`

Computer Science Department

Comfy View

Presented By:

Reham Elaraby **Roqaya Ashraf**
MennatAllah EssamEldin **Yasmine Osman**
Youssef Sayed

Supervised By:
Professor Dr: Mahmoud GadAllah

Assistant Supervised By:
Eng: Mennatullah Ramadan

2023-2024

`

Acknowledgment: -

Many thanks to everyone who contributed to the completion of this project to be accomplished.

We would like to thank the administration of our university Modern Academy in Maadi presented in…

Professor Dr. Nabil Debes

To provide all necessary educational materials and supplies to assist students in Build their academic and professional lives and complete those projects.

Professor Dr. Mahmoud GadAllah

To give us the honor of being under her supervision and her wise advice for a better achievement and the positive energy that helped us achieve it.

Eng. Mennatullah Ramadan

To give us time and effort whenever we need it to get support, information and assistance.

`

Abstract:

In an age where technology is transforming lives, the Comfy View project seeks to empower individuals with visual impairments through cutting-edge assistive technologies.

This all-encompassing solution combines an embedded system with a website and a Flutter mobile application, enhanced with AI capabilities, to deliver real-time object detection, description, and navigation assistance for the blind.

The embedded system, featuring ultrasonic sensors and a temperature detector, identifies objects, detects hazards, and captures the surroundings via a wearable glove.

This information is transmitted to the website and mobile app, where AI technology provides object descriptions and navigation feedback. Inspired by the need for independence and autonomy in the blind community, this project integrates multiple components to significantly improve the quality of life.

Similar initiatives, such as the Anora Smart Glove, have demonstrated the value of multifunctional solutions for the visually impaired. With approximately 1.3 billion people globally experiencing vision impairments, the Comfy View project represents a stride towards inclusive technology, offering not just assistance, but empowerment for those with visual disabilities.

`

Table of Contents

Acknowledgment: - **2**

Chapter 1 (Introduction) **10**

1.1) Overview **11**

1.2) Problem Definition **11**

1.3) Solution Approach **11**

1.4) Scope **12**

1.5) Objectives **12**

1.6) System Description **13**

Chapter 2 (Survey) **14**

2.1) Literature Survey **15**

2.1.1) Mobile assistive technologies for the visually **15**

2.1.2) Smart Assistive Arm Applications Beyond **15**

2.1.3) Wearable Assistive Technologies **15**

2.1.4) Robotic Assistive Arms for Navigation **16**

2.2) Related works **16**

2.2.2) Bone Conduction Headphones **18**

2.2.4) Braille Displays **20**

2.3) Conclusion **22**

Chapter 3 (Software life cycle and Analysis) **23**

Planning Phase **24**

Analysis Phase **24**

Design Phase **24**

Coding Phase **25**

Testing Phase **25**

Implementation and Maintenance Phase **25**

3.2) Planning **27**

3.3) Requirements Analysis **28**

3.3.1) Requirements Analysis **28**

3.3.2) System requirements **28**

3.4) Software requirements specification **29**

`

3.4.1) Functional requirements **29**
3.4.2) Nonfunctional requirements **30**
3.5) System Users **31**
3.6) Tools **31**
Mobile Application using Visual Studio & Android Studio **31**
Databases using Firebase **32**
Embedded Part using Arduino IDE **32**
Artificial Intelligence using Anaconda Environment **32**
3.7) System Design **33**
3.7.1) Entity Relationship Diagram (ERD) **34**
3.7.2) Use Case Diagram **35**
3.7.3) Class Diagram **36**
3.7.5) Activity Diagram **38**
3.8) System Implementation **39**
3.9) System testing **39**
3.10) Documentation **39**
Chapter 4 (Tools) **40**
4.1) Overview **41**
4.2) Programming Languages we have used **41**
4.3) Basic work **42**
Visual Studio Code **42**
Android Studio **43**
4.4) Mobile Content **44**
Flutter using dart **44**
4.5) Web content **45**
Node.js **45**
React **46**
4.6) Embedded Content **47**
Embedded C **47**
The ESP/ISF (Embedded Systems Programming and Industrial Systems Fundamentals) **48**
The Arduino IDE (Integrated Development Environment) **49**

`

4.7) Database (Firebase) 50
4.8) AI content 51
Python 51
Anaconda (Python distribution) 52
Chapter 5 (System Implementation) 54
5.1) Embedded Implementation 55
5.2) AI implementation 65
5.2.1) Object Detection Model 65
5.2.2) face recognition model 70
5.3) Back-end implementation 74
5.4) Back-end integration with Front-end 83
5.5) Front-end Implementation 88
5.6) Mobile application implementation 92
5.6.1) UI Implantation 92
5.6.2) Logic Implantation 98
Chapter 6 (Conclusion & Future work) 113
6.1) Conclusion 114
6.2) Future Work 115
6.3) References 116

`

List of figures: -

Figure 1 (Ai Smart Glasses)..... 17
Figure 2(Bone Conduction Headphones)..... 18
Figure 3(Braille Keyboards) 19
Figure 4 (Braille Displays)..... 20
Figure 5(Audio Labelers)..... 21
Figure 6(SDLC cycle) 26
Figure 7 (ERD)..... 34
Figure 8(Use Case Diagram) 35
Figure 9(Class Diagram) 36
Figure 10(Sequence Diagram) 37
Figure 11 (Activity Diagram)..... 38
Figure 12 (VS code)..... 42
Figure 13(Android Studio)..... 43
Figure 14(Flutter & Dart) 44
Figure 15(Node.js)..... 45
Figure 16(React) 46
Figure 17(C)..... 47
Figure 18(ESP-IDF) 48
Figure 19(Arduino)..... 49
Figure 20(Fire Base)..... 50
Figure 21 (Python) 51
Figure 22 (Anaconda)..... 52
Figure 23 (Arduino Uno) **55**
Figure 24 (Ultrasonic Sensor) **56**
Figure 25 (LM 35)..... **56**
Figure 26 (Simulation 1) **57**
Figure 27(Serial monitor) **57**
Figure 28 (Arduino code snippet 1) **58**
Figure 29 (Arduino code snippet 2) **58**
Figure 30 (ESP32 CAM) **59**
Figure 31(Code snippet3 initwifi) **59**
Figure 32(Code snippet4 initLittleFs)..... 60
Figure 33(Code snippet5 initCamera)..... 60
Figure 34(Code snippet6 initTime)..... 61
Figure 35(Code snippet7 saveDataTofirestore)..... 62

`

Figure 36(Code snippet8 GPS) 62
Figure 37 RehamCollectiononFirebase 63
Figure 38 (GPS Neo 6) 63
Figure 39(Simulation 2) 64
Figure 40(Collected data-set 1) 65
Figure 41(Collected data-set 2) 65
Figure 42 (Processed data-set 1) 66
Figure 43(Processed data-set 2) 66
Figure 44 (CNN model) 67
Figure 45 (CNN accuracy) 67
Figure 46 (CNN convolution detail) 68
Figure 47(Object detection 1) 68
Figure 48(Object detection 4) 69
Figure 49 (Object detection 2) 69
Figure 50 (Object detection 3) 69
Figure 51(Dlib) 70
Figure 52(OpenCV) 70
Figure 53(Code snippet Face recognition) 71
Figure 54 (Face recognition result) 73
Figure 55(Code snippet Backend findUserByEmail) 74
Figure 56 (Code snippet Backend signIn) 74
Figure 57 (Code snippet Backend signUp) 74
Figure 58 (Code snippet Backend CheckImage) 75
Figure 59 (Code snippet Backend expose) 76
Figure 60 (Code snippet Backend Search) 76
Figure 61(Code snippet Backend Express.js) 77
Figure 62(Code snippet Backend Express.js2) 78
Figure 63(Code snippet Backend. checkAuthUser) 79
Figure 64(Firebase collection results) 80
Figure 65(Class model Image) 81
Figure 66 (Firebase Collection user) 81
Figure 67 (Class model User) 82
Figure 68 (Backend integration 1) 83
Figure 69 (Backend integration 2) 84
Figure 70 (Backend integration 3) 84
Figure 71 (React integration using blob) 85
Figure 72 (send photo to Object detection Api) 86
Figure 73 (Face recognition code) 87

`

Figure 74 (Website About page) 88
Figure 75 (Website Result page) **88**
Figure 76 (Website Signup page) **89**
Figure 77 (Website Login page) **89**
Figure 78 (Website Personal information page) **90**
Figure 79 (Website Lost found page) **90**
Figure 80 (Website Landing page) **91**
Figure 81 (Website E-commerce page) **91**
Figure 82 (Mobile application UI 1) **92**
Figure 83 (Mobile application UI 2) **93**
Figure 84 (Mobile application UI 3) **94**
Figure 85 (Mobile application UI 4) **95**
Figure 86 (Mobile application UI 5) **96**
Figure 87 (Mobile application UI 6) **97**
Figure 88 (Code snippets Flutter Pubspec.yaml) **98**
Figure 89(Code snippets Flutter firebase initialization) **99**
Figure 90 (Code snippets Flutter Bloc Observer) **99**
Figure 91 (Code snippets Flutter initialize dio) **99**
Figure 92 (Code snippets Flutter MultiBlocProvider) **100**
Figure 93 (Code snippets Flutter Theming) **100**
Figure 94(Code snippets Flutter Routing) **101**
Figure 95 (Code snippets Flutter Firebase integration) **102**
Figure 96(Code snippets Flutter Model class user) **103**
Figure 97(Code snippets Flutter Model class Result) **104**
Figure 98(Code snippets Flutter Dio helper setup) **105**
Figure 99(Code snippets Flutter integrates with API userLogin) **106**
Figure 100(Code snippets Flutter integrates with API userRegister) **107**
Figure 101(Code snippets Flutter integrates with API face recognition) **108**
Figure 102(Code snippets Flutter integrates with API Location) **109**
Figure 103(Code snippets Flutter getData from firebase) **111**
Figure 104(Code snippets Flutter display data) **112**

`

Chapter 1
(Introduction)

`

1.1) Overview

The project aims to develop a wearable assistive arm to aid visually impaired individuals in navigating their surroundings, addressing the global issue of vision impairment affecting over 2.2 billion people worldwide. By combining various advanced technologies, the Smart Assistive Arm offers a comprehensive solution for enhancing independence and safety in everyday life for the blind community.

1.2) Problem Definition

Blind and visually impaired individuals face significant challenges in navigating their surroundings safely and independently, including mobility limitations, limited environmental awareness, and difficulty locating specific objects. Existing assistive technologies often lack real-time data visualization and user-friendly interfaces, leading to inefficiencies and frustration. Additionally, these technologies may not adequately address the diverse needs of users, requiring a more holistic and adaptable solution.

1.3) Solution Approach

The project focuses on developing a wearable assistive arm equipped with advanced sensors and haptic feedback technology to enhance spatial awareness and obstacle detection. The gloves are designed to be intuitive, portable, and user-friendly, promoting mobility, confidence, and independence for visually impaired users. The solution includes AI-driven features for object detection and face recognition, a mobile application for real-time interaction, and a backend system for seamless integration and data management.

`

1.4) Scope

The project aims to create a comprehensive solution for visually impaired individuals, encompassing:

- A wearable assistive arm with embedded sensors.
- AI models for object detection and face recognition.
- A mobile application for user interaction and feedback.
- A backend system for integration and data management, utilizing a Firebase database.
- A website interface for additional accessibility and configuration options.

1.5) Objectives

1. User-Centric Design: Design the gloves with a focus on user comfort, adjustability, and ease of use.
2. Portability and Compactness: Ensure the arm is lightweight, portable, and easy to carry.
3. Simplified User Interface: Implement an intuitive interface that requires minimal training to operate.
4. Plug-and-Play Functionality: Make the arm easy to set up and use without complex configurations.
5. Accessible Feedback: Design haptic feedback patterns that are easy to interpret and provide clear information about the environment.
6. Cost-Effectiveness: Select affordable components and manufacturing processes to keep the arm accessible.
7. Community Engagement: Foster a user community for feedback, support, and ongoing improvement.
8. Regulatory Compliance and Safety: Ensure the arm meets safety and accessibility standards.
9. Battery Life and Charging: Optimize power efficiency and charging methods for convenience and longevity.
10. Scalability: Design the system to allow future enhancements and scalability for broader applications.

`

1.6) System Description

The project comprises four main parts:

- Embedded System: Utilizes ultrasonic sensors, temperature detectors, and other sensors for obstacle detection and environmental data collection. The system is integrated into a wearable glove, providing real-time feedback to the user.
- AI Model: Provides robust object detection and face recognition capabilities, utilizing machine learning algorithms to interpret and describe the surroundings accurately.
- Mobile Application: Interacts with users, offering audio feedback and answering questions about surroundings. The app is developed using Flutter for cross-platform compatibility.
- Backend System: Links all components together, manages data flow, and integrates with a Firebase database for real-time updates and data storage. The backend ensures seamless communication between the glove, mobile app, and web interface.

`

Chapter 2
(Survey)

`

2.1) Literature Survey

The concept of smart assistive devices for the blind and visually impaired is a growing field with ongoing research and development. Here's a breakdown of relevant areas in the literature survey

2.1.1) Mobile assistive technologies for the visually

impaired: Mobile assistive technologies allow people with disabilities to benefit from portable, lightweight, discrete aids that are delivered by devices that are famous between the general population and therefore do not carry the same stigma as other, more traditional, assistive aids.

2.1.2) Smart Assistive Arm Applications Beyond

Navigation: Look for research on smart assistive arms designed for purposes beyond navigation, such as object recognition and manipulation. Explore the potential of these arms for tasks such as extracting objects, identifying items through touch or voice commands, and assisting with daily activities. Analyze the challenges and opportunities associated with integrating object recognition and manipulation capabilities into assistive arms.

2.1.3) Wearable Assistive Technologies

This part would delve into wearable assistive technologies for the blind, such as haptic feedback vests or smart glasses with obstacle detection features. Analyze how these technologies provide feedback to users and their impact on spatial awareness and obstacle avoidance. Consider studies on user comfort, usability, and the integration of these wearables with other assistive technologies.

`

2.1.4) Robotic Assistive Arms for Navigation

Research robotic arms designed for navigation assistance, focusing on obstacle detection and path planning algorithms. Explore different sensor technologies used, such as LiDAR, ultrasonic sensors, and cameras with object detection models. Evaluate their effectiveness in obstacle detection accuracy, range, and response time. Consider studies on user experience and acceptance of robotic arm navigation aids, including factors like weight, maneuverability, and ease of use.

2.2) Related works

The field of assistive technology for people who are blind or have low vision has been making tremendous strides in recent years. Thanks to advances in artificial intelligence and computer vision, there are now a wide range of tools and devices available to help people who are blind or have low vision.

For example, smart glasses that use computer vision to identify objects and text can provide real-time audio feedback to the wearer, articulating everyday visual information into speech. With each passing day, it seems that new and exciting innovations are emerging that promise to make life easier and more accessible for those who are blind or have low vision.

Here’s our pick of 5 highly useful assistive technology devices that can enhance the accessibility and independence of people who are blind. These devices provide better access to the world and enable users to overcome obstacles they may encounter in their daily lives.

`

2.2.1) AI Smart Glasses

AI smart glasses are a type of wearable assistive technology designed to help people who are blind or have low vision. These glasses typically use a combination of artificial intelligence, computer vision, and natural language processing to help users scan text, find objects, provide scene description and much more.

Figure 1 (Ai Smart Glasses)

Envision Glasses are one of the most innovative AI powered smart glasses for those who are blind or low vision.

Among its many features, the most widely used are its text to speech capabilities which allow users to read mail, signboards, recipes and basically any text they might come across. Features like 'Find Objects' and 'Describe Scene' also provide users with a better sense of their surroundings.

A recent update saw an addition of the “Ask Envision” feature which gave users access to the GPT language model. With GPT integration, users can scan a document and ask the glasses questions about the content. For example, they can ask for a summary of the text or ask specific questions about the information in the document.

With its powerful combination of AI, computer vision, and natural language processing, Envision Glasses represent a new standard in accessibility and independence for people who are blind or visually impaired. Whether you're navigating a crowded street or simply trying to read a menu at a restaurant, Envision Glasses can provide you with an unparalleled level of independence and accessibility. Envision Glasses retail at $1899 for the read edition, $2499 for the home edition and $3499 for the professional edition.

`

2.2.2) Bone Conduction Headphones

Bone conduction headphones are a type of headphone that transmits sound through the bones in the skull, rather than through the air like traditional headphones. This allows the wearer to hear sound while still being able to hear ambient sounds, such as traffic or conversations, which can be useful for outdoor activities or instances where situational awareness is important.

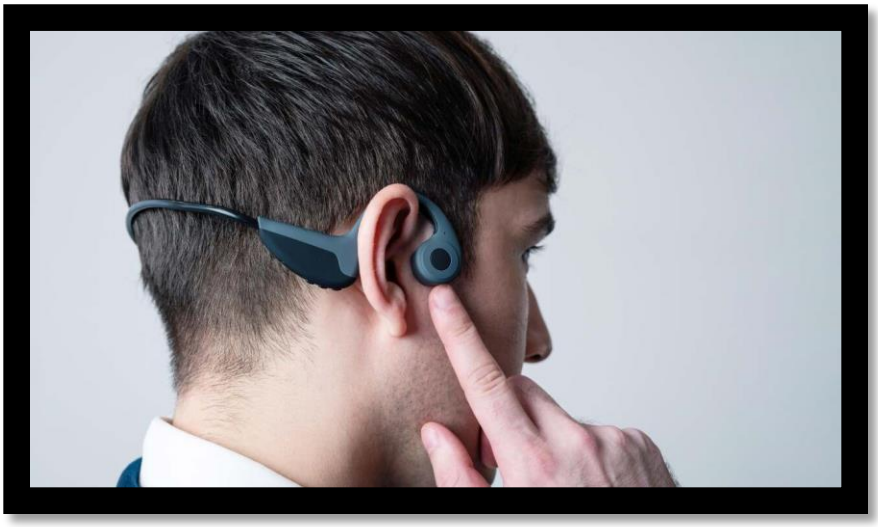


Figure 2(Bone Conduction Headphones)

Bone conduction headphones typically consist of a band that goes around the head and rests on the cheekbones, with small transducers that sit in front of the ears and vibrate to transmit sound through the bones.

They can connect to devices via Bluetooth or wired connections, and some models may also have built-in microphones for making phone calls or using voice assistants.

In the most recent edition of Inclusive Innovations, Mike May, Chief Evangelist for GoodMaps and Founder at Sendero Group, spoke about the many benefits of bone conduction headphones.

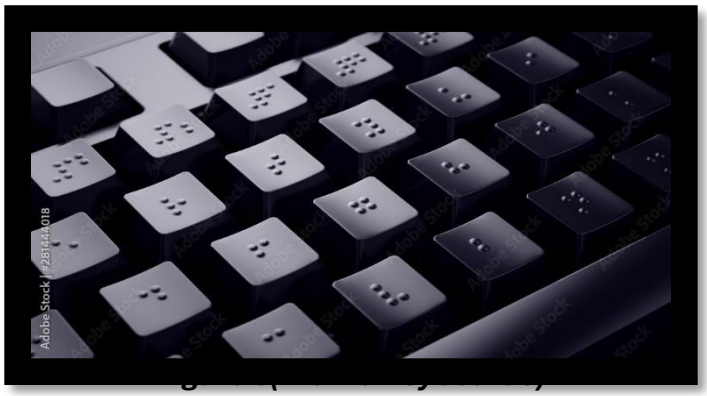
He stated that using traditional headphones in public spaces can be quite risky because they cover your ears, limiting important information from your surroundings, such as incoming vehicles.

Click here if you would like to listen to the entire speech given by Mike May at Inclusive Innovations 2023.

The cost of a basic bone conduction headphone can range from approximately $50 to $100. More advanced models with additional features, such as waterproofing, noise-cancellation, or longer battery life, can cost up to $300 or more.

`

2.2.3) Braille Keyboards



A Braille keyboard is a type of input device that allows users to type in Braille characters on a computer or other digital device.

Braille keyboards typically feature six keys, each of which corresponds to one of the six dots in a Braille cell.

When a user presses one or more of these keys, the corresponding dots are raised or lowered on a connected Braille display, which the user can read through touch.

Braille keyboards work by using electronic sensors to detect which keys are being pressed. When a key is pressed, it sends a signal to a microprocessor that processes the input and sends the appropriate signals to the connected Braille display.

The display then raises or lowers the corresponding dots, allowing the user to read the output.

Braille keyboards can be connected to a variety of digital devices, including computers, smartphones, and tablets, and they are an important tool for people who are blind or visually impaired.

By providing a way to input Braille characters, Braille keyboards make it possible for users to communicate and access information through touch, rather than relying solely on audio feedback.

Generally, basic Braille keyboards can range from around $100 to $500, while more advanced models with additional features like Bluetooth connectivity and built-in speech synthesis can cost upwards of $1000.

`

2.2.4) Braille Displays

A Braille display is an assistive technology device that allows people who are blind or visually impaired to read digital content using Braille characters.

Braille displays typically consist of a row of cells with small pins that can be raised or lowered to form Braille characters. These cells are controlled electronically and can be used to display text from a computer, tablet, or smartphone.

Figure 4 (Braille Displays)

When connected to a device, a Braille display can output text in real-time, allowing the user to read and navigate digital content using Braille. This can be particularly useful for activities such as reading emails, browsing the web, or using applications that have Braille support.

Braille displays can vary in size and the number of cells they have, with some models having as few as 20 cells, while others may have up to 80 or more cells. The larger displays are typically more expensive but can provide more detailed and comprehensive access to digital content.

A basic Braille display with 14-20 cells can range from approximately $1,500 to $3,000. Larger displays with 40-80 cells or more can cost anywhere between $5,000 to $10,000 or more

`

2.2.5) Audio Labelers

Figure 5(Audio Labelers)

Audio Labelers are devices that enable people who are blind or visually impaired to label items in their environment with spoken audio labels. These labels can then be played back using a special reader or scanner, allowing the user to identify and locate the item more easily.

Audio Labelers typically consist of a handheld device with a microphone and speaker, as well as a set of special labels that can be affixed to items in the user's environment.

The user records a spoken label for each item using the device's microphone, and then affixes the corresponding label to the item. When the user later wants to identify the item, they can scan the label using a reader or scanner, which will play back the corresponding spoken label.

Audio Labelers can be a useful tool for people who are blind or visually impaired, as they can help to increase independence and mobility by providing a way to easily identify and locate items in the user's environment. They can also be used to label items in public places, such as in a classroom or workplace, to help the user navigate unfamiliar environments more easily.

The cost of a basic Audio Labeler can range from approximately $20 to $50. However, more advanced models with additional features, such as the ability to store multiple labels or connect to a computer, can cost up to $200 or more.

`

2.3) Conclusion

Advancements in technology are revolutionizing the ways in which people who are blind can connect with and navigate the world around them. These innovative devices each offer unique features that empower users to live more independently. Whether it's using voice or braille displays to send and receive information or gaining a greater understanding of their environment through object recognition and people identification, these technologies are transforming the way people with blindness and low vision interact with the world.

`

Chapter 3
(Software life cycle and Analysis)

`

3.1) Project Development Methodology (phases)

The main characteristics of the Waterfall model are a sequential progression through the different stages of a project, from initiation to the delivery phase. The Waterfall model has its limitations, leading to the creation of many spin-off models over the years. However, the Waterfall model remains a foundational approach in project development. It consists of the following phases:

- Introduction
- Analysis
- Design
- Coding
- Testing
- Implementation and Maintenance

In the Systems Development Life Cycle (SDLC), the Waterfall model progresses through six phases, resembling a waterfall with one phase flowing into the next.

Planning Phase

The purpose of the planning phase is to conduct an initial high-level investigation of the business need and recommend a solution. Once approved by the management team, stakeholders, client, or project sponsor, it proceeds to the next phase.

Analysis Phase

The purpose of the analysis phase is to conduct a detailed examination of the current business needs and identify options to achieve those needs. During this phase, the Business Analyst creates the Business Requirements Document (BRD).

Design Phase

The purpose of the design phase is to identify and document a solution, including technical and procedural specifications. A design document is created, covering technical, environmental, data, program, procedural, and testing specifications.

`

Coding Phase

In the coding phase, the design document is translated into a functional program or system. This phase involves the actual construction or development of the software.

Testing Phase

The purpose of the testing phase is to test the system and related procedures to ensure they meet the requirements specified by stakeholders and documented in the BRD, design plan, and testing plan.

Implementation and Maintenance Phase

The purpose of the implementation phase is to release a fully tested and operational product to the end user or customer. The product must meet all documented requirements and pass the testing phase before being released into the production environment.

`

The SDLC is a process of creating or altering information systems, consisting of steps or phases where each phase uses the results of the previous one.

