YouTube Video Summarizer Project Documentation

Overview

The YouTube Summarizer project is a comprehensive system that allows users to extract the transcript from YouTube videos, summarize the content using Google's Gemini AI, and store the summarized information along with video details in a MongoDB database. The project includes both web and mobile applications, built using React, React Native, and Flask for the backend.

Key Features:

YouTube Video Summarization: Users can input a YouTube video URL, and the system extracts the transcript and summarizes the content into key points using Google's Gemini AI.

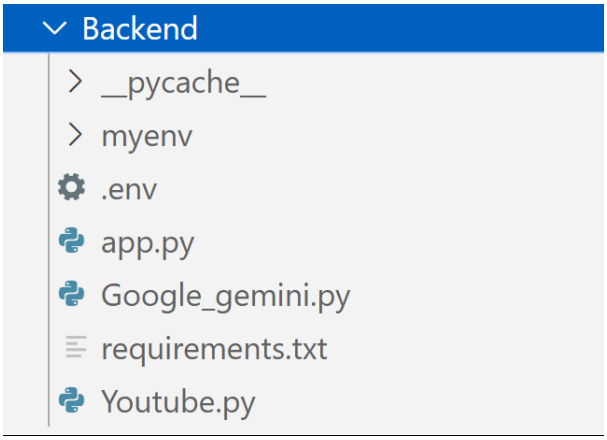
User Authentication: Users can register and log in to access the YouTube summarization feature.

Chat History: Logged-in users can view their chat history, including summarized YouTube videos they've accessed.

Web and Mobile Applications: The system offers both web and mobile applications for user convenience.

Project Structure

Backend



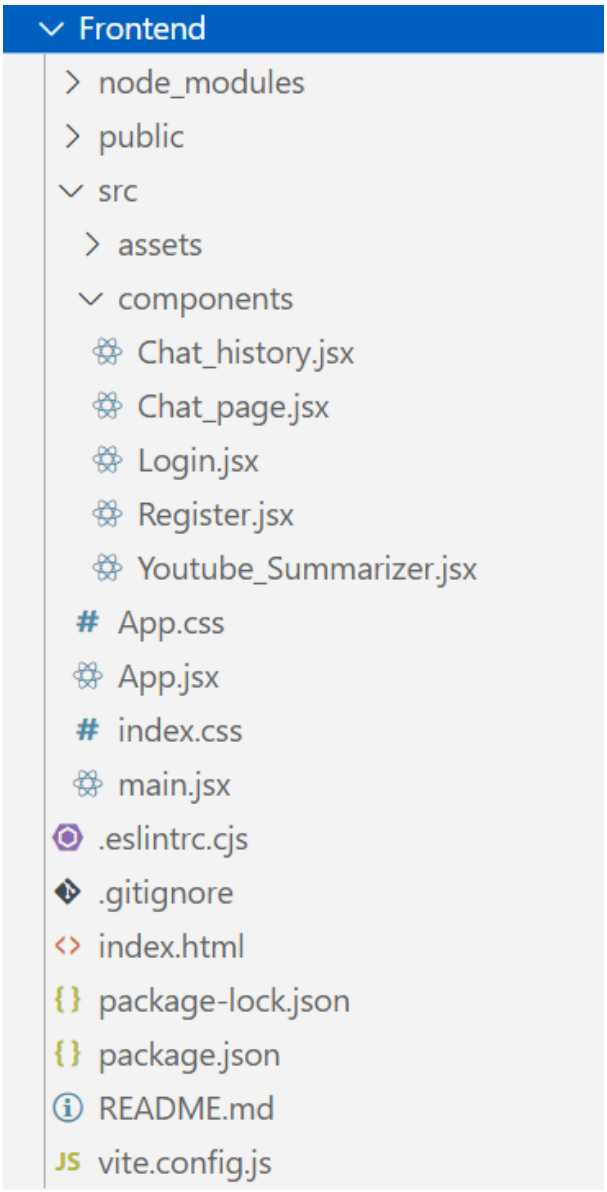
Flask: Backend server framework handling HTTP requests and responses.

MongoDB: NoSQL database for storing user data and chat history.

Google Gemini API: Utilized for generating summaries of YouTube video transcripts.

Frontend

Web Application



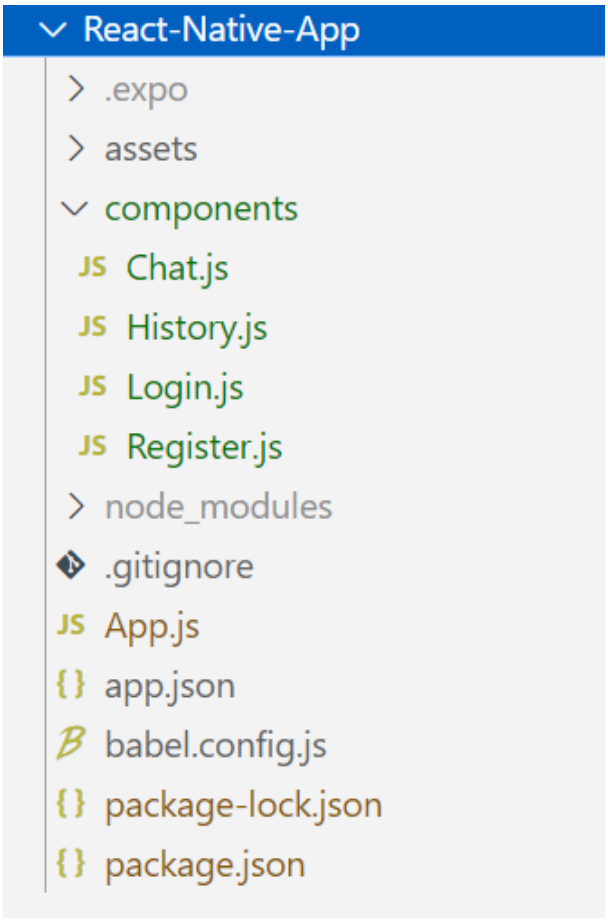
React: JavaScript library for building the user interface.

Vite: Fast frontend build tool.

Material-UI: React components for faster and easier web development.

React Router: Declarative routing for React applications.

Mobile Application (React Native)



Expo: Framework and platform for universal React applications.

React Navigation: Routing and navigation for React Native applications.

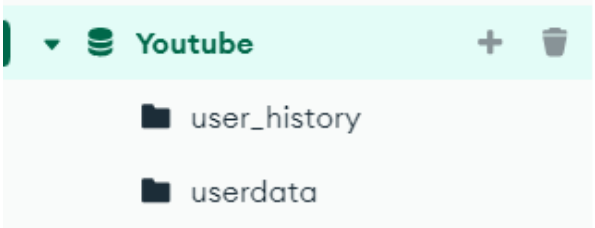
AsyncStorage: Persistent, key-value storage system for React Native applications.

Installation and Setup

Backend

Install Python dependencies: pip install -r requirements.txt

Set up a MongoDB database.



Configure .env file with Google Gemini API key.

Run Flask server: python app.py

Frontend

Web Application

Navigate to the Frontend directory.

Install dependencies: npm install

Run the development server: npm run dev

Mobile Application (React Native)

Navigate to the React-Native-App directory.

Install dependencies: npm install

Start the server: npm start

Scan the QR code with the Expo Go app on your mobile device.

