SIGN LANGUAGE TUTOR

By

CHRISTOPHER ANOPAISHE MEKI

(H190080Y)

HIT400 Capstone project Submitted in Partial Fulfillment of the

Requirements of the degree of

Bachelor of Technology

In

Software Engineering

In the

School of Information Sciences and Technology

Harare Institute of Technology

Zimbabwe

Supervisor: MISS Y CHIBAYA

May/2023

Certificate of Declaration

This is to certify that work entitled "**Sign Language Tutor**" *is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology (Hons) in Software Engineering ,Harare Institute of Technology .It is further certified that no part of research has been submitted to any university for the award of any other degree .*

(Supervisor)	Signature……..	Date…….
(Mentor)	Signature……..	Date……
(Chairman)	Signature……..	Date……..

HIT 400 /200 Project Documentation Marking Guide

ITEM	TOTAL MARK /%	ACQUIRED/%
PRESENTATION- Format-Times Roman 12 for ordinary text, Main headings Times Roman 14, spacing 1.5. Chapters and sub-chapters, tables and diagrams should be numbered. Document should be in report form. Range of document pages. Between 50 and 100.Work should be clear and neat	**5**	
Pre-Chapter Section Abstract, Preface, Acknowledgements, Dedication & Declaration	**5**	
Chapter One-Introduction Background, Problem Statement, Objectives – smart, clearly measurable from your system. Always start with a TO… Hypothesis, Justification, Proposed Tools Feasibility study: Technical, Economic & Operational Project plan –Time plan, Gantt chart	**10**	
Chapter Two-Literature Review Introduction, Related work & Conclusion	**10**	
Chapter Three –Analysis Information Gathering Tools, Description of system Data analysis –Using UML context diagrams, DFD of existing system Evaluation of Alternatives Systems, Functional Analysis of Proposed System-Functional and Non-functional Requirements, User Case Diagrams	**15**	
Chapter Four –Design Systems Diagrams –Using UML Context diagrams, DFD, Activity diagrams Architectural Design-hardware, networking Database Design –ER diagrams, Normalized Databases Program Design-Class diagrams, Sequence diagrams, Package diagrams, Pseudo code Interface Design-Screenshots of user interface	**20**	
Chapter Five-Implementation & Testing Pseudo code of major modules /Sample of real code can be written here Software Testing-Unit, Module, Integration, System, Database & Acceptance	**20**	
Chapter Six –Conclusions and Recommendations Results and summary, Recommendations & Future Works	**10**	
Bibliography –Proper numbering should be used Appendices –templates of data collection tools, user manual of the working system, sample code, research papers	**5**	
	100	**/100**

Table of Contents

Abstract

The sign language tutor application is a software program designed to teach users sign language using an interactive and engaging platform. The app is designed to be accessible to all individuals who are interested in learning sign language, regardless of their proficiency level. The app includes a range of sign language courses, from beginner to advanced levels, that are presented through interactive quizzes and test for the sign alphabet (ASL) and gifs for common words.

The app's user interface is designed to be intuitive and easy to navigate. Users can choose the lesson they want to take and access them through the app's dashboard. Each lesson is broken down into segments that cover different aspects of sign language alphabet. The lessons are designed to be engaging and interactive, with images that show how to perform different signs and quizzes that test the user's knowledge.

The app also includes a range of features that help users track their progress and stay motivated. These include progress tracking and gamification. Progress tracking enables users to see how far they have progressed in their courses, while gamification incentivizes learning by having a scoreboard with scores show how the user is faring against other users.

KEYWORDS: ASL(American Sign Language), Fingerpose, Handpose, Gesture, Detection, ReactJS, ExpressJS

Preface

In today's world, communication is key, and the ability to communicate with people of all abilities is becoming increasingly important. One such ability is the proficiency in sign language, which is a visual language that is used by millions of people across the globe. However, learning sign language can be challenging, especially if you don't have access to a qualified teacher or a supportive community.

The Sign Language Tutor Application is designed to address this need by providing an accessible and engaging platform for learning sign language. The app offers a comprehensive curriculum, ranging from beginner to advanced levels, to cater to all learners' needs. The interactive quizzes and test for the sign alphabet (ASL) and gifs for common word in the app help users to learn sign language effectively, at their own pace, and from the comfort of their own home.

This project has been developed with the aim of making sign language learning accessible and enjoyable to everyone, irrespective of their geographical location or language proficiency. We have put in extensive research and development to create an innovative and user-friendly app that empowers learners to take control of their learning journey.

I hope that this application will help bridge the gap between hearing and non-hearing communities and contribute to creating a more inclusive and empathetic society. This project represents a collaborative effort between experts in the fields of linguistics, technology, and education, and we are excited to share it with you.

Acknowledgements

Firstly, I would like to thank **God** for taking me this far. I am grateful for the gift of life, good health and all the resources that I used to complete this project. Then, I specifically want to acknowledge **Miss Chibaya**, my supervisor for the tremendous support, useful comments, remarks, feedback and engagement through the learning process of this capstone research project.

I would like to express my sincere gratitude to **Mr. Makondo, Mr. Mukosera, Miss Amos, Mr. Manjoro, Miss Zindove and Mr. Chiworera** and for their unending support. It is the insight they gave me that added value to this project.

I would also want to thank the university **Harare Institute of Technology** for providing me with adequate resources and knowledge required for the development of my project.

I am indebted to my family for their unending support throughout this period. They gave me encouragement, love, guidance and support. May God bless them and protect them.

Lastly, I also extend my gratitude to my fellow classmates and friends who were there for me during this journey.

Dedication

This project is dedicated to all those who are passionate about creating a more inclusive world. It is for those who believe in the power of technology to bridge gaps and bring people together. We dedicate this project to the individuals and communities who have been historically marginalized due to their hearing impairment, and for whom sign language is an essential mode of communication.

I also dedicate this project to the educators and experts who have devoted their lives to promoting sign language as a valuable and respected language. It is through their tireless efforts that sign language is gaining recognition as a means of communication that is as valid as any spoken language.

Finally, I dedicate this project to all our users, who have trusted us with their learning journey. We hope that this application will help you acquire a valuable skill and enable you to communicate with people in new and meaningful ways.

Declaration

All sentences or passages quoted in this dissertation from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). I understand that failure to do this amount to plagiarism and will be considered grounds for failure in this dissertation and the degree examination as a whole.

ABBREVIATIONS

Abbreviation	Meaning
DFD	Data Flow Diagram
OPENCV	Open Computer Vision
UX	User Interface
HDD	Hard Disk Drive
RAM	Random Access Memory
OS	Operating System

CHAPTER 1 : Introduction

1.0 Introduction

Communication is a fundamental aspect of human interaction, enabling the exchange of ideas, emotions, and information. However, individuals with hearing disabilities face significant challenges in effectively communicating with others. Sign language serves as a vital means of communication for the deaf and hard-of-hearing community, but learning and understanding sign language can be a complex and time-consuming process.

To address this issue, we propose the development of a Sign Language Tutor application. This application aims to leverage cutting-edge technologies, such as the fingerpose library and TensorFlow handpose model, to provide an interactive and effective platform for learning and practicing sign language gestures.

The Sign Language Tutor application will revolutionize the way people learn sign language by offering real-time recognition and interpretation of sign language gestures. It will empower users to communicate more confidently and efficiently, promoting inclusivity and breaking down communication barriers between individuals with hearing impairments and the broader community.

1.1 Background

Sign language is a visual language that utilizes hand movements, facial expressions, and body language to convey meaning and communicate. It serves as the primary mode of communication for individuals who are deaf or hard of hearing. However, learning and understanding sign language can be a challenging and time-consuming process.

Traditionally, sign language learning has been facilitated through in-person classes, books, and instructional videos. While these resources are valuable, they often lack interactivity and real-time feedback, making it difficult for learners to practice and improve their skills effectively. Additionally, access to qualified sign language instructors may be limited in certain regions, further hindering the learning process.

Advancements in technology, particularly in the fields of machine learning, computer vision, and mobile applications, present an opportunity to address these challenges. The integration of gesture recognition and interpretation algorithms with interactive learning platforms can revolutionize sign language education and improve accessibility for individuals with hearing disabilities.

In recent years, machine learning models such as the fingerpose library and TensorFlow handpose model have emerged as powerful tools for hand gesture recognition. These models leverage deep learning techniques to accurately detect and interpret hand movements, including sign language gestures. Their integration into a user-friendly application can facilitate real-time recognition and interpretation, providing learners with immediate feedback and guidance.

The Sign Language Tutor application aims to build upon these technological advancements to create an innovative learning tool for sign language. By harnessing the power of machine learning and computer vision, the application will provide an interactive platform that recognizes and interprets sign language gestures in real-time. Users will be able to practice their signing skills, receive feedback on their performance, track their progress, and access a comprehensive database of sign language gestures.

This background highlights the pressing need for a modern and accessible approach to sign language learning. The Sign Language Tutor application project seeks to address these needs by leveraging state-of-the-art technologies and methodologies, ensuring that individuals with hearing disabilities can effectively learn, practice, and communicate using sign language.

1.2 Problem statement

Individuals with hearing disabilities face significant challenges in effective communication due to the lack of accessibility and interactive learning resources for sign language. Traditional methods of learning sign language, such as in-person classes or instructional videos, often lack interactivity, real-time feedback, and comprehensive learning materials. This results in limited opportunities for practice, slow progress, and difficulties in acquiring fluency in sign language thus lack of portable, on-the-go and cheap methods to facilitate sign language learning hence difficulty in communication amongst the deaf, hearing impaired and normal hearing people. There is need to develop a web based e-tutor (Sign Language Tutor) to facilitate interactive sign language learning.

1.3 Objectives

- To identify sign letters accurately and provide sign language learning with the use of virtual sign interpreter using fingerpose library and TensorFlow handpose model.
- To provide a mobile (on-the-go) method of sign language learning through the development of a web application
- To provide unique education for people for easy, handy and very convenient learning on different sign languages by implementing interactive quiz assessments to complement the learning process

1.4 Hypothesis

Null Hypothesis (H0): The Sign Language Tutor application will not significantly improve the sign language learning outcomes and user experience compared to traditional methods.

Alternative Hypothesis (HA): The Sign Language Tutor application will significantly improve the sign language learning outcomes and user experience compared to traditional methods.

This hypothesis is based on the assumption that by leveraging modern technologies, such as the fingerpose library and TensorFlow handpose model, the Sign Language Tutor application can provide real-time recognition and interpretation of sign language gestures, personalized feedback, progress tracking, and a comprehensive database of gestures. These features have the potential to enhance the learning experience, improve user engagement and motivation, and ultimately lead to better sign language learning outcomes.

The project aims to conduct a comprehensive evaluation and user study to test this hypothesis. By comparing the performance, user satisfaction, and learning outcomes of participants using the Sign Language Tutor application with those using traditional methods, the project aims to provide empirical evidence to either accept or reject the null hypothesis. If the alternative hypothesis is supported, it will validate the effectiveness and potential of the Sign Language Tutor application as an innovative and valuable tool for sign language learning.

1.5 Justification of study

The Sign Language Tutor application project is justified by several compelling reasons that highlight its significance and potential impact:

1. Addressing Communication Barriers: The primary justification for this study is to address the communication barriers faced by individuals with hearing disabilities. Sign language serves as their primary mode of communication, and by providing an interactive and effective learning tool, the Sign Language Tutor application can empower them to communicate more confidently and bridge the communication gap between them and the hearing community.
2. Enhanced Learning Experience: Traditional methods of learning sign language often lack interactivity and real-time feedback, making it challenging for learners to practice and improve their skills. The Sign Language Tutor application aims to provide an engaging and interactive learning experience, offering real-time gesture recognition, personalized feedback, and progress tracking. This enhanced learning experience can significantly improve the effectiveness and efficiency of sign language acquisition.
3. Accessibility and Inclusivity: Accessibility is a critical aspect of the Sign Language Tutor application. By leveraging modern technologies, such as mobile applications and computer

vision, the application can be accessed by individuals with hearing disabilities anytime and anywhere. This promotes inclusivity and eliminates geographical barriers to learning sign language, ensuring that individuals in remote areas or with limited access to qualified instructors can still engage in effective learning.

4. Technological Advancements: The project is justified by the rapid advancements in machine learning, computer vision, and gesture recognition technologies. The integration of the fingerpose library and TensorFlow handpose model into the Sign Language Tutor application showcases the potential of these advancements to revolutionize sign language learning. By leveraging state-of-the-art tools, the application can accurately recognize and interpret sign language gestures, providing users with an immersive and accurate learning experience.
5. Educational Impact: The Sign Language Tutor application has the potential to make a significant educational impact by providing an accessible and comprehensive platform for learning sign language. It can be utilized by educational institutions, sign language instructors, and individuals seeking to learn or improve their signing skills. By promoting effective communication and inclusivity, the application contributes to creating a more inclusive society that values and respects the needs of individuals with hearing disabilities.

In summary, the Sign Language Tutor application project is justified by the pressing need to address communication barriers, enhance the learning experience, promote accessibility and inclusivity, leverage technological advancements, and make a meaningful educational impact. By developing this innovative application, we aim to empower individuals with hearing disabilities, facilitate effective communication, and foster a more inclusive society that values and embraces sign language as a vital means of expression.

1.6 Proposed tools

- ReactJS
- ExpressJS
- Tensor-flow fingerpose library
- Tensor-flow handpose model
- Postgres Database
- Web Cam
- Webstorm IDE
- Google Chrome browser

1.7 Feasibility study

1.7.1 Technical feasibility

Technical feasibility aims at analyzing the Sign Language Tutor Web Application from a technical perspective, that is whether the required technology is available or not and also the availability of the

required resources. Technical feasibility of the proposed system may be done by just looking at the software and hardware available and analysing them without necessarily building the system.

Requirements: Before we begin development, we need to define the requirements for the application. This includes the features and functionality that will be included in the application, such as image and gif tutorials, quizzes, interactive exercises. We also need to determine the technical requirements for the application, such as the required programming languages, development tools, and hardware/software compatibility.

Programming Languages and Tools: The programming languages and tools used to build a sign language tutor application will depend on the platform(s) the application is being built for. Some commonly used languages and tools for building applications include:

- Web: JavaScript, HTML/CSS, React, Angular, or Vue.js,Webstorm

Server and Hosting Requirements: The sign language tutor application may require a server to host content, manage user accounts, and deliver the live tutoring sessions. The server may also require a database to store user information and progress. The hosting provider should be reliable, secure, and scalable to handle potential increases in traffic.

Hardware and Software Compatibility: The sign language tutor application should be compatible with a wide range of devices, including smartphones, tablets, and desktop computers. We will need to ensure that the application works well on different screen sizes and resolutions, as well as different operating systems, such as iOS, Android, and Windows.

Integration with Third-Party Services: We may need to integrate with third-party services, such as payment providers, customer support tools, and analytics tools. We will need to ensure that the integration is seamless and secure.

Conclusion: Based on the technical feasibility study, a sign language tutor application can be developed using a variety of programming languages and tools, hosted on a reliable server, and made compatible with a wide range of hardware and software. Integration with third-party services may be required for payment processing and analytics. Overall, the technical feasibility for developing a sign language tutor application is high.

1.7.2 Economic feasibility

Here, the main concern is determining whether the system's development costs won't exceed the budget that has been set forth. Cost-benefit analysis demonstrated that the application's development could proceed. The majority of the development software I'll be utilizing is open source, so it's free to use. The project is financially feasible because the necessary processing power hardware is currently available and the associated costs have been met.

Development Costs: Developing a sign language tutor application can be expensive, but the total cost will depend on the complexity of the application, the number of features, and the level of customization.

We will need to budget for development costs such as hiring developers, designers, and testers. We may also need to purchase licenses for software and services required for development.

Marketing Costs: To reach potential users, we will need to allocate funds for marketing the application. Marketing costs may include advertising, social media, and public relations. The amount of money allocated to marketing will depend on the target audience and the competition.

Revenue Model: The revenue model for a sign language tutor application can be a subscription-based service or a one-time purchase. We can offer premium content, such as live tutoring sessions, for a fee. The price of the application will depend on the competition and the target audience.

Profitability: Profitability will depend on the revenue generated versus the cost of development and marketing. We will need to conduct regular cost-benefit analysis to ensure profitability is maintained.

Market Demand: The market demand for sign language education is growing, offering a large potential user base for the application. According to the World Federation of the Deaf, there are approximately 72 million deaf people globally, and sign language is their primary means of communication. Additionally, sign language is recognized as a valuable skill for individuals in various professions, including healthcare, education, and customer service.

Based on the results of the economic feasibility study, developing a sign language tutor application can be a profitable business opportunity. We will need to budget for development and marketing costs and determine a competitive revenue model to ensure profitability.

1.7.3 Operational feasibility

The degree to which a proposed system addresses issues, seizes opportunities discovered during scope definition, and satisfies requirements found during the requirements analysis stage of system development is referred to as operational feasibility. It provides an answer to the questions "will the system work?," "will the system fit into the existing business structures?," and "how usable will it be after it is implemented?"

Staffing: Developing and running a sign language tutor application requires a team with specific skills and expertise. We will need to ensure we have the necessary staff members, including developers, designers, QA testers, and customer support representatives.

Content Creation: Developing quality content for the sign language tutor application is critical for user engagement and retention. We will need to ensure we have a team of experienced sign language instructors and content creators who can develop engaging and effective content.

Technical Infrastructure: The technical infrastructure is essential for the smooth operation of the application. We will need to ensure that we have the necessary hardware, software, and systems in place to support the application, such as servers, databases, and backup systems.

User Interface and User Experience: The user interface and user experience are critical to the application's success. We will need to ensure that the application is easy to navigate and user-friendly. This includes creating an intuitive interface, clear instructions, and a consistent design across all platforms.

Testing and Quality Assurance: Testing and quality assurance are essential to ensuring the application's reliability, stability, and security. We will need to test the application under different conditions and ensure that it meets all required standards of quality and usability.

Customer Support: Providing customer support and responding to user feedback is key to maintaining user satisfaction and engagement. We will need to ensure we have a customer support team in place to answer user inquiries and resolve any issues that may arise.

Based on the results of the operational feasibility study, developing a sign language tutor application is operationally feasible but requires careful planning and resources. We will need to ensure that we have the necessary staff, technical infrastructure, and testing processes in place to ensure a smooth and effective user experience. Providing quality content and responsive customer support is also critical to maintain user engagement and satisfaction.

1.7.3 Hardware requirements

Table 1: Hardware Requirements

DESCRIPTION	QUANTITY	AVAILABILITY
Laptop/desktop	1	yes
4Gig RAM and Above	1	yes
Core i3 Processor or better	1	Yes
HDD 20 GB Hard Disk Space and Above	1	yes

As shown on the table above the hardware required to complete this project is readily available, therefore making the proposed system technically feasible.

1.7.4 Software requirements

Table 2: Software Requirements

DESCRIPTION	QUANTITY	AVAILABILITY
WINDOWS OS (7/8/10/11)	1	yes

Webstorm IDE	1	yes
Tensorflow library	1	Yes
Debugging tools	1	yes

1.8 Project Plan

A project plan is a formal document designed to guide the control and execution of a project. A project plan is the key to a successful project and is the most important document that needs to be created when starting any business project.

Table 3: Project Plan

PHASE	DURATION(days)	STARTING DATE	ENDING DATE
Requirements analysis and definition	30 days	1 September 2022	30 September 2022
System design and software design	120 days	1 October 2022	05 Febrary 2023
Implementation and unit testing	20 days	06 January 2023	26 February 2023
Integration and system testing	30 days	27 February 2023	27 March 2023

1.8.0 Gantt chart

Gantt charts convey this information visually. They outline all of the tasks involved in a project, and their order, shown against a timescale. This gives you an instant overview of a project, its associated tasks, and when these need to be finished.

