WEB-BASED IMAGE-AUGMENTED REALITY (AR) MATCHING GENERATOR

BY

BILL HUNDSON DAVID

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONOURS) INFORMATION SYSTEMS ENGINEERING

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2023

WEB-BASED IMAGE-AUGMENTED REALITY (AR) MATCHING GENERATOR

BY

BILL HUNDSON DAVID

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONOURS) INFORMATION SYSTEMS ENGINEERING

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2023

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: WEB-BASED IMAGE-AUGMENTED REALITY (AR) MATCHING GENERATOR

Academic Session: MAY 2023

I BILL HUNDSON DAVID

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

[illegible] [illegible]

(Author's signature) (Supervisor's signature)

Address:
Kampung Tapaang,
Peti Surat 45,
89320 Telupid, Sabah.

Syed Muhammad Bin Syed Omar
Supervisor's name

Date: 15/9/2023 **Date**: 15/09/2023

Universiti Tunku Abdul Rahman			
Form Title : **Sample of Submission Sheet for FYP/Dissertation/Thesis**			
Form Number: **FM-IAD-004**	Rev No.: **0**	Effective Date: **21 JUNE 2011**	Page No.: **1 of 1**

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 15/9/2023

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that ***Bill Hundson David*** (ID No: ***18ACB01007***) has completed this final year project entitled "Web-Based Image-Augmented Reality (AR) Matching Generator" under the supervision of Syed Muhammad Bin Syed Omar (Supervisor) from the Department of Information Systems, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

Bill

(*Bill Hundson David*)

DECLARATION OF ORIGINALITY

I declare that this report entitled "**WEB-BASED IMAGE-AUGMENTED REALITY (AR) MATCHING GENERATOR**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature	:	[illegible]
Name	:	Bill Hundson David
Date	:	15/9/2023

ACKNOWLEDGEMENTS

I would like to express my sincerest gratitude and appreciation to my supervisor and academic advisor, Mr. Syed Muhammad Bin Syed Omar who has provided me invaluable guidance throughout working on this project and throughout my years of studies at UTAR as a whole. This project has been a source of vital hands-on experience for developing my first practical application project. The lessons that I have gathered throughout the whole duration on developing the system for this project are all immensely invaluable.

I would also like to extend my most heartfelt thanks to my entire family for the overwhelming support that they have continuously provided me throughout my entire time studying at UTAR. I especially would like to acknowledge the help that has been given by my younger sister who has been the assistant in testing out the project system’s operations for the System Implementation section of this report. Thank you, Elysia Jane. Most important of all, I wholeheartedly express my unending gratitude for my mother, Fillista Jagnal, who has been the backbone of my family for years and the one who has been financing my studies at UTAR. All my achievements at this university were only possible thanks to her. With that, I therefore dedicate this Final Year Project as well as my Bachelor’s Degree to her. Thank you, mommy.

Lastly, I would also like to remember my late father, David Siani. Right from the start, he had always pushed me to continuously further my studies and climb the education ladder as high as possible, right until the end of his lifetime. “*Belajarlah ya, anak*” were his final words to me. I dedicate this work in loving memory of my father. May he rest in peace.

ABSTRACT

This is a project paper on the development for a cross-platform augmented reality application. To achieve its cross-platform nature, the completed system will be developed as a web-based application. In addition to using standard JavaScript, the project will also be incorporating React.js and Node.js to develop the system. In a nutshell, the project aims to deliver a system for video conferencing albeit with enhanced functionalities through exploiting the potentials of AR technology. Multiple video streams of users will be processed in real-time to extract the prominent human subject in a given video scene. The extracted video image of multiple human subjects will then be rendered and displayed within a single video scene. This is the process of augmenting user video images onto a virtual environment. The virtual environment, also referred to as the background element of the rendered output video, will be customizable to a certain degree. This system offers a unique software solution to deliver an AR experience with many creative implications. The completed system will incorporate the Selfie Segmentation Model developed by MediaPipe to perform the segmentation process in order to separate the foreground and background elements of a given video stream. The foreground subject refers to the human subject in a video. The completed system will allow manipulation of the background element. Other than that, the system will also utilize WebRTC technology to facilitate the video communication aspect between users of the system. The successfully developed system will be deployed to the web after which it will be publicly accessible through a given URL.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
AR	Augmented Reality
CSCW	Computer Supported Collaborative Work
FYP	Final Year Project
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ICE	Interactive Connectivity Establishment
ID	Identification
IDE	Integrated Development Environment
NAT	Network Address Translation
NPM	Node Package Manager
PC	Personal Computer
SDP	Session Description Protocol
STUN	Session Traversal Utilities for NAT
TURN	Traversal Using Relay NAT
UI	User Interface
URL	Uniform Resource Locator
UUID	Universal Unique Identifier
WebRTC	Web Real-Time Communication

Chapter 1
Introduction

1.1 Problem Statement and Motivation

Projecting a foreground image onto a desired background image is a technology that has existed for many years. One of the most well-known examples of this technique is chroma keying or better familiarized as using green screens. Chroma keying refers to a process where specific elements of a particular color in a video or image is replaced with a different element. This process dictates how green screens are used to segregate a foreground subject, typically a person, and virtually replace the background environment. This technology has its use in various context, popularly in video and image productions.

A more modern example is the virtual background feature used for video conferencing systems such as those in Microsoft Teams and Zoom. This feature enables a virtual background to be projected behind the user, enriching the video meeting experience. However, this feature is intended to be used solely within the specific applications themselves. These are only among the more well-known scenarios in which the user image-background matching technology is practiced. There are various more potential ways this technology can be used.

The major constrain to current system implementations is that users a limited to using specific systems intended for use only within the intended application use case. Furthermore, most current implementations are through operating system dependent applications while also requiring the devices to install the specific software application for access to their system. This may be an issue when AR software providers decide to develop the application native to specific platforms only. For example, Instagram’s AR camera feature is only accessible to users on its social media platform. Therefore, there is an opportunity to develop more cross-platform AR applications. Thanks to the rapid popularity of video conferencing systems due to the recent Covid-19 pandemic, developers have come up with new API tools and algorithms which finally enables development of web-based peer-to-peer video calling as well as AR applications.

1.2 Objectives

The end-goal of this project foresees the successful development and deployment of a fully functional web-based augmented reality application that would allow multi-user video image matching to dynamic background image. There are several objectives tied to the development and integration of key software components necessary for building the system. These components will be developed using the specified development frameworks and libraries which will be most suitable for developing modern web applications.

i. To develop a peer-to-peer video streaming module.

The functionalities of the application fundamentally rely on the capability for remote peer-to-peer video communication. Therefore, a basic module to allow video communication between remote users will be developed based on WebRTC technology. Development will also utilize a popular open-source JavaScript library which is React.js alongside standard JavaScript to build the front-end of the application. Furthermore, Node.js which is an open-source cross-platform server environment together with the installation of required dependency modules will be used to develop the back-end. The application should allow an initial user to host a call and generate a unique call ID which can be shared to other users. A secondary user will then be able to join in to the call session by providing the given call ID. This will then establish peer-to-peer remote communication allowing transmission of video stream between the users.

ii. To develop a module for video capture and perform image segmentation.

A module will be developed to allow access to a device's camera and capture a video stream of the user. An algorithm will then attempt to identify the user's image in the video stream and perform segmentation to separate between the foreground (human subject) and background. Development for this module will utilize the Selfie Segmentation Model developed by MediaPipe. This is a machine learning model which enables segmentation of the portrait of a person in a still or video image and will enable manipulation of the person's background.

iii. To integrate video streaming and image segmentation modules.

The peer-to-peer video streaming module will be further developed to work with the image segmentation module. The application should identify the incoming video stream of the call session participants. Segmentation is done on each video stream to separate between the human subject identified within the video frame and their corresponding background. The application will then be able to draw the segmented video images of a human subject from one user's video stream onto another user's video stream in real-time. For example, the application can have the segmented user video image of the call host projected on top of the peer call participant's video stream. This will make it look as if both users are in the same physical space.

iv. To deploy the augmented reality matching application to the web.

The fully integrated and working application will be deployed to the web so that it may be publicly accessible to users using a web browser. The application will be split into client and server ends, each of which will be hosted on separate web service hosting platforms. The client side will be hosted through GitHub Pages which is suitable for hosting static sites such as the front-end of the project application. Furthermore, the server side will be hosted using Render, a cloud-based application hosting platform. The project will utilize the free-tier service that Render offers which provides generous usage limitations suitable for projects of this scale.

v. To develop further enhancements for application functionalities.

Once the base application is successfully deployed, further development will be done as necessary to bring added functionality and further enhancements to the application. For instance, the application will provide the functionality to take a screenshot of the generated output video. The application will also allow swapping between having either the one of the user's videos to be the background element with the other user to be augmented onto it. Moreover, the application would have a custom background mode option that would allow the user to upload an image file that the application will then use to set as the background element.

1.3 Project Scope

The scope of this project involves the development of a cross-platform user to background image-augmented reality matching application. The application will be web-based and thus will be developed using the appropriate tools, technologies, and practices for building web-based applications. The system will be developed mainly utilizing Node.js and React.js, a component-based open-source JavaScript library alongside standard JavaScript. Furthermore, the project will also incorporate WebRTC technology which will allow two remote users of the system to establish video communications over the internet. Fundamentally, the target system will be a hybrid between a video calling application and an augmented reality program. Once developed, the application will be deployed to the web which will then be publicly accessible to users by running the application on a web browser through a given URL.

The application program will be made to capture a video image of the user after which it will perform segmentation to separate between the user's image and background element. The application will then 'augment' the user's segmented video image onto the other user's video, making them appear as if they are physically together in the same environment. Additionally, the application will provide functionalities for customization options to allow for a dynamic background. Users are given different operational options for the final video output such as taking screenshots and using a custom background image. The application is aimed to support two users each on different devices collaborating within the same video calling session.

1.4 Contributions

As a justification for working on this project, the completed system seeks to address the limitations of modern AR applications. Firstly, most features of user image-background matching are embedded within an application. For example, the virtual background feature in video conferencing applications like Microsoft Teams and Zoom are intended to enhance the video calling experience. The AR camera feature in social media applications such as Facebook, Instagram, and Snapchat are intended for social postings or for entertainment purposes. Thus, all these AR features are made available solely to improve the capabilities of the individual applications themselves. This application will help fulfil the need for a user-background matching tool that provides more general usage. It may be used for leisure activities such as taking a photo together with a remote user or it may also be practical for more formal purposes like capturing photos for e-graduation ceremonies for university students. Furthermore, development of AR tools today mostly leans towards building native applications for specific platforms. This may pose problems where an application is exclusively available for only one platform, or that performance for the same application might differ on different platforms. The choice to create this tool as a web-based application means it will be available for use across all platforms, fulfilling the need for a cross-platform solution. Other than that, this solution discards the need to use physical components such as specialized AR goggles or marker objects to perform AR as well as making it functional wherever there is network access.

1.5 Report Organization

This FYP II report consists of 7 chapters. Chapter 1 provides an introduction to the entire project. It discusses the project background including problem statements to address and motivations for engaging with this project. It also includes the project scope which discusses what aspects are and are not covered, objectives to be met for the project to succeed, and the potential impact or significance that the project hopes to bring. Chapter 2 includes the literature review done and comparison between previous works to the completed system of this project. Chapter 3 presents the development methodology, tools, and design for the completed system including system architecture, system use cases, and system activity design models. Furthermore, it also contains the timeline schedule applied in working on the project. Chapter 4 is the section which will describe in detail how the complete system is developed in its entirety and will provide all the information necessary for rebuilding the system. Additionally, the section will also include system design diagrams which shows the major system components, each of their specifications, and how each component interacts with one another to conduct system operations. Next is Chapter 5 which is closely tied to the contents in the previous chapter. This section contains process descriptions for setting up and configuring the system to have it deployed to the web. It also lists the hardware setup used for developing the system and software setup which includes the software programs installed as a prerequisite prior to starting the development work as well as all the Node package dependencies that the complete system relies on. Moreover, the section also shows images taken of the system operation in action and discusses issues and challenges faced during system implementation. In Chapter 6, an evaluation is done to test the system's compatibility to run on different web browsers, major project challenges that were faced, and the overall evaluation on project objectives that were achieved. Lastly is Chapter 7 which includes a concluding statement and some words on recommendations for future readers of this paper to rebuild the system.

Chapter 2
Literature Review

2.1 Review of the Existing Systems/Applications

2.1.1 Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System

This paper describes the works for an augmented reality conferencing system [1]. The system was developed as a tool to support CSCW (Computer Supported Collaborative Work), where a group of remotely located individuals work together on a project in real-time with the assistance of software tools and related technologies. As written [1], the example given to illustrate the system involves two groups of users: the AR user which will be the remote field worker and desktop computer users who are also referred to as desk bound experts. The AR user is equipped with an optical see-through head mounted display and a camera, with the desktop computers being mounted on with cameras as well. The head mounted display worn by the AR user is used to view the video image of desktop computer users which are captured through the camera mounted on their computer. Similarly, the desktop computer users will be able to view video image captured by the AR user's camera.

Other sets of components involved is the use of small marked cards and a larger piece of paper [1]. Each small marked card bears the identification for each desktop computer user and serves as a placeholder for where virtual monitors will show up on when seen through the AR interface. Each virtual monitor will display the camera image of the corresponding desktop computer user according to the identification detected on each marked card. Lastly, the larger piece of paper will be used to provide the interface for the virtual shared white board where the users can utilize to do collaborative work on. Figure 2.1.1 is an image taken from the article showcasing the desktop computer user's video image displayed on the virtual monitors.

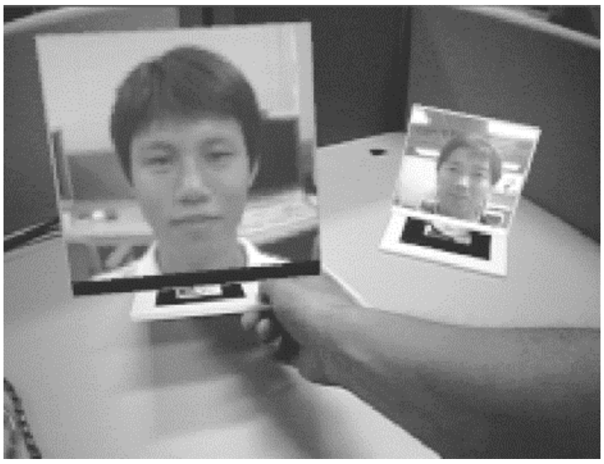


Figure 2.1.1 Virtual monitors displayed on the AR interface

2.1.2 The Effect of Spatial Cues in Augmented Reality Video Conferencing

This paper describes a continuation of the work in [1]. Written in this paper is the introduction of a set of new features added in as an upgrade to the previous system [1]. As described in [2], the authors acknowledge a few weaknesses of the augmented reality conference system in [1]. One key upgrade to the system which is of interest to the research for this paper is the elimination of the background scene for desktop computer users. It is mentioned how the existence of the background alongside the user image in the virtual monitors causes a sense of "discontinuity" from the real world [2]. Thus, the removal of the background in the virtual monitor was done leaving only the desktop user's image displayed on the virtual monitor. Figure 2.1.2 is an image from the article showcasing the result of background elimination.

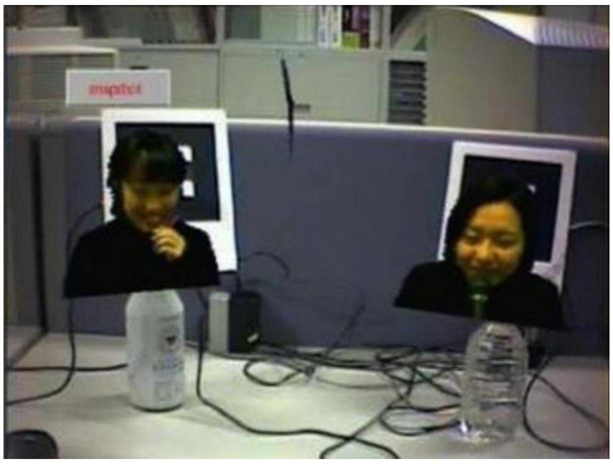


Figure 2.1.2 Elimination of background from user image

2.1.3 Visual Conditioning for Augmented-Reality-Assisted Video Conferencing

This work demonstrates an algorithm for a system that receives teleconferencing video and processes it into the desired output which is called as visual conditioning [3]. This system is intended for use in AR assisted video conferencing. Initially, the system first receives a raw video teleconferencing image. The system will attempt to identify the foreground and background elements in the video. Then, the boundary pixels of the foreground are captured, wherein a drawing of foreground and background boundaries is then derived. A drawing area for the background is then determined through the newly drawn boundaries. The system will then proceed to add in the background. This can be done by selecting from sets of available backgrounds. Alternatively, a new background may be designed according to characteristics of the foreground such as its colors. Finally, the segmented foreground, drawing area, and designed background are combined into a single composite image. Figure 2.1.3 is an image taken from the paper illustrating the visual conditioning process.

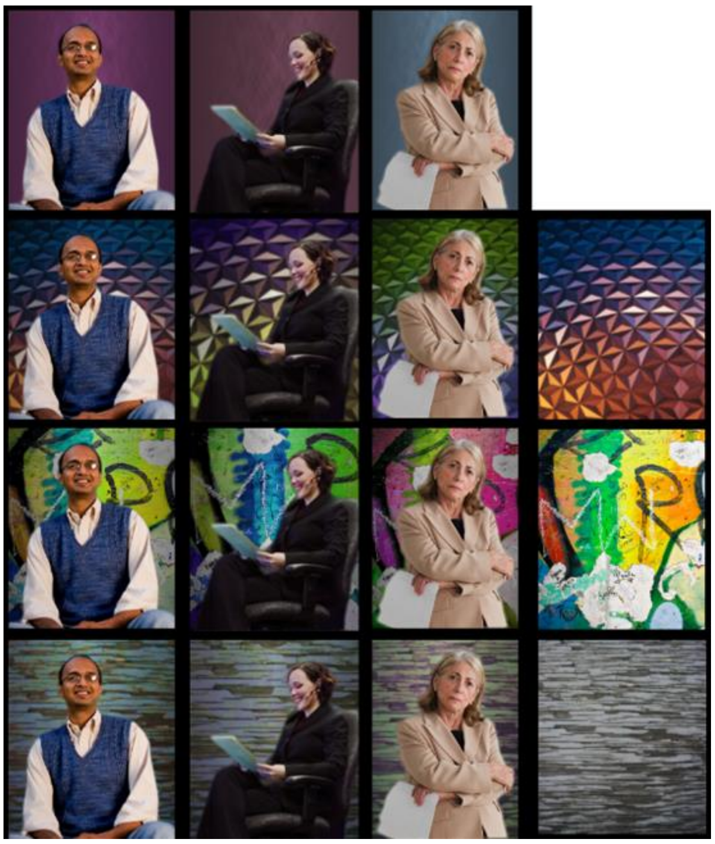


Figure 2.1.3 Visual Conditioning output

2.1.4 Virtual Presence Via Mobile

This work discusses is a patent for a mobile video conferencing system [4]. In this system, the video image of a user is first captured using the camera on their mobile device. This video is then sent over to a server via cellular network for processing. The user initially has the option to retain the original background in the video or replace it with a new one. If the user chooses to replace the background, the server will firstly extract the foreground of the video to create a new output, segmenting it from its background. The new output will have the background replaced with a new background of the user's choice. The final video output will then be transmitted to the receiving user's mobile device for viewing. Figures 2.1.4.1 and 2.1.4.2 are images taken from this paper which illustrates background replacement for two user videos.

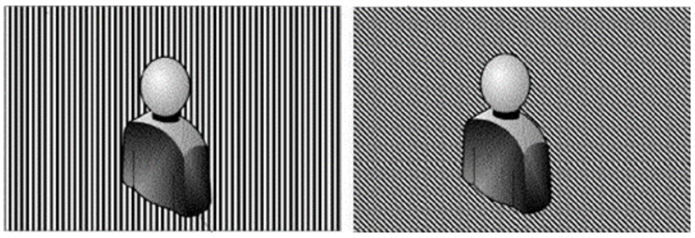


Figure 2.1.4 Two users with separate backgrounds

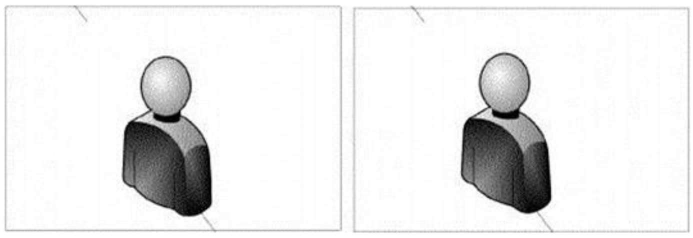


Figure 2.1.5 Two users matched to the same background

2.2 Critical Remarks of Previous Works

2.2.1 Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System

Strength – Uses physical markers to display peer users video streams on virtual monitors. Allows for multiple user videos to appear within same video frame, creating the feeling as if peer users are in the same physical room space.

Weakness – No actual segmentation is done on user video streams to separate between the human subject and background. Human subjects are displayed on top of two layers of background image resulting in them not blending in well together.

2.2.2 The Effect of Spatial Cues in Augmented Reality Video Conferencing

Strength – Background Subtraction Technique [2] is applied to user video streams with the users having a single-colored background image. This means segmentation being done to separate between the human subject and background. As a result, user video images are able to blend in much better with the artificial background.

Weakness – This method still relies on using physical markers to display peer user video streams where they can only be displayed on the virtual monitors, resulting in very limited flexibility on how the human subjects are displayed in the AR video output.

2.2.3 Visual Conditioning for Augmented-Reality-Assisted Video Conferencing

Strength – This work introduces a system to be used in video teleconferencing. The system has the advantage of being able to process multiple user video streams for human subject and background segmentation. Additionally, the system is also able to analyze the foreground subject to dynamically create a suitable background.

Weakness – Although able to perform human subject segmentation on multiple video streams, the system does not blend them together into a single video. Each segmented human subjects are still displayed on separate video outputs with distinct backgrounds.

2.2.4 Virtual Presence Via Mobile

Strength – This work showcases an AR matching model similar to the ones used in many of the modern applications. Ability to run on mobile devices allows for higher usage flexibility as the system can be run by remote peers anywhere as long as the cellular network bandwidth supports it. Moreover, users also have full control on background selection meaning they may share the same background if they wish to.

Weakness – Similar to most modern applications, it is still only limited to augmenting human subjects within separate video outputs. It does not support augmenting users to appear within each other's videos or within the same virtual environment.

2.3 Summary of the Existing Systems

Feature	**System 2.2.1**	**System 2.2.2**	**System 2.2.3**	**System 2.2.4**	**Proposed Solution**
Segments human subject and background		✓	✓	✓	✓
Functions without physical marker			✓	✓	✓
Works on mobile devices				✓	✓
Full background customization				✓	✓
Augmentation of peer users within the same video/virtual environment					✓

Table 2.3.1 Features comparison of previous works with this project's system

Chapter 3

System Methodology/Approach

3.1 System Design Diagram

3.1.1 System Architecture Diagram

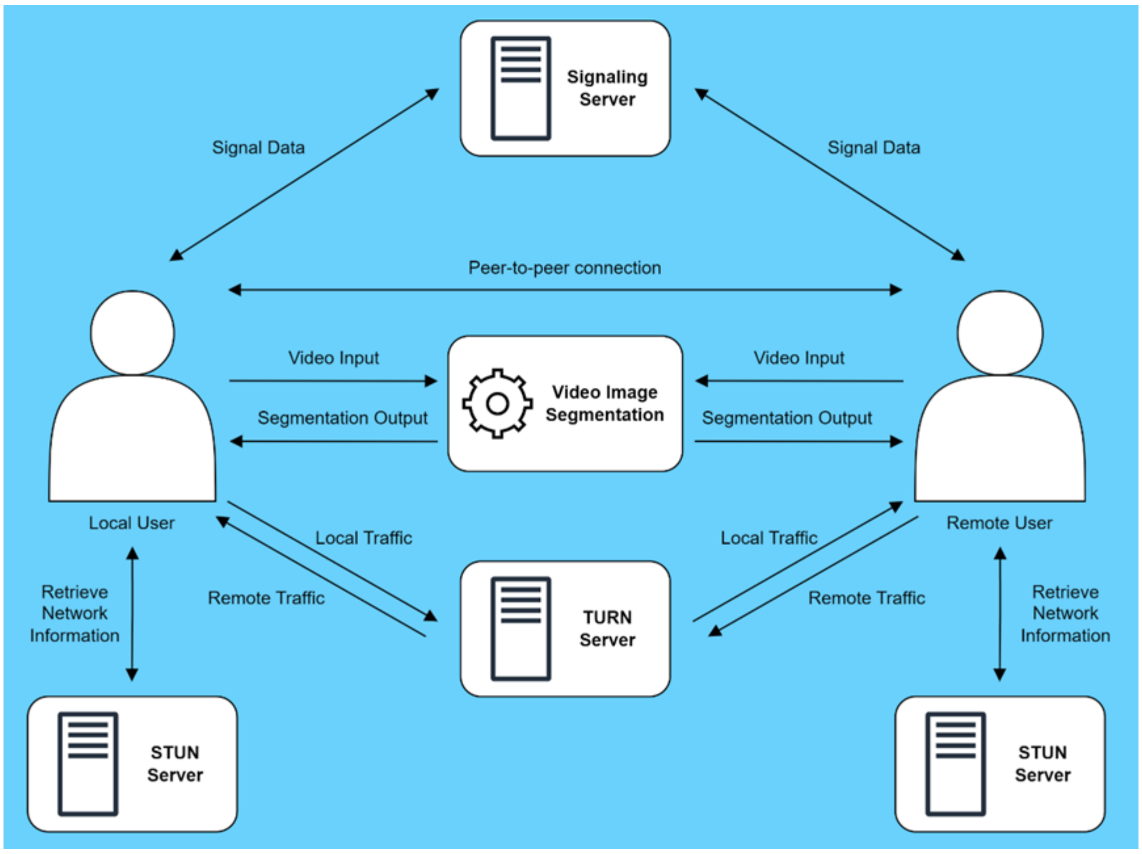


Figure 3.1.1 System architecture diagram

The above figure illustrates the architecture of the system to be built in this project. As a general overview, the system's architecture is built upon WebRTC technology for establishing peer-to-peer connection between two or more system users. Once the connection is established, peers will be able to transmit video streams to each other over the internet through their browser. While peers are connected, the system will capture each user's video stream and feed it to the video segmentation algorithm as input. The algorithm will output a generated AR video which will be displayed to the users. In order to establish the peer-to-peer connection, peer users will first need to exchange various necessary information.

