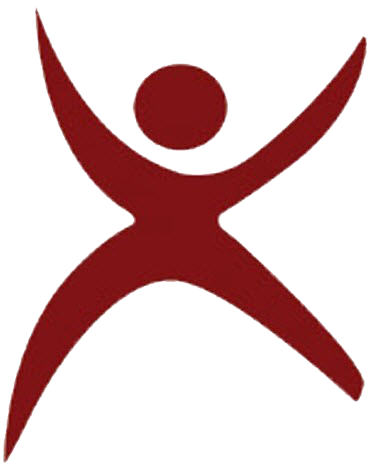
A Major Project Report   
 On

**QUESTOR – A STUDY ASSISTANT**

Submitted in partial fulfillment of the requirements for the End Semester Exam of **MAJOR PROJECT(CS3291)**   
 In   
 Fourth Year Semester-II   
 By   
 **BALASANI RAJESH(O190957)**  
 **VALAVALA RAJESWARI (O191021)**   
 **SHAIK NASEEMA(O190126)**   
 **MALAKA PRIYAMADHURI(O190787)**   
 **MELAM NARENDHRA BABU(O191106)**   
 Under the Guidance of   
 **Mr.M.KRISHNA**   
 Assistant Professor Dept of CSE (C)   
 Department of Computer Science and Engineering



**RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES – A.P.**

**ONGOLE CAMPUS**   
Kurnool Road, ONGOLE, Prakasam Dt.,Andhra Pradesh-523225

I



**BONAFIDE CERTIFICATE**

This is to certify that the project entitled “QUESTOR- A STUDY ASSISTANT”being submitted by, BALASANI RAJESH(O190957), VALAVALA RAJESWARI(O191021), SHAIKNASEEMA(O190126), MALAKAPRIYAMADHURI(O190787), MELAM NARENDRA BABU(O191106) partial fulfillment of the Requirements for the award of the degree of the Bachelor of Technologies in Computer Science and Engineering to Rajiv Gandhi University of Knowledge Technologies-AP. ONGOLE Campus, is a record of bonafide work carried out under my supervision during the academic year 2023-24.extremely thankful and pay my gratitude to **Mr.M.KRISHNA, (C) Assistant Professor, CSE**, for his valuable guidance and support on the completion of this project.The report hasn’t been submitted previously in part or in full to this or any other university or institution for the award of any degree.

**Mr.M.KRISHNA,Mr.N.MALLIKARJUNA,** Department of CSE, HOD CSE

RGUKT,ONGOLE.

II

**CERTIFICATE**

This is to certify that the project entitled “**QUESTOR – A STUDY ASSISTANT”** being submitted by **BALASANI RAJESH(O190957),VALAVALA RAJESWARI (O191021), SHAIK NASEEMA (O190126), MALAKA PRIYAMADHURI (O190787) and MELAM NARENDRA BABU(O191106)** in partial fulfillment of the requirements for the End Semester Lab Exams and for award of the degree of the Bachelor of Technology in Computer Science and Engineering to **Rajiv Gandhi University of Knowledge Technologies -A.P. ONGOLE Campus**, is a bonafide work carried out by them under my guidance and supervision from **January 2025** to **April 2025.**

The results presented in this project have been verified and found to be satisfactory. The results embodied in this project report have not been submitted to any other University for the award of any other degree or diploma.

**Faculty-In-Charge Head of the Department**

**Internal Examiner External Examiner**

III

**ACKNOWLEDGEMENT**

It is our privilege to express a profound sense of respect, gratitude and indebtedness to our guide **Mr. M. KRISHNA Assistant Professor(c),** Department of Computer Science and Engineering, Rajiv Gandhi University Of Knowledge Technologies-A.P, ONGOLE Campus,for her indefatigable inspiration, guidance, cogent discussion, constructive criticisms and encouragement throughout the dissertation work.

We express our sincere gratitude to **Mr. NANDI.MALLIKARJUNA** HOD - CSEfor his suggestions,motivations and co-operation for the successful completion of the work.

We extend our sincere thanks to our Academic Dean Mr. **MEESALA RUPAS KUMAR,** for his encouragement and constant help.

We extend our sincere thanks to **Prof.Dr.BHASKAR PATEL** Director,RGUKT-AP, ONGOLE CAMPUS for his encouragement.

**BALASANI RAJESH(O190957)**  
 **VALAVALA RAJESWARI(O191021)**  
 **SHAIK NASEEMA(O190126)**   
 **MALAKA PRIYAMADHURI(O190787)**   
 **MELAM NARENDRA BABU(O191106)**

IV

**APPROVAL SHEET**

This project entitled “QUESTOR – A STUDY ASSISTANT” submitted by Balasani Rajesh(O190957) ,Valavala Rajeswari(O191021),Shaik Naseema(O190126) ,Malaka Priyamadhuri(O190787) ,Melam Narendra babu(O191106)is approved by Mr.M.Krishna, Assistant Professor for the Degree of Bachelor of Technology in COMPUTER SCIENCE AND ENGINEERING.

**Examiner:**   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Supervisor:**   
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**   
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**   
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  
 **Date:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**   
**Place:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

V

**DECLARATION**

We hereby declare that the project work entitles **“QUESTOR – A STUDY**

**ASSISSTANT**” submitted to the **Rajiv Gandhi University Of Knowledge Technologies-**

**A.P. ONGOLE** Campus in partial fulfillment of the requirements for the End

Semester Lab Exams of Third Year Semester-I in Computer Science and Engineering is a

record of an original work done by us under the guidance of **Mr.M.KRISHNA** ASSISTENT

PROFESSOR CSE(c), **Dept. of Computer Science and Engineering** and this project work

have not been submitted to any university for the award of any other degree or diploma.

**BALASANI RAJESH(O190957 ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**VALAVALA RAJESWARI (O191021) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**SHAIK NASEEMA(O190126)**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**MALAKA PRIYAMADHURI(O190787) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**MELAM NARENDRA BABU(O191106) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Date:**

**Place:**

VI

**ABSTRACT**

QUESTOR is an intelligent, document-based study assistant designed to simplify and enhance academic preparation. Developed using Python for the backend and Streamlit for the frontend, Questor allows users to upload study materials in PDF format and interact with them through natural language queries. The system intelligently processes and retrieves relevant answers based on the uploaded content, effectively acting as a personalized AI tutor.

The assistant is powered by advanced large language models (LLMs) such as DeepSeek R1, LLaMA 8B, LLaMA 70B, Gemini, and Mixtral, allowing for accurate and context-aware responses. Questor leverages vector databases and embeddings for document understanding, enabling students to gain precise insights, summaries, or clarifications without manual searching. This tool is particularly useful for exam preparation, note revision, and fast retrieval of complex information from study material.

VII

**CONTENT**  
 **S.NO TITLE PAGE NO**  1. INTRODUCTION 1 1.1 MOTIVATION  
 1.2 PROBLEM DEFINITION  
 1.3 OBJECTIVES OF THE PROJECT  
 2. LITERATURE REVIEW 4 3. ANALYSIS 5 3.1 EXISTING SYSTEM  
 3.2 PROPOSED SYSTEM  
 3.3 SOFTWARE REQUIREMENT SPECIFICATION  
 4. DESIGN 8 4.1 UML DIAGRAMS  
 5. IMPLEMENTATION 21 5.1 MODULES  
 5.2 MODULES DESCRIPTION  
 5.3 INTRODUCTION OF TECHNOLOGIES USED  
 5.4 SAMPLE CODE  
 6. TESTING 30 6.1 UNIT TESTING  
 6.2 INTEGRATION TESTING  
 6.3 BLACKBOX TESTING  
 6.4 WHITE BOX TESTING  
 7. RESULT 34 8. CONCLUSION 38 9. FUTURE ENHANCEMENT 39 10. BIBLIOGRAPHY 41

VIII

**1.INTRODUCTION**

In the digital era, the sheer volume of academic content and learning resources can be overwhelming for students. Traditional study methods often lack efficiency when it comes to quickly finding specific information within documents. As technology advances, integrating artificial intelligence into education offers promising solutions to this challenge. \*QUESTOR\* is a study assistant built using \*Python\* and \*Streamlit\*, designed to help students interact with their study material more efficiently. It uses powerful language models and document processing techniques to provide instant, relevant answers based on the documents uploaded by the user.