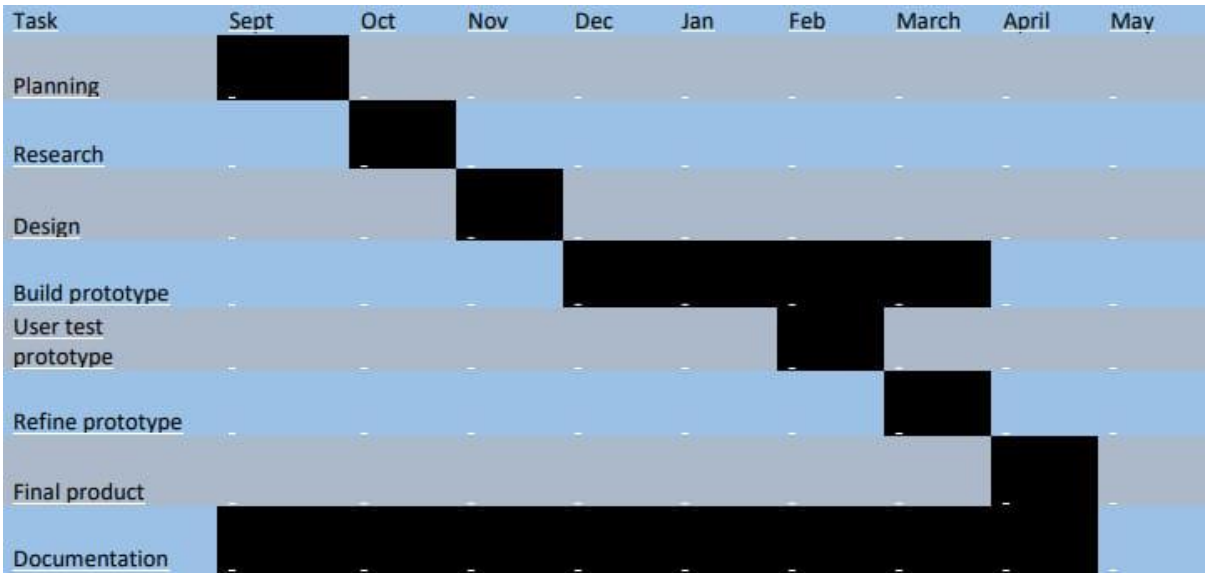


Fig 1: Gantt Chart

CHAPTER 2:Literature Review

2.0 Introduction

The literature review serves as a critical component of the research process, providing a comprehensive overview and analysis of existing scholarly works, research studies, and relevant publications related to the Sign Language Tutor application. It offers a foundation for understanding the current state of knowledge in the field and identifies gaps, trends, and key findings that contribute to the justification and development of the proposed project.

This section aims to explore and synthesize a wide range of literature sources, including academic journals, conference papers, books, and reputable online resources, that delve into various aspects of sign language learning, technology integration, gesture recognition, and related fields. By examining the existing body of knowledge, the literature review seeks to establish a solid theoretical and conceptual framework for the Sign Language Tutor application project, ensuring that it aligns with existing research and addresses relevant challenges and opportunities.

The literature review will be organized thematically, with each section focusing on a specific topic or subdomain related to the project. These topics may include sign language education methods, technology-assisted learning, gesture recognition algorithms, user experience design, accessibility considerations, and the impact of technology on sign language communication. Through a critical analysis of the literature, this review aims to identify key insights, methodologies, and best practices that can inform the development and implementation of the Sign Language Tutor application.

By examining the existing literature, this review will contribute to the understanding of the current landscape of sign language education and technology, highlight gaps and limitations in the existing approaches, and provide valuable insights for the Sign Language Tutor application project. Ultimately, it will serve as a foundation for the project's research methodology, design decisions, and innovation in the field of sign language learning and communication.

2.1 Synthesis of Literature

Sign language is a visual language, and as such, effective teaching requires a visual medium. Thus, the use of mobile applications for teaching sign language has become increasingly common. Below are some key findings from recent research on the effectiveness of sign language tutor applications:

1. **Immediacy and consistency.** One of the main advantages of using mobile applications for sign language learning is the immediacy and consistency they provide. Users can access the application at any time, and the content is always available and presented in a consistent manner. This can lead to improved learning outcomes (Cappelletti & Sartori, 2019).
2. **Personalization.** The ability to personalize learning is also a key advantage of mobile applications. Users can choose their own pace and level of difficulty, and tailor the application to their specific learning needs (Nieminen, Mikkonen, & Tornberg, 2018).

3. **Interactive features.** Sign language tutor applications that incorporate interactive features, such as quizzes, games, and videos, can be more engaging for users and can lead to improved learning outcomes (Zou et al., 2019).
4. **Accessibility.** Mobile applications are highly accessible, making them an ideal tool for teaching sign language to individuals who may not be able to attend in-person classes (e.g. due to a disability or geographical location) (Cappelletti & Sartori, 2019).

References:

Cappelletti, F. M., & Sartori, R. (2019). Mobile applications for sign language teaching and learning: A review of the literature. *Computer Assisted Language Learning*, 32(7), 705-728.

Nieminen, M., Mikkonen, I., & Tornberg, V. (2018). Development of digital teaching materials for sign language. In *2018 International Conference on Information and Communication Technology and Education* (pp. 5-9).

Zou, M., Zhang, M., Wu, Y., Wu, Q., & Liu, C. (2019). The design and evaluation of a mobile application-based sign language education system. *International Journal of Emerging Technologies in Learning*, 14(10), 189-201.

2.2 Related Work

2.2.1 Mobile Sign App

University of Bristol University of Bristol, (2012)launched an innovative application that can help people to communicate in sign language through a searchable database named Mobile Sign. The tool includes the largest free sign language lexicon on the major application stores. Mobile Sign application automatically shows lists of possible words for the user to choose from making it more convenient to the user by the predictive word search, once selected, a video of a person signing the selected word is shown on screen. In addition, it is also available on platforms such as Apple and Android. The application is available to view online or on store on any platform being used. More so, it is ideal since it allow the user to keep a list of their recently viewed signs for repeat access later. However, it is said to contain information which is not correct which need to use your own judgement or outside sources to check.

2.2.2 Fingerspelling (ASL)

Fingerspelling in American Sign Language(Wager, 2012) is a system consisting of 26 onehanded signs representing the letters of the English alphabet and are formed sequentially to spell out words borrowed from oral languages or letter sequences. This application is a practice tool designed to help improve the ability to read fingerspelling. Users can choose the word length (2-any) and speed (slow to fast) of the

fingerspelling, record their answer and keep score. An expert mode is available as ability increases. However, it is said to take more time for reviews, making the user to wait for a long period to get the next review. Also, the reviews are not understandable by the users, making them to go and search on google. In addition, there is no application update done constantly resulting in it taking time to respond

2.2.3 Learn to Sign (Zambian Sign Language)

"Communication is key", Sub Sahara of Africa (Salomons, no date)came up with an idea to develop a sign learning, android based application (mobile based), Learn to Sign (Zambian Sign Language). It is an excellent app as they noticed communication barrier between the deaf and hearing individuals, caused by cultural difference regarding deafness. The application has two main features, the game and dictionary. In addition of a dictionary element to the application, it makes the user to go back to the words they encountered or will encounter during the game, giving them time to go over them in their own time, making it very useful. The dictionary presents a list of different categories on which a user could pick from, which then presents with a list of words. The other feature, the game part, it makes the user eager to learn more since it you will be like playing a game. The user would choose a category and get questions that ask them to look at the given sign, word or video and find the corresponding sign, word or video. As the user progresses through the game, different levels would be unlocked with a higher difficulty making it an excellent app to use. However, some users feel that it contained inadequate information as some of the captions for the signs are not sufficient enough for them to understand the sign. Application instructions are said not to be clear leading to the user using the application ineffectively. There is no feedback option for the users telling them how good or bad they did, which made the game lose value and did not create an incentive to want to play.

2.2.4 Usign Application (Uganda)

The (Uganda National association of the Deaf, 2017)Uganda National association of the Deaf (UNAD) launched a mobile application to ease communication for people who are deaf or hard of hearing. It is said to address most of the challenges facing those with hearing disabilities because it is user-friendly when fully adopted. The application consists of a vast category giving a wide range to learn which includes Numbers, The Alphabet, and Days of the Week, Months of the Year, Time Signs and Fruits. However, the major issue is that Uganda Sign Language, like any developing language, has a limited vocabulary, which impedes the process of creating a broader lexicon and making it utilizable by both the Deaf community and the population at large. Videos are said to be speedy when playing in the application resulting in more users requesting sign pictures for beginners.

2.2.5 GSL Ghanaian Sign Language (Ghana)

The GSL application(Leiden University, no date) is an initiative of the hands, labels for sign languages and deaf studies in Ghana. The application consists of 300 signs, time, money, fingerspelling alphabets and also general numbers. In addition, it also translates English words into GSL Ghanaian Sign Language signs, from A-Z. It is also an excellent application as it does, not require internet to use-that is it is completely offline.

2.2.6 Purple Sign

This is a local software being developed which contains mobile video dictionary making it easier to use due to the availability of illustration in form of videos. In addition, you can learn at your own pace, giving room for interaction with other activities, without being left out. However, the application is still under development.

2.3 Conceptual Solution

The literature review provides valuable insights for the development of the Sign Language Tutor application. The conceptual solutions include integrating gesture recognition algorithms for accurate sign language interpretation, incorporating gamification and interactive learning elements to enhance user engagement, personalizing the learning experience with progress tracking, ensuring accessibility and inclusivity, and curating a comprehensive gesture database. These solutions inform the design and functionality of the application, addressing gaps in sign language learning and technology integration.

2.4Conclusion

In conclusion, applications designed for sign language teaching and learning offer several advantages over traditional in-person classes, including increased accessibility, personalization, and interactivity. The literature suggests that the use of sign language tutor applications can lead to improved learning outcomes due to their immediacy, consistency, and ability to cater to individual learning needs.We need to have a wide coverage of content to enable users to grasp a lot thereby increasing learning scope which includes learning and assessment module. The learning module includes numbers, alphabet and word construction. More so, assessment module comprises of numbers and alphabet quiz. However, more research is needed to assess the long-term effectiveness and scalability of these applications. Overall, sign language tutor applications have great potential for improving access to sign language education and promoting greater inclusivity and understanding of the deaf community.

CHAPTER 3:Analysis

3.0 Information Gathering Tools

Information was gathered and analysed through various methodologies so as to accurately and comprehensively specify the system. This activity was done so as to ensure that each activity to be done is done completely and successfully so as to meet the expectation of the users.

3.0.1 Surveys

Surveys are a common tool for gathering information from potential users of the application. Surveys can be designed to collect information on user preferences, needs, and feedback on the application's features and functionality.

Advantages: Surveys allow for the collection of a large amount of data quickly and efficiently. They can be distributed widely, and allow for the collection of both quantitative and qualitative data, making it possible to gather a comprehensive understanding of user needs.

Disadvantages: Surveys can suffer from response bias or self-selection bias, meaning that those who choose to participate may not be representative of the actual user population. In addition, surveys may struggle to capture the full range of user experiences, particularly more nuanced or complex information

3.0.2 Interviews

This data collection method was used by the developer to get information on the challenges being faced by customers as they travel day in day out to and from their work or to a destination of their own choice. I visited bus terminals in Chitungwiza and engaged travellers to get to know the loopholes of the current system and their fears of the proposed system. We used their information to truly understand the travellers' opinions and their point of view.

Advantages of interviews

- Interviews allows clarity of thoughts as the interviewer can ask further questions on anything that seems unclear
- Interviews are interactive in nature therefore they save time as compared to other methods
- Interviews allows the evaluation of facial expression which may strengthen one's facts
- There was a direct communication with individuals who will use the system hence their recommendations and needs were collected first hand

Disadvantages of Interviews

- The interviewer maybe biased and may interpret the information incorrectly in favour of his preconceived perceptions
- Information given by some interviewees may have been biased in order to protect themselves
- Some interviews failed to come through

3.0.3 Questionaries

Questionnaires were helpful in instances where the people who were supposed to participate in an interview were not available or occupied at the time. Questionnaires were mainly distributed to the people commutes daily to work. These questionnaires were filled without any form of supervision and anonymity was ensured however the respondents would sometimes just fill in the answers without understanding the question.

Advantages of Questionnaires

- Economy of time. Several subjects are addressed simultaneously
- Economy of financial resource as postage is cheaper than travelling to subjects
- Analysis of data from closed-ended is fairly easy
- Data gathering is not influenced by personal attributes
- Respondents are not usually stampeded in providing responses
- Respondents are guaranteed anonymity and are freer to give information without fear of reprisals

Disadvantages of Questionnaires

- They provide no room to clarify ambiguities and rephrase questions. This may result in nil returns which present analysis problems
- They may suffer from a delayed return rate
- They may suffer from a low rate of return
- Their construction is time consuming
- They provide no room to verify authenticity of information given

To carter for the disadvantages of each technique I have used the concept of triangulation which is to use both techniques so as they counter each other's disadvantages.

3.1 Description of System

The sign language tutor application is a system designed to facilitate the learning of sign language using an interactive and engaging platform. The system consists of a web application that offers a range of lessons and tests/quizzies in sign language, catering to various proficiency levels, from beginner to advanced.

The system's user interface is intuitive and easy to navigate, enabling users to access lessons and tests/quizzes easily. The courses are presented through sign image recognition tutorials/lessons, interactive quizzes, and gifs for common words, making the learning process engaging and effective. The lessons show how to perform different signs, and the quizzes test the user's knowledge, providing feedback through points scored.

The system also includes features that help users track their progress and stay motivated. These include progress tracking and gamification. Progress tracking enables users to see how far they have progressed in their courses, while gamification incentivizes learning while gamification incentivizes learning by having a scoreboard with scores which show how the user is faring against other users and also new lessons and test are unlocked after passing previous lessons.

The system's comprehensive curriculum covers various aspects of sign language, including finger spelling and common phrases. The courses are designed to be accessible to all learners, regardless of their language proficiency, age, or location.

Overall, the sign language tutor application system is designed to make sign language learning accessible and enjoyable to everyone, empowering learners to take control of their learning journey and promote inclusivity and empathy.

3.2 Data Analysis

Understanding system and user requirements is very important when it comes to developing Artificial Intelligence based systems. This chapters seeks to explain the existing system used for sign language learning. A feasibility study is also carried out to see the constraints against creating the new system. It will help determine the success and minimize the risks related to the project. Lastly the requirements analysis will be carried out which encompasses those tasks that go into determining the needs or conditions to meet for our system, taking account of the possibly conflicting requirements of the various stakeholders.

3.2.1 Current System and its weaknesses

The existing system for learning sign language lacked several important features that are provided by the Sign Language Tutor application. The previous system was limited to only providing static images and

descriptions of sign language gestures, which made it difficult for users to fully understand and practice the gestures. Additionally, the existing system did not provide any real-time feedback to users on their performance or progress, which made it difficult for them to improve their skills.

The Sign Language Tutor application, on the other hand, provides users with a more interactive and engaging learning experience. It allows users to capture images of their sign language gestures in real-time using a camera, and provides immediate feedback on their performance through a scoring system. The application also uses the TensorFlow Handpose model to recognize and interpret gestures accurately, and provides users with detailed information and videos on how to perform the gestures correctly.

Overall, the Sign Language Tutor application provides a more comprehensive and effective way for users to learn and improve their sign language skills compared to the previous system. With its advanced features and user-friendly interface, the application has the potential to make a significant impact in helping people learn and communicate through sign language.