First is the configuration data on media stream that will be transmitted which tells clients 'what' is to be communicated and the information regarding clients' network connection which tells 'how' they will communicate. The signaling server will be responsible for handling this initial data exchange. The signaling server acts as an intermediary to allow two remote peers discover each other on a network, conduct negotiations for a connection, coordinate and control connections, as well as terminate them if needed. The process begins when the local user or client contacts the signaling server by sending an SDP (Session Description Protocol) offer which contains configuration for media that the client intends to deliver. The signaling server passes this SDP information to the remote user or peer client which it intends to establish a connection with. The remote user will then store the SDP information it has received, replies back with its own SDP information known as an SDP answer for the signaling server to receive and pass it on back to the local user.

At this point, both clients have received each other's SDP information. The next step is to have the two clients exchange information regarding their respective network connections. This is done by having the clients exchange their ICE (Interactive Connectivity Establishment) candidates. ICE candidates are essentially sets of public IP address and port that clients may utilize to connect with each other. ICE candidates can be retrieved by the client by contacting the STUN (Session Traversal Utilities for NAT) server which allow clients to discover their public IP address, port, and the NAT (Network Address Translation). The client will contact the STUN server and in turn the server replies with ICE candidates. Both clients will exchange this information with each other, again through the signaling server.

Once both clients have each other's SDP information and ICE candidates, they will then be able to establish a direct peer-to-peer connection. However, in the case that the STUN server fails, clients will not be able to retrieve their ICE candidates and instead will rely on using the TURN server to establish the connection. The TURN (Traversal Using Relay NAT) implies a protocol for relaying network traffic, acting as an intermediary to allow clients to transmit data to and from one another. This alternative method involves a client directly routing network traffic which includes the media stream to the TURN server and then have it forward to other peer clients. Although more direct, using the TURN server to establish connection is less preferable as it introduces additional latency, negatively affecting system performance.

3.1.2 Use Case Diagram and Description

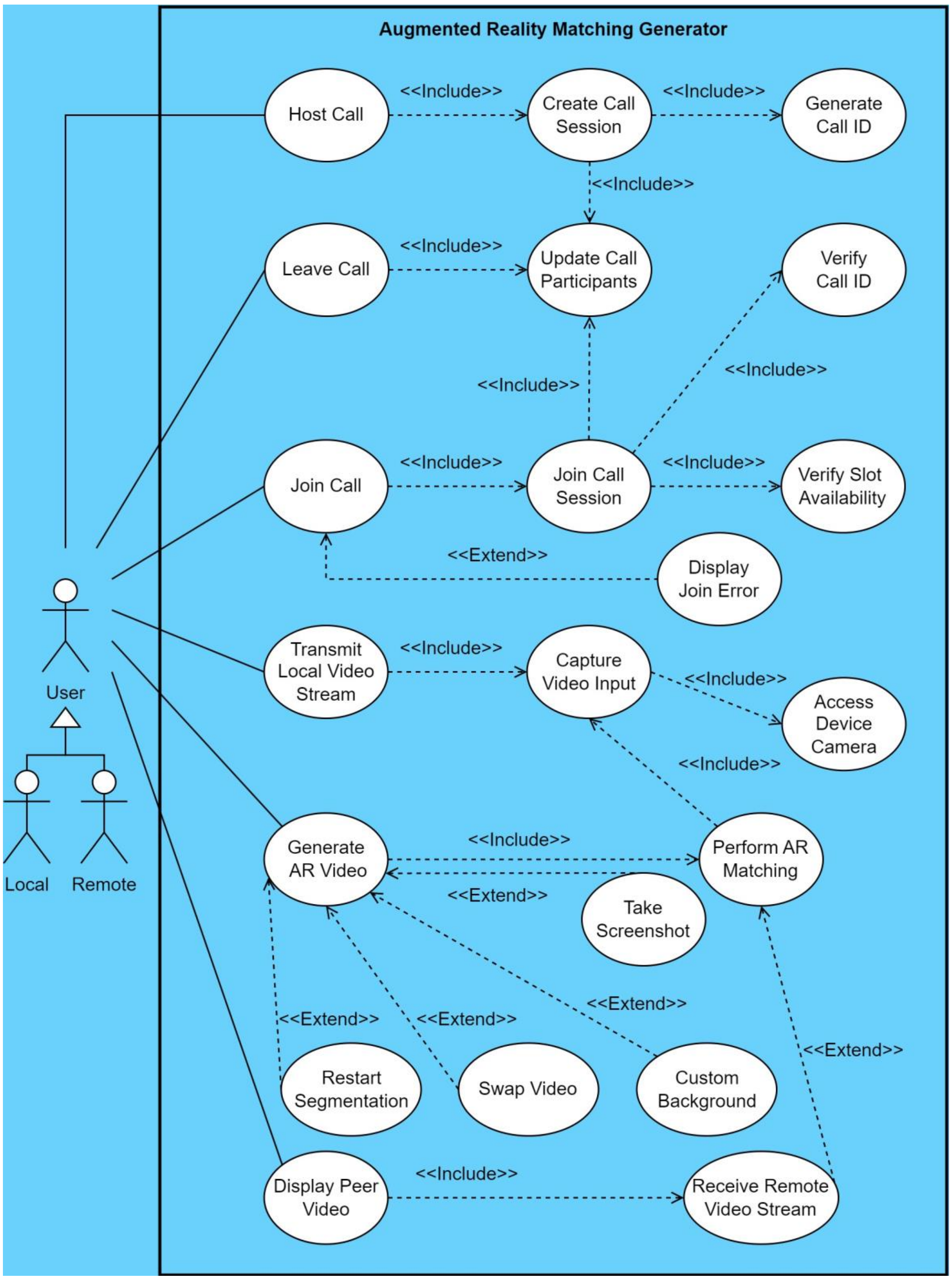


Figure 3.1.2 Use case diagram

Use Case: Host call	**Actor:** User
Purpose: For user to create a new call session and generate call ID.	
Trigger: User clicks on “host call” button.	
Normal Event Flow: 1. User clicks “host call” button and is redirected to a new page. 2. User inputs their name and clicks “host” button. 3. System contacts server to create a new call session with unique ID. 4. System creates new user and sets user as call host. 5. System adds user to call session’s participants list. 6. System redirects user to call session page.	
Alternate Event Flows: 2a. An error is displayed if user leaves the name field empty. 3a. An error is displayed if system is unable to contact the server.	

Table 3.1.1 Host Call use case description

Use Case: Join call	**Actor:** User
Purpose: For user to join an existing call and establish peer-to-peer connection.	
Trigger: User clicks on “join call” button.	
Normal Event Flow: 1. User clicks “join call” button and is redirected to a new page. 2. User inputs their name with call ID and clicks “join” button. 3. System contacts server to verify if call session exists and is not full. 4. System creates new user and updates call session's participants list. 5. System redirects user to call session page.	
Alternate Event Flows: 2a. An error is displayed if user leaves either the name or call ID field empty. 3a. An error is displayed if system unable to contact the server. 3b. An error is displayed if call session with the given ID does not exist. 3c. An error message will display to notify user if a call session is full.	

Table 3.1.2 Join Call use case description

Use Case: Transmit local video stream	**Actor:** User
Purpose: For capturing user video and transmit stream to other peers.	
Trigger: When user is successfully redirected to a call session page.	
Normal Event Flow: 1. System requests device camera permission access. 2. User grants camera permission access. 3. System accesses video stream captured through the camera. 4. System displays local preview of user's video. 5. System transmits video stream to connected peers.	
Alternate Event Flows: 2a. An error is displayed if user denies permission access. 5a. System will not transmit video stream if current user is the only call participant.	

Table 3.1.3 Transmit Local Video Stream use case description

Use Case: Display peer video	**Actor:** User
Purpose: For receiving and displaying connected peers' video stream.	
Trigger: When user is successfully redirected to a call session page.	
Normal Event Flow: 1. System checks if other call participants exist. 2. System attempts to establish connection with other peers. 3. System retrieves incoming video stream from other peers. 4. System displays video preview from other call participants.	
Alternate Event Flows: 1a. System will not proceed with next events if current user is the only participant. 2a. An error will be displayed if unable to establish peer-to-peer connection.	

Table 3.1.4 Display Peer Video use case description

Use Case: Generate AR video	**Actor:** User
Purpose: For processing user video streams to generate AR video output.	
Trigger: When video stream from user is successfully captured and received.	
Normal Event Flow: 1. System captures local user video stream. 2. System checks if other call participants exist. 3. System reads local and remote video stream. 4. System performs segmentation and AR matching process on videos. 5. System displays processed AR video output.	
Alternate Event Flows: 3a. System will only read local video stream if current user is sole call participant.	

Table 3.1.5 Generate AR Video use case description

Use Case: Leave call	**Actor:** User
Purpose: To allow user disconnect from call session.	
Trigger: When user clicks “leave call” button.	
Normal Event Flow: 1. User clicks on “leave call” button. 2. System checks if other call participants exist. 3. System updates call session participants list. 4. Current user’s user object is destroyed. 5. System redirects user to application homepage.	
Alternate Event Flows: 3a. If current user is the last participant, system will instead destroy call session.	

Table 3.1.6 Leave Call use case description

3.1.3 Activity Diagram

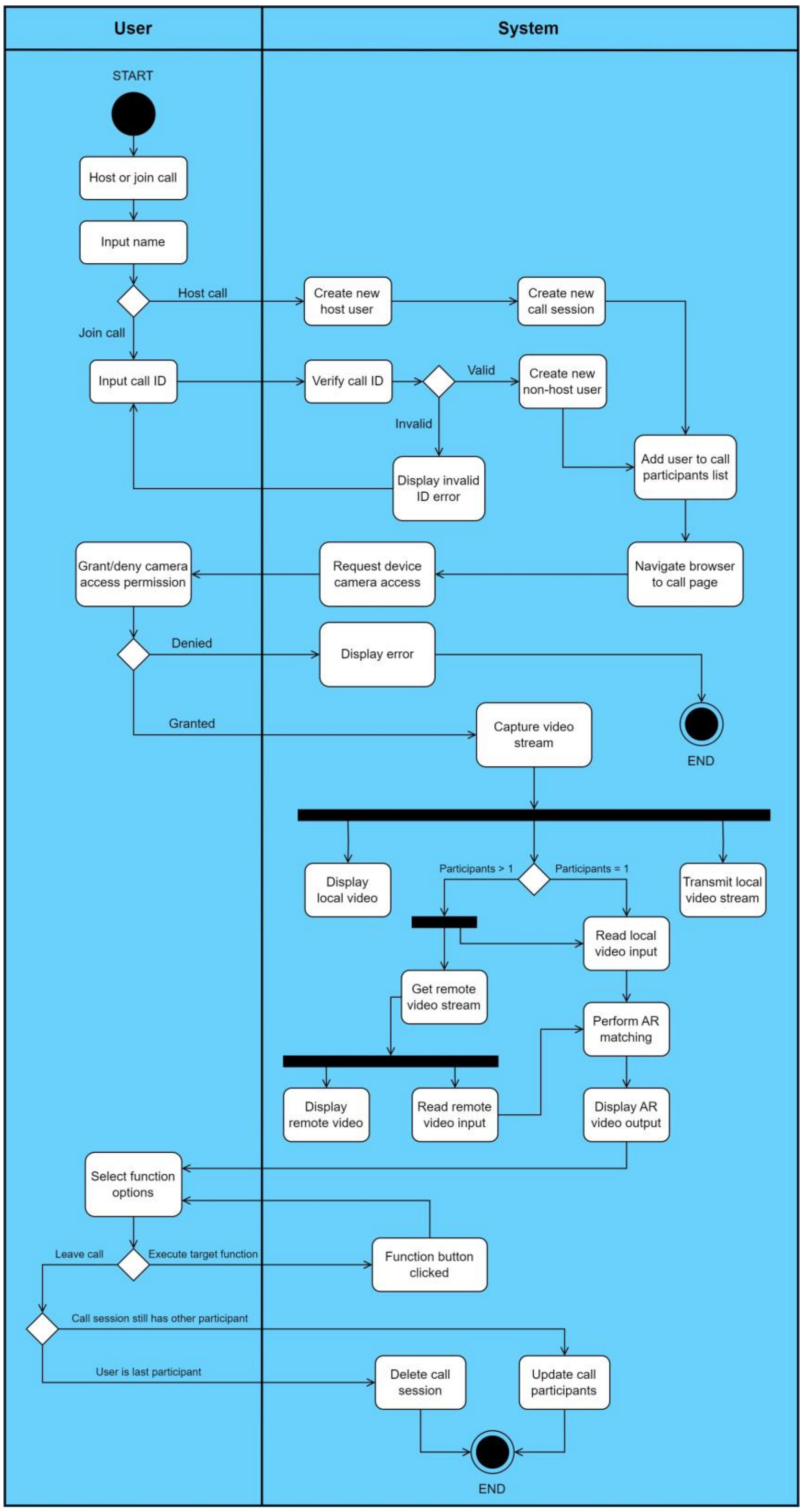


Figure 3.1.3 Activity diagram

Chapter 4

System Design

4.1 System Block Diagram

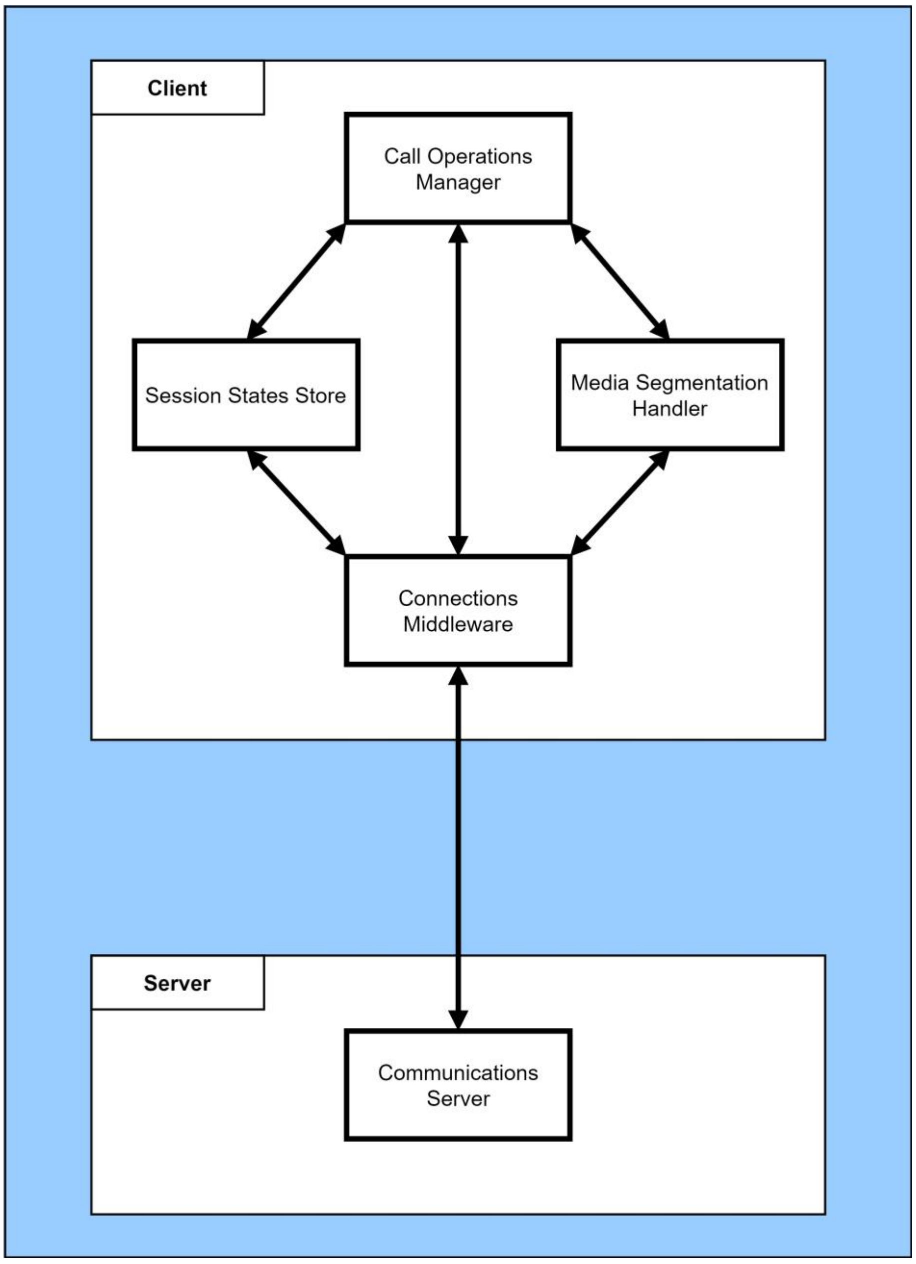


Figure 4.1.1 System Block Diagram

Figure 4.1.1 shows the top-down system block design which visualizes the interrelationship between the multiple components that comprises the system. The system components are separated into two main groups namely the client side and server side. As shown in the figure, the client components group consists of 4 system components while the server components group has a single system server component. Each component encapsulates varying sets of system application files housing functions, classes, and other data variables all interacting with one another to deliver each component's respective role. As a general overview, the Call Operations Manager interacts with the Session States Store and Media Segmentation Handler components to facilitate the major system functionality which is to establish and manage call sessions between the application clients as well as process incoming media stream to perform AR matching. To facilitate various operations between the remote application clients, these components interact through the Connections Middleware component which acts as an intermediary between the client components group and the Communications Server.

4.2 System Components Specifications

4.2.1 Call Operations Manager

This component makes up for the biggest bulk of the application's code structure. The main role of this component is to manage the overall operations and logic that are involved with creating and destroying call sessions as well as facilitating real-time communications between remote application users by connecting to and disconnecting them from the particular call session. Consequently, this component encapsulates the application codes for building most of the application's front-end including the overall application user interface design. The UI delivered through this component will provide the means for a user to host or join a call session, view incoming video stream from a peer user, as well as view and conduct a number of operations on the generated AR video output.

4.2.2 Session States Store

This component is essentially for storing and managing the application session states which are temporary data necessary to the system flow and logic. The session states the component should store includes the user's identity, call session ID, call host, call participants list, and loading overlay. These session state values are supplied to the other interacting components through accessing the given get and set functions for the respective session states.

4.2.3 Connections Middleware

This is the middleware component that will act as the medium to allow the application on the client machine to 'talk' with the remote server. This component will also synonymously be in the form of the signaling server. As previously explained, the signaling server enables two remote peers to discover each other on a network and coordinate real-time communications between them. The mechanism for which this component employs to facilitate communications is through message passing. When a component wishes to interact with the remote server, it does so by invoking one of the 'emit' functions provided by the signaling server. The target method will emit a specific message to the remote server and will wait for the server to respond. Upon receiving the response, it should communicate the information to other interacting components so that they would act accordingly.

4.2.4 Communications Server

This is the remote server component for the application which is also regarded as an extension to the signaling server. This component will be deployed as an independent Node.js application separate from the client-side program. This remote server will be hosted on a cloud hosting platform using Render. This component communicates with the client application through message passing which is initiated by the Connections Middleware. Upon receiving an emit message from the client, it will respond by emitting back the appropriate message. Among the various data that this server will send to the client may include the server port number, call session parameters, as well as calls to UI functions to name a few, all which are necessary to allow application flow and logic to synchronize between the remote host and peer clients.

4.2.5 Media Segmentation Handler

This is the component fundamentally responsible for processing incoming user video streams to produce the system's intended AR video output. Once a connection is established between the host and peer clients, this component will implement the MediaPipe Selfie Segmentation Model to run the segmentation process on the video source which will either be the client or peer's video. Aside from generating the AR-image matching, this component also encapsulates a number of functions that the user can trigger through interacting with the provided UI elements to control how the application should render the output. This includes functions to use a custom background and swap between using host or peer video background.

4.3 System Components Interaction Operations

This subchapter will provide more detailed illustrations of the individual processes and operations each system components carry out in order to interact with one other.

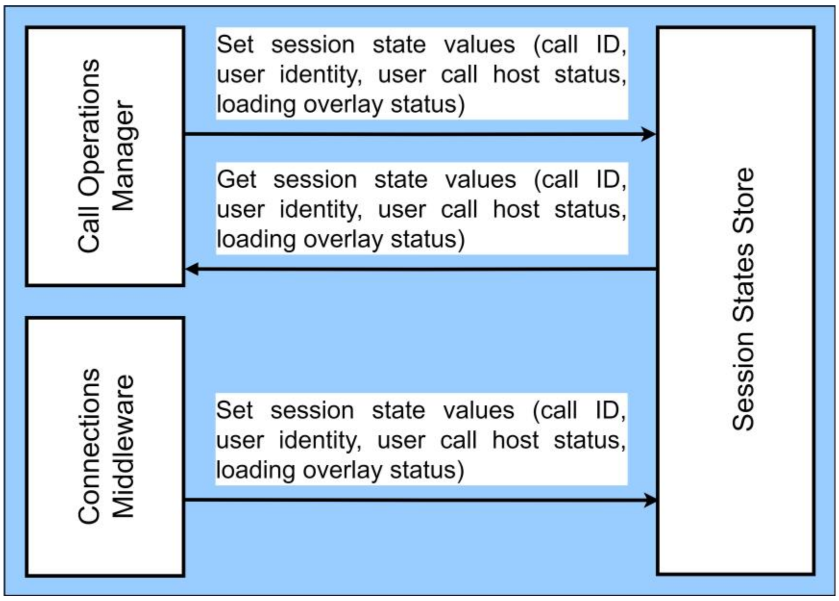

Call Operations Manager
Connections Middleware
Session States Store
Set session state values (call ID, user identity, user call host status, loading overlay status)
Get session state values (call ID, user identity, user call host status, loading overlay status)
Set session state values (call ID, user identity, user call host status, loading overlay status)

Figure 4.3.1 Interaction operations between the Call Operations Manager and Connections Middleware with the Session States Store system components

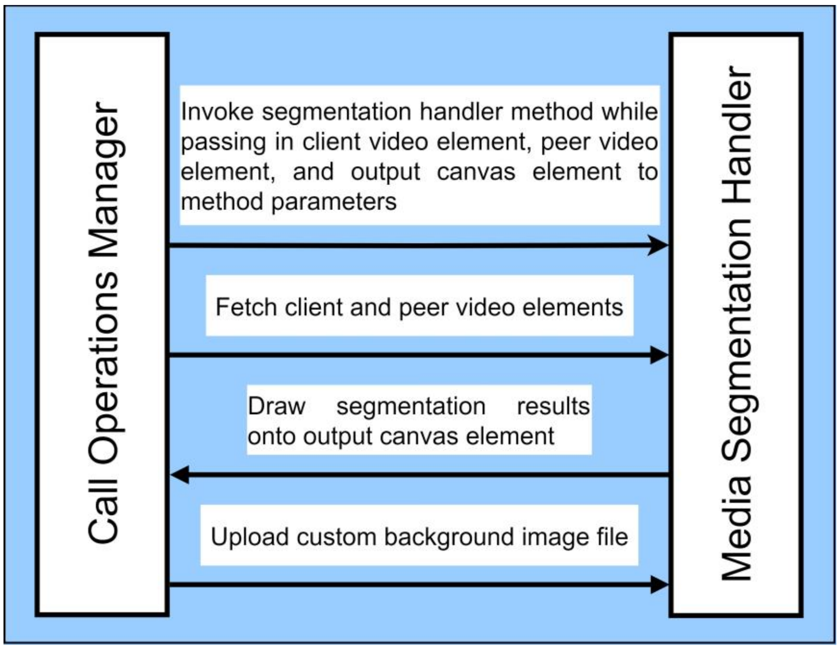

Call Operations Manager
Media Segmentation Handler
Invoke segmentation handler method while passing in client video element, peer video element, and output canvas element to method parameters
Fetch client and peer video elements
Draw segmentation results onto output canvas element
Upload custom background image file

Figure 4.3.2 Interaction operations between the Call Operation Manager and Media Segmentation Handler system components