Development

Backend

- ➔ Create a folder inside your project repository called Backend.
- ➔ Navigate into Backend directory.
- ➔ Create a virtual environment myenv.

```
python -m venv myenv
```

- ➔ Create a text file called requirements.txt

```
flask
python-dotenv
google-generativeai
flask-cors
youtube-transcript-api
BeautifulSoup4
pymongo
requests
```

- ➔ Install all the requirements in backend using the command:

```
pip install -r requirements.txt
```

➔ Create a .env file and provide the Gemini API Key

➔ .env

```
GEMINI_API_KEY = "TYPE YOUR API KEY HERE"
```

➔ Create a file named Google_gemini.py

➔ Google_gemini.py

```
from dotenv import load_dotenv
import os
import google.generativeai as genAi

load_dotenv()

apiKey = os.getenv("GEMINI_API_KEY")

genAi.configure(api_key = apiKey)

def get_gemini_pro_response(input):
  model = genAi.GenerativeModel("gemini-pro")
  resp = model.generate_content(input)
  return resp.text
```

- from dotenv import load_dotenv: This line imports the load_dotenv function from the dotenv module, which is used to load environment variables from a .env file into the environment.
- import os: This line imports the built-in os module, which provides a portable way to use operating system-dependent functionality, such as reading environment variables.
- import google.generativeai as genAi: This line imports the generativeai module from the google package and assigns it an alias genAi.
- load_dotenv(): This line invokes the load_dotenv() function to load environment variables from a .env file located in the same directory as the Python script or from the specified path.
- apiKey = os.getenv("GEMINI_API_KEY"): This line retrieves the value of the environment variable named GEMINI_API_KEY using the os.getenv() function and assigns it to the variable apiKey. This environment variable likely contains the API key required to authenticate requests to the Gemini API.

- genAi.configure(api_key = apiKey): This line configures the Google Generative AI library (genAi) to use the API key retrieved from the environment variable. It sets the API key for authentication purposes.
- def get_gemini_pro_response(input): This line defines a Python function named get_gemini_pro_response that takes an input parameter, which presumably represents the input text for which a response is generated.
- model = genAi.GenerativeModel("gemini-pro"): This line creates an instance of the GenerativeModel class from the genAi module, specifically using the "gemini-pro" model. This model is likely a pre-trained AI model designed for generating text responses.
- resp = model.generate_content(input): This line invokes the generate_content() method of the GenerativeModel instance, passing the input text as an argument. It generates content (text) based on the provided input using the configured AI model.
- return resp.text: This line returns the generated text content as the output of the get_gemini_pro_response() function.

➔ Create a file named Youtube.py

➔ Youtube.py

```
from youtube_transcript_api import YouTubeTranscriptApi
from urllib.parse import urlparse, parse_qs
import requests
from bs4 import BeautifulSoup
```

- from youtube_transcript_api import YouTubeTranscriptApi: This line imports the YouTubeTranscriptApi class from the youtube_transcript_api module. This class allows fetching transcripts of YouTube videos.
- from urllib.parse import urlparse, parse_qs: This line imports the urlparse and parse_qs functions from the urllib.parse module. These functions are used for parsing URLs and extracting query parameters from them.
- import requests: This line imports the requests module, which is used for making HTTP requests.
- from bs4 import BeautifulSoup: This line imports the BeautifulSoup class from the bs4 module. BeautifulSoup is a library for parsing HTML and XML documents.

```
def extract_transcript_details(youtube_video_url):
```

```
    try:
        parsed_url = urlparse(youtube_video_url)
        video_id = parse_qs(parsed_url.query).get('v')[0]
        transcript_text = YouTubeTranscriptApi.get_transcript(video_id)
        transcript = " ".join([i["text"] for i in transcript_text])
        thumbnail_url = f"http://img.youtube.com/vi/{video_id}/0.jpg"
        return transcript, thumbnail_url
    except Exception as e:
        raise e
```

- def extract_transcript_details(youtube_video_url): This line defines a function named extract_transcript_details that takes a youtube_video_url parameter. This function extracts the transcript text and thumbnail URL of a YouTube video.
- parsed_url = urlparse(youtube_video_url): This line parses the input YouTube video URL using the urlparse function from earlier.
- video_id = parse_qs(parsed_url.query).get('v')[0]: This line extracts the video ID from the parsed URL's query parameters. The video ID uniquely identifies the YouTube video.
- transcript_text = YouTubeTranscriptApi.get_transcript(video_id): This line fetches the transcript of the YouTube video with the given video ID using the get_transcript method of the YouTubeTranscriptApi class.
- transcript = " ".join([i["text"] for i in transcript_text]): This line joins the text of individual transcript entries into a single string, separating them with spaces.
- thumbnail_url = f"http://img.youtube.com/vi/{video_id}/0.jpg": This line constructs the URL for the thumbnail image of the YouTube video using the video ID.
- return transcript, thumbnail_url: This line returns the extracted transcript text and thumbnail URL as a tuple.
- except Exception as e: and raise e: These lines handle any exceptions that occur during the execution of the function and re-raise them for error handling in the calling code.

```
def get_video_title(youtube_video_url):
    try:
        response = requests.get(youtube_video_url)
        soup = BeautifulSoup(response.text, 'html.parser')
        title = soup.find('title').get_text()
        return title
    except Exception as e:
        raise e
```

- def get_video_title(youtube_video_url):: This line defines a function named get_video_title that takes a youtube_video_url parameter. This function retrieves the title of a YouTube video.
- response = requests.get(youtube_video_url): This line makes an HTTP GET request to the provided YouTube video URL using the requests.get function, fetching the HTML content of the video page.
- soup = BeautifulSoup(response.text, 'html.parser'): This line creates a BeautifulSoup object soup from the HTML content of the video page, using the 'html.parser' parser.
- title = soup.find('title').get_text(): This line finds the <title> tag in the HTML content and extracts its text, which represents the title of the video.
- return title: This line returns the extracted video title.
- Error handling with except Exception as e: and raise e is similar to the previous function, raising any encountered exceptions for handling in the calling code.

➔ Create a file named app.py

➔ app.py

```
from flask import Flask, jsonify, request
```

- This line imports the Flask class from the flask module, as well as the jsonify and request functions. Flask is a web framework for Python, jsonify is used to create JSON responses, and request is used to access incoming request data.

```
from flask_cors import CORS
```

- This line imports the CORS class from the flask_cors module. CORS (Cross-Origin Resource Sharing) is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served.

```
from Google_gemini import get_gemini_pro_response
from Youtube import extract_transcript_details, get_video_title
```

- These lines import functions (get_gemini_pro_response, extract_transcript_details, and get_video_title) from separate modules (Google_gemini and Youtube). These functions are likely used for generating responses based on input data.

```
from pymongo import MongoClient
```

- This line imports the MongoClient class from the pymongo module. MongoClient is used to establish a connection with the MongoDB database.

```
app = Flask(__name__)
CORS(app)
```

- These lines create a Flask application instance named app and enable CORS for the application using the CORS class. Enabling CORS allows the application to handle cross-origin requests.

```
client = MongoClient('mongodb://localhost:27017')
db = client['Youtube']
collection = db['user_history']
user_collection = db['userdata']
```

- These lines establish a connection to the MongoDB database running on localhost at port 27017. It creates a database named Youtube and two collections named user_history and userdata.

```
@app.route("/youtube_summary", methods=['POST'])

def generate_summary():
    try:
        username = request.args.get('username')

        data = request.json
        youtube_video_url = data.get('youtube_url')
        transcript_text, thumbnail_url = extract_transcript_details(youtube_video_url)
        video_title = get_video_title(youtube_video_url)

        prompt = """
            You are youtube video summarizer. You will be taking the transcript text and
summarizing the entire video and providing the important summary in points within 250
words. Please provide the summary of the text given here.
        """
        final_prompt = transcript_text + prompt
        response = get_gemini_pro_response(final_prompt)

        history = {
            'username': username,
            'Title': video_title,
            'Thumbnail': thumbnail_url,
            'Summary': response
        }
```

```
        collection.insert_one(history)

        return jsonify({"response": response, "thumbnail_url": thumbnail_url,
"video_title": video_title})

    except Exception as e:
        return jsonify({"error": str(e)})
```