The systems development life cycle (SDLC), or software development process in systems engineering, information systems and software engineering, is a process of creating or altering information systems, and the models and methodologies that people use to develop these systems. It consists of a set of steps or phases in which each phase of the SDLC uses the results of the previous one.

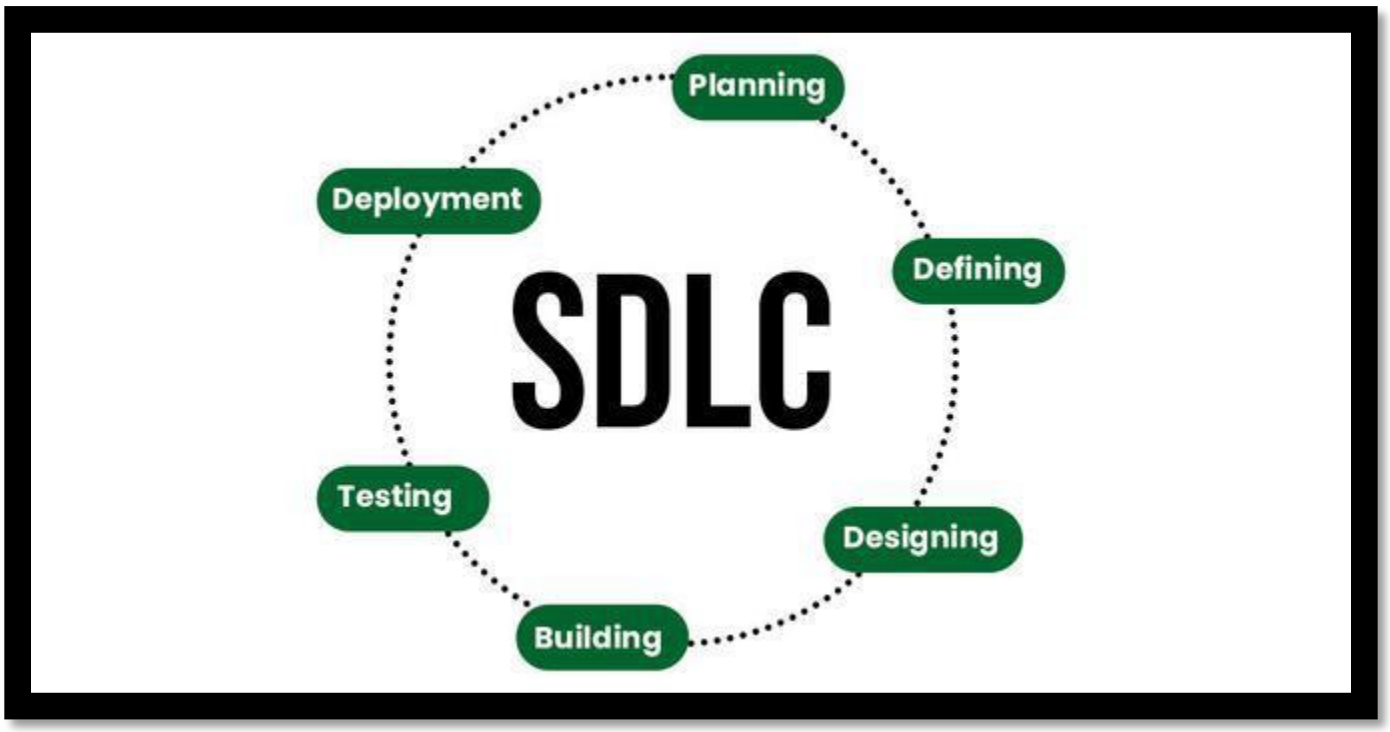


Figure 6(SDLC cycle)

`

3.2) Planning

Our goal with Comfy View is to enhance mobility, independence, and safety for blind and visually impaired individuals through a wearable assistive arm. The project aims to include several features:

- Develop an arm that detects and avoids obstacles in the user's path.
- Integrate AI-powered object detection to inform users about objects in their environment.
- Implement face recognition to assist with social interaction and identification.
- Create a user-friendly mobile application and website for data visualization and customization.
- Ensure the arm is comfortable, lightweight, and easy to wear for extended periods.

User Needs Assessment

- Conduct surveys and research about blind and visually impaired individuals to understand their specific needs and challenges.
- Analyze existing assistive technologies to identify areas for improvement.
- Evaluate ethical considerations regarding face recognition and data privacy.

Hardware Design

- Design a comfortable and ergonomic arm that can be easily attached and adjusted.
- Integrate sensors like ultrasonic sensors and cameras for obstacle detection.
- Include haptic feedback mechanisms to alert users about obstacles and objects.
- Consider additional sensors like temperature sensors for environmental awareness

Software Design

- Develop AI algorithms for object detection and face recognition using appropriate training datasets.
- Design a mobile application and website for real-time data visualization, audio feedback, and alerts.

`

3.3) Requirements Analysis

The first step in building a desired software product is extracting its requirements. While consumers may understand what the program is supposed to accomplish, recognizing incomplete, unclear, or conflicting requirements often requires expertise and experience in software engineering.

3.3.1) Requirements Analysis

The first step in building a desired software product is to extract its requirements. While consumers may feel they understand what the program is supposed to accomplish, recognizing incomplete, unclear, or conflicting requirements may need expertise and experience in software engineering.

3.3.2) System requirements

3.3.2.1) Hardware:

- Sensors: Ultrasonic sensors for obstacle detection and temperature sensors for measuring the surrounding temperature.
- Microcontrollers: Low-power microcontrollers (e.g., Arduino, ESP32 CAM) to process sensor data and control feedback mechanisms.
- Feedback Mechanisms: Buzzers and alarms to provide non-visual feedback to the user.
- Wireless Modules: Wireless modules (e.g., Wi-Fi, GPS) to connect with external devices.
- Physical Design: Design the gloves to be lightweight, comfortable, and durable for everyday use.

3.3.2.2) Software

- PC Requirements: Windows 7 or higher for optimal performance in Android Studio and Visual Studio.
- Database: Firebase for constructing the database.
- Algorithm Design: React, NodeJS for backend, Flutter-API, Bloc state management, and shared preferences for mobile applications.

`

3.4) Software requirements specification

3.4.1) Functional requirements

System User

- Register
- Log in with password
- Log in with face print
- Edit personal data
- Change password
- Check results and captured data
- Check guiding/leading
- Search for an object
- Buy gloves
- Contact customer service
- Log out

System Client

- Register
- Log in
- Buy gloves
- Contact customer service
- Log out

System Admin

1- Hardware Functions

- Capture picture
- Measure temperature
- Locate objects

2- Software Functions

`

- Login
- Create user account
- Check images
- Save images in local storage
- Display detection results
- Convert URL to blob

Artificial Intelligence Functions

- Object detection functionality
- Camera integration
- Face recognition
- Login process
- User interface

3.4.2) Nonfunctional requirements

1. Usability: Easy to put on, comfortable to wear for extended periods, intuitive to use with minimal training.
2. Portability: Lightweight and compact, easily carried in a bag or pocket.
3. Safety: Non-toxic, non-allergenic materials ensuring no harm or discomfort to the user.
4. Reliability: Consistent and accurate feedback and information.
5. Affordability: Cost-effective to ensure accessibility to a wide range of users.
6. Compatibility: Compatible with other assistive devices or technologies commonly used by blind individuals, such as smartphones or navigation aids.
7. Privacy: Secure data collection and transmission, used only for its intended purpose.
8. Availability: System availability 24/7 for daily use.

`

3.5) System Users

In the system, we have two main users:

Blind Users: These users wear the assistive arm device to regain mobility and functionality for daily activities, including grasping, lifting, and manipulating objects.

Buyers of the Wearable Assistive Arm: Individuals who purchase the device for a blind person to enhance their mobility and independence.

- Reasons for Purchasing a Wearable Assistive Arm
- Increased Mobility: Real-time feedback about obstacles and hazards, allowing confident and independent navigation.
- Promotion of Independence: Enhances autonomy and efficiency in daily tasks, fostering a greater sense of self-sufficiency.

3.6) Tools

Web Design (Front-end) using Visual Studio Code

- **Frontend**: React
- **Backend**: Node.js
- **Database**: Firebase

Mobile Application using Visual Studio & Android Studio

- Dart
- Bloc state management
- Dio
- HTTP request
- Shared preference
- Clean architecture

`

Databases using Firebase

- Save user data into Firebase Fire store
- Upload photos captured by the glove into Firebase Storage

Embedded Part using Arduino IDE

- ESP/ISP
- HTTP request
- Socket programming
- Camera integration
- Wi-Fi module

Artificial Intelligence using Anaconda Environment

- OpenCV (Open Source Computer Vision Library)
- NumPy (Numerical Python)
- Scikit-image
- Matplotlib
- Pillow (Python Imaging Library)
- Dlib library
- CUDA Toolkit

`

3.7) System Design

The purpose of the Design phase is to plan a solution for a problem specified by the requirements. System design aims to identify the modules that should exist in the system, the specification of those modules and how they interact with each other to produce the results. The goal of the design process is to produce a model that can be used later to build that system. The produced model is called design of the system.

System design is the process of defining the architecture, components, modules, interfaces and data for a system to satisfy specified requirements.

Normally, the design proceeds in two stages:

• Physical design is a graphical representation of a system showing the system's internal and external entities and the flow of data into and out of these entities. An internal entity is an entity within the system that transforms data.

• Database design: It's the implementation of the schema into a database. To represent the physical design of the system, we use diagrams like use case diagrams, etc.

`

3.7.1) Entity Relationship Diagram (ERD)

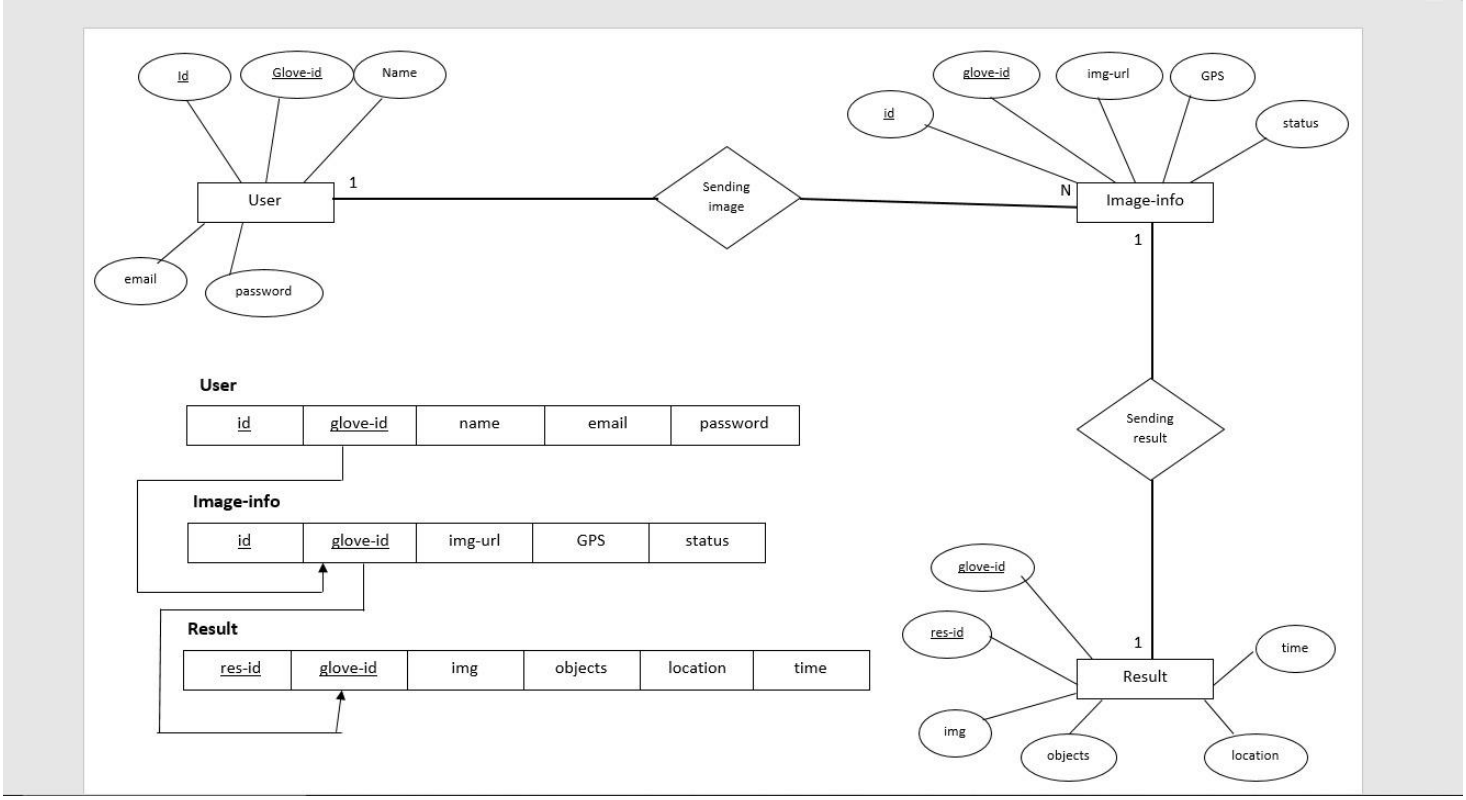


Figure 7 (ERD)

`

3.7.2) Use Case Diagram

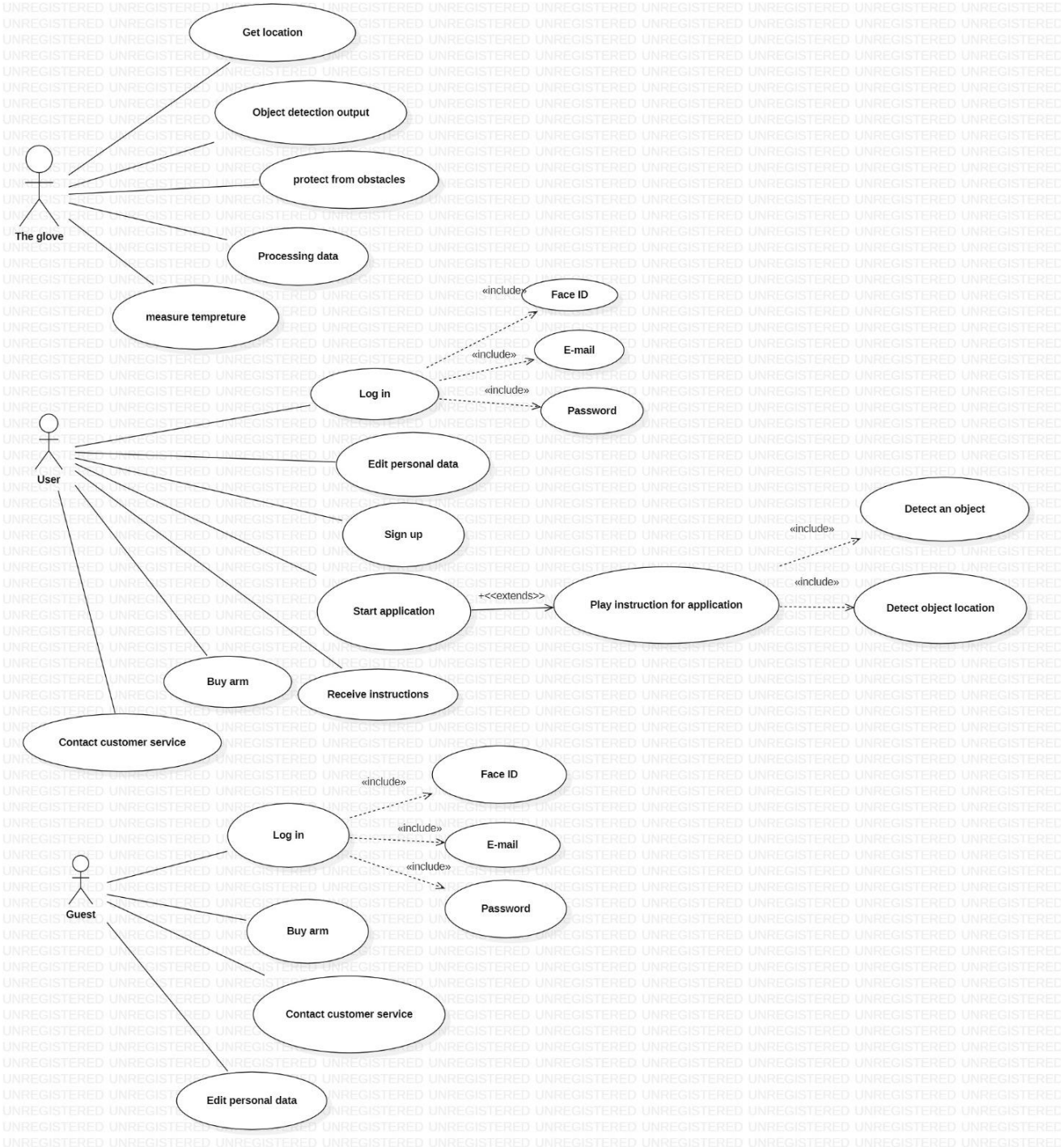


Figure 8(Use Case Diagram)

`

3.7.3) Class Diagram

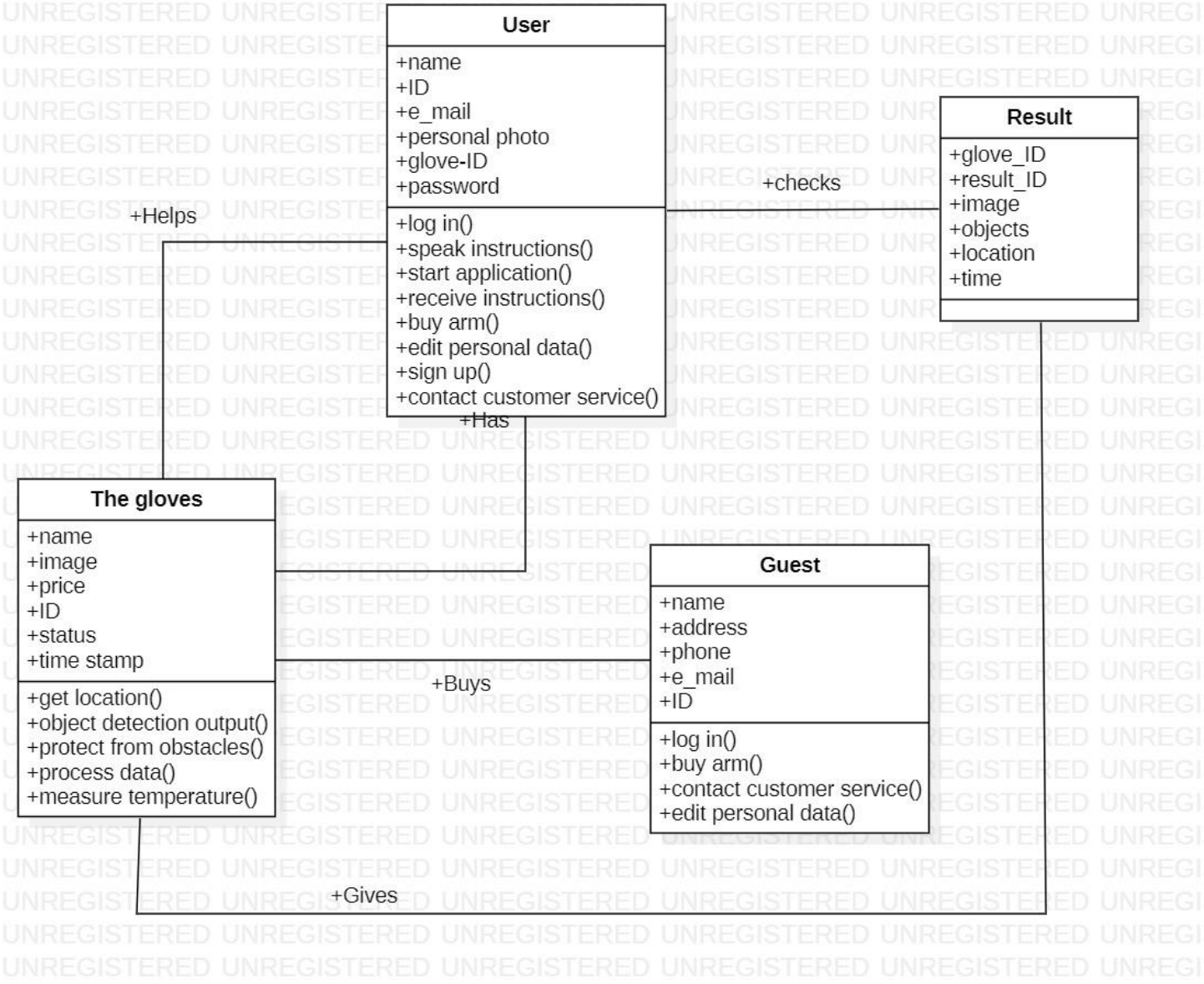


Figure 9(Class Diagram)

`

3.7.4) Sequence Diagram

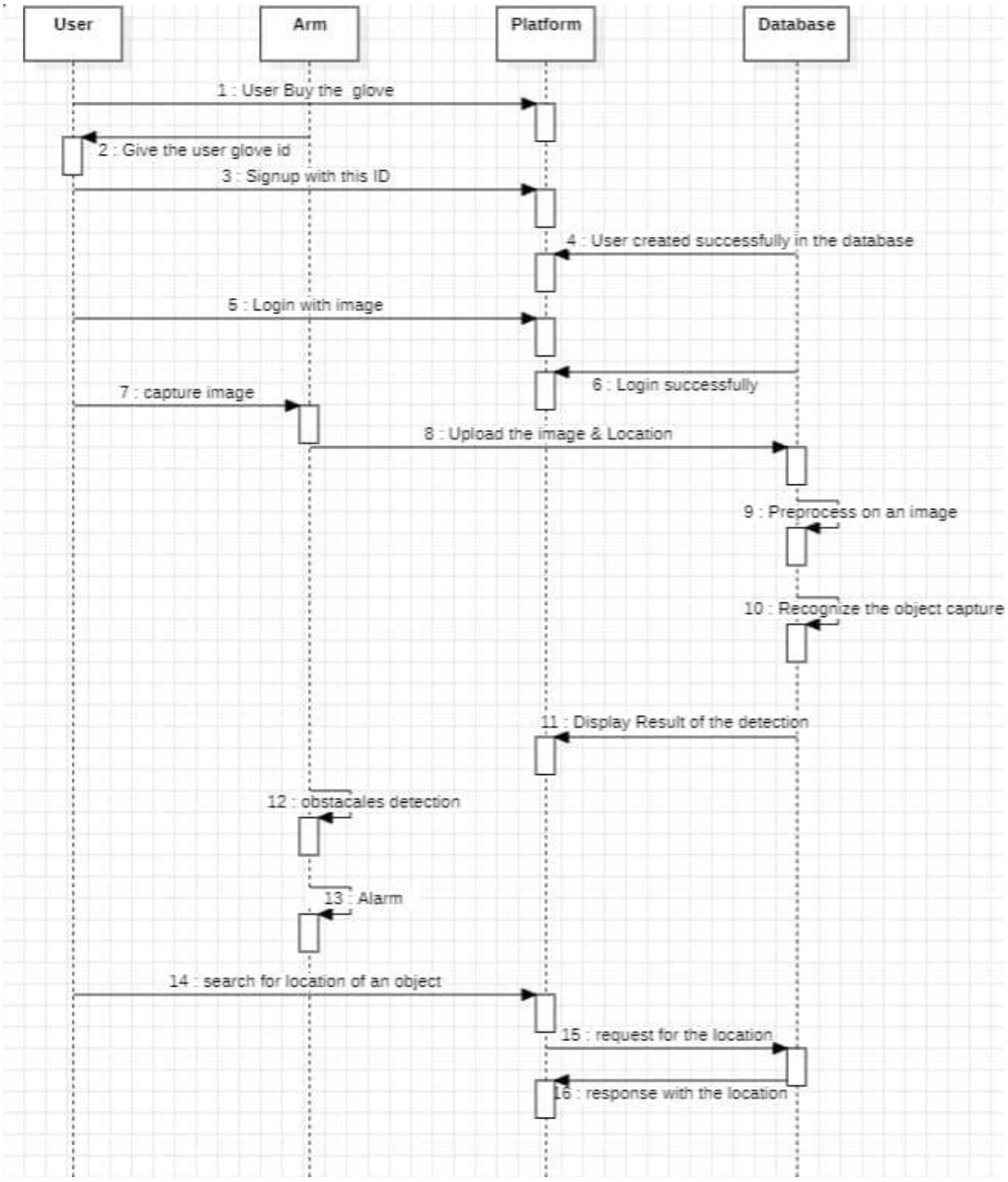


Figure 10(Sequence Diagram)