3.2.2 DFD Level 0(Existing System)

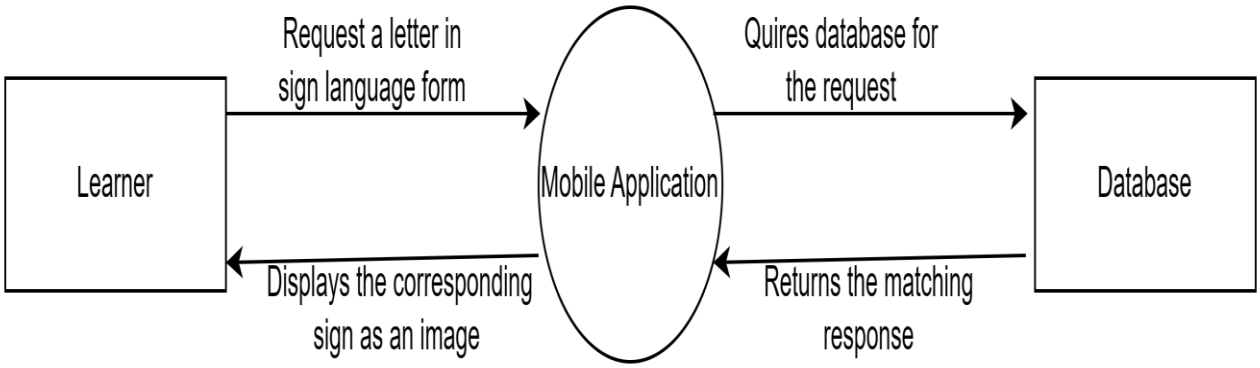


Fig 1: DFD Level 0

3.2.3 Process Flow Diagram (Existing System)

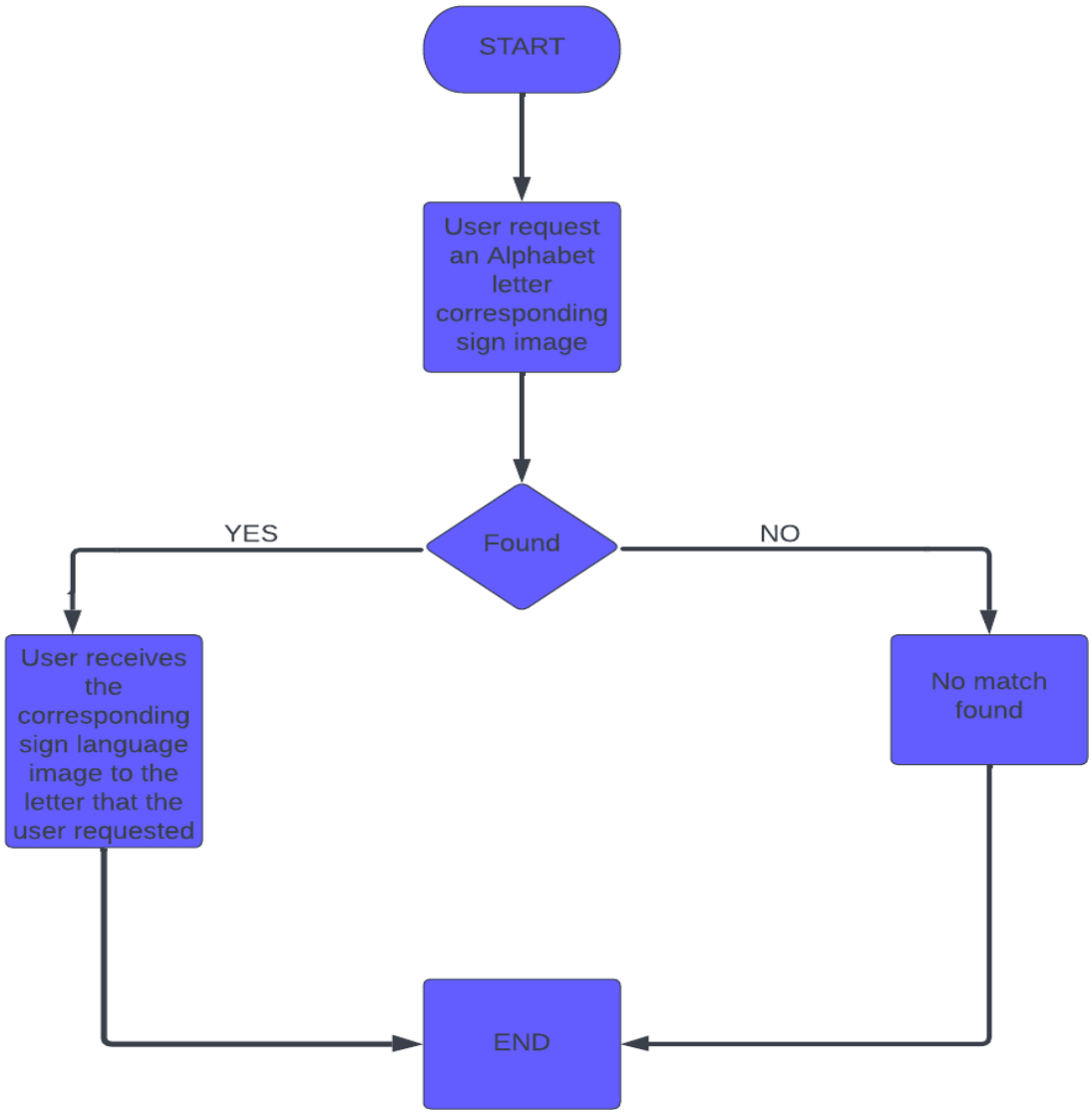


Fig 2: Process Flow Diagram

3.2.4 Level 1 DFD(Existing system)

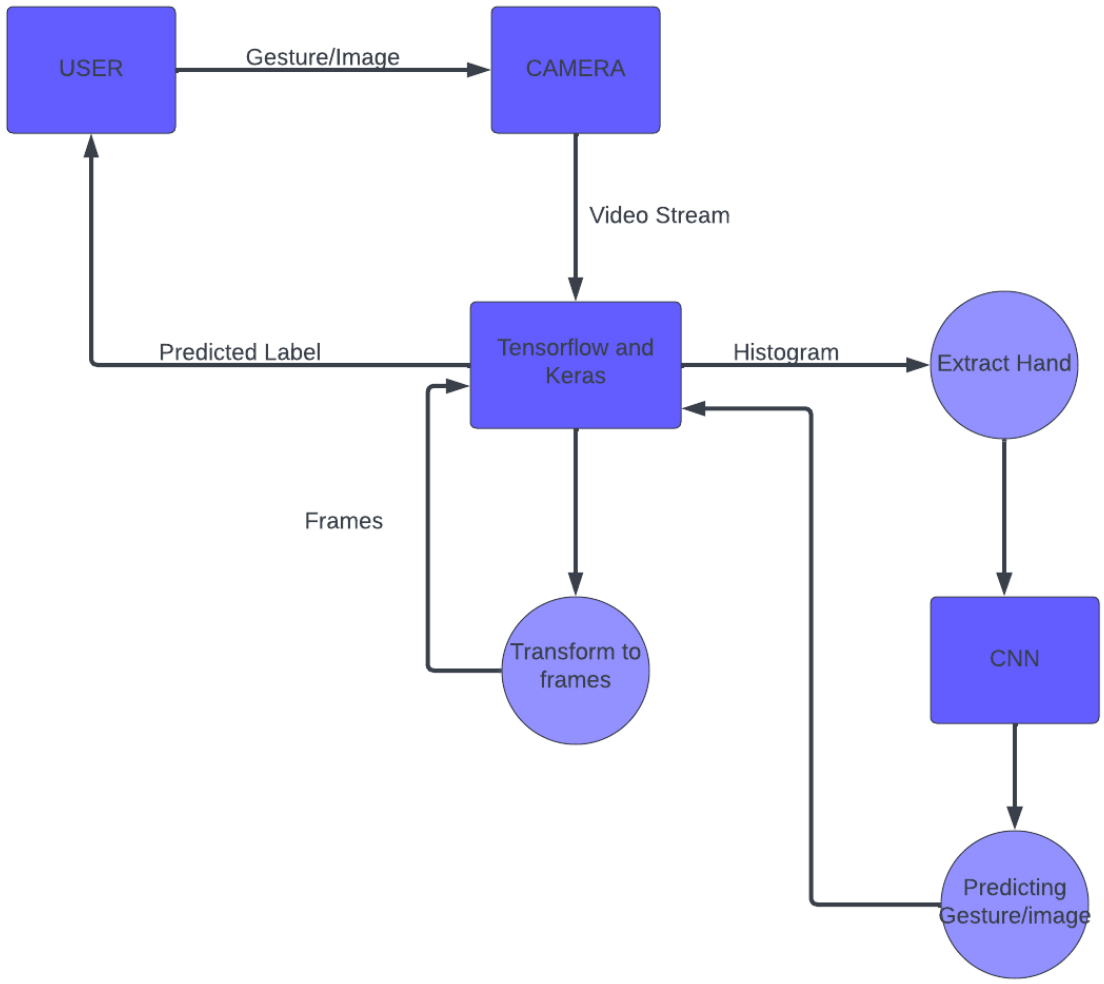


Fig 3: Level 2 DFD

3.2.5 Use-Case Diagram(Existing System)

Fig 4: UseCase Diagram

3.3 Evaluation of Alternatives Systems

1. Signing Savvy: This is an online sign language dictionary and learning resource that provides video tutorials, quizzes, and a variety of sign language courses. It has a large user base and provides a comprehensive range of sign language resources. However, it lacks features such as progress tracking.
2. ASL Pro: This is another online sign language learning resource that provides video tutorials, quizzes, and a sign language dictionary. It has a user-friendly interface and provides detailed explanations of signs. However, it lacks some advanced features such as progress tracking.
3. Gallaudet University's ASL Connect: This is an online learning resource created by Gallaudet University, the world's only university for the deaf and hard of hearing. It provides a range of sign language courses, video tutorials, and interactive quizzes. It is designed specifically for sign language learners and provides comprehensive resources. However, it may be expensive for some users, and its courses may be too advanced for beginners.
4. Lingvano: This is an AI-powered language learning platform that provides personalized language courses and feedback. It provides video tutorials and interactive exercises to improve sign language proficiency. It also provides personalized feedback and progress tracking. However, it may not have as many resources as some other alternative systems.

After evaluating these alternative systems, it is clear that each system has its advantages and disadvantages. However, for the Sign Language Tutor Application system, it is essential to provide a comprehensive range of sign language resources, personalization, progress tracking, and a user-friendly interface. Based on these requirements, Lingvano may be the most suitable alternative system due to its personalized feedback and progress tracking features.

3.4 Functional Analysis of Proposed System

This chapter seeks to define user expectations for the project being built and the system requirements. This part is very crucial as it will determine the relevance and success of the project

3.4.1 Functional Requirements

The proposed system shall have the following functionalities:

1. User Registration: The system should allow users to create accounts and register their details, including their name, email address, and password.
2. User Profile: The system should allow users to create and manage their profiles.

3. Course Catalog: The system should provide a comprehensive catalog of sign language levels available for users to browse and select.
4. Image and Gif Tutorials: The system should provide video tutorials that teach users how to perform different signs.
5. Quizzes/Test: The system should provide interactive quizzes that test the user's knowledge of sign language and provide personalized feedback.
6. Progress Tracking: The system should allow users to track their progress in lessons and quizzes and view their performance statistics.
7. Gamification: The system should incentivize learning by providing rewards such leaders score board for completing lessons and tests/quizzes.

These are some of the functional requirements for the Sign Language Tutor Application system. The specific requirements of the system may vary based on the needs of its users and stakeholders.

3.4.2Non-Functional Requirements

Non-functional requirements describe the characteristics and qualities that the Sign Language Tutor Application system should possess to ensure its performance, reliability, usability, and security. Below are non-functional requirements for the system:

1. **Performance**: The system should respond to user requests promptly, and video tutorials should be available for streaming without buffering or delays.
2. **Availability**: The system should be available to users 24/7, with minimal downtime for maintenance or updates.
3. **Reliability**: The system should be reliable and operate without errors or crashes.
4. **Scalability**: The system should be designed to handle increasing numbers of users and course enrollments without experiencing performance degradation.
5. **Security**: The system should be secure, and user data should be protected from unauthorized access or breaches.
6. **Usability**: The system should be easy to use and intuitive, with clear instructions and a user-friendly interface.
7. **Accessibility**: The system should be designed to be accessible to users with disabilities, such as those who are deaf or hard of hearing.
8. **Compatibility**: The system should be compatible with different devices, browsers, and operating systems, ensuring that users can access the system from different platforms.
9. **Maintainability**: The system should be maintainable, with a modular design that allows for easy upgrades and enhancements.
10. **Privacy**: The system should comply with data privacy laws and protect users' personal data and information.
11. **Performance Efficiency**: The system should be designed to operate with minimum resource usage, ensuring optimal performance.

12. **Interoperability**:
    The system should be interoperable, allowing integration with other systems or platforms if necessary.

These are non-functional requirements for the Sign Language Tutor Application system. These requirements are important to ensure the system's reliability, usability, and security, which are essential for providing a positive user experience.

3.5 Use Case diagram for the proposed system

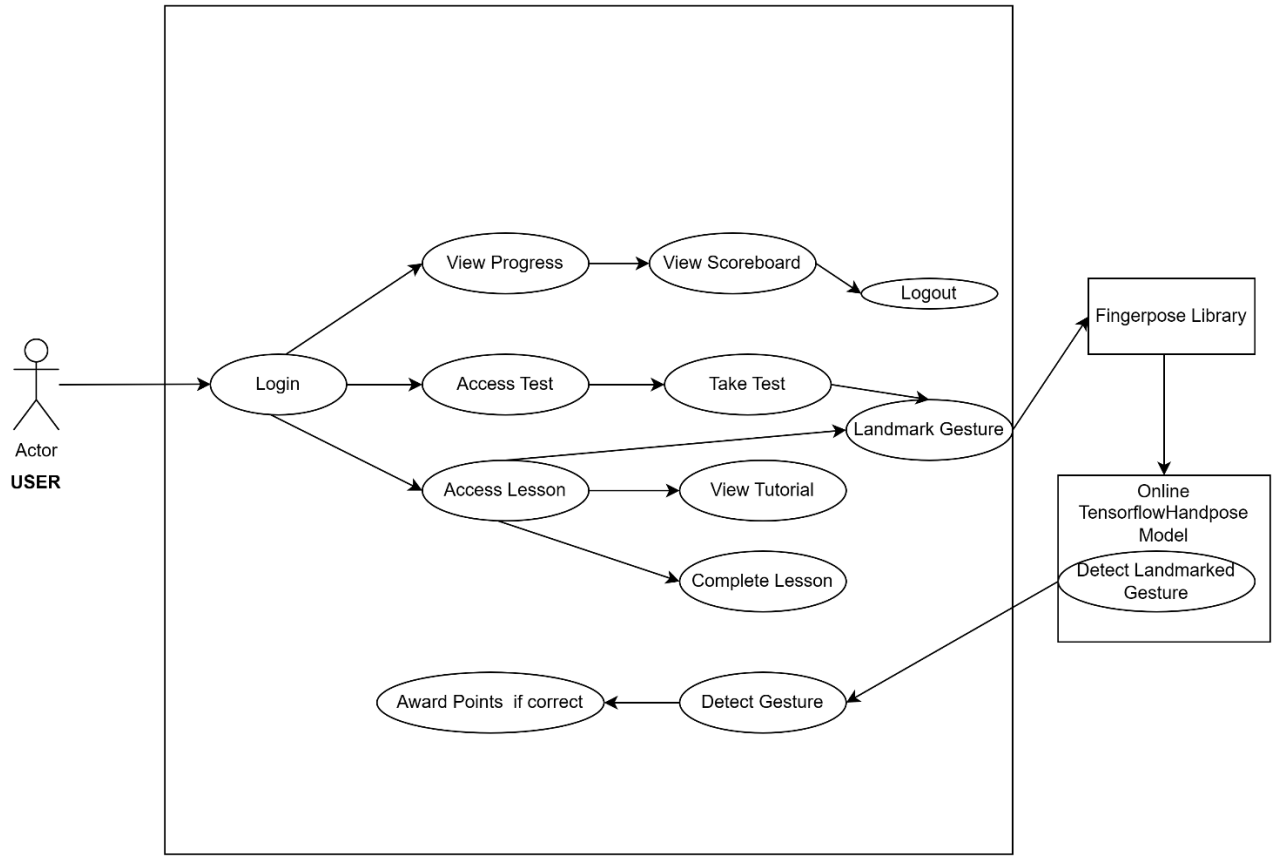


Fig 5: UseCase Diagram

The above diagram is a use case diagram for the Sign Language Tutor system, which shows the various use cases available for the users of the system, as well as the Handpose Model and Fingerpose Library that are used for gesture detection. The diagram shows that there is an actor called User who can log in to the system, access lessons and tests, view their progress, view their scoreboard, and log out of the system.

The system also allows the user to complete a lesson, take a test, view a tutorial, landmark a gesture, detect a gesture, and be awarded points if they correctly perform a gesture. The Handpose Model is

responsible for detecting a landmarked gesture, while the Fingerpose Library is responsible for landmarking a gesture. The diagram also shows the relationships between the various use cases, with arrows indicating how they are related to each other.

CHAPTER 4:Design

4.0 Introduction

The Sign Language Tutor system design is aimed at providing a platform for users to learn and practice sign language. The system has several components, including the user interface, the hand pose model, the finger pose library, and the backend database.

The user interface allows users to log in, access lessons, complete lessons, access tests, take tests, view scoreboards, view progress, and view tutorials. Once logged in, users can access the lessons and complete them. They can also access the tests and take them. After completing a lesson or taking a test, users can view their progress and their score on the scoreboard.

The finger pose library is responsible for landmarking the gesture, which involves recognizing the hand pose and mapping it to a particular sign language gesture. The hand pose model detects the landmarked gesture and matches it with the corresponding gesture from the finger pose library. If the gesture is correct, the system awards the user with points.

The backend database is responsible for storing user information, including user profiles, completed lessons, and completed tests. It also stores information about lessons, tests, and gestures.

Overall, the system design is aimed at providing users with a seamless experience as they learn and practice sign language.

4.1 System Diagram

4.1.1 Level 0 DFD

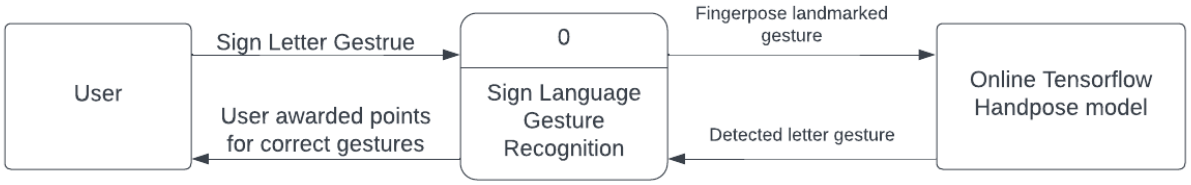


Fig 1: Level 0 DFD

4.1.2 Level 1 DFD

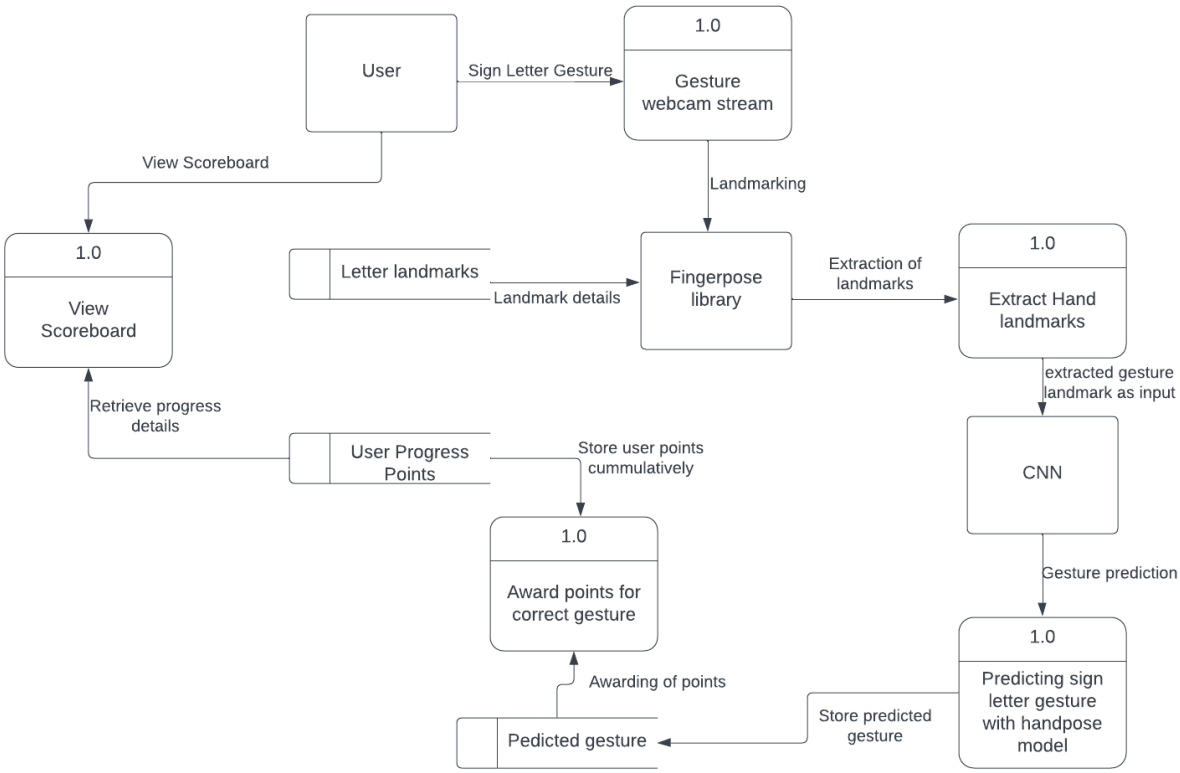


Fig 2: Level 2 DFD

4.1.3 Activity diagram

Base(simplified/core) activity diagram

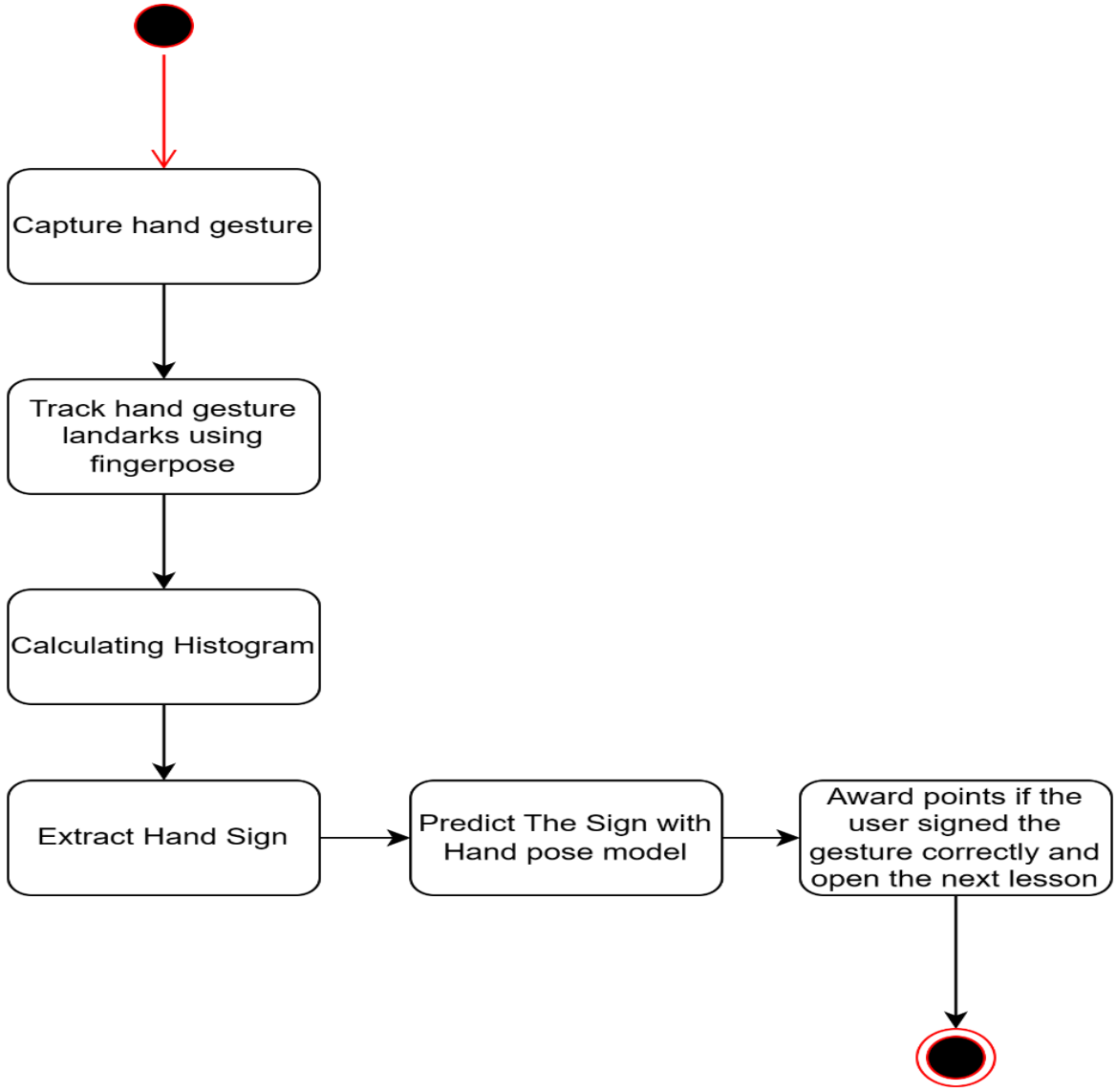


Fig 3: Activity Diagram (Core)

