Mini Project

Software Architecture Document

Table of Contents

Software Architecture Document

1. Introduction

This document provides a high level overview and explains the architecture of the Online learning web application.

The document defines goals of the architecture, the use cases supported by the system, architectural styles and components that have been selected. The document provides a rationale for the architecture and design decisions made from the conceptual idea to its implementation.

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

The scope of this SAD is to explain the architecture of the Distributed Development Monitoring and Mining system.
This document describes the various aspects of the OLA system design that are considered to be architecturally significant. These elements and behaviors are fundamental for guiding the construction of the OLA and for understanding this project as a whole.

1.3 Definitions, Acronyms and Abbreviations

MiniProject means this application.
OLA means Online Learning Application

1.4 References

[Project Details]:	Reports of all the teams
[Django Documentation]:	https://docs.djangoproject.com/en/3.1/

1.5 Overview

In order to fully document all the aspects of the architecture, the Software Architecture Document contains the following subsections.

Section 2: describes the use of each view

Section 3: describes the architectural goals and constraints of the system

Section 4: describes the most important use-case realizations

Section 5: describes logical view of the system including interface and operation definitions.

Section 6: describes significant persistence elements.

Section 7: describes how the system will be deployed.

2. Architectural Representation

Use Case view

Audience: all users.
Area: describes the set of scenarios and/or use cases that represent some significant, central functionality of the system. Describes the actors and use cases for the system, this view presents the needs of the user and is elaborated further at the design level to describe discrete flows and constraints in more detail. This domain vocabulary is independent of any processing model or representational syntax (i.e. XML).
Related Artifacts : Use-Case Model, Use-Case documents

3. Architectural Goals and Constraints

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

1. The system is meant for purely learning purposes. Therefore one of the primary users in this document and the system as a whole are students. As a result, one goal of this document is for future youth.

2. The system will be written using the Django framework built upon python. Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

3. The system must communicate with multiple APIs - Google,Youtube,Facebook. Defining how the system interfaces with these APIs is a primary concern of the architecture.

4. Use-Case View

The purpose of the use-case view is to give additional context surrounding the usage of the system and the interactions between its components. For the purposes of this document, each component is considered a use-case actor. Section 4.1 lists the current actors and gives a brief description of each in the overall use context of the system. In section 4.2, the most common use-cases are outlined and illustrated using UML use-case diagrams and sequence diagrams to clarify the interactions between components.

4.1 Actors

User

The user will drive all operations of the software.

The user interacts with all available interfaces to initiate and monitor all application operations.

Web Portal

The web portal is the main user interface for the system.

Drive Service

The Drive Service is the link between our application and the Google Drive API.

Youtube Service

The Youtube Service is the link between our application and the Youtube API.

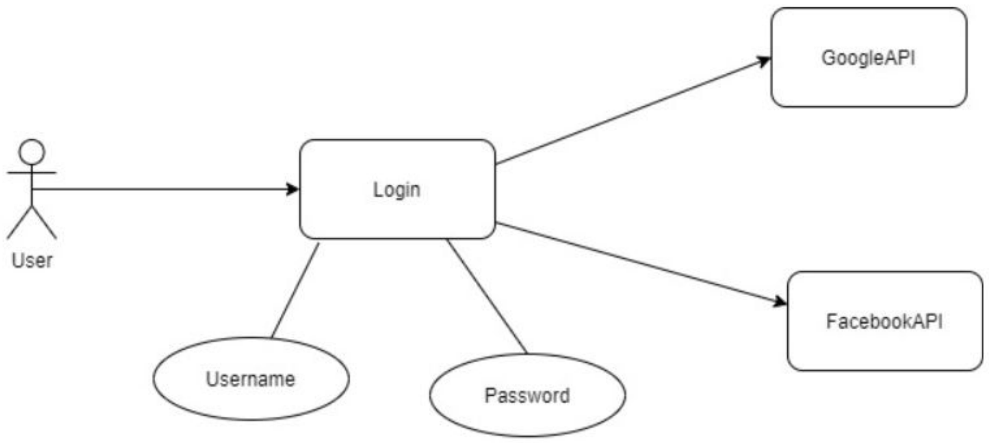
Facebook Service

The Facebook Service is the link between our application and the Facebook API.

4.2 Use-Case Realizations

Login

User credentials are authenticated and the user is redirected to the application home page.