`

3.7.5) Activity Diagram

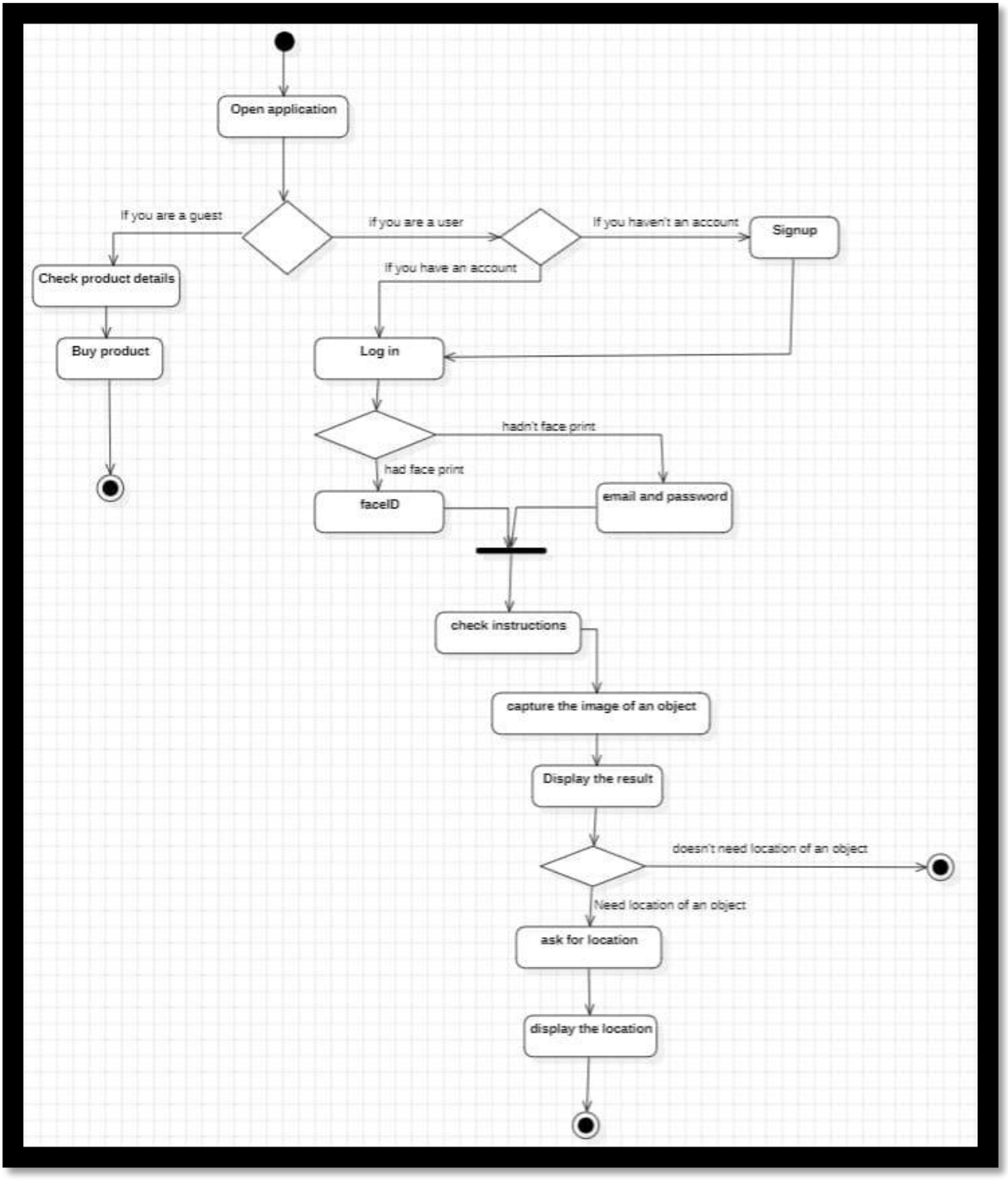


Figure 11 (Activity Diagram)

`

3.8) System Implementation

After obtaining the system design documents, work is divided into modules/units, and actual coding begins. This phase involves developing the code, which will be detailed in the following chapters.

3.9) System testing

Following code development, the product is tested against the requirements to ensure it meets the identified needs. Testing phases include unit testing, integration testing, system testing, and acceptance testing.

3.10) Documentation

Documenting the internal design of software for future maintenance and development is essential. Documentation of the external interface is equally important, ensuring clarity and usability for end users.

This chapter outlines the comprehensive development process, emphasizing a structured approach to creating a robust and user-centric wearable assistive arm for visually impaired individuals.

`

Chapter 4
(Tools)

`

4.1) Overview

Comfy View Glove is designed as a comprehensive solution to address the challenges faced by visually impaired individuals in navigating their surroundings. It combines advanced technology with user-centric design to enhance independence and safety for the blind community. The product perspective encompasses various aspects including its overview, problem definition, solution approach, scope, objectives, and system description.

4.2) Programming Languages we have used

- - flutter
- - Dart
- -python
- - c
- -react
- - NodeJS

`

4.3) Basic work

Visual Studio Code

Is a dual-licensed source-code editor made by Microsoft for Windows, Linux and macOS. In the Stack Overflow 2019 Developer Survey, Visual Studio Code was ranked the most popular developer environment tool, with 50.7% of 87,317 respondents reporting that they use it.

Visual Studio Code is a source-code editor that can be used with a variety of programming languages, including Java, JavaScript, Go, Node.js, Python and C++.It is based on the Electron framework, which is used to develop Node.js Web applications that run on the Blink layout engine. Visual Studio Code employs the same editor component (codenamed "Monaco") used in Azure DevOps (formerly called Visual Studio Online and Visual Studio Team Services)

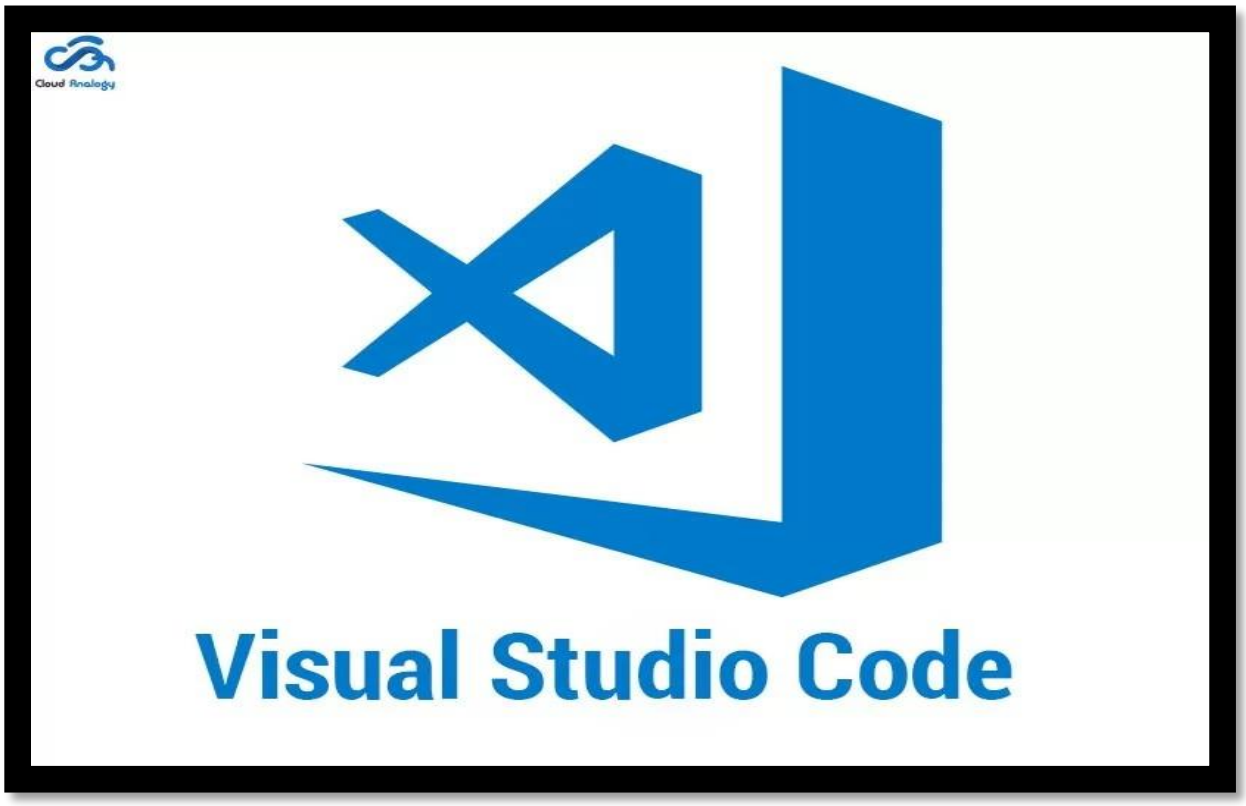


Figure 12 (VS code)

`

Android Studio

Is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating

Figure 13(Android Studio)

systems or as a subscription-based service in 2020.It is a replacement for the Eclipse Android Development Tools (E-ADT) as the primary IDE for native Android application development.

`

4.4) Mobile Content

Flutter using dart

Is an open-source UI software development kit created by Google. It is used to develop cross platform applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, and the web from a single codebase.

Flutter apps are written in the Dart language and make use of many of the language's more advanced features.

On Windows, macOS, and Linux Flutter runs in the Dart virtual machine, which features a just-in-time execution engine. While writing and debugging an app, Flutter uses Just In Time compilation, allowing for "hot reload", with which modifications to source files can be injected into a running application. Flutter extends this with support for stateful hot reload, where in most cases changes to source code are reflected immediately in the running app without requiring a restart or any loss of state.

Figure 14(Flutter & Dart)

`

4.5) Web content

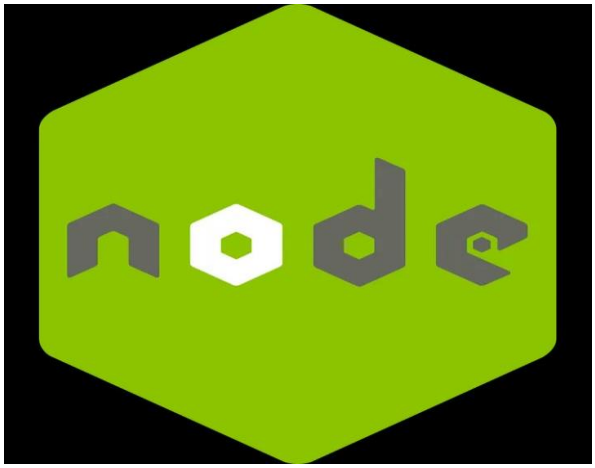
Node.js

is a runtime environment that allows developers to run JavaScript code on the server-side. It's built on Chrome's V8 JavaScript engine, providing an event-driven, non-blocking I/O model that makes it lightweight and efficient for building scalable network applications.

With Node.js, developers can create web servers, APIs, real-time applications, and more. Its package manager, npm, hosts a vast ecosystem of open-source libraries, frameworks, and tools, enabling rapid development and deployment. Node.js excels in handling asynchronous operations, making it suitable for handling concurrent requests and I/O-bound tasks without blocking the event loop.

Its single-threaded event loop architecture facilitates high concurrency, making it ideal for building microservices and real-time applications such as chat applications, streaming platforms, and IoT applications. Additionally, its ability to use JavaScript on both the client and server sides allows for seamless code sharing and faster development cycles. It's widely adopted by companies like Netflix, LinkedIn, and Uber for its performance, scalability, and flexibility in building modern web applications

Figure 15(Node.js)



`

React

React is a JavaScript library for building user interfaces, developed by Facebook. It enables developers to create interactive and dynamic UI components for web and mobile applications. React follows a component-based architecture, where UIs are composed of reusable components, making it easier to manage and maintain large-scale applications.

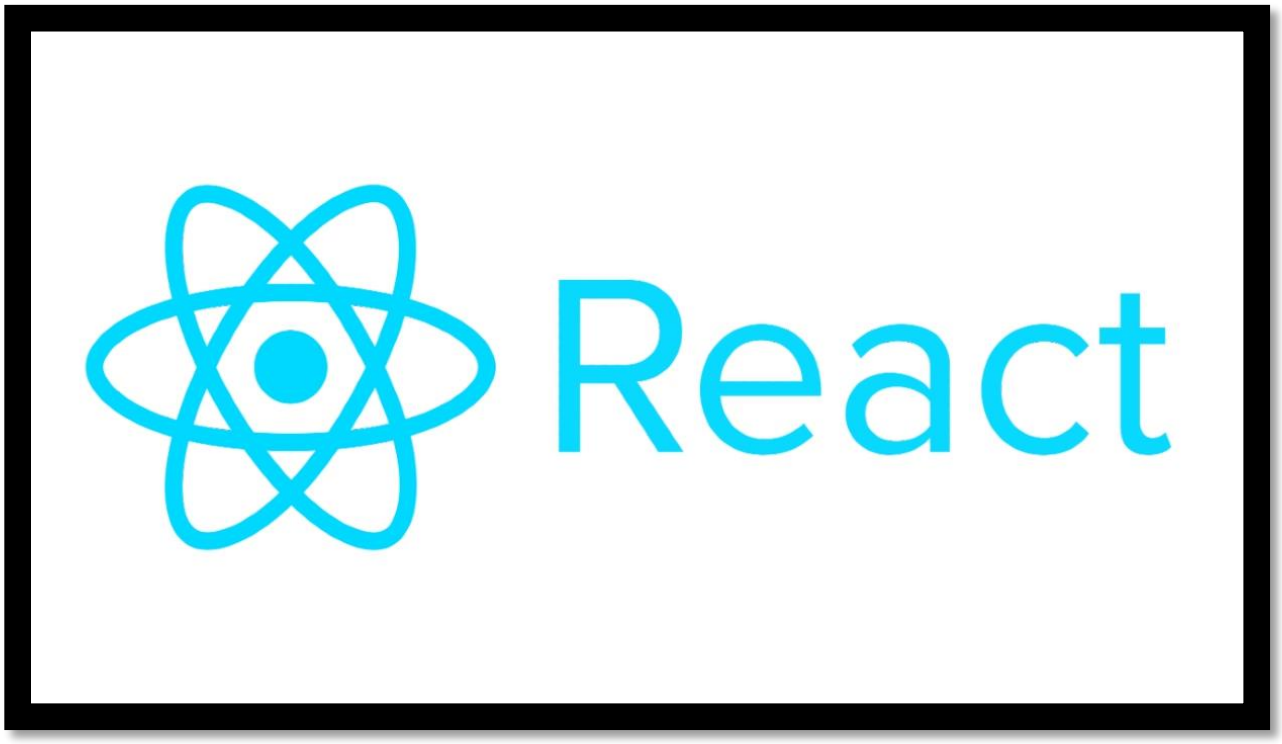


Figure 16(React)

`

4.6) Embedded Content

Embedded C

is a programming language specifically tailored for developing software for embedded systems, which are computing devices with limited resources like microcontrollers or microprocessors. It's a subset of the C programming language, optimized for efficiency, portability, and direct hardware manipulation.

In embedded C, developers have direct access to hardware resources such as memory-mapped registers, interrupts, and peripherals, allowing for precise control over the device's behavior. This low-level access is crucial for tasks like interfacing with sensors, controlling motors, or managing communication protocols.

The language retains many features of standard C, including variables, control structures, functions, and data types, making it familiar to C programmers. However, it may lack certain features like dynamic memory allocation due to the limited resources of embedded systems.

Figure 17(C)

`

The ESP/ISF (Embedded Systems Programming and Industrial Systems Fundamentals)

framework is a comprehensive approach to embedded systems development and industrial automation. It combines principles of embedded systems programming with knowledge of industrial processes, aiming to create efficient, reliable, and scalable solutions for various industries.

Embedded systems are specialized computing systems designed to perform specific tasks within larger systems. They are found in a wide range of applications, from consumer electronics and automotive systems to industrial machinery and IoT devices. The ESP/ISF framework provides a structured approach to designing, developing, and deploying embedded systems in industrial settings.

At its core, ESP/ISF emphasizes the use of microcontrollers and microprocessors to control hardware peripherals and interact with the physical world. This involves writing firmware in languages like C or C++, optimizing code for performance, memory usage, and power efficiency. Understanding hardware constraints and utilizing low-level programming techniques are essential skills in ESP/ISF.

Industrial Systems Fundamentals (ISF) encompasses knowledge of industrial processes, sensors, actuators, and control systems. It involves understanding the requirements of specific industries, such as manufacturing, energy, or transportation, and designing embedded systems to meet those needs. ISF also covers topics like real-time operating systems (RTOS), communication protocols (such as Modbus or CAN bus), and safety standards (like IEC 61508 or ISO 26262).

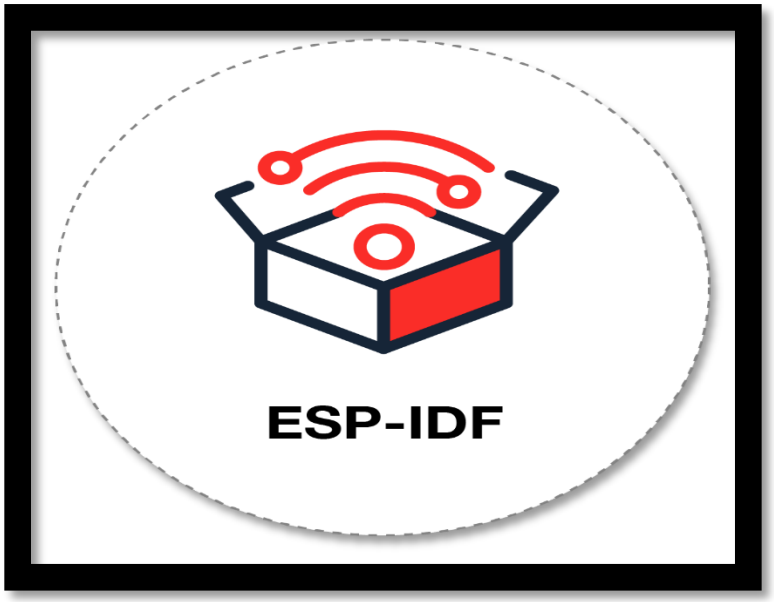


Figure 18(ESP-IDF)

`

The Arduino IDE (Integrated Development Environment)

is a software tool used for programming Arduino microcontroller boards. It provides a simple yet powerful platform for beginners and experts alike to write, compile, and upload code to Arduino-compatible hardware.

At its core, the Arduino IDE is built around the Wiring programming language, which is based on the C and C++ languages. This language simplifies the process of writing code for microcontroller-based projects, making it accessible to those with little or no programming experience.

One of the key features of the Arduino IDE is its user-friendly interface. It includes a text editor with syntax highlighting and auto-completion, making it easier for users to write and edit code. The IDE also provides a variety of built-in functions and libraries that simplify common tasks, such as reading from sensors or controlling motors.

Another important aspect of the Arduino IDE is its compatibility with a wide range of Arduino boards. Whether you're using the classic Arduino Uno, the powerful Arduino Mega, or one of the newer boards like the Arduino Nano or Arduino MKR series, the IDE provides support for compiling and uploading code to these devices seamlessly.

The Arduino IDE simplifies the process of uploading code to Arduino boards through its built-in bootloader and serial communication capabilities. Users can connect their Arduino board to their computer via USB, select the appropriate board and serial port in the IDE, and upload their code with the click of a button.

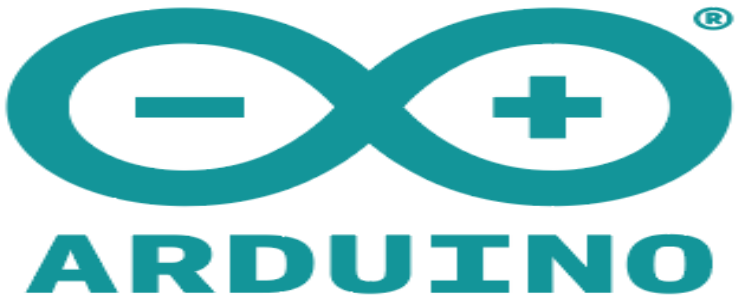


Figure 19(Arduino)

`

4.7) Database (Firebase)

Figure 20(Fire Base)

This article is about Google's cloud computing and development platform. For Google's other cloud computing offerings, see Google Cloud. For artillery firebases, see Fire support base. For other uses, see Firebase (disambiguation).

Firebase is a platform developed by Google for creating mobile and web applications. It was originally an independent company founded in 2011. In 2014, Google acquired the platform and it is now their flagship offering for app development.

Firebase provides the tools and infrastructure you need to develop, grow, and earn money from your app.
This package supports web (browser), mobileweb, and server (Node.js) clients.

The Firebase Realtime Database lets you store and query user data, and makes it available between users in real-time.

Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud Platform.

Firebase Storage lets you upload and store user generated content, such as files, and images.

Firebase Cloud Messaging is a cross-platform messaging solution that lets you reliably deliver messages at no cost

. Firebase helps you authenticate and manage users who access your application.

`

4.8) AI content

Python

Figure 21 (Python)

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation.

Its language constructs as well as its objectoriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming.

Python is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 85 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a garbage collection system using reference counting.

`

Anaconda (Python distribution)

Figure 22 (Anaconda)

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS.

It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012.

As an Anaconda, Inc. product, it is also known as Anaconda Distribution or Anaconda Individual Edition, while other products from the company are Anaconda Team Edition and Anaconda Enterprise Edition, both of which are not free. Package versions in Anaconda are managed by the package management system conda.

This package manager was spun out as a separate open-source package as it ended up being useful on its own and for other things than Python. There is also a small, bootstrap version of Anaconda called Miniconda, which includes only conda, Python, the packages they depend on, and a small number of other packages.

`

Python libraries:
1- OpenCV (Open Source Computer Vision Library): OpenCV is a powerful library for computer vision tasks. It provides tools for image and video processing, object detection, facial recognition, and more, with support for various programming languages.

2- NumPy (Numerical Python): NumPy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

3- scikit-image: scikit-image is a Python library for image processing built on top of NumPy and SciPy. It offers a comprehensive collection of algorithms for tasks such as image filtering, segmentation, feature extraction, and more, making it ideal for scientific and industrial applications.

4- Matplotlib: Matplotlib is a plotting library for Python. It allows users to create a wide variety of plots and visualizations, including line plots, scatter plots, histograms, and more. Matplotlib is highly customizable and widely used in data analysis and scientific research.

5- Pillow (Python Imaging Library): Pillow is a fork of the Python Imaging Library (PIL) that adds support for Python 3.x and maintains compatibility with the original library. It provides tools for image processing tasks such as opening, manipulating, and saving various image file formats.

6- Dlib library: Dlib is a modern C++ toolkit containing machine learning algorithms and tools for building complex software in C++ to solve real-world problems. It is particularly known for its implementations of facial recognition, facial landmark detection, object detection, and more.

7- CUDA Toolkit: The CUDA Toolkit is a development environment for building GPU-accelerated applications. It provides libraries, tools, and APIs for developers to leverage the parallel computing power of NVIDIA GPUs. It's widely used for scientific computing, deep learning, and other high-performance computing task

`

Chapter 5 (System Implementation)

`

5.1) Embedded Implementation

Hardware implementation:

For the microcontroller we used:

Arduino uno for the following reasons:

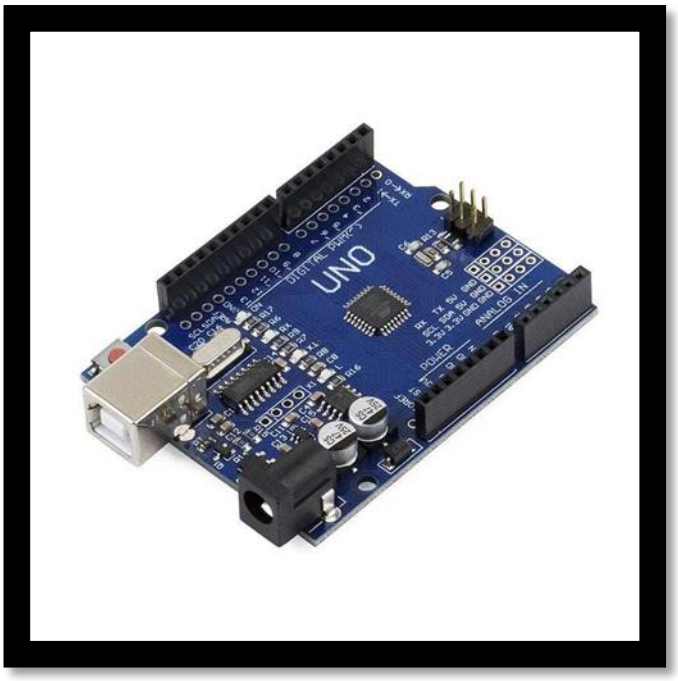


Figure 23 (Arduino Uno)

1- Ease of Use: The Arduino Uno is designed with a user-friendly interface.