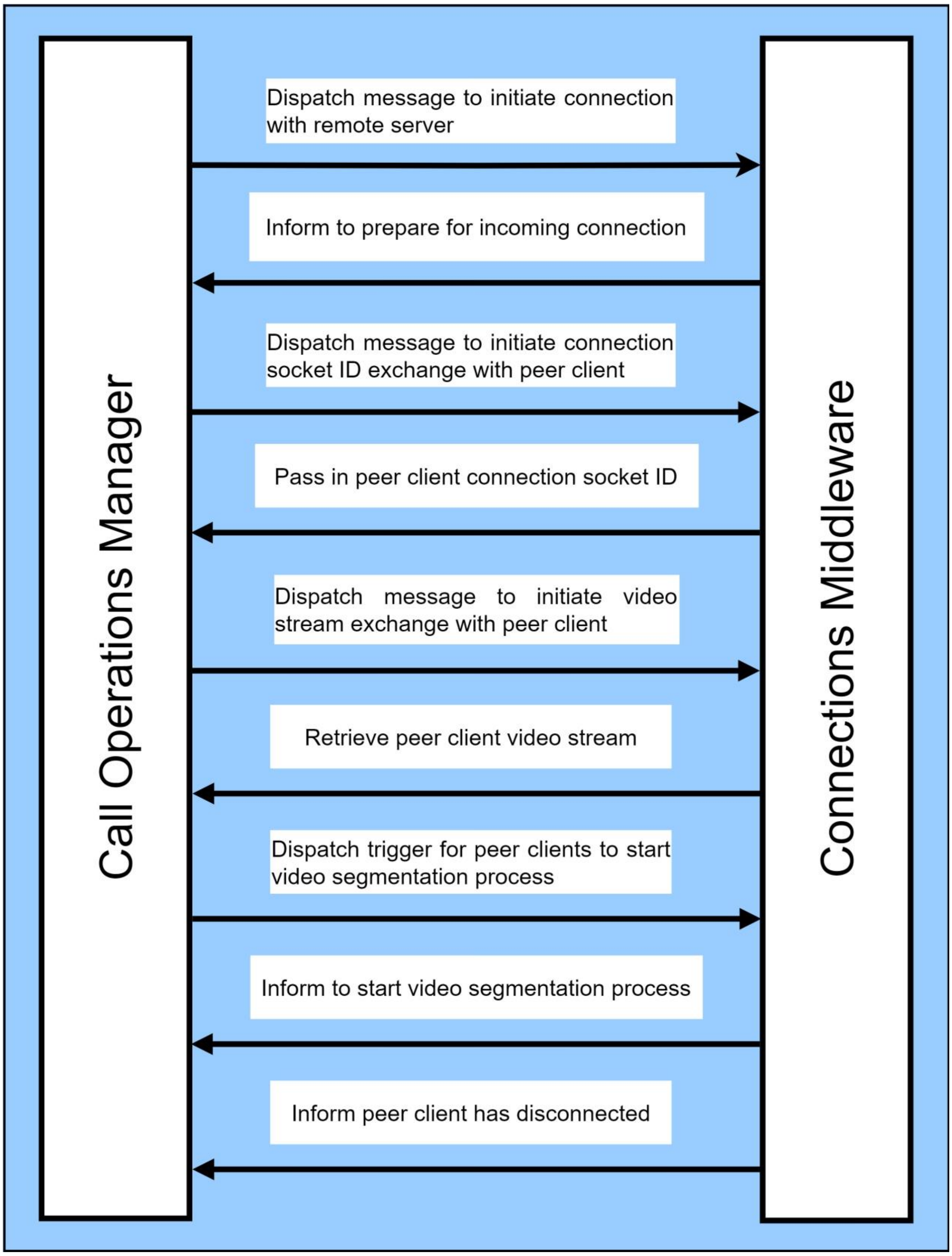

Call Operations Manager
Connections Middleware
Dispatch message to initiate connection with remote server
Inform to prepare for incoming connection
Dispatch message to initiate connection socket ID exchange with peer client
Pass in peer client connection socket ID
Dispatch message to initiate video stream exchange with peer client
Retrieve peer client video stream
Dispatch trigger for peer clients to start video segmentation process
Inform to start video segmentation process
Inform peer client has disconnected

Figure 4.3.3 Interaction operations between the Call Operations Manager and Connections Middleware system components

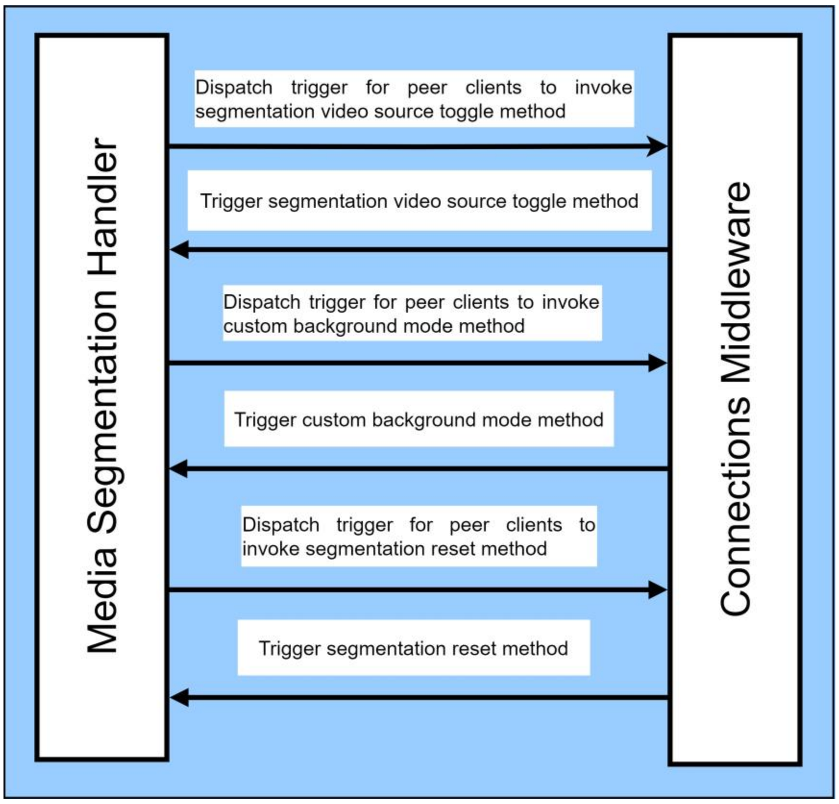

Media Segmentation Handler
Connections Middleware
Dispatch trigger for peer clients to invoke segmentation video source toggle method
Trigger segmentation video source toggle method
Dispatch trigger for peer clients to invoke custom background mode method
Trigger custom background mode method
Dispatch trigger for peer clients to invoke segmentation reset method
Trigger segmentation reset method

Figure 4.3.4 Interaction operations between the Media Segmentation Handler and Connections Middleware system components

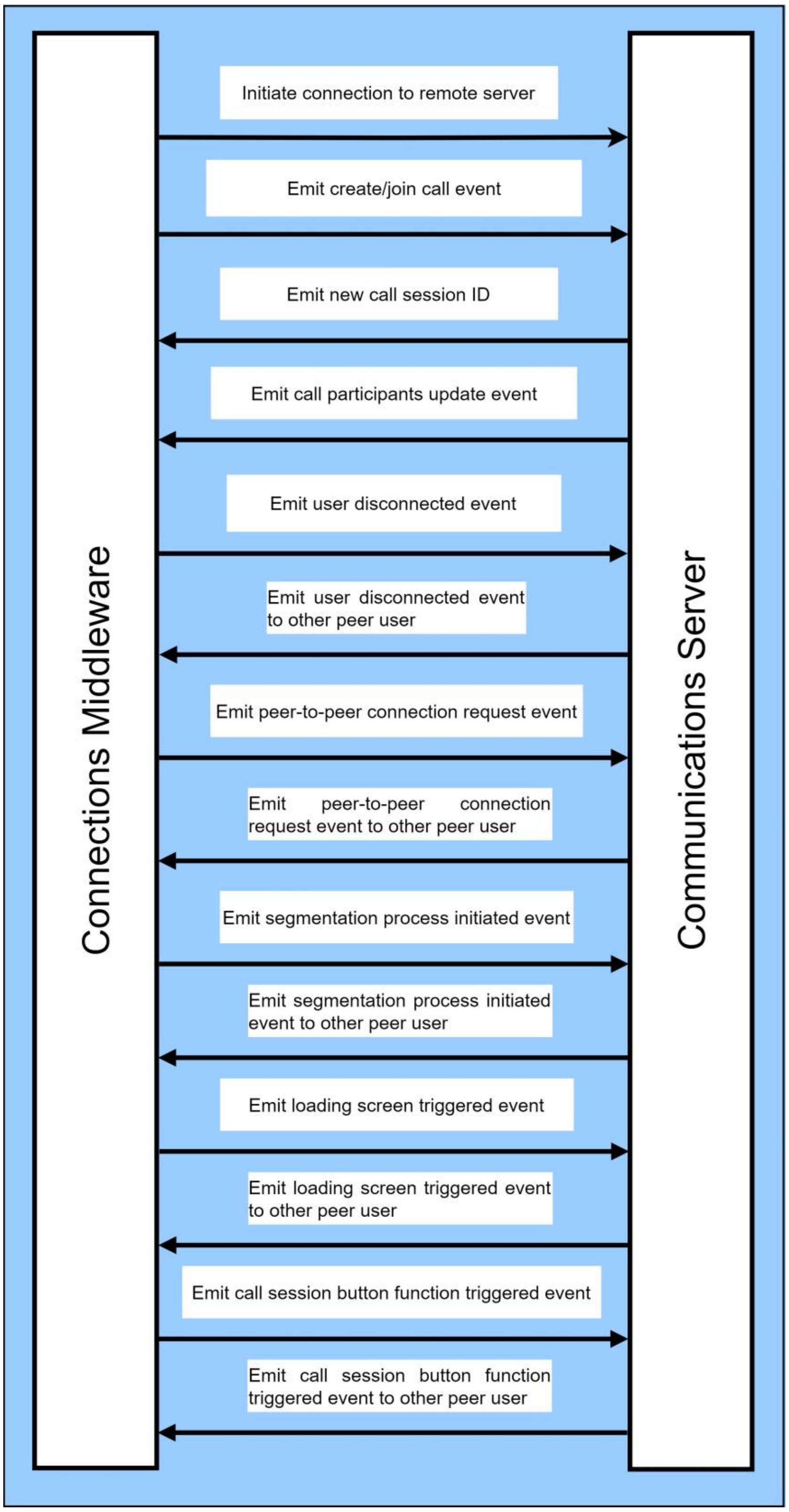

Connections Middleware
Communications Server
Initiate connection to remote server
Emit create/join call event
Emit new call session ID
Emit call participants update event
Emit user disconnected event
Emit user disconnected event to other peer user
Emit peer-to-peer connection request event
Emit peer-to-peer connection request event to other peer user
Emit segmentation process initiated event
Emit segmentation process initiated event to other peer user
Emit loading screen triggered event
Emit loading screen triggered event to other peer user
Emit call session button function triggered event
Emit call session button function triggered event to other peer user

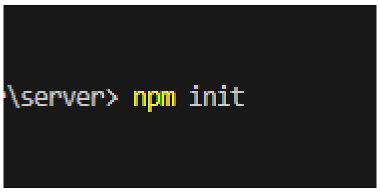
Figure 4.3.5 Interaction operations between the Connections Middleware and Communications Server system components

4.4 System Development

There are a number of prerequisite software components that will require setting up prior to starting the development work on the system. A complete list of these software components is written in Chapter 5.2 Software Setup. This subchapter will focus on showcasing the steps involved to develop the system from scratch.

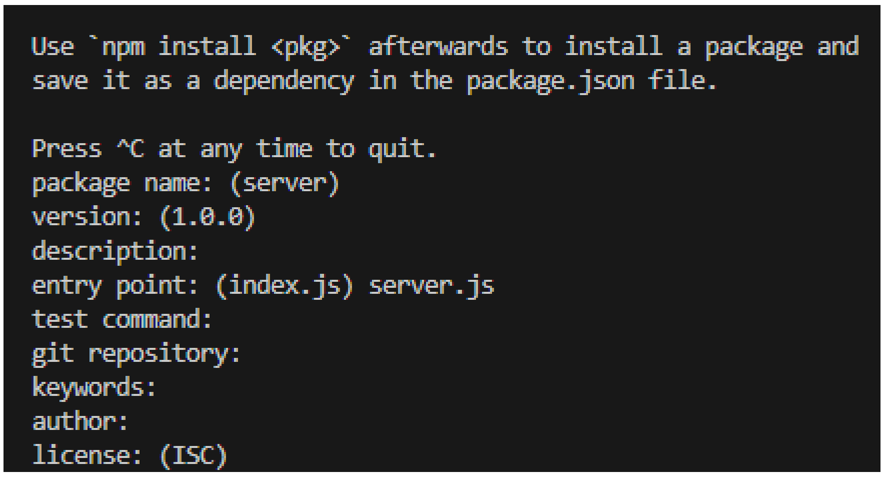
4.4.1 Creating the Remote Server Application

This section demonstrates the steps in creating the remote server for the Communications Server system component. To start, a new directory is created for the server. With the directory opened in the IDE, which in this case using VS Code, the command in Figure 4.4.1 is executed in the terminal to create a new .json package file. This file is for storing the installed package dependencies which the server program relies on. Additionally, the entry point field is set to *server.js* as seen in Figure 4.4.2. The other field values are just set to their defaults.



```
\server> npm init
```

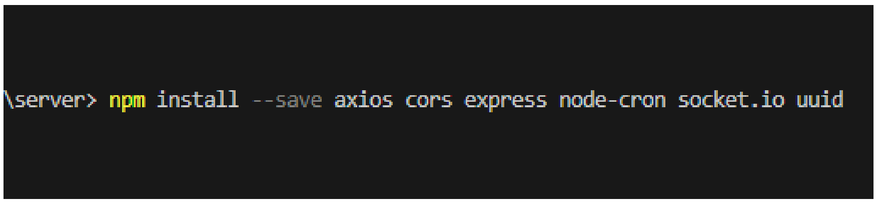
Figure 4.4.1 Terminal command create a .json package file



```
Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (server)
version: (1.0.0)
description:
entry point: (index.js) server.js
test command:
git repository:
keywords:
author:
license: (ISC)
```

Figure 4.4.2 Configuration for the .json package file

Next, the package dependencies are installed through the following command in Figure 4.4.3.



```
\server> npm install --save axios cors express node-cron socket.io uuid
```

Figure 4.4.3 Terminal command to install the required package dependencies

After the packages are installed, a JavaScript file named *server.js* is created. This file will be the main server node application file. Below are the codes written in the file.

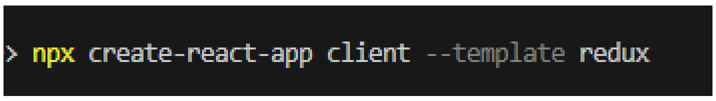
server.js

```
const express = require("express");
const http = require("http");
const { v4: uuidv4 } = require("uuid");
const cors = require("cors");
const cron = require("node-cron");
const { default: axios } = require("axios");

const PORT = process.env.PORT || 10000;
const app = express();
app.use(cors());
const server = http.createServer(app);

let connectedUsers = [];
let calls = [];

app.get("/api/call-exists/:callID", (req, res) => {
  const { callID } = req.params;
  const call = calls.find((call) => call.id === callID);

  if (call) {
    if (call.connectedUsers.length > 1) {
      return res.send({ callExists: true, callFull: true });
    } else {
      return res.send({ callExists: true, callFull: false });
    }
  } else {
    return res.send({ callExists: false });
  }
});

const io = require("socket.io")(server, {
  cors: {
    origin: "*",
    methods: ["GET", "POST"],
  },
  maxHttpBufferSize: 1e7,
});
```

```
io.on("connection", (socket) => {
  console.log(`User connected ${socket.id}`);

  socket.on("create-call", (data) => {
    createCallHandler(data, socket);
  });

  socket.on("join-call", (data) => {
    joinCallHandler(data, socket);
  });

  socket.on("disconnect", () => {
    disconnectHandler(socket);
  });

  socket.on("connection-signal", (data) => {
    signalingHandler(data, socket);
  });

  socket.on("connection-init", (data) => {
    connectionInitHandler(data, socket);
  });

  socket.on("ar-image-matcher", (data) => {
    const { tempCallID } = data;
    io.in(tempCallID).emit("ar-image-matcher");
  });

  socket.on("swap-video", (data) => {
    const { tempCallID } = data;
    io.in(tempCallID).emit("swap-video");
  });

  socket.on("reset", (data) => {
    const { tempCallID } = data;
    io.in(tempCallID).emit("reset");
  });

  socket.on("custom-background", (data) => {
    const { tempCallID, img } = data;
    const imgFile = { img };
    io.in(tempCallID).emit("custom-background", imgFile);
  });

  socket.on("end-call", (data) => {
    const { tempCallID } = data;
    io.in(tempCallID).emit("end-call");
  });

  socket.on("await-feed", (data) => {
    const { tempCallID } = data;
    io.in(tempCallID).emit("ar-matching");
  });

  socket.on("signal-loader", (data) => {
    const { tempCallID } = data;
    io.in(tempCallID).emit("signal-loader");
  });
});

const createCallHandler = (data, socket) => {
  const { identity } = data;
  const callID = uuidv4().substring(0, 4);
  const newUser = {
    identity,
    id: uuidv4(),
    socketID: socket.id,
    callID,
  };

  connectedUsers = [...connectedUsers, newUser];

  const newCall = {
    id: callID,
    connectedUsers: [newUser],
  };

  socket.join(callID);

  calls = [...calls, newCall];

  socket.emit("call-id", { callID });

  socket.emit("call-update", { connectedUsers: newCall.connectedUsers });
};

const joinCallHandler = (data, socket) => {
  const { identity, callID } = data;

  const newUser = {
    identity,
    id: uuidv4(),
    socketID: socket.id,
    callID,
  };

  const call = calls.find((call) => call.id === callID);
  call.connectedUsers = [...call.connectedUsers, newUser];

  socket.join(callID);

  connectedUsers = [...connectedUsers, newUser];

  call.connectedUsers.forEach((user) => {
    if (user.socketID !== socket.id) {
      const data = {
        connectedUserSocketID: socket.id,
      };

      io.to(user.socketID).emit("connection-prepare", data);
    }
  });

  io.to(callID).emit("call-update", { connectedUsers: call.connectedUsers });
};

const disconnectHandler = (socket) => {
  const user = connectedUsers.find((user) => user.socketID === socket.id);

  if (user) {
    const call = calls.find((call) => call.id === user.callID);

    call.connectedUsers = call.connectedUsers.filter(
      (user) => user.socketID !== socket.id
    );

    socket.leave(user.callID);
    if (call.connectedUsers.length > 0) {
      console.log("call.connectedUsers.length > 0");
      io.to(call.id).emit("user-disconnected", { socketID: socket.id });
      io.to(call.id).emit("end-call");
    } else {
      calls = calls.filter((c) => c.id !== call.id);
    }
  }
};

const signalingHandler = (data, socket) => {
  const { connectedUserSocketID, signal } = data;
  const signalingData = { signal, connectedUserSocketID: socket.id };
  io.to(connectedUserSocketID).emit("connection-signal", signalingData);
};

const connectionInitHandler = (data, socket) => {
  const { connectedUserSocketID } = data;
  const initData = { connectedUserSocketID: socket.id };
  io.to(connectedUserSocketID).emit("connection-init", initData);
};

if (server.listen(PORT, () => {})) {
  cron.schedule("0,15,30,45 * * * * *", () => {
    axios
      .get("https://image-ar-matcher.onrender.com")
      .then((response) => {
        response = null;
      })
      .catch((response) => {
        response = null;
      });
  });
}
```

4.4.2 Initial Setup of the Main Client Application

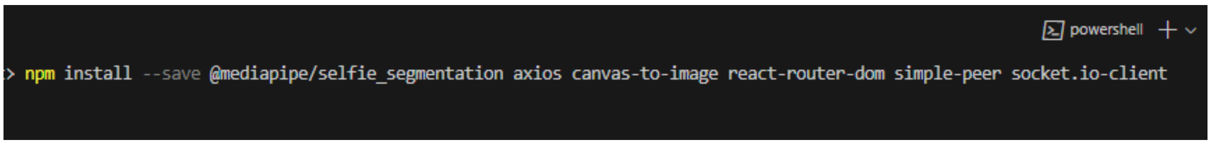
This section will demonstrate the initial setup for the main client-side application which makes up the major bulk of the system's program structure. To begin, a new directory is created where it will be opened in VS Code for us to create a new React.js application to. Once opened in VS Code, the following command in Figure 4.4.4 is executed in the terminal where *client* being the name of the application. This will create a new React.js application built with the Redux template which will be used in developing the Session States Store system component later.



```
> npx create-react-app client --template redux
```

Figure 4.4.4 Terminal command to create a React.js application using the Redux template

Inside the newly created React application directory, some cleaning up may be done to remove additional files that would not be essential to the system. These include *App.test.js*, *logo.svg*, *setupTests.js*, and the *features* directory along with all its content. Of course, updates should be made accordingly to any files that had references to those files removed. Additionally, the *app* folder that contains the *store.js* file should also be renamed to *store* for convenience. Lastly, the following command in Figure 4.4.5 is run to install the Node package dependencies for this client application. Details on each dependency is written further in Chapter 5.2.



```
> npm install --save @mediapipe/selfie_segmentation axios canvas-to-image react-router-dom simple-peer socket.io-client
```

Figure 4.4.5 Terminal command to install the required package dependencies

4.4.3 Creating the Redux Store

This section showcases configuring the Session States Store system component using the Redux state management library. Inside the *store* directory containing the *store.js* file, two new files are created namely *actions.js* and *reducer.js*. Below are the source codes for each file.

actions.js

```
const Actions = {
  SET_IS_CALL_HOST: "SET_IS_CALL_HOST",
  SET_IDENTITY: "SET_IDENTITY",
  SET_CALL_ID: "SET_CALL_ID",
  SET_SHOW_LOADING_OVERLAY: "SET_SHOW_LOADING_OVERLAY",
  SET_PARTICIPANTS: "SET_PARTICIPANTS",
};

export const setIsCallHost = (isCallHost) => {
  return {
    type: Actions.SET_IS_CALL_HOST,
    isCallHost,
  };
};

export const setIdentity = (identity) => {
  return {
    type: Actions.SET_IDENTITY,
    identity,
  };
};

export const setCallID = (callID) => {
  return {
    type: Actions.SET_CALL_ID,
    callID,
  };
};
```

```
export const setShowLoadingOverlay = (showLoadingOverlay) => {
  return {
    type: Actions.SET_SHOW_LOADING_OVERLAY,
    showLoadingOverlay,
  };
};

export const setParticipants = (participants) => {
  return {
    type: Actions.SET_PARTICIPANTS,
    participants,
  };
};

export default Actions;
```

reducer.js

```
import Actions from "./actions";

const initState = {
  identity: "",
  isCallHost: false,
  callID: null,
  showLoadingOverlay: true,
  participants: [],
};

const reducer = (state = initState, action) => {
  switch (action.type) {
    case Actions.SET_IS_CALL_HOST:
      return {
        ...state,
        isCallHost: action.isCallHost,
      };
    case Actions.SET_CALL_ID:
      return {
        ...state,
        callID: action.callID,
      };
    case Actions.SET_IDENTITY:
      return {
        ...state,
        identity: action.identity,
      };
    case Actions.SET_SHOW_LOADING_OVERLAY:
      return {
        ...state,
        showLoadingOverlay: action.showLoadingOverlay,
      };
    case Actions.SET_PARTICIPANTS:
      return {
        ...state,
        participants: action.participants,
      };
    default:
      return state;
  }
};

export default reducer;
```

store.js

```
import { configureStore } from "@reduxjs/toolkit";
import reducer from "./reducer";

export const store = configureStore({
  reducer: reducer,
});

export default store;
```

4.4.4 Creating the WebRTC Module

This section demonstrates the implementation of the WebRTC open framework that will provide the real-time communications capability for the system. Inside the *src* directory, a new directory named *utils* is created. In the new directory, a new JavaScript file is added named *api.js* of which the codes for are given below. Next, another file named *webRTCHandler.js* is also created. As its name implies, this file will be the main handler for the system's WebRTC module which accommodates various functions that can be called externally by the other system. This WebRTC handler primarily works in conjunction with the signaling server in order to establish real-time communications between remote clients. Calling the *getLocalVideoAndInitConnection* function will have the application to start capturing the client's local video stream and make a call to the signaling server to create or join a call session. To create the signaling server, a new file named *signalingServer.js* is added to the *utils* directory. The primary role of the signaling server is to facilitate message passing between this client application and the remote server. When the web application page is opened on a browser, it will immediately attempt to establish a handshake with the remote server which upon successful reception, a *socket* instance is instantiated which will be the point for the signaling server to make contact with the remote server. Fundamentally, all the signaling server does is to listen for and emit events to and from the remote server. The functions for the event listeners are denoted as *socket.on* while the event emitters are implied by the *socket.emit* functions. Event emitting is done to request for certain data from the remote server or for sending data that the server should utilize. Furthermore, the signaling server constantly runs event listeners to listen for responses emitted by the remote server. Upon reception of an emitted event, the signaling server would optionally assign any data object received to a new variable instance and execute the appropriate actions that the event listener has been specified to do. The coding for *webRTCHandler.js* and *signalingServer.js* are as given in the next pages.

api.js

```
import axios from "axios";

const serverAPI = "https://image-ar-matcher.onrender.com/api";

export const getCallExists = async (callID) => {
  const response = await axios.get(`${serverAPI}/call-exists/${callID}`);
  return response.data;
};
```

webRTCHandler.js

```
import { setShowLoadingOverlay } from "../store/actions";
import store from "../store/store";
import * as signalingServer from "./signalingServer";
import Peer from "simple-peer";
import SegmentationHandler from "./segmentationHandler";
import { showUtilityButtons } from "../pages/call/videoSection/VideoButtons";

const defaultConstraints = {
  audio: false,
  video: {
    width: "1920",
    height: "1080",
  },
};

let localStream;
let isHost = false;
let tempConnectedUserSocketID;

export const getLocalVideoAndInitConnection = async (
  isCallHost,
  identity,
  callID = null
) => {
  navigator.mediaDevices
    .getUserMedia(defaultConstraints)
    .then((stream) => {
      if (isCallHost) {
        isHost = true;
      }

      localStream = stream;
      showLocalVideoPreview(localStream);

      isCallHost
        ? signalingServer.createCall(identity)
        : signalingServer.joinCall(identity, callID);
    })
    .catch((err) => {
      console.log("Failed to access webcam.");
      console.log(err);
    });
};

let peers = {};
let streams = [];
```

```
const getConfiguration = () => {
  return {
    iceServers: [
      {
        urls: "stun:a.relay.metered.ca:80",
      },
      {
        urls: "turn:a.relay.metered.ca:80",
        username: "1e845039e49ee8c52cf102ef",
        credential: "kIyLpMJOFcmNx5g/",
      },
      {
        urls: "turn:a.relay.metered.ca:80?transport=tcp",
        username: "1e845039e49ee8c52cf102ef",
        credential: "kIyLpMJOFcmNx5g/",
      },
      {
        urls: "turn:a.relay.metered.ca:443",
        username: "1e845039e49ee8c52cf102ef",
        credential: "kIyLpMJOFcmNx5g/",
      },
      {
        urls: "turn:a.relay.metered.ca:443?transport=tcp",
        username: "1e845039e49ee8c52cf102ef",
        credential: "kIyLpMJOFcmNx5g/",
      },
    ],
  };
};
```