**1.1 MOTIVATION**

The motivation behind developing Questor stems from the challenges students face in quickly accessing and understanding information from large sets of study materials. During exam preparation or quick revision, students often waste time searching through textbooks, PDFs, or handwritten notes for specific topics or definitions. The rise of AI models capable of understanding and generating human-like text has created new opportunities for developing smart educational tools. The goal is to leverage these advancements to assist students by providing a virtual assistant that can understand study material and answer questions naturally and accurately.

1

**1.2 PROBLEM DEFINITION**

Students often deal with extensive academic content, including textbooks, notes, and research papers, which makes the process of information retrieval time-consuming and inefficient. There is a need for a system that:

- Understands the content of uploaded study documents.

- Allows users to ask natural language questions based on these documents.

- Delivers accurate, context-aware responses instantly.

Existing tools do not offer seamless integration of personalized document-based AI interactions, especially with support for multiple powerful language models. This gap leads to missed opportunities for deeper learning and efficient revision.

**1.3. OBJECTIVE OF THE PROJECT**

The primary objective of the QUESTOR project is to develop an intelligent, document-based study assistant with the following goals:

- To allow users to upload multiple PDF documents related to their studies.

- To process and store these documents efficiently using vector embeddings and databases.

- To enable question-answering over uploaded content using advanced language models like DeepSeek R1, LLaMA 8B/70B, Gemini, and Mixtral.

- To provide a user-friendly interface using Streamlit for easy access and interaction.

- To assist students in learning, revising, and understanding topics faster and more accurately.

2

**2. LITERATURE SURVEY**

**[1] Python-Based Question Answering Systems Using NLP and Transformers: A Review by S. Mehta & R. Verma (2021):** This survey explores the use of Python and transformer-based models in building question answering systems. It highlights the implementation of models like BERT and RoBERTa using Python libraries such as HuggingFace Transformers and PyTorch. The paper emphasizes the effectiveness of semantic similarity and embedding-based retrieval for intelligent QA systems.

**[2] A Survey on AI-Powered Educational Chatbots Using Python Frameworks by K. Reddy & M. Iqbal (2022):** This paper discusses various educational chatbots developed using Python tools such as Streamlit, Flask, and NLP libraries. It examines how AI assistants can enhance learning through interactive conversations. The survey also reviews the use of language models like GPT and T5, and how these can be integrated with user-friendly interfaces using Python.

**[3] Document-Based Information Retrieval Using Vector Embeddings in Python by L. Banerjee & A. Desai (2021):** This survey focuses on document retrieval systems that use vector embeddings for semantic search. It explores the use of Python-based libraries like HuggingFace Embeddings, FAISS, and Chroma for storing and querying large document sets. The paper highlights the advantages of embedding-based search over traditional keyword matching.

3

**[4] Comparative Study of Large Language Models in QA Application by T. Roy & V. Sharma (2023):** This review compares the performance of various large language models such as LLaMA, GPT-3, DeepSeek, and Mixtral in QA and text generation tasks. It presents benchmarking results and discusses their integration with frameworks like LangChain for building end-to-end AI pipelines in Python. The study emphasizes the flexibility of switching between models for optimized responses.

**[5] Streamlit-Based Interfaces for Interactive NLP Applications: A Case Study by R. Thomas & D. Kulkarni (2022):** This paper reviews the usage of Streamlit for building interactive user interfaces for NLP and machine learning projects. It presents case studies where Streamlit is used for building chatbots, document viewers, and real-time summarization apps. The ease of integrating models and handling file uploads with Streamlit is emphasized.

4

**3. ANALYSIS**

**3.1 EXISTED SYSTEM**   
 The existing systems in the domain of study assistants typically rely on basic search algorithms to retrieve information from uploaded documents. These systems mostly involve simple keyword matching and static content retrieval, where the user inputs a query, and the system matches the query to stored information or documents without any advanced understanding of context. Current solutions may lack flexibility and often fail to offer deep contextual understanding, which is crucial for more complex learning scenarios.

While some systems use pre-trained machine learning models, they often do not leverage cutting-edge technologies like embeddings, document vectorization, and retrieval-augmented generation (RAG) models. Additionally, existing systems typically lack real-time document processing and dynamic interaction with the user, offering a less interactive experience.

**­3.2 PROPOSED SYSTEM**

The proposed QUESTOR - AI-Based Study Assistant aims to enhance the existing systems by integrating advanced AI models, such as LLaMA, Mixtral, and Groq, alongside sophisticated document processing techniques. By using HuggingFace Embeddings for document vectorization and Chroma for efficient storage and retrieval of document embeddings, the system can process and answer questions with better context and accuracy.

5

**3.3. SOFTWARE REQUIREMENT SPECIFICATION**

**Software Requirements:**

The software requirements fo this project include the following   
**1.PYTHON**:The code provided is written in python,so you will need to have python installed on your system. It is recommended to uses python 3.6version.

**2.LIBRARIES AND DEPENDENCIES:** This project depends on several Python libraries and external packages to function properly. These include:

- streamlit: Used for building the interactive frontend of the application.

- langchain: The core library used to manage document loaders, chains, retrievers, prompts, and chat history.

- langchain-community: For community-supported integrations such as PDF loaders and message history.

- langchain-core: Provides base classes and interfaces for chains and prompts.

- langchain-groq: Used to integrate and interact with LLMs like DeepSeek, LLaMA, and Mixtral via the Groq API.

- langchain-huggingface: Provides embedding generation functionality using HuggingFace models.

- chromadb: Used for storing and querying vector embeddings of document chunks.

- dotenv: Loads environment variables from a .env file, including API keys and configuration.

- PyPDFLoader: Used for reading and extracting text from PDF documents.

- time and os: Standard Python libraries used for time tracking and file management.

**Hardware Requirements:**

Personal Computers, Laptops, Tablets,Smart Phones , e.t.c.,

Processor: Dual-core or higher

RAM: 2GB or more

Storage: 100GB HDD/SDD or Higher

Network Interface: Ethernet or Wi-Fi for internet connectivity

6

**3.3.1 PURPOSE**

The purpose of this project is to develop an AI-powered Study Assistant that utilizes advanced Natural Language Processing (NLP) and machine learning models to assist students in their exam preparations. By uploading study-related documents, the system can process these documents, allowing users to ask questions related to the content, providing accurate and context-aware answers. This system aims to enhance learning efficiency by making study materials more interactive and accessible, helping students retrieve relevant information quickly during their revision.

**3.3.2 SCOPE**

The QUESTOR - AI-Based Study Assistant project is designed to enhance the study experience for students by providing an intelligent platform that helps them interact with their study materials more effectively. The scope of this project includes enabling users to upload and process PDF documents, which are then indexed for easy retrieval of relevant information. Once the documents are processed, students can ask questions based on the content of these documents, and the system will provide answers with accurate context, leveraging advanced AI models like LLaMA and Groq. The project also aims to provide a simple, interactive interface through Streamlit, making it accessible and easy to use for students. With these capabilities, QUESTOR allows students to access and understand their study materials more efficiently and in real-time, making their learning process more interactive and productive.

**3.3.3 OVERALL DESCRIPTION**

The QUESTOR project is an AI-based study assistant built using Python and Streamlit, designed to help students interact with their study materials more effectively. It allows users to upload PDF documents and ask questions related to the content. The system processes the files and uses powerful language models like LLaMA and Groq to generate accurate, context-aware answers. With its simple interface and real-time response system, Questor makes studying more efficient and engaging. It aims to support students in better understanding their notes and preparing effectively for exams and academic tasks.