2- Open-Source Platform: Which means that the hardware design and software libraries are freely available for anyone to use and modify.

3-Versatility: The Arduino Uno can be used for a wide range of projects, from simple LED blinking and sensor reading to more complex tasks like robotics, home automation, and data logging. Its versatility makes it a great tool for learning and prototyping.

3- Affordability: The Arduino Uno is relatively inexpensive compared to other microcontroller boards with similar capabilities.

4- Wide Range of Libraries and Support: The Arduino platform has a vast collection of libraries and examples that make it easy to interface with sensors, displays, motors, and other components.

5- Availability: The Arduino Uno is widely available from various sources, including official distributors, making it easy to purchase and replace if needed.

`

For the sensors we used:

1- Ultrasonic sensor to measure the distance and velocity. This sensor operates on sound wave property to measure the velocity and distance of the object.

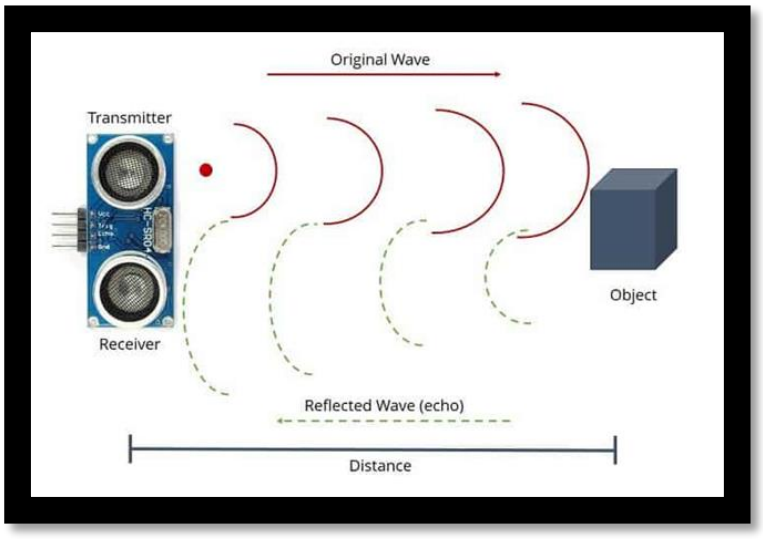


Figure 24 (Ultrasonic Sensor)

2- Temperature sensor lm35, it's a measuring device having an analog output voltage proportional to the temperature. It provides output voltage in centigrade (Celsius).

Figure 25 (LM 35)

`

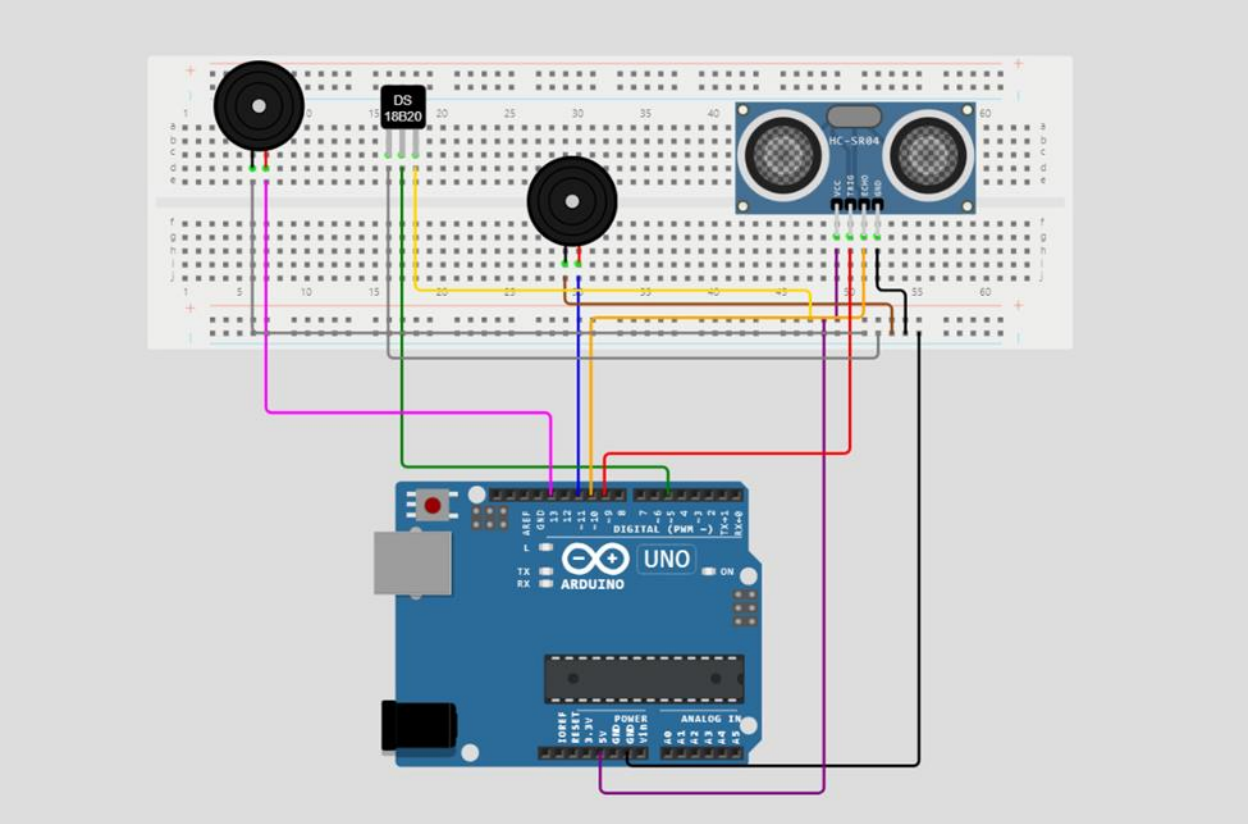


Figure 26 (Simulation 1)

The connection:

We will connect the ultrasonic sensor trig pin to pin 9 on the Arduino uno board, echo pin on pin 10 on the board and the buzzer connected to it is on pin 11, once the object is closer than 15 cm the buzzer will turn on alarming the user of the obstacle.

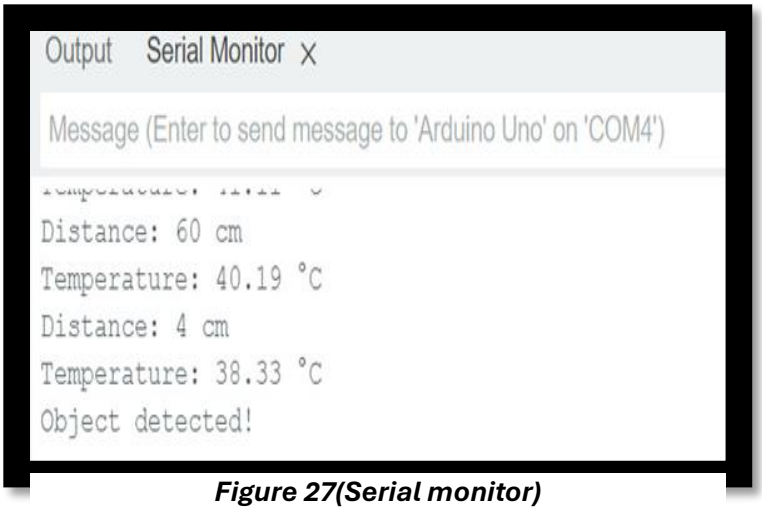


Figure 27(Serial monitor)

For the temperature sensor it will be connected to pin 5 and the buzzer connected to it will be on pin 13, once the temperature is above 50 degrees the buzzer will turn on alarming the user.

`

sketch_may7a.ino

```
#define TRIG_PIN 9
#define ECHO_PIN 10
#define LM35_PIN 5
#define LED_PIN 13

const int threshold_distance = 15;   // Threshold distance for activating buzzer (in cm)
const int threshold_temperature = 50; // Threshold temperature for turning on LED (in degrees Celsius)

void setup() {
  Serial.begin(9600);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  // Ultrasonic sensor reading
  long duration, distance;
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
  duration = pulseIn(ECHO_PIN, HIGH);
  distance = (duration / 2) / 29.1;

  // LM35 temperature sensor reading
  int sensorValue = analogRead(LM35_PIN); // Read the analog pin connected to LM35
```

Figure 28 (Arduino code snippet 1)

sketch_may7a.ino

```
  float voltage = (sensorValue / 1024.0) * 5.0; // Convert analog reading to voltage (assuming 5V reference)
  float temperatureC = (voltage * 100.0) / 10.0;

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  Serial.print("Temperature: ");
  Serial.print(temperatureC);
  Serial.println(" °C");

  // Check distance and activate buzzer if an object is within the threshold distance
  if (distance < threshold_distance) {
    digitalWrite(BUZZER_PIN, HIGH);
    Serial.println("Object detected!");
  } else {
    digitalWrite(BUZZER_PIN, LOW);
  }

  // Check temperature and turn on LED if it exceeds the threshold
  if (temperatureC > threshold_temperature) {
    digitalWrite(LED_PIN, HIGH);
    Serial.println("Temperature above threshold, LED ON");
  } else {
    digitalWrite(LED_PIN, LOW);
  }

  delay(1000); // Delay between sensor readings
}
```

Figure 29 (Arduino code snippet 2)

`

3- For capturing pictures will be using the esp32-cam

Figure 30 (ESP32 CAM)

void initWiFi()

Explanation: This function initializes and connects to the WiFi network using the provided SSID and password. It continuously checks the connection status and waits until the ESP32 is successfully connected.

```
void initWiFi() {
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
}
```

Figure 31(Code snippet3 initwifi)

void initLittleFS()

Explanation: This function initializes the LittleFS file system. If mounting the file system fails, it prints an error message and restarts the ESP32. Otherwise, it confirms the successful mounting.

`

```
void initLittleFS() {
  if (!LittleFS.begin(true)) {
    Serial.println("An Error has occurred while mounting LittleFS");
    ESP.restart();
  } else {
    delay(500);
    Serial.println("LittleFS mounted successfully");
  }
}
```

Figure 32(Code snippet4 initLittleFs)

void initCamera()

Explanation: This function sets up and initializes the camera module with specified configurations such as pin assignments and image settings. It checks for PSRAM availability to set frame size and JPEG quality. If initialization fails, it restarts the ESP32

Figure 33(Code snippet5 initCamera)

`

void initTime()

Explanation: This function synchronizes the system time with NTP servers. It waits until a valid time is received and then prints the current time

```
void initTime() {
  configTime(0, 0, "pool.ntp.org", "time.nist.gov");

  Serial.print("Waiting for NTP time sync: ");
  time_t now = time(nullptr);
  while (now < 8 * 3600 * 2) {
    delay(500);
    Serial.print(".");
    now = time(nullptr);
  }
  Serial.println("");
  struct tm timeinfo;
  gmtime_r(&now, &timeinfo);
  Serial.printf("Current time: %s", asctime(&timeinfo));
}
```

Figure 34(Code snippet6 initTime)

void capturePhotoSaveLittleFS()

Explanation: This function saves the image URL and GPS coordinates to Firestore. It creates a JSON object with the necessary data, generates a random document ID, and uploads the data to a specific Firestore collection. It handles the creation of the document and prints relevant status messages.

`

```
void saveDataToFirestore(String imageUrl, String gpsCoordinates) {
  String fileNameOnly = filePath.substring(1 + filePath.lastIndexOf('/'), filePath.lastInd

  FirebaseJson content;
  time_t now = time(nullptr);
  struct tm* p_tm = gmtime(&now);
  char timeString[30];
  strftime(timeString, sizeof(timeString), "%Y-%m-%dT%H:%M:%SZ", p_tm);
  String currentTime = String(timeString);

  content.set("fields/url/stringValue", fileNameOnly.c_str());
  content.set("fields/gps/stringValue", gpsCoordinates);
  content.set("fields/glove_id/integerValue", 12100);
  content.set("fields/done/booleanValue", false);
  content.set("fields/time/timestampValue", currentTime);

  String collectionPath = "Reham";
  String documentId = String(random(1, 100000));
  String documentPath = collectionPath + "/" + documentId;

  if (Firebase.Firestore.createDocument(&fbdo, FIREBASE_PROJECT_ID, "", documentPath.c_str
    Serial.println("Document created in Firestore.");
    Serial.printf("ok\n%s\n\n", fbdo.payload().c_str());
  } else {
    Serial.println("Error creating/documenting in Firestore.");
    Serial.println(fbdo.errorReason());
  }
}
```

Figure 35(Code snippet7 saveDataTofirestore)

Gps

sketch_may7c.ino

```
#include <Arduino.h>
#include <SoftwareSerial.h>
#include <TinyGPS++.h>

#define GPS_RX_PIN 15   // Connect NEO-6 TX to this pin
#define GPS_TX_PIN 14    // Connect NEO-6 RX to this pin
#define GPS_BAUD_RATE 9600

SoftwareSerial gpsSerial(GPS_RX_PIN, GPS_TX_PIN);
TinyGPSPlus gps;

void setup() {
  Serial.begin(115200);
  gpsSerial.begin(GPS_BAUD_RATE);
}

void loop() {
  while (gpsSerial.available() > 0) {
    if (gps.encode(gpsSerial.read())) {
      if (gps.location.isValid()) {
        Serial.print("Latitude: ");
        Serial.println(gps.location.lat(), 6);
        Serial.print("Longitude: ");
        Serial.println(gps.location.lng(), 6);
      }
    }
  }
}
```

Figure 36(Code snippet8 GPS)

`

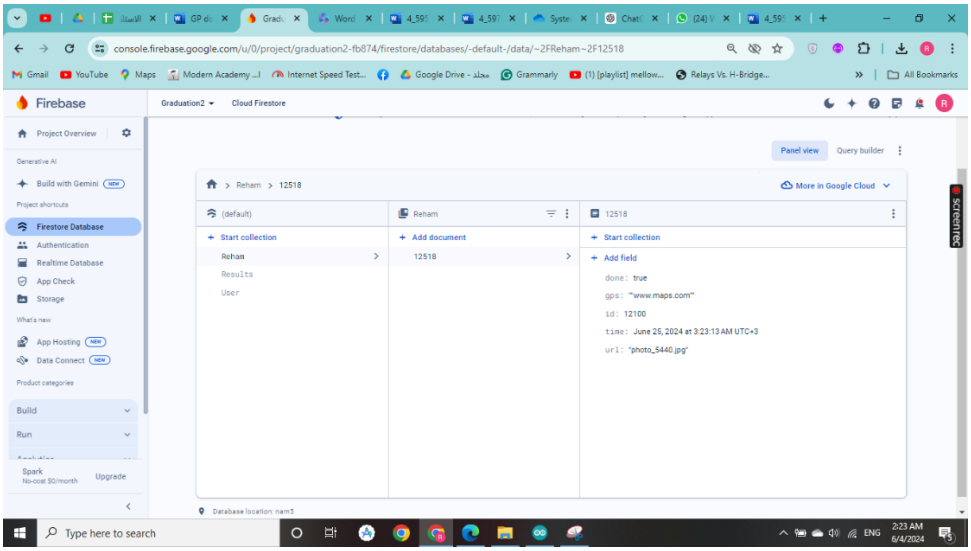


Figure 37 RehamCollectiononFirebase

4- GPS module neo6 for navigation and positioning.
The connection:

We will connect the Rx pin on pin 15 on the board and Tx pin on pin 14.

Figure 38 (GPS Neo 6)

`

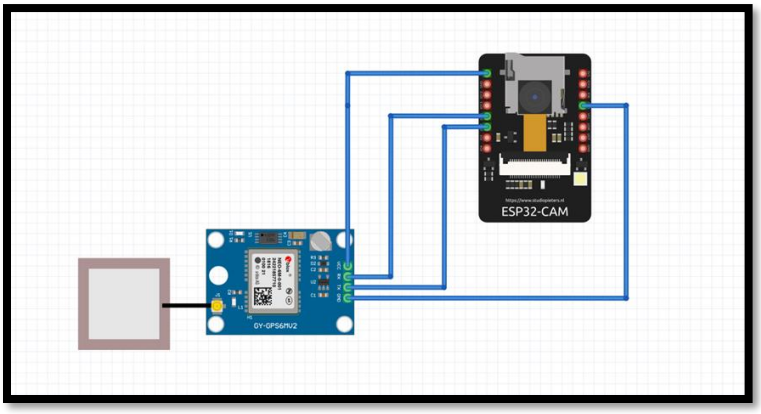


Figure 39(Simulation 2)

`

5.2) AI implementation

5.2.1) Object Detection Model

Object Detection Model:
First, we gather data sets by ourselves by capturing hundreds of images of objects.
Sample of dataset:

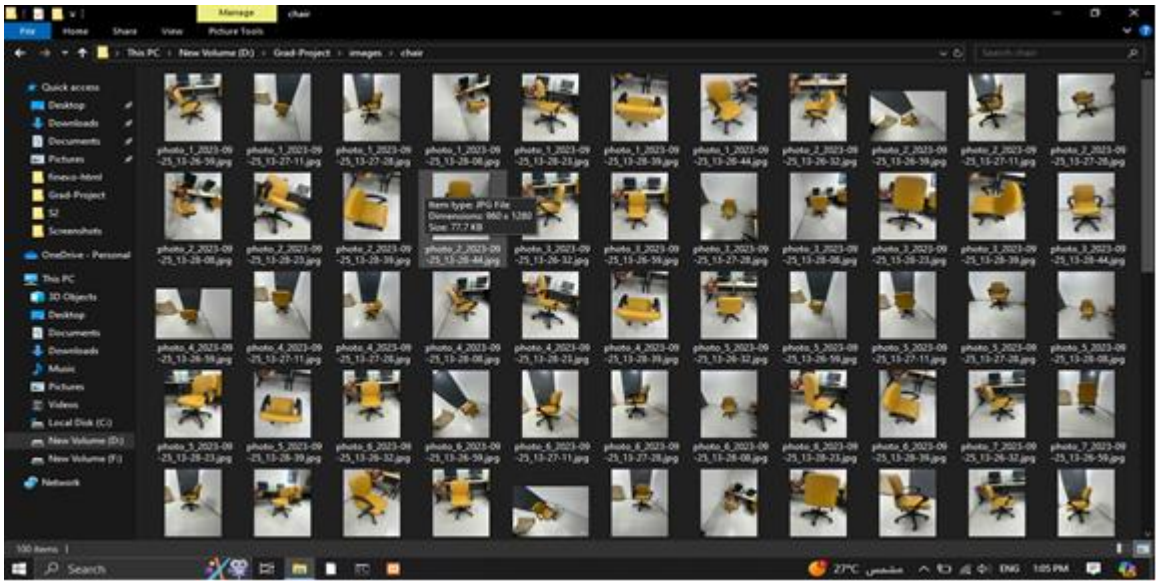


Figure 40(Collected data-set 1)

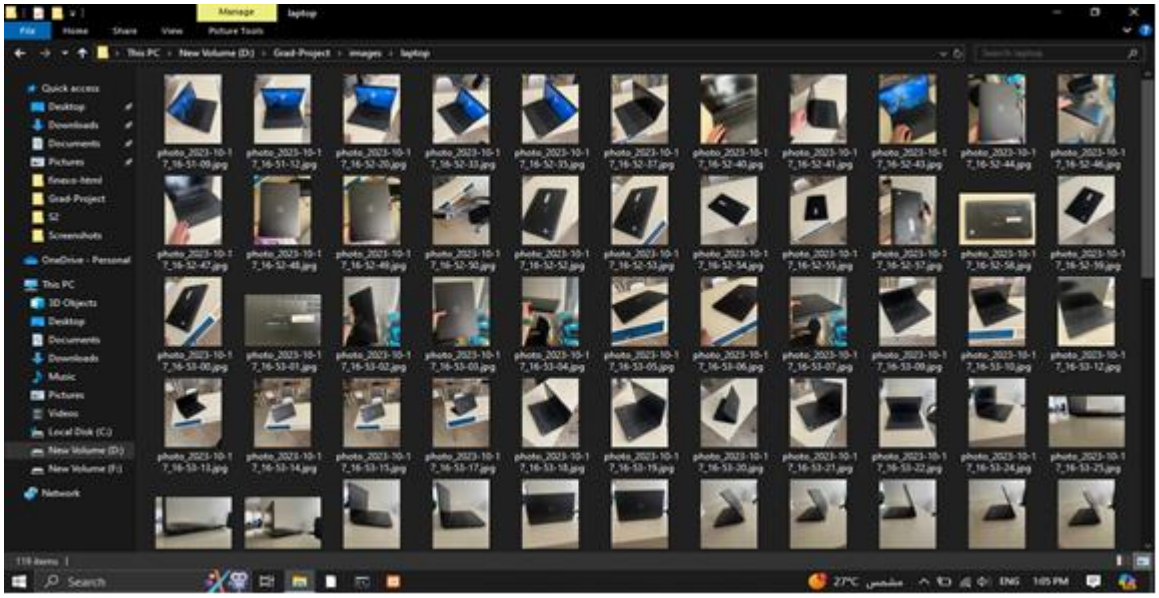


Figure 41(Collected data-set 2)

`

Secondly, preprocessing on the data:

- Annotation on the images by using labeling application
- Resize and normalize to all images to have the same size and pixels
- Augmentation on images to increase their number with different angels and rotations to achieve better accuracy

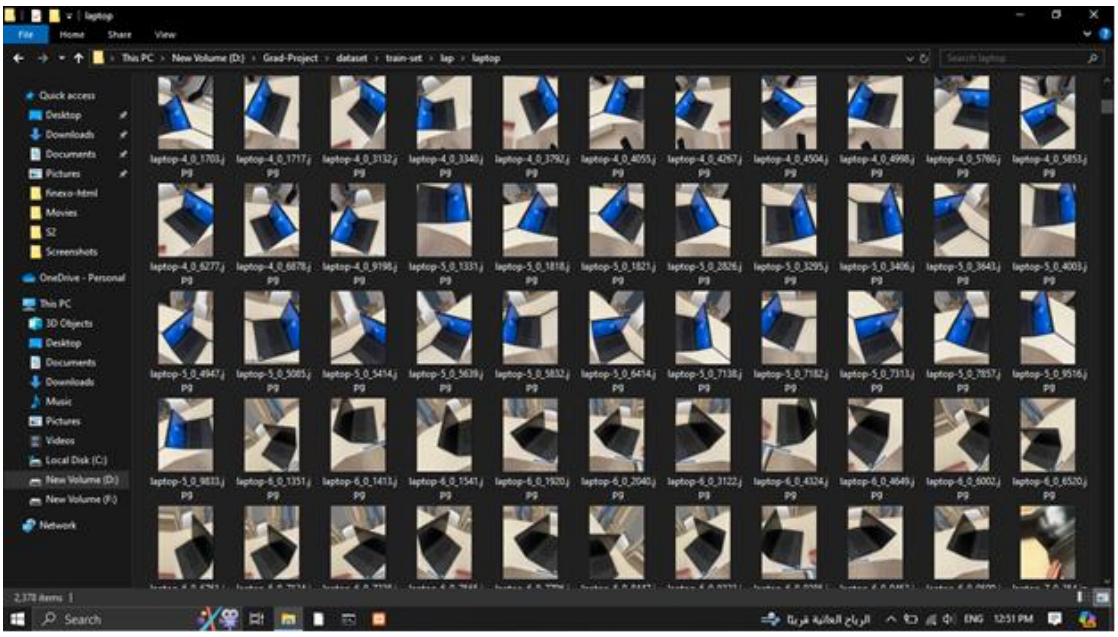


Figure 42 (Processed data-set 1)