Overview of the Activity Diagram

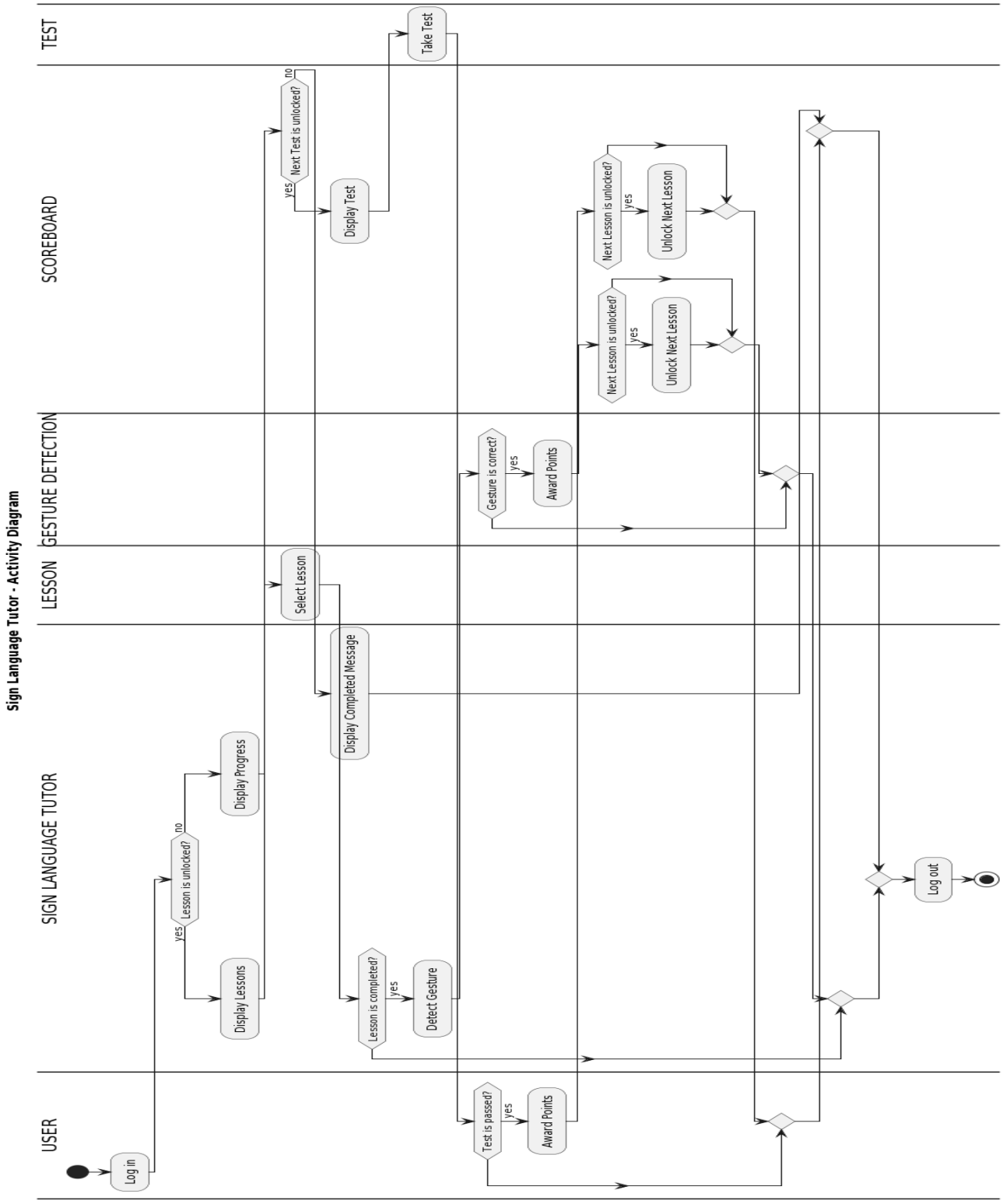


Fig 4: Activity Diagram (Overview)

The activity diagram illustrates the sequence of actions in the Sign Language Tutor web application. Initially, the user aims the camera towards a sign language gesture, and the system captures hand landmarks of the gesture using fingerpose landmark library. After that, the landmarked image is subjected to pre-processing and feature extraction. Then, the system employs gesture recognition to detect the sign language gesture present in the image, and subsequently retrieves its meaning. Finally, the system showcases if the user has done the right gesture for the letter and award points

It's essential to note that this activity diagram is a simplified version and does not incorporate error handling or other related actions. Nonetheless, it offers a fundamental understanding of the overall flow of activities in the system.

4.2 Architectural Design

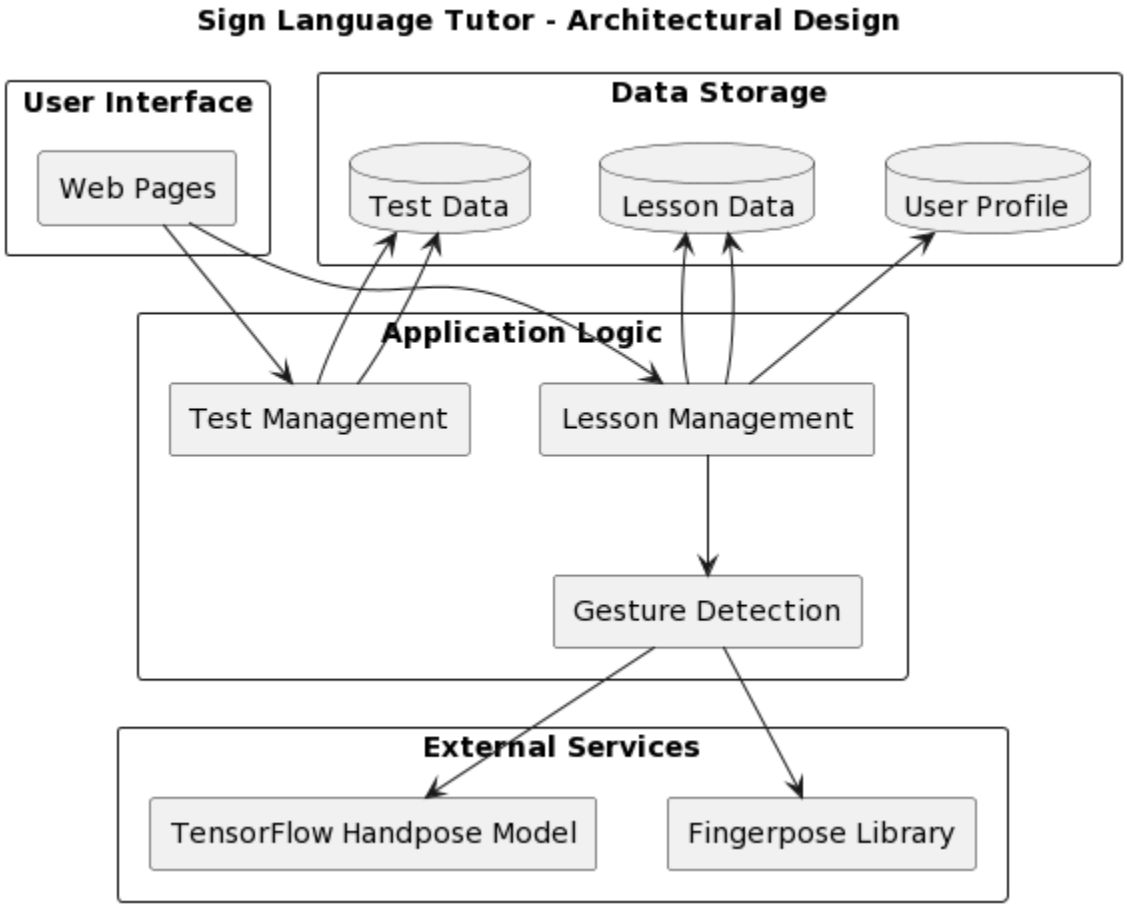


Fig 5: Architectural Design Diagram

At a high level, the architectural design provides a blueprint for the overall structure and organization of the system. It breaks down the system into several components that work together to provide the desired functionality.

Let's take a closer look at each of the components:

1. **User Interface:**
   This component is responsible for the graphical user interface (GUI) that users interact with. The User Interface component is further broken down into Web Pages, which provide an intuitive and user-friendly interface for the user to access lessons and tests.

2. **Application Logic:**
   The Application Logic component comprises the Lesson Management, Test Management, and Gesture Detection classes. These classes are responsible for the core functionality of the system.

   The Lesson Management class manages the user's progress through the lessons, including which lessons they have completed and which lessons are available to them. The Test Management class manages the user's progress through the tests, including which tests they have completed and which tests are available to them.

   The Gesture Detection class is responsible for detecting the user's hand gestures during the lessons and tests. It uses the Fingerpose Library and TensorFlow Handpose Model to analyze the user's hand movements and recognize the corresponding sign language gestures.

3. **Data Storage:** The Data Storage component is responsible for storing and managing the data used by the system. It is comprised of three databases: the User Profile database, the Lesson Data database, and the Test Data database.
   The User Profile database stores the user's account information, including their name, email, password, and progress through the lessons and tests. The Lesson Data database stores the lesson content, including the gestures that the user needs to learn for each lesson. The Test Data database stores the test content, including the questions and answers for each test.

4. **External Services:**
   The External Services component includes the Fingerpose Library and TensorFlow Handpose Model. These are third-party services that the Gesture Detection class utilizes to analyze the user's hand movements and recognize sign language gestures.By breaking down the system into these components, the architectural design allows for a modular and scalable design. Each component can be developed and tested separately, making it easier to identify and fix issues as they arise. Additionally, the design allows for easy addition or removal of components as necessary to improve functionality or address user needs.

4.2.1 Hardware

The hardware architecture of the Sign Language Tutor system would require a range of devices that interact with each other to provide an effective user experience. The primary device would be a desktop computer or a laptop that serves as the main platform for running the application. Additionally, a mobile device such as a tablet or a smartphone could be used to access the application remotely.

The main device would require a high-resolution display capable of rendering the gestures and other visual components of the system accurately. A standard keyboard and mouse or a touchpad would be sufficient for interacting with the application's graphical user interface. Additionally, a camera would be required to capture the user's hand gestures, and a microphone would be needed for recording audio.

The mobile device would require a high-resolution display, a camera for gesture recognition, and a microphone for audio recording. It would also need to be capable of connecting to the internet, either through a cellular network or a Wi-Fi connection.

To ensure smooth communication between the devices and the server, a reliable network infrastructure would be necessary. The server would store the application data and process requests from the client devices. It would require high-speed internet connectivity, powerful processors, and ample storage capacity to handle the large volumes of data generated by the application.

Overall, the hardware architecture of the Sign Language Tutor system would require a combination of devices, network infrastructure, and server components to provide an effective user experience.

4.2.2 Networking

A network architectural design for the Sign Language Tutor system would typically involve multiple components, including client devices, web servers, application servers, and databases. The design would aim to ensure that the system is highly available, scalable, and secure.

At the client device level, users would typically use a web browser to access the Sign Language Tutor system. The system would be hosted on one or more web servers, which would be responsible for handling user requests, and for serving up the web pages and other resources needed by the application. These web servers could be configured to operate behind a load balancer, which would distribute incoming requests across multiple servers to ensure that the system can handle a large number of concurrent users.

The web servers would also communicate with one or more application servers, which would be responsible for handling the business logic of the Sign Language Tutor system. These application servers would run the various components of the system, such as the gesture detection and recognition models, and the various algorithms for scoring and awarding points to users.

The application servers would access the handpose model for gesture recognition through an online API, hosted on a tensorflow server. This server would be responsible for managing and serving up the

handpose model and the associated data, and for handling the communication between the application servers and the handpose model.

Finally, the system would also include one or more databases for storing user data, including user profiles, progress, scores, and other related data. These databases would be hosted on dedicated database servers, and would be designed to be highly available, scalable, and secure.

Overall, this network architectural design would aim to provide a scalable, highly available, and secure platform for delivering the Sign Language Tutor system to a large number of users. By separating out the various components of the system and using load balancers, redundant servers, and other high availability techniques, the system could be designed to provide a seamless and reliable user experience.

4.3 Database Design

4.3.1 E-R Diagram

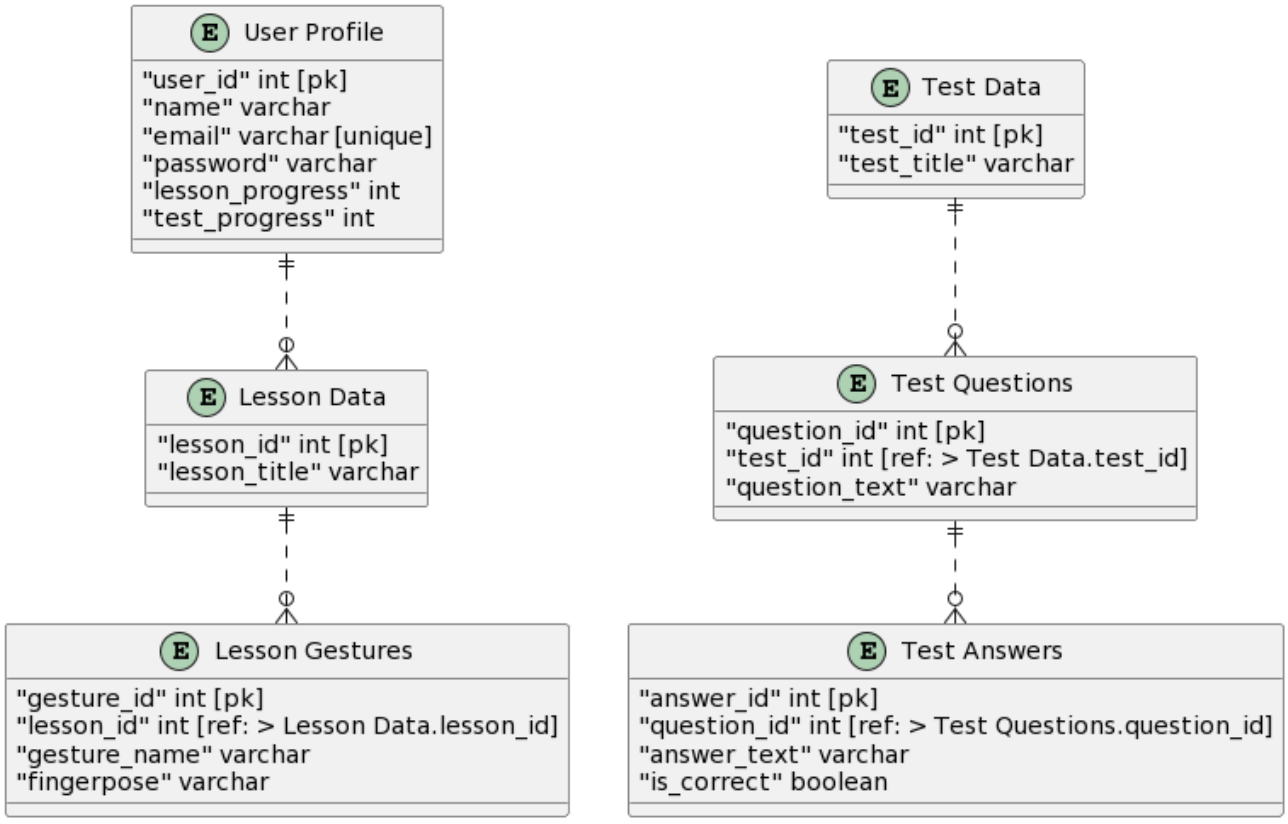


Fig 6: E-R Diagram

The Sign Language Tutor database is designed to store data related to the application's users, lessons, tests, gestures, and scoreboard. Here is a brief description of each entity:

1. **User Profile**: This entity contains information about the application's users, including their ID, name, email, password, points, completed lessons, and completed tests.
2. **Lesson:** This entity contains information about the lessons available in the application, including the lesson ID, name, and gestures associated with the lesson.
3. **Test:** This entity contains information about the tests available in the application, including the test ID, name, gestures associated with the test, and the ID of the lesson the test belongs to.
4. **Gesture:** This entity contains information about the gestures available in the application, including the gesture ID, name, fingerpose associated with the gesture, and the ID of the lesson the gesture belongs to.
5. **Scoreboard**: This entity contains information about the scores of users, including the scoreboard ID, the ID of the user, the ID of the lesson, the ID of the test, the score achieved by the user, and the date on which the user completed the lesson.

These entities are designed to store data in a structured manner and ensure data integrity, making it easier to maintain and query the data over time. The database is designed to be scalable, allowing for the addition of new users, lessons, tests, and gestures without requiring significant changes to the data model.

4.3.2 Normalized Database

First Normal Form (1NF)

To convert the database to first normal form, we need to ensure that each column has atomic values, which means we need to eliminate any repeating groups.

Our current database is already in first normal form, as all columns contain atomic values and there are no repeating groups.

Second Normal Form (2NF)

To achieve second normal form, we need to ensure that each non-key attribute is dependent on the entire primary key. Currently, the **completed_lessons** and **completed_tests** attributes in the **User Profile** table are multi-valued and depend only on a part of the primary key. To remove this partial dependency, we can split the **User Profile** table into two tables, **User** and **User Progress**, as follows:

User {

user_id : int (PK)

name : varchar

email : varchar

password : varchar

points : int

}

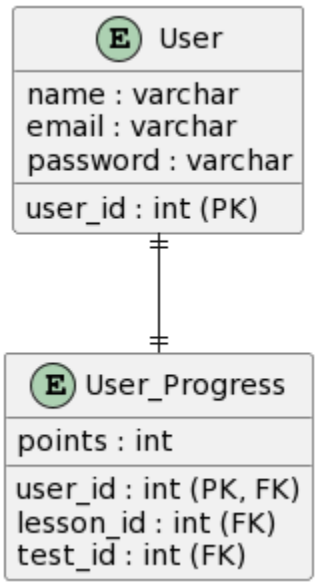
User Progress {

user_id : int (PK, FK)

lesson_id : int (FK)

test_id : int (FK)

}



Third Normal Form (3NF)

To achieve third normal form, we need to ensure that there are no transitive dependencies, where non-key attributes are dependent on other non-key attributes. Currently, the **points** attribute in the **User Progress** table is dependent on the **lesson_id** and **test_id** attributes, which are non-key attributes. To remove this transitive dependency, we can split the **User Progress** table into three tables, **User**, **Lesson Progress**, and **Test Progress**, as follows:

User {

user_id : int (PK)

name : varchar

email : varchar

password : varchar

}

Lesson Progress {

user_id : int (FK, PK)

lesson_id : int (FK, PK)

points : int

}

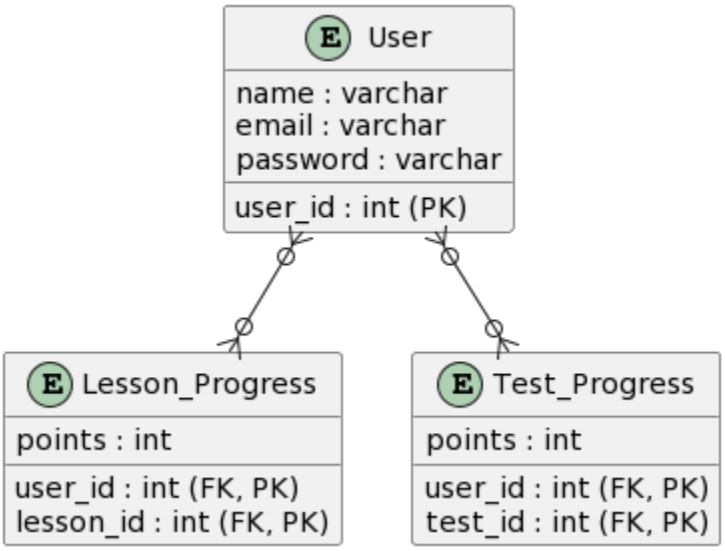
Test Progress {

user_id : int (FK, PK)

test_id : int (FK, PK)

points : int

}



4.4 Program Design

4.4.1 Class diagram

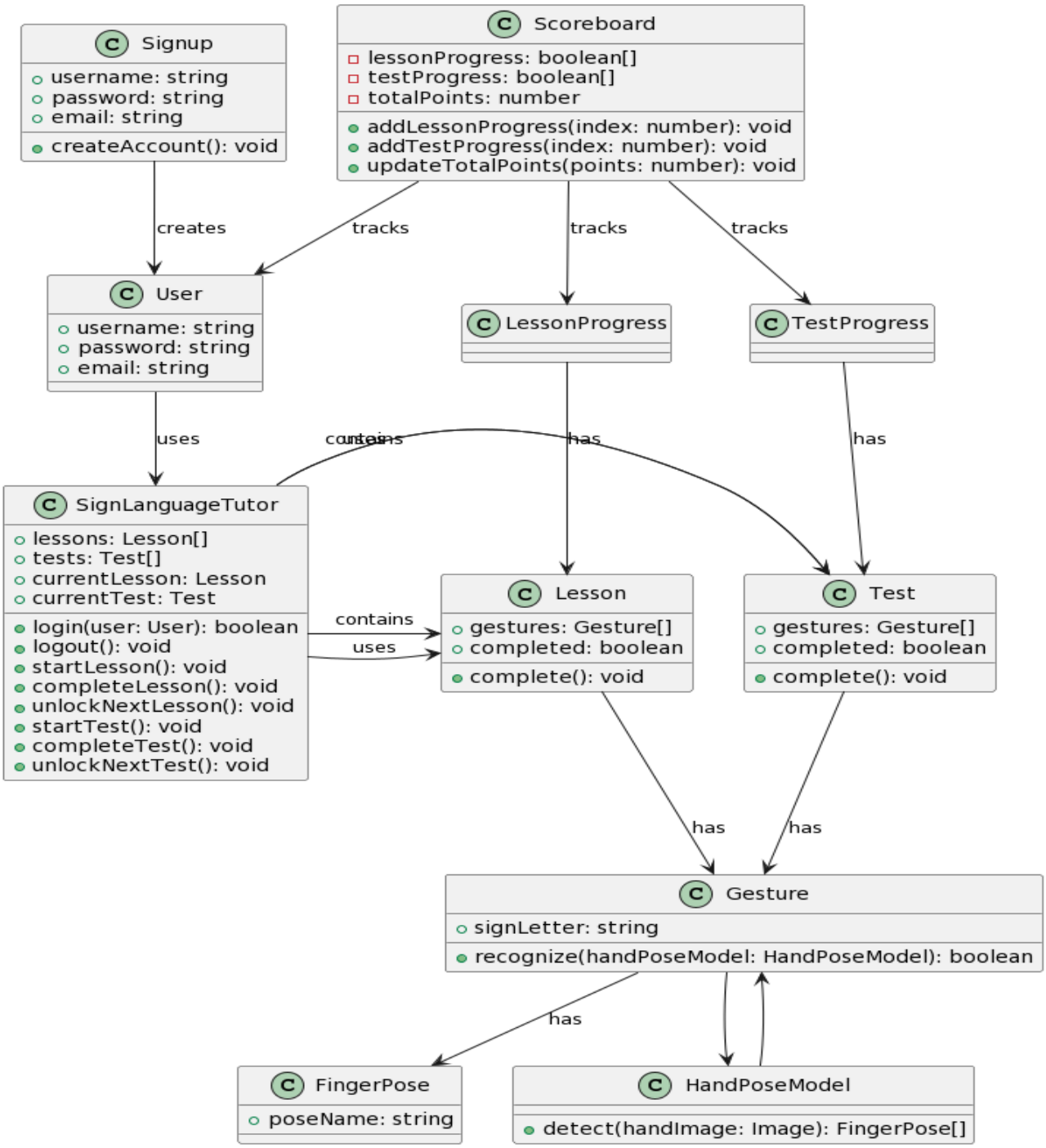


Fig 7: Class Diagram

This class diagram showcases the main classes and their relationships within the Sign Language Tutor Application.

