DIU_COURSE_TUBE

Submitted By

Shiddhartha Pal

Student Id

221-35-1069

Supervised By

Dr. Md. Fazla Elahe

Assistant Professor & Associate Head

This project report has been submitted in fulfilment of the requirements for the degree of Bachelor of Science in Software Engineering

APPROVAL

This project titled on "**DIU_COURSE_TUBE**", submitted by **Shiddhartha Pal (ID: 221-35-1069)** to the Department of Software Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Software Engineering and approval as to its style and contents.

BOARD OF EXAMINERS

Fazla Elahe

Dr. Fazla Ealhe — **Chairman**
Assistant Professor & Associate Head
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Marzia

Dr. Marzia Ahmed — **Internal Examiner 1**
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Dr. Shabnom Mustary — **Internal Examiner 2**
Assistant Professor
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Md. Rajib Mia — **Internal Examiner 3**
Lecturer (Senior Scale)
Department of Software Engineering
Faculty of Science and Information Technology
Daffodil International University

Mohammod Abul Kashem, PhD — **External Examiner**
Professor
Department of Computer Science and Engineering
DUET, Bangladesh

DAFFODIL INTERNATIONAL UNIVERSITY

DECLARATION OF THESIS AND COPYRIGHT

I declare that this thesis is classified as:

I acknowledge that Daffodil International University reserves the following rights:

1. The Project is the Property of Daffodil International University.
2. The Library of Daffodil International University has the right to make copies of the Project for the purpose of research only.
3. The Library of Daffodil International University has the right to make copies of the Project for academic exchange.

Certified by:

NOTE: * If the Project is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

PROJECT DECLARATION LETTER (OPTIONAL)

Librarian,
Daffodil International University,
Daffodil Smart City,
Ashulia.Dhaka,Bangladesh

Dear Sir,

CLASSIFICATION OF Project AS RESTRICTED

Please be informed that the following project is classified as RESTRICTED for a period of three (3) years from the date of this letter. The reasons for this classification are as listed below.
Author's Name
Project Title

Reasons (i)

(ii)

(iii)

Thank you.

Yours faithfully,

_____
(Supervisor's Signature)

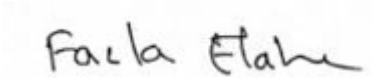
Date:

Stamp:

Note: This letter should be written by the supervisor and addressed to the Librarian, *Daffodil International University* with its copy attached to the thesis.

SUPERVISOR'S DECLARATION

hereby declare that I have checked this project and in my opinion, this project is adequate in terms of scope and quality for the award of the degree of Bachelor of Science.



_____

(Supervisor's Signature)

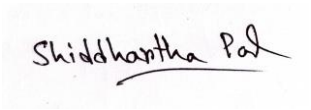
Full Name : Dr. Md. Fazla Elahe

Position : Assistant Professor & Associate Head

Date : 22 December 2025

STUDENT'S DECLARATION

I hereby declare that the work in this project is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Daffodil International University or any other institution.



_____

(Student's Signature)

Full Name : **SHIDDHARTHA PAL**

ID Number : 221-35-1069

Date : 22 December 2025

DIU_COURSE_TUBE

SHIDDHARTHA PAL

Project submitted in fulfillment of the requirements
for the award of the degree of
Bachelor of Science of Science

Department of Software Engineering
DAFFODIL INTERNATIONAL UNIVERSITY

DECEMBER 2025

ACKNOWLEDGEMENTS

Above all else, I offer my deepest gratitude to the Almighty Allah for bestowing upon me the strength and opportunity to successfully conclude this thesis endeavor.

My sincere and heartfelt appreciation goes directly to my supervisor, **Dr. Md Fazla Elahe**, Assistant Professor & Associate Head of the Department of Software Engineering. His expertise was a source of strength for me as I utilized it constantly, and I believe it is important to note that both the trust she had in providing me with his sincere and valuable advice and also his motivation by encouraging me, helped propel this research further than I could have ever gone alone.

In addition, I would like to express my gratitude to each and every person who participated in the survey (which was an integral part of this thesis) and gave their time to do so. Their energy and thoughtfulness of each response to the survey was instrumental to me completing the validation survey on behalf of myself.

Additionally, I want to thank the entire faculty of the Department of Software Engineering for the assistance and encouragement they provided to me during my studies at all times.

Lastly, I am grateful for my parents, whose love and support has been there for me unconditionally, and who has supported me completely as I moved through this journey toward achieving this goal and milestone.

DEDICATION

Henceforth, I am confirming that I did this project under the supervision of Dr. Md Fazla Elahe, Assistant Professor & Associate Head, Department of Software Engineering, Daffodil International University. Likewise, I am stating that the entire or any part of this record has not been submitted in the form of a record anywhere for my degree.

ABSTRACT

DIU_COURSE_TUBE represents an innovative, modernized, and interactive video sharing platform that will serve as a model for a contemporary video sharing service such as YouTube. Built upon a state-of-the-art technology stack, DIU_COURSE_TUBE uses a modern front-end, based upon React, along with TypeScript for the user interface (UI), and a back-end API to manage all business logic, data processing and user administration. DIU_COURSE_TUBE's focus is centered around the video viewing experience. Upon selecting a video from the available selection, the application retrieves all required information (i.e., the video itself, its title, its description, etc.) and, importantly, the creator's channel details. As DIU_COURSE_TUBE utilizes a CDN for streaming video content, it is able to deliver this content efficiently and reliably to any location globally where internet access exists.

As user engagement is a major area of emphasis with DIU_COURSE_TUBE, viewers are provided with a number of methods by which they may engage with DIU_COURSE_TUBE's content:

Reactions: Users may utilize a traditional Like/Dislike system to react to videos viewed at DIU_COURSE_TUBE.

Subscribing: Users may also elect to subscribe to their preferred channels so that they may be notified when new content is released.

Comments: Each video viewed at DIU_COURSE_TUBE has a comments section allowing for a form of community interaction and dialogue between the creators and their audiences.

Sharing/Saving: Users are also able to easily share each video across a variety of social media outlets or simply copy the direct link to the video. Additionally, users are able to save videos to a personally created list for easy future reference.

In addition to this, it has standard functionalities such as User Authentication (Login/Register), Viewing History (for tracking which videos were viewed) and a function for fetching recommended videos to continue engaging users. It is created to have an intuitive User Interface with modern aesthetic features to improve the overall viewing experience.

From a technical perspective, the Project has a strong Architecture and the Frontend and Backend are communicating with each other via a defined API Endpoint, allowing for actions such as retrieving video data, commenting on videos, and managing video subscriptions. The state is being managed in a way that ensures the UI will always be in sync with the Back-end Data, providing an efficient and seamless experience for the End-User.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CDN	Content Delivery Network
DIU	Daffodil International University
ER	Entity-Relationship
FR	Functional Requirement
LTS	Long-Term Support
SSD	Solid State Drive
UI	User Interface
UML	Unified Modeling Language

LIST OF APPENDICES

CHAPTER:1 INTRODUCTION

1.1 Background

This section gives an overview of DIU_COURSE_TUBE, which defines why the project exists (the context), explains the problem it was designed to solve, identifies the project's goals, and describes the scope of the project.

1.1.1 Context and Relevance

With the explosion of the Internet as a viable means of communication, online video has evolved into the most widely consumed type of media on the web. With such services as YouTube, Vimeo, and TikTok, the way we obtain information, learn new skills, and enjoy entertainment have been dramatically changed. Therefore, there is now a very high expectation from end-users about the video sharing service they use. Users do not simply desire to watch videos, they want high quality video streaming and the ability to quickly load these streams; they desire interaction with both the creator and the community of viewers; and, they desire a video sharing experience that is tailored to their interests. As a result, the most recent trend in the design of video sharing services is to create "all in one" video sharing services that are not only video players, but also social communities. In the development of all-in-one video sharing services, the greatest challenge is to provide a seamless, buffering free video viewing experience to a global audience while simultaneously providing real time processing of a large number of user interactions, including comments, likes, and subscriptions.

1.1.2 Problem Identification

Although there are several video-based applications available today, developing an application from the ground-up has its own unique challenges. The specific problem this project aims to address is the difficulty in bringing together the entirety of the capabilities required of a modern video platform into a singular, unified and easy-to-use interface. Most existing video platforms, or smaller projects that attempt to provide some subset of functionality (e.g., video playback) do not have a complete suite of engagement options. For example, most video platforms provide comments but do not include like/dislike buttons; most video platforms provide a way for users to subscribe to videos, but rarely does it have a well-implemented method for users to share those videos with others. DIU_COURSE_TUBE was developed to fill this gap by providing a platform that integrates high-performance video streaming with a comprehensive suite of modern ways for users to interact with video and each other, creating a complete and engaging environment for both video consumers and video producers.

1.1.3 Purpose and Justification

The primary goal of the DIU_COURSE_TUBE project is to design and deliver a working model of a current video-sharing/streaming platform. A project such as this has importance in that it will demonstrate the process involved in building an actual, real-

world, multi-faceted Web Application by utilizing some of today's most popular technologies, including; React, TypeScript, and a Backend API.
A good reason for building a project such as the DIU_COURSE_TUBE project is to be able to show one's capability to resolve complex engineering issues. This demonstrates an individual's knowledge and/or skill level in all aspects of the web development process – i.e., from designing the User Interface (UI) on the Frontend to managing Data on the Backend as well as integrating with third party services – e.g., a CDN for quick video playback. Ultimately, this type of project provides the tangible, working application/prototype necessary to create a functional platform that emulates the core functionality of industry leaders while providing an opportunity to showcase one's expertise in advanced web development skills through an attractive and useful Portfolio piece.

1.1.4 Scope

The Scope of the DIU_COURSE_TUBE project/application defines all of the Key Features that are needed for a user to have a modern video watching experience. This includes:
Video Playback: Secure and High Performance Video Streaming via an External Media Delivery Service (i.e., CDN).

The key element of the user authentication is the registration and login processes of users' accounts through user-specific data.
Other key elements include:
Channel and video content: providing users with the details about each video including title and description; also, providing users with the information related to the channel, i.e., creator name, creator profile picture, number of subscribers.

User Interactivity Elements:
- **Subscription**: Users may add or remove themselves from a subscription to a channel.
- **Reactions**: Users may provide reactions to videos using a like and dislike system.
- **Comment Section:** Providing users with an opportunity to interact with one another via a full featured comment section.
- **Save Videos**: Allowing users to save videos they wish to view at a future date to a personalized video list.
- **Share Modal**: Allowing users to easily share videos across all major social media platforms either by copying the URL of the video or by using a direct share button.
- **User Watch History**: Tracking which videos are viewed by a user.
- **Related Videos**: Providing users with recommendations of videos that are likely to be of interest to them, thereby keeping them engaged within the platform.

1.2 Project Planning and Initiation

Feasibility Study (Step-by-Step)

A preliminary feasibility study was conducted prior to commencing development to assess whether the DIU_COURSE_TUBE project was possible and reasonable to accomplish. This feasibility study was accomplished in stages to provide assurance regarding the project's viability from various perspectives. It was concluded that developing a working prototype of a video sharing platform is very feasible, due to the availability of modern technologies, and due to clearly defining the project's scope.

Phase 1 Preliminary Analysis & Project Scope Definition:

Within the first phase of the study, I assessed the underlying concept and established the project's parameters.

Analysis: Initially, the objective of creating a web-based application that replicates the fundamental experience of popular video sharing platforms, e.g. YouTube. The most important features that contribute to the success of such platforms were identified as follows: quality video playback, user accounts and a rich set of user interaction tools (e.g. likes, comments, subscriptions). The results of the preliminary analysis suggested that developing the features mentioned above will establish a solid base upon which a complete and engaging user experience could be developed.

The Project Scope: So as to limit the project to something I could accomplish, I established an exact scope. I focused this project on the use of video and the user interaction experience. I included all of the following in my focus:

- A single, video-watching webpage that users will be able to access and watch video on.
- Information about each video and channel being viewed on the webpage.
- All possible engagement features: Subscribe to Channels; Like or Dislike Videos; Full Comment Section; Share Video; Save Video(s).
- Video History: Track how much time a user has spent watching videos on the website.
- I also built the UI using modern web technologies (React, Redux and TypeScript).

Phase 2 Market Feasibility Analysis (or Market Research):

In this phase I examined the current marketplace to find out if the project has value. Although there are large players in the online video space such as YouTube, the goal of DIU_COURSE_TUBE is not to compete with these large players. Rather, the study of the marketplace provided me with the opportunity to define which "must have" features that today's users require. Through the analysis of other successful platforms, I was able

to confirm that features such as simple user registration processes, real-time reaction mechanisms (i.e., likes), and comprehensive sharing options are no longer discretionary -- they are now required for users to interact with your platform. From a "market features" standpoint this project was found to be feasible since it is designed to incorporate the well-documented, highly sought-after functionalities, and therefore provides evidence of a solid understanding of how to create a modern, user-friendly web application.

Phase 3 Technical Feasibility Analysis:

This phase evaluated whether I had the necessary technical tools and skills to build the project.

The project was found to be technically feasible. I chose a very modern, robust, and well-supported technology stack:

Front-end: I used React with TypeScript to rapidly create a dynamic, and type-safe User Interface.

Back-end: A Back-end API (which is implied by the fetch calls in the code) manages the Business Logic, Database Interactions, and User Authentication.

Video Streaming: To ensure fast and buffer-free video streaming capabilities for this project, the project utilizes a professional Content Delivery Network (CDN), Bunny.net. Bunny.net will handle all the heavy-lifting tasks associated with delivering video streams, which is a common and extremely successful approach in the video streaming industry.
The development team has developed the necessary knowledge in order to accomplish the requirements of this project. The design pattern, which creates a separation between the front-end and back-end and utilizes a CDN for media, provides scalability and stability as well as reliability and makes the project have solid technical merit.

Phase 4 Financial Feasibility Analysis:

During this phase, the financial implications for developing and operating the project were taken into account.

Considering the project is a student or prototype, the budget for the project was minimized and feasible. The primary costs of the project are for the third party services:
Video Hosting & Streaming (CDN): Many CDN providers, such as Bunny.net, provide free trial accounts or charge per usage for their service, which make the costs for a small-scale prototype almost zero or free.

Hosting: Both front-end and back-end components of the application can be hosted on platforms that provide free accounts for small projects (such as Vercel, Netlify, or Heroku).

Development Time/Development Costs: The most significant expense incurred by the developer was time spent on development rather than money.

1.3 Target User Profile and Tentative Elicitation Process

1.3.1 Target Users

The DIU_COURSE_TUBE platform is intended for a wide range of audiences with varying purposes and roles. The three main types of users and stakeholders can be characterized as follows:

Viewer (User): The viewer is the largest category of users. Viewers use the platform primarily for viewing and discovering video content, interacting with video content and other users.

Content Creator (Owner): The content creator are the users who create and (in a complete release) publish video content to the platform. Content creators wish to develop an audience, distribute their content and monitor how well their content is being engaged with.

Administrator (Moderator): The administrator is a technical user responsible for ensuring the continued operation of the platform (health of the platform), administering content provided by users, responding to reports to maintain a safe community.

1.3.2 User profile

Table 1.3.2.1: **User Profile for The Viewer**

User Class	**Note on Characteristics**
Types of user	Casual and frequent users seeking entertainment or information.
Age range	Broad, typically 19-65 years old.
Frequency of use	Varies from daily to a few times a week
Mandatory	No, use is voluntary and based on personal interest
Computer experience	Ranges from Novice to Expert. The interface must be intuitive.
Education	All levels. Content appeals to a wide educational background.
goal	To be entertained, learn new things, follow favorite creators, and interact with a

	community through comments and reactions.
Language skills	Primarily English, but a real-world version would need to be multilingual.
Number of users	The largest group, potentially scaling to thousands or more.
Training	None. The platform must be self-explanatory and easy to navigate.
Others system use	Familiar with YouTube, TikTok, and other social media platforms. Expects similar features and usability.
Way of working	Accesses the platform on various devices, primarily desktops and laptops for a rich viewing experience.

Table 1.3.2.2: **User Profile for The Content Creator**

User Class	**Note on Characteristics**
Type of user	A dedicated user who contributes content to the platform.
Age range	Typically 24-50 years old.
Frequency of use	Frequent; daily or weekly to check on video performance and engage with their audience.
Mandatory	No, but their use is essential for the platform's content growth.
Computer experience	Intermediate to Expert. Comfortable with web applications.
Education	Varies, but often has expertise in the topic of their content.
goal	To share their knowledge or creativity, build a community of subscribers, and track the engagement on their videos (likes, comments)
Language skills	Proficient in the language of their target audience, likely English.
Number of users	A smaller, but vital, group compared to viewers.

Training	Minimal. The process for managing their channel and viewing stats should be straightforward.
Others system use	YouTube Studio, Patreon, and other creator-focused platforms.
Way of working	Primarily uses a desktop to manage their channel and analyze video performance.

Table1.3.2.3: **User Profile for The Administrator**

User Class	**Note on Characteristics**
Type of user	A technical and professional user with privileged access.
Age range	Typically 22-60 years old.
Frequency of use	Daily, as part of their job responsibilities.
Mandatory	Yes, their use is required to manage and maintain the platform.
Computer experience	Expert. Deep understanding of web systems and content management.
Education	Often has a background in IT, Computer Science, or a related field.
goal	To ensure platform safety, handle user reports (e.g., inappropriate content), manage users, and monitor overall system health.
Language skills	Proficient in English and potentially other languages to handle a global user base
Number of users	A very small, specialized team.
Training	Specific training on the platform's administrative tools and moderation policies is required.
Others system use	Admin dashboards, content management systems (CMS), and database management tools.
Way of working	Works from a dedicated administrative interface, likely on a desktop computer.