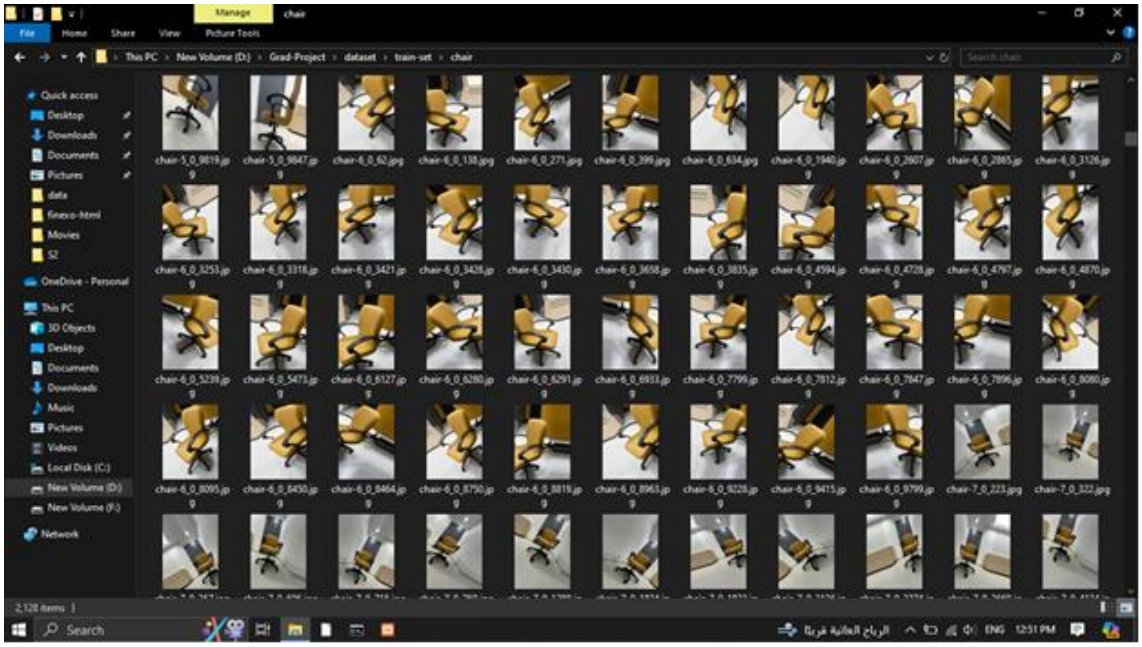


Figure 43(Processed data-set 2)

`

Then we tried to build a CNN model from scratch to train and test these data

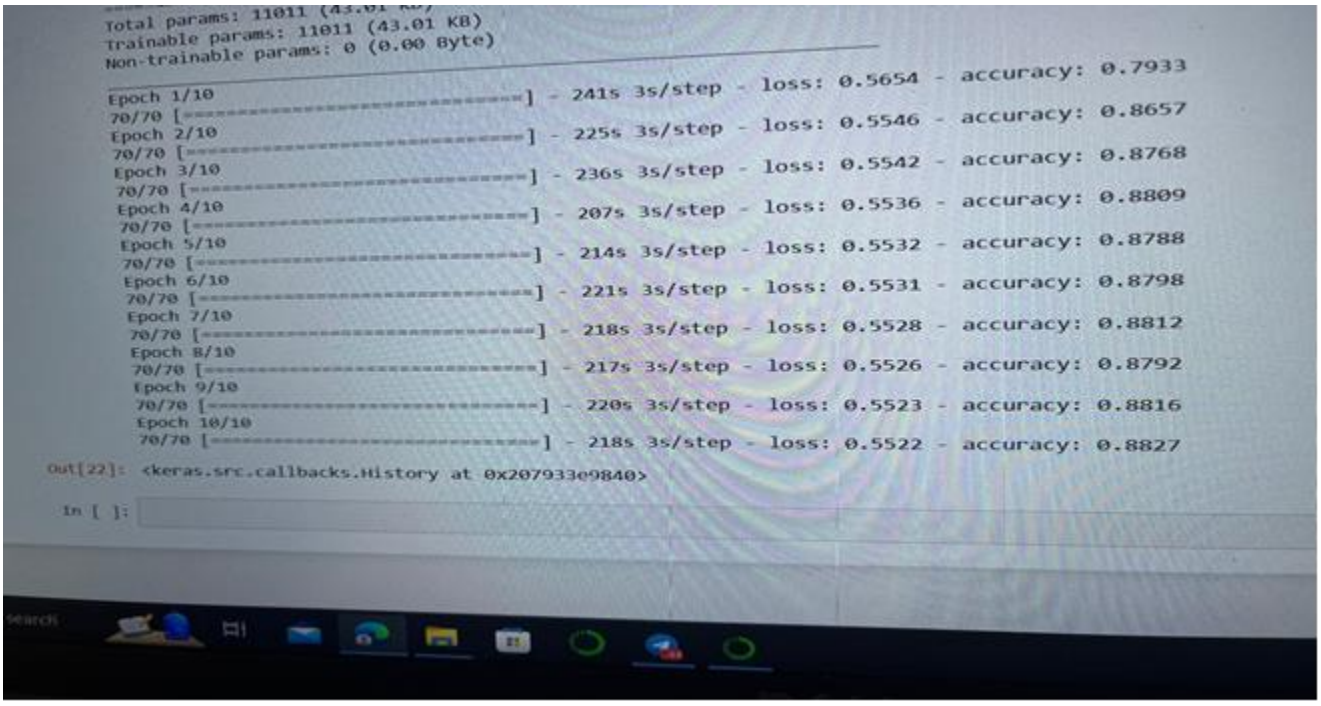


Figure 44 (CNN model)

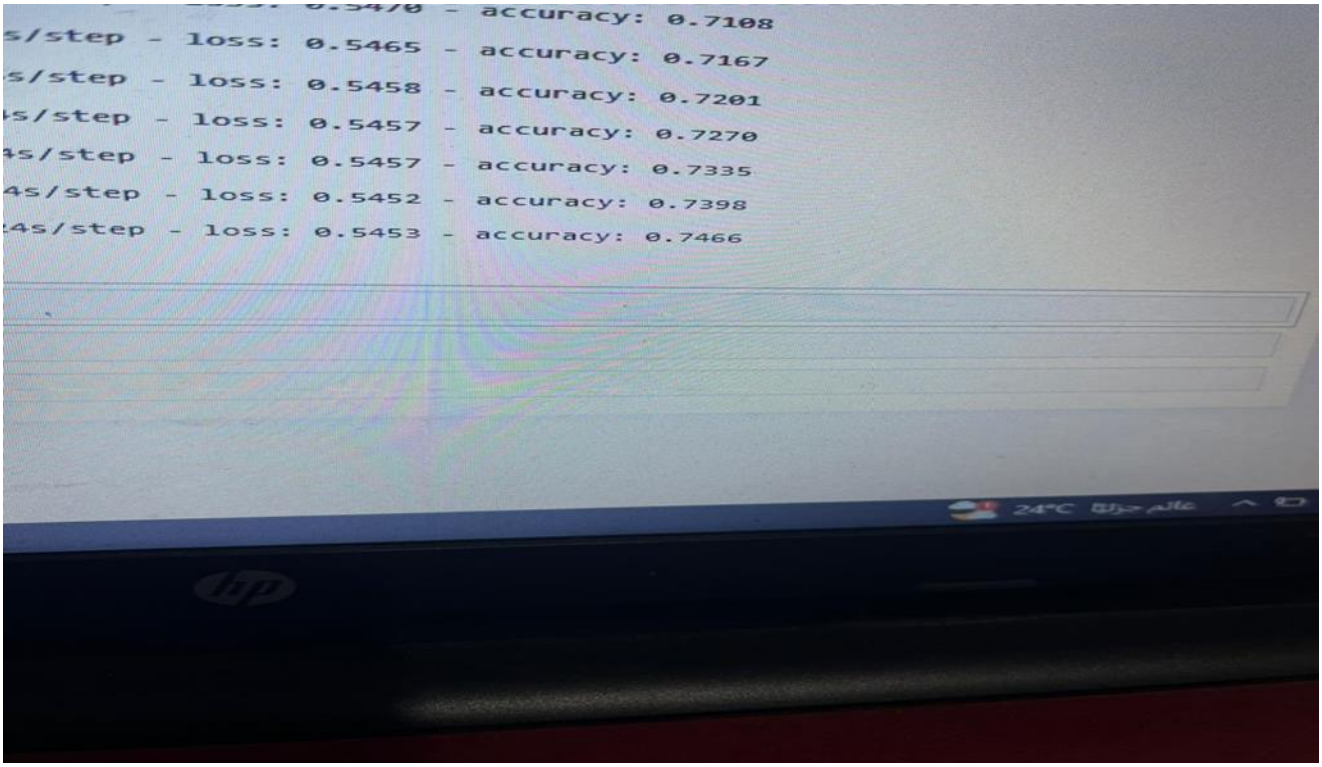


Figure 45 (CNN accuracy)

`

but it was not the best solution for object detection.
After searching we found that the best solution is to use YOLO object detection model, it is a pre-trained object detection model made with CNN architecture.

The model work with accurate and fast algorithm, in which:

1. divide the input image to number(x) of images of equal size
2. process on each small image and return coordinates of the object
3. merge all coordinates coming from all small images to make one bounding box on object on the input image
4. return the output image with the detection and accuracy

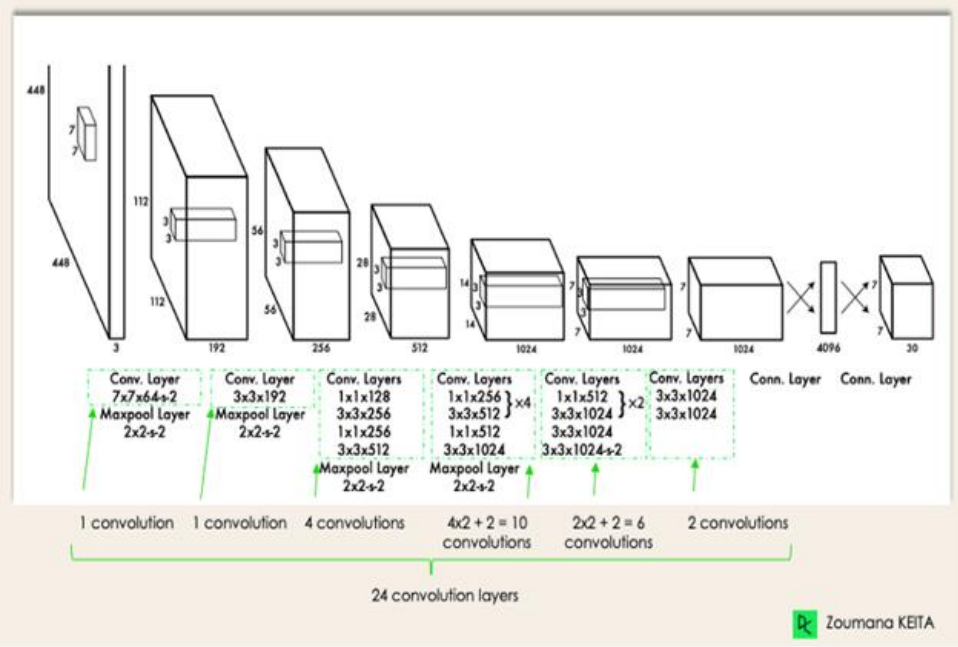


Figure 46 (CNN convolution detail)

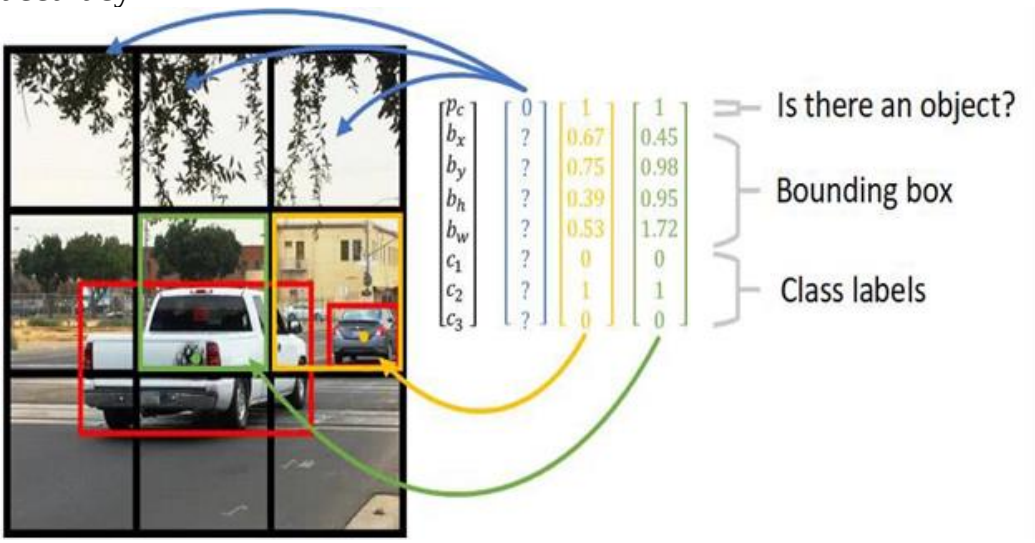


Figure 47(Object detection 1)

We used YOLO version 8 because of its high accuracy and high speed in real time detection, so we customized the model with our dataset and achieve best solutions.

`

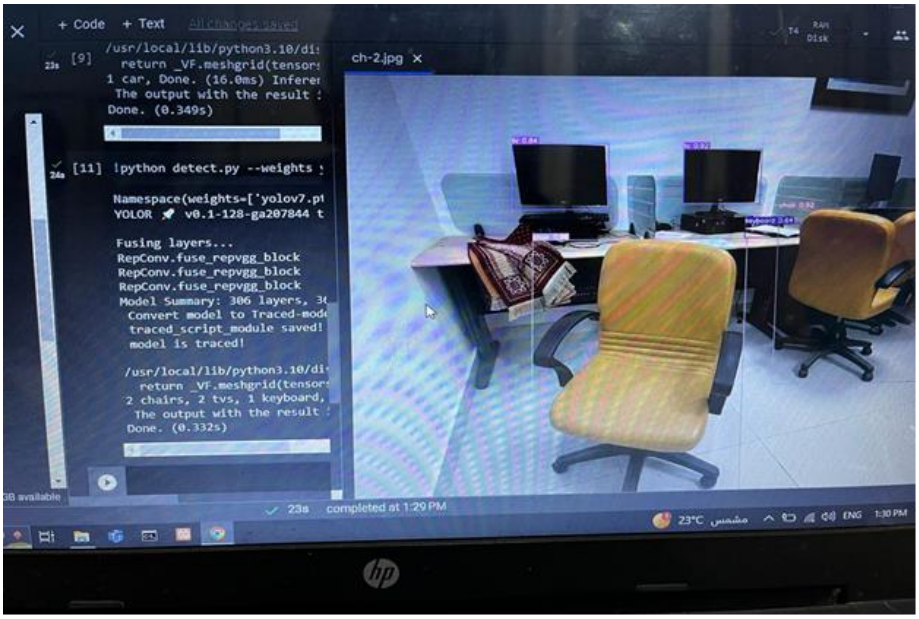


Figure 49 (Object detection 2)

Figure 50 (Object detection 3)

Figure 48(Object detection 4)

`

5.2.2) face recognition model

Implementing a face recognition model involves several steps. Here's a breakdown of the implementation process:

Requirement Gathering and Planning:

Define the requirements of face recognition system, including the target platform (e.g., desktop application, web service), we need this model to make user's interaction easy to login mobile and web application.

Environment Setup:

Set up our development environment with the necessary tools and libraries. Installed Python, a code editor, and libraries such as OpenCV and dlib for face detection and encoding.

Figure 52(OpenCV)

Figure 51(Dlib)

`

1- Data Collection and Preprocessing:

Gather a dataset of face images containing the individuals we want to recognize (7 persons).

Preprocess the images such as resizing, normalization, and alignment to ensure consistency in facial appearance.

2- Face Detection:

Implement a face detection algorithm to locate faces in images frames. We use methods like deep learning-based detectors like dlib's face detector and ORB similarity but it goes in unexpected way with loss in similarity.

```
def orb_sim(img1, img2):
  # SIFT is no longer available in cv2 so using ORB
  orb = cv2.ORB_create()

  kp_a, desc_a = orb.detectAndCompute(img1, None)
  kp_b, desc_b = orb.detectAndCompute(img2, None)

  bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

  matches = bf.match(desc_a, desc_b)
  similar_regions = [i for i in matches if i.distance < 50]
  if len(matches) == 0:
    return 0
  return len(similar_regions) / len(matches)

def structural_sim(img1, img2):

  sim, diff = structural_similarity(img1, img2, full=True)
  return sim
```

Figure 53(Code snippet Face recognition)

`

-ORB_create(): This is a method provided by OpenCV to create an ORB object. It initializes an ORB detector with default parameters.

When we create an ORB detector object using cv2.ORB_create(), we use it to detect keypoints and compute descriptors in images. Keypoints are points of interest or distinctive features in an image, while descriptors are numerical representations of the keypoints' local neighborhood.

- kp_a will contain a list of keypoints detected in img1, and desc_a will contain their corresponding descriptors. These keypoints and descriptors used for various computer vision tasks such as feature matching.

-BFMatcher: This is a class in OpenCV used for matching keypoint descriptors between two images. "BF" stands for "Brute Force," which means it considers all possible pairs of descriptors from the two sets and finds the best matches.

-cv2.NORM_HAMMING: This parameter specifies the distance measurement method used for comparing descriptors. In this case, it's using the Hamming distance, which is commonly used with binary descriptors such as ORB (Oriented FAST and Rotated BRIEF) descriptors.

3- Face Encoding:

   Use a face encoding library to extract feature vectors (encodings) from detected faces. Dlib provides methods to compute face embeddings, which represent unique characteristics of each face.

-face_recognition.face_encodings(): This function computes the facial encodings for faces detected in the input image. It takes the input image (real_time_photo) as its argument.

- real_time_photo: This is the input image for which facial encodings are being computed. It is a NumPy array representing an image.
- Drive_image: This is the images we need to compare with.

-len(face_encodings_1) > 0 and len(face_encodings_2) > 0: This condition checks if both face_encodings_1 and face_encodings_2 contain at least one facial encoding each. It ensures that there are faces detected in both images before proceeding with the comparison.

-results = face_recognition.compare_faces(face_encodings_1, face_encodings_2, tolerance=0.5): If both images contain detected faces, this line compares the facial encodings of the faces from face_encodings_1 with the facial encodings of the faces from face_encodings_2.

`

It returns a list of boolean values indicating whether the corresponding pairs of faces are considered to be the same person or not.

-The tolerance parameter: specifies the maximum distance threshold within which two facial encodings are considered to be a match.

4- Integration with Application:

Integrate the face recognition model into the target application or system where face recognition functionality is required. This involve developing APIs to enable seamless interaction between

Figure 54 (Face recognition result)

the model and the application.

5- Integration with User Interface: Develop a user interface (UI) for interacting with the face recognition system. Implement feaures such as face capture and recognition.
6- Testing and Debugging:

Test the system thoroughly to identify and fix any issues or bugs. Perform unit tests, integration tests, and end-to-end tests to ensure the system functions correctly in different scenarios.

7- Deployment:

Deploy the face recognition system to the target environment. This made by a server(Ngrok).

Configure the deployment environment, set up any dependencies, and ensure scalability and reliability.

`

5.3) Back-end implementation

findUserByEmail: This function is an asynchronous function that searches for a user by their email in a Firestore database. It takes an email as input, queries the Firestore collection named "User" for documents where the "email" field matches the

```
async function findUserByEmail(email) {
  const usersCollection = collection(db, "User");
  const q = query(usersCollection, where("email", "==", email));
  const querySnapshot = await getDocs(q);
  return !querySnapshot.empty;
}
```

Figure 55(Code snippet Backend findUserByEmail)

provided email. If there are any documents matching the query, it returns true, indicating that the user exists; otherwise, it returns falsefindUserByEmail: This function is an asynchronous function that searches for a user by their email in a Firestore database.

Figure 56 (Code snippet Backend signUp)

signUp: This function is an asynchronous function that handles user signup requests. It expects data such as name, password, email, and glove_id in the request body. If any of these fields are missing, it returns a 400 status with an error message.

It then checks if a user with the provided email already exists by calling the findUserByEmail function. If the user exists, it returns a 400 status with an
"Email already in use" message. If the user does not exist, it adds a new document to the "User" collection in the Firestore database with the provided user information and returns a 200 status with a "User created successfully" message.

`

```
export const checkimage = async (req, res) => {
  try {
    if (!req.query.glove_id) {
      throw new Error("no glove id");
    }
    if (!req.file) {
      throw new Error("no image");
    }
    console.log("passed auth");
    console.log(req.file.path);

    // Read the file contents
    const fileData = fs.readFileSync(req.file.path);

    // Create FormData object
    const formData = new FormData();
    formData.append("file", fileData, { filename: req.file.originalname });

    // Axios POST request to send the file
    const response = await axios.post(detectURL + "/detect", formData, {
      headers: {
        ...formData.getHeaders(), // Get FormData headers
      },
    });
    //upload image path on firebase and the objects found
    console.log(response.data["objects"]);
    const result = new images(
      req.query.glove_id,
      req.file.path.split("\\")[1],
      response.data["objects"]
    );
    const obj = result.toObject();
    await addDoc(collection(db, "Results"), obj);
    console.log("added");
    res.status(200).json(response.data["objects"]);
  } catch (error) {
    res.status(500).json({ error: error.message });
```

Figure 57 (Code snippet Backend CheckImage)

checkimage: This function is an asynchronous function that handles image checking requests. It expects a glove_id in the query and an image file in the request. If the glove_id or image file is missing, it returns a 500 status with an appropriate error message. It reads the contents of the image file, creates a FormData object, and sends a POST request using Axios to a detection URL with the image data. It then processes the response, saves the image path and detected objects to Firestore, and returns a 200 status with the detected objects.

`

```
export const expose = (req, res) => {
  console.log(req.query.url);
  detectURL = req.query.url;
  res.status(200).json({ msg: "done" });
};
```

Figure 58 (Code snippet Backend expose)

expose: This function handles requests to update the detection URL. It expects a URL in the query and updates the detectURL variable accordingly. It returns a 200 status with a "done" message.

```
export const Search = async (req, res) => {
  const { searchValue } = req.body;

  if (!searchValue) {
    return res.status(400).send({ error: "searchValue is required" });
  }

  try {
    const collectionRef = collection(db, "Results");
    const snapp = await getDocs(collectionRef);

    if (snapp.empty) {
      return res.status(404).send({ error: "Location not found" });
    }

    const firstDoc = snapp.docs[0];
    const location = firstDoc.data().location; // Assuming 'location' is a f

    res.status(200).json({ location });
  } catch (error) {
    console.error("Error fetching data", error);
    res.status(500).json({ error: "Internal Server Error" });
  }
```

Figure 59 (Code snippet Backend Search)

Search: This function is an asynchronous function that handles search requests. It expects a searchValue in the request body. If the searchValue is missing, it returns a 400 status with an error message. It queries the Firestore collection named "Results" and retrieves the first document. It then extracts the "location" field from the document and returns a 200 status with

`

the location. If no documents are found, it returns a 404 status with an "Location not found" message. If any errors occur during the process, it returns a 500 status with an "Internal Server Error" message.

```
import express from "express";
import cors from "cors";
import firebase from "./firebase.js";
import config from "./config.js";
import userRoute from "./Route/userRoute.js";
import morgan from "morgan";
import helmet from "helmet";
import bodyParser from "body-parser";
import multer from "multer";
import path from "path";
//import Date from "Date";
const __dirname = path.resolve();
const app = express();

app.use(cors());

// Middleware to parse incoming JSON requests with a limit of 2mb
app.use(bodyParser.json({ limit: "2mb" }));

// Middleware to parse incoming URL-encoded data with extended mode so it can accept files
app.use(bodyParser.urlencoded({ extended: true }));

// Middleware to parse incoming JSON requests with a specific content type and disable inflation
app.use(bodyParser.json({ type: "application/*+json", inflate: false }));

// Middleware to enhance security by setting various HTTP headers
app.use(helmet());

// Middleware for logging HTTP requests using Morgan middleware with a custom format only for logging to check if the request is sent correctly
app.use(
  morgan(":method :url :status :res[content-length] - :response-time ms")
```

Figure 60(Code snippet Backend Express.js)

This index.js file is for setting up a server using Express.js framework in Node.js. Let's break down its functionality:

Imports: The code imports necessary modules like Express, CORS, Firebase, configuration files, routes, logging middleware (Morgan), security middleware (Helmet), and body parsing middleware (BodyParser).

`

```
app.use(express.json());
const filestorage = multer.diskStorage({
  destination: (req, file, cb) => {
    return cb(null, "images");
  },
  filename: (req, file, cb) => {
    var date = Date.now();
    console.log(date);
    return cb(
      null,
      req.query.glove_id + "-" + date + "." + file.mimetype.split("/")[1]
    );
  },
});
const filter = (req, file, cb) => {
  if (!req.query.glove_id) {
    return cb(null, false);
  }
  if (
    file.mimetype === "image/png" ||
    file.mimetype === "image/jpg" ||
    file.mimetype === "image/jpeg"
  ) {
    return cb(null, true);
  }
  else {
    return cb(null, false);
  }

};


app.use(multer({ storage: filestorage, fileFilter: filter }).single("image"));
app.use('/image', express.static(path.join(__dirname, "images")))
app.use("/api", userRoute);
```

Figure 61(Code snippet Backend Express.js2)

Express App Setup: It initializes an Express application.