7

**4.DESIGN**

**4.1 UML DIAGRAMS**

A UML Diagram is based on **UML(Unified Modeling Language)**with the purpose of

visually representing a system along with its main actors, roles, actions, artifacts or classes, in order

to better understand, alter, maintain ,or document information about the system. The UML diagrams

are divided into Structural and Behavioural UML Diagrams.

**4.1.1 STRUCTURAL UML DIAGRAMS:**

Structural diagrams depict a static view of a structure of a system.It is widely used in the

Documentation of software architecture. The Structural UML Diagrams involves 6 diagrams

They are:

 Class Diagram

 Object Diagram

 Component Diagram

 Composite Diagram

 Package Diagram

 Deployment Diagram

 Profile Diagram

**4.1.2 BEHAVIOURAL UML DIAGRAMS:**

Behavioural diagrams potray a dynamic view of a system or the behaviour of a system which

describes the functioning the system. It involves diagrams They are:

 Use Case Diagram

 State Machine Diagram

 Timing Diagram

 Communication Diagram

 Sequential Diagram

 Interaction diagram

 Activity Diagram

8

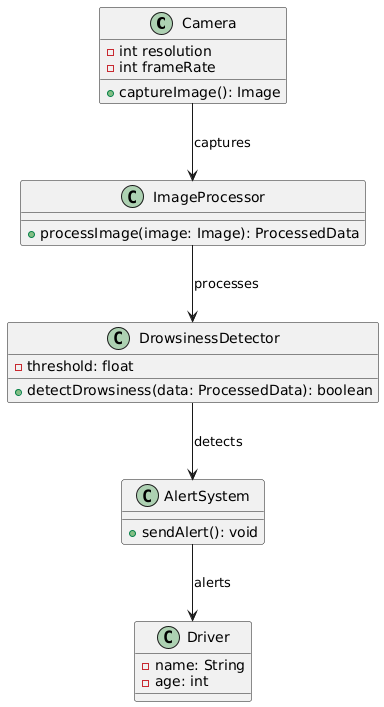
**CLASS DIAGRAM**

The class diagram is the main building block of object-oriented modeling. It is used for

general conceptual modeling of the structure of the application, and for detailed modeling

translating the models into programming code. Class diagrams can also be used for data

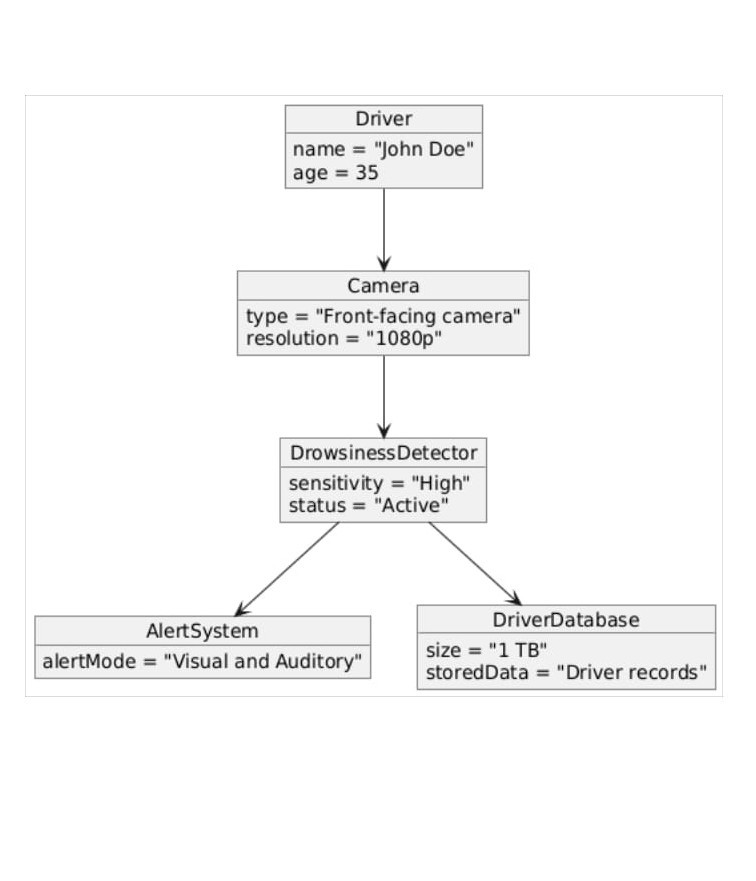
modelling.



9

**OBJECT DIAGRAM**

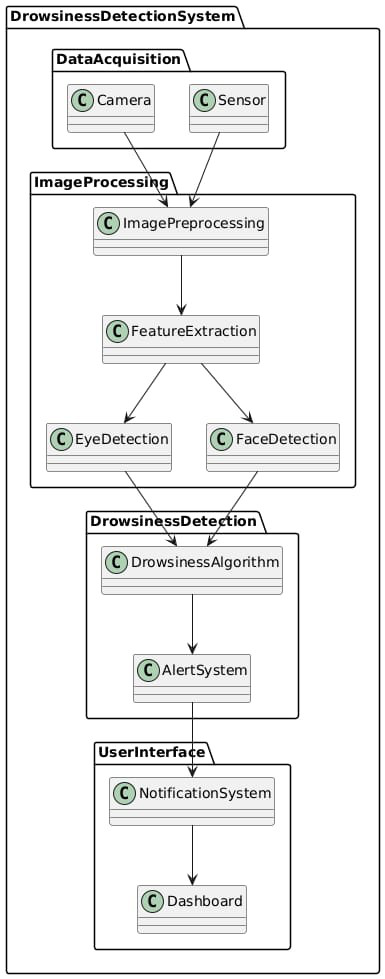
It describes the static structure of a system at a particular point in time. It can be used to test the accuracy of class diagrams. It represents distinct instances of classes and the relationship between them at a time.



10

**PACKAGE DIAGRAM**

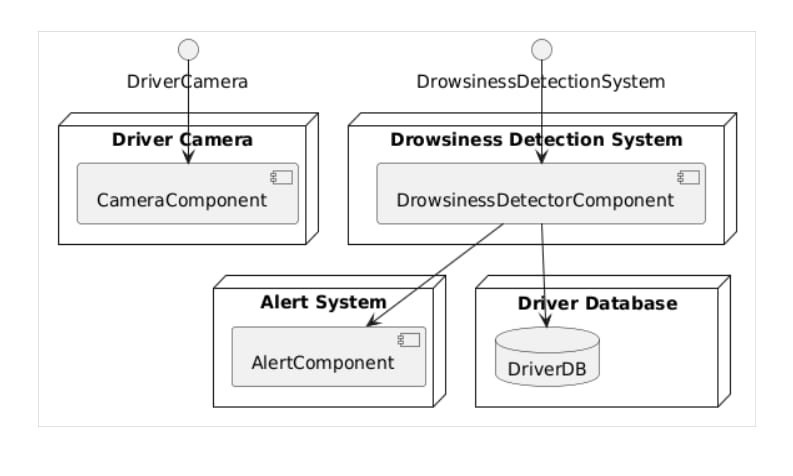
Package diagrams are used, in part, to depict import and access dependencies between packages, classes, components, and other named elements within your system. Each dependence is rendered as a connecting line with an arrow representing the type of relationship between the two or more elements.