1.3.3 Elicitation Process

A method for obtaining requirements for DIU_COURSE_TUBE will need to exist so that the needs of the users can be satisfied. As this is a prototype project, I would combine both Direct and Indirect Methods.

Competitive Analysis (Indirect Method): The main method I have used for this project has been to complete competitive analysis from successful video sharing websites such as YouTube. The study of their features, user interface and interaction models would allow us to determine a base line of "must-haves". This allowed me to identify the main features to be included within the project including the like/dislike systems, subscription buttons and share modals.

User Surveys (Direct Method):
I would develop an online survey for potential viewers and content creators.

- **Description**: A combination of multiple choice and open-ended questions would be used to ask users about their preferred features on other platforms, what they dislike, and what they would like to see in a new video sharing platform. This would assist in determining which features to focus on developing first.

User Interviews (Direct Method):
I would conduct individual interviews with a small number of target users representing three different demographics (e.g. casual viewer, power user, content creator).

- Description: Semi-structured interviews would be conducted to ask targeted questions regarding user behavior while at the same time providing users the opportunity to express their opinions freely. This method allows for the discovery of user goals and frustrations through qualitative data.

1.4 Project Block Diagram

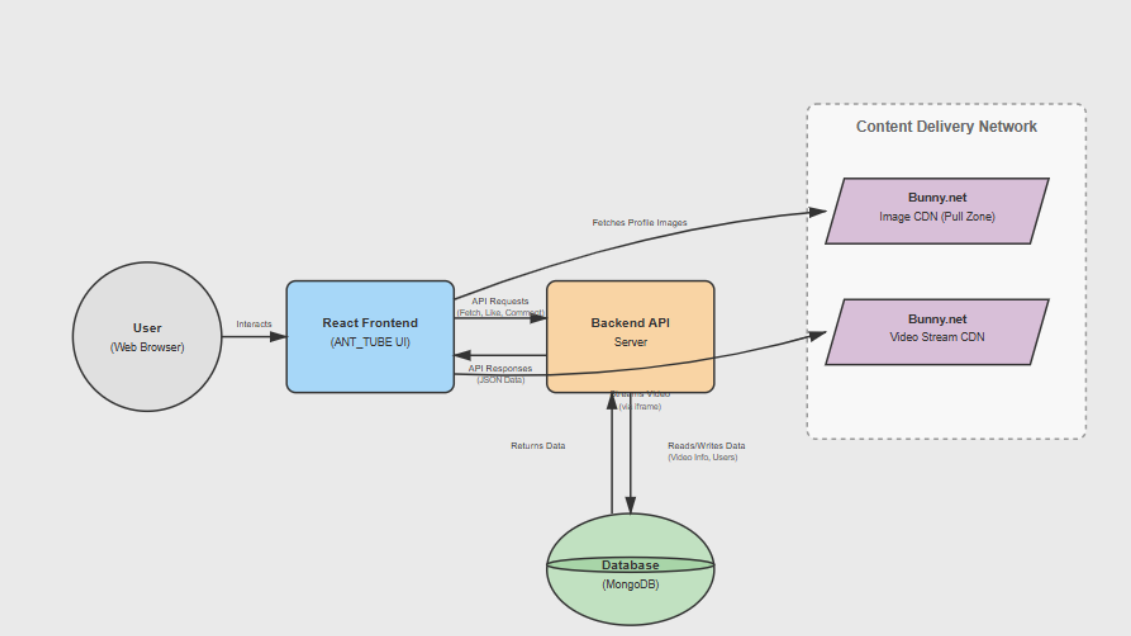


Figure 1: Block Diagram of a System

1.5 Requirement of the System

1.5.1 Needs in Hardware

The hardware requirements for the developer of the project and end-user who uses the videos will be outlined below:

End Users (Video Viewers):

- Computer/Laptop/Tablet/Desktop or Smart phone that supports internet browsing
- High-speed, stable broadband internet access (DSL/Cable/Fiber/4G/5G) to enable smooth viewing of Video Content

Developer's Perspective:

- Multicore processor (i.e., Intel Core i5, AMD Ryzen 5, Apple M1/M2/M3 and above).
- Minimum 8GB RAM for smooth performance; 16GB or more to allow seamless simultaneous use of development server, code editor, browser.

1.5.2 Software Requirements

Like Hardware, the Software Requirements will be different for Developers than for End Users.

End User (Viewers):

- The Operating System that will run on is any recent OS like; Windows 10/11, macOS, Linux, Android, or iOS.
- The Browser that will work best with our application will be one of the latest versions of;
    1. Chrome
    2. Firefox
    3. Edge
    4. Safari

For the Developer:

- The operating systems are Windows 10/11, macOS, and Linux (specifically developer-friendly distributions of Linux, such as Ubuntu).
- A code editor (i.e., a tool used to write, edit, and debug code) is required (e.g., Visual Studio Code; in addition to WebStorm and Sublime Text), in addition to an environment that can run the application (Node.js with LTS version 18.x or higher and npm/yarn/pnpm); and version control for the application's source code (Git).

- An optional tool is an API testing tool (such as Postman) which may be utilized to test the backend API endpoints.

1.5.3 Constraints and Dependencies

This part of the project discusses the constraints of the project, as well as how it relies on external systems to be able to operate effectively.

Constraints (Limitations):

- **Web-Based Application - Internet Dependency:** The entire project is based upon a web interface and will need a continuous active internet connection to run. It has no ability to run in an "offline" state.
- **Browser Compatibility:** Although the project will primarily target modern browsers, there are many features which may not perform properly on older browsers or those which do not have the latest versions of JavaScript (ES6+), and CSS.
- **Prototype Scalability:** The current design and architecture for the project is based upon a prototype model. In order to scale to a million plus users, the project would likely need substantial updates to the existing infrastructure, database optimizations, and/or development of a microservice-based backend.

Dependencies (External Systems):

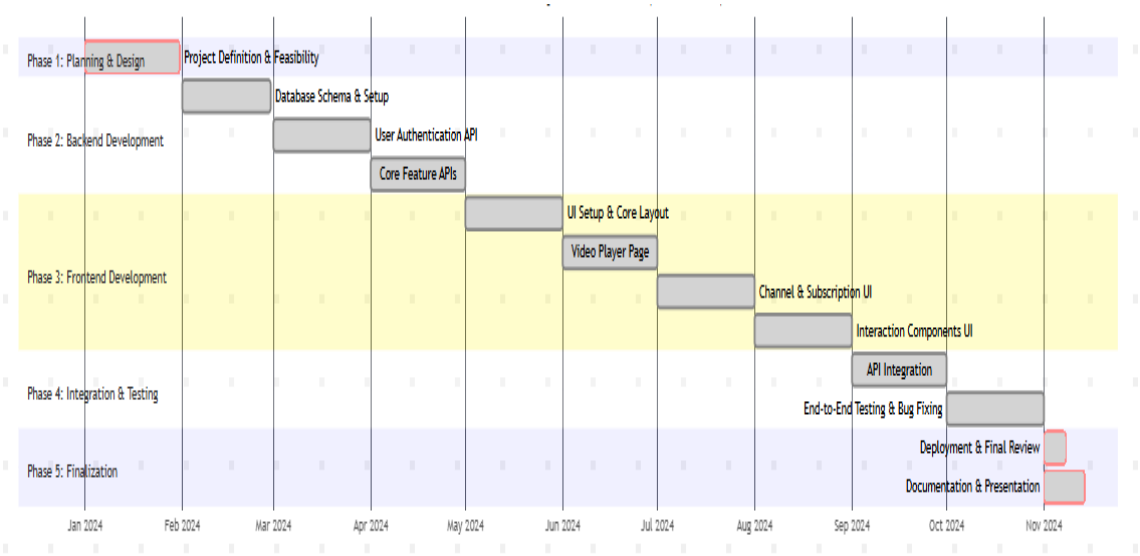
The DIU_COURSE_TUBE project is not a self-contained system; it depends very heavily on numerous other external systems to be able to provide its primary functions:

- **Custom Backend API Server:** The backend API server is fully dependent on the project's custom API server for the functionality of the user interface. Users will not be able to pull video information, login, make comments, or subscribe if the backend API server is unavailable.
- **Database:** The backend API server is dependent upon a database (such as MongoDB) to store user data, video metadata, comments, and subscription information.
- **Bunny.net CDN:** A significant portion of the infrastructure used for the application are dependencies on services hosted by Bunny.net.

  This includes:
  - All media content delivery through Bunny.net CDN.
  - The video streaming is also provided through Bunny.net Stream.
  - Channel image hosting is delivered via Bunny.net CDN Pull Zone.

1.6 Project Scheduling

DIU_COURSE_TUBE Timeline (Jan-Nov)



Explanation of the Timeline

January (Phase 1): The project started out as foundational planning. We established the scope of the project, completed a feasibility study, and outlined the entire system architecture, which included technology selections such as React, Node.js, and Bunny.net.

February-April (Phase 2): This period was dedicated to developing the back end. The first step in development was establishing the database design. Next, we created the user registration/login API. Lastly, we established the API endpoints for each core feature of the site including retrieving video information, processing subscriptions, and handling reaction functionality.

May-August (Phase 3): Once the back end was complete, we transitioned into building the front end user experience. This was the most extensive phase of development during the project where we developed the video watch page, integrated the video player into the page, and developed all the interactive elements of the page including the subscribe button, like/dislike buttons, and comment section layout.

September (Phase 4): During this month, we focused on integrating all the different elements of the front end into the back end API. We made sure the flow of data between the front end and back end was correct and that users' actions were being stored in the database properly.

October (Phase 4): During this month, we dedicated our time to doing a comprehensive test of the entire application. We did end-to-end tests to mimic how actual users would behave on the application, found bugs and corrected them to make sure the application was as stable and reliable as possible.

November 1-13 (Phase 5): The last two weeks of the project were for wrapping everything up. We put the application online, so it could be accessed by anyone. The

rest of the time was used to write the final project report and prepare for the defense presentation.

1.7 Summary

DIU_COURSE_TUBE has been created to be a single web-based solution for students to view educational video content as a whole. This project provides an example of how students and instructors can use web-based solutions to support distance learning.

DIU_COURSE_TUBE utilizes many of the same features that YouTube users have grown accustomed to: viewing videos quickly and efficiently, posting comments, liking/disliking videos, sharing videos and viewing history. However, creating such a system is difficult. Most smaller projects attempt to provide only a few of these functions well. As a result, we decided to build a system that would integrate all of the functions above into a single cohesive application that would allow for high-performance video playback through a CDN.

My goal was to develop a working prototype of a video-sharing platform to demonstrate that it is possible to develop a large-scale, real-world application using current programming languages and development methodologies such as React and TypeScript.

I planned out the feasibility of our project as follows:

Were I Able to Plan This Project Realistically?

- I determined whether or not my project was viable based on its scope, technology and timeline.
- **Scope**: The project had a very specific scope which included the following:
    - 1. Video Playback – utilizing a CDN to deliver high performance video playback
    - 2. User Login
    - 3. Subscription Functionality
    - 4. Like/Dislike Functionality
    - 5. Comment Functionality
    - 6. Share Functionality
    - 7. View History Functionality
- **Technology**: My project was technologically viable due to our utilization of a well-proven and modern tech-stack (React/TypeScript frontend, API backend and Bunny.net CDN for high-performance video delivery)
- **Timeline**: I established a timeline for completing the project and adhered to it from January to November with the following milestones:
    - Jan-Mar: Planning phase
    - Feb-Apr: Backend implementation
    - May-Aug: Frontend implementation
    - Sep-Oct: Integration and Testing

    - Nov: Deployment
- **Target Audience**:
  There are three primary target audiences for my platform:
    - **Viewer**: My viewer audience will be the majority of users; they will simply be watching content and interacting with it.
    - **Content Creator**: The second audience will be content creators who will post content and will want to see how their content is reacted to.
    - **Administrator**: The third audience will be administrators who will be responsible for ensuring the security and continued operation of the platform.

What do I require?

I based upon the following;

- **Hardware**: Any modern pc will work along with good internet for viewing, developers can use modern standard pcs.
- **Software**: Users will be able to view the content in most modern browsers (chrome, firefox, etc.); developers will use Node.js and VS Code.
- **Restrictions**: It needs to be connected to the internet, as well as many of the features depend on the third-party services that we are using (backend api, bunny.net cdn) to provide video streaming.

CHAPTER 2 DESIGN AND IMPLEMENTATION

2.1 Introduction

The objective of Chapter 2 is to establish the context for DIU_COURSE_TUBE. The purpose of Chapter 2 is to provide an overview of the project development of DIU_COURSE_TUBE from its original idea through to the details of the planning phase. At the end of Chapter 2, you should be able to understand the project's history, how DIU_COURSE_TUBE fits into today's Internet world, the problems DIU_COURSE_TUBE addresses and the benefits it provides to both the viewer and content creators.

Chapter 2 also covers the development process of the project. In this area, Chapter 2 identifies the feasibility study carried out to determine if the project could be completed, the identification of the scope of the project to identify where the focus should be placed and the identification of the target user group for the platform (content creators and viewers).

Chapter 2 will also describe the key requirements for the system to operate effectively including hardware and software. A Gantt chart will also be included to illustrate the timeline for the completion of all phases of the project. When I have completed Chapter 2, I will have an understanding of the goals and objectives of the DIU_COURSE_TUBE project, who the target audience is and how the strategy used to develop DIU_COURSE_TUBE was determined.

2.2 Functional Requirements

This area explains the specific tasks that the DIU_COURSE_TUBE system has to accomplish. Each function describes what the system can do, what the system accomplishes, and for whom the system accomplishes those things.

FR01	**User Registration**
Description	New users, whether they plan to watch videos (Viewers) or upload them (Content Creators), must be able to create a personal account to access the platform's features.
Stakeholder	Viewer, Content Creator

FR02	**User Login**
Description	Registered users (Viewers, Creators, and Admins) must be able to log in securely to the system to access their personalized content and role-specific functionalities

Stakeholder	Viewer, Content Creator

FR03	**View Video Content**
Description	All users, including guests who are not logged in, must be able to play and watch videos available on the platform. The system will stream video content efficiently from a CDN.
Stakeholder	Viewer, Content Creator

FR04	**View Channel Information**
Description	When watching a video, the system must display information about the content creator's channel, including their name, profile picture, and the current number of subscribers.
Stakeholder	Viewer

FR05	**Rook Booking**
Description	A logged-in user must be able to subscribe to a content creator's channel. The system will update the subscriber count and reflect the user's subscription status on the UI. A user cannot subscribe to their own channel.
Stakeholder	Viewer, Content Creator

FR06	**React to a Video**
Description	A logged-in user must be able to "Like" or "Dislike" a video. The system will record the reaction, update the like/dislike counts, and prevent a user from giving more than one reaction at a time.
Stakeholder	Viewer

FR07	**Comment on a Video**

Description	A logged-in user must be able to post comments on a video. The system will display all comments associated with that video, allowing for community interaction.
Stakeholder	Viewer, Content Creator

FR08	**Share a Video**
Description	Any user of the system must have the ability to post a video to the courseTube. The system will create a modal window where the user will be able to copy the video link and/or post the video directly to their preferred social media platforms (Facebook, Messenger, Twitter, and WhatsApp) as a way to share the video.
Stakeholder	Viewer, Content Creator

FR09	**Save a Video**
Description	Once a user logs in, they will be able to add a video to a "watch later" or "saved" list that is associated with them and remove a video from that list.
Stakeholder	Viewer

FR10	**Track Viewing History**
Description	For a logged-in user, the system must automatically record the videos they have watched. This creates a personal viewing history for the user.
Stakeholder	Viewer

FR11	**View Related Videos**
Description	The system will also need to show a user a list of recommended/related videos while the user is viewing a video so that they may continue to find more content and engage with other parts of the site.
Stakeholder	Viewer

2.3 Non-Functional Requirements

This section provides the non-functional requirements for the DIU_COURSE_TUBE system. These are similar to the functional requirements because instead of specifying what the system must do, these specify how well the system must do it; in other words, these requirements focus on the performance characteristics of the system such as how fast, stable, etc., the system must be.

2.3.1 Performance

This describes how fast and responsive the DIU_COURSE_TUBE platform must be to the user. For the system to maintain user interest, the system must be very high performance. Some key points about performance include:

1. **Fast video loading**: The time it takes for a video to begin playing after clicking on a video should be less than a few seconds and should require as little buffering as possible. This is accomplished through the use of a high-speed content delivery network (CDN) which streams video content directly to the user's browser.
2. **Responsive UI**: The user interface should respond to all user input (e.g. "like", "subscribe", "save") immediately providing the user with immediate feedback regarding the result of their actions.
3. **Fast page loads:** The main video watch page (the page which includes the video, title, description, comments, etc.) should be able to load rapidly regardless of whether the user has a slow connection to the Internet. The system will display a "loading..." indicator to help manage the user's expectation.

2.3.2 Reliability

Reliability refers to the reliability and consistency of the overall system that users can rely upon for consistent performance. In order for a user to have confidence that their interactions are being recorded and reflected within the system, it must be reliable. Main points:

1. **High availability:** the web site and applications must always be available to users and there should be as little down time as possible. This means the web hosting server must remain stable and that all third party services such as CDNs are stable and working well.
2. **Error handling:** if something happens to prevent normal operation (for example a loss of net connectivity or a failed api) the program should not stop working. Instead, the program should provide the user with a clean error message indicating the problem. Try/catch blocks are used in the code to handle these errors in a graceful manner.
3. **Data consistency:** when a user performs some action (for example likes a video or subscribes to a channel) that action needs to be properly recorded by the system. The system will ensure that what the user sees on the screen accurately represents what has been recorded in the system database.