CORS Configuration: It enables Cross-Origin Resource Sharing (CORS) using the cors middleware, allowing the server to handle requests from different origins.
Body Parsing Middleware: It sets up middleware for parsing incoming requests with various options:
JSON parsing with a limit of 2mb.
URL-encoded data parsing with extended mode to accept files.
Specific JSON content type parsing with inflation disabled.
Security Middleware: It enhances security by setting various HTTP headers using the helmet middleware.
Logging Middleware: It logs HTTP requests using the morgan middleware with a custom format.

`

Static File Serving Configuration: It sets up a route to serve static files from the images directory using Express's built-in express.static middleware.

Multer Configuration: It configures Multer for handling file uploads. Multer is a middleware for handling multipart/form-data, which is primarily used for uploading files. It specifies the destination folder and filename for uploaded files and defines a filter to accept only certain types of images (png, jpg, jpeg).

Routes: It sets up routes for handling API requests. In this case, it uses the userRoute for handling requests starting with /api.

Fallback Route: It defines a catch-all route to respond with "working" for any unhandled requests, just to ensure that the server is up and running.

Server Listening: It starts the server, specifying the port from the configuration file and logging the server's URL.

In summary, this code sets up a server with various middleware for handling requests, serving static files, and processing file uploads, along with defining routes for handling API requests.

```
import jwt from 'jsonwebtoken'
export const check_auth_user = async (req, res, next) => {
    try {
        const token = req.get('Authorization').split(' ')[1]
        const decode = jwt.verify(token, 'passwordkey')
        if (!decode)
            throw new Error("auth_failed")
    } catch (error) {
        req.error = "auth_failed"
    }
    finally { next() }

};
```

Figure 62(Code snippet Backend. checkAuthUser)

This middleware can be used to protect routes that require authentication. It verifies the JWT provided in the request headers and passes the request along if the token is valid. Otherwise, it sets an error flag on the request object for handling authentication failures downstream.[Text Wrapping Break]JWT Import: It imports the jsonwebtoken library for working with JWT.

`

Middleware Function: check_auth_user is an asynchronous middleware function that takes three parameters: req (request), res (response), and next (next middleware function).

Token Extraction: It attempts to extract the JWT from the request headers. It assumes that the token is included in the Authorization header using the Bearer scheme (Authorization: Bearer <token>).

Token Verification: It attempts to verify the extracted token using the jwt.verify method. It uses a hardcoded secret key ('passwordkey') for decoding the token.

Error Handling: If there's an error during token verification (e.g., token is invalid or expired), or if no token is provided, it catches the error and sets req.error to "auth_failed". This allows downstream middleware or route handlers to handle the authentication failure appropriatel.

Next Function: Finally, it calls the next() function to proceed to the next middleware in the chain. Whether the token verification succeeds or fails, the request processing continues

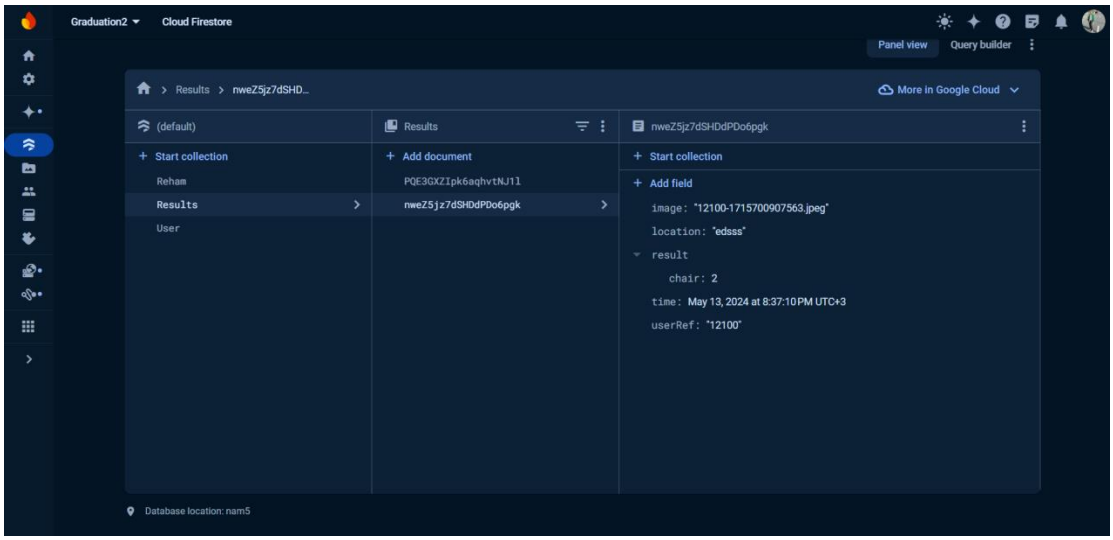


Figure 63(Firebase collection results)

`

```
class images {
    constructor(userRef, image, result , location , time) {
        (this.userRef = userRef),
            (this.image = image),
            (this.result = result),
            (this.location = location),
            (this.time = time)
            ;
    }
    toObject() {
        return {
            userRef: this.userRef,
            image: this.image,
            result: this.result,
            location : this.location,
            time : this.time

        };
    }
}

export default images;
```

Figure 64(Class model Image)

This class used to be model image data within a Firebase database. Each instance of the images class represents a single image entry in the database, with properties storing relevant information such as the image itself, associated user reference, location, and timestamp. It provides a convenient way to work with image data in your Firebase application.[Text Wrapping Break][Text Wrapping Break]

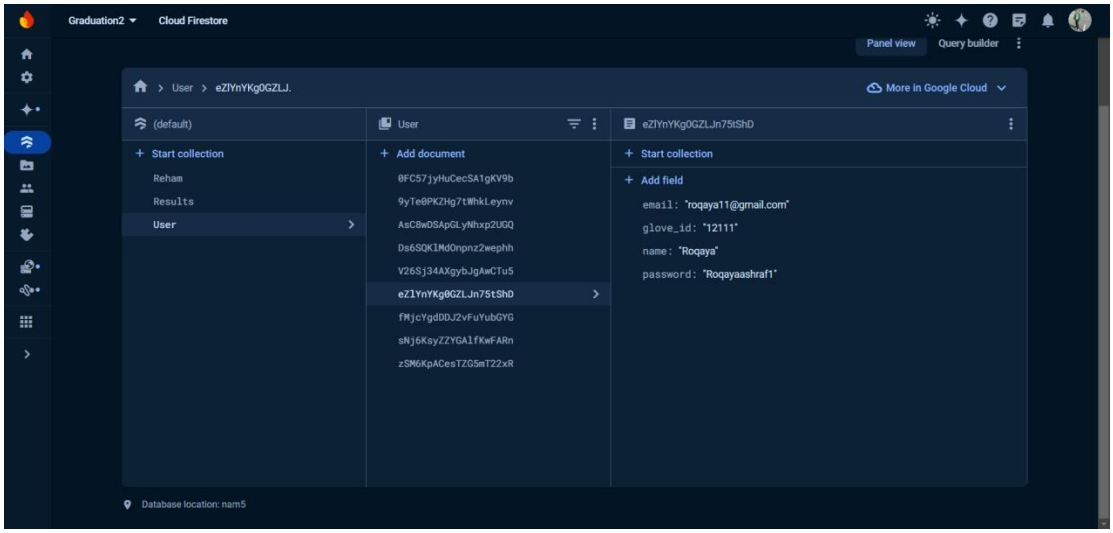


Figure 65 (Firebase Collection user)

`

```
class User {
  constructor(id, name,password,email,photo,glove_id) {
    (this.id = id),
      (this.name = name),
      (this.password = password),
      (this.email = email),
      (this.photo = photo),
      (this.glove_id = glove_id);
  }
}

export default User;
```

Figure 66 (Class model User)

Regarding Firebase, this class be designed to model user objects in a Firebase application. You can use instances of this class to represent users in your Firebase database, where each instance corresponds to a user entry.

You would typically interact with Firebase to perform CRUD (Create, Read, Update, Delete) operations on user data, utilizing instances of this class to represent user objects in your application's code.

`

5.4) Back-end integration with Front-end

```
import React from "react";
import { useResult } from "../hooks/useFirestore.js";
import './results.css'
export default function Results({gloveId}) {
  const results = useResult()
  console.log(results)




  return (
    <>
<div className="container">
      {results ? results.map((item, index) => (
        <div className="card" key={index}>
          {Object.keys(item.result).map((key, innerIndex) => (
            <p key={innerIndex}>{`${key}: ${item.result[key]}`}</p>
          ))}
        </div>
      ))
     :<p>no results available</p>
     }
     </div>


    </>
  );
}
```

Figure 67 (Backend integration 1)

Import Statements: The component imports necessary dependencies including React, a custom hook useResult, and a CSS file for styling.
Function Component: The Results component is a functional component that accepts a gloveId prop (though it is not used within the component).

useResult Hook: This hook fetches the results data. The results variable will contain the data fetched from Firestore.

Rendering Logic:
It checks if results is available.
If results is available, it maps through each item and displays the key-value pairs inside a card.
If no results are available, it displays a message saying "no results available".

`

```
import { useSyncExternalStore } from "react";
import {firebaseStore , resultStore} from "./firebaseStore.js"
export const useFirestore = () => {
    const data = useSyncExternalStore(firebaseStore.subscribe, firebaseStore.getSnapshot);
    return data;
  };

export const useResult = ()=>{
  const data = useSyncExternalStore(resultStore.subscribe, resultStore.getSnapshot);
    return data;
};
```

Figure 68 (Backend integration 2)

useSyncExternalStore: Both hooks use this React hook to manage subscriptions to the external Firestore stores (firebaseStore and resultStore).

Subscription Logic: The subscribe method in each store sets up Firestore snapshot listeners to react to real-time updates.

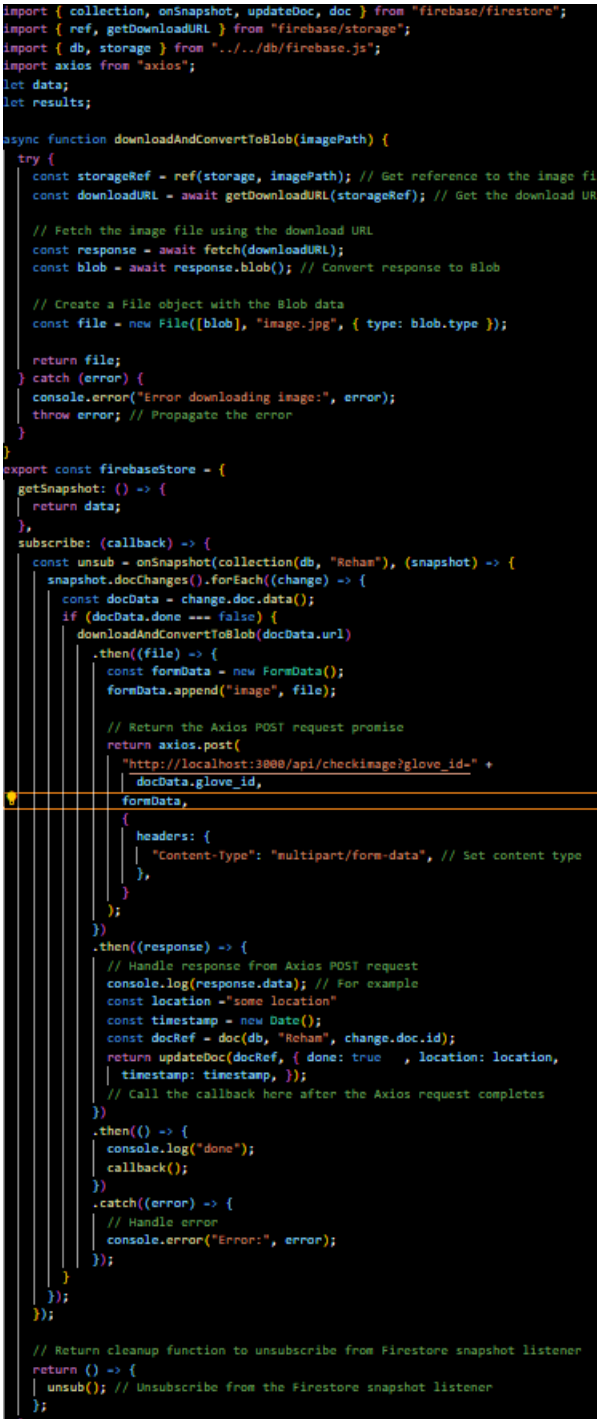
Snapshot Retrieval: The getSnapshot method in each store returns the current state of the data.

```
export const resultStore = {
  getSnapshot: () => {
    return results;
  },

  subscribe: (callback) => {
    const unsub = onSnapshot(collection(db, "Results"), (snapshot) => {
      results = [];
      snapshot.docChanges().forEach((change) => {
        //console.log(change)
        const docData = change.doc.data();
        results.push(docData);
      });
      callback();
    });
    return () => {
      unsub();
    };
  },
};
```

Figure 69 (Backend integration 3)

`



```
import { collection, onSnapshot, updateDoc, doc } from "firebase/firestore";
import { ref, getDownloadURL } from "firebase/storage";
import { db, storage } from "../../db/firebase.js";
import axios from "axios";
let data;
let results;

async function downloadAndConvertToBlob(imagePath) {
  try {
    const storageRef = ref(storage, imagePath); // Get reference to the image fi
    const downloadURL = await getDownloadURL(storageRef); // Get the download UR

    // Fetch the image file using the download URL
    const response = await fetch(downloadURL);
    const blob = await response.blob(); // Convert response to Blob

    // Create a File object with the Blob data
    const file = new File([blob], "image.jpg", { type: blob.type });

    return file;
  } catch (error) {
    console.error("Error downloading image:", error);
    throw error; // Propagate the error
  }
}
export const firebaseStore = {
  getSnapshot: () => {
    return data;
  },
  subscribe: (callback) => {
    const unsub = onSnapshot(collection(db, "Reham"), (snapshot) => {
      snapshot.docChanges().forEach((change) => {
        const docData = change.doc.data();
        if (docData.done === false) {
          downloadAndConvertToBlob(docData.url)
            .then((file) => {
              const formData = new FormData();
              formData.append("image", file);

              // Return the Axios POST request promise
              return axios.post(
                "http://localhost:3000/api/checkimage?glove_id=" +
                  docData.glove_id,
                formData,
                {
                  headers: {
                    "Content-Type": "multipart/form-data", // Set content type
                  },
                }
              );
            })
            .then((response) => {
              // Handle response from Axios POST request
              console.log(response.data); // For example
              const location ="some location"
              const timestamp = new Date();
              const docRef = doc(db, "Reham", change.doc.id);
              return updateDoc(docRef, { done: true   , location: location,
                timestamp: timestamp, });
              // Call the callback here after the Axios request completes
            })
            .then(() => {
              console.log("done");
              callback();
            })
            .catch((error) => {
              // Handle error
              console.error("Error:", error);
            });
        }
      });
    });

    // Return cleanup function to unsubscribe from Firestore snapshot listener
    return () => {
      unsub(); // Unsubscribe from the Firestore snapshot listener
    };
```

Figure 70 (React integration using blob)

getSnapshot: This function returns the current state of results.

subscribe: This function sets up a listener on the "Results" collection in Firestore:

It uses Firestore's onSnapshot to listen for real-time updates.

When a change occurs, it clears the results array and populates it with the latest data from the snapshot.

It then calls the provided callback function to notify subscribers of the change.

The function returns an unsubscribe function to clean up the listener when it is no longer

needed.

The downloadAndConvertToBlob function handles the process of downloading an image from Firebase Storage and converting it to a File object.

The firebaseStore object manages real-time Firestore document changes:

It listens for updates in the "Reham" collection.

When a document is not marked as done, it downloads the image, posts it to an API, updates the Firestore document, and triggers the callback to notify subscribers.

`

State Management: The component uses the useState hook to manage the similarities state, which stores the results of the face comparison.

Capture Component: It leverages a Capture component to handle photo capturing, which is not detailed here but is expected to call the onCapture function prop with the captured photo.

Sending Photo: The sendPhoto function handles the process of sending the captured photo to the backend server, receiving the

response, and updating the state with the similarities data.

```
import React, { useState } from 'react';
import Capture from './Capture';

const FaceRecognition = () => {
  const [similarities, setSimilarities] = useState(null);

  const sendPhoto = async (photo) => {
    try {
      const formData = new FormData();
      formData.append('photo', photo);

      const response = await fetch('http://localhost:8000/compare_faces?photo', {
        method: 'POST',
        body: formData
      });

      const data = await response.json();
      setSimilarities(data.similarities);
    } catch (error) {
      console.error('Error sending photo:', error);
    }
  };

  return (
    <div>
      <h1>Face Recognition App</h1>
      <Capture onCapture={sendPhoto} />
      {similarities && <p>Similarities: {similarities}</p>}
    </div>
  );
};
```

Figure 71 (send photo to Object detection Api)

Conditional Rendering: It conditionally renders the similarities result, only displaying it if there is data available.

`

State Management: The component uses the useState hook to manage the mediaStream state, which holds the video stream from the webcam.

```
import React, { useState } from 'react';

const Capture = ({ onCapture }) => {
  const [mediaStream, setMediaStream] = useState(null);

  const startCapture = async () => {
    try {
      const stream = await navigator.mediaDevices.getUserMedia({ video: true });
      setMediaStream(stream);
    } catch (err) {
      console.error('Error accessing camera:', err);
    }
  };

  const takeSnapshot = () => {
    if (mediaStream) {
      const videoTrack = mediaStream.getVideoTracks()[0];
      const imageCapture = new ImageCapture(videoTrack);
      imageCapture
        .takePhoto()
        .then(blob => {
          onCapture(blob);
        })
        .catch(error => {
          console.error('Error taking snapshot:', error);
        });
    }
  };

  return (
    // <div>
    //   <button onClick={startCapture}>Start Camera</button>
    //   <button onClick={takeSnapshot}>Take Photo</button>
    // </div>
    <div class="btn-group" role="group" aria-label="Basic example">
    <button onClick={startCapture}  class="btn btn-primary">Start Camera</button>
    <button onClick={takeSnapshot}  class="btn btn-primary">Take Photo</button>
  </div>
  );
};

export default Capture;
```

Figure 72 (Face recognition code)

Start Capture: The startCapture function initializes the webcam and sets the video stream.

Take Snapshot: The takeSnapshot function captures a still image from the video stream and passes it to the onCapture prop.

Rendering: The component renders two buttons: one to start the webcam and one to take a photo. The buttons are styled using Bootstrap classes.

`

5.5) Front-end Implementation

About web page contains main information about the product

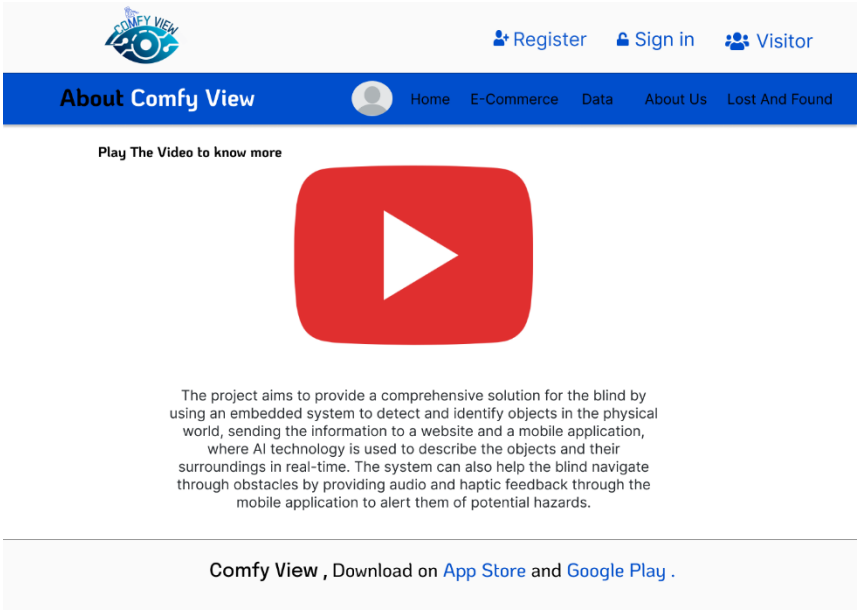


Figure 73 (Website About page)

The Result web page contains name of the founded object

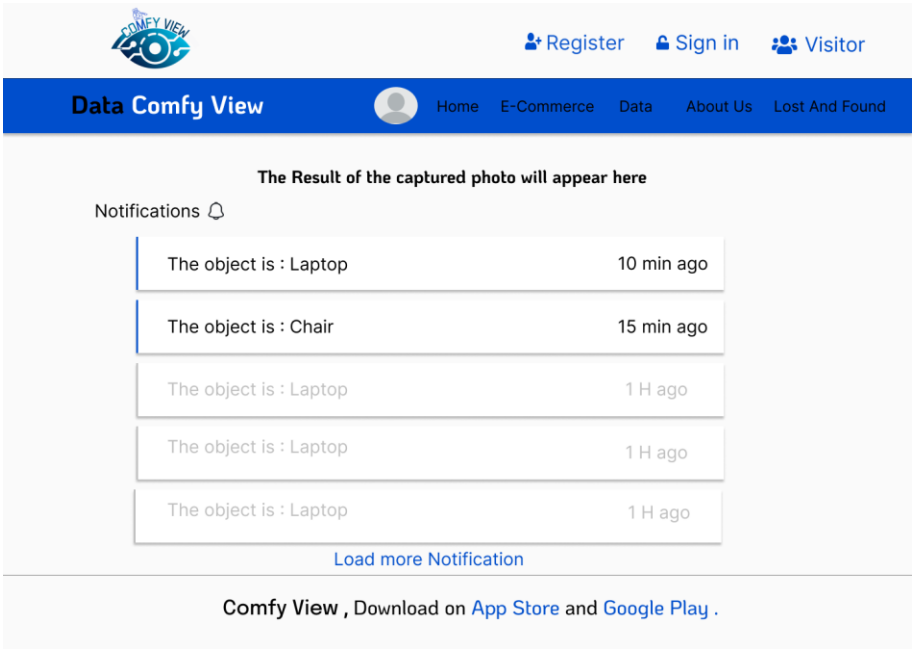


Figure 74 (Website Result page)

`

Signup web page the user can create an account

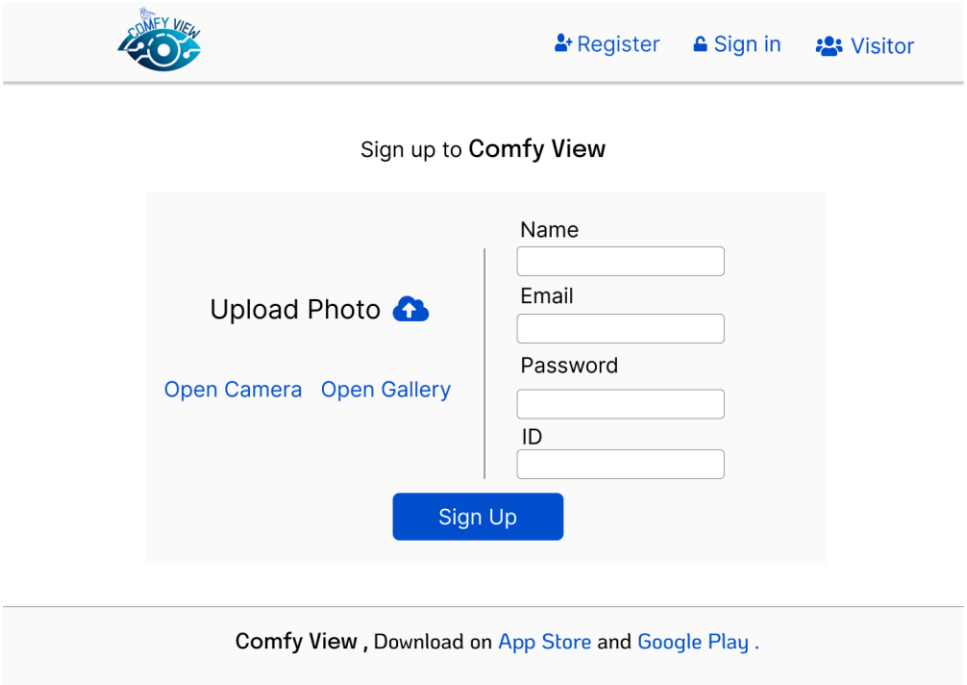


Figure 75 (Website Signup page)