11

**DEPLOYMENT DIAGRAM**

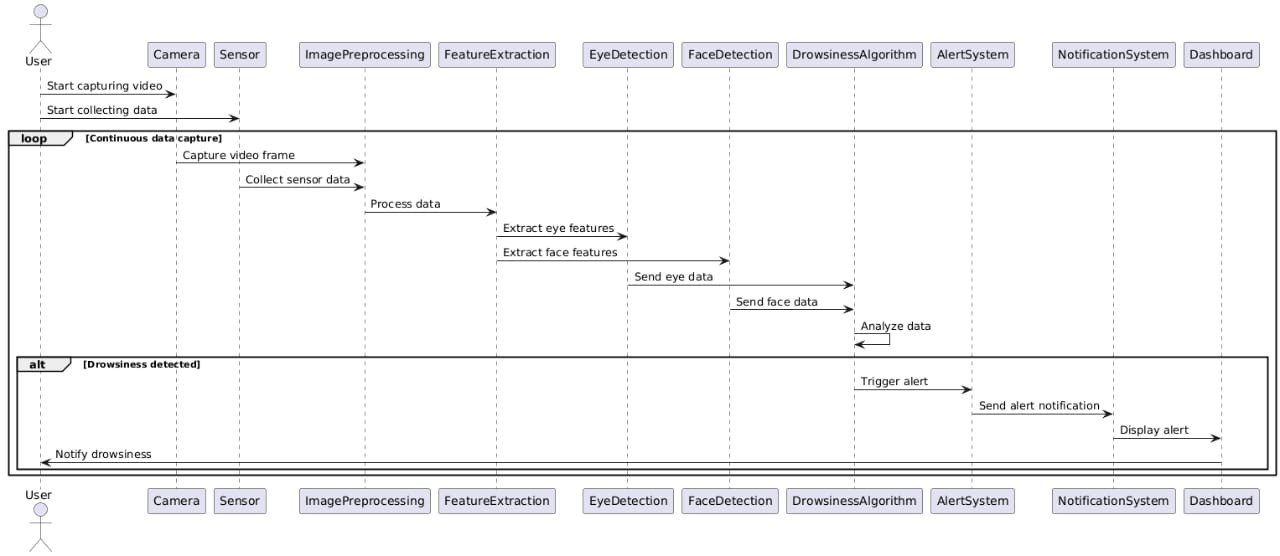
A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them. Deployment diagrams are typically used to visualize the physical hardware and software of a system.



12

**SEQUENTIAL DIAGRAM**

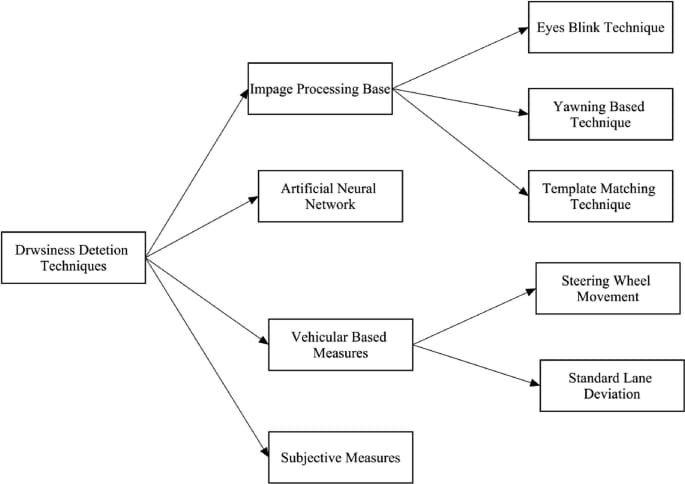
A sequence diagram consists of a group of objects that are represented by lifelines and the messages that they exchange over time during the interaction. A sequence diagram shows the sequence of messages passed between objects. Sequence diagrams can also show the control structures between objects.



13

**COMMUNICATION DIAGRAM**

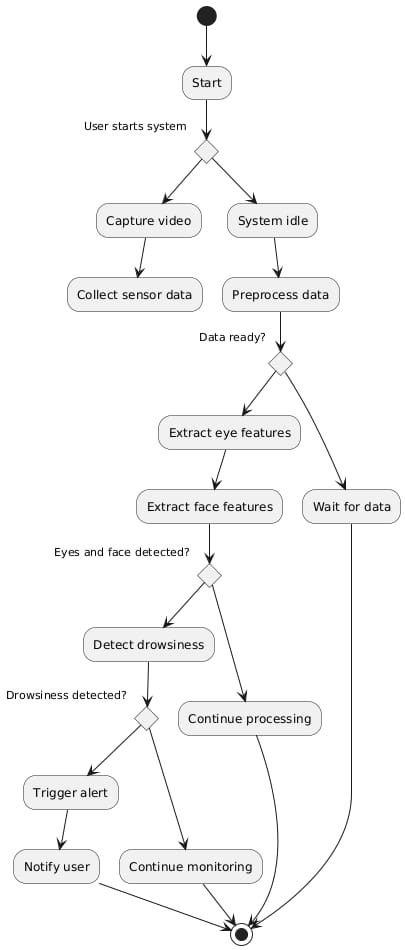
A Communication diagram models the interactions between objects or parts in terms of sequenced messages. Communication diagrams represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.



14

**ACTIVITY DIAGRAM**

An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram. Activity diagrams are often used in business process modeling. They can also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent.

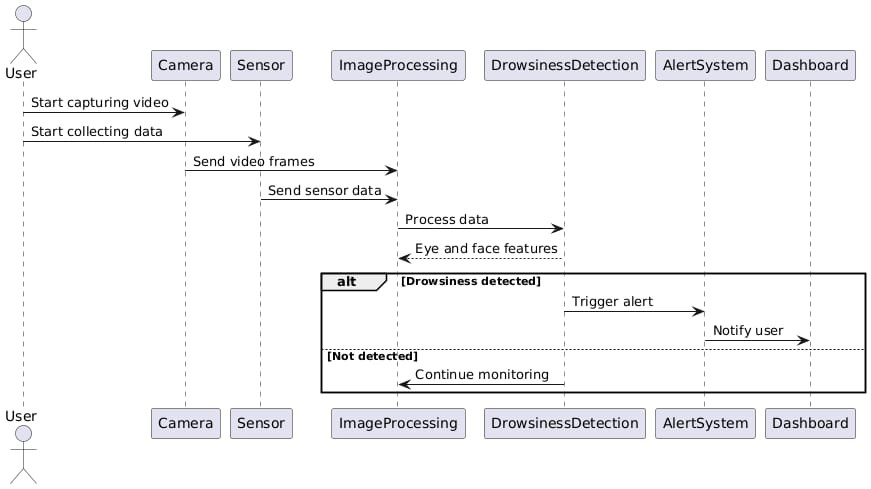


15

**INTERACTION OVERVIEW DIAGRAM**   
An interaction diagram is a type of UML diagram that's used to capture the interactive behavior of

a system. Interaction diagrams focus on describing the flow of messages within a system, providing

context for one or more lifelines within a system.