2.3.3 Portability

This defines how easily the application can run on different environments. Key Points: 1. **Cross-Browser Compatibility**: The DIU_COURSE_TUBE website must function correctly on all major modern web browsers, including Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.

2. **Responsive Design:** The User Interface must be able to adjust automatically based upon the size, providing a good experience on desktops, laptops, tablets, and mobile phones. The layout is built using a responsive grid system to ensure this.

3. **Operating System Independence:** As a web-based application, it is independent of the user's operating system (Windows, macOS, Linux, etc.) and only requires a compatible web browser to run.

2.3.4 Usability

Portability refers to the ease with which an application may be used in various different environments. Key Points:
1. **Cross-Browser Compatibility**: This DIU_COURSE_TUBE Website MUST operate successfully across all modern web browsers (Google Chrome, Mozilla Firefox, Safari, Microsoft Edge).
2. **Responsive Design:** This User Interface MUST adapt itself automatically depending on the screen resolution and provide a quality user experience regardless of whether it is being viewed on a desktop computer or laptop, tablet or mobile phone.
The User Interface was designed using a responsive grid system so that it would fit any of these options.
3. **Independence from Operating Systems:** It does not matter what the user has as their Operating System (Windows, macOS, Linux, etc.), since this is a web-based application, they only need to have a compatible web browser for the application to work.

2.3.5 Security

Security is the measures used to defend the System as well as the User’s data against unauthorized access. Primary Points:

1. Authentication — Users will be required to login in order to perform actions such as posting comments, subscribing to channels, or save videos. This will ensure that all of the actions associated with those users are directly related to their specific user accounts.

2. Authorization — The System utilizes user based roles (for example, Viewer, Creator, Admin) to provide for restrictions of what capabilities/feature the user has access to. For instance, a user can't subscribe to their own channel.

3. Data Protection — All user based information as well as all of the interactions between users and the System are passed through a secure backend API that protects sensitive data from being potentially visible on the frontend of the System.

2.4 Object Oriented Systems Design Using UML

2.4.1 Use Cases Diagram

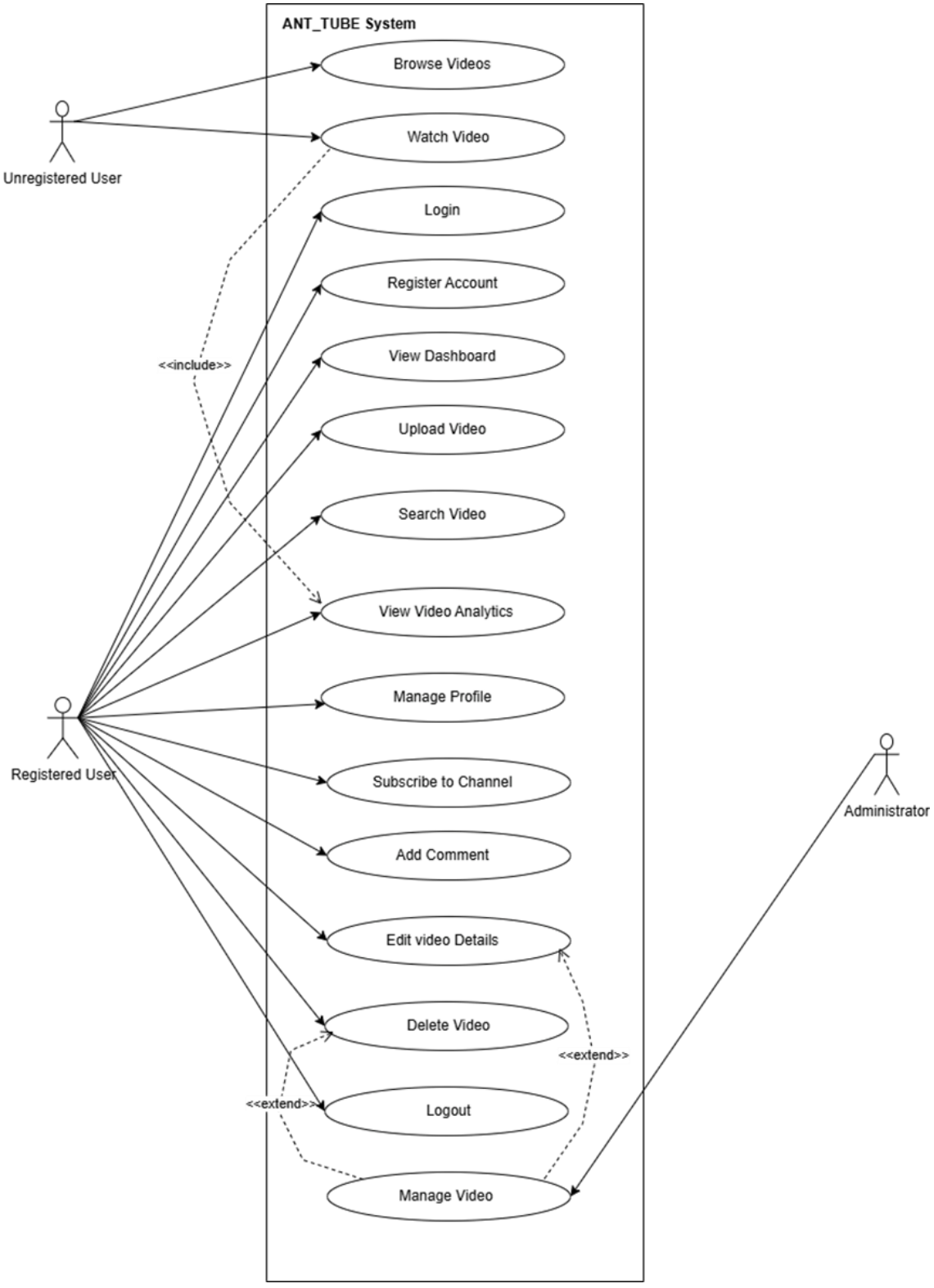


Figure 2: Use case Diagram

2.4.2 Case Description

Case Description- 2.4.2.1: **User Registration**

<table>
<tr><td>Use Case</td><td>User Registration</td></tr>
<tr><td>Goal</td><td>A new user can create a personal account to access features like subscribing, commenting, and saving videos.</td></tr>
<tr><td>Precondition</td><td>The user must have access to the DIU_COURSE_TUBE website via a web browser.</td></tr>
<tr><td>Success End Condition</td><td>Account creation for the user is successful, and a Success Notification is shown to the user.</td></tr>
<tr><td>Failed End Condition</td><td>Account creation was unsuccessful; an appropriate Error Message is shown to the user (for example, "Email Already In Use") .</td></tr>
<tr><td>Primary Actors:
Secondary Actors:</td><td>New User (Viewer or Content Creator)
DIU_COURSE_TUBE System (Backend, Database)</td></tr>
<tr><td>Trigger</td><td>Clicking the "Sign Up" or "Register" link from the website.</td></tr>
<tr><td>Description
Success
Scenario</td><td>
<table>
<tr><td>1.</td><td>The user clicks the "Register" button.</td></tr>
<tr><td>2.</td><td>Registration Form is shown by the system to the user.</td></tr>
<tr><td>3.</td><td>Required information for the Registration Form is entered by the user (for example, name, email, password).</td></tr>
<tr><td>4.</td><td>Users click the “Create Account” button.</td></tr>
<tr><td>5.</td><td>System validate User’s Input and send it to Backend Server.</td></tr>
<tr><td>6.</td><td>System create New User Record on Database & Confirm Success!</td></tr>
<tr><td>7</td><td>The system displays a "Registration Successful!" notification and logs the user in.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>Alternative Flows</td><td><table>
<tr><td>1.1</td><td>The user enters an invalid email format.</td></tr>
<tr><td></td><td>1.1.a. The system displays an inline error message: "Required a valid email or Enter a valid email."</td></tr>
<tr><td>2.1</td><td>The user leaves a required field blank.</td></tr>
<tr><td></td><td>2.1.a. The system highlights the empty field and displays a message: "This field is required."</td></tr>
<tr><td>3.1</td><td>The submitted email address already exists in the database.</td></tr>
<tr><td></td><td>3.1.a. The system displays an error notification: "An account with this email already exists. Please log in."</td></tr>
<tr><td>4.1</td><td>A network or server error occurs during submission.</td></tr>
<tr><td></td><td>4.1.a. The system displays an error notification: "Failed to create account. Please try again later."</td></tr>
</table></td></tr>
<tr><td>Quality Requirements</td><td>A completed registration should take less than 3 seconds to submit. From the moment of submitting your registration to receiving your confirmation should be no longer than 5 seconds over a stable Internet connection.</td></tr>
</table>

Case Description-2.4.2.2: **Subscribe to a Channel**

Use Case	Subscribe to a Channel
Goal	A logged-in user can follow a content creator's channel to stay updated on their new content.
Precondition	Users attempt to Login by clicking the "Login" button and logged into their DIU_COURSE_TUBE account and on the Video Watch Page.
Success End Condition	The user is successfully subscribed to the channel, the subscriber count updates, and the button changes to a "Subscribed" state.
Failed End Condition	The subscription action fails, and an error notification is shown
Primary Actors: Secondary Actors:	Viewer DIU_COURSE_TUBE System
Trigger	The user clicks the "Subscribe" button on a channel's page.

<table>
<tr><td>Description / Main

Success

Scenario</td><td><table>
<tr><td>1.</td><td>User attempt to Subscribe to Channel by clicking the "Subscribe" button</td></tr>
<tr><td>2.</td><td>Frontend Send request to Backend API to Add Subscription.</td></tr>
<tr><td>3.</td><td>The system verifies the user is not subscribing to their own channel.</td></tr>
<tr><td>4.</td><td>System Save subscription to Database.</td></tr>
<tr><td>5.</td><td>System Update Channel Subscriber Count.</td></tr>
<tr><td>6.</td><td>The frontend UI updates to show the "Subscribed" button and the new subscriber count.</td></tr>
</table></td></tr>
<tr><td>Alternative Flows</td><td><table>
<tr><td>1.1</td><td>User Not Logged In.</td></tr>
<tr><td></td><td>1.1.a. System Prompt User to “Please Log In To Subscribe.”</td></tr>
<tr><td>2.1</td><td>The user attempts to subscribe to their own channel.</td></tr>
<tr><td></td><td>2.1.a.The "Subscribe" button is disabled or hidden, preventing the action.</td></tr>
<tr><td>3.1</td><td>A network or server error occurs.</td></tr>
<tr><td></td><td>3.1.a. The system displays an error notification: "Failed to subscribe. Please try again."</td></tr>
</table></td></tr>
<tr><td>Quality Requirements</td><td>The UI must update to reflect the subscription status in under 1 second after the user clicks the button (optimistic update).</td></tr>
</table>

Case Description-2.4.2.3: **React to a Video**

Use Case	React to a Video (Like/Dislike)
Goal	A logged-in user can express their opinion on a video by liking or disliking it.
Pre-condition	Users Must Be Logged In and Viewing Video.
Success End Condition	The user's reaction is recorded, the like/dislike count is updated, and the button is highlighted to show their selection.

<table>
<tr><td>Failed End Condition</td><td>The reaction is not saved, and an error message is displayed.</td></tr>
<tr><td>Primary Actors:

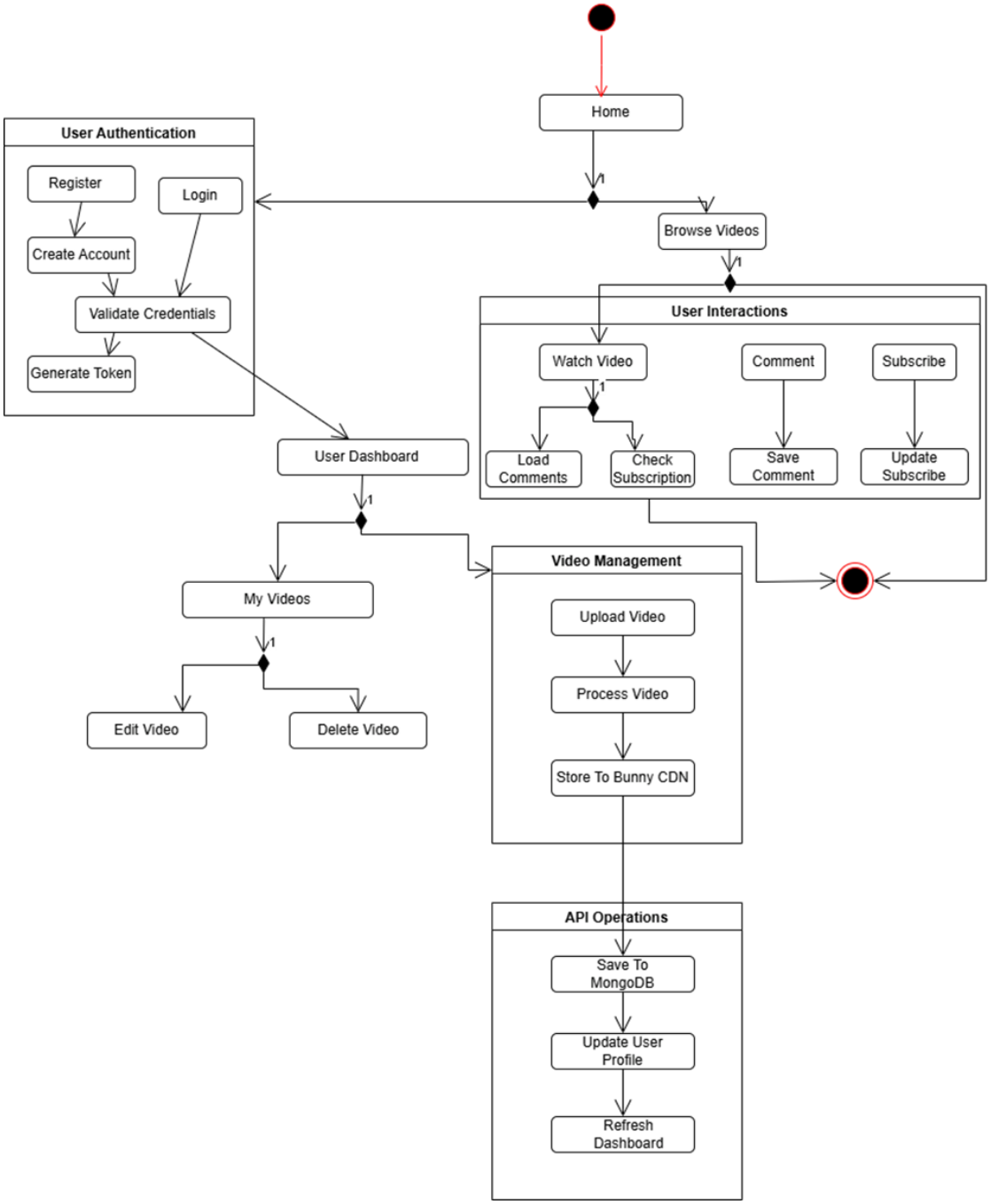
Secondary Actors:</td><td>Viewer

DIU_COURSE_TUBE System</td></tr>
<tr><td>Trigger</td><td>User Click “Like” (or) “Dislike” (button).</td></tr>
<tr><td>Description

Success

Scenario</td><td>
<table>
<tr><td>1.</td><td>User Clicks “Like” Button.</td></tr>
<tr><td>2.</td><td>The UI immediately highlights the "Like" button and increases the like count (optimistic update).</td></tr>
<tr><td>3.</td><td>The frontend sends a request to the backend API to record the "like" reaction.</td></tr>
<tr><td>4.</td><td>System Save User Reaction to Database.</td></tr>
<tr><td>5.</td><td>System Return Final Counts of Reactions to Frontend.</td></tr>
</table>
</td></tr>
<tr><td>Alternative Flows</td><td>
<table>
<tr><td>1.1</td><td>The user clicks the "Like" button when they have already liked the video.</td></tr>
<tr><td></td><td>1.1.a. The system removes the "like" reaction, and the UI un-highlights the button and decrements the count.</td></tr>
<tr><td>2.1</td><td>The user clicks the "Like" button when they have already disliked the video..</td></tr>
<tr><td></td><td>2.1.a.The system switches the reaction from "dislike" to "like," updating both counts and button states accordingly</td></tr>
<tr><td>3.1</td><td>The backend API call fails.</td></tr>
<tr><td></td><td>3.1.a. The system reverts the UI to its original state before the click and shows an error: "Failed to add reaction."</td></tr>
</table>
</td></tr>
<tr><td>Quality Requirements</td><td>The user interface must provide immediate visual feedback (< 200ms) upon clicking a reaction button.</td></tr>
</table>