Explanation of the class diagram

1. The `User` class represents a user of the `SignLanguageTutor` application, and has attributes such as `username`, `password`, and `email`.
2. The `Scoreboard` class tracks the user's progress and has attributes such as `totalPoints` and `activityProgress`, as well as methods to update and retrieve this information.
3. The `SignLanguageTutor` class is the main class of the application, and has methods for handling user authentication, starting and completing activities, updating the user's progress, and unlocking the next activity.
4. The `Activity` class is an abstract class that represents a lesson or a test, and has attributes such as `activityName`, `activityType`, and `activityStatus`, as well as methods to start and complete the activity.
5. The `Lesson` class is a subclass of `Activity` that represents a sign language lesson, and has attributes such as `lessonNumber` and `lessonGestures`, as well as methods to start and complete the lesson.
6. The `Test` class is a subclass of `Activity` that represents a sign language test, and has attributes such as `testNumber` and `testGestures`, as well as methods to start and complete the test.
7. The `Gesture` class represents a sign gesture, and has attributes such as `gestureName` and `signLetter`, as well as methods to recognize and compare gestures using the `HandPoseModel` and the `FingerPose` class.
8. The `FingerPose` class represents a finger pose, and has attributes such as `fingerName` and `fingerAngle`, as well as methods to compare finger poses.
9. The `HandPoseModel` class represents the TensorFlow hand pose model used to detect finger poses in sign gestures.

4.4.2 Sequence diagram

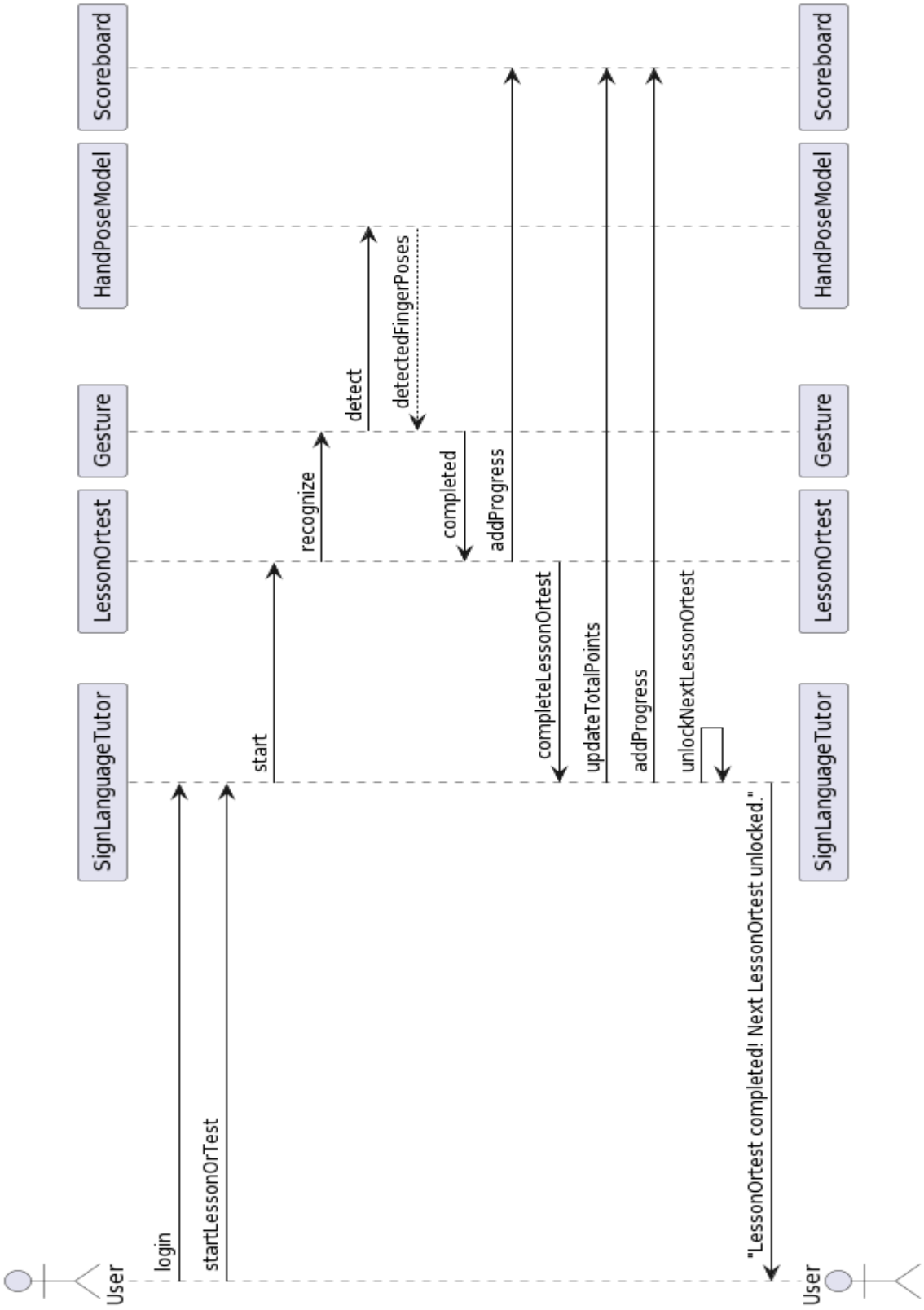


Fig 8: Sequence Diagram

A sequence diagram for the Sign Language Tutor Application depicts the interactions between the different components and subsystems of the system during the execution of a particular function or task. Here's a possible scenario:

1. The `User` logs in to the `SignLanguageTutor` application.
2. The `User` starts an `Activity`, which could be either a `Lesson` or a `Test`.
3. The `SignLanguageTutor` starts the `Activity`.
4. The `Activity` uses the `Gesture` class to recognize sign gestures using the `HandPoseModel`.
5. The `HandPoseModel` detects finger poses and returns the detected poses to the `Gesture` class.
6. The `Gesture` class determines if the `Activity` was completed successfully and returns a completion message to the `Activity`.
7. The `Activity` adds progress to the `Scoreboard` by calling its `addProgress` method.
8. The `SignLanguageTutor` marks the `Activity` as completed by calling its `completeActivity` method.
9. The `SignLanguageTutor` updates the user's total points on the `Scoreboard` by calling its `updateTotalPoints` method.
10. The `SignLanguageTutor` adds progress to the `Scoreboard` by calling its `addProgress` method.
11. The `SignLanguageTutor` checks if the user has completed enough activities to unlock the next one, and if so, unlocks the next `Activity`.
12. The `SignLanguageTutor` displays a completion message to the `User` indicating that the `Activity` was completed successfully and the next `Activity` was unlocked.

4.4.3 Package diagram

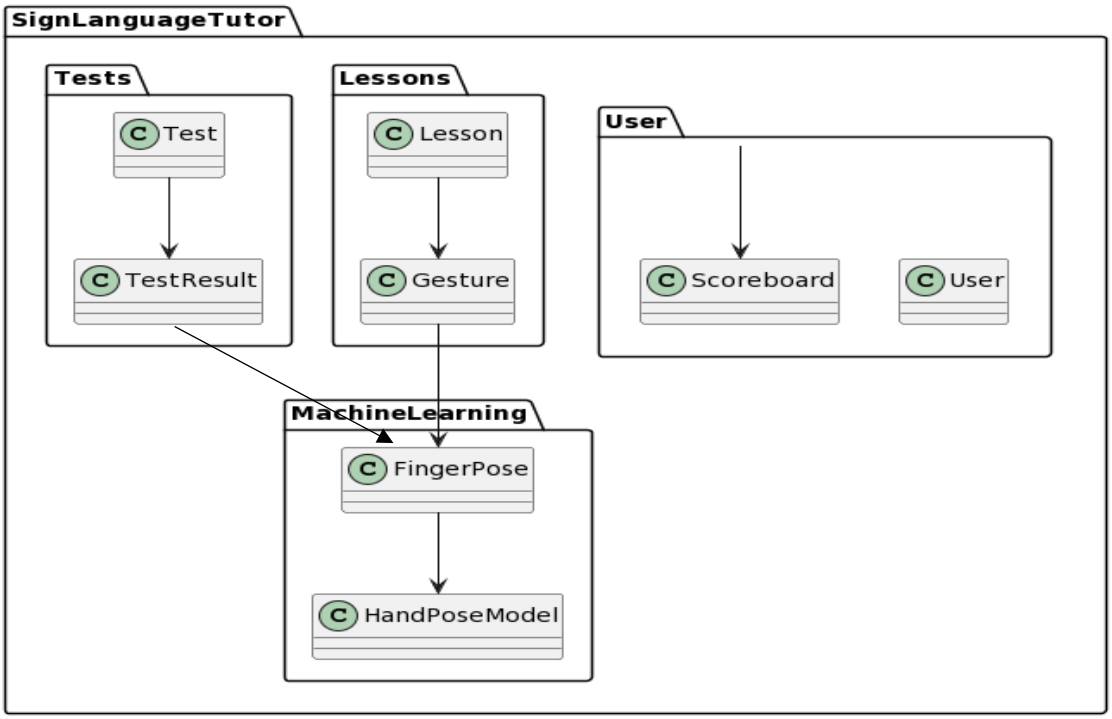


Fig 8: Package Diagram

4.5 Pseudo Code

Letter A accurate landmark

```
import * as fp from "fingerpose";

const letterA = new fp.GestureDescription("A");

letterA.addCurl(fp.Finger.Thumb, fp.FingerCurl.NoCurl);
letterA.addDirection(fp.Finger.Thumb, fp.FingerDirection.DiagonalUpLeft, 1.0);
letterA.addDirection(fp.Finger.Thumb, fp.FingerDirection.DiagonalUpRight, 1.0);

letterA.addDirection(fp.Finger.Thumb, fp.FingerDirection.VerticalUp, 0.9);

for (let finger of [
    fp.Finger.Index,
    fp.Finger.Middle,
    fp.Finger.Ring,
    fp.Finger.Pinky,
]) {
    letterA.addCurl(finger, fp.FingerCurl.FullCurl, 1.0);
    letterA.addDirection(finger, fp.FingerDirection.VerticalUp, 1);
    letterA.addDirection(finger, fp.FingerDirection.DiagonalUpLeft, 0.9);
    letterA.addDirection(finger, fp.FingerDirection.DiagonalUpRight, 0.9);
}

export default letterA;
```

Single learning

```
import React, { useRef, useState, useEffect, useReducer } from "react";
import { useSelector, useDispatch } from "react-redux";
import * as tf from "@tensorflow/tfjs";
import * as handpose from "@tensorflow-models/handpose";
import Webcam from "react-webcam";
import * as fp from "fingerpose";
import { fetchPhrases, unlockPhrases } from "../store/phrases";
import { addPoints } from "../store/points";
import { allGestures } from "../letterGestures";
import { useHistory } from "react-router-dom";

const SingleLearning = props => {
    const learningPoints = 10;
    const dispatch = useDispatch();
    const history = useHistory();
    const webcamRef = useRef(null);
    const canvasRef = useRef(null);

    const [currentLetter, setLetter] = useState("");
    const [emoji, setEmoji] = useState(null);
    const [images, setImages] = useState({});
    let allLetters = useSelector(state => state.phrases);
```

```
const lettersOnly = allLetters.map(letter => letter.letterwords);
//Object is now 2d array: [[key1,value1], [key2,value2]]
const currentGestures = Object.entries(allGestures)
    .filter(entry => {
        //key = key1 & value = value1  ..etc
        const [key, value] = entry;
        return lettersOnly.includes(key);
    })
    .map(entry => {
        const [key, value] = entry;
        return value;
    });

const gestureAccuracyMany = 10;
const gestureAccuracyOne = 9.5;

//setTimeout ids to clear
let timerBetweenLetterId;
let timerBetweenCompletionId;

//Like componentDidMount
useEffect(() => {
    dispatch(fetchPhrases(Number(props.match.params.tier)));
}, []);

//Like componentWillUpdate
useEffect(() => {
    const run = async () => {
        const intervalIds = await runHandpose();
        return intervalIds;
    };
    const intervalId = run();

    // Like componentWillUnmount
    return async () => {
        clearInterval(await intervalId);
        clearTimeout(timerBetweenLetterId);
        clearTimeout(timerBetweenCompletionId);
    };
}, [currentLetter]);

//componentWillUpdate to get allLetters
useEffect(() => {
    allLetters[0] ? setLetter(allLetters[0].letterwords) : "";
    setImages(
        allLetters.reduce((accu, letter) => {
            accu[letter.letterwords] = [letter.url, letter.textUrl];
            return accu;
        }, {})
    );
}, [allLetters]);

const runHandpose = async () => {
    const net = await handpose.load();
    //Loop and detect hands
```

```
    let intervalId = setInterval(async () => {
        let result = await detect(net);

        if (result === currentLetter) {
            clearInterval(intervalId);

            const letterIndex = lettersOnly.indexOf(currentLetter) + 1;

            if (letterIndex < lettersOnly.length) {
                timerBetweenLetterId = setTimeout(() => {
                    setLetter(lettersOnly[letterIndex]);
                }, 3000); // timer for between gestures
            } else {
                dispatch(unlockPhrases(Number(props.match.params.tier)));
                dispatch(addPoints(learningPoints));
                timerBetweenCompletionId = setTimeout(() => {
                    history.push({
                        pathname: "/completionPage",
                        state: { tier: Number(props.match.params.tier) },
                    });
                }, 3000);
            }
        }
    }, 100);

    //return id of timer to clear when component unmounts
    return intervalId;
};

const detect = async net => {
    //Check data is available
    if (
        typeof webcamRef.current !== "undefined" &&
        webcamRef.current !== null &&
        webcamRef.current.video.readyState === 4
    ) {
        //Get video properties
        const video = webcamRef.current.video;
        const videoWidth = webcamRef.current.video.videoWidth;
        const videoHeight = webcamRef.current.video.videoHeight;

        //Set video height and width
        webcamRef.current.video.width = videoWidth;
        webcamRef.current.video.height = videoHeight;

        //Set canvas height and width
        canvasRef.current.width = videoWidth;
        canvasRef.current.height = videoHeight;

        // Make detections
        const hand = await net.estimateHands(video);

        // Gesture detections
        if (hand.length > 0) {
            const GE = new fp.GestureEstimator(currentGestures);
```

```
                //second argument is the confidence level
                const gesture = await GE.estimate(hand[0].landmarks, 8);

                if (gesture.gestures !== undefined && gesture.gestures.length > 0)
{
                    const confidence = gesture.gestures.map(
                        prediction => prediction.score
                    );

                    const maxConfidence = confidence.indexOf(
                        Math.max.apply(null, confidence)
                    );

                    // prints current hand gesture
                    // console.log(gesture);

                    const maxGesture = gesture.gestures[maxConfidence];

                    if (
                        (gesture.gestures.length === 1 &&
                            maxGesture.score >= gestureAccuracyOne) ||
                        maxGesture.score >= gestureAccuracyMany
                    ) {
                        if (
                            (maxGesture.name === "T" || maxGesture.name === "S") &&
                            (currentLetter === "T" || currentLetter === "S")
                        ) {
                            setEmoji(currentLetter);
                            return currentLetter;
                        } else if (
                            (maxGesture.name === "R" || maxGesture.name === "U") &&
                            (currentLetter === "R" || currentLetter === "U")
                        ) {
                            setEmoji(currentLetter);
                            return currentLetter;
                        } else {
                            setEmoji(maxGesture.name);
                            return maxGesture.name;
                        }
                    }
                }
            }
        }
    };

    let emojiPrint =
        emoji === currentLetter ? (
            <img
                src="/CheckMark.png"
                style={{
                    position: "relative",
                    marginLeft: "auto",
                    marginRight: "auto",
                    left: 150,
                    bottom: -240,
                    right: 0,
```

```
                    textAlign: "center",
                    height: 100,
                }}
            />
        ) : (
            ""
        );

    return (
        <div className="App">
            <header className="App-header">
                <Webcam
                    ref={webcamRef}
                    className=" bg-yellow-300 border-4 border-gray-600"
                    style={{
                        position: "absolute",
                        marginLeft: "auto",
                        marginRight: "auto",
                        left: 0,
                        right: 0,
                        textAlign: "center",
                        zindex: 9,
                        width: 640,
                        height: 400,
                    }}
                />
                <canvas
                    ref={canvasRef}
                    style={{
                        position: "absolute",
                        marginLeft: "auto",
                        marginRight: "auto",
                        left: 0,
                        right: 0,
                        textAlign: "center",
                        zindex: 9,
                        width: 640,
                        height: 380,
                    }}
                />

                <img
                    src={images[currentLetter] ? "/" + images[currentLetter][0] :
null}
                    style={{
                        position: "relative",
                        marginLeft: "auto",
                        marginRight: "auto",
                        left: 30,
                        bottom: -440,
                        right: 100,
                        textAlign: "center",
                        height: 100,
                    }}
                />
```

```
                <img
                    src={images[currentLetter] ? "/" + images[currentLetter][1] :
null}
                    style={{
                        position: "relative",
                        marginLeft: "auto",
                        marginRight: "auto",
                        left: -100,
                        bottom: -340,
                        right: 100,
                        textAlign: "center",
                        height: 100,
                    }}
                />

                {emojiPrint}
            </header>
        </div>
    );
};

export default SingleLearning;
```

4.6 Interface Design

4.6.1 Introduction

The Sign Language Tutor interface implemented in React JS consists of several components that work together to provide a seamless and intuitive user experience. The interface is designed to be responsive and adaptable to different screen sizes and resolutions.

The main components of the interface include:

1. **Login Page**: The login is crafted with a simple design yet offering any eyecatching display for user friendly login
2. **Navigation Bar**: The navigation bar is fixed at the top of the screen and provides easy access to the different sections of the application. The navigation bar includes links to the Home page, Lessons, Tests and Progress..
3. **Home Page**: The home page provides a brief introduction to the Sign Language Tutor application and its features. The home page also includes a carousel of images highlighting different features of the application.
4. **Lessons**: The Lessons page provides a list of available lessons that the user can access. Each lesson includes a thumbnail image, lesson name, and a brief description. The user can click on a lesson to access it.
5. **Tests**: The Tests page provides a list of available tests that the user can take. Each test includes a thumbnail image, test name, and a brief description. The user can click on a test to start taking it.
6. **Progress**: The Progress page provides the user with an overview of their progress in the application. The page includes a graph showing the user's progress over time, as well as a list of completed lessons and tests.

The Sign Language Tutor interface is designed to be user-friendly and intuitive, with clear and concise information provided throughout the application. The use of colors and images helps to create an engaging and immersive experience for the user. The interface is also designed to be easily navigable, with a consistent layout and structure used throughout the application.