- This block of code defines a route /youtube_summary that accepts POST requests. When a request is received, it extracts the username and YouTube video URL from the request data. It then extracts the transcript text and thumbnail URL of the video using the imported functions. It generates a prompt and response using other functions and inserts the summarized data into the user_history collection in the database. Finally, it returns a JSON response containing the response, thumbnail URL, and video title.

```
@app.route("/chat_history", methods=['GET'])
def get_chat_history():
    try:
        username = request.args.get('username')
        chat_history = list(collection.find({'username': username}, {'_id': 0}))
        return jsonify(chat_history)

    except Exception as e:
        return jsonify({"error": str(e)})
```

- This block of code defines a route /chat_history that accepts GET requests. When a request is received, it extracts the username from the request parameters and retrieves the chat history for that user from the user_history collection in the database. It then returns the chat history as a JSON response.

```
@app.route("/login", methods=['POST'])
def login():
    try:
        data = request.json
        username = data.get('username')
        password = data.get('password')

        user = user_collection.find_one({'username': username, 'password': password})
        if user:
            return jsonify({"message": "Success"}), 200
        else:
            return jsonify({"message": "Invalid username or password"}), 401

    except Exception as e:
```

```
        return jsonify({"error": str(e)})
```

- This block of code defines a route /login that accepts POST requests. When a request is received, it extracts the username and password from the request data. It then queries the userdata collection in the database to check if the username and password are valid. It returns a JSON response indicating whether the login was successful or not.

```
@app.route("/register", methods=['POST'])
def register():
    try:
        data = request.json
        name = data.get('Name')
        username = data.get('username')
        password = data.get('password')

        if user_collection.find_one({'username': username}):
            return jsonify({"message": 'Exists'}), 400

        new_user = {
            'Name': name,
            'username': username,
            'password': password
        }

        user_collection.insert_one(new_user)

        return jsonify({"message": 'Success'}), 201

    except Exception as e:
        return jsonify({"error": str(e)})
```

- This block of code defines a route /register that accepts POST requests. When a request is received, it extracts the user's name, username, and password from the request data. It then checks if the username already exists in the database. If not, it inserts the new user data into the userdata collection and returns a JSON response indicating successful registration.

```
if __name__ == "__main__":
    app.run(host='192.168.0.6', port = 5000,debug=True)
```

- This conditional block ensures that the Flask web application is run only when the script is executed directly (not when imported as a module). It starts the Flask development server on host 192.168.0.6 and port 5000, with debug mode enabled. This allows the application to

automatically reload when code changes are detected and provides a helpful debugger in case of errors.

Frontend

Web Application

➔ Create a vite project called Frontend using the command:

npm create vite Frontend -- --template react

➔ Navigate into the Frontend directory.

➔ Run the command to install all the dependencies required for the project:

npm install @emotion/react @emotion/styled @fontsource/roboto @mui/icons-material @mui/material react react-dom react-router react-router-dom react-youtube

➔ main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'

import { RouterProvider, createBrowserRouter } from 'react-router-dom'
import Login from './components/Login.jsx'
import Register from './components/Register.jsx'
import Chat_page from './components/Chat_page.jsx'

const routes = createBrowserRouter([
    {path: "/", element: <App/>, children: [
        {path: "/login", element: <Login/>},
        {path: "/register", element: <Register/>},
        {path: "/Chat", element: <Chat_page/>}
    ]}
])

ReactDOM.createRoot(document.getElementById('root')).render(
    <RouterProvider router={routes}/>
)
```

- import React from 'react': This line imports the React library, which is necessary for writing React components and applications.
- import ReactDOM from 'react-dom/client': This line imports the ReactDOM module from the react-dom/client package. ReactDOM is used for rendering React components into the DOM.
- import App from './App.jsx': This line imports the App component from the App.jsx file. The App component is typically the root component of a React application.

- import './index.css': This line imports a CSS file named index.css, which likely contains styles used throughout the application.
- import { RouterProvider, createBrowserRouter } from 'react-router-dom': This line imports the RouterProvider and createBrowserRouter components from the react-router-dom package. These components are used for managing routing in a React application.
- import Login from './components/Login.jsx': This line imports the Login component from the Login.jsx file. This component is presumably used for the login functionality of the application.
- import Register from './components/Register.jsx': This line imports the Register component from the Register.jsx file. This component is likely used for user registration.
- import Chat_page from './components/Chat_page.jsx': This line imports the Chat_page component from the Chat_page.jsx file. This component probably represents the main chat page of the application.
- const routes = createBrowserRouter([...]): This line creates a router configuration using the createBrowserRouter function. It defines the routes of the application. Here, it sets up routes for the App, Login, Register, and Chat_page components.
- ReactDOM.createRoot(document.getElementById('root')).render(...): This line renders the root component of the application (<RouterProvider>) into the DOM. It uses ReactDOM.createRoot to create a root render tree and then renders the RouterProvider component, passing the routes as a prop.

➔ App.jsx

```
import { NavLink, Outlet, useLocation } from "react-router-dom";
import React, { useState, useEffect } from 'react';

import Button from '@mui/material/Button';

function App() {
  const [clicked, setClicked] = useState(false);

  const location = useLocation();

  useEffect(() => {
    if (location.pathname === "/") {
      setClicked(false);
    }
  }, [location]);

  const handleButtonClick = () => {
    setClicked(true);
  };

  return (
```

```
    <div>
        <>
          {!clicked && (
            <>
            <div className="home-buttons-container">
              <Button variant="outlined" onClick={handleButtonClick} component={NavLink}
to="/login" style={{ textDecoration: 'none' }}>Login</Button>
              <Button variant="outlined" onClick={handleButtonClick} component={NavLink}
to="/register" style={{ textDecoration: 'none' }}>Register</Button>
            </div>
            </>
          )}
          <Outlet/>
        </>
    </div>
  );
}

export default App;
```

- import { NavLink, Outlet, useLocation } from "react-router-dom";: This line imports the NavLink, Outlet, and useLocation components from the react-router-dom package. These components are used for navigation and accessing routing-related information in a React application.
- import React, { useState, useEffect } from 'react';: This line imports the React library along with the useState and useEffect hooks from the react package. useState is a hook used for managing state in functional components, and useEffect is a hook used for handling side effects in functional components.
- import Button from '@mui/material/Button';: This line imports the Button component from the Material-UI library. Material-UI provides pre-designed components that can be used to build the user interface of React applications.
- function App() { ... }: This defines a functional component named App.
- const [clicked, setClicked] = useState(false);: This line uses the useState hook to declare a state variable named clicked and a function named setClicked to update its value. The initial value of clicked is false.
- const location = useLocation();: This line uses the useLocation hook to access the current location object from the React Router. It returns an object containing information about the current URL.
- useEffect(() => { ... }, [location]);: This line uses the useEffect hook to execute a function when the component mounts or when the location object changes. It checks if the current pathname is / and sets clicked to false accordingly.

- const handleButtonClick = () => { ... };: This defines a function named handleButtonClick, which sets the clicked state to true when called.
- return (...);: This is the JSX returned by the App component. It renders a div element containing conditional rendering logic and the Outlet component, which renders child routes based on the current URL.
- {!clicked && (...)}: This is a conditional rendering block that renders its children only if clicked is false. Inside this block, two Button components are rendered, each linked to a different route (/login and /register).
- <Outlet/>: This component is a placeholder where child routes are rendered based on the current URL. It is provided by React Router and is typically used in parent components to render child routes.

➔ Inside src folder create a new folder called components and inside components folder create a new component called Login.jsx

➔ Login.jsx