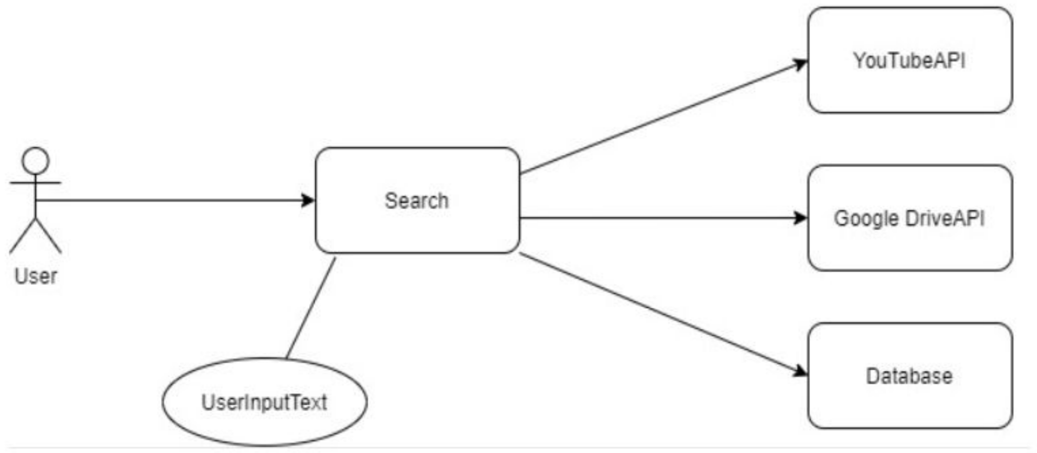


Link : https://github.com/IITDhCSE/miniproject-cs305autumn2020/tree/develop/ProjectDetails

(Refer this link for FaceBookAPI,Google Login UMLs)

Search Video

User searches a video on users desire and the list of videos is displayed from YouTube,Google Drive and Django database.

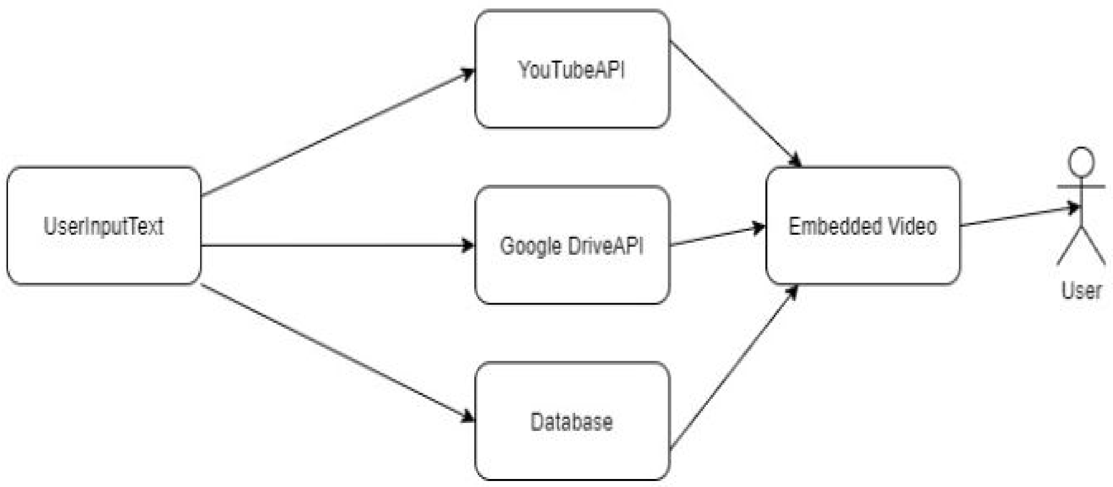


Link : https://github.com/IITDhCSE/miniproject-cs305autumn2020/tree/develop/ProjectDetails

(Refer this link for YouTubeAPI, Google Drive and RestAPI teams UML)

Play Video

After a user requests a video for a user-specified search, the video is played.



Link : https://github.com/IITDhCSE/miniproject-cs305autumn2020/tree/develop/ProjectDetails

(Refer this link for YouTubeAPI, Google Drive and RestAPI teams UML)

5. Logical View

5.1 Overview

The main goal of the logical view is to define the components that will make up the system and to define the interfaces through which they will communicate and interact with one another. The primary decision-making factor behind defining the system components is the need to isolate the components that are likely to change from the rest of the system. By clearly defining the interfaces of these components and hiding their internal implementations from the rest of the system, the impact of expected changes can be minimized.