2.4.3 Activity Diagram



2.4.4 Sequence Diagram

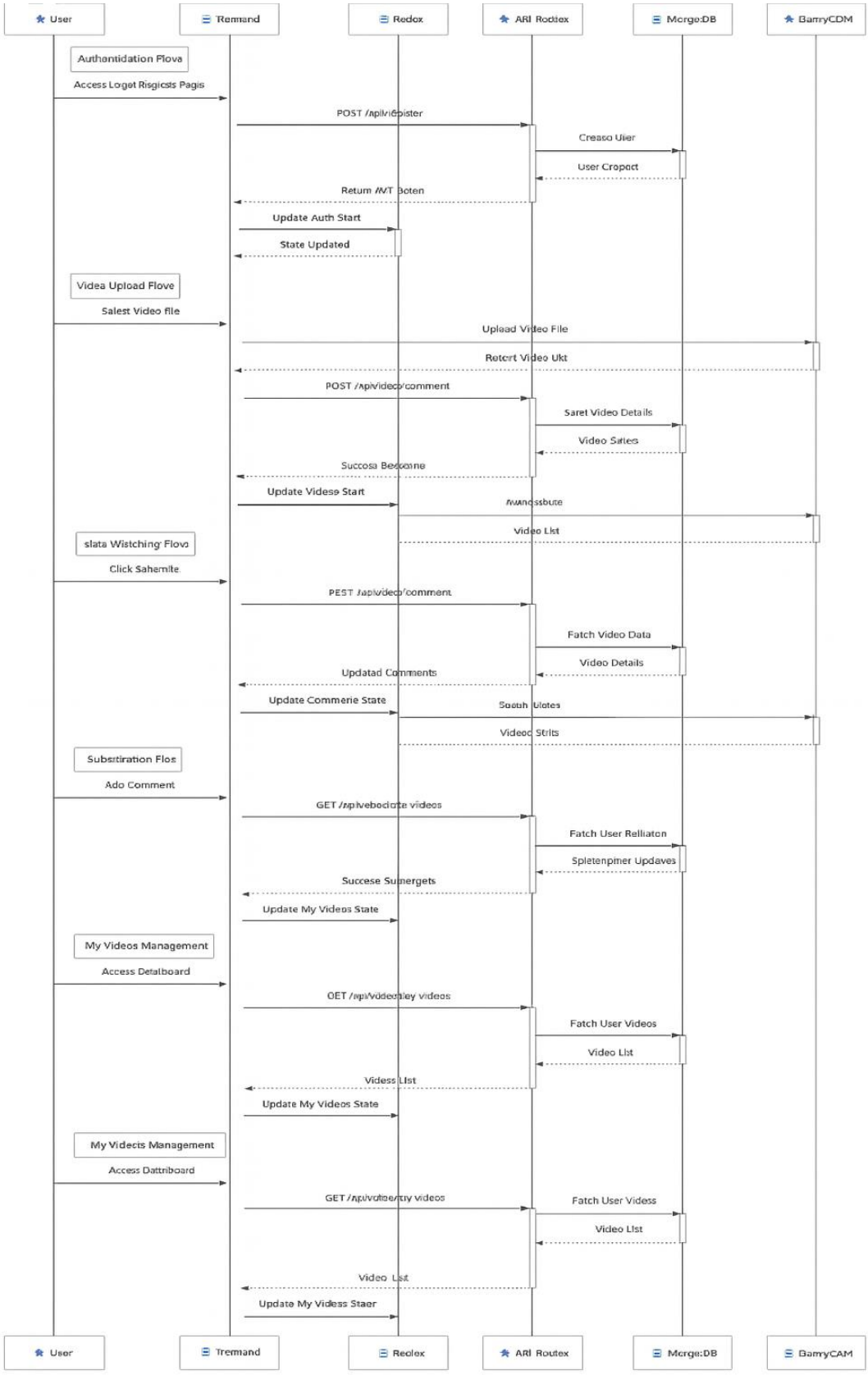
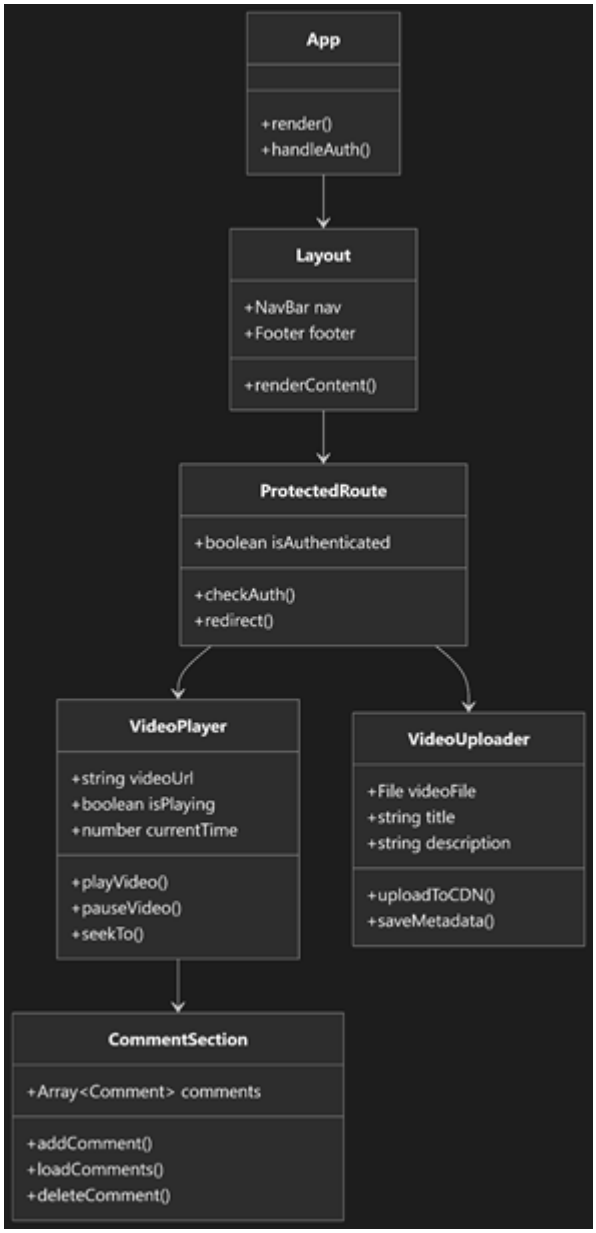


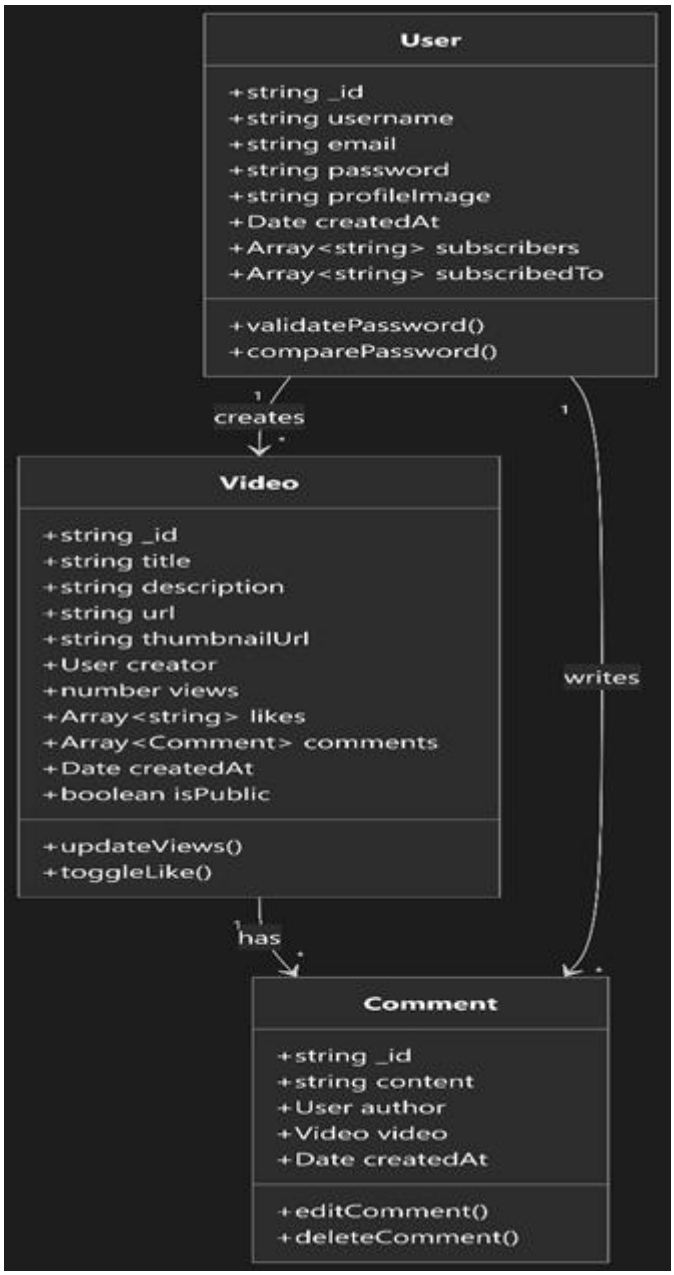
Figure 3: Add Profile

2.4.5 Class Diagram

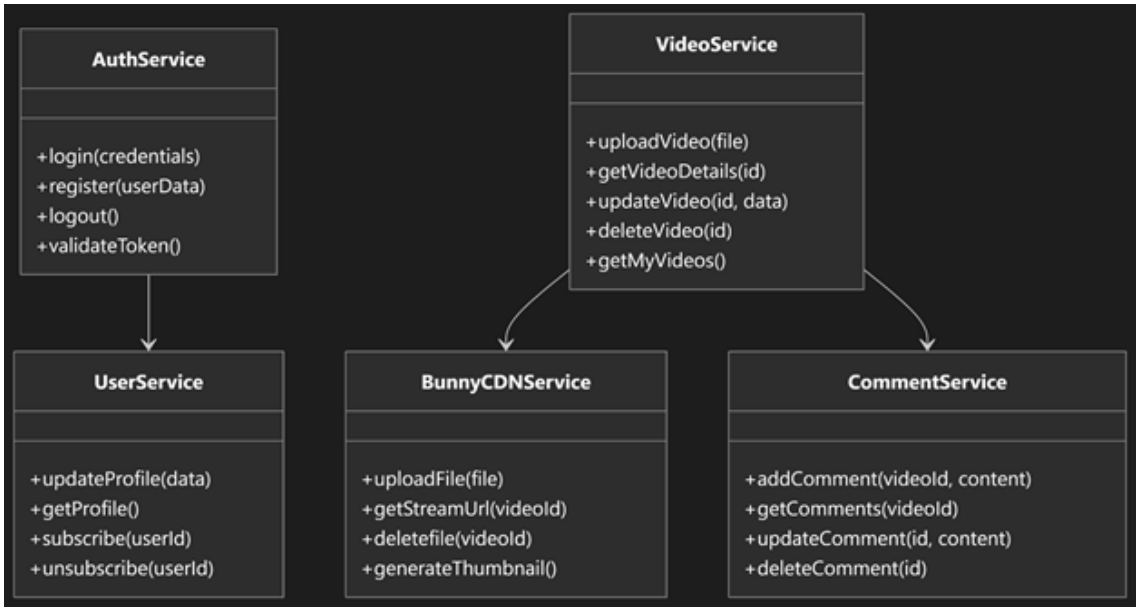
Core domain model:



Frontend Component Structure:



API/Service Layer:



Redux State Management:

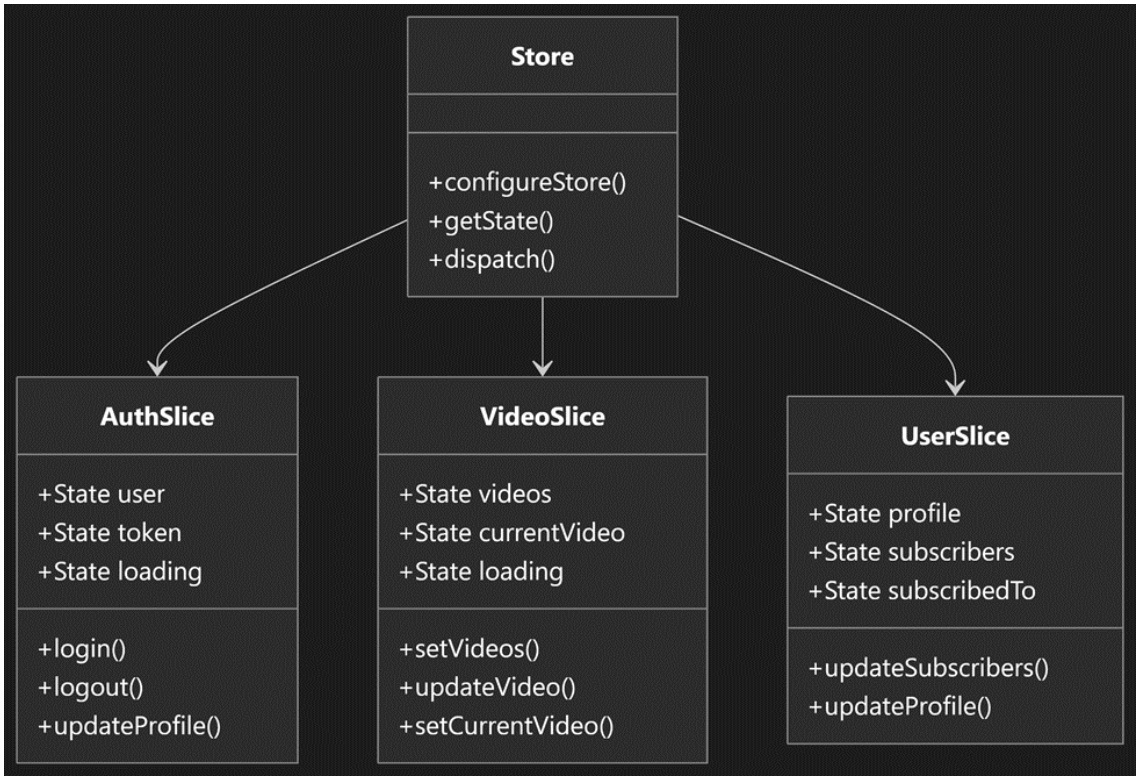


Figure 4: Class Diagram

2.4.6 ERD Diagram

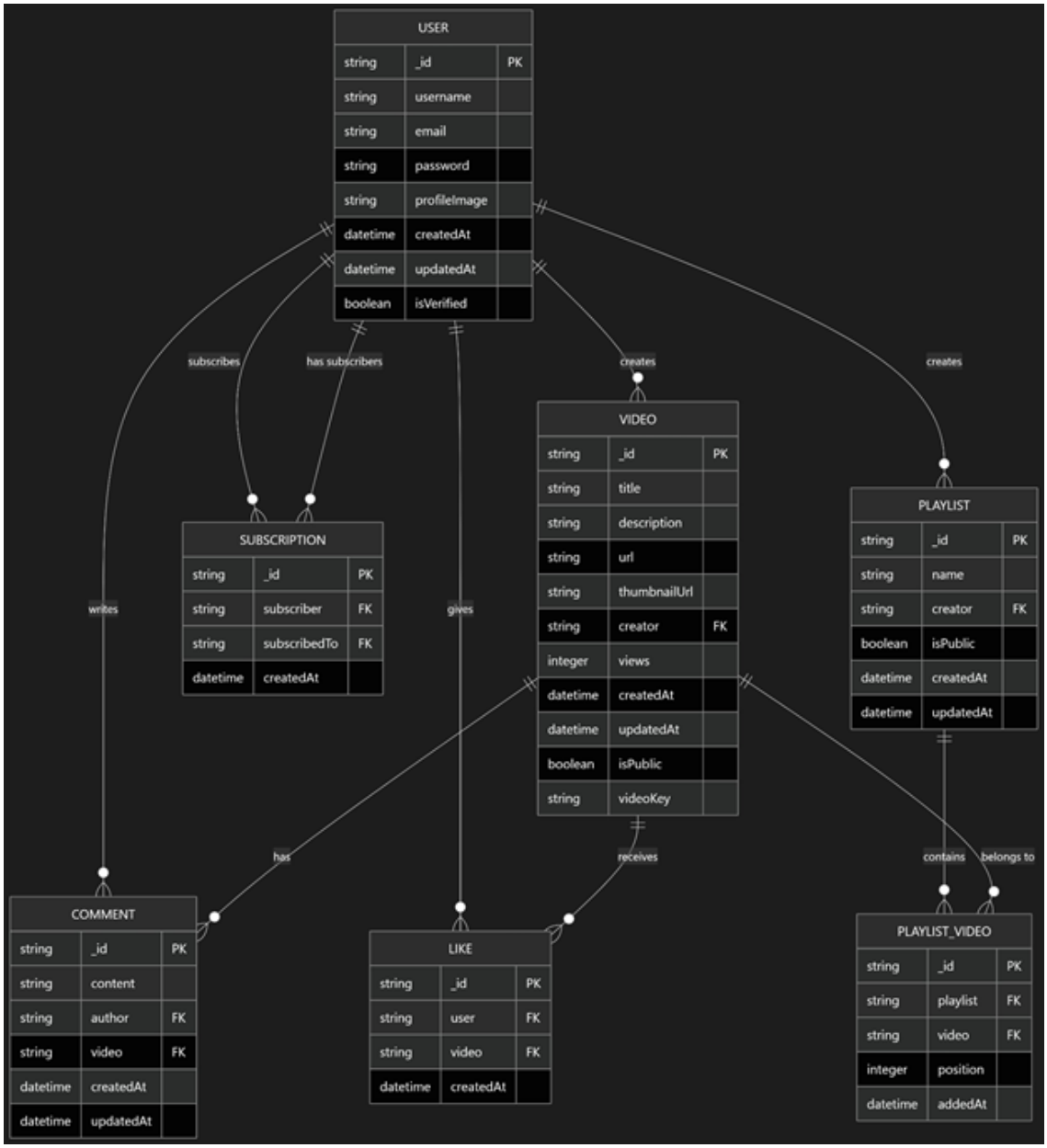


Figure-5: ERD Diagram

2.5 Coding: Appendix A

This appendix has a few examples of the code used in the DIU_COURSE_TUBE application. It has been extracted from the key video-watching page for this application (WatchVideo.tsx). The examples provided below include code that illustrates how to get ALL of the required video information from the back end and how to handle user interaction using an optimistic user interface.