```
import { useNavigate } from "react-router-dom"
import TextField from '@mui/material/TextField';
import Button from '@mui/material/Button';
import { useState } from "react";
import Alert from '@mui/material/Alert';

const Login = () => {
  const navigate = useNavigate();
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [showLoginErrorAlert, setShowLoginErrorAlert] = useState(false);

  const handleLogin = () => {
    fetch('http://192.168.0.6:5000/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ username: username, password: password }),
    })
    .then(response => response.json())
    .then(data => {
      if(data.message === "Success"){
        sessionStorage.setItem('username', username);
          navigate("/Chat")
      }
      else {
        setShowLoginErrorAlert(true);
        setTimeout(() => {
          setShowLoginErrorAlert(false);
```

```
        }, 1000);
      }
    })
    .catch(error => {
      console.error('Error fetching data:', error);
    });
  }

  const isLoginDisabled = !username || !password;

  return (
    <div>
      <>
        <div className="login-container">
          <TextField label="Username" variant="outlined" onChange={(e) =>
setUsername(e.target.value)} />
          <br /><br />
          <TextField label="Password" variant="outlined" type="password" onChange={(e) =>
setPassword(e.target.value)} />
          <br /><br />
          <Button variant="outlined" onClick={handleLogin}
disabled={isLoginDisabled}>Login</Button> <br /> <br />
        </div>
      </>

      <div className="login-failure-alert-message">
        {showLoginErrorAlert && (
          <Alert severity="error">
            Invalid Username or Password
          </Alert>
        )}
      </div>

    </div>
  );
}

export default Login;
```

- import { useNavigate } from "react-router-dom": This imports the useNavigate hook from the react-router-dom package. This hook allows for programmatic navigation in React applications.
- import TextField from '@mui/material/TextField';: This imports the TextField component from the Material-UI library. TextField is used to create input fields in forms.
- import Button from '@mui/material/Button';: This imports the Button component from Material-UI. The Button component creates clickable buttons in the user interface.

- import { useState } from "react";: This imports the useState hook from the react package. useState is a React hook used to manage state in functional components.
- import Alert from '@mui/material/Alert';: This imports the Alert component from Material-UI. The Alert component is used to display alert messages in the user interface.
- const Login = () => { ... }: This defines a functional component named Login.
- const navigate = useNavigate();: This line initializes the navigate function using the useNavigate hook imported earlier. This function allows the component to navigate to different routes programmatically.
- const [username, setUsername] = useState('');: This line declares a state variable username and a function setUsername to update its value. The initial value of username is an empty string.
- const [password, setPassword] = useState('');: This line is similar to the previous one, but it declares a state variable password instead.
- const [showLoginErrorAlert, setShowLoginErrorAlert] = useState(false);: This line declares a state variable showLoginErrorAlert and a function setShowLoginErrorAlert to control whether to show the login error alert message. The initial value is false, indicating that the alert is initially hidden.
- const handleLogin = () => { ... }: This defines a function handleLogin to handle the login process. It sends a POST request to the server with the entered username and password, then navigates to the chat page if login is successful, or displays an error alert message otherwise.
- const isLoginDisabled = !username || !password;: This line calculates the value of isLoginDisabled based on whether the username or password fields are empty. If either field is empty, the login button will be disabled.
- The return statement contains JSX code representing the component's UI. It renders input fields for username and password, a login button, and an error alert message if the login fails.
- {showLoginErrorAlert && (...)}: This is a conditional rendering block that displays the error alert message if showLoginErrorAlert is true.

➔ Create another component inside the components folder called Register.jsx

➔ Register.jsx

```
import { useState } from "react";
import { useNavigate } from "react-router-dom";
import Button from '@mui/material/Button';
import TextField from '@mui/material/TextField';
import Alert from '@mui/material/Alert';

const Register = () => {
```

```
  const navigate = useNavigate();
  const [name, setName] = useState('');
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [showSuccessAlert, setShowSuccessAlert] = useState(false);
  const [showErrorAlert, setShowErrorAlert] = useState(false);

  const handleRegister = () => {
    fetch('http://192.168.0.6:5000/register', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ Name: name, username: username, password: password }),
    })
    .then(response => response.json())
    .then(data => {
      if (data.message === "Success") {
        setShowSuccessAlert(true);
        setTimeout(() => {
          setShowSuccessAlert(false);
          navigate("/login");
        }, 1000);
      } else {
        setShowErrorAlert(true);
        setTimeout(() => {
          setShowErrorAlert(false);
        }, 1000);
      }
    })
    .catch(error => {
      console.error('Error fetching data:', error);
    });
  };

  const isRegisterDisabled = !name || !username || !password;

  return <div>
    <>
      <div className="register-container">
        <TextField label="Name" variant="outlined" onChange={(e) =>
setName(e.target.value)} />
        <br /><br />
        <TextField label="Username" variant="outlined" onChange={(e) =>
setUsername(e.target.value)} />
        <br /><br />
        <TextField label="Password" type="password" variant="outlined" onChange={(e) =>
setPassword(e.target.value)} />
        <br /><br />
        <Button onClick={handleRegister} variant="outlined" disabled={isRegisterDisabled}>
          Register
        </Button> <br /> <br />
        </div>
    </>
```

```
      <div className="register-success-alert-message">
        {showSuccessAlert && (
          <Alert severity="success">
            Account Created Successfully 😘😘
          </Alert>
        )}
      </div>

      <div className="register-failure-alert-message">
        {showErrorAlert && (
          <Alert severity="error">
            User already exists. Please choose a different username.
          </Alert>
        )}
      </div>