The *getConfiguration* function returns TURN server credentials configurations for preparing peer connections. Acquiring these credentials is detailed in Chapter 5.3 Web Deployment.

```
export const prepareNewPeerConnection = (
  connectedUserSocketID,
  isInitiator
) => {
  const configuration = getConfiguration();

  peers[connectedUserSocketID] = new Peer({
    initiator: isInitiator,
    config: configuration,
    stream: localStream,
  });

  peers[connectedUserSocketID].on("signal", (data) => {
    const signalData = {
      signal: data,
      connectedUserSocketID: connectedUserSocketID,
    };

    signalingServer.signalPeerData(signalData);
  });

  peers[connectedUserSocketID].on("stream", (stream) => {
    addStream(stream, connectedUserSocketID);
    streams = [...streams, stream];
  });
};
```

```
export const signalDataHandler = (data) => {
  peers[data.connectedUserSocketID].signal(data.signal);
};

export const removePeerConnection = (data) => {
  const { socketID } = data;
  const videoContainer = document.getElementById(`${socketID}-video-container`);
  const videoElement = document.getElementById(`${socketID}-video`);

  if (videoContainer && videoElement) {
    const tracks = videoElement.srcObject.getTracks();

    tracks.forEach((t) => t.stop());

    videoElement.srcObject = null;
    videoContainer.removeChild(videoElement);
    videoContainer.parentNode.removeChild(videoContainer);

    if (peers[socketID]) {
      peers[socketID].destroy();
    }
    delete peers[socketID];
  }
};

const showLocalVideoPreview = (stream) => {
  const videosContainer = document.getElementById("videos_container");
  videosContainer.classList.add("videos_container");
  const videoContainer = document.createElement("div");
  videoContainer.id = "local-video-container";
  videoContainer.classList.add("video_track_container");

  const videoElement = document.createElement("video");
  videoElement.autoplay = true;
  videoElement.muted = true;
  videoElement.srcObject = stream;
  videoElement.id = "local-video";
  videoElement.classList.add("video");
  videoElement.onloadedmetadata = () => {
    videoElement.play();
  };

  videoContainer.appendChild(videoElement);
  videosContainer.appendChild(videoContainer);
};
```

```
const addStream = (stream, connectedUserSocketID) => {
  const videosContainer = document.getElementById("videos_container");
  const videoContainer = document.createElement("div");
  videoContainer.id = `${connectedUserSocketID}-video-container`;
  videoContainer.classList.add("video_track_container");

  const videoElement = document.createElement("video");
  videoElement.autoplay = true;
  videoElement.muted = true;
  videoElement.srcObject = stream;
  videoElement.id = `${connectedUserSocketID}-video`;
  videoElement.classList.add("video");
  videoElement.onloadedmetadata = () => {
    videoElement.play();
  };

  videoContainer.appendChild(videoElement);
  videosContainer.appendChild(videoContainer);

  tempConnectedUserSocketID = connectedUserSocketID;
  store.dispatch(setShowLoadingOverlay(false));
};

export const ARImageMatcher = () => {
  const videosContainer = document.getElementById("videos_container");
  const AROutputContainer = document.createElement("div");
  AROutputContainer.id = "ar-output-container";
  AROutputContainer.classList.add("ar_output_container");

  const canvasElement = document.createElement("canvas");
  canvasElement.id = "ar-output-canvas";
  canvasElement.classList.add("ar_output_canvas");
  canvasElement.width = 1920;
  canvasElement.height = 1080;

  let hostVideoElement, peerVideoElement;

  if (isHost) {
    hostVideoElement = document.getElementById("local-video");
    peerVideoElement = document.getElementById(
      `${tempConnectedUserSocketID}-video`
    );
  } else {
    hostVideoElement = document.getElementById(
      `${tempConnectedUserSocketID}-video`
    );
    peerVideoElement = document.getElementById("local-video");
  }

  AROutputContainer.appendChild(canvasElement);
  videosContainer.appendChild(AROutputContainer);

  showUtilityButtons();

  SegmentationHandler(
    hostVideoElement.id,
    peerVideoElement.id,
    canvasElement.id
  );
};

export const getIsHost = () => {
  return isHost;
};
```

signalingServer.js

```
import io from "socket.io-client";
import { setCallID, setParticipants } from "../store/actions";
import store from "../store/store";
import * as webRTCHandler from "./webRTCHandler";
import {
  customBackground,
  resetSegmentation,
  swapVideo,
} from "./segmentationHandler";
import { leaveCallHandler } from "../pages/call/videoSection/LeaveCallButton";
import { setShowLoadingOverlay } from "../store/actions";

const SERVER = "https://image-ar-matcher.onrender.com";

let socket = null;
let tempCallID;

export const connectWithSocketIOServer = () => {
  socket = io(SERVER);

  socket.on("connect", () => {
    console.log("Successfully connected with socket.io server");
    console.log(socket.id);
  });

  socket.on("call-id", (data) => {
    const { callID } = data;
    tempCallID = callID;
    store.dispatch(setCallID(callID));
  });

  socket.on("call-update", (data) => {
    const { connectedUsers } = data;
    store.dispatch(setParticipants(connectedUsers));
  });

  socket.on("connection-prepare", (data) => {
    const { connectedUserSocketID } = data;
    webRTCHandler.prepareNewPeerConnection(connectedUserSocketID, false);

    socket.emit("connection-init", {
      connectedUserSocketID: connectedUserSocketID,
    });
  });

  socket.on("connection-signal", (data) => {
    webRTCHandler.signalDataHandler(data);
  });

  socket.on("connection-init", (data) => {
    const { connectedUserSocketID } = data;
    webRTCHandler.prepareNewPeerConnection(connectedUserSocketID, true);
  });

  socket.on("user-disconnected", (data) => {
    webRTCHandler.removePeerConnection(data);
    leaveCallHandler();
  });
socket.on("ar-image-matcher", () => {
    webRTCHandler.ARImageMatcher();
  });

  socket.on("swap-video", () => {
    swapVideo();
    store.dispatch(setShowLoadingOverlay(false));
  });

  socket.on("reset", () => {
    resetSegmentation();
    store.dispatch(setShowLoadingOverlay(false));
  });

  socket.on("custom-background", (data) => {
    const { img } = data;
    customBackground(img, false);
    store.dispatch(setShowLoadingOverlay(false));
  });

  socket.on("end-call", () => {
    leaveCallHandler();
  });

  socket.on("signal-loader", () => {
    store.dispatch(setShowLoadingOverlay(true));
  });
};

export const createCall = (identity) => {
  const data = {
    identity,
  };

  socket.emit("create-call", data);
};

export const joinCall = (identity, callID) => {
  const data = {
    identity,
    callID,
  };
  tempCallID = callID;

  socket.emit("join-call", data);
};

export const signalPeerData = (data) => {
  socket.emit("connection-signal", data);
};

export const signalARImageMatcher = () => {
  const data = { tempCallID };
  socket.emit("ar-image-matcher", data);
};

export const signalSwapVideo = () => {
  signalLoader();
  const data = { tempCallID };
  socket.emit("swap-video", data);
};
```

```
export const signalReset = () => {
  signalLoader();
  const data = { tempCallID };
  socket.emit("reset", data);
};

export const signalCustomBackground = (img) => {
  signalLoader();
  const data = { tempCallID, img };
  socket.emit("custom-background", data);
};

export const endCall = () => {
  const data = { tempCallID };
  socket.emit("end-call", data);
};

const signalLoader = () => {
  const data = { tempCallID };
  socket.emit("signal-loader", data);
};
```

4.4.5 Creating the Image-AR Matching Module

This section demonstrates building the Media Segmentation Handler system component which will deliver the application's intended capability to generate image to AR matching. This module constitutes of a single application file; *segmentationHandler.js* that is created inside the *utils* directory. Referring to coding, the *SegmentationHandler* function is first run as soon as two remote clients successfully join a call session together. Inside *webRTCHandler.js*, the *SegmentationHandler* function is called while passing in the host and peer user HTML video element ID as well as the output HTML canvas element ID to the function's parameters. On every initial run, the remote peer user's video is set as the media source for the algorithm to run the segmentation process on. The function to run the video segmentation is the *sendToMediaPipe* function where the peer user's video is passed in as the image source. *onResults* is the function that executes to generate the final output image frames. It takes in *results* as parameter which is the resulting image frame after it has been processed for segmentation through the *sendToMediaPipe* function. The *onResults* function reiterates with every frame of the segmented image received and with each execution it draws the background image, segmented image and the segmentation mask with them layered on top of one another onto the output HTML canvas element. This process creates the effect that the human subject in the segmented video frame is augmented onto the virtual environment which is the background video image. Furthermore, calling the *swapVideo* function enables switching between having either the host or peer to have their video become the segmentation source. This will then allow the host user to be augmented onto the peer's video environment instead. Additionally, there is also the *customBackground* function which allows the user to upload their own image file and have it drawn onto the output canvas as the background element. Unfortunately, due to the limitations of the Selfie Segmentation Model, only one instance of the *SelfieSegmentation* object can exist at one time. Only one video source may be processed for segmentation, meaning the system will not be able to augment both host and peer users together in the output video when the custom background mode is selected. Furthermore, there is also the *screenshotHandler* function which allows saving an image of the output video and the *resetSegmentation* function which resets the segmentation mode to its initial state. Whenever a user triggers a function, a call is made to the signaling server to emit an event to the remote server, emit the event back to the other peer user, and also trigger the specified function for them. This is done to keep application operation for both users synchronized.

segmentationHandler.js

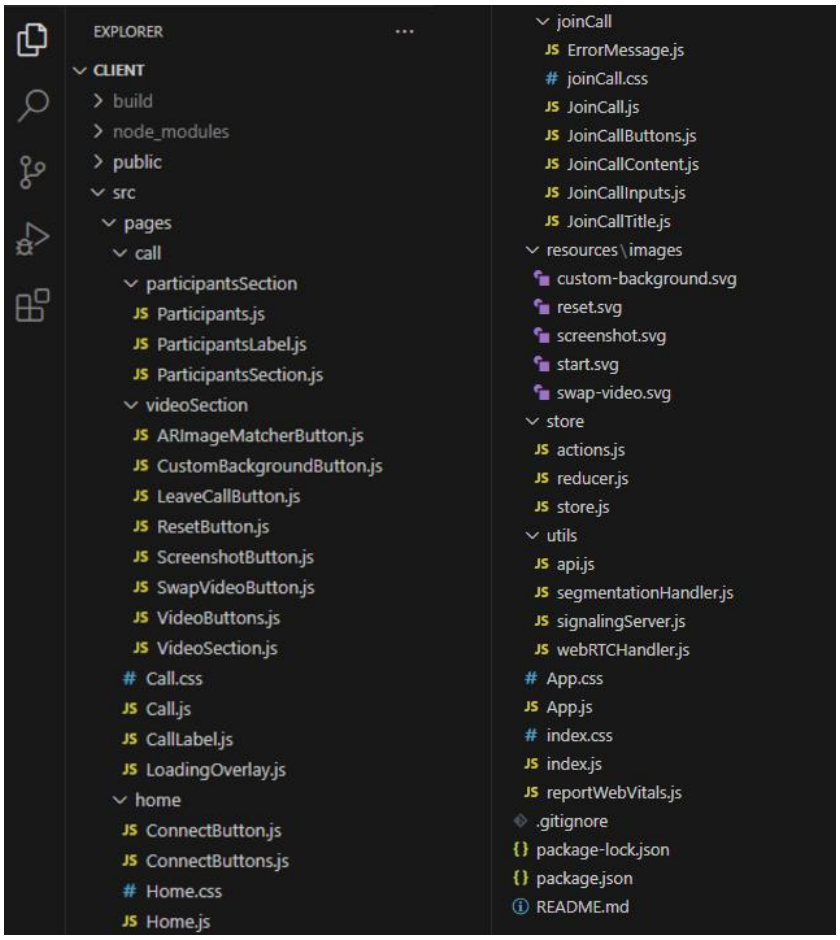
```
import { SelfieSegmentation } from "@mediapipe/selfie_segmentation";
import { signalCustomBackground } from "./signalingServer";
import canvasToImage from "canvas-to-image";

let isVideoSwapped = false;
let isCustomBackground = false;
let tempInitiator = false;
let stopSegmentation = false;
let tempHostVideoID, tempPeerVideoID, tempCanvasID;
let backgroundImageURL;

function SegmentationHandler(hostVideoID, peerVideoID, canvasID) {
  if (
    tempHostVideoID !== null &&
    tempPeerVideoID !== null &&
    tempCanvasID !== null
  ) {
    tempHostVideoID = hostVideoID;
    tempPeerVideoID = peerVideoID;
    tempCanvasID = canvasID;
  }

  let backgroundVideoElement;
  let segmentedVideoElement;

  if (!isCustomBackground) {
    if (isVideoSwapped) {
      backgroundVideoElement = document.getElementById(tempPeerVideoID);
      segmentedVideoElement = document.getElementById(tempHostVideoID);

      console.log("Peer video is set as background");
    } else {
      backgroundVideoElement = document.getElementById(tempHostVideoID);
      segmentedVideoElement = document.getElementById(tempPeerVideoID);

      console.log("Host video is set as background");
    }
  } else {
    let backgroundImg = new Image();
    backgroundImg.src = backgroundImageURL;
    backgroundVideoElement = backgroundImg;

    if (isVideoSwapped) {
      segmentedVideoElement = document.getElementById(tempPeerVideoID);
      console.log("Running segmentation on peer video");
    } else {
      segmentedVideoElement = document.getElementById(tempHostVideoID);
      console.log("Running segmentation on host video");
    }
  }

  const canvasElement = document.getElementById(tempCanvasID);
  const canvasCtx = canvasElement.getContext("2d");
```

```
function onResults(results) {
  canvasCtx.save();

  canvasCtx.clearRect(0, 0, canvasElement.width, canvasElement.height);

  canvasCtx.globalCompositeOperation = "source-over";
  canvasCtx.drawImage(
    results.segmentationMask,
    0,
    0,
    canvasElement.width,
    canvasElement.height
  );

  canvasCtx.globalCompositeOperation = "source-in";
  canvasCtx.drawImage(
    results.image,
    0,
    0,
    canvasElement.width,
    canvasElement.height
  );

  canvasCtx.globalCompositeOperation = "destination-over";
  canvasCtx.drawImage(
    backgroundVideoElement,
    0,
    0,
    canvasElement.width,
    canvasElement.height
  );

  canvasCtx.restore();

  if (stopSegmentation) {
    closeSegmentation();
    stopSegmentation = false;
  }
}

const selfieSegmentation = new SelfieSegmentation({
  locateFile: (file) => {
    return `https://cdn.jsdelivr.net/npm/@mediapipe/selfie_segmentation/${file}`;
  },
});

selfieSegmentation.setOptions({
  modelSelection: 1,
  selfieMode: false,
});

selfieSegmentation.onResults(onResults);

const sendToMediaPipe = async () => {
  try {
    await selfieSegmentation.send({ image: segmentedVideoElement });
    requestAnimationFrame(sendToMediaPipe);
  } catch (e) {
    console.log(e);
  }
};

sendToMediaPipe();
```

```
  const closeSegmentation = () => {
    selfieSegmentation.close();
  };
}

const restartSegmentation = () => {
  stopSegmentation = true;
  return SegmentationHandler(tempHostVideoID, tempPeerVideoID, tempCanvasID);
};

export const swapVideo = () => {
  isVideoSwapped = !isVideoSwapped;
  restartSegmentation();
};

export const customBackground = (img, initiator) => {
  if (initiator) {
    backgroundImageURL = img;
    isCustomBackground = true;
    tempInitiator = true;
    return;
  }

  if (!tempInitiator) {
    backgroundImageURL = img;
    isCustomBackground = true;
  } else { tempInitiator = false; }

  restartSegmentation();
};

export const backgroundImageHandler = (e) => {
  if (e.target.files[0].size <= 1e7) {
    const reader = new FileReader();
    reader.onload = () => {
      if (reader.readyState === 2) {
        customBackground(reader.result, true);
        signalCustomBackground(reader.result);
      } else { return; }
    };

    reader.readAsDataURL(e.target.files[0]);
  } else { alert("Maximum file size: 10MB."); }
};

export const screenshotHandler = () => {
  const canvasElement = document.getElementById(tempCanvasID);
  canvasToImage(canvasElement);
};

export const resetSegmentation = () => {
  isVideoSwapped = false;
  isCustomBackground = false;
  tempInitiator = false;
  backgroundImageURL = null;

  restartSegmentation();
};

export default SegmentationHandler;
```

4.4.6 Building the Front-End User Interface Components

This section showcases the rest of the system's file components and structure in which all of it encompasses the entire client application user interface. To store the files, inside the *src* directory, a parent directory titled *pages* is created contains three subdirectories namely *call*, *home*, and *joinCall*. The *call* directory also contains two other sub-directories namely *participantsSection* and *videoSection*. Moreover, another directory placed inside *src* named *resources* with its own subdirectory *images* is where .svg image files for UI button elements are stored. Figure 4.4.6 is an image of the system file structure of this section for reference. It should be noted that these are merely how the front-end UI has been specifically implemented for this project application and that those interested in rebuilding this system may develop the front-end UI in their own design. The important bit is being able to develop the front-end UI to work with all the components in the previous sections.



```
EXPLORER ...
v CLIENT
  > build
  > node_modules
  > public
  v src
    v pages
      v call
        v participantsSection
            Participants.js
            ParticipantsLabel.js
            ParticipantsSection.js
        v videoSection
            ARImageMatcherButton.js
            CustomBackgroundButton.js
            LeaveCallButton.js
            ResetButton.js
            ScreenshotButton.js
            SwapVideoButton.js
            VideoButtons.js
            VideoSection.js
          Call.css
          Call.js
          CallLabel.js
          LoadingOverlay.js
      v home
          ConnectButton.js
          ConnectButtons.js
          Home.css
          Home.js
      v joinCall
          ErrorMessage.js
          joinCall.css
          JoinCall.js
          JoinCallButtons.js
          JoinCallContent.js
          JoinCallInputs.js
          JoinCallTitle.js
    v resources\images
        custom-background.svg
        reset.svg
        screenshot.svg
        start.svg
        swap-video.svg
    v store
        actions.js
        reducer.js
        store.js
    v utils
        api.js
        segmentationHandler.js
        signalingServer.js
        webRTCHandler.js
      App.css
      App.js
      index.css
      index.js
      reportWebVitals.js
    .gitignore
    package-lock.json
    package.json
    README.md
```

Figure 4.4.6 The system's file structure

call/Call.js

```
import React, { useEffect } from "react";
import ParticipantsSection from "./participantsSection/ParticipantsSection";
import VideoSection from "./videoSection/VideoSection";
import CallLabel from "./CallLabel";
import { connect } from "react-redux";
import * as webRTCHandler from "../../utils/webRTCHandler";
import LoadingOverlay from "./LoadingOverlay";
import "./Call.css";

const Call = ({ callID, identity, isCallHost, showLoadingOverlay }) => {
  useEffect(() => {
    if (!isCallHost && !callID) {
      const siteUrl = window.location.origin;
      window.location.href = siteUrl;
    } else {
      webRTCHandler.getLocalVideoAndInitConnection(
        isCallHost,
        identity,
        callID
      );
    }
  }, []);

  return (
    <div className="call_container">
      <ParticipantsSection />
      <VideoSection />
      <CallLabel callID={callID} />
      {showLoadingOverlay && <LoadingOverlay />}
    </div>
  );
};

const mapStoreStateToProps = (state) => {
  return {
    ...state,
  };
};

export default connect(mapStoreStateToProps)(Call);
```

call/CallLabel.js

```
import React from "react";

const CallLabel = ({ callID }) => {
  return (
    <div className="call_label">
      <p className="call_label_paragraph">ID: {callID}</p>
    </div>
  );
};

export default CallLabel;
```

call/LoadingOverlay.js

```
import React from "react";

const LoadingOverlay = () => {
  return (
    <div className="loading_overlay_container">
      <div className="loader"></div>
    </div>
  );
};

export default LoadingOverlay;
```

call/Call.css

```
.call_container {
  width: 100%;
  height: 100vh;
  display: flex;
}

.message_container {
  margin-top: 5px;
  display: flex;
  flex-direction: column;
}

/* width */
::-webkit-scrollbar {
  width: 5px;
  background: rgba(0, 188, 164, 0.1);
}

/* Track */
::-webkit-scrollbar-track {
  box-shadow: inset 0 0 5px grey;
  border-radius: 10px;
}

/* Handle */
::-webkit-scrollbar-thumb {
  background: #e5e5e5;
  border-radius: 10px;
}

.loading_overlay_container {
  width: 100%;
  height: 100vh;
  background: rgb(10, 152, 218);
  opacity: 0.8;
  display: flex;
  justify-content: center;
  align-items: center;
  position: absolute;
  z-index: 2;
}
```

```
.loader {
  border: 16px solid #f3f3f3; /* Light grey */
  border-top: 16px solid #0052c9; /* Blue */
  border-radius: 50%;
  width: 120px;
  height: 120px;
  animation: spin 2s linear infinite;
}

@keyframes spin {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}

.participants_container {
  width: 100%;
  display: flex;
  justify-content: flex-start;
  flex-direction: column;
  align-content: flex-start;
  height: 50%;
}

.participants_paragraph {
  color: black;
  text-align: start;
  font-weight: 500;
  margin-left: 40px;
  transition: 0.5s;
  margin: 0 40px;
  padding: 10px 0px;
}

.participants_paragraph:hover {
  background-color: #e5e5e5;
  border-radius: 8px;
}

.participants_separator_line {
  width: calc(100% - 80px);
  height: 2px;
  background-color: #e5e5e5;
  margin-left: 40px;
  margin-top: 5px;
  margin-bottom: 5px;
}

.participants_section_container {
  height: 100%;
  width: 20%;
  background-color: white;
  display: flex;
  flex-direction: column;
}
```

```css
.participants_label_container {
  display: flex;
  align-items: flex-start;
  justify-content: flex-start;
}

.participants_label_paragraph {
  font-weight: 700;
  color: #9ca5ab;
  margin-left: 40px;
  font-size: 16px;
}

.video_buttons_container {
  width: 100%;
  height: 10%;
  display: flex;
  align-items: center;
  justify-content: center;
  position: absolute;
  left: 0;
  bottom: 0;
  background: linear-gradient(168.68deg, #0052c9 1.12%, #0a91db 100%);
  border-top-left-radius: 25px;
  border-top-right-radius: 25px;
}

.video_button_container {
  margin-left: 30px;
  display: flex;
  flex-direction: column;
}

.video_button_image {
  height: 80%;
  border-radius: 25px;
  transition: 0.3s;
  background: whitesmoke;
}

.video_button_image:hover {
  background: cornflowerblue;
  border-radius: 25px;
}

.video_button_image:active {
  transition: 0.3s;
  background: white;
}

.video_button_end {
  width: 200px;
  height: 60px;
  font-size: 18px;
  font-weight: 600;
  border-radius: 65px;
  border: none;
  color: white;
  background: #fc5d5b;
  box-shadow: 0px 3px 30px rgba(252, 93, 91, 0.1);
  transition: 0.2s;
}
```

```css
.video_button_end:hover {
  background: red;
}

.participants {
  display: flex;
  width: 100%;
  height: 100%;
}

.participant {
  flex-grow: 1;
}

.video {
  display: inline-block;
  width: 70%;
  height: 70%;
  position: relative;
  margin-top: 18%;
}

.call_label {
  position: absolute;
  display: flex;
  width: 100%;
  justify-content: center;
  z-index: 4;
}

.call_label_paragraph {
  font-size: 17px;
  color: white;
  margin-top: 0;
  padding: 18px 38px;
  background: linear-gradient(168.68deg, #0052c9 1.12%, #0a91db 100%);
  border-radius: 0px 0px 15px 15px;
}

.ar_output_container {
  width: 100%;
  height: 50%;
  position: relative;
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: -1;
}

.ar_output_canvas {
  /* width: 100%; */
  height: 100%;
  position: relative;
  display: inline-block;
}

.video_track_container {
  width: 50%;
  height: 40%;
  position: relative;
  text-align: center;
}
```

```
.video_section_container {
  height: 100%;
  width: 60%;
  display: flex;
  flex-direction: column;
  justify-content: flex-end;
}
```

call/participantsSection/Participants.js

```
import React from "react";
import { connect } from "react-redux";

const SingleParticipant = (props) => {
  const { identity, lastItem } = props;
  return (
    <>
      <p className="participants_paragraph">{identity}</p>
      {!lastItem && <span className="participants_separator_line"></span>}
    </>
  );
};
const Participants = ({ participants }) => {
  return (
    <div className="participants_container">
      {participants.map((participant, index) => {
        return (
          <SingleParticipant
            key={participant.identity}
            lastItem={participants.length === index + 1}
            participant={participant}
            identity={participant.identity}
          />
        );
      })}
    </div>
  );
};
const mapStoreStateToProps = (state) => {
  return {
    ...state,
  };
};
export default connect(mapStoreStateToProps)(Participants);
```

call/participantsSection/Participants.js

```
import React from "react";

const ParticipantsLabel = () => {
  return (
    <div className="participants_label_container">
      <p className="participants_label_paragraph">CALL PARTICIPANTS</p>
    </div>
  );
};

export default ParticipantsLabel;
```

call/participantsSection/ParticipantsSection.js

```
import React from "react";
import ParticipantsLabel from "./ParticipantsLabel";
import Participants from "./Participants";

const ParticipantsSection = () => {
  return (
    <div className="participants_section_container">
      <ParticipantsLabel />
      <Participants />
    </div>
  );
};

export default ParticipantsSection;
```

call/videoSection/VideoButtons.js

```
import React from "react";
import LeaveCallButton from "./LeaveCallButton";
import SwapVideoButton from "./SwapVideoButton";
import CustomBackgroundButton from "./CustomBackgroundButton";
import ScreenshotButton from "./ScreenshotButton";
import ResetButton from "./ResetButton";
import ARImageMatcherButton from "./ARImageMatcherButton";
import { getIsHost } from "../../../utils/webRTCHandler";

const VideoButtons = (props) => {
  if (getIsHost()) { return (
      <div id="video-buttons-container" className="video_buttons_container">
        <ARImageMatcherButton />
        <CustomBackgroundButton />
        <ScreenshotButton />
        <SwapVideoButton />
        <ResetButton />
        <LeaveCallButton />
      </div>
    );
  } else {
    return (
      <div id="video-buttons-container" className="video_buttons_container">
        <CustomBackgroundButton />
        <ScreenshotButton />
        <SwapVideoButton />
        <ResetButton />
        <LeaveCallButton />
      </div>
    );
  }
};
export const showUtilityButtons = () => {
  if (getIsHost()) { document.getElementById("ar-image-matcher-button").remove(); }
  document.getElementById("custom-background-button-icon").hidden = false;
  document.getElementById("screenshot-button-icon").hidden = false;
  document.getElementById("swap-video-button-icon").hidden = false;
  document.getElementById("reset-button-icon").hidden = false;
};
export default VideoButtons;
```