Sample 1: Fetching Video and Channel Data

The fetchVideoData function is responsible for obtaining all the information required by the video watch page. This function will make a single API request to the server when the WatchVideo component is mounted. Once it receives the response from the server it will process that data and update the state of the component so that the video player, channel details, whether or not the viewer has subscribed to the channel, and how many likes each video has can be displayed.

```
//c:\All Projects,Code\Projects\ANT_TUBE\src\components\pages\watch_video
\index.tsx
// ... (state declarations for video, channeDetails, likeCount, etc.)
 // Function to fetch all data related to the video being watched

 async function fetchVideoData() {
  try {
   // Reset any previous errors
   setError('');
   // Make a GET request to the backend API to get video details
   const res = await fetch(`/api/video/get-video-details?_id=${_id}`, {
    method: 'GET',
    headers: {
     // Pass user ID for personalized data like subscription status
     Authorization: user?._id || '',
   },
   });
   if (!res.ok) {
    console.logr('Failed to fetch video data');
```

```
    }
    // Parse the JSON response from the server
    const data = await res.json();

    // --- Set Component State with Fetched Data ---
    // Get the latest channel profile image and construct its CDN URL
    let len = data.channel.image.length > 0 ? data.channel.image.length - 1 : 0;
    let profileImageInfo = data.channel.image[len];
    setProfile(getBunnyCDNUrl(profileImageInfo));

    // Check if the current user has already saved this video
    const isVideoSaved = Array.isArray(data?.saved)
      ? data.saved.some((save: any) => save.userId === user?._id)
      : false;
    setIsSaved(isVideoSaved);

    // Set all relevant states to re-render the UI
    setVideos(data); // Video metadata (title, description, etc.)
    setChanneDetails(data?.channel); // Channel details
    setNumberOfSubscribe(data?.numberofSubscribe); // Subscriber count
    setIsSubscribed(data?.isSubscribed); // Is the current user subscribed?
    setLikeCount(data?.like?.likecount); // Total likes
    setDislikeCount(data?.dislike?.dislikecount || 0); // Total dislikes

  } catch (e: any) {
    // If an error occurs, store the error message to display to the user
    setError(e.message);
  } finally {
    // Set loading to false to hide the loading indicator
```

```
    setLoading(false);
  }
 }

 // useEffect hook to call fetchVideoData when the component mounts or _id changes
 useEffect(() => {
  if (_id) fetchVideoData();
 }, [_id]);
```

Sample 2: Processing User Reactions using Optimistic UI

The handleReaction method is responsible for processing the likes/dislikes. Using an "Optimistic UI" approach, it updates the user's view of the UI as soon as they click on the button. After updating the UI, it sends a request to the backend. If that call to the backend is unsuccessful for whatever reason, it will revert the UI back to how it was before, so that the data will be consistent.

```
//c:\AllProjects ,Code\Projects\ANT_TUBE\src\components\pages\watch_video
\index.tsx
 // Function to handle user's like or dislike reaction
 const handleReaction = async (type: 'like' | 'dislike') => {
  // Prevent non-logged-in users from reacting
  if (!user?._id) {
   toast.error('Please log in to react to videos');
   return;
  }

  // Store the current state in case we need to revert
  const previousLikeCount = likeCount;
  const previousDislikeCount = dislikeCount;
  const previousUserReaction = userReaction;

  try {
```

```
// --- 1. Optimistic UI Update ---
// Calculate the new state before the API call
let newLikeCount = likeCount;
let newDislikeCount = dislikeCount;
let newUserReaction = userReaction;

if (userReaction === null) { // User has not reacted yet
  type === 'like' ? newLikeCount++ : newDislikeCount++;
  newUserReaction = type;
} else if (userReaction === type) { // User is removing their reaction
  type === 'like' ? newLikeCount-- : newDislikeCount--;
  newUserReaction = null;
} else { // User is switching their reaction (e.g., from dislike to like)
  if (type === 'like') {
    newLikeCount++;
    newDislikeCount--;
  } else {
    newDislikeCount++;
    newLikeCount--;
  }
  newUserReaction = type;
}

// Update the UI immediately with the new optimistic state
setLikeCount(newLikeCount);
setDislikeCount(newDislikeCount);
setUserReaction(newUserReaction);

// --- 2. API Call ---
```

```
    // Send the reaction to the backend to be saved in the database
    const response = await fetch( '/api/video/addreaction', {
      method: 'POST',
      body: JSON.stringify({ _id, type, id: user?._id, role: user?.role }),
      headers: { 'Content-Type': 'application/json' },
    });

    if (!response.ok) {
      // If the API call fails, throw an error to trigger the catch block
      throw new Error('API request failed');
    }

    // --- 3. Sync with Server (Optional but good practice) ---
    // Update state with the definitive data from the server response
    const result = await response.json();
    setLikeCount(result.data.likes);
    setDislikeCount(result.data.dislikes);
    setUserReaction(result.data.userReaction);

  } catch (err) {
    // --- 4. Rollback on Failure ---
    // If any error occurred, revert the UI to its original state
    console.error('Error toggling reaction:', err);
    toast.error('Failed to update reaction');
    setLikeCount(previousLikeCount);
    setDislikeCount(previousDislikeCount);
    setUserReaction(previousUserReaction);
  }
};
```

2.6 Summary

This chapter explains the complete plan for how the DIU_COURSE_TUBE project was designed and built.

First, I defined exactly what the system must do (Functional Requirements). This includes all the key user actions, such as registering an account, logging in, watching videos, subscribing to channels, liking, commenting, and sharing content.

After that I set up my quality requirements for the site (Non-Functional Requirements). I made sure the system will be fast, it will be reliable, it will be secure, and it will be very simple for everyone to use; it will also work with every current web browser and device. Then I created the system's design by using standard UML diagrams. These diagrams, such as the Use Case Diagram, the Sequence Diagram, and the ER Diagram, help us to see how the users will interact with the system, how the different pieces of software will communicate, and how the data will be organized in the database.
Lastly, I have included actual code examples from the project to demonstrate how these designs were used to create an operational version of this application. The code displays how I built critical functionality of the app (like retrieving video data from the server), and how I was able to build a high response rate for the user.

Chapter 3 Software/Application Testing

3.1 Introduction

The Testing Phase – Software Testing is a critical component for the successful implementation of the DIU_COURSE_TUBE video sharing application; therefore the purpose of this testing phase is to validate that the entire functionality of the application works correctly, to identify and correct any bugs or errors, and to provide assurance that the completed product will be reliable, consistent, and provide a quality user experience. In addition, this chapter will outline the major components of the testing process to provide assurance that the fundamental functions of the video sharing application are being met by the DIU_COURSE_TUBE Project. The testing process included validating every user interface (UI) action from registration through viewing and engaging in content on videos.

3.2 Testing Features

3.2.1 Feature to be Tested

- **User Registration**:
  Ensure to be able to register as a new user by using valid login information and ensure to prevent users from registering with an email that has already been used by another user. Ensure error messages appear when bad information is entered (i.e. password too weak or email address is not formatted properly).
- **User Login**:
  Test to see if you can login into your user account with valid login information and test to see if the application will deny your login attempt when entering the wrong login information and display an error message.
- **Video Load and Display**:
  Test to see if the video player loads the video content from the Content Delivery Network (CDN) as expected. Test to see if all of the additional video information (video title, description, channel name etc.) is loaded from the server and is displayed on the page. Test to see if a loading screen displays while the data is being loaded.
- **Channel Subscriptions**:
  - Examined whether an authenticated user can subscribe and un-subscribe from a Channel.
  - Confirmed the subscribers count for the channel and the state of the subscription button (i.e., “Subscribe” or “Subscribed”) are both updated accurately after performing the actions.
  - Examined whether an authenticated user can subscribe to their own Channel.

- **Video Reactions (Likes/Dislikes):**

    - Examined whether an authenticated user can react with either likes or dislikes on a video.
    - Examined whether the total number of likes/dislikes are updated correctly.
    - Examined whether an authenticated user can toggle between like/dislike reactions (or remove them altogether).
- **Save and Unsave A Video**:
    - Examined whether an authenticated user can save a video into their personal list of videos.
    - Examined whether the “Save” button will be updated to show “Saved”.
    - Examined whether an authenticated user can un-save a video they had previously saved to their list of videos.
- **Share a Video**:
    - Examined whether the share modal opens appropriately upon clicking the “Share” button.
    - Examined whether the “Copy Link” feature functions as intended.
    - Examined whether the social media share buttons correctly open a new tab for each respective platform’s share modal.
- **Posting a Comment**:
    - Are users able to leave comments when they are logged into the site?
    - Can users see their comment(s) posted as soon as they have posted them?
- **User Logout**:
    - Can users successfully logout from the site?
    - When users logout, will they lose their ability to view all of their comments, subscribe etc.? (i.e. are all of those options removed?)

3.3 Testing Strategies

3.3.1 Test Approach

My testing for the DIU_COURSE_TUBE project was largely accomplished through manual testing, which provides an effective opportunity to evaluate the user experience from the viewpoint of a real user. My manual testing included a variety of approaches to testing, some of the primary ones include:

Functional Testing: This was the bulk of my strategy. I ran manual test cases on each feature identified in the functional specifications. To give you an idea of how this worked, to test the "Subscribe" function, a tester would have logged into the system, gone to a video and clicked on the "Subscribe" button, then verified whether the "Subscribe" button changed to reflect that the user had subscribed to the channel and if the subscriber count

reflected an increase in subscribers. Each feature was tested with the same methodology; these included testing for the ability to login, add comments, like videos, save videos, etc.

Usability Testing: I attempted to make sure the platform was both intuitive and simple to use. Testers used the website similar to how a first time user would be expected to use it, verifying to see if the layout made sense, if buttons and icons were self-explanatory, and if the flow of completing tasks felt natural and fluid. The goal was to ensure that a user could complete their objective (such as viewing a video or adding a comment), without feeling confused or frustrated about how to do so.

Error Handling and Robustness Testing: I have tested Error Handling and Robustness testing by attempting to "crash" the system and observe its reaction when different types of failures occur. This has involved entering information into fields which are required for form submission and attempting to access or use some features of the system before logging in. In addition, I simulated different failure cases for backend API interactions to test whether the application will either be able to handle such an occurrence (and display an appropriate error) or crash and provide no feedback to the user. As expected, the application did not crash but provided a clear and user-friendly error message when errors occurred (for example "You need to login to post comments").

The try...catch blocks embedded within the code allowed the application to behave appropriately during these tests and prevented errors from displaying to the end-user.

Compatibility Testing: To confirm the DIU_COURSE_TUBE is accessible to all users regardless of their environment (i.e. operating system and/or browser), I conducted compatibility testing by running the site on each version of the three most widely used web browsers currently available: Google Chrome, Mozilla Firefox, Microsoft Edge. By testing in this way, I confirmed the layout, styling and functional behavior of the site were consistent across all three browser platforms.

3.3.2 Pass/Fail Criteria

I created objective guidelines for determining whether a test was successful (passed) or unsuccessful (failed).

A test case is considered successful when it meets the following three requirements:

- **Expected Results Were Achieved:** The functionality acted exactly as described by the requirements. For example, clicking the “like” button increased the number of likes correctly.
- **No Unforeseen Error Occurred:** There were no instances where the application froze, crashed, or produced an unhandled error message in either the user interface or browser console upon performing an action.

- **Data Was Handled Correctly:** All data was correctly stored to the database and shown correctly on the user interface. For example, a new comment would appear in the comments section after submitting it.
- **User Interface Updated Correctly:** The user interface provided the proper visual feedback after a user interaction. For example, after a subscription occurred, the button's text changed from "Subscribe" to "Subscribed."

The "Failure" status of a Test Case will be based upon whether at least one of the following occurs during execution of the Test Case.

- Incorrect Functionality: The feature did not do what it was supposed to do. For example, a click on "Save Video" did not save the video to the user's saved videos list.
- Application Failure (Crash/Freeze): The user's actions caused the application to stop working or shut-down unexpectedly.
- Incorrect Error Handling: The system produced an error message that was cryptic, unhelpful, or failed to produce an error message where one was expected.
- UI/Visual Errors: There were errors with the user interface updating properly, or there were obvious visual errors such as buttons off center and/or text over other objects.
- Inconsistent Data: The information being presented in the user interface is different from the information stored in the back-end database. For example; the user clicks the "like" button on a video and the counter increases on the screen, however the increase is not reflected in the database.

3.4 System Testing & Test Case With Report

Test Case 01: User Registration

Test Case:	TC-01	Test Case Name:	User Registration
System:	DIU_COURSE_TUBE	Subsystem:	User Authentication
Designed by:	Shiddhartha Pal	Design Date:	29-03-2025
Executed by:	Shiddhartha Pal	Execution Date:	29-08-2025

Description: A new user attempts to create an account on the DIU_COURSE_TUBE platform by providing registration information.

Pre-condition: The user is on the registration page of the ANT_TUBE website.

Step	Name	Email	Password	Confirm Password	Expected Response	Pass/ Fail	Comment
1	Shiddhartha Pal	shiddhartpal01355@gmail.com	11111 1Aa@	111111Aa@	"Registration Successful!" notification.	Pass	The user is successfully registered with valid information.
2	Probin Pal	probin2017@gmail.com	11111 1Aa@	111111Aa@	"Name is required" error message.	Pass	The system correctly validates that the name field cannot be empty.
3	Sanjoy Pal	sanjoy2017@gmail.com	11111 1Aa@	111111Aa@	"Email is required" error message.	Pass	The system correctly validates that the email field cannot be empty.
4	Sajol Pal	sajol2017@gmail.com	11111 1Aa@	111111Aa@	"Password is required" error message.	Fail	The system correctly validates that the password field cannot be empty.
5	Test User	test@example.com	password123	password456	"Passwords do not match" error message.	Pass	The system correctly identifies a mismatch between the password fields.

Post-condition: Upon successful registration, a new user account is created in the database, and the user is automatically logged into the system.

Test Case 02: Subscribe to a Channel

Test Case:	TC-02	Test Case Name:	Subscribe to a Channel
System:	DIU_COURSE_TUBE	**Subsystem:**	Video Interaction
Designed by:	Shiddhartha Pal	**Design Date:**	29-03-2025
Executed by:	Shiddhartha Pal	**Execution Date:**	29-08-2025

Description: A logged-in user attempts to subscribe to or unsubscribe from a content creator's channel.

Pre-condition: The user is logged into their account and is viewing a video on the "Watch Video" page.

Step	Action	Expected Response	Pass/Fail	Comment
1	The user (not logged in) clicks "Subscribe".	A prompt or toast notification appears: "Please log in to subscribe".	Pass	The system correctly blocks non-authenticated users from subscribing.
2	Logged-in users click "Subscribe" on a channel they are not subscribed to.	The button text changes to "Subscribed", and the subscriber count increases by one.	Pass	The subscription action is successful.
3	Logged-in users click the "Subscribed" button on a channel they are already subscribed to.	The button text changes back to "Subscribe", and the subscriber count decreases by one.	Pass	The unsubscription action is successful.
4	A content creator views their own video.	The "Subscribe" button is disabled or not visible.	Pass	The system correctly prevents users from subscribing to their own channel.

Post-condition: The user's subscription status for the channel is correctly updated in the database and reflected on the user interface.

Test Case 03: React to a Video (Like/Dislike)

Test Case:	TC-03	Test Case Name:	React to a Video
System:	DIU_COURSE_TUBE	**Subsystem:**	Video Interaction
Designed by:	Shiddhartha Pal	**Design Date:**	29-03-2025
Executed by:	Shiddhartha Pal	**Execution Date:**	29-08-2025

Description: A logged-in user gives feedback on a video by clicking the "Like" or "Dislike" button.

Pre-condition: The user is logged in and is on the "Watch Video" page.

Step	Action	Expected Response	Pass/ Fail	Comment
1	The user clicks the "Like" button for the first time.	The "Like" button becomes highlighted, and the like count increases by one.	Pass	The "like" reaction is successfully recorded.
2	The user clicks the "Like" button again.	The "Like" button becomes un-highlighted, and the like count decreases by one.	Pass	The "like" reaction is successfully removed.
3	Users click the "Dislike" button when they have already liked the video.	The "Like" button becomes un-highlighted, the "Dislike" button becomes highlighted, the like count decreases, and the dislike count increases.	Pass	The system correctly switches the user's reaction from "like" to "dislike".
4	The user (not logged in) clicks the "Like" button.	A prompt or toast notification appears: "Please log in to react".	Pass	The system correctly prevents non-authenticated users from reacting.

Post-condition: The user's reaction to the video is correctly stored in the database, and the like/dislike counts are updated on the UI.

3.5 Summary

This section of the chapter discusses the process of performing software tests on the DIU_COURSE_TUBE project. The major objective of this testing phase is to confirm all of the features in the application are working properly, identify and correct any error or bugs which may have been created during development, and confirm that the final version of the application is reliable and provides an acceptable user experience.

What was Tested: The aspects of the application that were tested included key functionalities such as user registration, user login, video playback, subscribing to channels, liking/disliking videos, saving videos, sharing videos, commenting on videos and logout. These functionalities were tested to verify whether the application complied with the project's requirements.

How it was Tested: The majority of the testing was manual in nature by the tester to simulate how an end-user would interact with the system. The manual testing included:

- **Functional Testing**: Test cases were run through to verify each functional area of the application executed as expected.
- **Usability Testing:** An evaluation of the usability and ease of use of the application was made.
- **Error Handling Testing:** An attempt was made to "crash" the system (for example; attempting to react to a video while not logged in) to determine if there were clear, usable error messages generated by the application rather than having it fail completely.
- **Compatibility Testing:** Verification was done to ensure that the website displayed properly and functioned as expected in the most popular internet browsers, including but not limited to, Google Chrome, Mozilla Firefox, and Microsoft Edge.