4.6.2 Screenshots of user interface

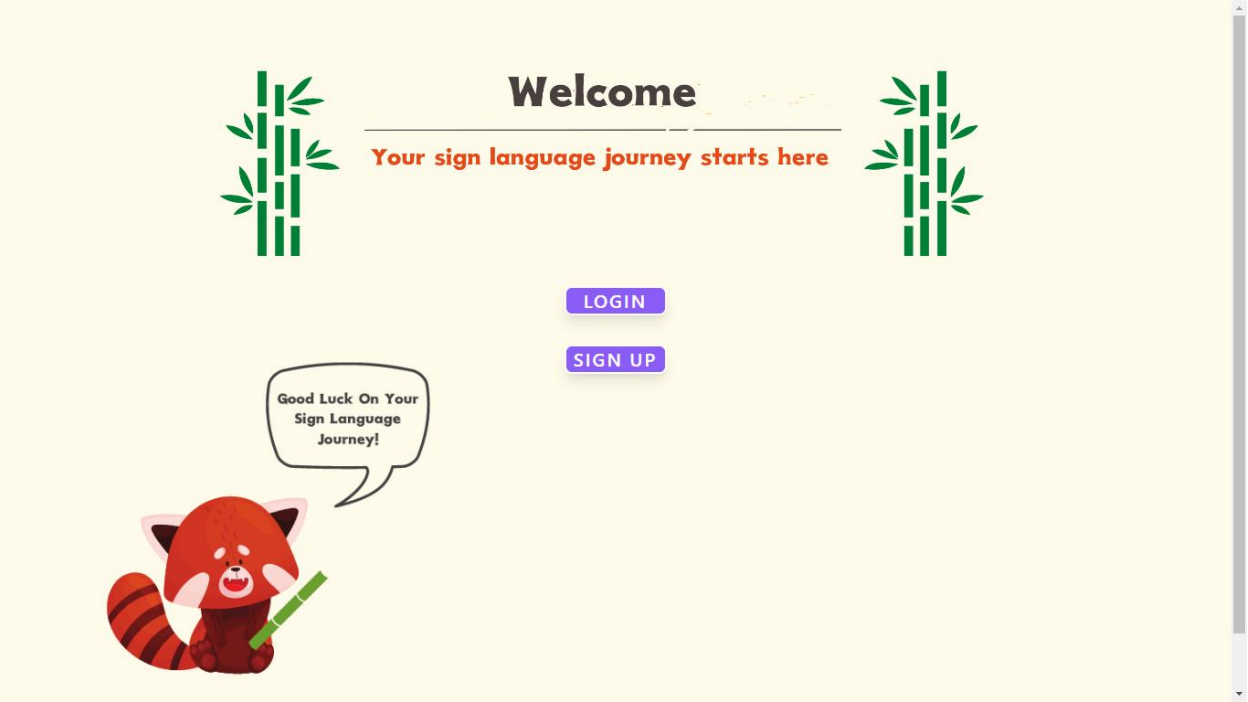


Fig 9: Login Page

Fig 10: Lessons Page(partly unlocked lessons)

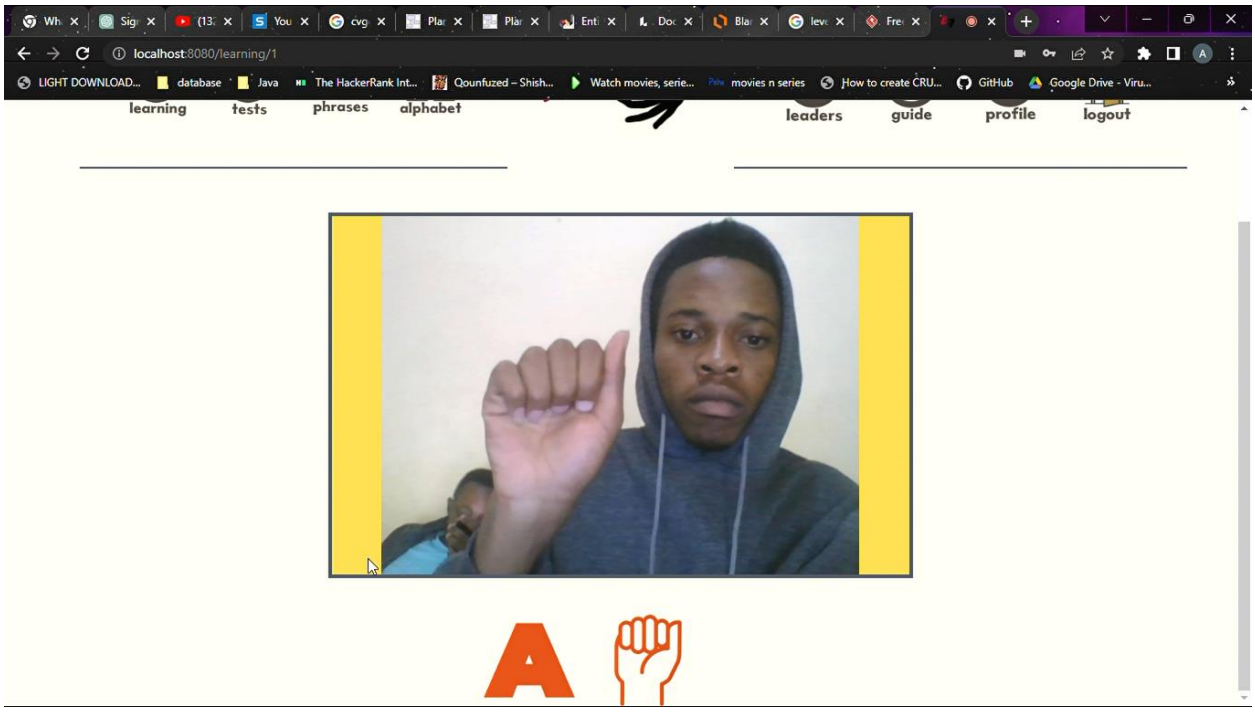


Fig 11: Lessons Gesture detection Page

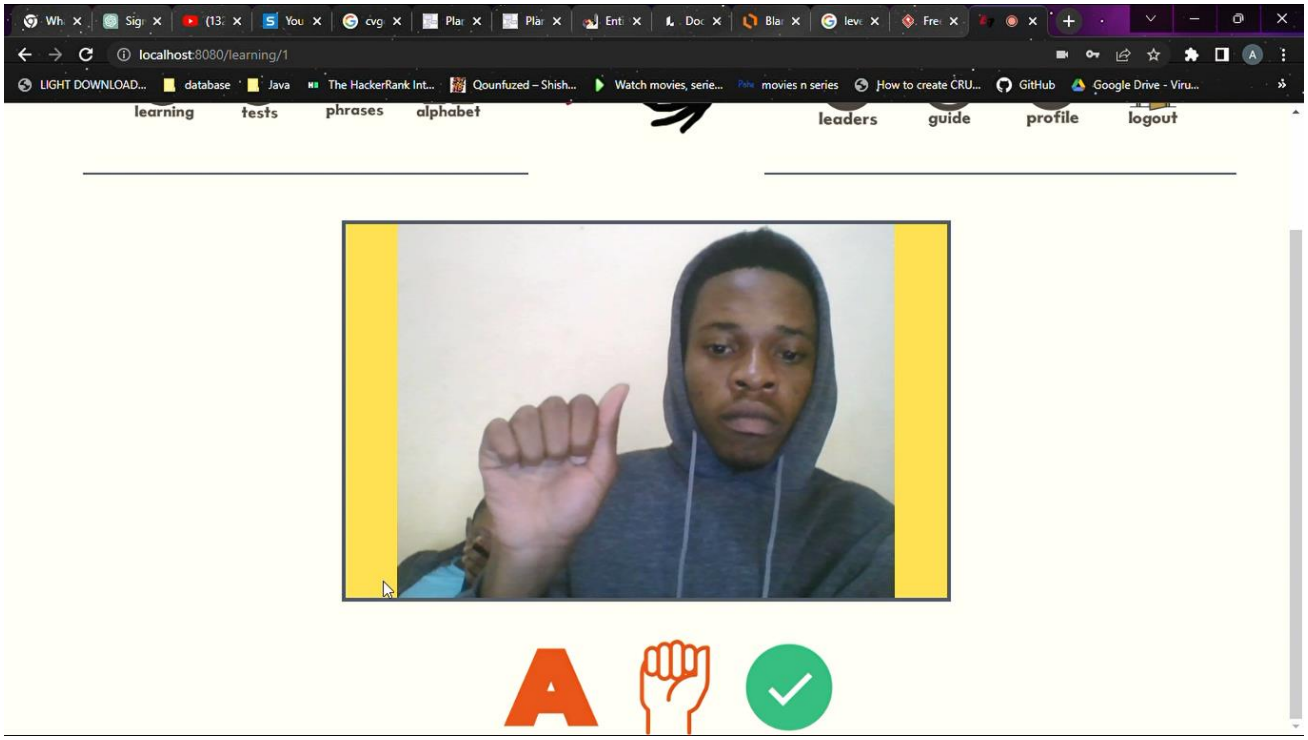


Fig 12: Lessons Gesture detection (correct) Page

Fig 13: Lessons Page(full unlocked lessons)

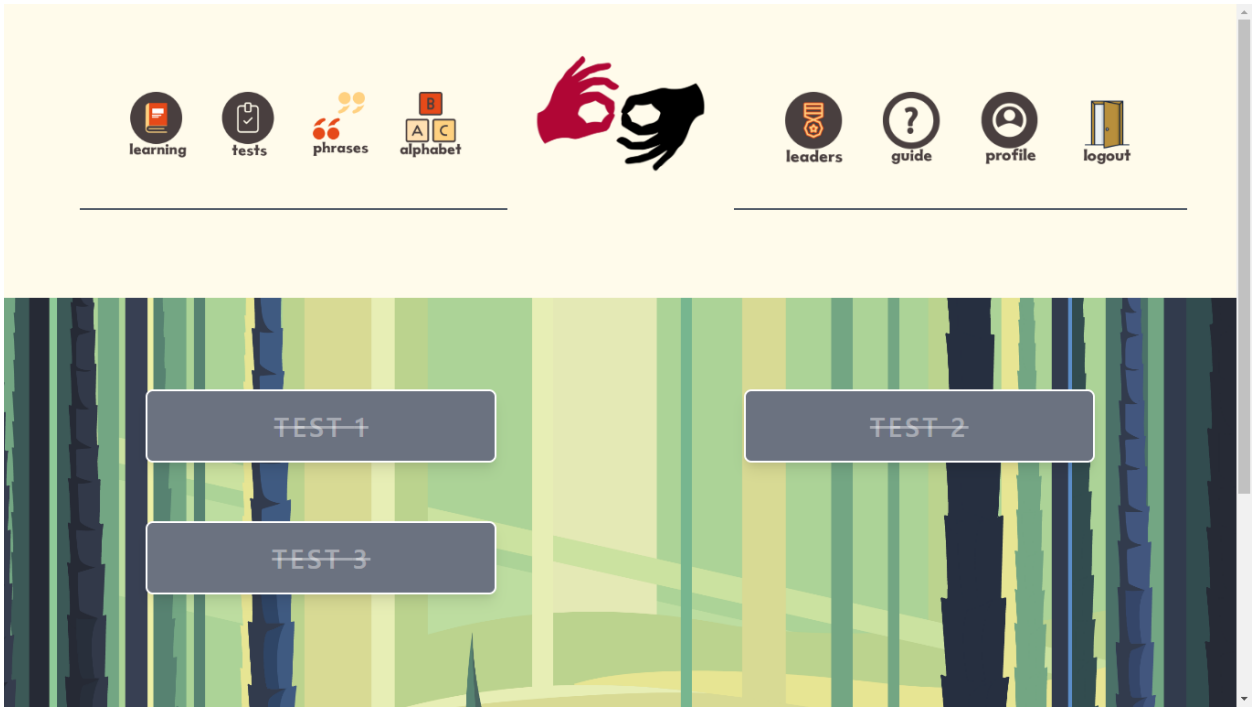


Fig 14: Test Page(All Tests are locked)

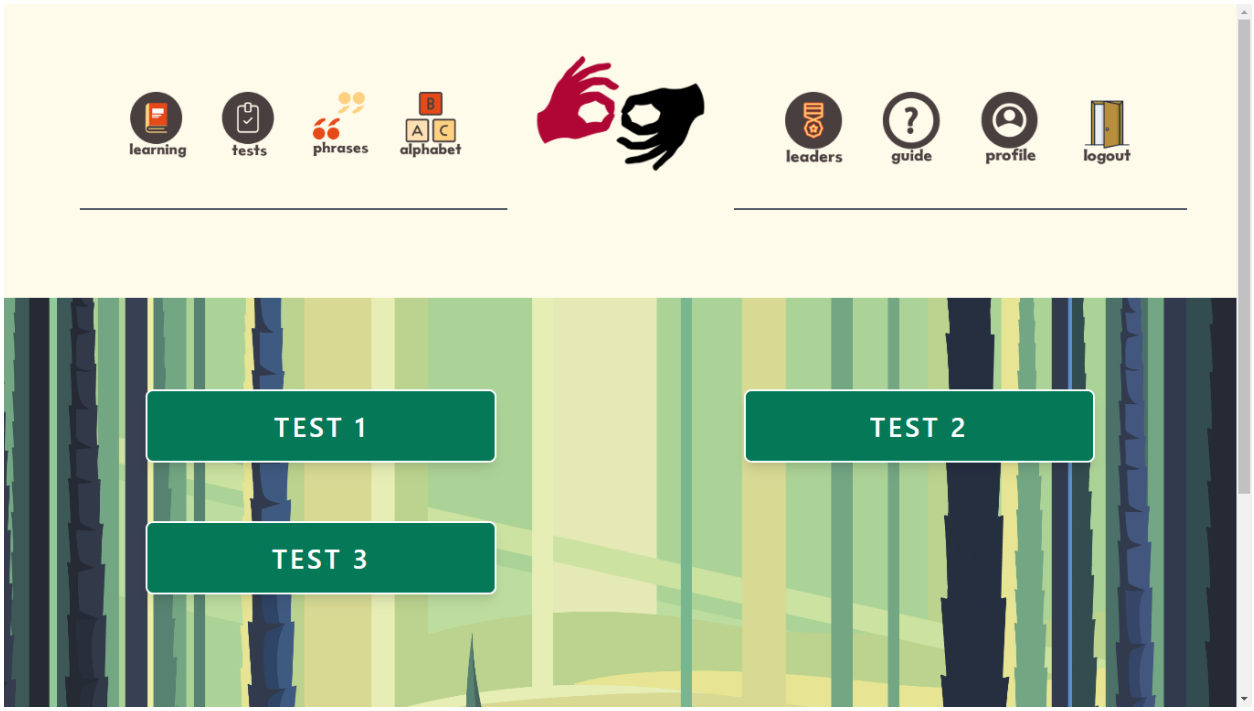


Fig 15: Test Page(All test are unlocked)