call/videoSection/ARImageMatcherButton.js

```
import startButton from "../../../resources/images/start.svg";
import { signalARImageMatcher } from "../../../utils/signalingServer";

const ARImageMatcherButton = () => {
  const ARImageMatcherButtonHandler = () => {
    signalARImageMatcher();
  };

  return (
    <div id="ar-image-matcher-button" className="video_button_container">
      <img
        alt="ar-img-matcher-button"
        src={startButton}
        className="video_button_image"
        onClick={ARImageMatcherButtonHandler}
      />
    </div>
  );
};

export default ARImageMatcherButton;
```

call/videoSection/CustomBackgroundButton.js

```
import customBackgroundButton from "../../../resources/images/custom-background.svg";
import { backgroundImageHandler } from "../../../utils/segmentationHandler";

const CustomBackgroundButton = () => {
  const customBackgroundButtonHandler = () => {
    document.getElementById("custom-background-input").click();
  };

  return (
    <div id="custom-background-button" className="video_button_container">
      <img
        id="custom-background-button-icon"
        alt="custom-background-button"
        src={customBackgroundButton}
        className="video_button_image"
        onClick={customBackgroundButtonHandler}
        hidden
      />

      <input
        id="custom-background-input"
        type="file"
        accept="image/*"
        multiple
        onChange={backgroundImageHandler}
        hidden
      />
    </div>
  );
};

export default CustomBackgroundButton;
```

call/videoSection/LeaveCallButton.js

```
import React from "react";
import { endCall } from "../../../utils/signalingServer";

const LeaveCallButton = () => {
  return (
    <div id="leave-call-button" className="video_button_container">
      <button
        className="video_button_end"
        onClick={(leaveCallHandler, endCall)}
      >
        Leave Call
      </button>
    </div>
  );
};

export const leaveCallHandler = () => {
  const siteURL = window.location.origin + "/image-ar-matcher";
  window.location.href = siteURL;
};

export default LeaveCallButton;
```

call/videoSection/ScreenshotButton.js

```
import screenshotButton from "../../../resources/images/screenshot.svg";
import { screenshotHandler } from "../../../utils/segmentationHandler";

const ScreenshotButton = () => {
  const screenshotButtonHandler = () => {
    setTimeout(screenshotHandler, 5000);
    setAnimation();
  };

  const setAnimation = () => {
    document.getElementById("screenshot-button").style.animation =
      "spin 1s ease 0s 5 normal forwards";
    setTimeout(resetAnimation, 5000);
  };

  const resetAnimation = () => {
    document.getElementById("screenshot-button").style.animation = "none";
  };
  return (
    <div id="screenshot-button" className="video_button_container">
      <img
        id="screenshot-button-icon"
        alt="screenshot-button"
        src={screenshotButton}
        className="video_button_image"
        onClick={screenshotButtonHandler}
        hidden
      />
    </div>
  );
};
export default ScreenshotButton;
```

call/videoSection/ResetButton.js

```
import resetButton from "../../../resources/images/reset.svg";
import { signalReset } from "../../../utils/signalingServer";

const ResetButton = () => {
  const resetHandler = () => {
    signalReset();
  };

  return (
    <div id="reset-button" className="video_button_container">
      <img
        id="reset-button-icon"
        alt="reset-button"
        src={resetButton}
        className="video_button_image"
        onClick={resetHandler}
        hidden
      />
    </div>
  );
};

export default ResetButton;
```

call/videoSection/SwapVideoButton.js

```
import swapVideoButton from "../../../resources/images/swap-video.svg";
import { signalSwapVideo } from "../../../utils/signalingServer";

const SwapVideoButton = () => {
  const swapVideoButtonHandler = () => {
    signalSwapVideo();
  };

  return (
    <div id="swap-video-button" className="video_button_container">
      <img
        id="swap-video-button-icon"
        alt="swap-video-button"
        src={swapVideoButton}
        className="video_button_image"
        onClick={swapVideoButtonHandler}
        hidden
      />
    </div>
  );
};

export default SwapVideoButton;
```

call/videoSection/VideoSection.js

```
import React from "react";
import VideoButtons from "./VideoButtons";

const VideoSection = () => {
  return (
    <div className="video_section_container">
      <VideoButtons />
    </div>
  );
};

export default VideoSection;
```

home/ConnectButton.js

```
import React from "react";

const ConnectButton = ({
  createCallButton = false,
  buttonText,
  onClickHandler,
}) => {
  const buttonClass = createCallButton
    ? "create_call_button" : "join_call_button";
  return (
    <button className={buttonClass} onClick={onClickHandler}>
      {buttonText}
    </button>
  );
};

export default ConnectButton;
```

home/Home.js

```
import React, { useEffect } from "react";
import ConnectButtons from "./ConnectButtons";
import { connect } from "react-redux";
import { setIsCallHost } from "../../store/actions";
import "./Home.css";

const Home = ({ setIsCallHostAction }) => {
  useEffect(() => { setIsCallHostAction(false); }, []);
  return (
    <div className="home_container">
      <div className="home_panel">
        <h2>AUGMENTED REALITY MATCHER</h2>
        <ConnectButtons />
      </div>
    </div>
  );
};

const mapActionsToProps = (dispatch) => { return {
    setIsCallHostAction: (isCallHost) => dispatch(setIsCallHost(isCallHost)), };
};
export default connect(null, mapActionsToProps)(Home);
```

home/Home.css

```css
.home_container {
  width: 100%;
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
}

.home_panel {
  width: 600px;
  height: 400px;
  background-color: white;
  border: 1px solid grey;
  filter: drop-shadow(0 0 0.2rem grey);
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: space-evenly;
}

.home_image {
  width: 150px;
}

.connect_buttons_container {
  display: flex;
  flex-direction: column;
}

.join_call_button {
  background-color: #2d8cff;
  border: 1px solid #e5e5e5;
  color: white;
  font-size: 13px;
  font-weight: 700;
  width: 180px;
  height: 30px;
  border-radius: 8px;
  transition: 0.3s;
}

.join_call_button:hover { background-color: blue; }

.create_call_button {
  margin-top: 15px;
  background-color: white;
  border: 1px solid #e5e5e5;
  color: black;
  font-size: 13px;
  width: 180px;
  height: 30px;
  border-radius: 8px;
  transition: 0.3s;
}

.create_call_button:hover {
  background-color: #e5e5e5;
}
```

home/ConnectButtons.css

```
import React from "react";
import ConnectButton from "./ConnectButton";
import { useNavigate } from "react-router-dom";

const ConnectButtons = () => {
  let navigate = useNavigate();

  const pushToJoinCall = () => {
    navigate("/join-call");
  };

  const pushToJoinCallAsHost = () => {
    navigate("/join-call?host=true");
  };

  return (
    <div className="connect_buttons_container">
      <ConnectButton
        buttonText="Join call"
        onClickHandler={pushToJoinCall}
      />
      <ConnectButton
        createCallButton
        buttonText="Host call"
        onClickHandler={pushToJoinCallAsHost}
      />
    </div>
  );
};

export default ConnectButtons;
```

joinCall/joinCall.css

```
.join_call_container {
  width: 100%;
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  position: relative;
}

.join_call_panel {
  width: 400px;
  height: 400px;
  background-color: white;
  border: 1px solid grey;
  filter: drop-shadow(0 0 0.2rem grey);
  display: flex;
  flex-direction: column;
  align-items: flex-start;
}

.join_call_title {
  font-size: 24px;
  font-weight: 700;
  margin-left: 35px;
  margin-top: 80px;
}
```

```css
.join_call_inputs_container {
  display: flex;
  flex-direction: column;
  height: 100px;
  justify-content: space-between;
  width: 100%;
  align-items: center;
  margin-top: 10px;
}

.error_message_container {
  display: flex;
  height: 50px;
  margin-left: 35px;
}

.error_message_paragraph {
  color: red;
}

.loading_overlay_container {
  position: absolute;
  display: flex;
  align-items: center;
  justify-content: center;
  width: 100%;
  height: 100%;
  left: 0;
  top: 0;
  background: rgba(0, 82, 201, 0.7);
}

.loading_overlay_loader {
  border: 16px solid #f3f3f3; /* Light grey */
  border-top: 16px solid #0052c9; /* Blue */
  border-radius: 50%;
  width: 120px;
  height: 120px;
  animation: spin 2s linear infinite;
}

@keyframes spin {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}

.checkbox_container {
  display: flex;
  margin-left: 35px;
  margin-top: 5px;
  align-items: center;
}
```

```css
.checkbox_connection {
  width: 25px;
  height: 25px;
  background: #2d8cff;
  border: 1px solid rgba(255, 255, 255, 0.5);
  border-radius: 8px;
  backdrop-filter: blur(80px);
  display: flex;
  align-items: center;
  justify-content: center;
}

.checkbox_container_paragraph {
  font-weight: 400;
  font-size: 16px;
  margin-left: 15px;
}

.checkbox_image {
  max-width: 100%;
  max-height: 100%;
}

.join_call_buttons_container {
  display: flex;
  justify-content: flex-end;
  width: 100%;
}

.join_call_cancel_button {
  width: 80px;
  margin-right: 35px;
  height: 30px;
  background-color: white;
  border: 1px solid gray;
  border-radius: 8px;
  transition: 0.3s;
}

.join_call_cancel_button:hover {
  background-color: #e5e5e5;
}

.join_call_success_button {
  width: 80px;
  margin-right: 15px;
  height: 30px;
  background-color: #2d8cff;
  border-radius: 8px;
  border: none;
  font-weight: 700;
  color: white;
  transition: 0.3s;
}

.join_call_success_button:hover {
  background-color: blue;
}
```

```
.join_call_inputs_container {
  display: flex;
  flex-direction: column;
  height: 100px;
  justify-content: space-between;
  width: 100%;
  align-items: center;
  margin-top: 10px;
}

.join_call_input {
  border-radius: 12px;
  border: 1px solid #e5e5e5;
  width: 300px;
  height: 40px;
  padding: 0px 15px;
}

.join_call_input:focus {
  border: 1px solid blue;
}
```

joinCall/JoinCall.js

```
import React, { useEffect } from "react";
import { useLocation } from "react-router-dom";
import { connect } from "react-redux";
import { setIsCallHost } from "../../store/actions";
import JoinCallTitle from "./JoinCallTitle";
import JoinCallContent from "./JoinCallContent";
import "./JoinCall.css";

const JoinCall = (props) => {
  const { setIsCallHostAction, isCallHost } = props;
  const search = useLocation().search;
  useEffect(() => {
    const isCallHost = new URLSearchParams(search).get("host");
    if (isCallHost) { setIsCallHostAction(true); }}, []);

  return (
    <div className="join_call_container">
      <div className="join_call_panel">
        <JoinCallTitle isCallHost={isCallHost} />
        <JoinCallContent />
      </div>
    </div>
  );
};

const mapStoreStateToProps = (state) => {
  return { ...state, };
};

const mapActionsToProps = (dispatch) => {
  return {
    setIsCallHostAction: (isCallHost) => dispatch(setIsCallHost(isCallHost)),
  };
};
export default connect(mapStoreStateToProps, mapActionsToProps)(JoinCall);
```

joinCall/ErrorMessage.js

```
import React from "react";

const ErrorMessage = ({ errorMessage }) => {
  return (
    <div className="error_message_container">
      {errorMessage && (
        <p className="error_message_paragraph">{errorMessage}</p>
      )}
    </div>
  );
};

export default ErrorMessage;
```

joinCall/JoinCallInputs.js

```
import React from "react";

const Input = ({ placeholder, value, changeHandler }) => {
  return (
    <input
      value={value}
      onChange={changeHandler}
      className="join_call_input"
      placeholder={placeholder}
    />
  );
};

const JoinCallInputs = (props) => {
  const { callIDValue, setCallIDValue, nameValue, setNameValue, isCallHost } = props;
  const callIDValueChangeHandler = (event) => {
    setCallIDValue(event.target.value);
  };
  const nameValueChangeHandler = (event) => {
    setNameValue(event.target.value);
  };

  return (
    <div className="join_call_inputs_container">
      {!isCallHost && (
        <Input
          placeholder="Enter call ID"
          value={callIDValue}
          changeHandler={callIDValueChangeHandler}
        />
      )}
      <Input
        placeholder="Enter your name"
        value={nameValue}
        changeHandler={nameValueChangeHandler}
      />
    </div>
  );
};

export default JoinCallInputs;
```

joinCall/JoinCallButtons.js

```
import React from "react";
import { useNavigate } from "react-router-dom";

const Button = ({ buttonText, cancelButton = false, onClickHandler }) => {
  const buttonClass = cancelButton
    ? "join_call_cancel_button"
    : "join_call_success_button";

  return (
    <button onClick={onClickHandler} className={buttonClass}>
      {buttonText}
    </button>
  );
};

const JoinCallButtons = ({ joinCallHandler, isCallHost }) => {
  const successButtonText = isCallHost ? "Host" : "Join";

  const navigate = useNavigate();
  const pushToHome = () => {
    navigate("/");
  };

  return (
    <div className="join_call_buttons_container">
      <Button buttonText={successButtonText} onClickHandler={joinCallHandler} />
      <Button
        buttonText="Cancel"
        cancelButton
        onClickHandler={pushToHome}
      />
    </div>
  );
};

export default JoinCallButtons;
```

joinCall/JoinCallTitle.js

```
import React from "react";

const JoinCallTitle = ({ isCallHost }) => {
  const titleText = isCallHost ? "Host Call" : "Join Call";
  return <p className="join_call_title">{titleText}</p>;
};
export default JoinCallTitle;
```

joinCall/JoinCallContent.js

```
import React, { useState } from "react";
import JoinCallInputs from "./JoinCallInputs";
import { connect } from "react-redux";
import ErrorMessage from "./ErrorMessage";
import JoinCallButtons from "./JoinCallButtons";
import { useNavigate } from "react-router-dom";
import { getCallExists } from "../../utils/api";
import { setIdentity, setCallID } from "../../store/actions";
```

```
const JoinCallContent = (props) => {
  const { isCallHost, setIdentityAction, setCallIDAction } = props;
  const [callIDValue, setCallIDValue] = useState("");
  const [nameValue, setNameValue] = useState("");
  const [errorMessage, setErrorMessage] = useState(null);
  const navigate = useNavigate();

  const joinCallHandler = async () => {
    if (!nameValue) {
      setErrorMessage("Your name is required");
    } else {
      setIdentityAction(nameValue);
      if (isCallHost) {
        createCall();
      } else {
        await joinCall();
      }
    }
  };

  const joinCall = async () => {
    const responseMessage = await getCallExists(callIDValue);
    const { callExists, callFull } = responseMessage;

    if (callExists) {
      if (callFull) {
        setErrorMessage("The call session is full");
      } else {
        setCallIDAction(callIDValue);
        navigate("/call");
      }
    } else {
      setErrorMessage("Unidentified call ID");
    }
  };

  const createCall = () => {
    navigate("/call");
  };

  return (
    <>
      <JoinCallInputs
        callIDValue={callIDValue}
        setCallIDValue={setCallIDValue}
        nameValue={nameValue}
        setNameValue={setNameValue}
        isCallHost={isCallHost}
      />
      <ErrorMessage errorMessage={errorMessage} />
      <JoinCallButtons
        joinCallHandler={joinCallHandler}
        isCallHost={isCallHost}
      />
    </>
  );
};
```

```
const mapStoreStateToProps = (state) => {
  return {
    ...state,
  };
};

const mapActionsToProps = (dispatch) => {
  return {
    setIdentityAction: (identity) => dispatch(setIdentity(identity)),
    setCallIDAction: (callID) => dispatch(setCallID(callID)),
  };
};

export default connect(
  mapStoreStateToProps,
  mapActionsToProps
)(JoinCallContent);
```

4.4.7 Additional Adjustments

Some final adjustments will be made before the system is ready for deployment to the web. Firstly, the fourth line of code in the snippet below is added to *index.html* which is located inside the *public* directory:

```
<body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <div id="videos_container"></div>
</body>
```

The next sets of adjustments are made to the applications files which will all be located inside the *src* directory. Codes for *App.js* are replaced with the following:

```
import React, { useEffect } from "react";
import { createHashRouter, RouterProvider } from "react-router-dom";
import Home from "./pages/home/Home";
import JoinCall from "./pages/joinCall/JoinCall";
import Call from "./pages/call/Call";
import { connectWithSocketIOServer } from "./utils/signalingServer";
import "./App.css";

function App() {
  const router = createHashRouter([
    { path: "/", element: <Home />, },
    { path: "/join-call", element: <JoinCall />, },
    { path: "/call", element: <Call />, },
  ]);

  useEffect(() => {
    connectWithSocketIOServer();
  }, []);

  return <RouterProvider router={router} />;
}

export default App;
```

The following class is added to *index.css*:

```
.videos_container {
  margin-left: 20%;
  width: 60%;
  height: calc(100vh - 10%);
  left: 0;
  top: 0;
  position: absolute;
  display: flex;
  flex-wrap: wrap;
}
```

The *React.StrictMode* component inside *index.js* is commented out or removed:

```
// <React.StrictMode>
<Provider store={store}>
  <App />
</Provider>
// </React.StrictMode>
```

The *App.css* file should either have its entire content removed or have the file itself deleted since it will not be needed. Next, the first line of code in the snippet below is added to *package.json* which will add the website URL for the system.

```
"homepage": "https://hundson.github.io/image-ar-matcher",
"name": "client",
"version": "0.1.0",
"private": true,
"dependencies": {
```

The third and fourth lines of code in the snippet below is also added to *package.json*.

```
"scripts": {
"start": "react-scripts start",
"predeploy": "npm run build",
"deploy": "gh-pages -d build",
```

The process for deploying the system to the web are detailed in Chapter 5.3 Web Deployment.

Chapter 5
System Implementation

5.1 Hardware Setup

Description	Specification
Operating System	Windows 10
CPU	AMD Ryzen 5 3500X
GPU	AMD Radeon RX 6600M
RAM	16GB
Webcam	PC Device Camera

Table 5.1.1 Device hardware specifications used in application development

5.2 Software Setup

Tools	Description
Visual Studio Code	Main integrated development environment (IDE) and code editor for application development.
Google Chrome	Browser used for running and testing application deployment.
Node.js	Cross-platform, open-source JavaScript runtime environment which enables JavaScript to run outside of a browser. This allows developing the back-end to handle client requests all in pure JavaScript, providing ease in the overall development and maintenance process.

WebRTC	Stands for "Web Real-Time Communication". Web framework to enable real-time communications through web browsers. Implemented through JavaScript API and allows browsers to exchange media including audio, video, and various other data.
MediaPipe Selfie Segmentation	Machine learning model that segments prominent humans in a given scene. The project application will utilize the JavaScript API provided to implement the image segmentation functionality.

Table 5.2.1 Software tools and technologies utilized in the system development

By using Node.js for development, this enables access to NPM (Node Package Manager) for installing JavaScript libraries known as Node.js packages (also called modules or dependencies). For this application project, various modules will be installed and made to work with each other to deliver the application's functionalities.

Node.js Package	**Description**
axios	Used for making HTTP requests, similar to Fetch API native for JavaScript but provides better ease of use.
canvas-to-image	Converts HTML canvas elements into images. Combined with html2canvas to provide screenshot functionality.
cors	Allows sending and receiving of resources and data between sites of different origin (Cross-Origin Resource Sharing).
express	Node.js web application framework. Provides robust set of features fundamental for developing the application back-end.
mediapipe/ selfie_segmentation	Node.js package to implement the MediaPipe Selfie Segmentation solution model.
node-cron	Node.js tool to schedule tasks to run on a periodic basis. Used to schedule making an HTTP GET request to the server hosted on Render to keep it awake and readily available at all times.
react	JavaScript library that enables much more ease in developing interactive web UI for the front-end of the application.

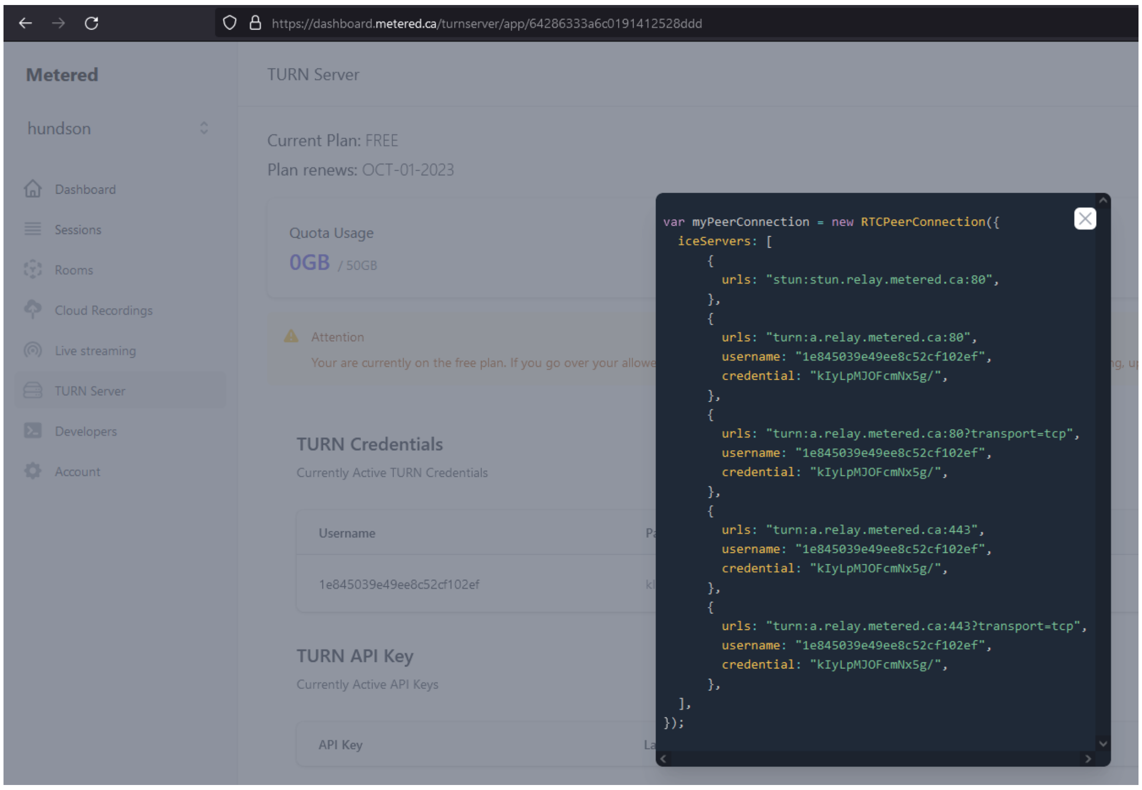
react-dom	React package for working with HTML DOM (Document Object Model). Installs automatically with React.
react-redux	React UI bindings layer for Redux. Enables React components to read data from a Redux store and dispatch actions to the Redux store to update state.
react-router-dom	Enables using React Router for client-side routing. Faster and more efficient method for website navigation.
react-scripts	Package that includes scripts and configurations used by React. As such, it is installed automatically with React.
reduxjs/toolkit	Redux Toolkit package to enable implementation of the Session States Store system component.
simple-peer	Part of the WebRTC API library. Establishes a connection between web browsers using WebRTC. This allows direct peer-to-peer communication without the need of a server.
socket.io	For creating WebSockets and building the signaling server. WebSockets allows for persistent, low-latency, and bidirectional communication between a client and server.
socket.io-client	Client-side implementation for Socket.IO.
uuid	UUID (Universal Unique Identifier). Enables application to generate unique IDs used for joining a call session.
gh-pages	Node.js package for deploying to GitHub Pages.

Table 5.2.2 Node.js package dependencies implemented in the system

5.3 Web Deployment

5.3.1 Acquiring TURN Server Credentials

This project system will utilize Open Relay which is a free TURN server provided by Metered Video. Credentials configuration for getting access to the TURN server can be acquired by signing up for an account over at https://www.metered.ca/. After logging in, the free TURN server configuration can be accessed in the Turn Server tab of the dashboard page. These configurations are placed in the *getConfiguration* function inside *webRTCHandler.js* as previously demonstrate in Chapter 4.4.4.