Pass/Fail Criteria: Test Case Marked Pass for Actual Result = Expected Result, No Unexpected Errors Occurred & Data Handled Correctly. Test Case Marked Fail if Feature Did Not Work As Intended, Caused Crash, Displayed Wrong Information & Documented as Bug.

Chapter 4 - Deployment and Maintenance

4.1 Introduction

This chapter outlines the ultimate and on-going phases of the DIU_COURSE_TUBE Project: deployment and maintenance. Deployment is the process of taking the finished application from a developer's computer and making it live on the internet so that anyone can access and use it. Maintenance describes the continuous effort required to keep the application running smoothly, fix any bugs that appear, and add new features over time. This chapter will explain how the DIU_COURSE_TUBE platform was deployed and outline the strategy for its long-term support, following the standard Software Release Life Cycle (SRLC).

4.2 To follow the SRLC (Software Release Life Cycle)

DIU_COURSE_TUBE software's development & deployment was based upon a formalized, structured life cycle, to deliver an optimal quality, stable software release.

Stage 1: Development In this initial stage of development all coding & feature creation was completed. The front-end was created utilizing React & TypeScript while the back-end API was created to manage all business logic. All coding for each feature (i.e., video play-back, user registration, comments, subscriptions) was accomplished during this stage as described in previous chapters.

Stage 2: Testing (Pre-Alpha) Once all features were developed the application underwent a thorough testing phase. As mentioned in Chapter 3, all manual testing was completed to discover & resolve bugs, confirm all features functioned as intended, and validate that the user interface was user-friendly. Although testing can be accomplished at any time in the development process, the internal testing phase is critical because it allows developers to identify problems before releasing the application to a larger audience.

Stage 3: Staging (Pre-production Deployment) Prior to launching the DIU_COURSE_TUBE application, the application was launched into a private "staging" environment. A staging environment is a duplicate version of the final, production environment.

- **Purpose:** The purpose of this stage was to provide a real world testing opportunity for the entire integrated application. It provided an opportunity to verify that the front-end, back-end, database and external Content Delivery Network (CDN) services are communicating properly with one another.
- **Process:** The React front-end was released as a service such as Vercel and the back-end API as a service like Heroku. The real-world environment deployment helped identify configuration issues or network problems that may have been missed while running locally.

Stage 4: The Production or Live Deployment Process – Once the application has passed testing successfully in the staging environment, then it is time to deploy the application into production and make DIU_COURSE_TUBE available to the public at large.

- **Frontend Deployment:** The React App was then hosted with a global hosting platform (Vercel or Netlify etc.) so that all of our users can load the app quickly regardless of their location.
- **Backend Deployment:** The API server was deployed using a cloud provider (such as Render). Therefore, the backend API server was continuously accessible by the frontend client to accept and process user requests.
- **Database and CDN:** Project database and media delivery will use cloud-based services (such as MongoDB Atlas and Bunny.net). These services have built-in redundancy and performance to provide the highest levels of service reliability, and at the completion of this stage, DIU_COURSE_TUBE will be fully operational and accessible to all users via the public url.

Stage 5: Maintenance Stage (Post-production): After the launch of your application, there is still much work to do. The maintenance stage is an ongoing process of maintaining the application's functionality, security, and relevance.

- **Monitoring:** In addition to bug fixes, we will continue to monitor the live application to determine when there may be a crash, slow-down in application performance, or error. We will utilize automated tools for this purpose so that we can discover potential problems with the application prior to a large amount of users experiencing these problems.
- **Bug Fixing:** Once a problem has been identified, we will develop a “patch” that will fix the problem. We will test the patch to confirm it works properly, and then deploy it to the production environment to resolve the issue.
- **Future Updates:** New features and enhancements to the DIU_COURSE_TUBE platform will be developed based on the needs of our users and the objectives of the project. As new features and enhancements are developed, each will follow their own cycle of development, testing and deployment; therefore, the DIU_COURSE_TUBE platform will continue to grow and improve over time.

Chapter 5 - User Manual

5.1 Introduction

The user guide will provide the necessary information for you to use the DIU_COURSE_TUBE application and is intended to help administrators understand and utilize the features and capabilities of the application. The purpose of this user guide is to enable administrators to effectively use the application; the application's contents and its users, ensuring an environment that is both safe and maintained properly by the administrator.

