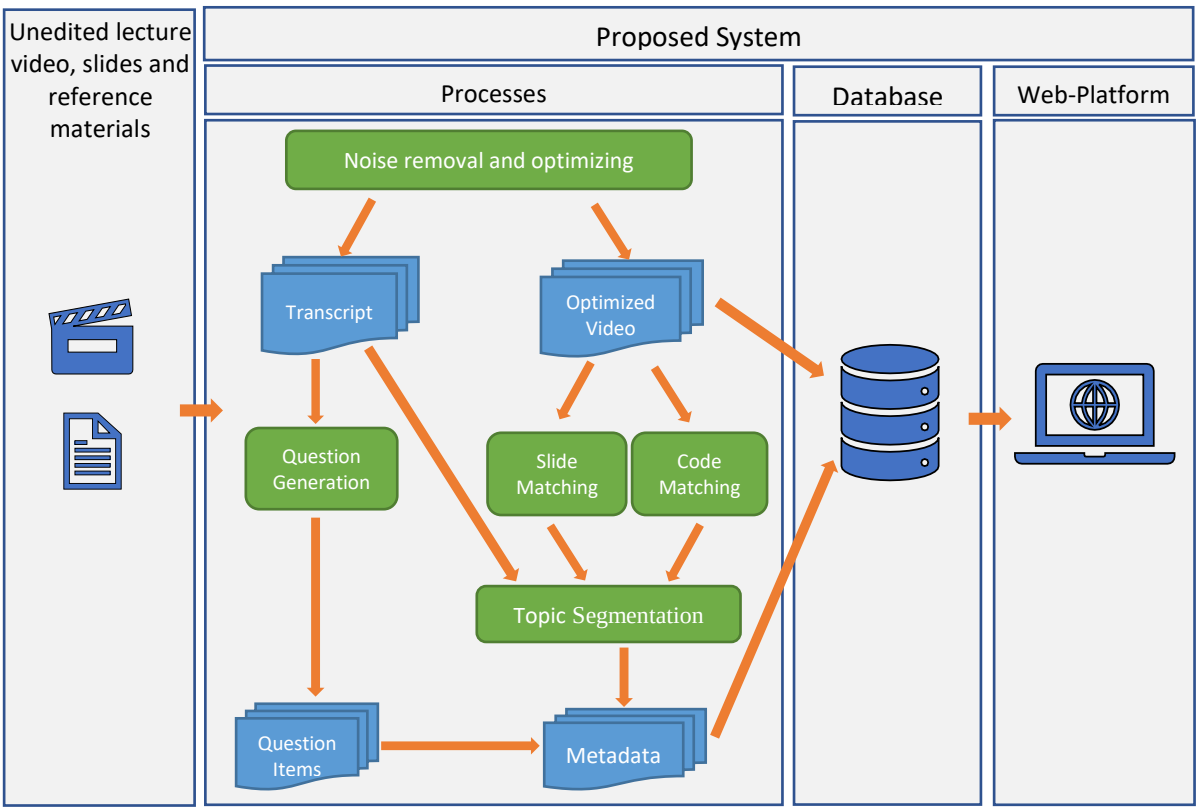

Figure 4.1 – High level system architecture

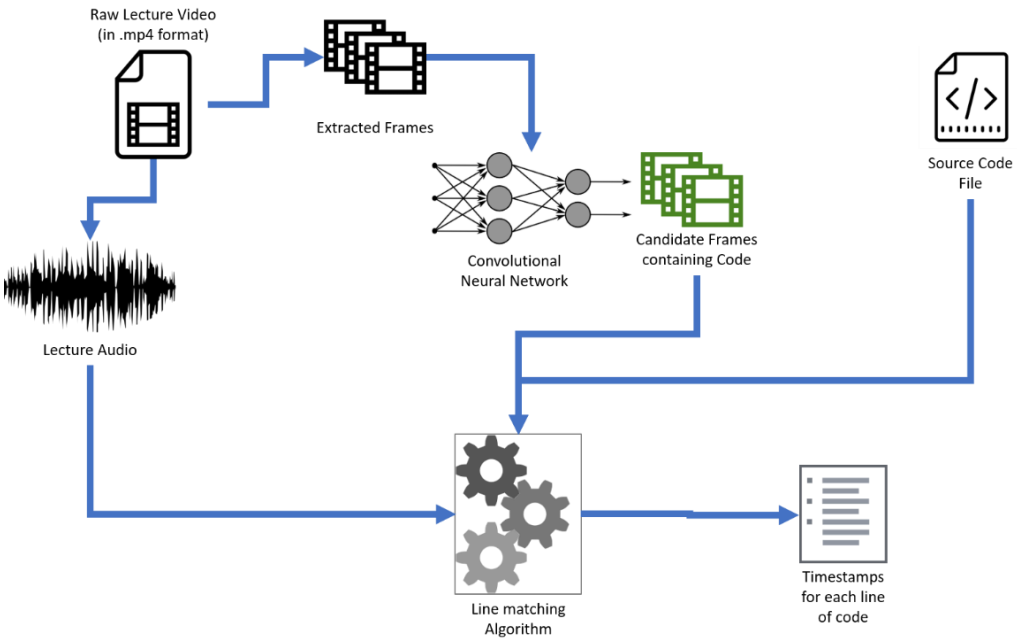

Figure 4.2 – High level architecture of code matching module