Current Plan: FREE
Plan renews: OCT-01-2023

Quota Usage
0GB / 50GB

Attention
Your are currently on the free plan. If you go over your allowe

TURN Credentials
Currently Active TURN Credentials

Username
1e845039e49ee8c52cf102ef

TURN API Key
Currently Active API Keys

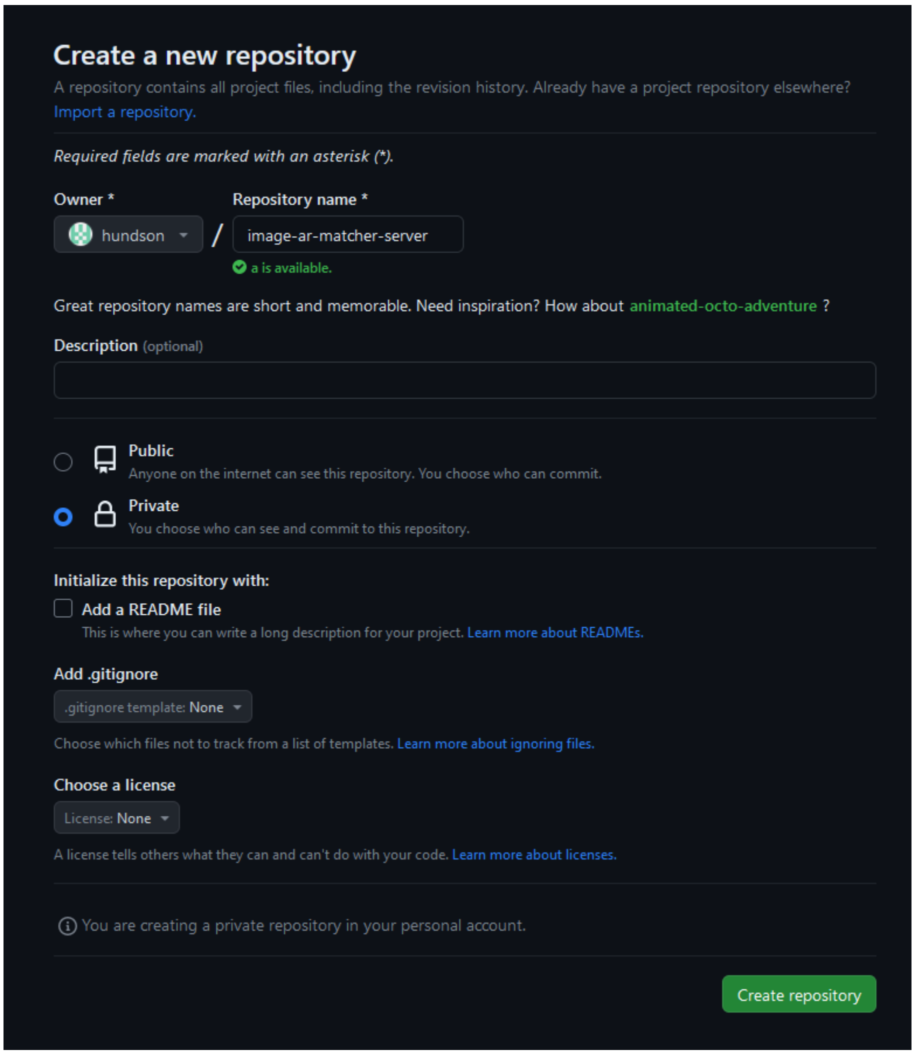
API Key

```
var myPeerConnection = new RTCPeerConnection({
  iceServers: [
      {
        urls: "stun:stun.relay.metered.ca:80",
      },
      {
        urls: "turn:a.relay.metered.ca:80",
        username: "1e845039e49ee8c52cf102ef",
        credential: "kIyLpMJOFcmNx5g/",
      },
      {
        urls: "turn:a.relay.metered.ca:80?transport=tcp",
        username: "1e845039e49ee8c52cf102ef",
        credential: "kIyLpMJOFcmNx5g/",
      },
      {
        urls: "turn:a.relay.metered.ca:443",
        username: "1e845039e49ee8c52cf102ef",
        credential: "kIyLpMJOFcmNx5g/",
      },
      {
        urls: "turn:a.relay.metered.ca:443?transport=tcp",
        username: "1e845039e49ee8c52cf102ef",
        credential: "kIyLpMJOFcmNx5g/",
      },
  ],
});
```

Figure 5.3.1 Open Relay free TURN server credentials configuration

5.3.2 Deploying the System Server Component

Before the server component can be deployed to Render, it will first be added as a new repository on GitHub. The following sets of figures will demonstrate the process. Additionally, a *.gitignore* file should be added to the root directory of the server component to configure ignoring the *node_modules* directory to prevent it from being uploaded to GitHub.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner *: hundson / Repository name *: image-ar-matcher-server

a is available.

Great repository names are short and memorable. Need inspiration? How about animated-octo-adventure ?

Description (optional)

Public — Anyone on the internet can see this repository. You choose who can commit.

[✓ Private] — You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file — This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore: .gitignore template: None

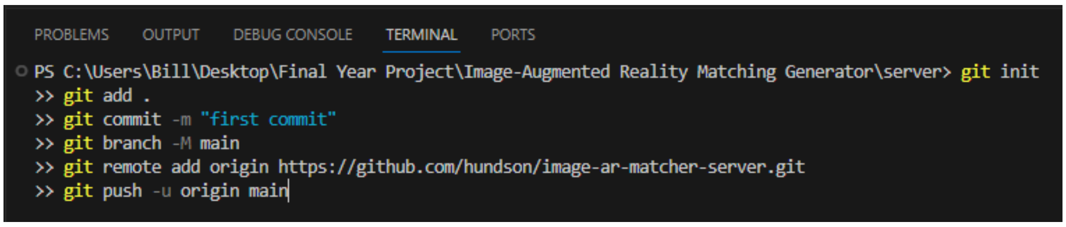
Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license: License: None

A license tells others what they can and can't do with your code. Learn more about licenses.

You are creating a private repository in your personal account.

Figure 5.3.2 Creating a new repository on GitHub



```
PS C:\Users\Bill\Desktop\Final Year Project\Image-Augmented Reality Matching Generator\server> git init
>> git add .
>> git commit -m "first commit"
>> git branch -M main
>> git remote add origin https://github.com/hundson/image-ar-matcher-server.git
>> git push -u origin main
```

Figure 5.3.3 Pushing the server component directory to the GitHub repository

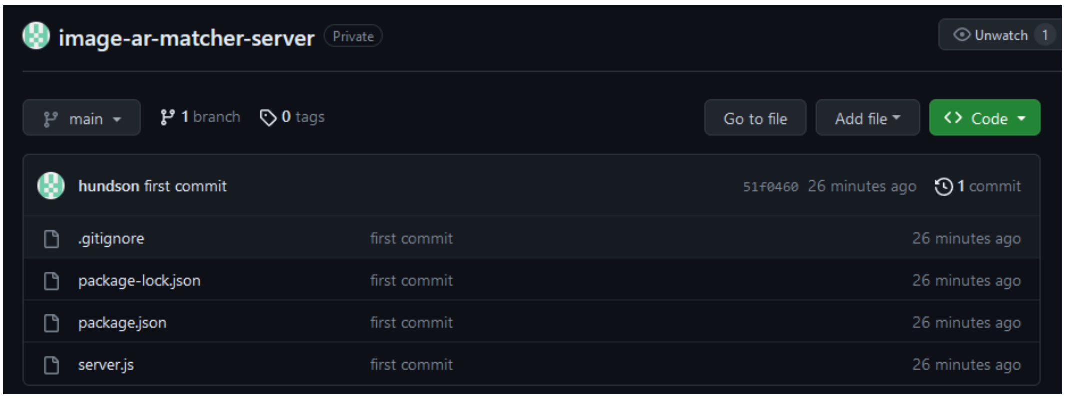
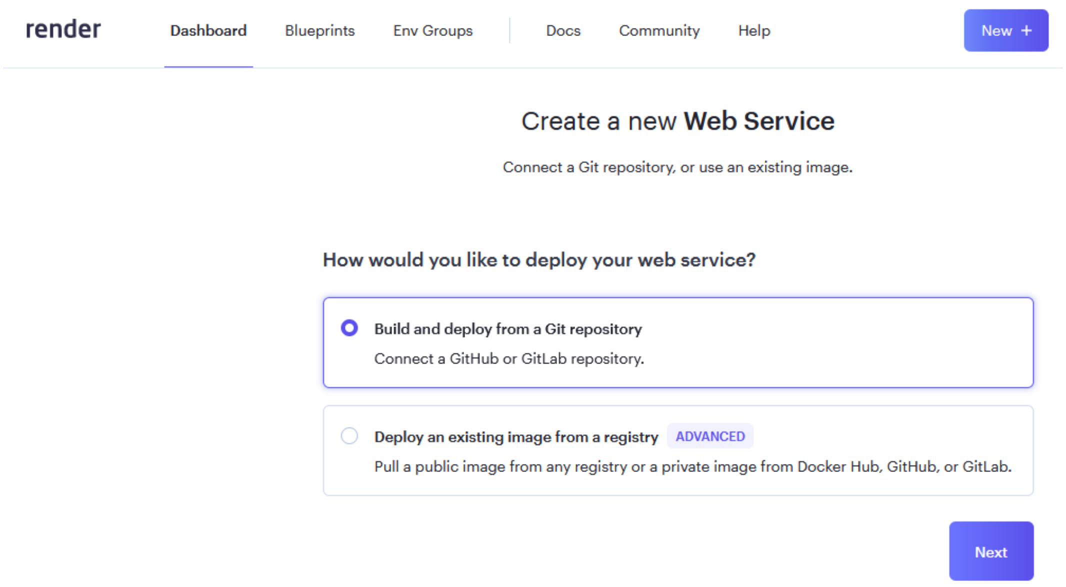


image-ar-matcher-server Private

hundson first commit		51f0460 26 minutes ago · 1 commit
.gitignore	first commit	26 minutes ago
package-lock.json	first commit	26 minutes ago
package.json	first commit	26 minutes ago
server.js	first commit	26 minutes ago

Figure 5.3.4 The server component successfully uploaded to GitHub

The server component is now ready to be deployed to Render. The next sets of figures will demonstrate the process.


Create a new Web Service

Connect a Git repository, or use an existing image.

How would you like to deploy your web service?

● Build and deploy from a Git repository
Connect a GitHub or GitLab repository.

○ Deploy an existing image from a registry ADVANCED
Pull a public image from any registry or a private image from Docker Hub, GitHub, or GitLab.

Figure 5.3.5 Creating a new service by building and deploying from the Git repository

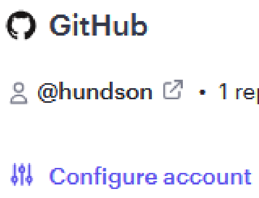
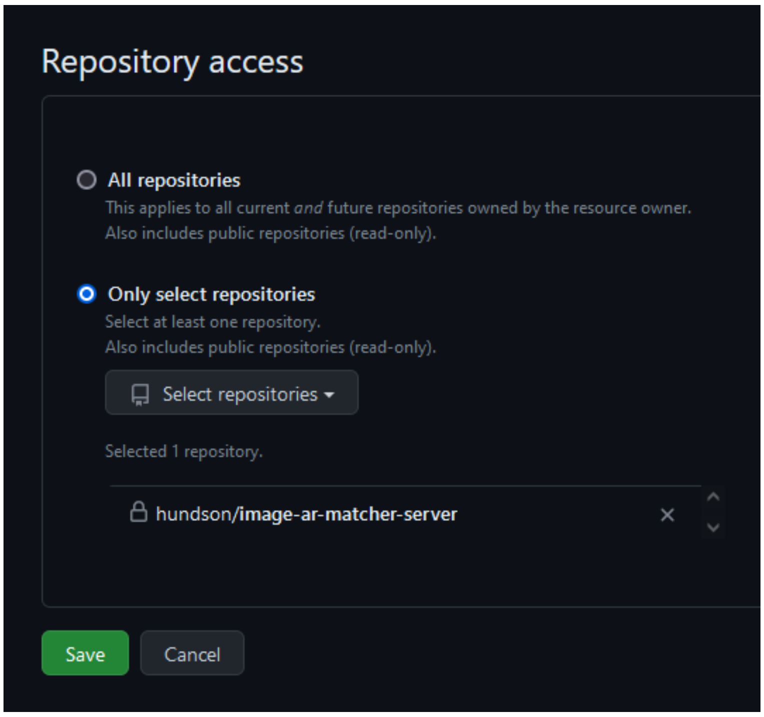


Figure 5.3.6 Link to Configure the GitHub account to work with Render


Repository access

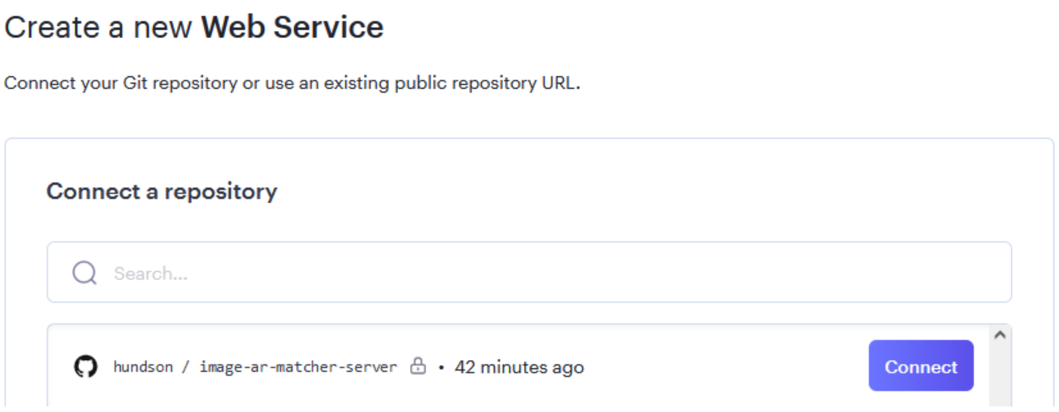
( ) All repositories
This applies to all current *and* future repositories owned by the resource owner.
Also includes public repositories (read-only).

(●) Only select repositories
Select at least one repository.
Also includes public repositories (read-only).

Selected 1 repository.

hundson/image-ar-matcher-server

Figure 5.3.7 Selecting the server component Git repository to link to Render

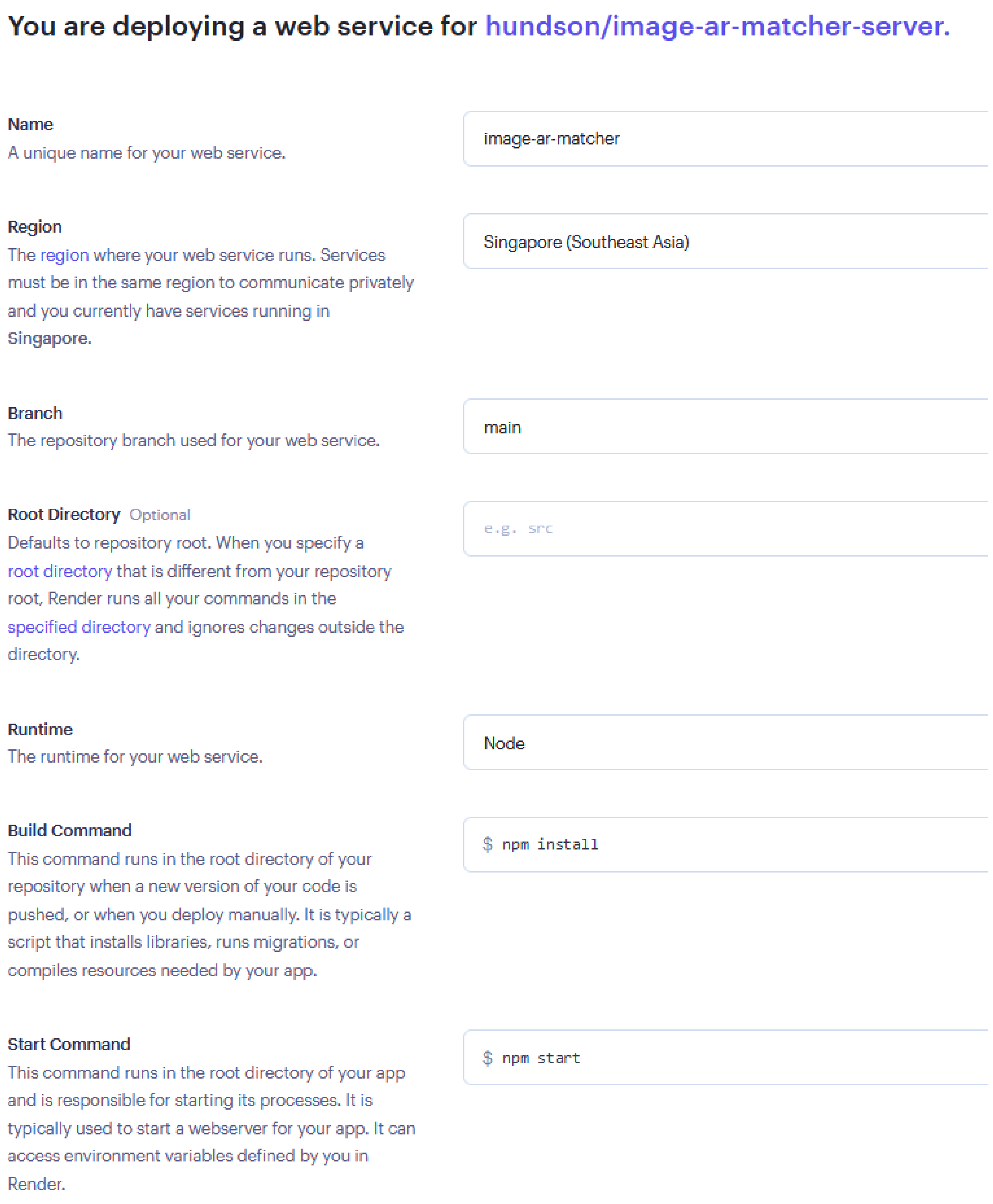

Create a new **Web Service**

Connect your Git repository or use an existing public repository URL.

Connect a repository

hundson / image-ar-matcher-server • 42 minutes ago

Figure 5.3.8 Connecting Render to the server component repository


You are deploying a web service for hundson/image-ar-matcher-server.

Name A unique name for your web service.	image-ar-matcher
Region The region where your web service runs. Services must be in the same region to communicate privately and you currently have services running in Singapore.	Singapore (Southeast Asia)
Branch The repository branch used for your web service.	main
Root Directory Optional Defaults to repository root. When you specify a root directory that is different from your repository root, Render runs all your commands in the specified directory and ignores changes outside the directory.	e.g. src
Runtime The runtime for your web service.	Node
Build Command This command runs in the root directory of your repository when a new version of your code is pushed, or when you deploy manually. It is typically a script that installs libraries, runs migrations, or compiles resources needed by your app.	`$ npm install`
Start Command This command runs in the root directory of your app and is responsible for starting its processes. It is typically used to start a webserver for your app. It can access environment variables defined by you in Render.	`$ npm start`

Figure 5.3.9 Configurations for the web service deployment

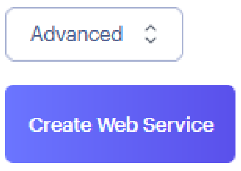
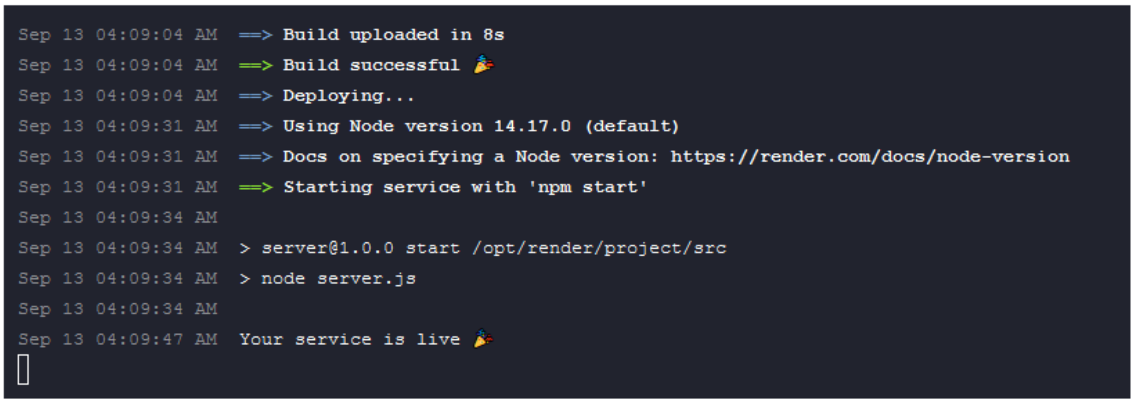


Figure 5.3.10 Button to start deployment of the web service

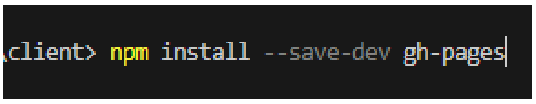


```
Sep 13 04:09:04 AM  ==> Build uploaded in 8s
Sep 13 04:09:04 AM  ==> Build successful 🎉
Sep 13 04:09:04 AM  ==> Deploying...
Sep 13 04:09:31 AM  ==> Using Node version 14.17.0 (default)
Sep 13 04:09:31 AM  ==> Docs on specifying a Node version: https://render.com/docs/node-version
Sep 13 04:09:31 AM  ==> Starting service with 'npm start'
Sep 13 04:09:34 AM
Sep 13 04:09:34 AM  > server@1.0.0 start /opt/render/project/src
Sep 13 04:09:34 AM  > node server.js
Sep 13 04:09:34 AM
Sep 13 04:09:47 AM  Your service is live 🎉
```

Figure 5.3.11 Successful deployment of the system server component web service

5.3.3 Deploying the System Client Component

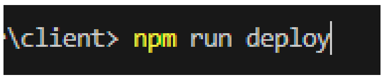
The client component will be added to a new Git repository and subsequently published through GitHub Pages. After deployment, the system will then be publicly available for visiting on the internet through the given URL. Before that, the gh-pages dependency is installed.



```
client> npm install --save-dev gh-pages
```

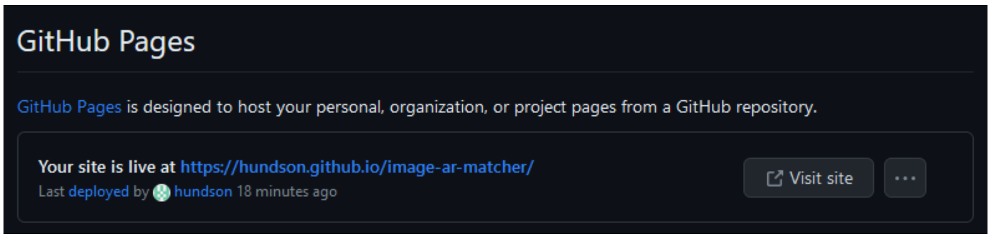
Figure 5.3.12 Installing gh-pages as a developer dependency

Adding the client component to a GitHub repository follows the same process as in Chapter 5.3.2 for the server component. If it doesn't already exists, a *.gitignore* file should also be added to ignore *node_modules*. After uploading to GitHub, the following command is run:



```
\client> npm run deploy
```

Figure 5.3.13 Terminal command for deploying to gh-pages



GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at https://hundson.github.io/image-ar-matcher/

Last deployed by hundson 18 minutes ago

Figure 5.3.14 Successful deployment to GitHub Pages

5.4 System Operation

The system has been hosted over at https://hundson.github.io/image-ar-matcher/. System operations demonstration for this section has been done with the help of an assistant acting as the remote peer client user and running the website from a remote location.

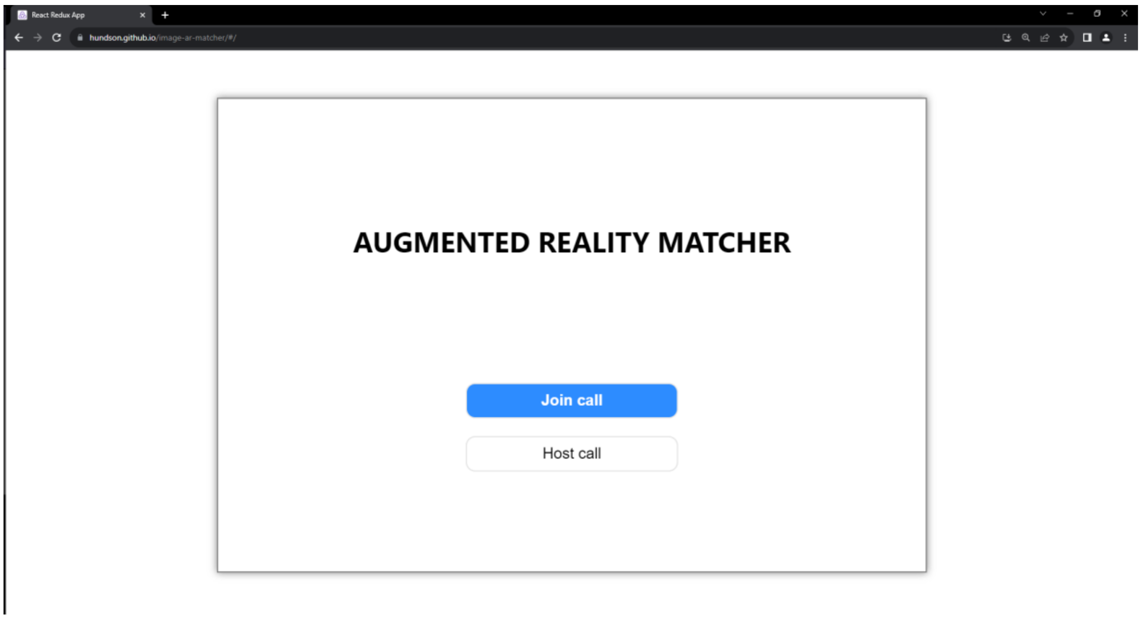

React Redux App
hundson.github.io/image-ar-matcher/#/
AUGMENTED REALITY MATCHER
Join call
Host call