5.2 Project Functionalities

Dashboard

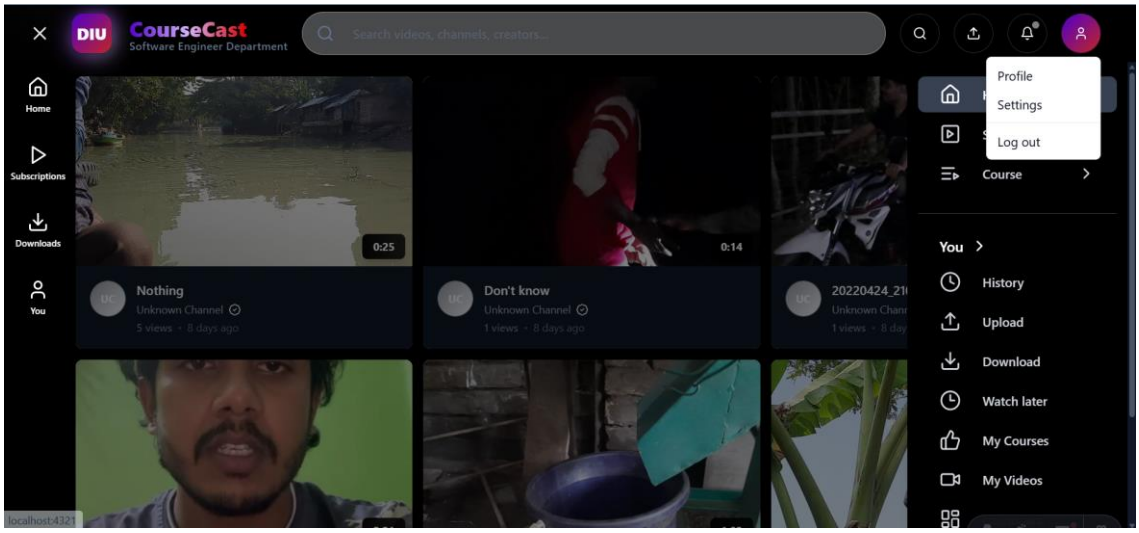


Figure: Dashboard

Teacher Registration

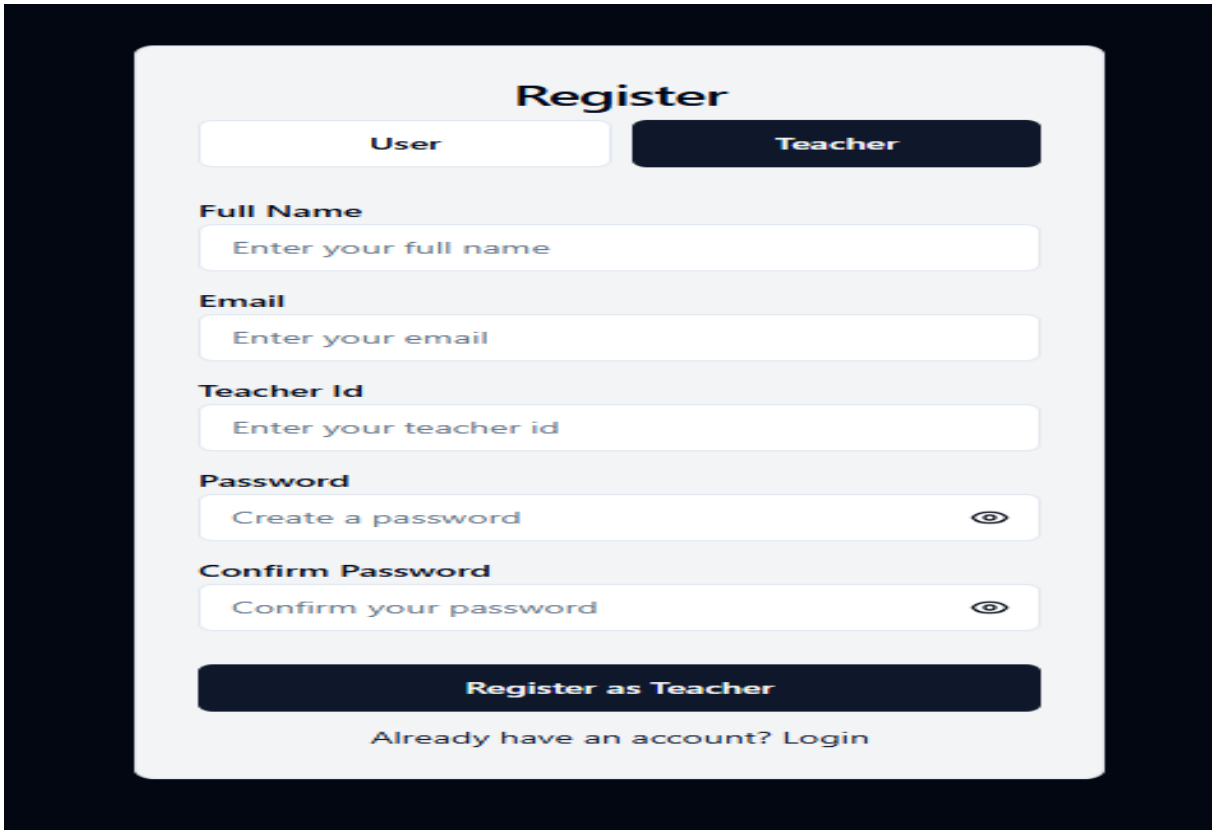


Figure: Teacher Registration

User Registration

Figure: User Registration

Teacher Login

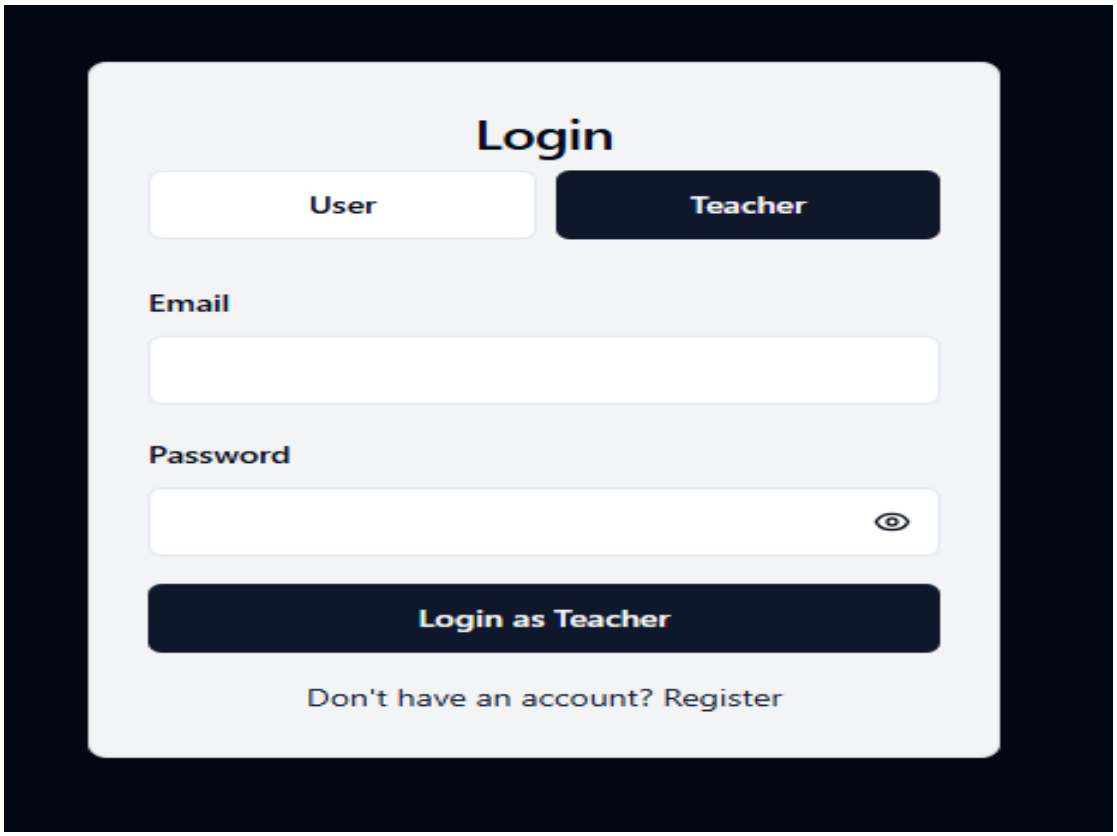


Figure: Teacher Login

User Login Page

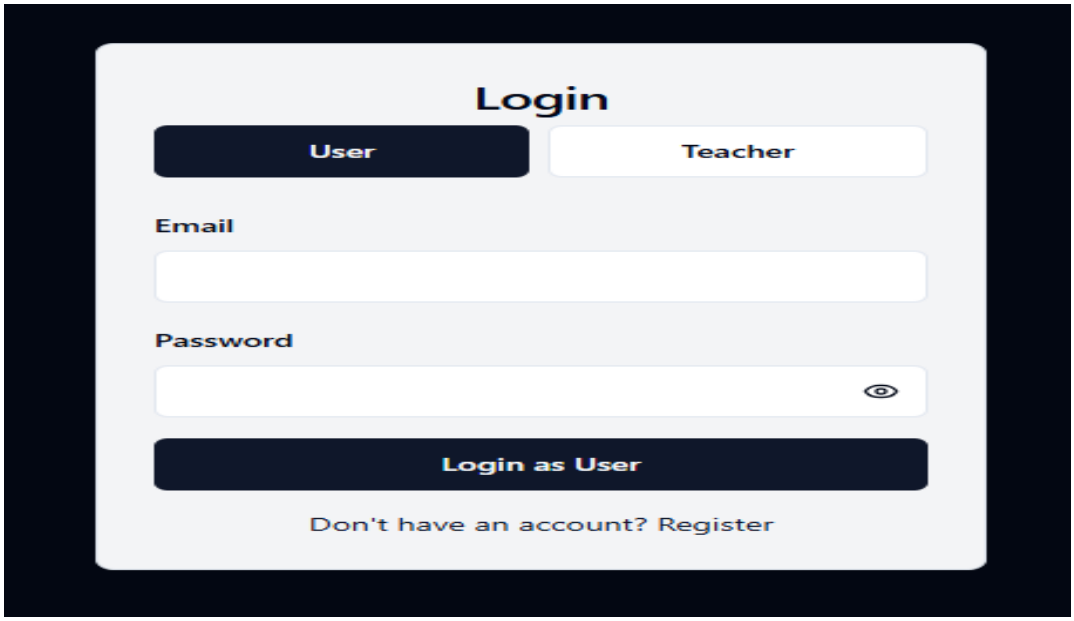


Figure : User Login

Watch Video

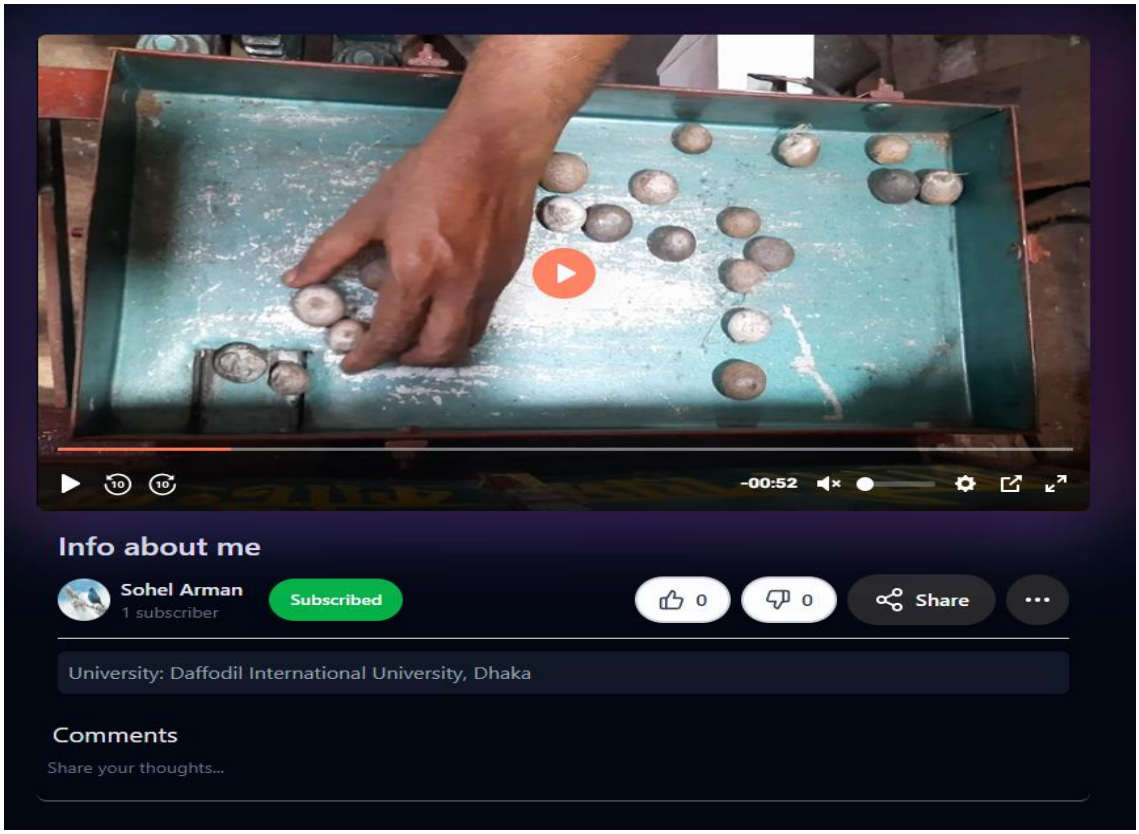


Figure : Watch Video

Share Video

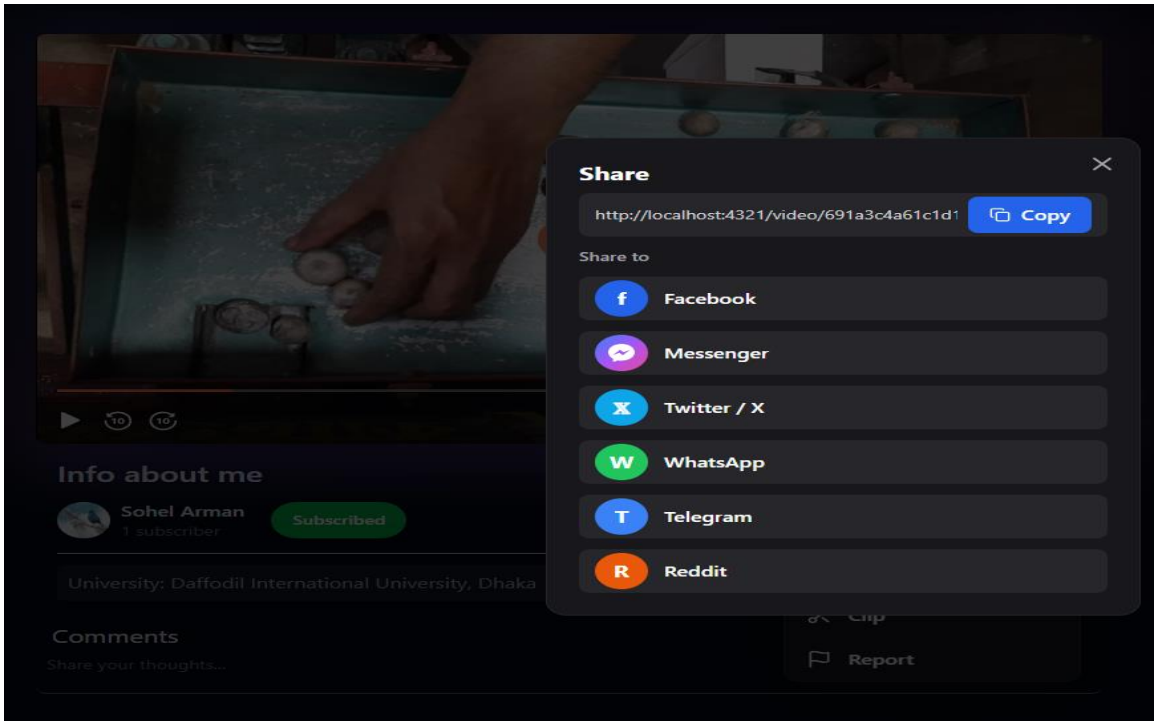


Figure: Share Video

Add Comment

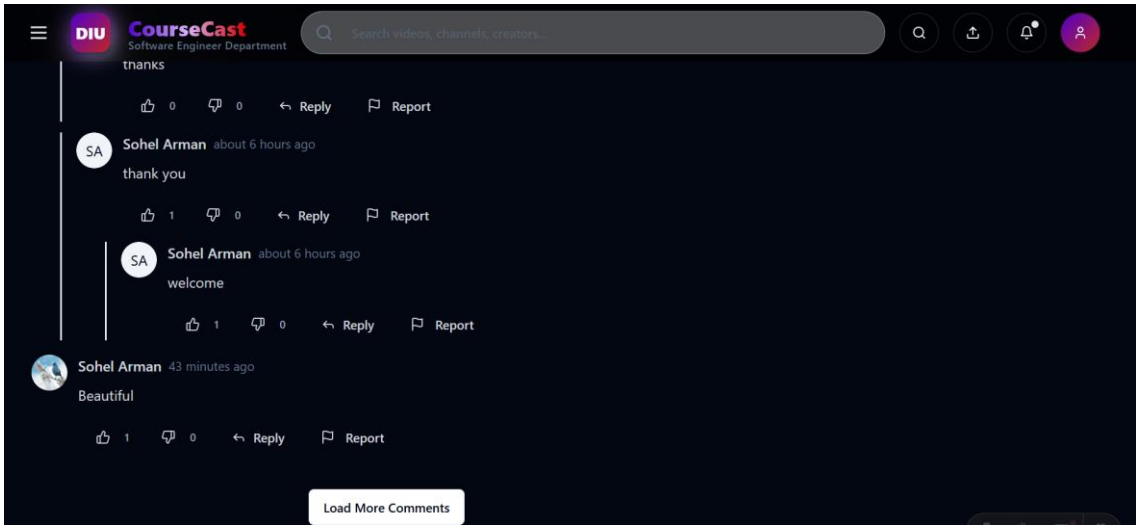


Figure: Add Comment

Add Like/Dislike

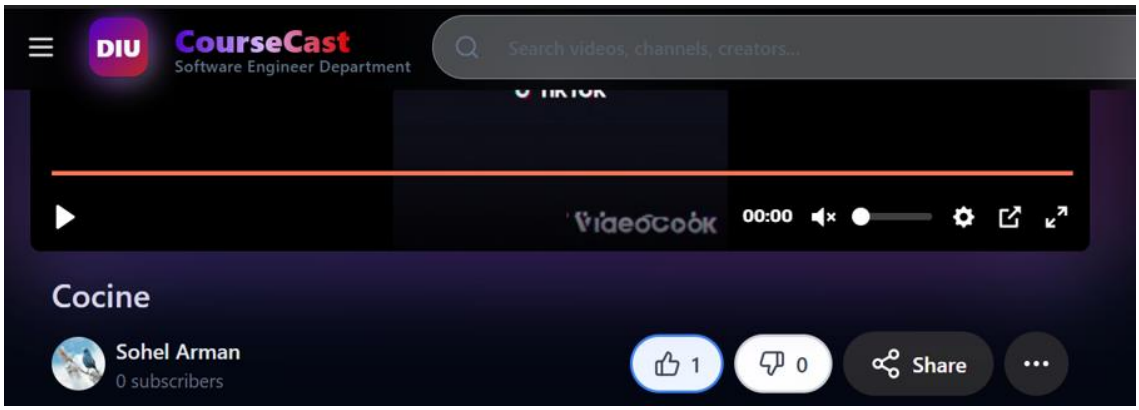


Figure: Add Like/Dislike

Subscribe

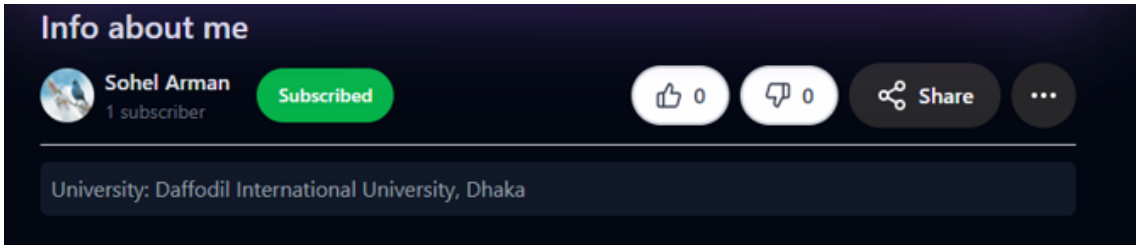


Figure: Subscribe

Create/Edit Profile

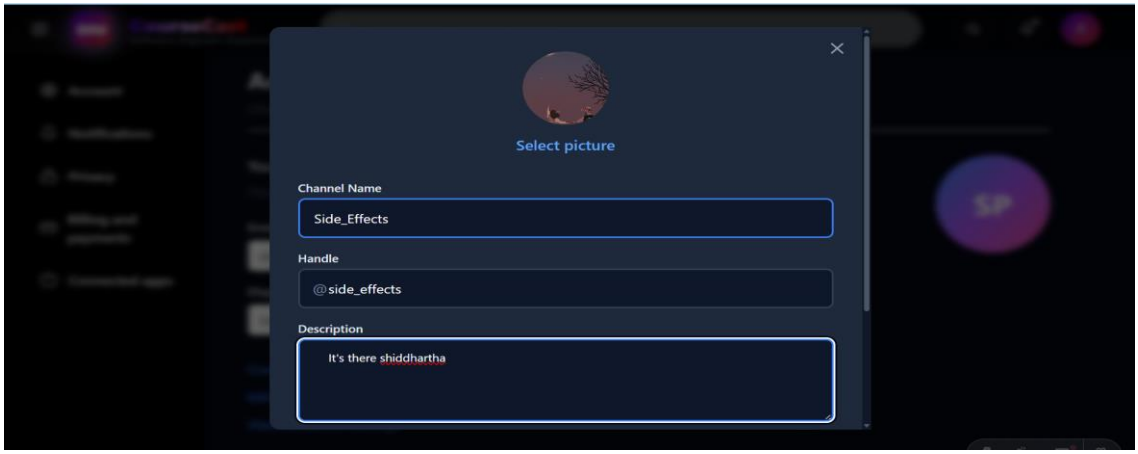


Figure: Create/Edit Profile

5.3 Summary

In this chapter here I will have an overall review of the main functionality of the DIU_COURSE_TUBE platform. It serves as a summary of the user manual, detailing the step-by-step processes for account management, content interaction, and community engagement. The system is designed with two primary roles in mind—Users (viewers) and Teachers (content creators)—each with a tailored experience to meet their specific needs. The following sections outline the essential features that constitute the platform's user experience.

1. Account Management

Account Management on an Accountable Platform - Providing Unique Experiences for Viewers & Creators.

- **User and Teacher Registration:** Separate pathways for User and Teacher registration exist in order to ensure that a user has the most suitable experience for their intended use. A user navigates to the sign-up page where the user enters their details (name, e-mail address, and a unique password) creating the users account and establishing what type of member they are within the system; thus providing them access to the appropriate functionality.
- **User and Teacher Login:** Members who have created accounts may access the site by using a secure login portal. When a member logs-in to the site, he/she will enter their e-mail address and password. If the account exists and the member is verified as having entered their correct log-in information, the system authenticates their membership and role. On successful log-in, users are directed to the main content discovery areas of the site. Teachers are usually redirected to their respective creator dashboards; therefore, providing them a quick way to begin working on their respective workflow.

- **Create/Edit Profile:** Users and Teachers alike have the capability to create and edit their individual profiles. This provides the opportunity for users to modify their name, update their profile picture/Avatar, and/or modify their account options. Teachers' profiles are also very important components of their channel's image, and help create a sense of identity which enables them to establish a connection with their audience.

2. Content Interaction and Engagement

The central theme of the DIU_COURSE_TUBE experience is centered on viewing, accessing, and collaborating with users through video content.

- **Watch Video:** The main purpose of the platform is to deliver an effective video experience. Users can select a video from their library and that video will be delivered to them using a robust content delivery system which allows for low buffering and good performance. The video player has all the common video control features including play/pause and volume control. The video player also provides access to the title, description and creator channel information related to the selected video.
- **Add Like/Dislike:** Users can provide immediate feedback on videos through a like and dislike system. By clicking the "thumbs up" or "thumbs down" icons, viewers can express their opinion. This interaction is not only a form of feedback for the creator but also helps the platform's algorithms understand user preferences to offer better content recommendations in the future.
- **Add Comment:** To add comments to your videos to create a space for community and communication with other users you are able to click on a "Comment" button under all of your videos. Users who have logged into the website will be able to make comments, ask questions, and respond to comments made by other users. The ability to comment directly with the author of the content (the person or people creating the content) allows the author and the users viewing the content to interact directly with one another. Additionally, users are also allowed to interact with each other by commenting on other users' comments and responding to comments they have received.

3. Community and Content Curation

The functions of this service allow users to track their favorite producers and share content beyond the confines of the service.

- **Subscribe:** When a user clicks “subscribe” to a teacher’s channel on the video player or channel page, they will automatically receive notifications when that

teacher uploads new content. This function is a necessary tool in helping teachers establish and sustain an engaged fan base.

- **Share Video:** A second element of the service is the ability to easily share videos. In order to provide a wider audience for its content, the platform has created a strong "share" feature. By clicking the "share" button, users are able to generate a URL to the video, copy it directly into an email, and/or post to their other social media accounts (i.e., Facebook, Twitter/X, WhatsApp), to share the video instantly.

Chapter 6 Project Summary

6.1 Introduction

DIU_COURSE_TUBE is an innovative and feature-rich video-sharing application that will allow Teachers to reach users in a global community. The purpose of DIU_COURSE_TUBE is to create a high performance, interactive video-sharing experience that mirrors all of today's major video-sharing applications. With DIU_COURSE_TUBE's strong suite of features — from secure user authentication and profiles to rich community engagement tools like commenting, reactions and subscriptions — DIU_COURSE_TUBE has created a complete ecosystem that supports both video consumption and video creation. This platform utilizes a state-of-the-art technology stack including a modern React-based front end and a scalable back-end API — along with media delivery that is optimized using a Content Delivery Network (CDN) — to provide an optimal user experience.

6.2 Project Limitation

Although the DIU_COURSE_TUBE project effectively executes all of the primary functions that define the project, the project was created in accordance with a set of constraints that ultimately limited the total scope of the project; therefore, the constraints are relevant to provide a full understanding of the project at the time it was completed.

Technological Constraints: The project is built as a client-side rendered Single Page Application (SPA) using React. While the use of this model will enhance the user experience, it may create issues for Search Engine Optimization (SEO) when compared to Server Side Rendering (SSR) or Static Site Generation (SSG) models. The system's performance and scalability are also dependent on third-party services, including BunnyCDN for image hosting and mediadelivery.net for video streaming, which come with their own operational limits and potential costs.

Time and Resource Constraints: Certain features were conceptualized but only 90% of the functionality was implemented due to the constraints associated with the project timeline.

Recommendation Engine: Currently, the "Related Videos" feature on the watch page contains dummy data. Developing an effective recommendation engine utilizing an algorithmic approach was beyond the scope of this phase.

Incomplete Engagement Features: The user interface includes buttons for "Clip" and "Report" functionalities. However, the backend logic and corresponding user workflows for these actions have not been developed.

Share Tracking: The API call to track video shares contains a placeholder for the user ID ('current-user-id') and requires further integration to be fully functional.

Unaddressed Requirements: For this version, several features typical of large-scale video platforms were intentionally deferred. This includes the absence of a comprehensive admin dashboard for site-wide content moderation, user management, and analytics. Furthermore, features like live streaming, real-time notifications, and monetization (e.g., ads or channel memberships) were not part of the project's requirements.

6.3 Scope

This DIU_COURSE_TUBE Project is focused around building a fully functional Video Sharing Platform which has all the necessary tools for both Content Creators and Viewers.

The project's scope includes:

- **Dual-Role System:** Development of Two Distinct User Roles, “Users” (Viewers), and “Teachers” (Content Creators).
- **Account Management:** Securely register and log in; create profiles for Users/Teachers; add avatars to channels; securely manage accounts of both User roles.
- **Video Playback:** High Performance Video Watch Page that streams video content from a CDN with Immersive UI with ambient lighting effects.
- **Core Engagement Features:**
    - Like and dislike videos in real time as they are being watched.
    - Comment System for all Users to comment on Videos; View comments on Videos by other Users.
    - Subscription System for Users to subscribe/un-subscribe to Channels.
    - Save For Later: Save videos to be able to view later.
- **Content History and Sharing:**
    - Track viewing history for each User automatically.
    - Easy to use sharing modal for copying video link(s); share to Major Social Media Platforms.

The following are explicitly excluded from the project's current scope:

- The algorithm: a dynamic ML-based recommendations system for videos.
- Live streaming capabilities.
- Platform monetization features.

- Responsive FrontEnd for DashBoard of Platform Management.
- Real-time user notifications.

6.4 Future Work

The present version of DIU_COURSE_TUBE has an excellent base for growth and further development in terms of several key enhancement areas and new features which would add a significant amount of added value and competitive advantage to the platform.

Some of the most important areas that are considered for enhanced functionality include:

- **Implement a Recommendation Engine:** Most importantly, for future development, it is recommended to develop a dynamic recommendation engine to replace the static “Recommended Videos” area of the platform. This can be developed using Content-Based Filtering (CBF) to recommend videos that share the same tag(s) or come from the same channel, and/or Collaborative Filtering (CF) based upon user behavior.
- **Develop an All-inclusive Administrative Dashboard:** Developing a secure administrative portal is also very important for the ongoing management of the platform. Using this administrative portal, platform administrators will be able to perform a variety of tasks including managing users (i.e. suspension of accounts), moderating content (i.e. removal of videos or comments that are deemed not in compliance with platform guidelines), and viewing various analytics regarding the performance of the platform.
- **Complete In-Progress Features:** The two features for reporting issues with video ("Report") and creating a clip of a section of a video ("Clip"), are currently being used as development tools but will need to be completed for use by end-users. The "Clip" tool allows users to create a small portion of a video that they can then share on social media platforms. The "Report" tool enables users to report issues with content to prevent inappropriate or illegal content from being hosted on the platform by having it reviewed by moderators within the community of users who have access to the platform.

- **Introduce Real-Time Notifications:** Implementing an Instant Notification System: This is something that could be accomplished utilizing WebSockets to provide instant notifications when there has been a new upload made by a channel that you subscribe to, you have received a reply to your comment(s), etc.
- **Enhance Video Processing:** Improving Video Processing: Future updates to the software could include server-based video processing that provides options for the user to select from different video playback resolutions (e.g., 480p, 720p, 1080p) and auto-generate a thumbnail image for a video if none was included by the user.