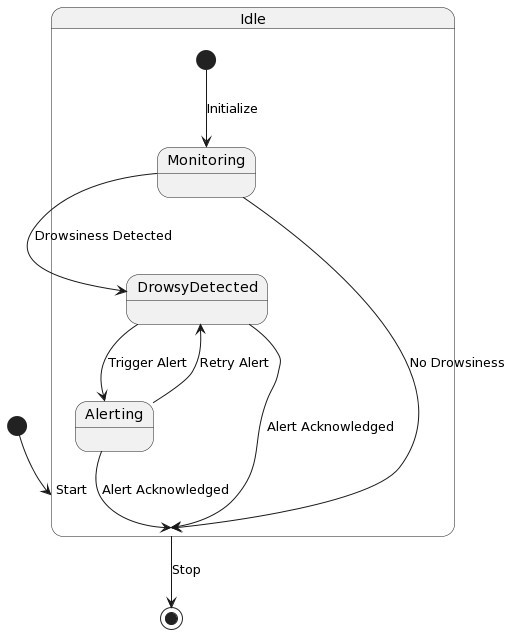


16

**STATE MACHINE DIAGRAM**   
A state diagram, also known as a state machine diagram or statechart diagram, is an

illustration ofthe states an object can attain as well as the transitions between those states in the

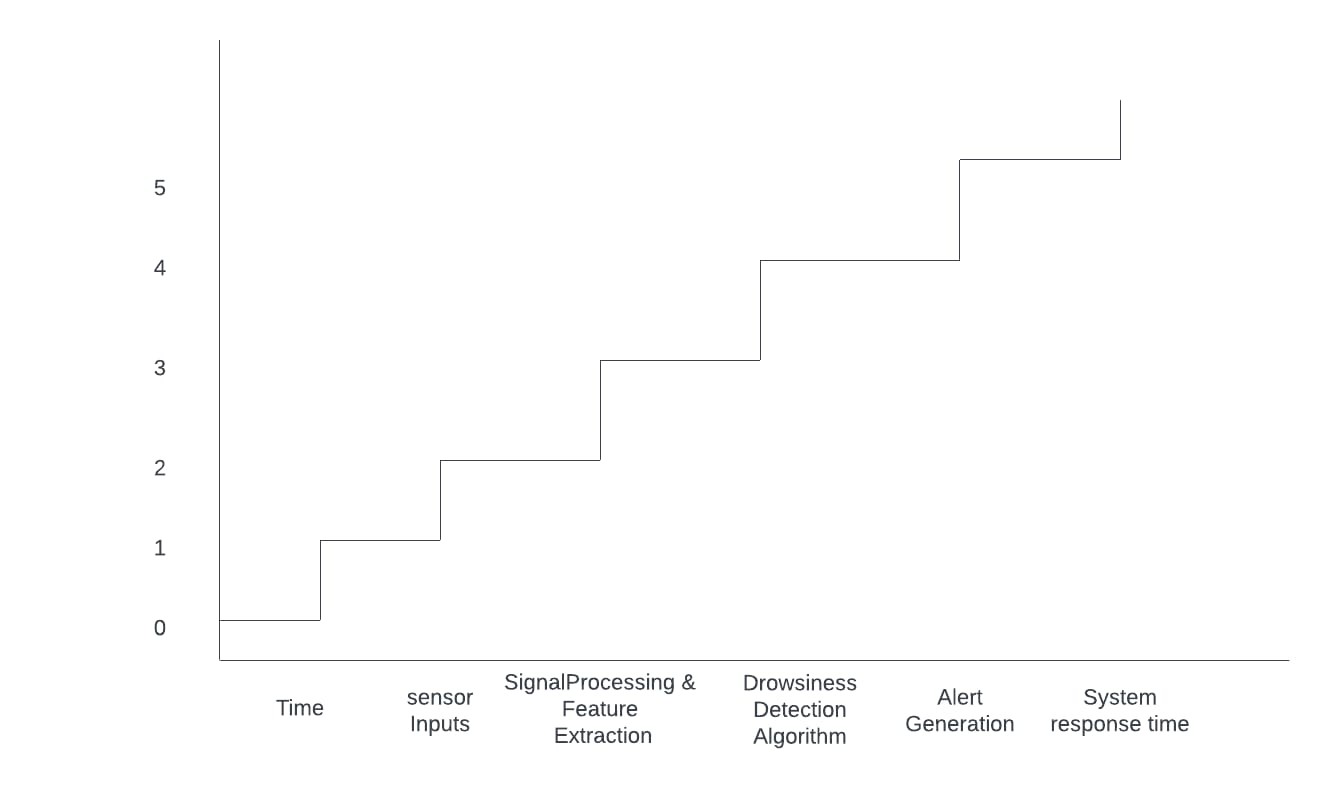
Unified Modeling Language (UML).



17

**TIMING DIAGRAM**

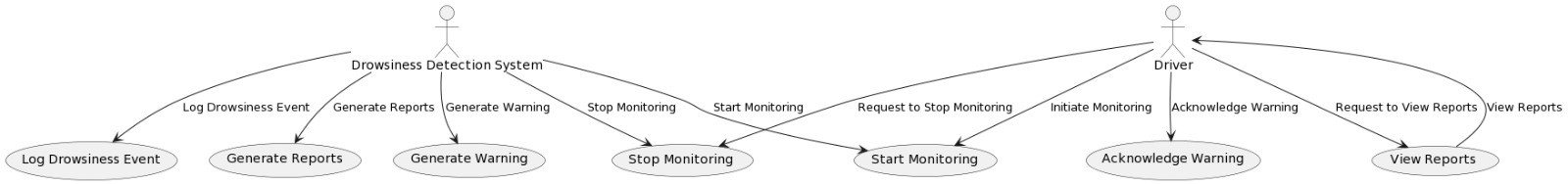
A timing diagram includes timing data for at least one horizontal lifeline, with vertical messages exchanged between states. Timing diagrams represent timing data for individual classifiers and interactions of classifiers. You can use this diagram to provide a snapshot of timing data for a particular part of a system.



18

**USE CASE DIAGRAM**

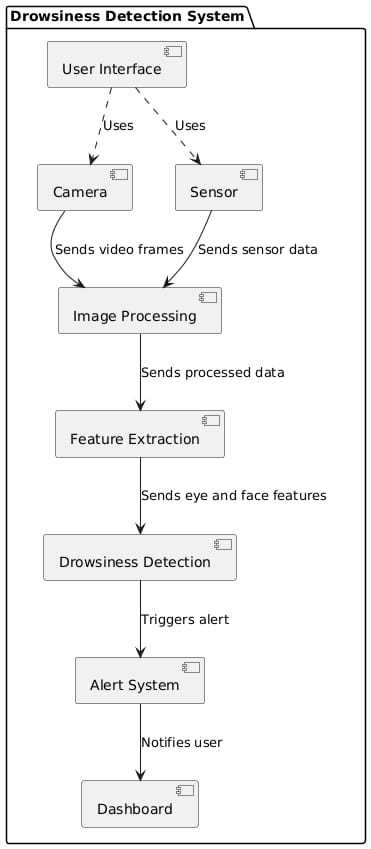
Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors. The use cases and actors in use case diagrams describe what the system does and how the actors use it, but not how the system operates internal.



19

**COMPONENT DIAGRAM**

Component diagrams are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static view of diagram.



20

**5.IMPLEMENTATION**

**5.1. MODULES:**

The QUESTOR AI-based Study Assistant project is divided into several key modules to handle the various aspects of the system. These modules include:

 Document Upload and Preprocessing Module

 Embedding and Vectorization Module

 Question-Answering (QA) Module

 Real-Time Interaction Module

 System Integration and Backend Module

**5.2. MODULE DESCRIPTION:**

**1. Document Upload and Preprocessing Module:**

This module allows the user to upload study-related documents (PDFs). Once uploaded, the system processes the documents by reading the content, extracting text, and storing the document in a format that can be easily processed by the subsequent modules. PDF files are loaded using the PyPDFLoader from the langchain\_community library, and preprocessing includes basic text cleaning, such as removing unwanted characters or formatting issues.

**2. Embedding and Vectorization Module:**

After the documents are uploaded and cleaned, this module generates vector embeddings for the document's content. The embeddings are created using models like HuggingFaceEmbeddings (such as all-MiniLM-L6-v2) which are used to convert the textual data into a mathematical representation (vector form). These embeddings allow the system to perform efficient semantic searches later. The embeddings are stored in a Chroma vector database, which helps in quick retrieval of relevant content based on user queries.

21

**3.** **Question-Answering (QA) Module:**

This module is responsible for answering the user's queries. When a user asks a question, the system uses the ChatGroq model or other LLMs like LLaMA or Mixtral to generate the response. The system retrieves relevant documents or text chunks from the Chroma vector database, then applies the chosen model to answer the question in a meaningful way. The system also ensures that the generated answer is contextually accurate by leveraging previously stored chat history.