Login web page contains email and password to login

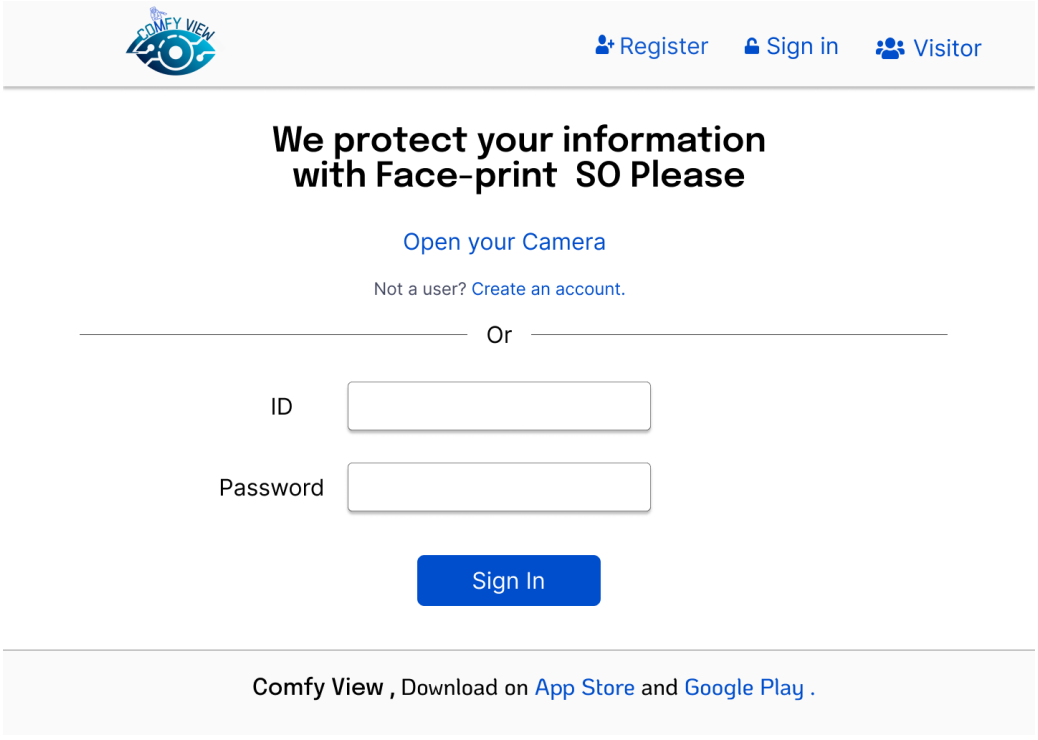


Figure 76 (Website Login page)

`

Personal information web page contains the user information

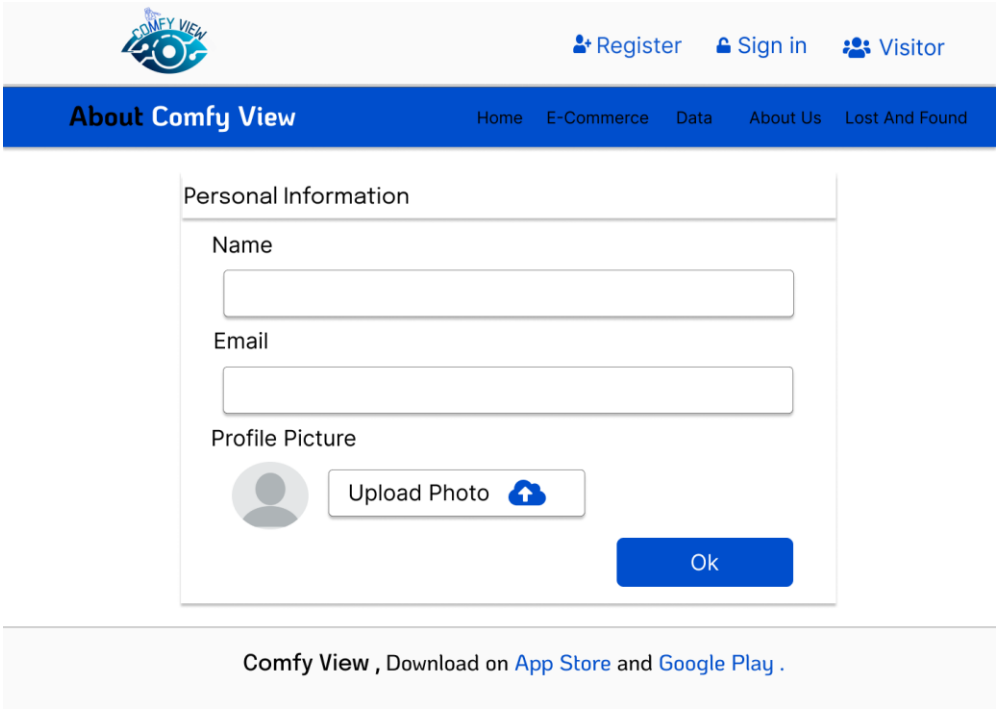


Figure 77 (Website Personal information page)

Lost and found web page searching for unfounded object

Figure 78 (Website Lost found page)

`

The landing page

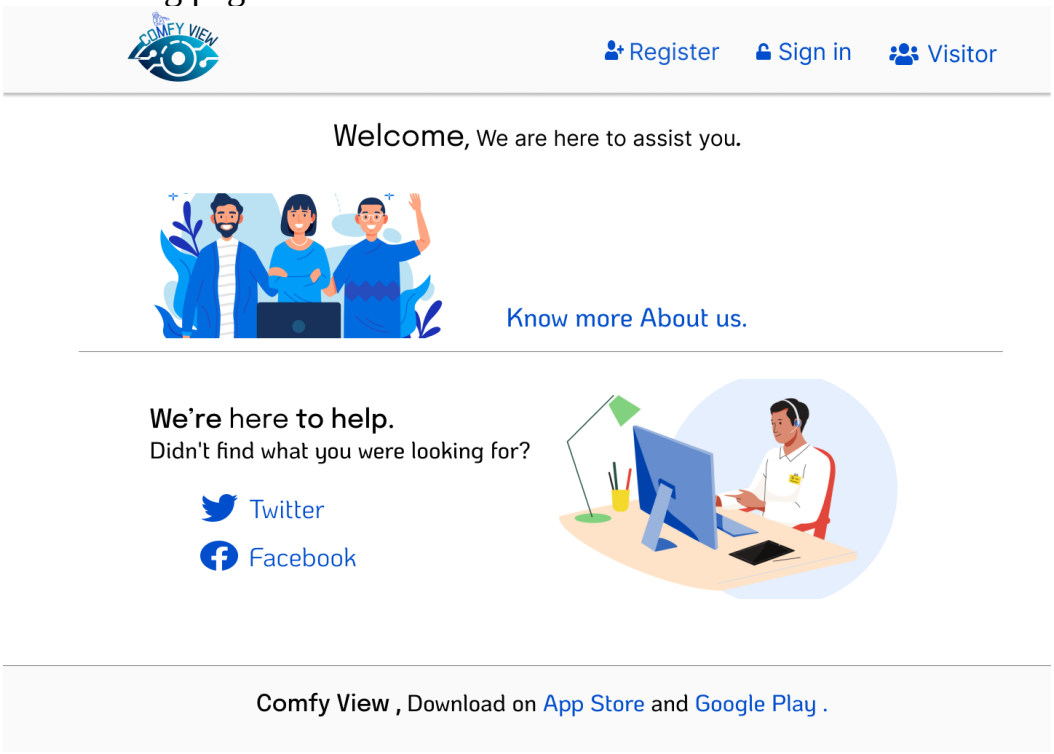


Figure 79 (Website Landing page)

The user can buy the product from this page

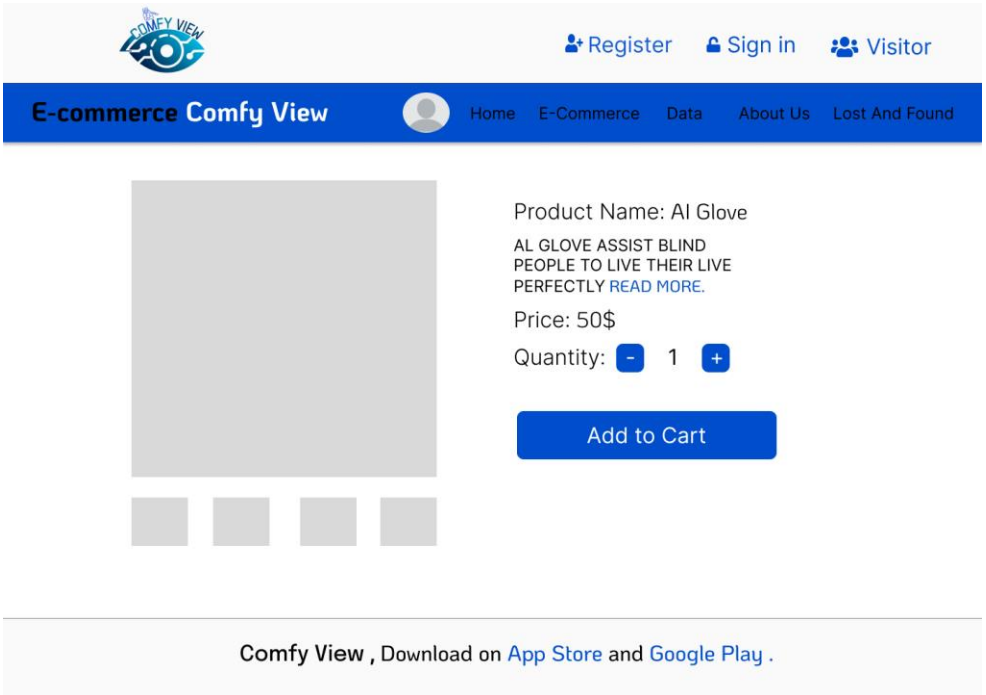


Figure 80 (Website E-commerce page)

`

5.6) Mobile application implementation

5.6.1) UI Implantation

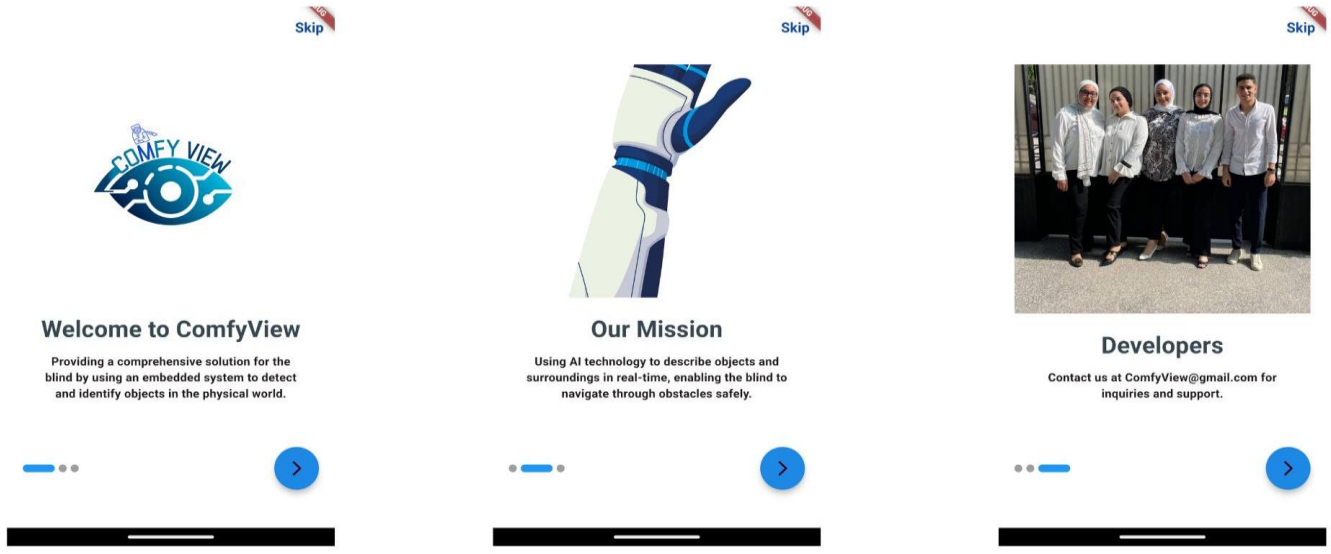


Figure 81 (Mobile application UI 1)

This UI appears to be the on-boarding screens for the Comfy View application. Let's go through each section:

- Welcome to Comfy View: This section provides a brief introduction to the Comfy View app, stating that it offers a "comprehensive solution for the blind by using an embedded system to detect and identify objects in the physical world."
- Our Mission: This section outlines the mission of the Comfy View app, which is to "Using AI technology to describe objects and surroundings in real-time, enabling the blind to navigate through obstacles safely."
- Developers: This section includes a group photo of the developers behind the Comfy View app and provides an email address (ComfyView@gmail.com) for users to contact the team for inquiries and support.

The overall design of the on-boarding screens is clean and visually appealing, with the Comfy View logo, a smooth page indicator, and a "Skip" button in the top-right corner of **each screen.**

`

The content is concise and effectively communicates the purpose and features of the Comfy View app to the user. The use of this on-boarding process is a common practice in mobile app development, as it allows users to quickly understand the app's key features and benefits before they proceed to the main functionality, such as the login screen mentioned in the code

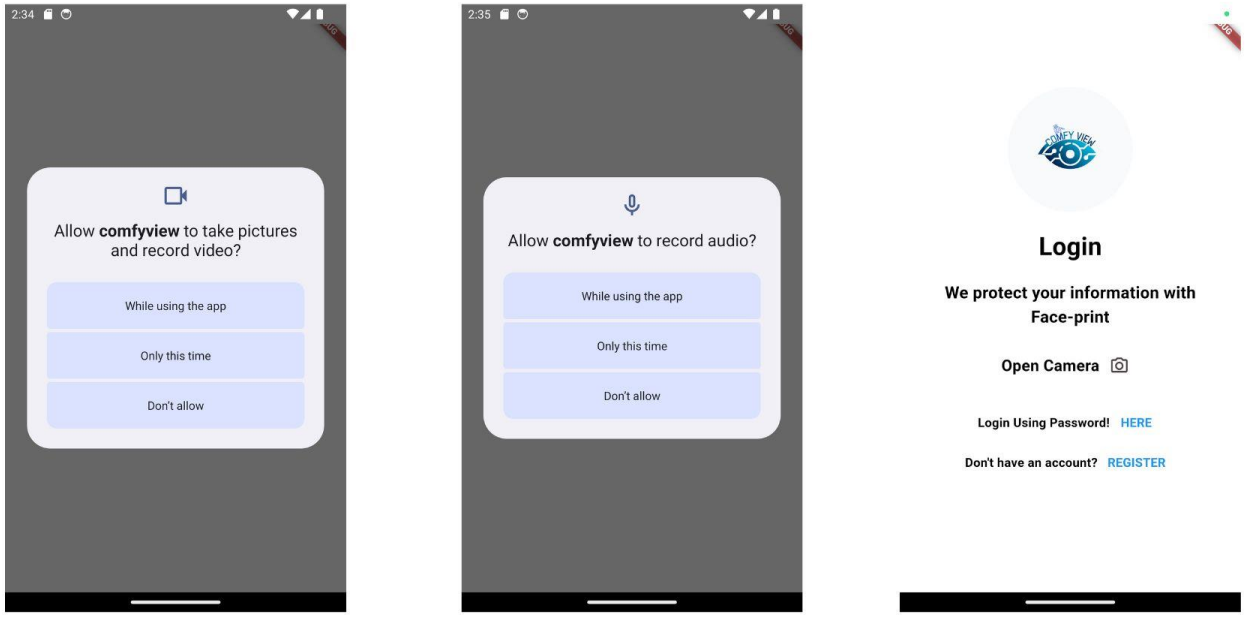


Figure 82 (Mobile application UI 2)

Screen 1 (Allow comfy view to take pictures and record video?):
This screen is requesting permission from the user to allow the app to access the device's camera and video recording capabilities. The user has the options to allow access "While using the app", "Only this time", or "Don't allow".
Screen 2 (Allow comfy view to record audio?):
This screen requests permission from the user to allow the app to access the device's microphone for audio recording. The user has the options to allow access "While using the app", "Only this time", "Don't allow".
Screen 3 (Login):
This screen appears to be a login screen for the app. It provides options for the user to "Login Using Password!" or "Register" if they don't have an account. The screen also mentions that the app "protects your information with Face-print" and provides an option to "Open Camera".

`

Overall, these screens are part of the onboarding and authentication process for the comfy view application, which seems to require permission to access the device's camera, microphone, and potentially use biometric authentication (face-print) for logging in.

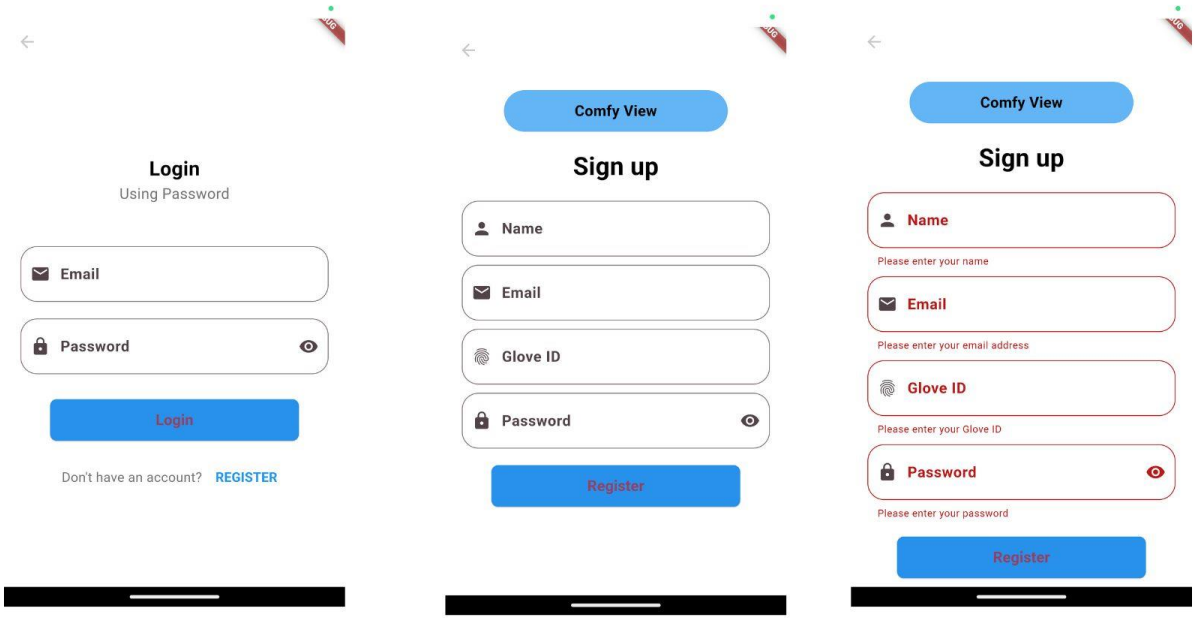


Figure 83 (Mobile application UI 3)

Login Screen:
This screen is the login screen for the application. It has input fields for the user to enter their Email and Password. There is a "Login" button and a "REGISTER" link for users who don't have an account.
Sign up Screen:
This screen is the sign-up or registration screen for the application. It has input fields for the user to enter their Name, Email, Glove ID, and Password. There is a "Register" button at the bottom.
Sign up Screen (another view):
This screen is another view of the sign-up or registration screen. It has the same input fields as the previous sign-up screen, but with slightly different formatting and label placement.

`

Overall, these screens are part of the authentication and onboarding process for the Comfy View application. The user can either log in with an existing account or create a new account by providing the required personal and account information.

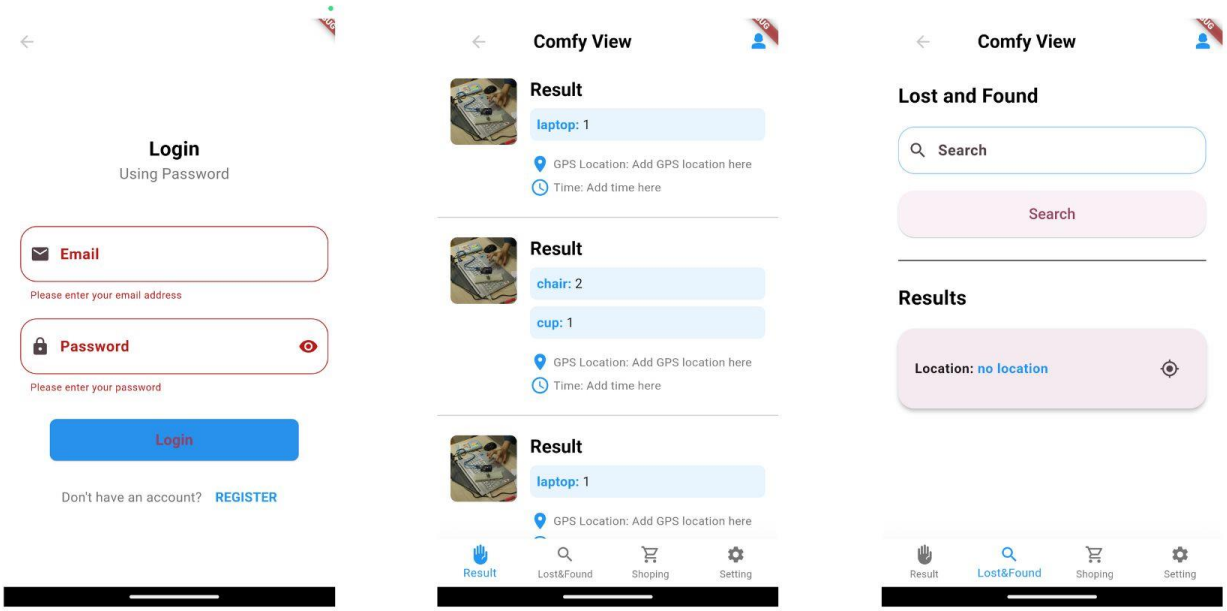


Figure 84 (Mobile application UI 4)

1-Login Screen:
This is the login screen where the user can enter their email address and password to access the application. There is also a "REGISTER" link for users who don't have an account.
2-Result Screen:
This screen displays the "Result" of a search or action performed within the app. It shows three different search results, each with information about the item (laptop, chair, cup) and the ability to add GPS location and time details.
3-Lost and Found Screen:
This screen is the "Lost and Found" section of the Comfy View app. It has a search bar where the user can search for lost or found items. Below the search bar, there is a "Results" section that displays information about the search, including the location (if available).

`

Overall, the application seems to be focused on helping users manage and track their personal belongings, with features for searching, recording details, and potentially finding lost items. The screens demonstrate the core functionality of the app, including authentication, search, and display of results.

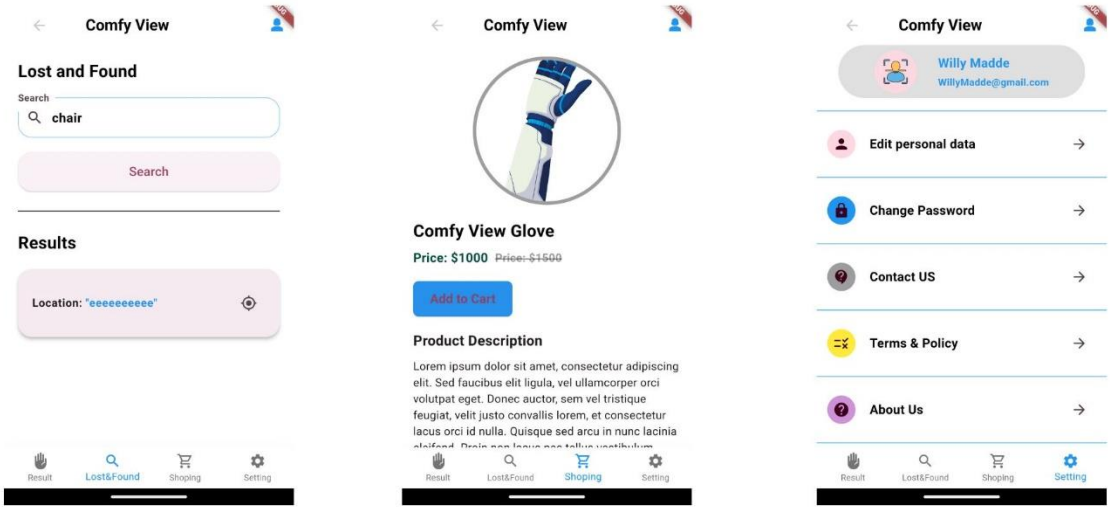


Figure 85 (Mobile application UI 5)

1-Lost and Found Screen:
This is the "Lost and Found" section of the app, where the user can search for lost or found items. The user has entered "chair" in the search bar, and the "Search" button is displayed.
2-Comfy View Glove Screen:
This screen displays a product called the "Comfy View Glove". It shows an image of the glove, its price ($1000 with a previous price of $1500), and a "Add to Cart" button. The "Product Description" section provides some lorem ipsum text about the glove.
3-User Profile Screen:
This screen shows the user's profile, with their name "Willy Madde" and email address displayed at the top. Below, there are several options for the user to manage their account, including "Edit personal data", "Change Password", "Contact US", "Terms & Policy", and "About Us".