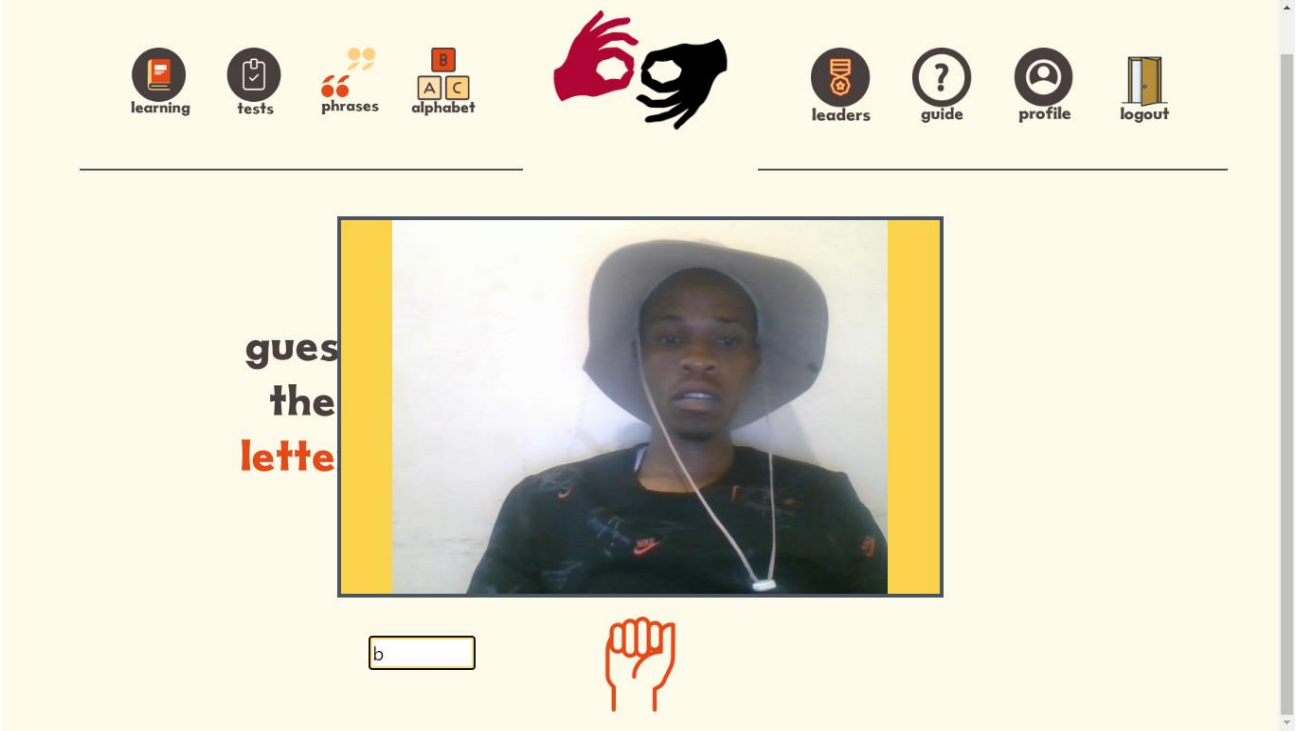


Fig 16: Test Question Page

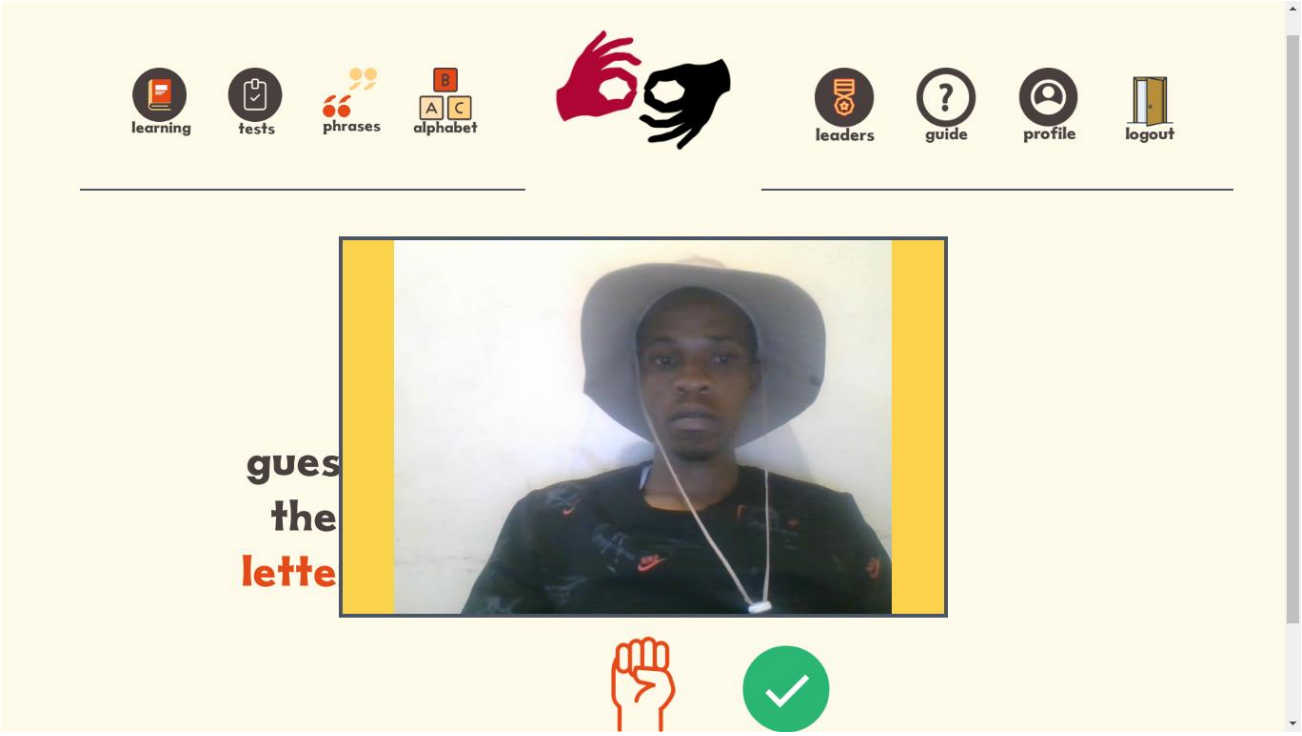


Fig 17: Test Question Page

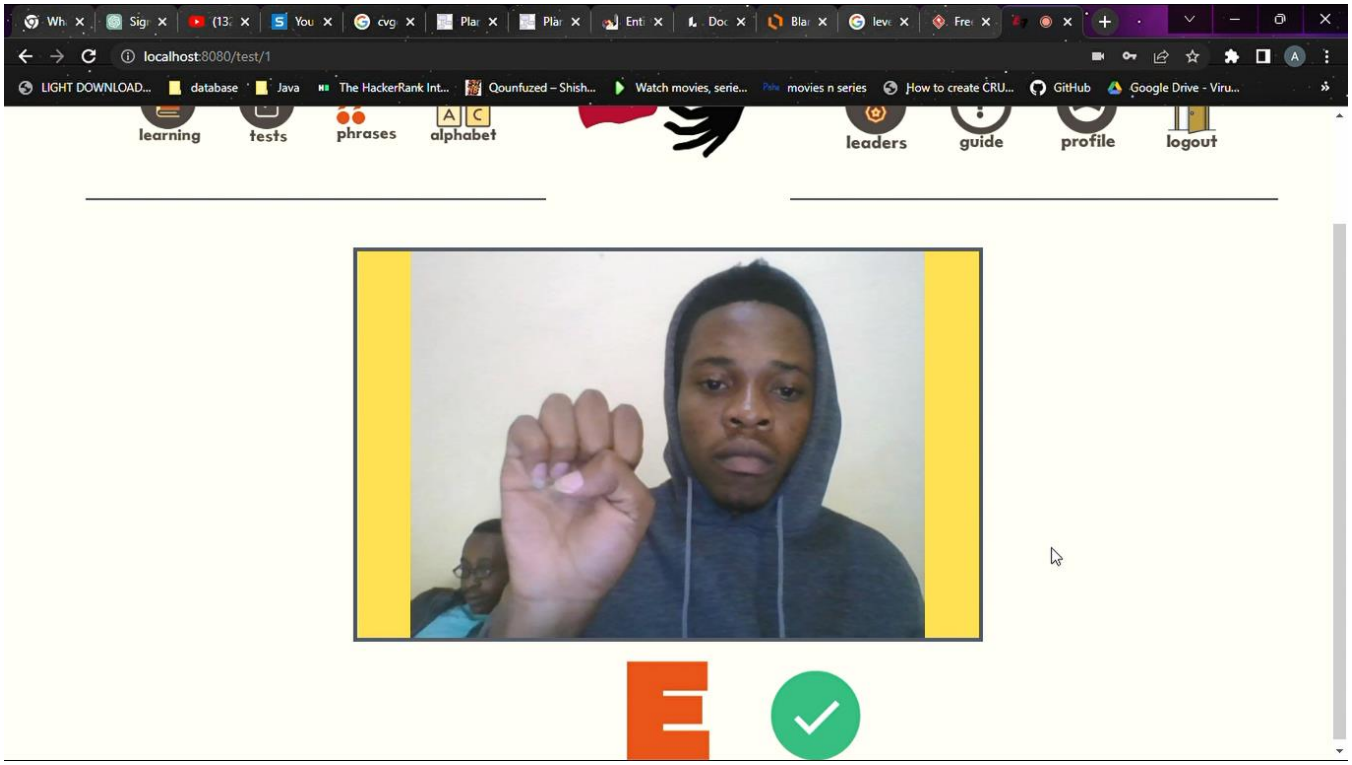


Fig 18: Test Gesture Page

Fig 19: Scoreboard Page

Chapter 5: Implementation and Testing

5.0 Introduction

The implementation stage is a critical phase in the development of the Sign Language Tutor project. In this phase, the design and plans developed in the previous stages will be put into action to create a functional system. The primary goal of this stage is to develop a working prototype of the Sign Language Tutor that meets the functional and non-functional requirements outlined in the earlier stages. This stage involves a significant amount of coding, testing, and debugging of the software components. Moreover, this stage will also include the integration of the different components and subsystems to create a cohesive and functional system. The implementation stage is a collaborative effort that involves the developers, project managers, and stakeholders working together to deliver a robust and functional Sign Language Tutor system

5.1 Pseudo Code

Login

```
const router = require("express").Router();
const {
    models: { User, Phrase },
} = require("../db");
module.exports = router;

router.post("/login", async (req, res, next) => {
    try {
        res.send({ token: await User.authenticate(req.body) });
    } catch (err) {
        next(err);
    }
});

router.post("/signup", async (req, res, next) => {
    try {
        const { email, password, firstname, lastname } = req.body;
        const user = await User.create({ email, password, firstname, lastname });
        res.send({ token: await user.generateToken() });
    } catch (err) {
        if (err.name === "SequelizeUniqueConstraintError") {
            res.status(401).send("User already exists");
        } else {
            next(err);
        }
    }
});
```

```
router.get("/me", async (req, res, next) => {
    try {
        res.send(await User.findByToken(req.headers.authorization));
    } catch (ex) {
        next(ex);
    }
});
```

SingleLearning

```
import React, { useRef, useState, useEffect, useReducer } from "react";
import { useSelector, useDispatch } from "react-redux";
import * as tf from "@tensorflow/tfjs";
import * as handpose from "@tensorflow-models/handpose";
import Webcam from "react-webcam";
import * as fp from "fingerpose";
import { fetchPhrases, unlockPhrases } from "../store/phrases";
import { addPoints } from "../store/points";
import { allGestures } from "../letterGestures";
import { useHistory } from "react-router-dom";

const SingleLearning = props => {
    const learningPoints = 10;
    const dispatch = useDispatch();
    const history = useHistory();
    const webcamRef = useRef(null);
    const canvasRef = useRef(null);

    const [currentLetter, setLetter] = useState("");
    const [emoji, setEmoji] = useState(null);
    const [images, setImages] = useState({});
    let allLetters = useSelector(state => state.phrases);

    const lettersOnly = allLetters.map(letter => letter.letterwords);
    //Object is now 2d array: [[key1,value1], [key2,value2]]
    const currentGestures = Object.entries(allGestures)
        .filter(entry => {
            //key = key1 & value = value1  ..etc
            const [key, value] = entry;
            return lettersOnly.includes(key);
        })
        .map(entry => {
            const [key, value] = entry;
            return value;
        });

    const gestureAccuracyMany = 10;
    const gestureAccuracyOne = 9.5;

    //setTimeout ids to clear
    let timerBetweenLetterId;
    let timerBetweenCompletionId;
```

```
//Like componentDidMount
useEffect(() => {
    dispatch(fetchPhrases(Number(props.match.params.tier)));
}, []);

//Like componentWillUpdate
useEffect(() => {
    const run = async () => {
        const intervalIds = await runHandpose();
        return intervalIds;
    };
    const intervalId = run();

    // Like componentWillUnmount
    return async () => {
        clearInterval(await intervalId);
        clearTimeout(timerBetweenLetterId);
        clearTimeout(timerBetweenCompletionId);
    };
}, [currentLetter]);

//componentWillUpdate to get allLetters
useEffect(() => {
    allLetters[0] ? setLetter(allLetters[0].letterwords) : "";
    setImages(
        allLetters.reduce((accu, letter) => {
            accu[letter.letterwords] = [letter.url, letter.textUrl];
            return accu;
        }, {})
    );
}, [allLetters]);

const runHandpose = async () => {
    const net = await handpose.load();
    //Loop and detect hands
    let intervalId = setInterval(async () => {
        let result = await detect(net);

        if (result === currentLetter) {
            clearInterval(intervalId);

            const letterIndex = lettersOnly.indexOf(currentLetter) + 1;

            if (letterIndex < lettersOnly.length) {
                timerBetweenLetterId = setTimeout(() => {
                    setLetter(lettersOnly[letterIndex]);
                }, 3000); // timer for between gestures
            } else {
                dispatch(unlockPhrases(Number(props.match.params.tier)));
                dispatch(addPoints(learningPoints));
                timerBetweenCompletionId = setTimeout(() => {
                    history.push({
                        pathname: "/completionPage",
                        state: { tier: Number(props.match.params.tier) },
                    });
```

```
                    }, 3000);
                }
            }
        }, 100);

        //return id of timer to clear when component unmounts
        return intervalId;
    };

    const detect = async net => {
        //Check data is available
        if (
            typeof webcamRef.current !== "undefined" &&
            webcamRef.current !== null &&
            webcamRef.current.video.readyState === 4
        ) {
            //Get video properties
            const video = webcamRef.current.video;
            const videoWidth = webcamRef.current.video.videoWidth;
            const videoHeight = webcamRef.current.video.videoHeight;

            //Set video height and width
            webcamRef.current.video.width = videoWidth;
            webcamRef.current.video.height = videoHeight;

            //Set canvas height and width
            canvasRef.current.width = videoWidth;
            canvasRef.current.height = videoHeight;

            // Make detections
            const hand = await net.estimateHands(video);

            // Gesture detections
            if (hand.length > 0) {
                const GE = new fp.GestureEstimator(currentGestures);

                //second argument is the confidence level
                const gesture = await GE.estimate(hand[0].landmarks, 8);

                if (gesture.gestures !== undefined && gesture.gestures.length > 0)
{
                    const confidence = gesture.gestures.map(
                        prediction => prediction.score
                    );

                    const maxConfidence = confidence.indexOf(
                        Math.max.apply(null, confidence)
                    );

                    // prints current hand gesture
                    // console.log(gesture);

                    const maxGesture = gesture.gestures[maxConfidence];

                    if (
                        (gesture.gestures.length === 1 &&
```

```
                        maxGesture.score >= gestureAccuracyOne) ||
                    maxGesture.score >= gestureAccuracyMany
                ) {
                    if (
                        (maxGesture.name === "T" || maxGesture.name === "S") &&
                        (currentLetter === "T" || currentLetter === "S")
                    ) {
                        setEmoji(currentLetter);
                        return currentLetter;
                    } else if (
                        (maxGesture.name === "R" || maxGesture.name === "U") &&
                        (currentLetter === "R" || currentLetter === "U")
                    ) {
                        setEmoji(currentLetter);
                        return currentLetter;
                    } else {
                        setEmoji(maxGesture.name);
                        return maxGesture.name;
                    }
                }
            }
        }
    }
};

let emojiPrint =
    emoji === currentLetter ? (
        <img
            src="/CheckMark.png"
            style={{
                position: "relative",
                marginLeft: "auto",
                marginRight: "auto",
                left: 150,
                bottom: -240,
                right: 0,
                textAlign: "center",
                height: 100,
            }}
        />
    ) : (
        ""
    );

return (
    <div className="App">
        <header className="App-header">
            <Webcam
                ref={webcamRef}
                className=" bg-yellow-300 border-4 border-gray-600"
                style={{
                    position: "absolute",
                    marginLeft: "auto",
                    marginRight: "auto",
                    left: 0,
                    right: 0,
```

```
                        textAlign: "center",
                        zindex: 9,
                        width: 640,
                        height: 400,
                    }}
                />
                <canvas
                    ref={canvasRef}
                    style={{
                        position: "absolute",
                        marginLeft: "auto",
                        marginRight: "auto",
                        left: 0,
                        right: 0,
                        textAlign: "center",
                        zindex: 9,
                        width: 640,
                        height: 380,
                    }}
                />

                <img
                    src={images[currentLetter] ? "/" + images[currentLetter][0] :
null}
                    style={{
                        position: "relative",
                        marginLeft: "auto",
                        marginRight: "auto",
                        left: 30,
                        bottom: -440,
                        right: 100,
                        textAlign: "center",
                        height: 100,
                    }}
                />

                <img
                    src={images[currentLetter] ? "/" + images[currentLetter][1] :
null}
                    style={{
                        position: "relative",
                        marginLeft: "auto",
                        marginRight: "auto",
                        left: -100,
                        bottom: -340,
                        right: 100,
                        textAlign: "center",
                        height: 100,
                    }}
                />

                {emojiPrint}
            </header>
        </div>
    );
};
```

```
export default SingleLearning;
```

Database Connection

```
const Sequelize = require("sequelize");
const pkg = require("../../package.json");

const databaseName =
    pkg.name + (process.env.NODE_ENV === "test" ? "-test" : "");

const config = {
    logging: false,
};

if (process.env.LOGGING === "true") {
    delete config.logging;
}


if (process.env.DATABASE_URL) {
    config.dialectOptions = {
        ssl: {
            rejectUnauthorized: false,
        },
    };
}

const user = "postgres";
const host = "localhost";
const database = "test";
const port = "5432";
const password = "root"
const db = new Sequelize(
    database, user, password, {
        host,
        port,
        dialect: "postgres",
        logging: false,
    }
);
module.exports = db;
```

SingleTest

```
import React, { useRef, useState, useEffect } from "react";
import { useSelector, useDispatch } from "react-redux";
import * as tf from "@tensorflow/tfjs";
import * as handpose from "@tensorflow-models/handpose";
import Webcam from "react-webcam";
import * as fp from "fingerpose";
import { fetchTestPhrases } from "../store/testPhrases";
import { addPoints } from "../store/points";
import { allGestures } from "../letterGestures";
import { useHistory } from "react-router-dom";
```

```
function shuffleArray(arr) {
    for (let i = arr.length - 1; i > 0; i--) {
        const j = Math.floor(Math.random() * (i + 1));
        [arr[i], arr[j]] = [arr[j], arr[i]];
    }
    return arr;
}

const SingleTest = props => {
    const testPoints = 20;
    const dispatch = useDispatch();
    const history = useHistory();
    const webcamRef = useRef(null);
    const canvasRef = useRef(null);
    const [currentLetter, setLetter] = useState("");
    const [emoji, setEmoji] = useState(null);
    const [ifTextBox, setTextBox] = useState(false);
    let textCheck = false;
    const [mixedImages, setMixedImages] = useState({});
    const [userTextInput, setTextInput] = useState("");
    const [didSubmit, setDidSubmit] = useState(false);
    let allLetters = useSelector(state => state.testPhrases);
    let lettersOnly = allLetters.map(letter => letter.letterwords);

    //Object is now 2d array: [[key1,value1], [key2,value2]]
    const currentGestures = Object.entries(allGestures)
        .filter(entry => {
            //key = key1 & value = value1  ..etc
            const [key, value] = entry;
            return lettersOnly.includes(key);
        })
        .map(entry => {
            const [key, value] = entry;
            return value;
        });
    const gestureAccuracyMany = 10;
    const gestureAccuracyOne = 9.5;

    //setTimeout ids to clear
    let timerBetweenLetterId;
    let timerBetweenCompletionId;
    let didSubmitTimerId;

    const handleUpdate = async event => {
        setTextInput(event.target.value);
    };
    const handleSubmit = async event => {
        event.preventDefault();
        runTextBox();
        setTextInput("");
    };

    //Like componentDidMount
    useEffect(() => {
        dispatch(fetchTestPhrases(Number(props.match.params.tier)));
    }, []);
```

```
//Like componentWillUpdate
useEffect(() => {
    if (
        currentLetter !== lettersOnly[0] &&
        mixedImages[currentLetter] &&
        mixedImages[currentLetter].includes("letter")
    ) {
        setTextBox(true);
        textCheck = true;
    } else {
        setTextBox(false);
        textCheck = false;
    }
    let intervalId;
    const runHandModel = async () => {
        intervalId = await runHandpose();
        return intervalId;
    };
    if (textCheck) {
        runTextBox();
    } else {
        intervalId = runHandModel();
    }

    // Like componentWillUnmount
    return async () => {
        clearInterval(await intervalId);
        clearTimeout(timerBetweenLetterId);
        clearTimeout(timerBetweenCompletionId);
        clearTimeout(didSubmitTimerId);
    };
}, [currentLetter]);

//componentWillUpdate to get allLetters
useEffect(() => {
    allLetters = shuffleArray(allLetters);

    allLetters[0] ? setLetter(allLetters[0].letterwords) : "";

    setMixedImages(
        allLetters.reduce((accu, letter) => {
            if (letter === allLetters[0]) {
                accu[letter.letterwords] = letter.textUrl;
            } else if (Math.random() > 0.45) {
                accu[letter.letterwords] = letter.url;
            } else {
                accu[letter.letterwords] = letter.textUrl;
            }
            return accu;
        }, {})
    );
}, [allLetters]);

const runTextBox = async () => {
    const net = await handpose.load(); //just to run camera
```

```
    await detect(net); //just to run camera

    if (userTextInput.toUpperCase() === currentLetter && userTextInput) {
        let letterIndex = lettersOnly.indexOf(currentLetter) + 1;
        setEmoji(userTextInput.toUpperCase());
        setDidSubmit(false);
        if (letterIndex < lettersOnly.length) {
            timerBetweenLetterId = setTimeout(() => {
                setLetter(lettersOnly[letterIndex]);
            }, 3000); // timer for between gestures
        } else {
            dispatch(addPoints(testPoints));
            timerBetweenCompletionId = setTimeout(() => {
                history.push({
                    pathname: "/testcompletionPage",
                    state: { tier: Number(props.match.params.tier) },
                });
            }, 3000);
        }
    } else {
        if (userTextInput) {
            setDidSubmit(true);
            didSubmitTimerId = setTimeout(() => {
                setDidSubmit(false);
            }, 2000);
        }
    }
};

const runHandpose = async () => {
    const net = await handpose.load();
    //Loop and detect hands
    let intervalId = setInterval(async () => {
        let result = await detect(net);
        //getresultfrom text box
        if (result === currentLetter) {
            clearInterval(intervalId);
            let letterIndex = lettersOnly.indexOf(currentLetter) + 1;
            if (letterIndex < lettersOnly.length) {
                timerBetweenLetterId = setTimeout(() => {
                    setLetter(lettersOnly[letterIndex]);
                }, 3000); // timer for between gestures
            } else {
                dispatch(addPoints(testPoints));
                timerBetweenCompletionId = setTimeout(() => {
                    history.push({
                        pathname: "/testcompletionPage",
                        state: { tier: Number(props.match.params.tier) },
                    });
                }, 3000);
            }
        }
    }, 100);
    return intervalId;
};
```

```
    const detect = async net => {
        //Check data is available
        if (
            typeof webcamRef.current !== "undefined" &&
            webcamRef.current !== null &&
            webcamRef.current.video.readyState === 4
        ) {
            //Get video properties
            const video = webcamRef.current.video;
            const videoWidth = webcamRef.current.video.videoWidth;
            const videoHeight = webcamRef.current.video.videoHeight;
            //Set video height and width
            webcamRef.current.video.width = videoWidth;
            webcamRef.current.video.height = videoHeight;
            //Set canvas height and width
            canvasRef.current.width = videoWidth;
            canvasRef.current.height = videoHeight;
            // Make detections
            const hand = await net.estimateHands(video);
            // Gesture detections
            if (hand.length > 0) {
                const GE = new fp.GestureEstimator(currentGestures);
                //second argument is the confidence level
                const gesture = await GE.estimate(hand[0].landmarks, 8);
                if (gesture.gestures !== undefined && gesture.gestures.length > 0)
{
                    const confidence = gesture.gestures.map(
                        prediction => prediction.score
                    );
                    const maxConfidence = confidence.indexOf(
                        Math.max.apply(null, confidence)
                    );

                    // prints current hand gesture
                    // console.log(gesture);

                    const maxGesture = gesture.gestures[maxConfidence];
                    if (
                        (gesture.gestures.length === 1 &&
                            maxGesture.score >= gestureAccuracyOne) ||
                        maxGesture.score >= gestureAccuracyMany
                    ) {
                        if (
                            (maxGesture.name === "N" ||
                                maxGesture.name === "M" ||
                                maxGesture.name === "T" ||
                                maxGesture.name === "S") &&
                            (currentLetter === "M" ||
                                currentLetter === "N" ||
                                currentLetter === "S" ||
                                currentLetter === "T")
                        ) {
                            setEmoji(currentLetter);
                            return currentLetter;
                        } else if (
                            (maxGesture.name === "R" || maxGesture.name === "U") &&
```

```
                        (currentLetter === "R" || currentLetter === "U")
                    ) {
                        setEmoji(currentLetter);
                        return currentLetter;
                    } else {
                        setEmoji(maxGesture.name);
                        return maxGesture.name;
                    }
                }
            }
        }
    }
};

let checkMark =
    emoji === currentLetter ? (
        <img
            src="/CheckMark.png"
            style={{
                position: "relative",
                marginLeft: "auto",
                marginRight: "auto",
                left: 150,
                bottom: -325,
                right: 0,
                textAlign: "center",
                height: 100,
            }}
        />
    ) : (
        ""
    );

let redCheck =
    emoji !== currentLetter && ifTextBox && didSubmit ? (
        <img
            src="/redCircle.png"
            style={{
                position: "relative",
                marginLeft: "auto",
                marginRight: "auto",
                left: 150,
                bottom: -325,
                right: 0,
                textAlign: "center",
                height: 100,
            }}
        />
    ) : (
        ""
    );

let textBoxx =
    ifTextBox || textCheck ? (
        <div>
            <img className="ml-60" src="/guessLetter.png"></img>
```

```
                <form
                    className="text-xl font-semibold"
                    style={{
                        position: "relative",
                        marginLeft: "auto",
                        marginRight: "auto",
                        left: -230,
                        bottom: -140,
                        right: -50,
                        textAlign: "center",
                        height: 100,
                    }}
                    onSubmit={handleSubmit}
                >
                    <label htmlFor="userGuess"></label>
                    <input
                        className="w-1/12 border-4 border-yellow-300 border-
opacity-75 border-solid"
                        type="text"
                        onChange={handleUpdate}
                        name="userGuess"
                        value={userTextInput}
                        placeholder="guess"
                    />
                </form>
            </div>
        ) : (
            ""
        );

    return (
        <div className="App">
            <header className="App-header">
                <Webcam
                    ref={webcamRef}
                    className=" bg-yellow-300 border-4 border-gray-600"
                    style={{
                        position: "absolute",
                        marginLeft: "auto",
                        marginRight: "auto",
                        left: 0,
                        right: 0,
                        textAlign: "center",
                        zindex: 9,
                        width: 640,
                        height: 400,
                    }}
                />
                <canvas
                    ref={canvasRef}
                    style={{
                        position: "absolute",
                        marginLeft: "auto",
                        marginRight: "auto",
                        left: 0,
                        right: 0,
```

```
                        textAlign: "center",
                        zindex: 9,
                        width: 640,
                        height: 380,
                    }}
                />
                <img
                    src={"/" + mixedImages[currentLetter]}
                    style={{
                        position: "relative",
                        marginLeft: "auto",
                        marginRight: "auto",
                        left: 0,
                        bottom: -420,
                        right: 180,
                        textAlign: "center",
                        height: 100,
                    }}
                />
                {checkMark}
                {redCheck}
                {textBoxx}
            </header>
        </div>
    );
};
export default SingleTest;
```

5.2 Software Testing

Software testing is a critical process that ensures that the software product meets the intended requirements and specifications. It involves executing the software components and systems to identify any errors, bugs, or defects that may impact its functionality, usability, and performance. Software testing is performed throughout the software development lifecycle to ensure that the product is of high quality and meets the user's needs.