  </div>
}

export default Register;
```

- import { useState } from "react";: This line imports the useState hook from the React library. The useState hook is used to add state to functional components in React.
- import { useNavigate } from "react-router-dom";: This line imports the useNavigate hook from the React Router DOM library. The useNavigate hook allows navigation to different pages within a React application.
- import Button from '@mui/material/Button';: This line imports the Button component from the Material-UI library. Material-UI is a popular library for building user interfaces in React, and the Button component is used to create clickable buttons.
- import TextField from '@mui/material/TextField';: This line imports the TextField component from Material-UI. The TextField component is used to create input fields for users to enter text.
- import Alert from '@mui/material/Alert';: This line imports the Alert component from Material-UI. The Alert component is used to display alert messages to the user.
- const Register = () => { ... }: This declares a functional component named Register. It initializes state variables using the useState hook and defines event handlers for handling registration and input changes.
- const navigate = useNavigate();: This line initializes the navigate variable with the useNavigate hook. It allows for programmatic navigation to different pages within the application.
- const [name, setName] = useState('');: This line initializes the name state variable using the useState hook. It also defines a function setName to update the name state.

- const [username, setUsername] = useState('');: This line initializes the username state variable using the useState hook. It also defines a function setUsername to update the username state.
- const [password, setPassword] = useState('');: This line initializes the password state variable using the useState hook. It also defines a function setPassword to update the password state.
- const [showSuccessAlert, setShowSuccessAlert] = useState(false);: This line initializes the showSuccessAlert state variable using the useState hook. It also defines a function setShowSuccessAlert to update the showSuccessAlert state.
- const [showErrorAlert, setShowErrorAlert] = useState(false);: This line initializes the showErrorAlert state variable using the useState hook. It also defines a function setShowErrorAlert to update the showErrorAlert state.
- const handleRegister = () => { ... }: This function is called when the user clicks the "Register" button. It sends a POST request to the backend server with the user's registration details (name, username, password). It then handles the response from the server, showing a success or error message based on the response.
- fetch('http://192.168.0.6:5000/register', { ... }): This line initiates a POST request to the specified URL (http://192.168.0.6:5000/register) using the Fetch API. It includes the registration data (name, username, password) in the request body as JSON.
- .then(response => response.json()): This line extracts the JSON body content from the response returned by the server.
- .then(data => { ... }): This block handles the data returned from the server. If the registration is successful (data.message === "Success"), it sets showSuccessAlert to true, displays a success message to the user, and navigates to the login page after a short delay. Otherwise, it sets showErrorAlert to true and displays an error message to the user.
- .catch(error => { ... }): This block catches any errors that occur during the fetch request and logs them to the console.
- const isRegisterDisabled = !name || !username || !password;: This line determines whether the "Register" button should be disabled based on whether the name, username, or password fields are empty.
- <TextField label="Name" variant="outlined" onChange={(e) => setName(e.target.value)} />: This line renders a text input field for the user's name. It uses the TextField component from Material-UI and sets up an onChange event handler to update the name state whenever the user types in the input field.
- <Button onClick={handleRegister} variant="outlined" disabled={isRegisterDisabled}>Register</Button>: This line renders a button labeled

"Register". It calls the handleRegister function when clicked, disables the button if isRegisterDisabled is true, and applies the outlined style variant.

- <Alert severity="success">...</Alert> and <Alert severity="error">...</Alert>: These lines render success and error alert messages using the Material-UI Alert component. They display messages to the user indicating the outcome of the registration attempt.

➔ Create another component inside the components folder called Youtube_Summarizer.jsx

➔ Youtube_Summarizer.jsx

```
import React, { useState } from 'react';
import Button from '@mui/material/Button';
import SendIcon from '@mui/icons-material/Send';
import TextField from '@mui/material/TextField';

const YoutubeSummarizer = () => {

  const username = sessionStorage.getItem('username');

  const [summary, setSummary] = useState('');
  const [thumbnailURL, setThumbnailURL] = useState('');
  const [youtubeURL, setYoutubeURL] = useState('');
  const [videoTitle, setVideoTitle] = useState('');

  const handleSummarize = () => {
    fetch(`http://192.168.0.6:5000/youtube_summary?username=${username}`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ youtube_url: youtubeURL }),
    })
        .then(response => response.json())
        .then(data => {
            setSummary(data.response);
            setThumbnailURL(data.thumbnail_url);
            setVideoTitle(data.video_title);
            console.log(data);
        })
        .catch(error => {
            console.error('Error fetching data:', error);
        });
  };

  const handleInputChange = (event) => {
    setYoutubeURL(event.target.value);
  };

  return (
    <div>
      <TextField id="outlined-basic" label="Enter YouTube video URL" variant="outlined"
onChange={handleInputChange} />
```

```
      <br /><br />
      <Button onClick={handleSummarize} variant="contained" endIcon={<SendIcon />}>
        Summarize
      </Button>
      <br /><br />
      <h2>Video Title: {videoTitle}</h2>
      <img src={thumbnailURL} alt="Video Thumbnail" />
      <h2>Summary:</h2>
      <p>{summary}</p>
    </div>
  );
};

export default YoutubeSummarizer;
```

- import React, { useState } from 'react';: This line imports the React library and the useState hook from React. The useState hook allows functional components to manage state.
- import Button from '@mui/material/Button';: This line imports the Button component from the Material-UI library. Material-UI provides pre-styled components for building user interfaces in React.
- import SendIcon from '@mui/icons-material/Send';: This line imports the SendIcon component from the Material-UI library. It represents an icon that will be displayed at the end of the button.
- import TextField from '@mui/material/TextField';: This line imports the TextField component from the Material-UI library. It provides a styled text input field.
- const YoutubeSummarizer = () => { ... }: This defines a functional component named YoutubeSummarizer.
- const username = sessionStorage.getItem('username');: This line retrieves the username from the session storage. The username is stored in the session storage when the user logs in.
- const [summary, setSummary] = useState('');: This line initializes a state variable summary and a function setSummary to update its value. The initial value of summary is an empty string.
- const [thumbnailURL, setThumbnailURL] = useState('');: This line initializes a state variable thumbnailURL and a function setThumbnailURL to update its value. The initial value of thumbnailURL is an empty string.
- const [youtubeURL, setYoutubeURL] = useState('');: This line initializes a state variable youtubeURL and a function setYoutubeURL to update its value. The initial value of youtubeURL is an empty string.

- const [videoTitle, setVideoTitle] = useState('');: This line initializes a state variable videoTitle and a function setVideoTitle to update its value. The initial value of videoTitle is an empty string.
- const handleSummarize = () => { ... }: This defines a function handleSummarize that is called when the "Summarize" button is clicked. It sends a POST request to the backend server to summarize the YouTube video.
- const handleInputChange = (event) => { ... }: This defines a function handleInputChange that is called when the value of the text input field changes. It updates the youtubeURL state variable with the new input value.
- return (...): This returns the JSX (JavaScript XML) markup that defines the component's UI.
- <TextField id="outlined-basic" label="Enter YouTube video URL" variant="outlined" onChange={handleInputChange} />: This line renders a text input field for entering the YouTube video URL. It uses the TextField component from Material-UI and calls the handleInputChange function when the input value changes.
- <Button onClick={handleSummarize} variant="contained" endIcon={<SendIcon />}>: This line renders a button labeled "Summarize". It calls the handleSummarize function when clicked and displays the SendIcon at the end of the button.
- <h2>Video Title: {videoTitle}</h2>: This line renders the video title retrieved from the backend server.
- : This line renders the thumbnail image of the YouTube video.
- <h2>Summary:</h2>: This line renders a heading for the summary section.
- {summary} : This line renders the summary of the YouTube video retrieved from the backend server.

➔ Create another component called Chat_history.jsx in the components folder.

➔ Chat_history.jsx

```
import React, { useState, useEffect } from 'react';
import Container from '@mui/material/Container';

const Chat_history = () => {
  const [chatHistory, setChatHistory] = useState([]);
  const username = sessionStorage.getItem('username');

  useEffect(() => {
    fetchChatHistory();
    const intervalId = setInterval(fetchChatHistory, 1000);
    return () => {
      clearInterval(intervalId);
    };
```