6.5 Conclusion

The DIU_COURSE_TUBE project has been successful in attaining its main goal of developing an operational video-platform that is easy to use. The project was able to develop a basic set of features: Video playback, User Authentication, Subscriptions, Likes, Comments, etc... The project shows an excellent understanding of how to build modern web applications. A React-based front end was used to create a user-friendly and responsive interface, and a Content Delivery Network (CDN) was used to efficiently distribute media content.

Some of the key lessons from the development process included the need for Optimistic UI updates in order to make the user-experience faster, and the value of separating concerns between the Client, Server, and Third Party Services. Although the project was constrained by time, it is still an excellent example of a Proof-of-Concept. The project established the foundation for future growth and demonstrated the feasibility of using the selected technology stack to create a scalable Media-Centric Web Application.

REFERENCES

Books and Official Documentation

1. **ReactJS Official Documentation.** (2024). *React*. Meta Platforms, Inc. Retrieved from https://react.dev
    - *Annotation: The primary and most authoritative source for the React library, used for building the user interface of the ANT_TUBE application. This reference covers hooks (useState, useEffect), component structure, and state management principles.*
2. **TypeScript Official Documentation.** (2024). *TypeScript Handbook*. Microsoft Corporation. Retrieved from https://www.typescriptlang.org/docs/handbook/intro.html
    - *Annotation: The definitive guide for TypeScript, the language used to ensure type safety and improve code quality throughout the project. It was essential for defining data structures like VideoMetadata, User, and FileUpload.*
3. **Redux Toolkit Official Documentation.** (2024). *Redux Toolkit*. Redux Maintainers. Retrieved from https://redux-toolkit.js.org
    - *Annotation: This documentation was referenced for implementing centralized state management. The use of useAppSelector to access user authentication state (auth.user, auth.token) is a direct application of Redux Toolkit's patterns.*
4. **Wathan, A., & Schoger, S.** (2018). *Refactoring UI*. Tailwind Labs.
    - *Annotation: A foundational resource for the design and utility-first CSS approach. The principles from this book guided the implementation of the user interface using Tailwind CSS, as seen in the component's class names (container, grid, items-center).*
5. **shadcn/ui Documentation.** (2024). *ui.shadcn.com*. Vercel Inc. Retrieved from https://ui.shadcn.com
    - *Annotation: The source for the unstyled, accessible UI components used in the project, such as Avatar, Button, and the dropdown menu structure. This library was key to rapidly building a consistent and high-quality user interface.*
6. **Bunny.net.** (2024). *Bunny CDN & Bunny Stream Documentation*. Bunny.net. Retrieved from https://bunny.net/docs
    - *Annotation: The official documentation for the Content Delivery Network services used. The getBunnyCDNUrl function, which constructs image URLs for the pull zone (my-lulu.b-cdn.net), and the video player's iframe pointing to mediadelivery.net are direct implementations based on this service's guidelines.*

Web Standards and APIs

7. **MDN Web Docs.** (2024). *Fetch API*. Mozilla Foundation. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
    - *Annotation: The standard reference for the Fetch API, which is used extensively throughout the project for asynchronous network requests to the backend, such as in fetchVideoData, handleSubscribe, and handleReaction.*

APPENDICES

Appendix A: Core API Endpoint Documentation

1. Get Video Details

- Retrieves all of the relevant data that a client needs to display the Video Watch Page, i.e., the video object itself, the Channel Object, as well as the current User State (e.g., subscribed to this Channel, liked this video, etc.).
- Endpoint: GET /api/video/get-video-details
- HTTP Method: GET
- URL Query Parameter: _id (string, required) - The video id that the client wants to retrieve the data for.
- Headers: Authorization - The ID of the logged in user, which will be used to determine if the user has a subscription to the Channel as well as other user specific attributes such as isLiked or isSubscribed.
- Success Response Code (200 OK):

```
{
 "_id": "string",
 "title": "string",
 "description": "string",
 "guid": "string",
 "libraryId": "string",
 "channelName": "string",
 "userId": "string",
 "comment": [/* ...array of comment objects */],
 "like": { "likecount": "number" },
 "dislike": { "dislikecount": "number" },
 "isSubscribed": "boolean",
 "numberofSubscribe": "number",
 "saved": [/* ...array of save objects */],
 "channel": {
```

```
    "_id": "string",
    "name": "string",
    "image": [/* ...array of FileUpload objects */]
  }
}
```

- Error Response (4xx/5xx):

  ```
  { "message": "Failed to fetch video data" }
  ```

2. Subscribe/Unsubscribe to Channel

Toggles the subscription status of the current user for a specific channel.

- Endpoint: /api/subscribe
- Method: POST
- Request Body:

  ```
  {
    "channelId": "string", // The channelId of the teacher who create channel
    "userId": "string",    // The userId of all the type user
    "role": "string"       // The role of all the type of  user
  }
  ```

- Success Response (200 OK): The server handles the subscription toggle and the client re-fetches video data to update the UI.
- Error Response (4xx/5xx):

  ```
  { "message": "Failed to subscribe" }
  ```

3. Add Reaction (Like/Dislike)

Adds, removes, or switches a user's reaction (like or dislike) to a video.

- Endpoint: /api/video/addreaction
- Method: POST
- Request Body:

```
{
  "_id": "string",   // The _id of each video
  "guid": "string",  // The guid of each video
  "id": "string",    // The id of each user reacting
  "type": "string",  // reaction type e.g "like" or "dislike"
  "role": "string"   // The role of all type user
}
```

- Success Response (200 OK):

```
{
  "data": {
    "likes": "number",
    "dislikes": "number",
    "userReaction": "string" // 'like', 'dislike', or null
  }
}
```

- Error Response (4xx/5xx):

```
{ "message": "Failed to update reaction" }
```

4. Save/Unsave Video

Adds or removes a video from the current user's "Saved" list.

- **Endpoint:** /api/video/save
- **Method:** POST (to save), DELETE (to unsave)
- **Request Body (POST):**

```
{
  "videoId": "string",
  "title": "string",
```

"guid": "string",

"thumbnail": "string",

"channel": "string",

"duration": "number",

"role": "string",

"id": "string" // User's _id

}

- Request Body (DELETE):

  {

  "videoId": "string",

  "guid": "string",

  "role": "string",

  "id": "string" // User's _id

  }

- Success Response (200 OK):

  { "message": "Video saved!" } // or "Video removed from saved"

- Error Response (4xx/5xx):

  { "error": "Failed to save video" }

5. Store Watch History

Silently records that a user has watched a specific video.

- Endpoint: /api/video/historysave
- Method: POST
- Request Body:

  {

  "videoId": "string",

  "guid": "string",

  "userId": "string",

```
  "role": "string"
}
```

- Success Response (200 OK): An empty success response is expected as this is a background task.
- Error Response (4xx/5xx): The client does not handle errors for this endpoint to avoid interrupting the user experience.

Appendix B: Core Data Models (TypeScript Interfaces)

This appendix defines the primary TypeScript interfaces used throughout the ANT_TUBE application. These data structures are fundamental to ensuring type safety and represent the core entities of the system, such as users, videos, and file uploads.

1. User Interface

Represents both a standard "User" and a content-creating "Teacher." The role property distinguishes between them.

typescript

```
interface User{
  _id: string;
  name: string;         // The display name of all user or channel who are teacher
  email: string;        //  User login email (not always exposed to the client)
  role: 'user' | 'teacher'; // Role defining user permissions
  image?: FileUpload[]; // Array of profile images, typically the last one is current
  createdAt: Date;
  updatedAt: Date;
}
```

2. VideoMetadata Interface

Represents the complete set of data for a particular video. This includes the video's metadata, creator info, and any engagement stats.

typescript

```
interface VideoMetadata {
  _id: string; // A unique ID for every video.
```

```
title: string; // The title of the video.
description: string; // The detailed description of the video.
guid: string;         // The unique identifier from the video streaming service
libraryId: string;    // The library ID from the video streaming service
thumbnail: string;    // URL to the video's thumbnail image
duration: number;     // Video duration in seconds
userId: string;       // The _id of the user who uploaded video ( Teacher)
channelName: string;  // Name of the channel at the time of upload
channel: User;        // The full user object for the video's creator
comment: Comment[];   // An array of comment objects associated with the video
like: { likecount: number };      // Object containing the total like count
dislike: { dislikecount: number };// Object containing the total dislike count
isSubscribed: boolean; // True if the current user is subscribed to the channel
numberofSubscribe: number; // Total subscriber count for the channel
saved: any[];         // Array of objects indicating which users have saved the video
createdAt: Date;
updatedAt: Date;
}
```

3. FileUpload Interface

Represents a file that was uploaded to this application; mainly used for displaying user avatars.

typescript

```
interface FileUpload {
_id: string;          // Unique identifier for the file record
userId: string;       // The _id of the user who owns the file
filename: string;     // The server-generated filename (e.g., a UUID)
originalName: string; // The original name of the file on the user's machine
fileType: string;     // The MIME type of the file (e.g., 'image/jpeg')
```

```
  fileSize: number;     // The size of the file in bytes
  path: string;         // The storage path on the server or cloud storage
  url: string;          // The direct URL to access the file
  createdAt: Date;
  updatedAt: Date;
}
```

Appendix C: Environment Configuration

In this appendix, we will describe the necessary environment variable(s) needed to allow the client side application to run properly. The environment variables should be listed in a .env file at the root of your project.

ini

```
# .env.local
# Base URL for the backend API.
# All client-side fetch requests are prefixed with this URL.

# Example for local development:

VITE_API_BASE_URL=http://localhost:5000



# The pull zone domain for the Bunny.net CDN.

# This is used to construct URLs for fetching images and other static assets.

# Example:

VITE_BUNNY_CDN_PULL_ZONE=my-lulu.b-cdn.net



# (Optional) The App ID for Facebook integrations, such as the Messenger share dialog.

# This is required for the "Share to Messenger" functionality to work.

# Example:

VITE_FACEBOOK_APP_ID=123456789012345
```

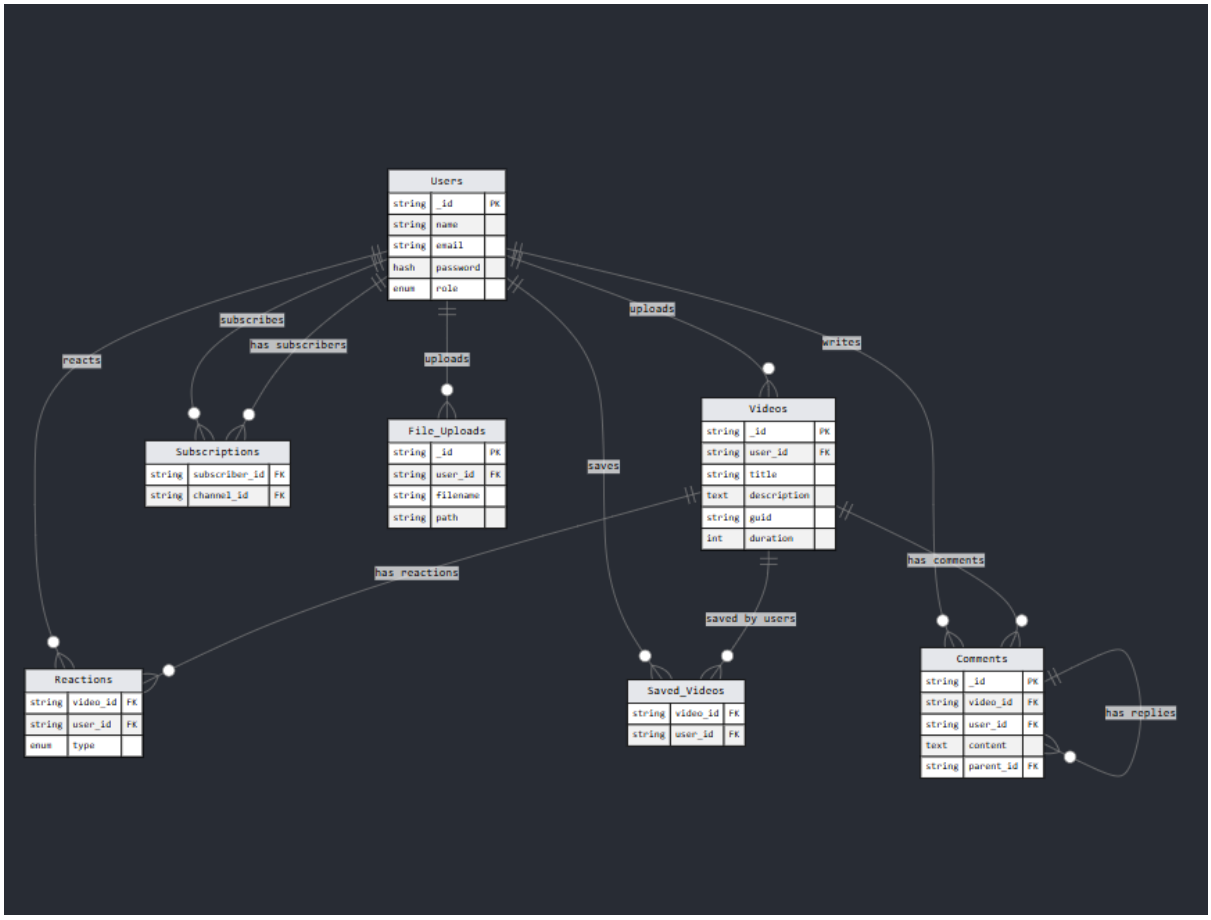
Appendix D: Inferred Database Schema Design (ERD)

This appendix presents a logical Entity-Relationship Diagram (ERD) for the ANT_TUBE database. The schema is inferred from the API endpoints and data models used within the application, such as VideoMetadata, User, and the various interaction handlers (handleSubscribe, handleReaction). This design supports all core functionalities of the platform.

Entities and Relationships:

- Users: Stores information for both 'Users' and 'Teachers'. A self-referencing relationship can model subscriptions.
- Videos: Contains all metadata related to an uploaded video. It has a one-to-many relationship with the Users table (one Teacher can have many Videos).
- Subscriptions: A join table to manage the many-to-many relationship between users (subscribers) and channels (teachers).
- Reactions: Tracks every individual like or dislike, linking a user to a video. This is more scalable than storing aggregated counts directly on the video.
- User_comments stores user comments; it has a self-referential field (parent_id) to support threaded replies.
- Saved_Videos is an association table to store which users have bookmarked which videos.
- File_uploads manages user profile images and other types of files stored on the system.

ERD Diagram:



Appendix E: Frontend Component Architecture

The current design of the WatchVideo component (index.tsx in src/components/pages/watch_video) is a single, very large component that serves as a container for the following tasks:

1. State Management

- There are over 15 state management operations via useState hooks for managing UI state, managing data, managing loading states, and managing error states.

2. Data Access

- All of the code for accessing APIs for getting video details, subscribing, reacting, saving, and sharing are contained within the WatchVideo component.

3. UI Rendering: To adhere to the Single Responsibility Principle, the WatchVideo component should be broken down into smaller, specialized components and custom hooks. 1. Custom Hooks (Logic Abstraction):

- **useVideoData(_id):** This hook would encapsulate all logic related to fetching and managing the core video data. ○ Returns: { video, loading, error, fetchVideoData }
- useSubscription(channelId): Manages the subscription state and API calls.manipulating the base video data.

- **The hook will return the following values:** { video, loading, error, fetchVideoData }
- **useSubscription(channelId)** – Holds the subscription state and is responsible for making the required API calls.
- **useReaction(videoId):** Manages the like/dislike state and optimistic updates.
    - Returns: { likeCount, dislikeCount, userReaction, handleReaction, isLoading }

2. Child Components (UI Decomposition):

The main WatchVideo component would become a container that fetches data using the custom hooks and passes props down to these smaller, presentational components:

- **<VideoPlayer video={video} />:** Renders the iframe and the ambient UI effects.
- **<VideoInfo video={video} />:** Displays the video title and description.
- **<ChannelInfo channel={video.channel} />:** Displays the channel avatar, name, and the <SubscribeButton />.
- **<ActionButtons video={video} />:** A container for the like, dislike, share, and save buttons. It would use the useReaction hook internally or receive props from the parent.
- **<ShareModal isOpen={isShareModalOpen} video={video} />:** The modal for sharing the video.
- **<CommentSection videoId={_id} />:** The existing, already separated, comments component.

Example of Refactored WatchVideo Component:

typescriptreact

```
// /src/components/pages/watch_video/index.tsx (Refactored)


import { useParams } from 'react-router-dom';

import { useVideoData } from './hooks/useVideoData';

import VideoPlayer from './components/VideoPlayer';

import VideoInfo from './components/VideoInfo';

import ChannelInfo from './components/ChannelInfo';

import ActionButtons from './components/ActionButtons';

import CommentSection from './comments';

// ... other imports


export default function WatchVideo() {
```

```
const { _id } = useParams();
const { video, loading, error, fetchVideoData } = useVideoData(_id);

if (loading) return <div>Loading...</div>;
if (error) return <div>Error: {error}</div>;
if (!video) return <div>Video not found.</div>;

return (
  <div className="container ...">
    <div className="grid ...">
      {/* Main Content */}
      <div className="lg:col-span-2 space-y-4">
        <VideoPlayer video={video} />
        <VideoInfo title={video.title} description={video.description} />
        <div className="flex items-center justify-between ...">
          <ChannelInfo video={video} />
          <ActionButtons video={video} />
        </div>
        <CommentSection
          videoId={_id!}
          comments={video.comment}
          fetchVideoData={fetchVideoData}
        />
      </div>

      {/* Sidebar */}
      <div className="space-y-4">
```

```
            {/* ... Related Videos ... */}
          </div>
        </div>
      </div>
    );
  }
```