Figure 5.4.1 System Homepage

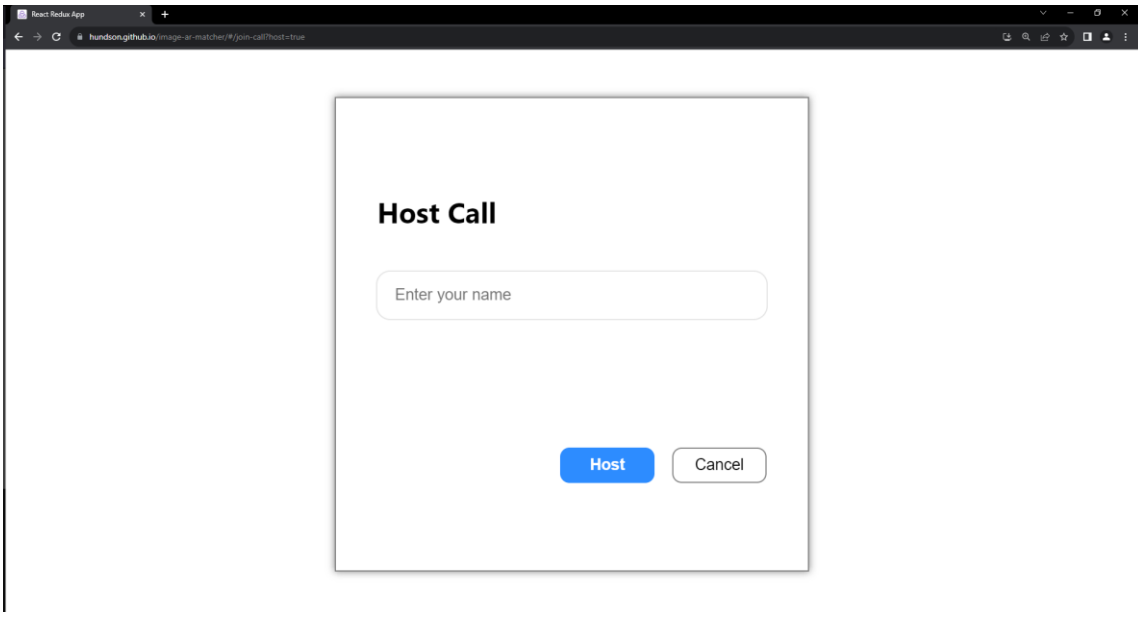


Figure 5.4.2 Host Call Page

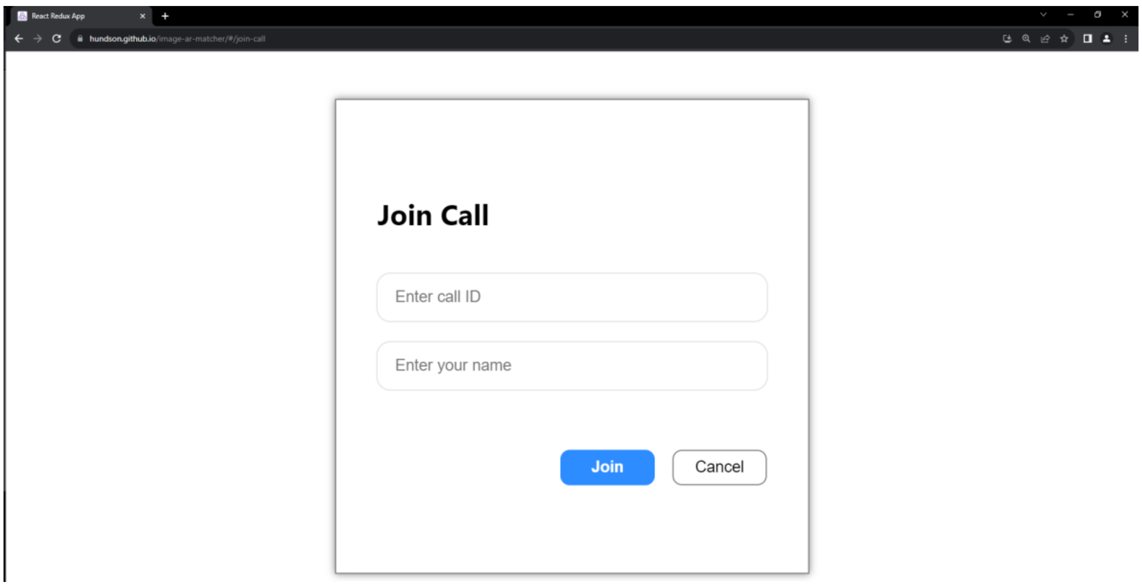
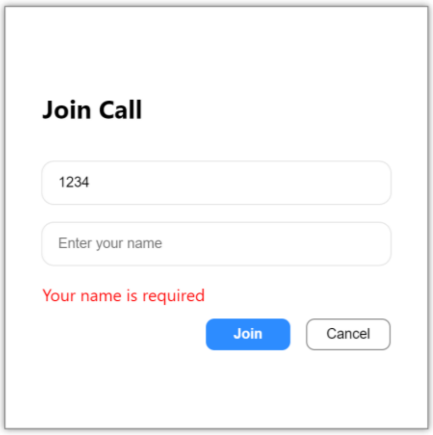
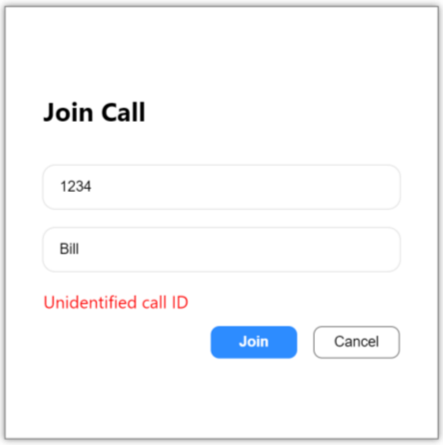


Figure 5.4.3 Join Call Page



Your name is required

Figure 5.4.4 Input validation for empty name field



Unidentified call ID

Figure 5.4.5 Input validation when entering a non-existent call session ID

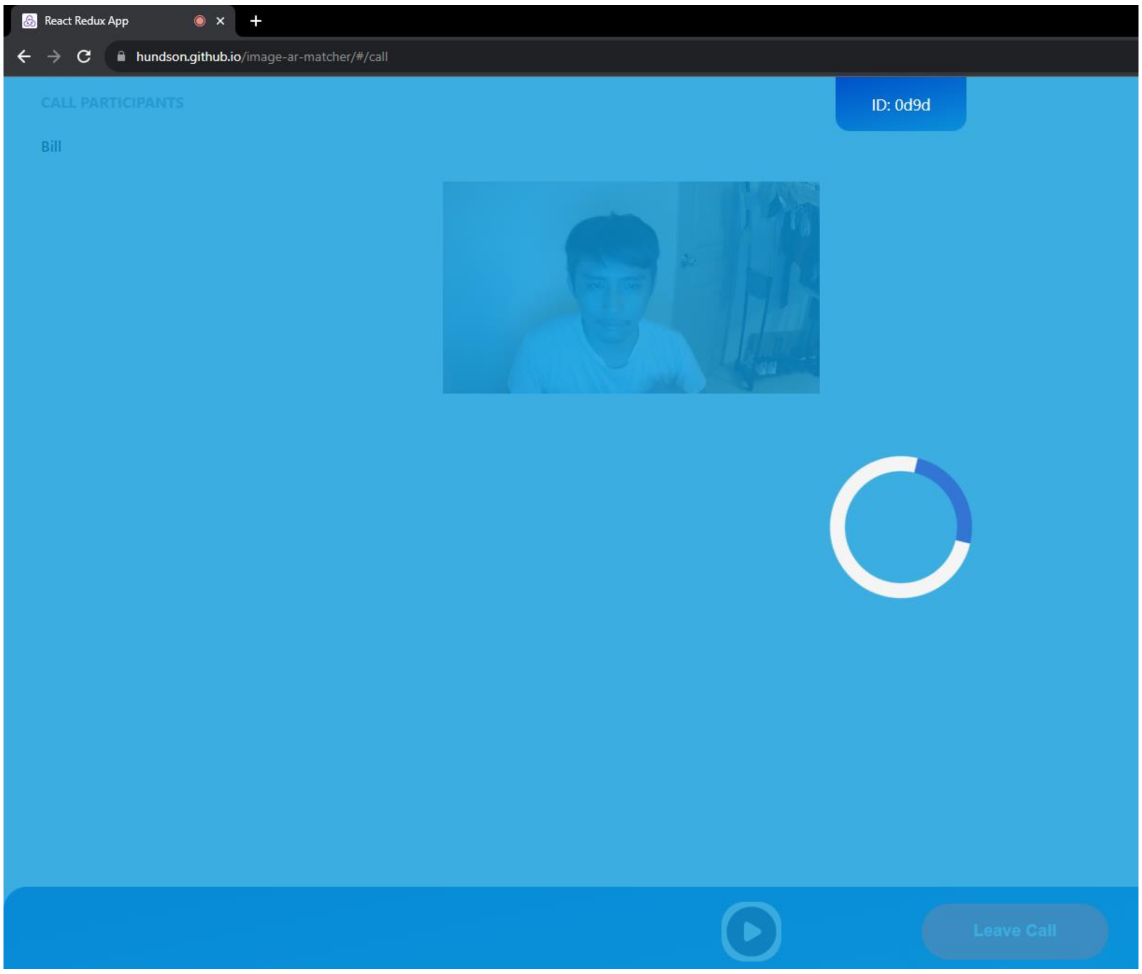


Figure 5.4.6 Page loading when creating a new call session as host

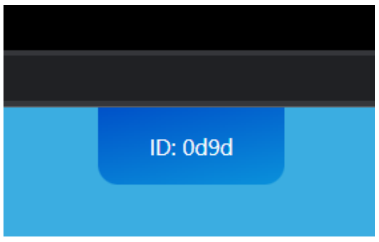


Figure 5.4.7 Call session ID to be shared with the connecting peer user

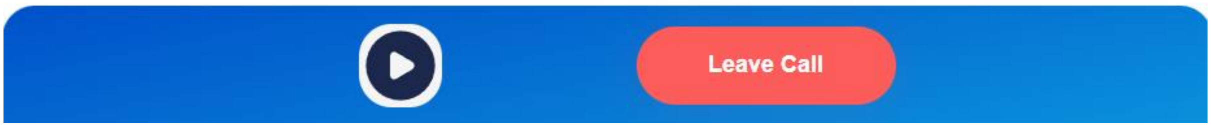


Figure 5.4.8 Start Segmentation button (left) to be clicked once peer has joined call session

Figure 5.4.9 shows the call session page UI once a peer user has connected and the segmentation process has been started. On each initial run, the host video is set as the background element while the peer video is set as the source for the segmentation process. This creates the effect of augmenting the peer user onto the video environment of the host user. On the left most side are the names of the call participants. In the buttons container at the bottom and starting from the left is the Custom Background button, Screenshot button, Swap Video button, Restart Segmentation button, and the Leave Call button.

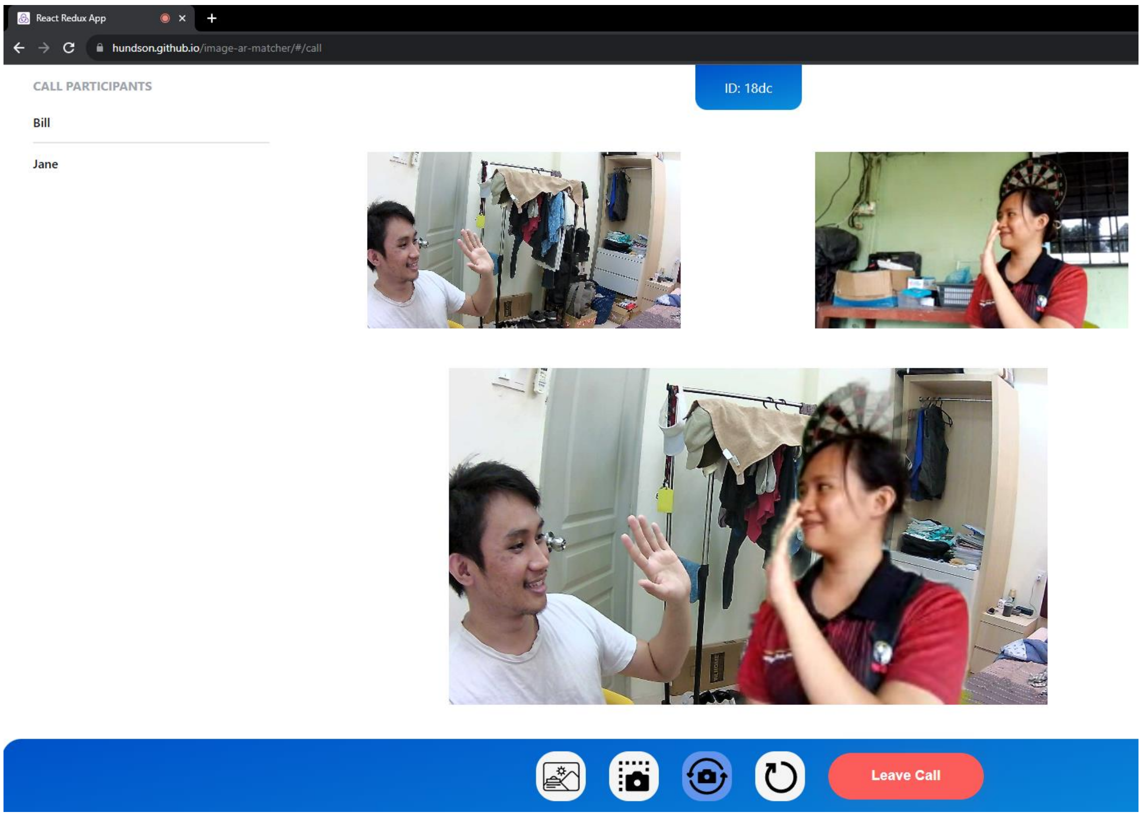


Figure 5.4.9 Call session page with the generated AR output video

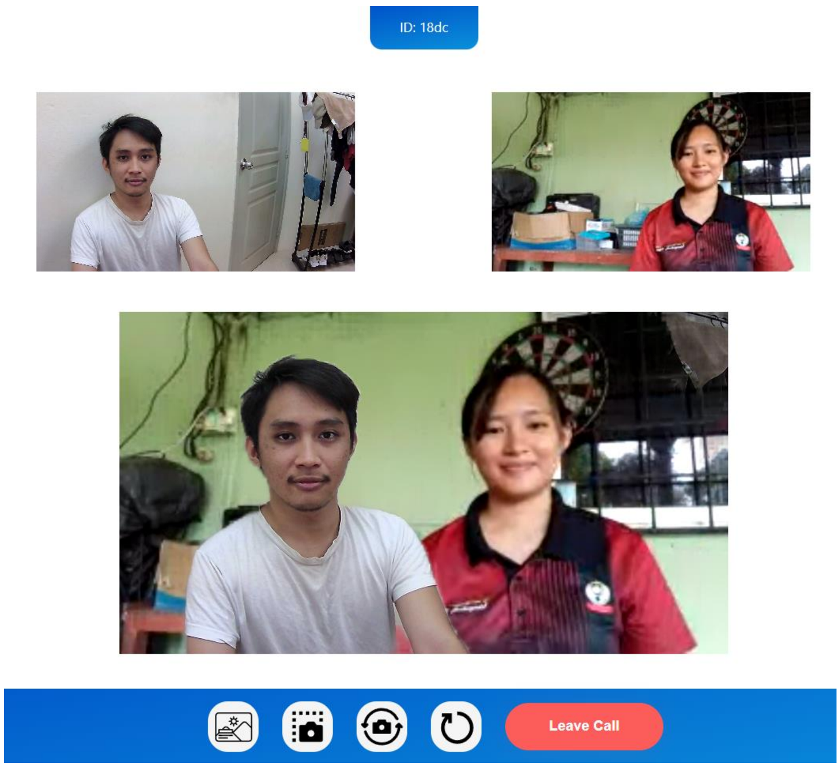


Figure 5.4.10 Swap Video function to swap video background and segmentation source

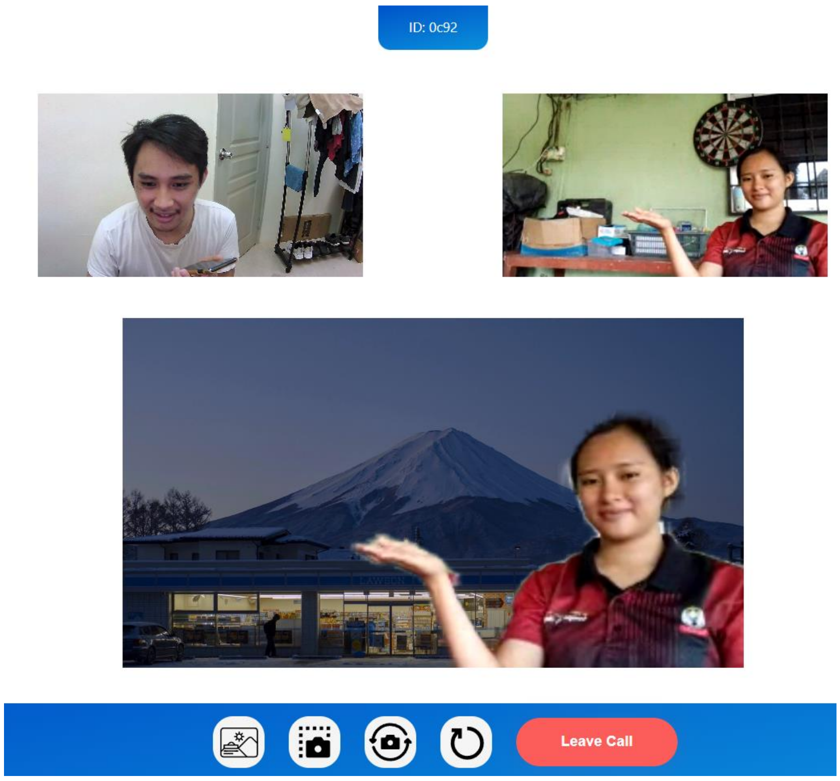


Figure 5.4.11 Custom Background function to upload a custom background image

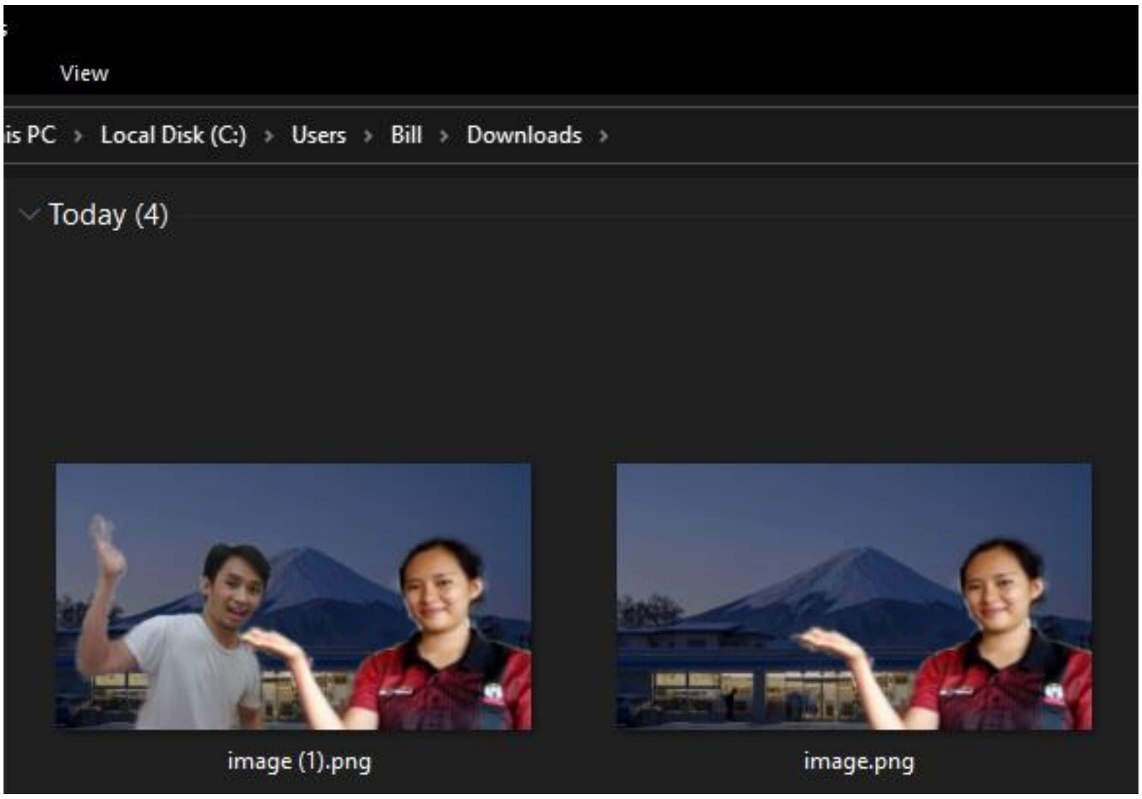


Figure 5.4.12 Image files of the AR output video saved using the Screenshot function

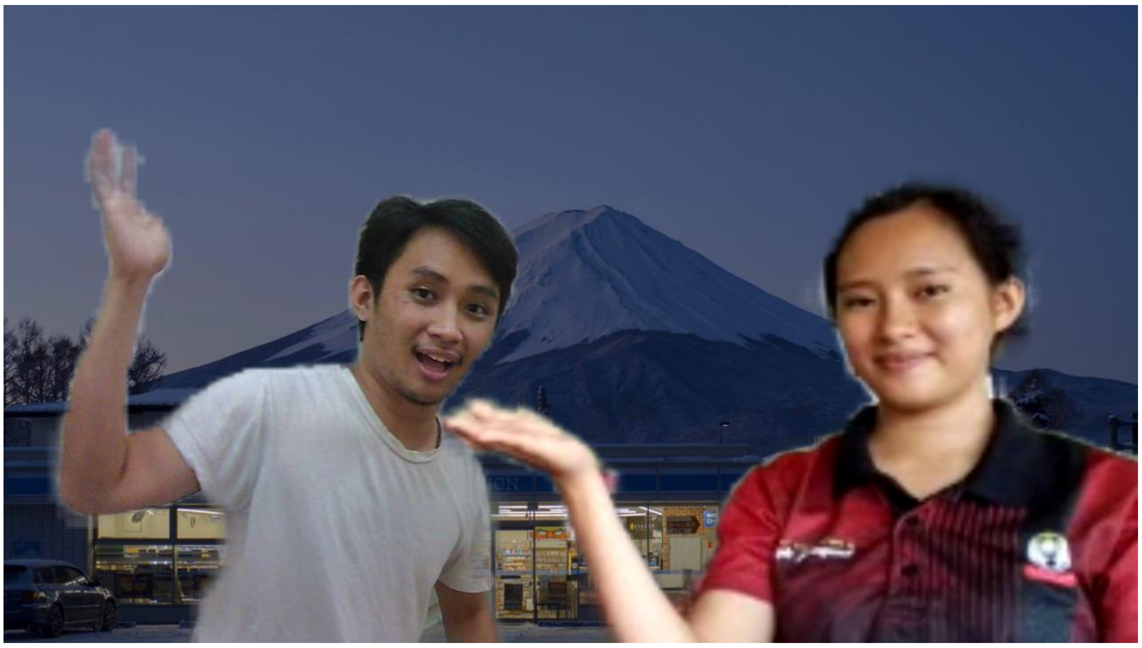


Figure 5.4.13 Saved image of the AR output video

5.5 Implementation Issues and Challenges

One of the main drawbacks of the system is due to the limitations of the MediaPipe Selfie Segmentation model itself. In the previous Figure 5.5.10, it can be seen that only the peer user is augmented onto the output video when the Custom Background mode is selected. This is due to the Selfie Segmentation Model's limitation of only being able to target one video source for processing at a time. The model is not able to perform segmentation for multiple video sources concurrently, meaning segmentation cannot be executed simultaneously for both host and peer videos when using a custom background and they therefore cannot be augmented together onto the output video at the same time. The result in Figure 5.5.12 is achieved by firstly saving an image of the segmented peer user with the custom background using the Screenshot function. Then, the saved image is then reused as the custom background. However, the host user is instead the one who gets their video processed for segmentation and have their body augmented onto the custom background. This workaround, however, is tedious. Other than that, there are also system stability issues arising from the implementation of the Selfie Segmentation Model where the application would sometimes fail to run the segmentation process. The Reset Segmentation function was added to counter this by killing and replacing the currently running segmentation process instance when the corresponding button is clicked. Moreover, there are also occasions where the system operation would gradually show signs of unresponsiveness. Although the Selfie Segmentation Model is only able to process a single video source at one time, it is still possible for the system to accidentally duplicate multiple running instances of the same segmentation process. This unwanted behavior causes the application to visibly slow down. The exact cause for this issue remains undetermined with multiple targeted testing done not being able to reproduce the issue. However, this issue is, at the very least, a rare occurrence. In regards to this issue and the one mentioned beforehand, the general factor behind them can only be due to either the experimental nature of the Selfie Segmentation Model itself or are direct results of the shortcomings in how the programming for this system has been implemented. In relation to that statement, the system can only perform at its best when using the Google Chrome browser on a desktop computer. Unfortunately, there was not much effort that was able to be put into optimizing the system to run for different devices and browsers. The system's UI is not mobile responsive so although the application can be run on mobile devices, the user might not be able to interact with the UI elements. Lastly, audio was also not been able to have captured together with the users' video streams.

5.6 Concluding Remark

Overall, the implementation process undergone for the system had been smoothly carried out without any major issues. The client-side application was successfully hosted through GitHub Pages while the server-side program was hosted using the Render cloud hosting platform. Although there is much room for improvement to further optimize system performance and polish the application UI for enhanced user experience, the project has achieved its goal of implementing a working web-based system delivering the intended use cases and functionalities. Much of the time and effort made throughout the project's lifetime were put into extensive prototyping, bug fixing, as well as developing workarounds for problems and much less so was able to be allocated for optimizing the UI and operations performance. Due to time constraint, there was much more emphasis put on working to fully deliver the intended system functionality and thus the user experience aspect was left slightly neglected. Nevertheless, the system's implementation overall has still been very satisfactory.

Chapter 6
System Evaluation and Discussion

6.1 Web Browser Compatibility Testing

Web Browser	Platform	Remark	Support
Google Chrome	Windows	System operation runs perfectly as intended with all functionalities working flawlessly.	Full
Firefox		System operation runs perfectly as intended with all functionalities working flawlessly.	Full
Microsoft Edge		System operation runs perfectly as intended with all functionalities working flawlessly.	Full
Opera/Opera GX		System operation runs perfectly as intended with all functionalities working flawlessly.	Full
Brave		System operation runs perfectly as intended with all functionalities working flawlessly.	Full
DuckDuckGo		System operation runs perfectly as intended with all functionalities working flawlessly.	Full
Google Chrome	Android	System is able to run with the user able to connect to peer user. System is able to run segmentation model and generate AR output video. However, Swap Video and Restart Segmentation button functions are not executable. UI is not fully responsive to scale dynamically to fit the sizes for the smaller screen of the mobile device.	Semi-supported

Table 6.1.1 Compatibility testing for running the system on different web browsers

6.2 Project Challenges

This project has been in the works over the period of a few years starting from the year 2020. The biggest challenge faced during the course of researching for and developing the project is having to work with the lack of guidelines and proper documentations. This is due to the nature of the project itself being fairly new or is at least less commonly explored. Moreover, this has also been further exaggerated by the fact that the MediaPipe Selfie Segmentation Model implemented in this project system is itself only very recently developed and was made as a result of the huge spike in popularity of video calling applications during the recent COVID-19 pandemic. There are very few projects that could be found which utilizes this model. There are also fewer numbers of systems developed that could be found publicly which incorporates both aspects of augmented reality and remote video communications together. In fact, the extensive researching that was done for this project was not able to find any specifically web-based application that blends AR with real-time video communications. Therefore, it can be assumed that this project is one which is quite unique. It is due to this reason that the project has taken a considerably long time to develop as there are very few resources found that could be used as a point of reference and thus many of the implementations in this system was achieved through numerous experimentations as well as countless trials and errors. Another major challenge in working on the project is working with the Selfie Segmentation Model itself. Due to the limitations of the model that was discussed earlier in the previous chapter, some sort of workaround was necessary to develop in an attempt to mitigate the model's shortcoming and still be able to deliver the intended system functionality albeit not to the degree as it was initially hoped to be. It still would have been ideal had the system been able to simultaneously run the segmentation process on both host and peer user's videos. Furthermore, there are also workarounds needed to be made to ensure the system operation during the call session was synchronized on both host and peer clients' end. Due to how the Selfie Segmentation Model works, the segmentation process is done locally by the client machine meaning that clients do not stream to each other the actual processed AR output video. In other words, each host and peer client will run the segmentation process as well as trigger any of the button functions individually on their own machine. In order to synchronize which functions are triggered between the two remote users, a workaround was implemented where the signaling server would emit an event to the user anytime one of them triggers a function. Upon receiving the emitted event, the receiving end would also trigger the same function, keeping users in sync.