Overall, the screens demonstrate the different features and functionalities of the Comfy View application, including the ability to search for lost/found items, view and purchase a product (the Comfy View Glove), and manage the user's account settings.

`

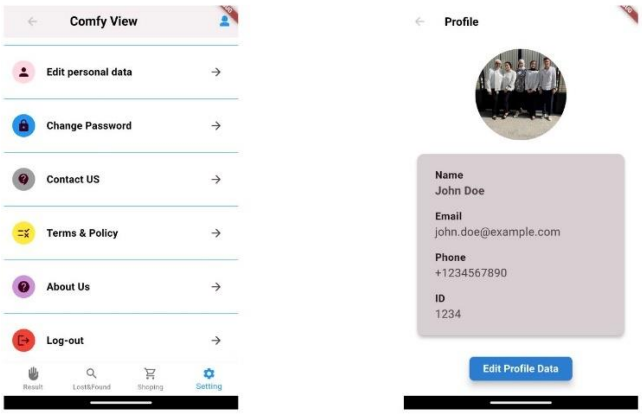


Figure 86 (Mobile application UI 6)

1-Settings Screen:

This is the Settings section of the Comfy View app, where the user can manage various account and app-related settings. The options displayed include:

- Edit personal data
- Change Password
- Contact US
- Terms & Policy
- About Us
- Log-out

2- Profile Screen:

This screen displays the user's profile information. It shows a profile picture of a group of people, the user's name "John Doe", email address "john.doe@example.com", phone number "+12345678901", and user ID "1234". There is also an "Edit Profile Data" button at the bottom.

Overall, these screens allow the user to access and manage their personal information, account settings, and various other options within the Comfy View application. The Settings screen provides access to important account management functions, while the Profile screen displays the user's details and provides the ability to edit them

`

5.6.2) Logic Implantation

Pubspec.yaml:

```yaml
dependencies:
  flutter:
    sdk: flutter


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.2
  camera: ^0.9.4+5
  bloc: ^8.1.2
  flutter_bloc: ^8.1.3
  dio: ^5.4.0
  conditional_builder_null_safety: ^0.0.6
  hexcolor: ^3.0.1
  shared_preferences: ^2.2.2
  webview_flutter: ^4.5.0
  smooth_page_indicator: ^1.1.0
  carousel_slider: ^4.2.1
  image_picker: ^0.8.4+8
  firebase_core: ^2.24.2
  cloud_firestore: ^4.14.0
```

Figure 87 (Code snippets Flutter Pubspec.yaml)

Dependencies

- dependencies: This section lists the package dependencies.
    - flutter: The Flutter SDK, which is a required dependency.
    - cupertino_icons: A package that provides Cupertino-style icons.
    - camera: A package that provides a camera plugin for Flutter.
    - bloc: A package that provides a state management solution using the BLoC architecture.
    - flutter_bloc: A package that provides a set of widgets and utilities for using the BLoC architecture in Flutter apps.
    - dio: A package that provides a powerful and flexible HTTP client for Dart and Flutter.
    - conditional_builder_null_safety: A package that provides a conditional builder widget that supports null safety.
    - hexcolor: A package that provides a simple way to use hex color codes in Flutter.
    - shared_preferences: A package that provides a way to store and retrieve simple key-value pairs in a platform-agnostic way.

`

- webview_flutter: A package that provides a WebView widget for Flutter, allowing apps to display web content.
- smooth_page_indicator: A package that provides a customizable page indicator for paginated views.
- carousel_slider: A package that provides a carousel slider widget for Flutter.
- image_picker: A package that provides a way to pick images from the device's gallery or camera.
- firebase_core: A package that provides the core Firebase SDK for Flutter, allowing apps to use Firebase services.
- cloud_firestore: A package that provides a NoSQL document database for Flutter apps using Cloud Firestore.

Main.dart :

```
await Firebase.initializeApp(
  options: DefaultFirebaseOptions.currentPlatform,
);
```

Figure 88(Code snippets Flutter firebase initialization)

Firebase Initialization: The Firebase.initializeApp function is called to initialize the Firebase app with the default options for the current platform.

```
Bloc.observer = MyBlocObserver();
```

Figure 89 (Code snippets Flutter Bloc Observer)

Bloc Initialization: The Bloc.observer property is set to MyBlocObserver, which is a custom BLoC observer that logs BLoC events.

```
DioHelper.dio;
```

Figure 90 (Code snippets Flutter initialize dio)

Dio Initialization: The DioHelper.dio property is accessed, which initializes the Dio HTTP client.

`

```dart
MultiBlocProvider(
  providers: [
    BlocProvider(
      create: (context) => ComfyviewCubit(),
    ),
    BlocProvider(
      create: (context) => LoginPasswordCubit(),
    ),
    BlocProvider(create: (context) => RegisterCubit()),
    BlocProvider(
      create: (context) => LostAndFoundCubit(),
    ),
  ],
  child: const MyApp(),
),
```

Figure 91 (Code snippets Flutter MultiBlocProvider)

MultiBlocProvider: The MultiBlocProvider widget is used to provide multiple BLoCs to the app. It takes a list of BlocProvider widgets, each of which provides a BLoC to the app.

```dart
ThemeData(
  primaryColor: Colors.blue[600],
  textTheme: const TextTheme(
    bodyLarge: TextStyle(
      fontSize: 18,
      fontWeight: FontWeight.bold,
      color: Colors.black,
    ),
    titleMedium: TextStyle(
      fontSize: 18,
      fontWeight: FontWeight.w400,
      color: Colors.black,
    ),
  ),
  scaffoldBackgroundColor: Colors.lightBlue[50],
  appBarTheme: AppBarTheme(
    // ...
  ),
  bottomNavigationBarTheme: const BottomNavigationBarThemeData(
    // ...
  ),
  colorScheme: ColorScheme.fromSeed(
    seedColor: const Color.fromARGB(99, 0, 0, 0),
  ),
  useMaterial3: true,
),
```

Figure 92 (Code snippets Flutter Theming)

`

Theme: The ThemeData object is used to set up the app's theme, including the primary color, text theme, scaffold background color, app bar theme, bottom navigation bar theme, and color scheme.

```
initialRoute: 'welcomeScreen',
routes: {
  'welcomeScreen': (context) => const OnBoardingScreen(),
  'loginScreenCamera': (context) => const LoginScreen(),
  'layoutScreen': (context) => const GlobalLayout(),
  'loginScreenPass': (context) => LoginScreenPassword(),
  'RegisterScreen': (context) => RegisterScreen(),
  'ResultScreen': (context) => const resultScreen(),
  'profileScreen': (context) => const ProfileScreen(),
  'settinfScreen': (context) => const SettingScreen(),
  'lostandfound': (context) => LostAndFound(),
  'ecommerceScreen': (context) => const EcommerceScreen(),
},
```

Figure 93(Code snippets Flutter Routing)

Routes: The routes property is used to set up the app's routes, which map route names to widgets.

`

Firebase integration:

```dart
class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
      return web;
    }
    switch (defaultTargetPlatform) {
      case TargetPlatform.android:
        return android;
      case TargetPlatform.iOS:
        return ios;
      case TargetPlatform.macOS:
        return macos;
      case TargetPlatform.windows:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for windows - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      case TargetPlatform.linux:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for linux - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      default:
        throw UnsupportedError(
          'DefaultFirebaseOptions are not supported for this platform.',
        );
    }
  }

  static const FirebaseOptions web = FirebaseOptions(
    apiKey: 'AIzaSyAqmXpCxb2-BHnZmEPXGeWA3okT_i1ItSI',
    appId: '1:33103850184:web:5de451f9bd3b9f85c9d3fd',
    messagingSenderId: '33103850184',
    projectId: 'graduation2-fb874',
    authDomain: 'graduation2-fb874.firebaseapp.com',
    databaseURL: 'https://graduation2-fb874-default-rtdb.firebaseio.com',
    storageBucket: 'graduation2-fb874.appspot.com',
    measurementId: 'G-J126KFNYPQ',
  );
```

Figure 94 (Code snippets Flutter Firebase integration)

This is a Dart file that contains the DefaultFirebaseOptions class, which provides default Firebase options for use with your Firebase apps. The class has a static method currentPlatform that returns the appropriate Firebase options based on the current platform (web, Android, iOS, macOS, etc.).

The class also contains several static constants for each platform, which include the API key, app ID, messaging sender ID, project ID, database URL, storage bucket, and measurement ID (for web only).

These options are used to initialize the Firebase app in your Flutter app using the Firebase.initializeApp method.

`

Models Classes:

```
class UserModel {
  var name;
  var email;
  var password;
  var gloveid;

  UserModel({this.email, this.gloveid, this.name, this.password});

  UserModel.fromJson(Map json) {
    name = json['name'];
    email = json['email'];
    password = json['password'];
    gloveid = json['glove_id'];
  }
}
```

Figure 95(Code snippets Flutter Model class user)

Explanation:

- Attributes:
  - name, email, password, gloveid: These are the properties of the UserModel class, representing the user's information.
- Constructor:
  - UserModel({this.email, this.gloveid, this.name, this.password}): Initializes a UserModel instance with optional named parameters.
- fromJson Method:
  - UserModel.fromJson(Map json): This constructor creates a UserModel instance from a JSON map. It extracts values from the map using keys and assigns them to the corresponding properties.

`

```
class ResultSModel {
  String? image;
  Map<String, dynamic>? result;

  ResultSModel({this.image, this.result});

  ResultSModel.fromJson(Map<String, dynamic> json) {
    image = json['image'];
    result = (json['result'] as Map<String, dynamic>).map((key, value) {
      return MapEntry<String, dynamic>(key, value);
    });
  }
}
```

Figure 96(Code snippets Flutter Model class Result)

Explanation:

- Dio Instance:
  - static Dio = Dio(...): Creates a static instance of Dio with predefined base options.
  - BaseOptions(baseUrl: APIPATH, receiveDataWhenStatusError: true): Sets the base URL for the API and configures Dio to receive data even when the status indicates an error.
- postData Method:
  - static Future<Response> postData(...): A static method that sends a POST request to the specified URL with optional query parameters and required data.
  - return dio.post(...): Uses the Dio instance to send the POST request and returns the response.

`

Integration with Api:

```
import 'package:dio/dio.dart';

class DioHelper {
  static Dio dio = Dio(
    BaseOptions(
      baseUrl: APIPATH,
      receiveDataWhenStatusError: true,
    ),
  );

  static Future<Response> postData({
    required String url,
    Map<String, dynamic>? query,
    required Map<String, dynamic> data,
  }) async {
    return dio.post(
      url,
      queryParameters: query,
      data: data,
    );
  }
}
const APIPATH = 'https://graduationproject-backend.onrender.com/api/';
```

Figure 97(Code snippets Flutter Dio helper setup)

Explanation:

- DioHelper: This class provides methods for making HTTP requests using the Dio package. The postData method is used to send POST requests to the specified URL with optional query parameters and data.
- APIPATH: This constant defines the base URL for the API.

`

```
void userLogin({required String email, required String Password}) async {
  emit(LoadingLoginPasswordstate());
  try {
    final response = await DioHelper.postData(
      url: SIGNIN,
      data: {'email': email, 'password': Password},
    );

    final responseData = response.data is String
        ? json.decode(response.data)
        : response.data;

    userModel = UserModel.fromJson(responseData);
    if (response.statusCode == 200) {
      emit(SucessesLoginPasswordstate(userModel));
    } else {
      emit(errorLoginPasswordstate());
    }
  } catch (error) {
    print(error.toString());
  }
}
```

Figure 98(Code snippets Flutter integrates with API userLogin)

Explanation:

- user Login Method: Sends a POST request to the SIGNIN endpoint with email and password.
- Handles Response: Decodes the response and updates the state accordingly.

`

```
void userRegister({
  required String email,
  required String Password,
  required String name,
  required String gloveid,
}) async {
  emit(LoadingRegisterstate());
  try {
    final response = await DioHelper.postData(
      url: SIGNUP,
      data: {
        'name': name,
        'email': email,
        'glove_id': gloveid,
        'password': Password,
      },
    );

    // Parse response data if it's a string
    final responseData = response.data is String
        ? json.decode(response.data) // Decode the JSON string
        : response.data;

    userModel = UserModel.fromJson(responseData); // Pass the parsed data
    if (response.statusCode == 200) {
      emit(SucessesRegisterstate(userModel));
    } else {
      emit(errorRegisterstate());
    }
  } catch (error) {
    print(error.toString());
  }
}
```

Figure 99(Code snippets Flutter integrates with API userRegister)

Explanation:

- UserRegister Method: Sends a POST request to the SIGNUP endpoint with email and password.
- Handles Response: Decodes the response and updates the state accordingly.

`

```
Future<void> _uploadPhoto(File imageFile, BuildContext context) async {
  try {
    Dio dio = Dio();
    FormData formData = FormData.fromMap({
      'photo': await MultipartFile.fromFile(imageFile.path),
    });

    Response response = await dio.post(
      'https://cc64-156-215-253-117.ngrok-free.app/compare_faces',
      data: formData,
    );

    if (response.data['similarities'] != 'None') {
      _showSuccessDialog(context, response.data['similarities']);
    } else {
      _showErrorDialog(context);
    }
  } catch (e) {
    print("Error uploading photo: $e");
  }
}
```

Figure 100(Code snippets Flutter integrates with API face recognition)

Explanation:

- File Upload: Creates a FormData object with the file to upload.
- POST Request: Sends the form data to the specified endpoint.
- Response Handling: Checks the response and shows appropriate dialogs.

Putting It All Together

The integration process involves setting up the Dio helper for making HTTP requests, defining API endpoints, and implementing methods in cubit classes or other parts of the application to handle the requests and responses.

`

```
Future<void> LocationBack({
  required String searchValue,
}) async {
  emit(LoadingStatesLost());
  try {
    final response = await DioHelper.postData(
      url: 'https://graduationproject-backend.onrender.com/api/Search',
      data: {'searchValue': searchValue},
    );
    if (response.data['location'] != null) {
      location = response.data['location'];
      emit(SucssesStatesLost());
    } else {
      emit(FailStatesLost());
    }
  } catch (e) {
    print("Error uploading photo: $e");
    emit(FailStatesLost());
  }
}
```

Figure 101(Code snippets Flutter integrates with API Location)

LocationBack Method: Performs a search by making an API call with searchValue. It manages different states (loading, success, failure) using Cubit to handle the UI's response to the operation.

`

State-management (BLOC - CUBIT):

State management refers to the process of managing the state of an application, which includes data, UI state, and business logic. In Flutter, a popular UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase, effective state management is crucial for building scalable and maintainable applications.

Bloc (Business Logic Component) and Cubit are two popular state management solutions in Flutter.

Bloc (Business Logic Component):

- Bloc is a design pattern that helps manage the state of a Flutter application by separating the business logic from the presentation layer. It typically consists of three main components: events, states, and a bloc class.
- Events: These are user actions or system events that trigger a state change in the application. Examples include button clicks, network requests, or timer expirations.
- States: States represent the different snapshots of the application's UI and data at any given moment. Each state corresponds to a specific UI configuration or data state.
- Bloc class: This is the heart of the Bloc pattern. It contains the business logic that processes events and emits new states accordingly. It listens to events, performs computations, and yields new states in response.
- Bloc helps to separate concerns and make the codebase more organized and easier to maintain. It also facilitates testability by allowing you to test the business logic independently of the UI.

11.Cubit:

- Cubit is a variation of Bloc that simplifies the Bloc pattern by removing the need for separate event and state classes. Instead, it uses a single class to handle both events and states.
- The name "Cubit" stands for "Control Cubit," emphasizing its role in controlling the application's state.
- With Cubit, you define a class that extends the Cubit class and manages both events and states internally. This reduces boilerplate code and makes the state management logic more concise.
- Cubit is particularly suitable for simpler applications or scenarios where the full power of the Bloc pattern is not necessary. It offers a more lightweight approach to

`

state management while still providing the benefits of separation of concerns and testability.

Both Bloc and Cubit are powerful tools for managing the state of Flutter applications, and the choice between them often depends on the complexity of the application and personal preference. They provide structure and maintainability to Flutter projects, making it easier to build and scale applications over time.

Firebase listener and display data:

```
Stream<List<ResultSModel>> _getResultsStream() {
  final CollectionReference resultsCollection =
      FirebaseFirestore.instance.collection('Results');

  return resultsCollection.snapshots().map((snapshot) {
    return snapshot.docs.map((doc) {
      final data = doc.data() as Map<String, dynamic>;
      return ResultSModel(
        image: data['image'],
        result: data['result'] as Map<String, dynamic>?,
      );
    }).toList();
  });
 }
}
```

Figure 102(Code snippets Flutter getData from firebase)

_getResultsStream Method:

- Connects to the 'Results' collection in Firestore and maps the snapshots to ResultSModel objects.
- resultsCollection.snapshots().map((snapshot) { ... }): Converts Firestore snapshots into a list of ResultSModel objects.

_getResultsStream Method: Connects to the 'Results' collection in Firestore and maps the snapshots to ResultSModel objects. resultsCollection.snapshots().map((snapshot) { ... }): Converts Firestore snapshots into a list of ResultSModel objects

`

```
Widget build(BuildContext context) {
  return StreamBuilder<List<ResultSModel>>(
    stream: _getResultsStream(),
    builder: (context, snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
        return const Center(
          child: CircularProgressIndicator(),
        );
      } else if (snapshot.hasError) {
        return Center(
          child: Text('Error: ${snapshot.error}'),
        );
      } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
        return const Center(
          child: Text('No data available.'),
        );
      } else {
        final List<ResultSModel> resultList = snapshot.data!;
        return ListView.separated(
          itemCount: resultList.length,
          itemBuilder: (context, index) {
            final ResultSModel result = resultList[index];
            String imageUrl = result.image != null
                ? 'https://firebasestorage.googleapis.com/v0/b/graduation2-fb874.appspot
                : 'images/faceprint.p
            return InkWell(
```

Figure 103(Code snippets Flutter display data)

Explanation:

- StreamBuilder:
  - StreamBuilder<List<ResultSModel>>: Listens to a stream of ResultSModel objects.
  - stream: _getResultsStream(): Connects to the Firestore stream provided by the _getResultsStream method.
- Builder Method:
  - Handles various states: waiting, error, no data, and displaying the list of results.
  - Displays a loading indicator while waiting for data.
  - Shows an error message if there's an error in the stream.
  - Displays a message if no data is available.
  - Shows a list of results if data is available.

`

Chapter 6 (Conclusion & Future work)

`

6.1) Conclusion

The development of Comfy View, a wearable assistive arm, is aimed at significantly enhancing the mobility, independence, and safety of blind and visually impaired individuals. Through a meticulous planning phase, which included user needs assessment, hardware and software design considerations, and requirements analysis, the project set a solid foundation for its implementation.

By integrating advanced technologies such as AI-powered object detection, face recognition, and haptic feedback mechanisms, Comfy View aims to provide real-time assistance in obstacle detection and social interaction, thereby empowering users in their daily activities.

The system caters to two primary user groups: blind users who directly benefit from wearing the assistive arm, and buyers who purchase the device to support their visually impaired acquaintances. The reasons for acquiring the wearable assistive arm revolve around its ability to increase mobility and promote independence, addressing crucial needs of the target users.

The implementation phase involves the development of various modules, leveraging tools and technologies like Arduino IDE, Firebase, and Anaconda Environment for hardware integration, database management, and artificial intelligence functionalities.

Following implementation, rigorous testing against requirements ensures the reliability, usability, and safety of the product. Documentation plays a vital role in capturing the internal design details for future maintenance and development, while also providing clarity and usability guidelines for end users.

`

6.2) Future Work

While the current iteration of Comfy View represents a significant advancement in assistive technology, there are several avenues for future exploration and improvement. Firstly, continuous refinement of AI algorithms for better object detection and face recognition accuracy can enhance the system's performance in real-world scenarios.

Additionally, exploring additional sensors and feedback mechanisms to provide more comprehensive environmental awareness and intuitive user interaction could further enhance the user experience. Moreover, ongoing user feedback and iterative design improvements are essential to ensure the device's effectiveness and usability in diverse situations.

Furthermore, expanding compatibility with other assistive devices and platforms, as well as addressing evolving ethical and privacy considerations surrounding data collection and usage, will be crucial for maintaining the trust and adoption of the technology among users and stakeholders.

Overall, the journey of Comfy View extends beyond its initial development phase, promising continuous innovation and refinement to meet the evolving needs of the blind and visually impaired community, ultimately striving towards a more inclusive and accessible future.

`

6.3) References

Academind. (2018, November 26). Docker Tutorial for Beginners - A Full DevOps Course on How to Run Applications in Containers [. Retrieved from YouTube: https://www.youtube.com/watch?v=fqMOX6JJhGo

Ageitgey. (n.d.). ageitgey/face_recognition. . Retrieved from GitHub. : https://github.com/ageitgey/face_recognition

AlexeyAB. (2022, December 1). yolov7. Retrieved from github: https://github.com/WongKinYiu/yolov7/tree/main/data

Drip, C. (2021, February 21). Python Tutorial: Decorators! [Video]. . Retrieved from YouTube. : https://www.youtube.com/watch?v=9czGfmporjs

García, D. (n.d.). Medium. Retrieved from https://medium.com/latinxinai/step-on-step-guide-to-deploy-yolo-model-using-fastapi-1a764dbd270d

García, D. (n.d.). medium. Retrieved from Step on Step guide to deploy YOLO model using FastAPI: https://medium.com/latinxinai/step-on-step-guide-to-deploy-yolo-model-using-fastapi-1a764dbd270d

glenn-jocher. (2024, January 6). Ultralytics HUB Inference API. Retrieved from ultralytics: https://docs.ultralytics.com/hub/inference-api/#segment-model-format

InfluxDB. (2024, May 14). ESP32: Getting Started with InfluxDB. Retrieved from randomnerdtutorials: https://randomnerdtutorials.com/esp32-influxdb/

KP. (2018, October 1). Image Matching Algorithm. Retrieved from Kaggle. : https://www.kaggle.com/code/vinitkp/image-matching-algorithm

Mathew, C. (2021, June 7). Face Recognition in Python - A Comprehensive Guide. Retrieved from Medium.: https://basilchackomathew.medium.com/face-recognition-in-python-a-comprehensive-guide-960a48436d0f

PowerBroker2. (2020, december 3). NEO-6M_GPS. Retrieved from arduino: https://www.arduino.cc/reference/en/libraries/neo-6m_gps/

Recognition, F. (n.d.). Face Recognition. (n.d.). face-recognition. Retrieved from Read the Docs: https://face-recognition.readthedocs.io/en/latest/readme.html

`

Schafer, C. (2020, June 11). Python Tutorial: Context Managers - Efficiently Managing Resources [Video]. . Retrieved from YouTube: https://www.youtube.com/watch?v=xoq2cExmaA0

ScrapeHero. (2021, November 4). Exploring Image Similarity Approaches in Python. Retrieved from Medium: https://medium.com/scrapehero/exploring-image-similarity-approaches-in-python-b8ca0a3ed5a3

SHARMA, A. (2022, octobar 10). Easiest Way To Train YOLOv7 On The Custom Dataset in 2024. Retrieved from machinelearningprojects: https://machinelearningprojects.net/train-yolov7-on-the-custom-dataset/

Shendy. (n.d.). Documentation Project: Face Recognition System with OpenCV. . Retrieved from Medium. : https://medium.com/@shendyeff/documentation-project-face-recognition-system-with-opencv-2d18793623d6

Srinivasan. (2021, February 12). GeeksforGeeks. Retrieved from Feature Matching using ORB Algorithm in Python OpenCV. : https://www.geeksforgeeks.org/feature-matching-using-orb-algorithm-in-python-opencv/

stackoverflow. (2023, june 8). stackoverflow. Retrieved from YOLOv8 with FASTAPI: https://stackoverflow.com/questions/76710772/yolov8-with-fastapi-error-could-not-find-a-writer-for-the-specified-extension-in

Step on Step guide to deploy YOLO model using FastAPI. (2023, Septamper 29). Retrieved from medium.: https://medium.com/latinxinai/step-on-step-guide-to-deploy-yolo-model-using-fastapi-1a764dbd270d

Tech2Explore. (2018, November 23). OpenCV Python Tutorial - Find Labeled Objects in Image or Video (YOLOv3). . Retrieved from YouTube.: https://www.youtube.com/watch?v=16s3Pi1InPU

Tim, T. W. (2020, September 2). Machine Learning Tutorial - Python Module 1: Introduction to Machine Learning [Video]. . Retrieved from YouTube: https://www.youtube.com/watch?v=yKtngUPFJbU