**4. Real-Time Interaction Module:**

This module enables the real-time interaction between the user and the system. It is built using Streamlit, which provides an intuitive interface for the user to interact with the system. Users can upload new documents, ask questions, and view responses in real-time. The chat interface is designed to display both user inputs and assistant responses in a conversational format, making the experience seamless.

**5. System Integration and Backend Module:**

This module handles the integration of all the above components and manages the backend processes such as handling session states, loading models, and managing API requests. It interacts with the HuggingFace API, Groq API, and local vector databases. The backend is designed to handle multiple user requests efficiently and supports scalability.

22

**5.3 INTRODUCTION OF TECHNOLOGIES USED:**

**1.Introduction to Python programming Language:**

**Python(language):**The code provided is written in python,ensure that you have Python installed on your system. It is recommended to use Python 3.6 for compatibility with the mentioned libraries.

**2. Streamlit**

Streamlit is used to build the frontend interface of the project. It allows the creation of interactive web applications with minimal effort. Streamlit is ideal for building user-friendly chat interfaces where users can upload documents and interact with the study assistant.

**3. HuggingFace Transformers**

HuggingFace provides pre-trained models like BERT, GPT, and others, which are used for generating embeddings and answering user queries. These models are integrated using the HuggingFaceEmbeddings class, enabling efficient natural language understanding and generation.

**4. Groq API**

Groq is an AI hardware and software platform used to power large language models and inference tasks. It is integrated into the backend to optimize the performance of models, improving response times and scalability for real-time query answering.

**5. PyPDFLoader**

PyPDFLoader is used for reading and extracting content from PDF documents. This module ensures that all uploaded documents are processed and converted into text, which can then be embedded into vector form for later retrieval.

**6. NLP Models**

The project integrates multiple state-of-the-art NLP models, including DeepSeek R1, LLaMA 8B, Mixtral, and others. These models are responsible for answering questions based on the content in the uploaded documents, providing the users with contextually accurate and relevant information.

**7. HuggingFace and LLM Models**

Large Language Models (LLMs) like LLaMA and DeepSeek R1 are used for understanding user queries and generating human-like responses based on the uploaded study materials. These models enhance the system’s ability to answer questions in natural language, making it more interactive and user-friendly.

23

**5.4 SAMPLE CODE**

**App.py:**

import os

import time

import streamlit as st

from dotenv import load\_dotenv

from langchain\_groq import ChatGroq

from langchain\_community.document\_loaders import PyPDFLoader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

from langchain\_huggingface import HuggingFaceEmbeddings

from langchain\_chroma.vectorstores import Chroma

from langchain.chains.combine\_documents import create\_stuff\_documents\_chain

from langchain\_core.prompts import ChatPromptTemplate, MessagesPlaceholder

from langchain.chains import create\_retrieval\_chain

from langchain\_community.chat\_message\_histories import ChatMessageHistory

from langchain.chains import create\_history\_aware\_retriever

from langchain\_core.runnables.history import RunnableWithMessageHistory

from utils.utils import MODEL\_MAPPING, PDF\_PATH, CHROMA\_DB\_PATH

from utils.prompts import System\_Prompt, Contextualize\_q\_system\_prompt

# Load Environment Variables

load\_dotenv()

24

load\_dotenv()

# API Keys

os.environ["HF\_TOKEN"] = os.getenv("HF\_TOKEN")

groq\_api\_key = os.getenv("GROQ\_API\_KEY")

st.set\_page\_config(page\_title="Questor - Study Assistant", layout="centered")

embeddings = HuggingFaceEmbeddings(model\_name="all-MiniLM-L6-v2")

def load\_or\_create\_chroma\_db(isCreate=False):

"""Loads existing Chroma DB or creates a new one from all PDFs in PDF\_PATH folder."""

all\_docs = []

for file\_name in os.listdir(PDF\_PATH):

if file\_name.endswith(".pdf"):

loader = PyPDFLoader(os.path.join(PDF\_PATH, file\_name))

all\_docs.extend(loader.load())

if not os.path.exists(CHROMA\_DB\_PATH) or isCreate:

# st.info("Rebulinding the knowledge base...")

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1000, chunk\_overlap=100)

final\_documents = text\_splitter.split\_documents(all\_docs)

vectordb = Chroma.from\_documents(

documents=final\_documents,

embedding=embeddings,

persist\_directory=CHROMA\_DB\_PATH)

return vectordb

25

else:

return Chroma(persist\_directory=CHROMA\_DB\_PATH, embedding\_function=embeddings)

with st.sidebar

st.title("Questor - A Study Assistant")

st.write("\***\*Chatbot Settings**\*\*")

selected\_model = st.selectbox("Select Model", list(MODEL\_MAPPING.keys()))

model\_name = MODEL\_MAPPING[selected\_model]

uploaded\_files = st.file\_uploader("Upload PDF Documents", type=["pdf"], accept\_multiple\_files=True)

if uploaded\_files:

for file in uploaded\_files:

file\_path = os.path.join(PDF\_PATH, file.name)

with open(file\_path, "wb") as f:

f.write(file.read())

# st.success("Documents uploaded successfully. Rebuilding knowledge base...")

vectordb = load\_or\_create\_chroma\_db(isCreate=True)

# st.success("Knowledge base updated!")

llm = ChatGroq(groq\_api\_key=groq\_api\_key, model\_name=model\_name)

qa\_prompt = ChatPromptTemplate.from\_messages([

("system", System\_Prompt),

MessagesPlaceholder("chat\_history"),

26

# User input

if user\_prompt := st.chat\_input("Ask me anything about your studies or exams..."):

st.session\_state.messages.append({"role": "user", "content": user\_prompt})

with st.chat\_message("user", avatar="👤"):

st.markdown(user\_prompt)

try:

start = time.process\_time()

response = rag\_agent.invoke({"input": user\_prompt}, config={"configurable": {"session\_id": "chat\_session"}})

elapsed\_time = time.process\_time() - start

bot\_response = response["answer"]

st.session\_state.messages.append({

"role": "assistant",

"content": bot\_response,

"caption": f"🟡 \***Response generated by {selected\_model}**\* – ⏳ {elapsed\_time:.2f} sec"

})

with st.chat\_message("assistant", avatar="🤖"):

st.markdown(bot\_response, unsafe\_allow\_html=True)

st.caption(f"🟡 \***Response generated by {selected\_model}**\* – ⏳ {elapsed\_time:.2f} sec")

except Exception as e:

 st.error(f"⚠️ Error generating response: {str(e)}"  
  
 27

**6.TESTING**

**6.1 Unit testing:**

Unit testing, a testing techniques using which individual modules are tested to determine If there are issues by the developer himself.it is concerned with functional correctness of there standalone modules. The main aims to isolate each unit of the system to identify, analyze and fix the defects.

Unit Testing Techniques:   
Black Box Testing:Using which the user interface,input and output are tested.

White Box Testing:Used to test each one of those function behavior is tested.

User Interface Testing: User interface testing, a testing technique used to identify the presence of defects is a product/software under test by Graphical User Interface [GUI]   
**6.2 Integration Testing:**

Integration Testing done upon completion of unit testing,The units or modules are to be integrated which gives raise too integration testing. The purpose of integration testing is to verify the functional ,performance, and reliability between the modules that are integrated.

**6.3 BlackBox Testing:**

**Over View:** Black-box testing focuses on testing the functionality of the system without any knowledge of the internal code structure, implementation details, or internal paths. Testers evaluate the system by providing inputs and examining the outputs. This type of testing is often referred to as behavioral or functional testing.

**Application in Qusetor A study assistant**   
**1.Functional Testing**:

•Verify that the system accurately detects drowsiness and triggers alerts when the driver’s eyes are closed for a specified period.

•Test the system’s response to various facial expressions and head movements to ensure it distinguishes between drowsy and non-drowsy states.