```
  }, []);

  const fetchChatHistory = () => {
    fetch(`http://192.168.0.6:5000/chat_history?username=${username}`)
      .then(response => response.json())
      .then(data => {
        if (Array.isArray(data)) {
          const reversedChatHistory = data.reverse();
          setChatHistory(reversedChatHistory);
        } else {
          console.error('Unexpected response format:', data);
        }
      })
      .catch(error => {
        console.error('Error fetching chat history:', error);
      });
  };

  return (
    <div>
      <h2>Chat History:</h2>
      <ul>
        {chatHistory.map((chat, index) => (
          <Container key={index} sx={{ bgcolor: '#cfe8fc', marginTop: 2, padding: 2 }}>
          <li key={index}>
            <strong>{chat.Title}: <br />
            <img src={chat.Thumbnail} alt="Thumbnail" style={{ maxWidth: '100px',
maxHeight: '100px' }} />
            <br /> </strong> {chat.Summary}
            </li>
          </Container>
        ))}
      </ul>
    </div>
  );
};

export default Chat_history;
```

- import React, { useState, useEffect } from 'react';: This line imports the necessary dependencies from the React library, including the useState and useEffect hooks.
- import Container from '@mui/material/Container';: This line imports the Container component from the Material-UI library. It's used to wrap and center content within a fixed-width container.
- const Chat_history = () => { ... }: This defines a functional component named Chat_history.
- const [chatHistory, setChatHistory] = useState([]);: This line initializes a state variable chatHistory and a function setChatHistory to update its value. The initial value of chatHistory is an empty array.

- const username = sessionStorage.getItem('username');: This line retrieves the username from the session storage. The username is stored in the session storage when the user logs in.
- useEffect(() => { ... }, []);: This is a useEffect hook that runs when the component mounts. It fetches the chat history and sets up an interval to fetch updates periodically. The empty dependency array [] indicates that the effect runs only once when the component mounts.
- const fetchChatHistory = () => { ... }: This is a function responsible for fetching the chat history from the server using a GET request.
- fetch(http://192.168.0.6:5000/chat_history?username=${username}` ) ...` : This line fetches the chat history from the backend server using the username as a query parameter.
- .then(response => response.json()) ...: This line converts the response from the server into JSON format.
- .then(data => { ... }): This line handles the JSON data received from the server. If the data is an array, it reverses the order of the chat history and sets it in the chatHistory state variable. If the data is not an array, it logs an error.
- .catch(error => { ... }): This line catches any errors that occur during the fetch operation and logs them.
- return (...): This returns the JSX markup that defines the component's UI.
- <h2>Chat History:</h2>: This line renders a heading for the chat history section.
- {chatHistory.map((chat, index) => (...))}: This line maps through the chatHistory array and renders each chat item as a list item.
- <Container key={index} sx={{ bgcolor: '#cfe8fc', marginTop: 2, padding: 2 }}>: This line wraps each chat item in a Material-UI Container component. It provides styling such as background color, margin, and padding.
- <li key={index}> ... : This line renders each chat item as a list item.
- {chat.Title}:
 ... : This line renders the title of the video followed by a line break within a strong tag.
- : This line renders the thumbnail image of the video with a maximum width and height of 100 pixels.
- {chat.Summary}: This line renders the summary of the video.
- </Container>: This line closes the Material-UI Container component.
- : This line closes the unordered list element.
- : This line closes the div element wrapping the entire chat history section.

➔ Create another component called Chat_page.jsx.

➔ Chat_page.jsx

```
import React from 'react';
import YoutubeSummarizer from './Youtube_Summarizer';
import Chat_history from './Chat_history';
import '../index.css';
import { NavLink, useNavigate } from "react-router-dom"
import Button from '@mui/material/Button';

const Chat_page = () => {
  const navigate = useNavigate();

  const handleLogout = () => {
    sessionStorage.removeItem('username');
    navigate("/login");
  };

  return (
    <div className="chat-page-container">
      <div className="left-column" style={{ height: 'calc(100vh - 64px)', overflow: 'auto'
}}>
        <Chat_history />
      </div>
      <div className="right-column" style={{ height: 'calc(100vh - 64px)', overflow:
'auto' }}>
        <Button variant="outlined" component={NavLink} to="/" style={{ textDecoration:
'none' }} onClick={handleLogout}>Logout</Button>
        <br /><br />
        <YoutubeSummarizer />
      </div>
    </div>
  );
};

export default Chat_page;
```

- import React from 'react';: This line imports the necessary dependencies from the React library.
- import YoutubeSummarizer from './Youtube_Summarizer';: This line imports the YoutubeSummarizer component from the Youtube_Summarizer.jsx file.
- import Chat_history from './Chat_history';: This line imports the Chat_history component from the Chat_history.jsx file.
- import '../index.css';: This line imports the CSS file to style the components in this file.
- import { NavLink, useNavigate } from "react-router-dom": This line imports the NavLink component and the useNavigate hook from the React Router DOM library. NavLink is used for navigation links, and useNavigate is used to programmatically navigate to different routes.

- import Button from '@mui/material/Button';: This line imports the Button component from the Material-UI library.
- const Chat_page = () => { ... }: This defines a functional component named Chat_page.
- const navigate = useNavigate();: This line initializes the navigate function using the useNavigate hook. It will be used to navigate to different routes programmatically.
- const handleLogout = () => { ... }: This defines a function handleLogout that removes the username from the session storage and navigates the user to the login page when called.
- return (...): This returns the JSX markup that defines the component's UI.
- : This line starts a div element with a class name of chat-page-container, which serves as the main container for the chat page.
- <div className="left-column" style={{ height: 'calc(100vh - 64px)', overflow: 'auto' }}>: This line starts a div element with a class name of left-column. It sets the height to calc(100vh - 64px) and enables vertical scrolling if the content overflows.
- <Chat_history />: This line renders the Chat_history component, displaying the chat history in the left column.
- <div className="right-column" style={{ height: 'calc(100vh - 64px)', overflow: 'auto' }}>: This line starts a div element with a class name of right-column. It sets the height to calc(100vh - 64px) and enables vertical scrolling if the content overflows.
- <Button variant="outlined" component={NavLink} to="/" style={{ textDecoration: 'none' }} onClick={handleLogout}>Logout</Button>: This line renders a logout button using the Button component from Material-UI. It is wrapped in a NavLink component to navigate to the login page when clicked. The handleLogout function is called when the button is clicked.
- <YoutubeSummarizer />: This line renders the YoutubeSummarizer component, displaying the YouTube summarizer in the right column.
- : This line closes the div element for the right column.
- : This line closes the div element for the main container.

➔ index.css

```
body{
    background-color: aliceblue;
    font-family: system-ui, -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen,
Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
}

  /* App.jsx styling */
  .home-buttons-container {
    display: flex;
    gap: 20px;
```

```
    margin-top: 342.5px;
    margin-left: 675px;
  }

  /* Register.jsx styling */

  .register-container {
    margin-top: 225px;
    margin-left: 647.5px;
  }

  .register-success-alert-message {
    max-width: 300px;
    margin-left: 622px;
  }

  .register-failure-alert-message {
    max-width: 350px;
    margin-left: 580px;
  }

  /* Login.jsx styling */

  .login-container {
    margin-top: 265px;
    margin-left: 647.5px;
  }

  .login-failure-alert-message {
    max-width: 261px;
    margin-left: 630px;
  }


/* ChatPage.jsx  */

.chat-page-container {
  display: flex;
  flex-direction: row;
}

.left-column {
  flex: 1;
  padding: 20px;
}

.right-column {
  flex: 1;
  padding: 20px;
}

::-webkit-scrollbar {
  width: 0;
}
```

Mobile Application

➔ Create a react native expo project using the command:

npx create-expo-app React-Native-App

➔ Navigate to React-Native-App directory.

➔ Run the following command to install the required dependencies for the project:

npm install @expo/vector-icons @react-native-async-storage/async-storage @react-native-community/masked-view @react-navigation/bottom-tabs @react-navigation/native @react-navigation/stack expo expo-status-bar react react-native react-native-gesture-handler react-native-reanimated react-native-safe-area-context react-native-screens react-native-webview

➔ App.js

```
import React from 'react';
import { NavigationContainer } from "@react-navigation/native";
import { createBottomTabNavigator } from "@react-navigation/bottom-tabs";
import { Alert, View, Button } from 'react-native';
import AsyncStorage from '@react-native-async-storage/async-storage';

import Login from "./components/Login";
import Register from "./components/Register";
import Chat from "./components/Chat";
import History from "./components/History";

import { MaterialIcons } from '@expo/vector-icons';
import { MaterialCommunityIcons } from '@expo/vector-icons';
```

- This block imports the necessary modules and components from React, React Navigation, React Native, and other libraries.
- It imports components for Login, Register, Chat, and History from their respective files.
- It also imports icons from @expo/vector-icons.