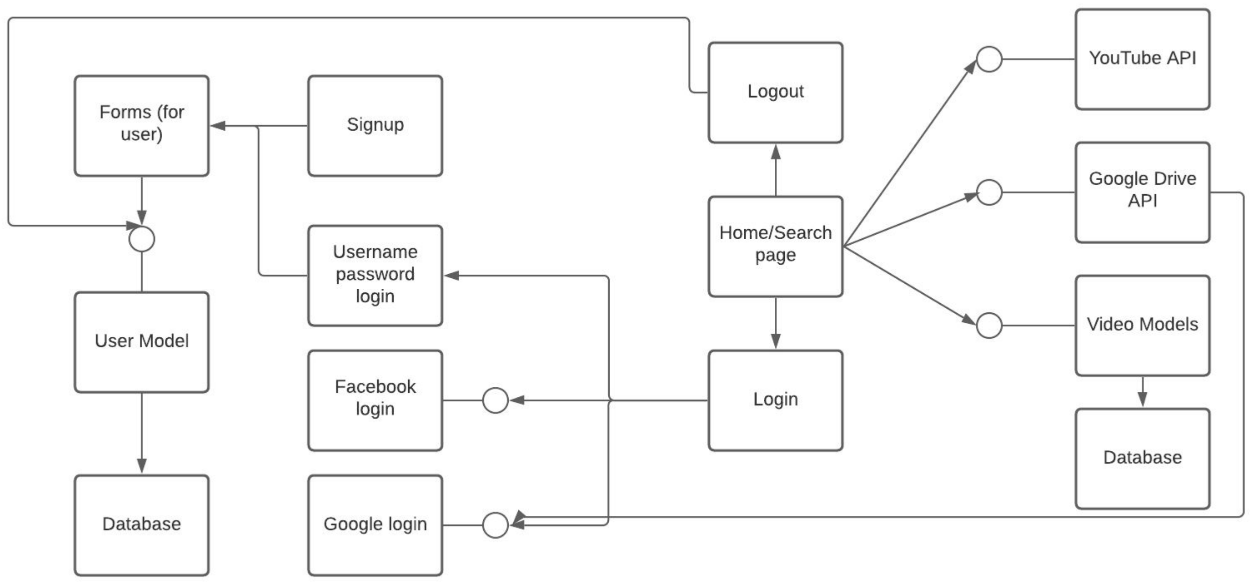


Fig. 5.1 Logical Component diagram

Element	Responsibility	Team
User Model	● Modification of Django’s user model for this project's needs.	Backend
Form (users)	● To manage all requests of signup and login ● Check for security breaches like SQL injections	Backend
Signup	● Basic Signup template (frontend) ● Integration of the template with user form (backend)	Backend + Frontend(login)
Username/Password Login	● Basic Login template (frontend) ● Integration of the template with user form (backend)	Backend + Frontend(login)
Facebook login	● Using Facebook API to logging in securely	Facebook
Google login	● Using Google API to logging in securely	Google login + Google drive
Logout	● Should logout all types of users (including those who have logged in through google and facebook)	Backend

Home/Search Page	● A search box where user can enter queries ● A efficient way to display search results ● Play the selected video	YouTube + REST + Google drive + Frontend (home)
YouTube API	● To take the query results and display some youtube videos related to that query	YouTube
Google Drive API	● To take the query results and display all the video files related to that query from google drive of the user (only if user logged in through google)	Google drive
Video Models	● To take the query results and display all the video files related to that query from the media database of the website.	REST

6. Data View

Figure 6.1 : Static Data Structure Diagram

This diagram illustrates the static data structure and relationships of the main entities that will be stored by the application in its database. Each element nominally represents a database table. Relationship cardinality is denoted with UML multiplicity notation.