28

2. **Usability Testing**:

•Ensure the user interface displays alerts and notifications clearly and understandably.

•Assess the system’s ease of use and how well it integrates with the vehicle’s existing dashboard and controls.

3. **Performance Testing**:

•Measure the system’s response time from detecting drowsiness to triggering an alert.

•Evaluate the system’s performance under different lighting conditions, camera angles, and driver positions.

4. **Error Handling**:

•Test how the system handles situations where the camera is obstructed or when there are multiple faces in the frame.

•Verify that the system recovers gracefully from sensor failures or interruptions in video feed.

**Example test cases for blackbox testing:**

•**Test Case 1**: Verify that the system triggers an alarm when the driver’s eyes are closed for more than 2 seconds.

•**Test Case 2**: Ensure that the system does not trigger a false alarm when the driver blinks or looks away momentarily.

•**Test Case 3**: Test the system’s ability to detect drowsiness in different lighting conditions (daylight, night, artificial light).

**6.4 WhiteBox Testing:**

**Overview:**White-box testing involves testing the internal structures, code, and logic of the system. Testers have full visibility into the codebase and can design tests that examine specific code paths, conditions, loops, and data structures. This type of testing is also known as structural, glass-box, or clear-box testing.

29

**Application in Driver Drowsiness Detection System**

1. **Unit Testing**:

•Test individual functions such as eye\_aspect\_ratio to ensure they return the correct calculations.

• Verify the accuracy of face and eye detection algorithms by examining the output of dlib and OpenCV functions.   
2. **Code Coverage**:

•Ensure that all branches and conditions in the code are tested to achieve high code coverage.

• Use tools like coverage.py to measure which parts of the code are executed during testing. 3. **Path Testing**:

•Examine all possible execution paths within the system to identify and test edge cases and potential logical errors.

•Verify the system’s behavior under various sequences of events, such as continuous eye closure, rapid blinking, and head movement.

4. **Integration Testing**:

•Test the interactions between different modules, such as video capture, face detection, and alarm triggering.

•Ensure that data flows correctly between components and that integrated modules work together as expected.

**Example test cases for blackbox testing:**

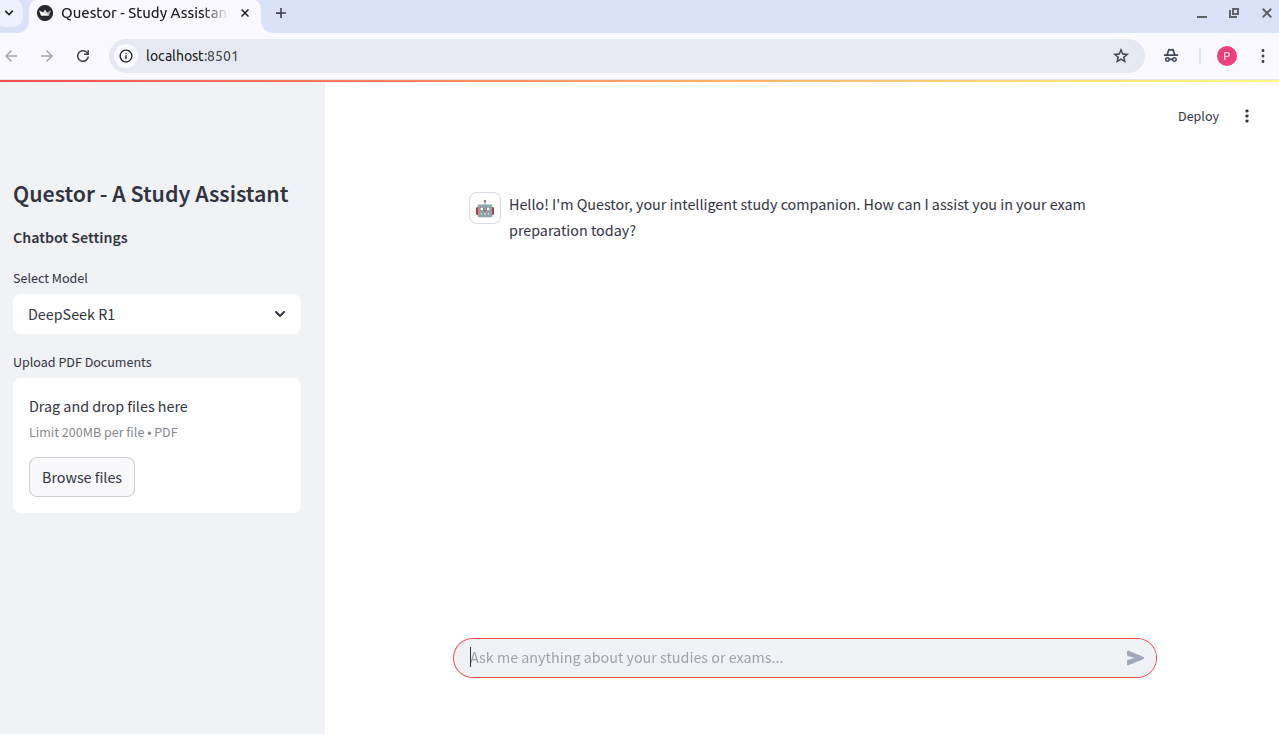
•**Test Case 1**: Verify that the eye\_aspect\_ratio function correctly calculates the aspect ratio for various eye positions.

•**Test Case 2**: Ensure that all branches in the drowsiness detection algorithm are tested, including normal eye movement, eye closure, and rapid blinking.

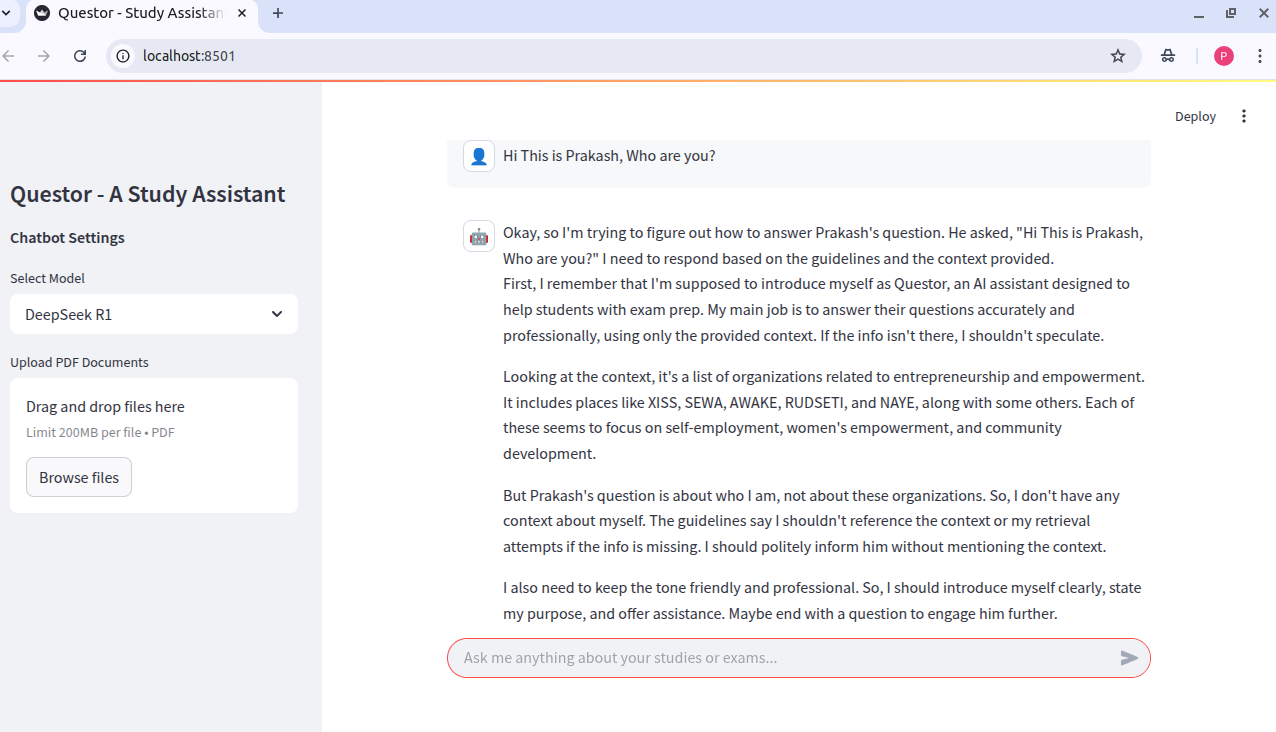
•**Test Case 3**: Test the interaction between the video capture module and the facial landmark detection to ensure data is processed accurately and efficiently.