```
const App = () => {
  const Tab = createBottomTabNavigator();

  const handleLogout = async () => {
      await AsyncStorage.removeItem('username');
      Alert.alert('Logout', 'You have been successfully logged out');
  };

  return <>
      <NavigationContainer>
          <Tab.Navigator>
              {/* Tab Screens */}
          </Tab.Navigator>
```

```
      </NavigationContainer>
      <View>
          <Button title="Logout" onPress={handleLogout} />
      </View>
  </>
};
```

- This defines the main App component as a functional component.
- It initializes a Tab variable using createBottomTabNavigator() from React Navigation.
- It defines a function handleLogout to handle the logout action. This function removes the 'username' from AsyncStorage and shows an alert indicating successful logout.
- The return statement renders the JSX elements.
- Within the NavigationContainer, it sets up a bottom tab navigator with multiple screens.
- Finally, it renders a Button component for logout functionality.

```
<Tab.Screen name="Login" component={Login} options={
            { tabBarIcon: () => <MaterialIcons name="login" size={24} color="black" /> }
          } />
<Tab.Screen name="Register" component={Register} options={
            { tabBarIcon: () => <MaterialCommunityIcons name="registered-trademark"
size={24} color="black" /> }
          } />
<Tab.Screen name="Chat" component={Chat} options={
            { tabBarIcon: () => <MaterialIcons name="chat" size={24} color="black" /> }
          } />
<Tab.Screen name="History" component={History} options={
            { tabBarIcon: () => <MaterialIcons name="history" size={24} color="black" /> }
          } />
```

- Each Tab.Screen represents a tab in the bottom tab navigator.
- It specifies the screen's name, component to render, and options.
- The options object defines the appearance of the tab icon using Material Icons or Material Community Icons.

```
<View>
      <Button title="Logout" onPress={handleLogout} />
</View>
```

- This renders a Button component for logout functionality.
- It triggers the handleLogout function when pressed.

➔ Create a new folder inside the project folder called components.

➔ Inside the components folder create a Tab file called Login.js

➔ Login.js

```
import React, { useState } from 'react';
import { SafeAreaView, StyleSheet, TextInput, Button, View, Alert } from 'react-native';
import AsyncStorage from '@react-native-async-storage/async-storage';
```

- This section imports necessary modules and components from React, React Native, and AsyncStorage.
- useState is imported from React to manage component state.
- Components like SafeAreaView, StyleSheet, TextInput, Button, View, and Alert are imported from React Native for UI building.

```
const Login = ({ navigation }) => {
   const [username, setUsername] = useState('');
   const [password, setPassword] = useState('');
```

- This defines the functional component Login.
- It takes navigation as a prop from React Navigation.
- useState hooks are used to manage the state variables username and password.

```
const handleLogin = async () => {
      const response = await fetch('http://192.168.0.6:5000/login', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ username: username, password: password }),
      });
      const data = await response.json();
      if (data.message === "Success") {
        await AsyncStorage.setItem('username', username);
        navigation.navigate("Chat");
      } else {
        Alert.alert('Login Failed', 'Invalid Username or Password');
      }
```

- This function is called when the login button is pressed.
- It sends a POST request to the server with the entered username and password.
- If the response message is "Success", it stores the username in AsyncStorage and navigates to the Chat screen.
- If the login fails, it displays an alert with the message "Login Failed: Invalid Username or Password".

```
const isLoginDisabled = !username || !password;
```

- This variable checks if either the username or password is empty.
- It's used to disable the login button if either field is empty.

```
 return (
    <SafeAreaView style={styles.container}>
      <View style={styles.inputContainer}>
        <TextInput
          style={styles.input}
          placeholder="Enter Username"
          onChangeText={text => setUsername(text)}
        />
        <TextInput
          style={styles.input}
          placeholder="Enter Password"
          onChangeText={text => setPassword(text)}
          secureTextEntry={true}
        />
        <View style={styles.buttonContainer}>
          <Button title="Login" onPress={handleLogin} disabled={isLoginDisabled} />
        </View>
      </View>
    </SafeAreaView>
  );
```

- This section returns the JSX to render the Login component.
- It includes a SafeAreaView as the parent container with styles from the styles.container.
- Inside, it has a View with styles.inputContainer for styling.
- Within this View, it includes TextInput components for entering username and password, and a Button component for login.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 180,
    marginLeft: 50
  },
  inputContainer: {
    width: '80%',
    marginBottom: 10,
  },
  input: {
    height: 40,
    margin: 12,
    borderWidth: 1,
    padding: 10,
```

```
    width: '100%',
  },
  buttonContainer: {
    width: '100%',
    marginLeft: 12
  },
});
```

➔ Create another Tab file called Register.js

➔ Register.js

```
import React, { useState } from 'react';
import { SafeAreaView, StyleSheet, TextInput, Button, View, Alert } from 'react-native';
```

- This section imports necessary modules and components from React and React Native.
- useState is imported from React to manage component state.
- Components like SafeAreaView, StyleSheet, TextInput, Button, View, and Alert are imported from React Native for UI building.

```
const Register = ({ navigation }) => {
  const [name, setName] = useState('');
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
```

- This defines the functional component Register.
- It takes navigation as a prop from React Navigation.
- useState hooks are used to manage the state variables name, username, and password.

```
const handleRegister = async () => {

      const response = await fetch('http://192.168.0.6:5000/register', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ Name: name, username: username, password: password }),
      });
      const data = await response.json();
      if (data.message === "Success") {
```

```
        Alert.alert('Registration Successful', 'Account Created Successfully 🥹🥹', [
          { text: 'OK', onPress: () => navigation.navigate("Login") }
        ]);
      } else {
        Alert.alert('Registration Failed', 'User already exists. Please choose a different username.');
      }
  };
```

- This function is called when the register button is pressed.
- It sends a POST request to the server with the entered name, username, and password.
- If the registration is successful, it displays an alert with the message "Registration Successful" and navigates to the Login screen.
- If the registration fails (due to an existing username), it displays an alert with the message "Registration Failed".

```
const isRegisterDisabled = !name || !username || !password;
```

- This variable checks if any of the input fields (name, username, or password) are empty.
- It's used to disable the register button if any field is empty.

```
return (
   <SafeAreaView style={styles.container}>
     <View style={styles.inputContainer}>
       <TextInput
         style={styles.input}
         placeholder="Enter Name"
         onChangeText={text => setName(text)}
       />
       <TextInput
         style={styles.input}
         placeholder="Enter Username"
         onChangeText={text => setUsername(text)}
       />
       <TextInput
         style={styles.input}
         placeholder="Enter Password"
         onChangeText={text => setPassword(text)}
         secureTextEntry={true}
       />
       <View style={styles.buttonContainer}>
         <Button
           title="Register"
```

```
            onPress={handleRegister}
            disabled={isRegisterDisabled}
          />
        </View>
      </View>
    </SafeAreaView>
  );
```

- This section returns the JSX to render the Register component.
- It includes a SafeAreaView as the parent container with styles from the styles.container.
- Inside, it has a View with styles.inputContainer for styling.
- Within this View, it includes TextInput components for entering name, username, and password, and a Button component for registration.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 120,
    marginLeft: 50
  },
  inputContainer: {
    width: '80%',
    marginBottom: 10,
  },
  input: {
    height: 40,
    margin: 12,
    borderWidth: 1,
    padding: 10,
    width: '100%',
  },
  buttonContainer: {
    width: '100%',
    marginLeft: 12
  },
});

export default Register;
```

➔ Create another Tab file called Chat.js

➔ Chat.js

```
import React, { useState } from 'react';
import { View, Text, SafeAreaView, Button, Image, StyleSheet, TextInput, ScrollView, Alert
} from 'react-native';
```

```
import AsyncStorage from '@react-native-async-storage/async-storage';
```

- This section imports necessary modules and components from React and React Native.
- useState is imported from React to manage component state.
- Components like View, Text, SafeAreaView, Button, Image, StyleSheet, TextInput, ScrollView, and Alert are imported from React Native for UI building.
- AsyncStorage is imported to store data asynchronously.