6.3 Objectives Evaluation

Objective	Evaluation	Result
To develop a peer-to-peer video streaming module.	The completed system has successfully been able to facilitate real-time video communication between two remote users, enabling an initial user to host a call session and generate a unique ID with which they may share to another user to use so that they may join the call session.	Fulfilled
To develop a module for video capture and perform image segmentation.	The completed system is able to access the client device's camera, capture a video stream of the user, identify the human subject, and perform segmentation to separate between the foreground (human subject) and background.	Fulfilled
To integrate video streaming and image segmentation modules.	The full integration of both the video streaming and image segmentation modules was successfully done through the implementation of the completed system. The system is able to take either the host or peer video as the media source, perform the segmentation process on the selected video source, and have the user's segmented image augmented onto the video of the other user.	Fulfilled
To deploy the augmented reality matching application to the web.	The completed system has been successfully deployed to the web and is public accessible through the given URL. The system structure is split into the client and server ends with the client hosted through GitHub Pages and the server hosted using the Render cloud hosting platform.	Fulfilled
To develop further enhancements for application functionalities.	Additional features were added to the system to enhance its functionality. However, the Custom Background function was not able to fully deliver its intended purpose as the system is not able to simultaneously run segmentation on multiple video sources. The system is then able to augment only one of either users onto the video when the Custom Background mode is activated.	Partially fulfilled

Table 6.3.1 Project Objectives Evaluation

6.4 Concluding Remark

The overall evaluation of the system establishes that the project was able to produce immensely satisfying results despite some of the shortcomings. Generally speaking, the project has been hugely successful as development work was able to develop and implement nearly the entirety of the major system functionalities that the project intended to deliver. Overall, there was only one significant functionality that the final system was unfortunately not able to deliver which is the simultaneous processing of both host and peer user videos for segmentation and concurrent augmentation of both remote users onto a custom background environment. Despite that, it should be understood that unless a different image segmentation machine learning model had been used in place of the MediaPipe Selfie Segmentation Model, nothing could have been done to change this outcome as the system was handicapped due to the model's limitation. Nevertheless, the final system's implementation has still been able to realize the most important deliverable of this entire project which is the development of a video conferencing system capable of generating image-to-augmented reality matching with user videos. The completed system managed to deliver a functionality to augment the image of a user to the video environment of another user in real-time. Taking into account the unique nature of this system, the results that the project was able to produce is nothing short of exceptional.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

The idea for this project was immensely inspired by the spike in popularity for video conferencing applications due to the COVID-19 pandemic. Emerging threats from that time has seen a huge shift in the work culture of the job industry. Companies and government bodies have enacted the work-from-home policy in interest of public safety. Workers have since then opted to utilize video conferencing applications in order to conduct their daily communications while working remotely. Motivation for developing the completed system stems from the recognition that there are various potential improvements that can be made to further enhance the video conferencing experience through harnessing the technological power of AR.

Development of the completed system brings about a unique solution for AR experience not yet offered in any known application. The biggest advantage the completed system intends to deliver is the functionality to render human images from the video of conference participants into a single video scene or virtual environment. This capability is in contrast to most of the current implementations where user videos are only rendered within their own separate video scenes. The completed system brings about a higher degree of realism where users are made to appear as if they are actually in the same physical space standing right next to each other. This project serves as a proof of concept that is hoped to be further experimented on and would see a better implementation in the future as web development tools for AR matures over time.

Nonetheless, this project serves as an attempt to work on a practical implementation of the web tools and frameworks currently available at the time of writing. The MediaPipe Image Segmentation Model, for instance, was developed as a result of the COVID-19 pandemic. This project has provided a good opportunity to put this algorithm into practical use. Nowadays, more and more powerful web-based applications are being developed thanks to the ever-growing innovations in web tools and framework and the completed system is a testimony to this. Such applications were previously only possible develop as native platform applications. Current tools are bringing the AR experience closer to be implemented on the web.

7.2 Recommendation

During the course of working on the project, MediaPipe has released a new version of their Selfie Segmentation Model. The latest release has compiled the Selfie Segmentation Model together alongside their other image segmentation machine learning models into a single collection named the MediaPipe Image Segmenter. This latest iteration of their segmentation model has brought numerous added functionalities. For instance, the newer model has now managed to incorporate detection for individual body parts such as the arms, legs, torso, etc. This feature alone brings about vastly bigger numbers of potential use cases. The reason that this project did not implement the newer release of this model is because of how different the programming implementation is between the new and old model. It was decided that the project would stick to utilizing the older model as time as well as other constraints did not allow for making any major overhaul to how the system was originally implemented for Final Year Project I. For anyone in the future who are interested in rebuilding this system, it is highly encouraged that they would look into utilizing the latest machine learning models for image segmentation that is available during that time. The Selfie Segmentation Model used specifically in this project in in the experimental state and thus is highly unstable making it unsuitable for any production builds. It is expected that machine learning models for image segmentation will continue to improve over time and increase in numbers into the future.

REFERENCES

[1] H. Kato and M. Billinghurst, "Marker tracking and HMD calibration for a video-based augmented reality conferencing system," Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99), San Francisco, CA, USA, 1999, pp. 85-94, doi: 10.1109/IWAR.1999.803809.

[2] H. Kato, M. Billinghurst, K. Morigana, and K. Tachibana, “The Effect of Spatial Cues in Augmented Reality Video Conferencing” Proceedings of the Ninth International Conference on Human-Computer Interaction, 2001, vol. 2, pp. 478-481

[3] O. G. Guleryuz and A. Kalker, "Visual conditioning for augmented-reality-assisted video conferencing," 2012 IEEE 14th International Workshop on Multimedia Signal Processing (MMSP), Banff, AB, Canada, 2012, pp. 71-76, doi: 10.1109/MMSP.2012.6343418.

[4] S. Kumar, S. Saxena, and A. K. Singh, “Virtual presence via mobile,” U.S. Patent 9 024 997, May 5, 2015.

[5] Google, “Getting started with WebRTC,” WebRTC, 28-May-2019. [Online]. Available: https://webrtc.org/getting-started/overview. [Accessed: 30-Jan-2023].

[6] MediaPipe, “Image segmentation task guide,” Google, 31-Mar-2023. [Online]. Available: https://developers.google.com/mediapipe/solutions/vision/image_segmenter. [Accessed: 31-Mar-2023].

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y5S2	**Study week no.: 3 (4/7/2023)**
Student Name & ID: Bill Hundson David 18ACB01007	
Supervisor: Syed Muhammad Bin Syed Omar	
Project Title: Web-Based Image-Augmented Reality (AR) Matching Generator	

1. WORK DONE - Conducted extensive planning to decide how work from FYP 1 should be carried over when going into FYP 2
2. WORK TO BE DONE - Review the newly released version of the MediaPipe Selfie Segmentation Model - Discuss with supervisor on what should be done for FYP 2
3. PROBLEMS ENCOUNTERED - None
4. SELF EVALUATION OF THE PROGRESS - Need to pick up the pace with the work progress

[illegible] [illegible]

Supervisor's signature Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y5S2	**Study week no.: 4 (11/7/2023)**
Student Name & ID: Bill Hundson David 18ACB01007	
Supervisor: Syed Muhammad Bin Syed Omar	
Project Title: Web-Based Image-Augmented Reality (AR) Matching Generator	

1. WORK DONE - Reviewed the newly released version of the MediaPipe Selfie Segmentation Model
2. WORK TO BE DONE **-** Start development work to integrate the image segmentation module with the video calling application
3. PROBLEMS ENCOUNTERED **-** Review on the new version of the Selfie Segmentation Model finds that its implementation far too different from the old model **-** It is decided that the project will stick to implementing the old segmentation model
4. SELF EVALUATION OF THE PROGRESS **-** System development progress needs to speed up

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y5S2	**Study week no.: 6 (25/7/2023)**
Student Name & ID: Bill Hundson David 18ACB01007	
Supervisor: Syed Muhammad Bin Syed Omar	
Project Title: Web-Based Image-Augmented Reality (AR) Matching Generator	

1. WORK DONE - Managed to renew source code design for the image segmentation module - Started work to integrate segmentation into the video calling app
2. WORK TO BE DONE **-** Continue developing the segmentation function for the system
3. PROBLEMS ENCOUNTERED **-** The application is still facing numerous bugs with the segmentation function
4. SELF EVALUATION OF THE PROGRESS **-** Work progress was shown to supervisor and feedback was received

[illegible]
Supervisor's signature

Bill
Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y5S2	**Study week no.: 8 (8/8/2023)**
Student Name & ID: Bill Hundson David 18ACB01007	
Supervisor: Syed Muhammad Bin Syed Omar	
Project Title: Web-Based Image-Augmented Reality (AR) Matching Generator	

1. WORK DONE

- Successfully developed the segmentation function for system
- System is now able to perform segmentation for user videos and render the AR output

2. WORK TO BE DONE

- Continue development work to add the custom background and video swapping functionalities
- Continue optimizing system and fixing bugs
- Start working on writing the report

3. PROBLEMS ENCOUNTERED

- There are still various bugs with the system that needs to be addressed especially when running the segmentation functionality

4. SELF EVALUATION OF THE PROGRESS

- Even though project is progressing steadily, there is still much work to be done
- Need to hasten progress as deadlines are nearing

[illegible] [illegible]

Supervisor’s signature   Student’s signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y5S2	**Study week no.: 10 (22/8/2023)**
Student Name & ID: Bill Hundson David 18ACB01007	
Supervisor: Syed Muhammad Bin Syed Omar	
Project Title: Web-Based Image-Augmented Reality (AR) Matching Generator	

1. WORK DONE

- Report writing is in progress
- Successfully developed the custom background and video swapping functions

2. WORK TO BE DONE

- Need to come up with a workaround to deal with the limitations of the segmentation model

3. PROBLEMS ENCOUNTERED

- The limitations of the Selfie Segmentation Model had just been realized

4. SELF EVALUATION OF THE PROGRESS

- Satisfied with project progress
- Supervisor is satisfied with the developed system as well

[illegible]

Supervisor's signature

Bill

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y5S2	**Study week no.: 13 (12/9/2023)**
Student Name & ID: Bill Hundson David 18ACB01007	
Supervisor: Syed Muhammad Bin Syed Omar	
Project Title: Web-Based Image-Augmented Reality (AR) Matching Generator	

1. WORK DONE

- Managed to add the restart segmentation function as a solution for working around the segmentation model's limitation
- Managed to fix the last remaining major bugs with the system
- Development for this project's system has been completed

2. WORK TO BE DONE

- Need to finish up writing the report

3. PROBLEMS ENCOUNTERED

- None

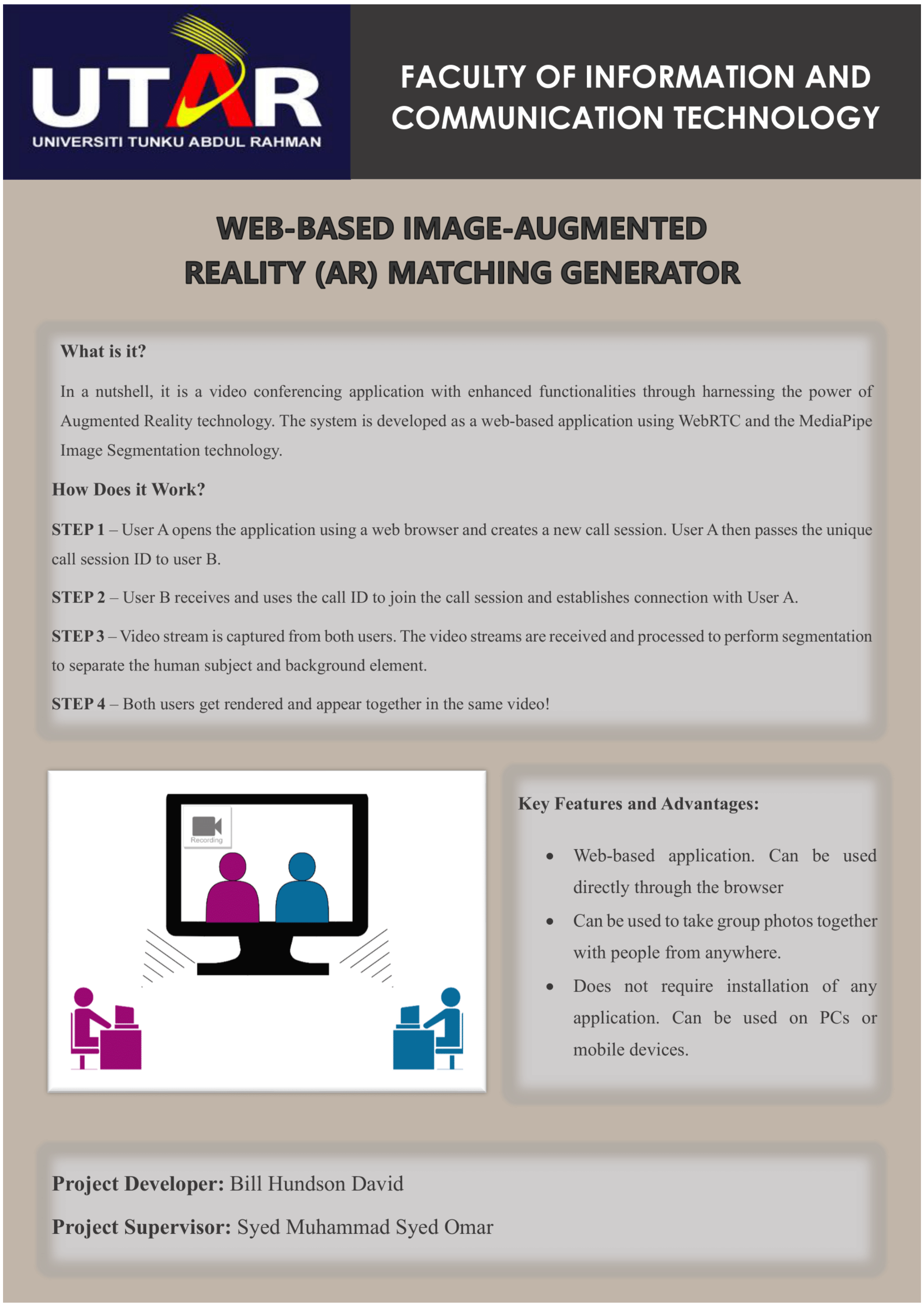
4. SELF EVALUATION OF THE PROGRESS

- Met with supervisor one final time before FYP submission

[illegible]
Supervisor's signature

[illegible]
Student's signature

POSTER



UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

WEB-BASED IMAGE-AUGMENTED REALITY (AR) MATCHING GENERATOR

What is it?

In a nutshell, it is a video conferencing application with enhanced functionalities through harnessing the power of Augmented Reality technology. The system is developed as a web-based application using WebRTC and the MediaPipe Image Segmentation technology.

How Does it Work?

STEP 1 – User A opens the application using a web browser and creates a new call session. User A then passes the unique call session ID to user B.

STEP 2 – User B receives and uses the call ID to join the call session and establishes connection with User A.

STEP 3 – Video stream is captured from both users. The video streams are received and processed to perform segmentation to separate the human subject and background element.

STEP 4 – Both users get rendered and appear together in the same video!

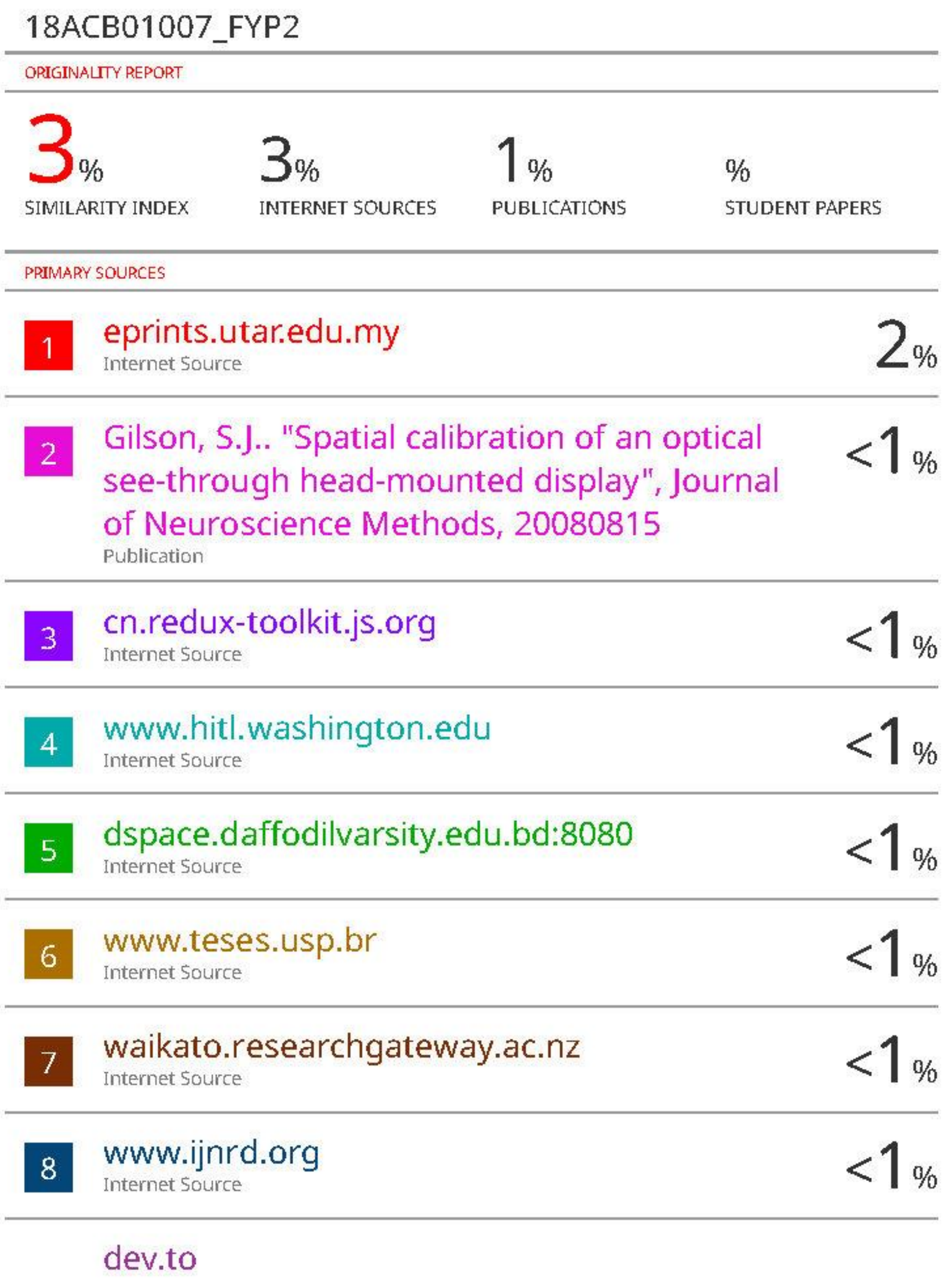
Key Features and Advantages:

- Web-based application. Can be used directly through the browser
- Can be used to take group photos together with people from anywhere.
- Does not require installation of any application. Can be used on PCs or mobile devices.

Project Developer: Bill Hundson David

Project Supervisor: Syed Muhammad Syed Omar

PLAGIARISM CHECK RESULT



18ACB01007_FYP2

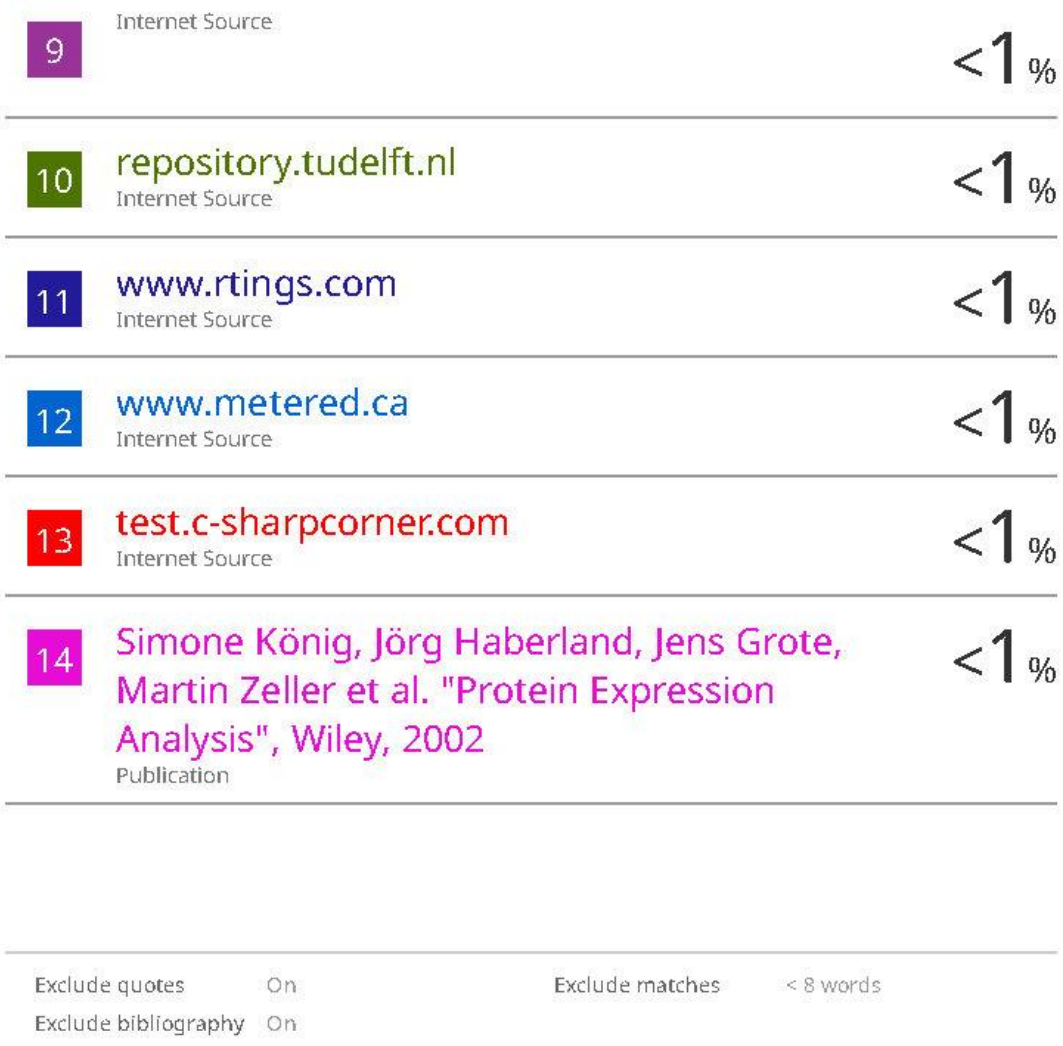
ORIGINALITY REPORT

3%	3%	1%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	eprints.utar.edu.my Internet Source	2%
2	Gilson, S.J.. "Spatial calibration of an optical see-through head-mounted display", Journal of Neuroscience Methods, 20080815 Publication	<1%
3	cn.redux-toolkit.js.org Internet Source	<1%
4	www.hitl.washington.edu Internet Source	<1%
5	dspace.daffodilvarsity.edu.bd:8080 Internet Source	<1%
6	www.teses.usp.br Internet Source	<1%
7	waikato.researchgateway.ac.nz Internet Source	<1%
8	www.ijnrd.org Internet Source	<1%

dev.to



9	Internet Source	<1%
10	repository.tudelft.nl Internet Source	<1%
11	www.rtings.com Internet Source	<1%
12	www.metered.ca Internet Source	<1%
13	test.c-sharpcorner.com Internet Source	<1%
14	Simone König, Jörg Haberland, Jens Grote, Martin Zeller et al. "Protein Expression Analysis", Wiley, 2002 Publication	<1%

Exclude quotes	On	Exclude matches	< 8 words
Exclude bibliography	On		

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Bill Hundson David
ID Number(s)	18ACB01007
Programme / Course	Bachelor of Information Systems (Honours) Information Systems Engineering
Title of Final Year Project	Web-Based Image-Augmented Reality (AR) Image Matcher

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 3 % **Similarity by source** Internet Sources: 3 % Publications: 1 % Student Papers: 0 %	
Number of individual sources listed of more than 3% similarity: 0	
Parameters of originality required and limits approved by UTAR are as Follows: **(i) Overall similarity index is 20% and below, and** **(ii) Matching of individual sources listed must be less than 3% each, and** **(iii) Matching texts in continuous block must not exceed 8 words** ***Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.***	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

[illegible]

Signature of Supervisor	Signature of Co-Supervisor
Name: Syed Muhammad Syed Omar	Name:
Date: 15/09/2023	Date:

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	18ACB01007
Student Name	Bill Hundson David
Supervisor Name	Syed Muhammad Bin Syed Omar

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

[illegible]
____
(Signature of Student)
Date: 15/9/2023