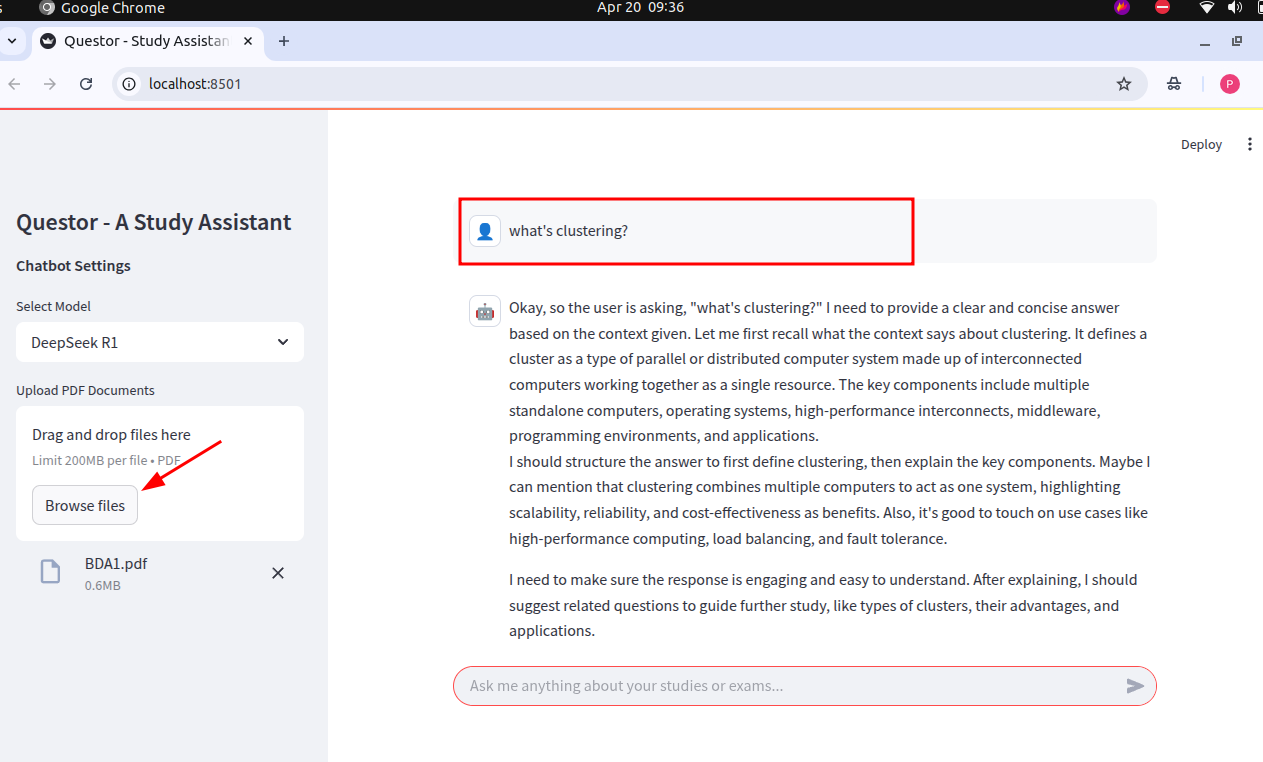
30

**7.RESULT**



31

 32

33

**8.CONCLUSION**

The QUESTOR - AI-Based Study Assistant project has successfully demonstrated the potential of integrating advanced machine learning models, document processing, and real-time question-answering functionalities into an intuitive study assistant. By leveraging cutting-edge models such as LLaMA, Mixtral, and Groq, along with the use of HuggingFace Embeddings for document vectorization and Chroma for efficient vector storage, Questor has created a powerful tool for students and learners.

The project integrates several components seamlessly, including real-time document uploads, contextual querying, and dynamic conversation generation based on stored knowledge. With the ability to answer questions based on uploaded study materials, Questor mimics a personalized tutor, assisting users in their study journey by providing accurate, contextually relevant answers to queries.

In addition to this core functionality, the system's adaptability with multiple pre-trained models ensures flexibility in the type of tasks it can handle. Whether a user wants to clarify concepts or retrieve specific details from documents, Questor provides an interactive platform to access information quickly and efficiently.

34

**9.FUTURE ENHANCEMENT**

**Future Enhancements**

For future enhancements, several avenues can be explored to further improve the *Questor – Study Assistant* and elevate the learning experience:

1. **Voice Interaction and Speech Recognition**

**Voice-Based Queries:** • Integrate speech-to-text capabilities to allow users to ask study questions using voice input.  
 • Enhance accessibility for visually impaired users or those with different learning preferences.

**Text-to-Speech (TTS):**

• Implement TTS to allow the assistant to read answers aloud, improving user engagement and auditory learning.

2. **Multilingual Support**

**Language Expansion:** • Add support for multiple languages, enabling users from different regions to interact with the assistant in their native language.  
 • Translate uploaded study materials and responses dynamically using translation models.

**Localized Content:** • Incorporate region-specific curriculum support, especially for users studying under various educational boards or systems.

35

### 3. **Integration with Learning Platforms and Tools**

**LMS Integration:** • Connect with popular Learning Management Systems (e.g., Moodle, Google Classroom) to fetch assignments and course material directly.

**Note-Taking and Highlighting:** • Enable users to take notes or highlight important parts of the conversation for later reference.  
 • Allow saving chat history to cloud platforms like Google Drive or Notion.

4. **Adaptive Learning and Personalization**

**Personalized Learning Paths:** • Use AI to recommend topics and questions based on a user's past queries and knowledge gaps.  
 • Generate personalized quizzes or summaries tailored to the user's performance and progress.

**User Profiles and History:** • Maintain detailed user profiles to provide more personalized interactions and track learning over time.

### 5. **Real-Time Collaboration and Group Study**

**Collaborative Chat Mode:** • Allow multiple users (students) to join a shared session and discuss uploaded documents with the assistant.  
 • Enable group annotations or shared question lists for collaborative study.

36

**10.BIBLIOGRAPHY**

For successfully completing our project. We have taken help from the following website’s links:

**Websites:**

The development of *Questor – A Study Assistant* was supported by the following platforms and tools:

1. [**LangChain**](https://www.langchain.com/) - Core framework for building the RAG (Retrieval-Augmented Generation) pipeline.
2. [**Streamlit**](https://streamlit.io/) - Used to build the interactive web-based chat interface.
3. [**Hugging Face**](https://huggingface.co/) - For sentence embeddings (all-MiniLM-L6-v2 model via HuggingFaceEmbeddings).
4. [**Chroma**](https://www.trychroma.com/) - Vector database used for storing and retrieving document embeddings.
5. [**Groq**](https://groq.com/) - LLM provider used through ChatGroq for fast and efficient response generation.
6. [**Python Software Foundation**](https://www.python.org/) - The core programming language used in development.
7. **PyPDFLoader (LangChain Community)** - For extracting text from PDF documents.
8. [**dotenv**](https://pypi.org/project/python-dotenv/) - For managing API keys and environment variables securely.

37