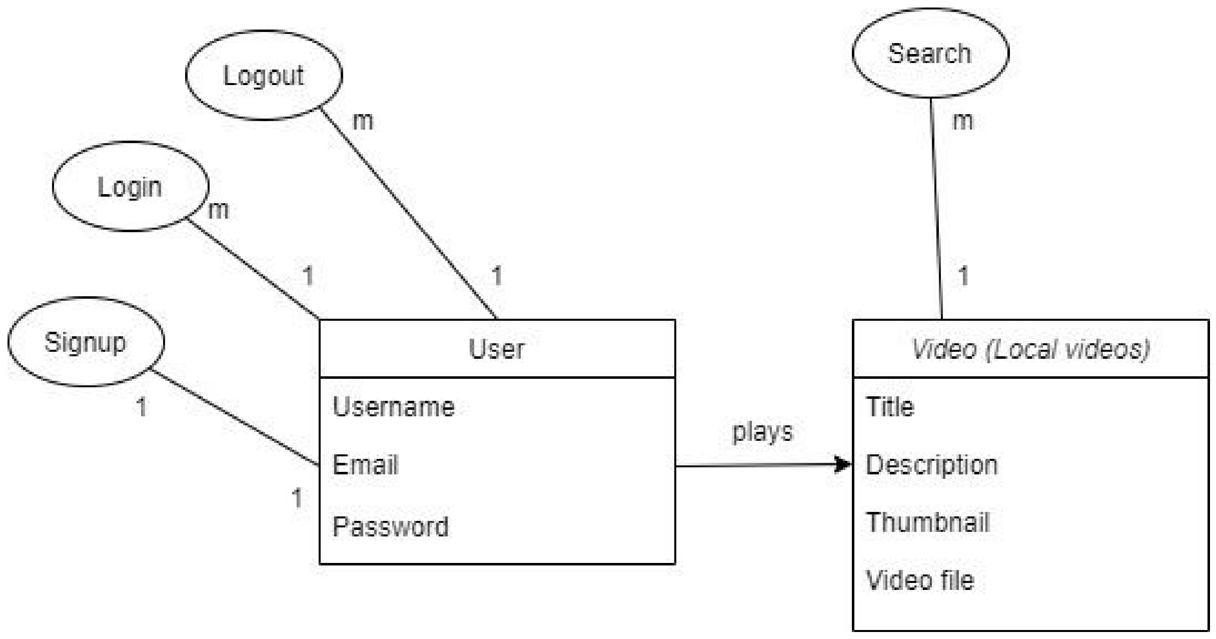


***Figure 6.1 :** Static Data Structure Diagram*

Figure 6.2 : Data Flow Diagram

This diagram illustrates how data will flow between external entities and the DMM application. Arrows show the direction of data flow, and short boxes represent persistent data stores.

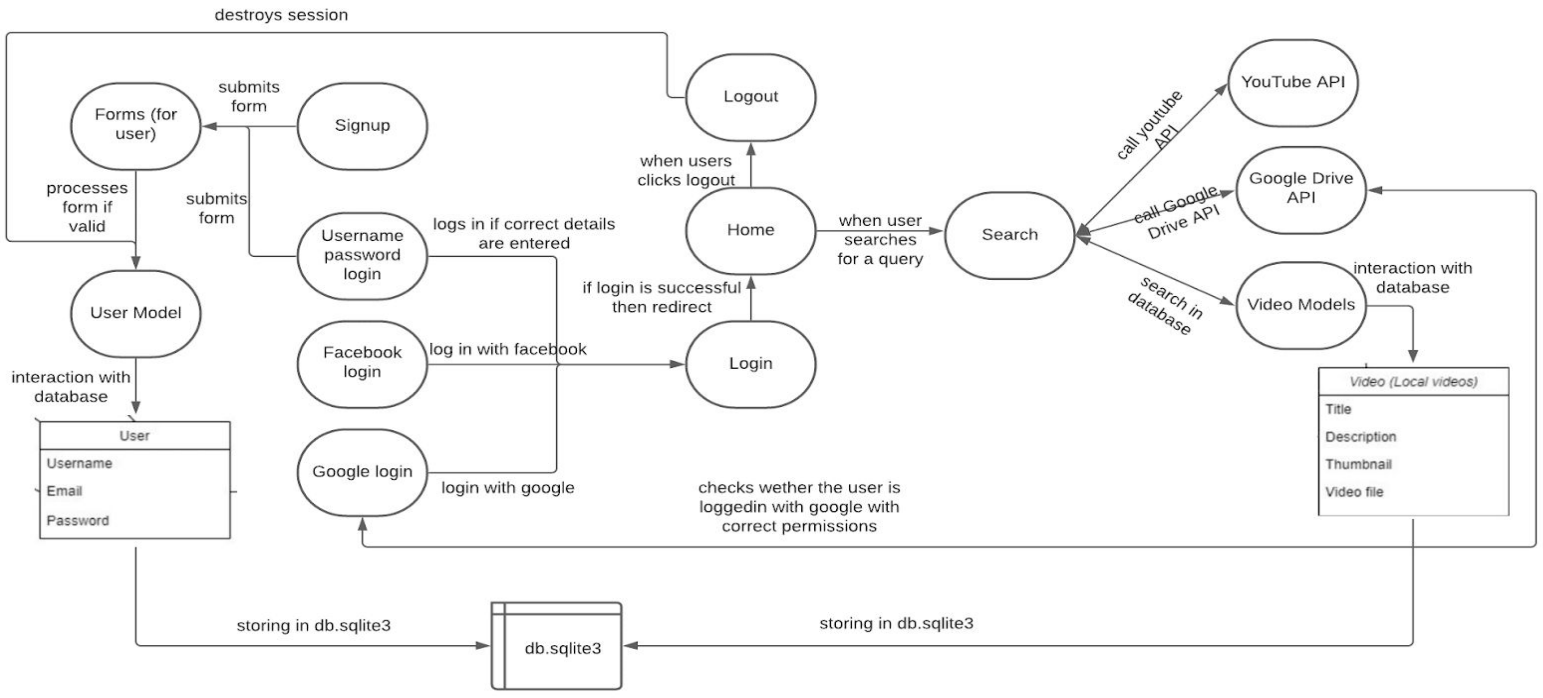


***Figure 6.2 :** Data Flow Diagram*

Contributors:
180020003 Ashish Kupsad
180020004 Biju Amruta Dathan
180010024 Paritosh Gavali
180010023 Hemanth Reddy
180010010 B Sai Yashwanth