```
const Chat = () => {
  const [youtubeURL, setYoutubeURL] = useState('');
  const [videoTitle, setVideoTitle] = useState('');
  const [thumbnailURL, setThumbnailURL] = useState('');
  const [summary, setSummary] = useState('');
```

- This defines the functional component Chat.
- It initializes state variables youtubeURL, videoTitle, thumbnailURL, and summary using the useState hook.

```
const storedUsername = await AsyncStorage.getItem('username');
      if (storedUsername != null){
        const response = await
fetch(`http://192.168.0.6:5000/youtube_summary?username=${storedUsername}`, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify({ youtube_url: youtubeURL }),
        });
        const data = await response.json();
        if (data.error) {
          Alert.alert('Error', data.error);
        } else {
          setSummary(data.response);
          setThumbnailURL(data.thumbnail_url);
          setVideoTitle(data.video_title);
        }
      }
      else{
        Alert.alert("Cannon't use Chat feature without loggin in");
      }
  };
```

- This function is called when the "Summarize" button is pressed.
- It retrieves the stored username from AsyncStorage.
- It sends a POST request to the server with the entered YouTube video URL and the stored username.
- If the response contains an error, it displays an alert with the error message.
- Otherwise, it sets the summary, thumbnail URL, and video title based on the response data.

```
const handleInputChange = (text) => {
   setYoutubeURL(text);
};
```

- This function is called when the text input for the YouTube URL changes.
- It updates the youtubeURL state variable with the new text value.

```
return (
    <SafeAreaView style={styles.container}>
      <View style={styles.inputContainer}>
        <TextInput
          style={styles.input}
          placeholder="Enter YouTube video URL"
          onChangeText={handleInputChange}
          value={youtubeURL}
        />
        <View style={styles.buttonContainer}>
          <Button title="Summarize" onPress={handleSummarize} />
        </View>
      </View>
      <ScrollView>
        <View style={styles.outputContainer}>
          <Text>Video Title: {videoTitle}</Text>
          {thumbnailURL !== '' && <Image source={{ uri: thumbnailURL }}
style={styles.thumbnail} />}
          <Text>Summary: {summary}</Text>
        </View>
      </ScrollView>
    </SafeAreaView>
  );
```

- This section returns the JSX to render the Chat component.
- It includes a SafeAreaView as the parent container with styles from the styles.container.

- Inside, it has an input container with a text input for entering the YouTube video URL and a button to summarize.
- Below the input container, it has an output container to display the video title, thumbnail image, and summary text within a ScrollView.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 20,
    width: '100%',
  },
  inputContainer: {
    width: '80%',
    marginBottom: 10,
  },
  input: {
    height: 40,
    margin: 12,
    borderWidth: 1,
    padding: 10,
    width: '100%',
  },
  buttonContainer: {
    width: '100%',
    marginLeft: 12,
  },
  outputContainer: {
    marginLeft: 12,
  },
  thumbnail: {
    width: 366,
    height: 250,
  },
});

export default Chat;
```

➔ Create another Tab file called History.js

➔ History.js

```
import React, { useState, useEffect } from 'react';
import { View, Text, ScrollView, Image, SafeAreaView, StyleSheet, Alert } from 'react-
native';
import AsyncStorage from '@react-native-async-storage/async-storage';
```

- This section imports necessary modules and components from React and React Native.

- useState and useEffect are imported from React to manage component state and handle side effects.
- Components like View, Text, ScrollView, Image, SafeAreaView, and StyleSheet are imported from React Native for UI building.
- AsyncStorage is imported to store data asynchronously.

```
const History = () => {
  const [chatHistory, setChatHistory] = useState([]);
  const [username, setUsername] = useState('');
  const [loggedIn, setLoggedIn] = useState(false);
```

- This defines the functional component History.
- It initializes state variables chatHistory, username, and loggedIn using the useState hook.

```
useEffect(() => {
    const fetchHistory = async () => {
      const storedUsername = await AsyncStorage.getItem('username');
      if (storedUsername) {
        setUsername(storedUsername);
        setLoggedIn(true);
        fetchChatHistory(storedUsername);
      } else {
        setChatHistory([]);
        setLoggedIn(false);
      }
    };

    fetchHistory();

    const interval = setInterval(fetchHistory, 1000);

    return () => clearInterval(interval);
  }, []);
```

- This useEffect hook is called when the component mounts.
- It fetches the chat history and sets up a refresh interval.
- The [] as the second argument ensures that the effect runs only once when the component mounts.

```
const fetchChatHistory = async (username) => {
    const response = await
fetch(`http://192.168.0.6:5000/chat_history?username=${username}`);
    const data = await response.json();
    if (Array.isArray(data)) {
      const reversedChatHistory = data.reverse();
      setChatHistory(reversedChatHistory);
    } else {
      console.error('Unexpected response format:', data);
    }
  };

  if (!loggedIn) {
    return null;
```

- This function fetches the chat history for the logged-in user and updates the chatHistory state variable.

```
return (
    <SafeAreaView style={styles.container}>
      <ScrollView>
        {chatHistory.map((chat, index) => (
          <View key={index} style={styles.outputContainer}>
            <Text>Chat Title: {chat.Title}</Text>
            <Image source={{ uri: chat.Thumbnail }} style={styles.thumbnail} />
            <Text>Summary: {chat.Summary}</Text>
          </View>
        ))}
      </ScrollView>
    </SafeAreaView>
  );
```

- This section returns the JSX to render the History component.
- It includes a SafeAreaView as the parent container with styles from the styles.container.
- Inside, it has a ScrollView to display the chat history.
- It maps through the chatHistory array and renders each chat item within a View component with appropriate styles.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 20,
    width: '100%',
  },
```

```
  outputContainer: {
    marginBottom: 20,
    padding: 10,
    borderWidth: 1,
    borderColor: '#ccc',
  },
  thumbnail: {
    width: 366,
    height: 250,
    marginTop: 10,
    marginBottom: 10,
  },
});

export default History;
```

Usage

Web Application

Register or log in to access the YouTube summarization feature.

Enter a YouTube video URL and click "Summarize" to generate a summary.

View chat history to see previously summarized videos.

Mobile Application

Login to access the YouTube summarization feature.

Enter a YouTube video URL and tap "Summarize" to generate a summary.

View chat history to see previously summarized videos.

Future Enhancements

User Profile: Allow users to update their profile information.

Improved Summarization: Implement advanced summarization techniques for more accurate results.

Real-Time Chat: Enable real-time chat functionality for users.

Customization Options: Provide users with customization options for summary length and format.

Conclusion

The YouTube Summarizer project offers a convenient solution for summarizing YouTube videos efficiently. With both web and mobile applications, users can access the summarization feature seamlessly across different devices. The project showcases the integration of various technologies to

deliver a user-friendly experience. Further enhancements and optimizations can be made to enhance the functionality and performance of the system.