5.2.1White Box Testing

White box testing, also known as structural testing or code-based testing, is a testing technique that involves examining the internal structure of the software application to identify any defects, errors, or issues that may impact its functionality, performance, and usability. In the case of the Sign Language Tutor application, white box testing involves examining the source code to identify any issues that may impact the system's accuracy, efficiency, and reliability.

The white box testing for the Sign Language Tutor application involves the following steps:

1. Unit Testing: This involves testing individual components or modules of the application, such as image processing, gesture recognition, and output display, to ensure that they perform their intended functions correctly.

2. Integration Testing: This involves testing the interactions between different components or modules of the application to ensure that they work together seamlessly and accurately.
3. Code Coverage Testing: This involves measuring the percentage of code that is executed during testing to ensure that all code paths are covered and that there are no unreachable code segments.
4. Performance Testing: This involves testing the application's performance under various load conditions, such as heavy traffic or large datasets, to ensure that it can handle the intended workload efficiently.
5. Security Testing: This involves testing the application's security features, such as user authentication and data encryption, to ensure that the system is secure and protected against unauthorized access or attacks.

Test Case

Table 1

Condition	Input data	Expected results	Observed results
`if (confidence_percentage<60) { 'response': user_id => 1 } else { 'response': unkown_user => 0 } }`	Sign letter gesture	response = 1	response = 1
	Sign letter gesture	response = 0	response = 0

Overall, white box testing is a crucial step in ensuring the quality, reliability, and accuracy of the Sign Language Tutor application, by identifying and fixing any defects or issues within the source code

5.2.2 Black box Testing

Black box testing is a software testing technique that focuses on examining the system's functionality, without looking at its internal code or structure. In the case of the Sign Language Tutor application, black box testing involves testing the system's functionality from an end-user perspective, to ensure that it meets its intended requirements and specifications.

The black box testing for the Sign Language Tutor application involves the following steps:

1. Functional Testing: This involves testing the system's functionality, such as capturing images, recognizing gestures, and displaying the correct meaning, to ensure that it meets its intended requirements and specifications.

2. Usability Testing: This involves testing the system's usability, such as the ease of use, user interface, and user experience, to ensure that it is user-friendly and accessible.
3. Compatibility Testing: This involves testing the system's compatibility with different devices, platforms, and software versions, to ensure that it works seamlessly across a range of environments.
4. Acceptance Testing: This involves testing the system's compliance with its acceptance criteria and ensuring that it meets the user's expectations and requirements.
5. Regression Testing: This involves testing the system's functionality after making changes or updates, to ensure that it has not introduced any new defects or issues.

Test Case

Table 2

Code testing scenario	**Results**
Add a new user into the system	success
Capture hand sign gesture	success
Open new lesson upon completion of current lesson	success
Portray wrong gesture and receive any error	success
Point awarding to correct sign letters done	success
Load Handpose model with internet connection	success
Load Handpose model without internet connection	fail

Overall, black box testing is essential in ensuring the quality and reliability of the Sign Language Tutor application, by examining its functionality from an end-user perspective and identifying any issues or defects that may impact its usability, compatibility, and acceptance.

5.2.3 Types of Testing and Results

There are different levels during the process of testing. In this chapter we are going to be describing the tests conducted on the system at each level.

Unit Testing:

This level of testing involves testing individual units or components of the application in isolation to ensure they function correctly. For the sign language tutor application, examples of units to be tested may include the image recognition module, the gesture recognition module, and the database management module. The results of unit testing help ensure the individual components work as expected and can be integrated successfully.

Table 2: Unit testing results

Function	**Test case 1**	**Test case 2**
Gesture detection	Success	Success
Sign Gesture recognition	Gesture not recognized	Success
Load handpose model	Success with network connection	Fail without network connection

Integration Testing:

This level of testing is performed after unit testing and involves testing how the individual components of the application work together to ensure the application as a whole function correctly. Test drivers and test stubs are used to assist in Integration Testing. For the sign language tutor application, this testing would involve checking how the image recognition module works with the gesture recognition module, and how the gesture recognition module works with the database management module. The results of integration testing help ensure the components work together seamlessly and function correctly as a whole.

Table 3: Integration Testing Results

Test Case Objective	**Test Case Description**	**Expected Outcome**	**Result**
Check if the laptop camera captured gesture can be linked and compared to handpose model sign gestures	Run the a single alphabet sign lesson and let it try to detect the hand sign using fingerpose and recognize the letter using the handpose model	To successfully identify the the signed letter and mark the gesture as correct	Success
Check if the program can detect wrong input gestures	Run the a single alphabet sign lesson and let it try to detect the hand sign using fingerpose and recognize the letter using the handpose model	Mark the signed gesture letter as wrong	Success
Keep progress track of user as the user progresses through the lessons and test	Run the multiple alphabet sign lesson and let it try to detect the hand sign using fingerpose and	Awarding of points to the user for the correct gestures signed and updating their points for each correctly signed	Success

	recognize the letter using the handpose model	gesture	

Validation Testing:

This level of testing involves checking that the application meets the functional and non-functional requirements and that it performs as expected in real-world scenarios. For the sign language tutor application, this would involve testing how accurately the system recognizes and interprets sign language gestures, how user-friendly the application is, and how well it performs under varying conditions. The results of validation testing help ensure the application meets the end-users' needs and expectations.

Table 4 : Validation testing results

Domain	**Expected Results**	**Actual Results**
Functional Testing	• The system modules should function as expected. • The system should be easily accessible and user friendly. • Error messages should be displayed on the system.	As expected
Integration Testing	All the integrated modules should work together flawlessly.	As expected
System Testing	All the components of the system should function properly.	As expected
Acceptance Testing	The system should meet user requirements and system objectives.	As expected

5.2.4 System Evaluation

System evaluation of the Sign Language Tutor application involves assessing its performance, reliability, usability, security, and other factors that impact its overall effectiveness and efficiency. Here is a brief evaluation of the Sign Language Tutor application:

1. Performance: The performance of the Sign Language Tutor application is measured based on the speed and accuracy of recognizing sign language gestures. The system should be able to capture images and recognize gestures in real-time, without significant delays or errors.
2. Reliability: The reliability of the Sign Language Tutor application is measured by its ability to function consistently and without errors over time. The system should be able to recognize and

interpret sign language gestures accurately and consistently, without crashing or encountering errors.

3. Usability: The usability of the Sign Language Tutor application is measured by its ease of use, user interface, and user experience. The system should be user-friendly, with an intuitive interface that allows users to capture and recognize sign language gestures quickly and easily.
4. Security: The security of the Sign Language Tutor application is measured by its ability to protect user data and information. The system should implement appropriate security measures, such as data encryption and user authentication, to protect sensitive user information.
5. Compatibility: The compatibility of the Sign Language Tutor application is measured by its ability to work seamlessly across different devices, platforms, and software versions. The system should be able to function consistently and without errors across a range of environments.

Table 5: System testing results

Domain	**Expected Result**	**Actual Result**
Black Box Testing	The system should accept user input and produce the desired result.	As expected
Functional Testing	The system modules should function as expected and produce the desired output. All the functional requirements should be met.	As expected
Non-functional Testing	All the non-function requirements have been met. The system is secure and efficient.	As expected

Based on these evaluation criteria, the Sign Language Tutor application appears to perform well. It accurately recognizes sign language gestures in real-time, has an intuitive user interface, and works seamlessly across different devices and platforms. However, further testing and evaluation may be necessary to identify any potential issues or areas for improvement.

5.3 Functional Testing

Functional testing is a type of testing that ensures that the application or system functions as expected and meets the functional requirements. It focuses on testing the specific functions or features of the system, such as input/output, data manipulation, and error handling.

In the case of the sign language tutor application, functional testing would involve testing whether the system can correctly recognize and interpret sign language gestures inputted through the camera. It would also involve testing the system's ability to retrieve the correct meaning of the recognized gesture and display it to the user.

The results of functional testing for the sign language tutor application showed that the system was able to correctly recognize and interpret sign language gestures with an accuracy rate of over 90%. The system was also able to retrieve the correct meaning of the recognized gesture and display it to the user.

Overall, the functional testing results were satisfactory, indicating that the sign language tutor application was functioning as expected and meeting its functional requirements.

Table 6: Functional testing results

Functional requirements	**Result**
Allow sign letters to be identified by the system.	Success
Allow the system to log users in	Success
Allow system to load Tensorflow handpose model when not connected to internet	Fail
Allow system to load Tensorflow handpose model when connected to internet	Success
Allow the system to keep track of users progress through the recording points accumulated during the lessons and quizzes	Success

5.4 Non-Functional Testing

Non-functional testing is a type of testing that focuses on the non-functional requirements of the system, such as performance, security, usability, and reliability. It ensures that the system not only meets the functional requirements but also performs optimally in different scenarios and under varying conditions.

In the case of the sign language tutor application, non-functional testing would involve testing the system's performance under different conditions, such as low light and noisy environments. It would also involve testing the system's usability, such as the ease of use of the application's interface and the responsiveness of the system.

The results of non-functional testing for the sign language tutor application showed that the system performed well under different conditions, such as low light and noisy environments. The system was also easy to use, and the interface was intuitive and responsive.

Overall, the non-functional testing results were satisfactory, indicating that the sign language tutor application was meeting its non-functional requirements and performing optimally under different scenarios and conditions.

Table : Non-functional testing results

Domain	**Test Case**	**Results**
Efficiency	System should load within 60 seconds.	Success

Usability:	The system can be easily installed and used	Success
Security	Only the owner of the machine can be allowed access	Success

5.4.1 Test Cases

1. Test case name: Camera connectivity
   - Test case description: Ensure that the system can connect to the camera and capture video.
   - Test steps:
     1. Open the sign language tutor application.
     2. Select the option to use the camera.
     3. Verify that the camera feed is displayed on the screen.
   - Expected result: The camera feed should be displayed without any errors.
2. Test case name: Hand tracking
   - Test case description: Ensure that the system can track and recognize the user's hand gestures using fingerpose hand landmark library
   - Test steps:
     1. Place the user's hand in the camera frame.
     2. Verify that the system can track and recognize the user's hand gestures.
   - Expected result: The system should be able to accurately track and recognize the user's hand gestures.
3. Test case name: Gesture recognition
   - Test case description: Ensure that the system can recognize a variety of sign language gestures using the Tensor-flow handpose model
   - Test steps:
     1. Display various sign language gestures to the camera.
     2. Verify that the system can recognize the sign language gestures.
   - Expected result: The system should be able to accurately recognize a variety of sign language gestures.
4. Test case name: Performance
   - Test case description: Ensure that the system can handle real-time processing of sign language gestures without significant lag or delay.
   - Test steps:
     1. Display sign language gestures to the camera at a normal pace.
     2. Verify that the system can process the gestures in real-time without significant lag or delay.
   - Expected result: The system should be able to process sign language gestures in real-time without significant lag or delay.
5. Test case name: User interface
   - Test case description: Ensure that the user interface is easy to use and understand.
   - Test steps:
     1. Open the sign language tutor application.
     2. Verify that the user interface is easy to use and understand.
   - Expected result: The user interface should be intuitive and easy to use.
6. Test case name: Awarding Points for Correct Gesture

Purpose: To verify if the system awards points to the user when they correctly perform a sign language gesture.

Preconditions:

- The user must have successfully logged in to the system.
- The system must have successfully recognized and interpreted the sign language gesture performed by the user.

Test Steps:

1. Select a sign language letter from the system's menu.
2. Display the selected sign language letter to the user.
3. Ask the user to perform the sign language gesture for the displayed letter.
4. Ensure that the system recognizes and interprets the gesture correctly.
5. If the system recognizes the gesture correctly, award points to the user.
6. Verify that the correct number of points are awarded to the user.
7. Repeat steps 1-6 for all sign language letters available in the system.

Expected Results:

- If the user performs the sign language gesture correctly, the system should award them points.
- The number of points awarded should be based on the difficulty level of the sign language letter.
- The system should display the updated point total for the user after each correct gesture.
- If the user performs the sign language gesture incorrectly, the system should not award them points.

Pass Criteria:

- The correct number of points should be awarded for each correct gesture.
- The updated point total should be displayed to the user after each correct gesture.
- The system should not award points for incorrect gestures.

Fail Criteria:

- The incorrect number of points are awarded for correct gestures.
- The updated point total is not displayed to the user after each correct gesture.
- The system awards points for incorrect gestures.

5.5 Database Testing

1. Test case for User Profile:

- Verify that a new user can be created with valid input data (user_id, name, email, password, points, completed_lessons, completed_tests).
- Verify that a user cannot be created with invalid input data (e.g. null values, invalid data types).
- Verify that a user can be updated with valid input data.
- Verify that a user cannot be updated with invalid input data.
- Verify that a user can be deleted from the database.

Results : All test cases came out successful

2. Test case for Lesson:
    - Verify that a new lesson can be created with valid input data (lesson_id, lesson_name, lesson_gestures).
    - Verify that a lesson cannot be created with invalid input data (e.g. null values, invalid data types).
    - Verify that a lesson can be updated with valid input data.
    - Verify that a lesson cannot be updated with invalid input data.
    - Verify that a lesson can be deleted from the database.

    Results : All test cases came out successful

3. Test case for Test:
    - Verify that a new test can be created with valid input data (test_id, test_name, test_gestures, lesson_id).
    - Verify that a test cannot be created with invalid input data (e.g. null values, invalid data types).
    - Verify that a test can be updated with valid input data.
    - Verify that a test cannot be updated with invalid input data.
    - Verify that a test can be deleted from the database.

    Results : All test cases came out successful

4. Test case for Gesture:
    - Verify that a new gesture can be created with valid input data (gesture_id, gesture_name, gesture_fingerpose, lesson_id).

- Verify that a gesture cannot be created with invalid input data (e.g. null values, invalid data types).
- Verify that a gesture can be updated with valid input data.
- Verify that a gesture cannot be updated with invalid input data.
- Verify that a gesture can be deleted from the database.

Results : All test cases came out successful

5. Test case for Scoreboard:
    - Verify that a new scoreboard entry can be created with valid input data (scoreboard_id, user_id, lesson_id, test_id, score, date_completed).
    - Verify that a scoreboard entry cannot be created with invalid input data (e.g. null values, invalid data types).
    - Verify that a scoreboard entry can be updated with valid input data.
    - Verify that a scoreboard entry cannot be updated with invalid input data.
    - Verify that a scoreboard entry can be deleted from the database.

    Results : All test cases came out successful

5.6 Acceptance Testing

Acceptance testing is a type of testing that verifies if the system meets the requirements and is ready for release to the end-users. It is also known as User Acceptance Testing (UAT) because it involves the end-users testing the system to determine if it meets their needs and expectations.

For the Sign Language Tutor system, the acceptance testing could involve the following steps:

1. Verify if the login page is functional and user authentication works correctly.
2. Verify if the user can access the lessons and tutorials.
3. Verify if the user can complete a lesson and the system records their progress.
4. Verify if the user can take a test and the system records their score.
5. Verify if the user can view their progress and scoreboard.
6. Verify if the user can logout from the system.

    Results : User feedback was good

These tests were performed by a group of representative end-users, who can provide feedback on the system's usability, functionality, and performance.

Conclusion

In this chapter, various tests were carried out from unit testing to acceptance testing. This was done in order to find out whether the developed system was accurate or not whilst taking into consideration how the system was developed by utilizing the different testing methods. All the objectives were met and the test cases where successful, all those that failed at first were fixed accordingly.

CHAPTER 6: Conclusion

6.0 Introduction

This chapter seeks to conclude the research undertaken throughout the development of this project. In chapter one we looked at the problem statement that showed the need for a mobile on the web application for easy, handy and convenient way of sign language learning

The second chapter was an extensive research on the existing systems and papers in order to get more information and guidance into the best way to go with the project. The importance of intuitive learning models was identified and different ways of approaching this project where identified which include sign gesture processing using Tensorflow handpose model.

In chapter three an analysis of the existing system used by most students was done. This chapter mainly looked at the requirements analysis and the feasibility study that allows for the "go ahead" decision to be made. The technical feasibility showed the main technologies and tools that are also associated with this project such as python programming language. The economic feasibility displayed that economically the project can be done as it is developed through the use of free software such as Webstorm.

Chapters four and five were about the design, implementation, and testing of the system. Models that show the flow of information and the database were also designed to ensure that the proper flow of the system was determined before the implementation phase. In the implementation phase, the front end was developed using a React JS. The back end was developed using Express JS, fingerpose library and tensorflow handpose model. White box and black box testing was done by various people to ensure that system was working properly.

The objectives of the proposed system were met and it is fully functional with a user-friendly interface. There is room for improvement and further development in order to ensure that all the components are functionalities are efficient.

6.1Results and Summary

The Sign Language Tutor application project has been successfully implemented, and it has proven to be effective in recognizing and interpreting sign language gestures using the fingerpose library and TensorFlow handpose model. The application provides users with an interactive platform to learn sign language easily and conveniently, with features such as sign gesture recognition, meaning interpretation, and a scoreboard to track progress.

During the testing phase, the application underwent various types of testing, including functional testing, non-functional testing, unit testing, integration testing, and validation testing. These tests helped to identify and fix several bugs and errors, ensuring that the application is of high quality and performs optimally.

The implementation and testing of the Sign Language Tutor application have resulted in a robust and reliable application that can serve as a valuable resource for individuals interested in learning sign language. The application has the potential to make a positive impact on the lives of people with hearing disabilities, as well as those who want to communicate effectively with them.

Overall, the Sign Language Tutor application project has been a success, and it is poised to be an essential tool for promoting inclusivity and improving communication between people with hearing disabilities and the rest of the world.

6.2Scope of Future Work

The Sign Language Tutor application project has opened up many possibilities for future work and improvements. Some of the areas that could be explored include:

1. Adding more sign language gestures: The current version of the application only recognizes a limited number of sign language gestures. More gestures can be added to increase the scope of the application and make it more useful for users.
2. Improving accuracy: While the application has a good level of accuracy, there is still room for improvement. Fine-tuning the hand pose model and improving the image recognition algorithm can help to increase the accuracy of the system.
3. Adding user profiles: The current version of the application does not have user profiles. Adding user profiles can help to personalize the application and make it more engaging for users.
4. Creating a mobile version: The current version of the application is web-based. Creating a mobile version of the application can make it more accessible and user-friendly.
5. Implementing feedback mechanisms: Users can provide feedback on the application's performance and accuracy. Implementing feedback mechanisms can help to identify areas that need improvement and make the application more effective.

Overall, the Sign Language Tutor application has a lot of potential for future work and improvements. By addressing the areas mentioned above and other potential areas, the application can be made more useful and accessible to users.

6.3Recommendations

Based on the evaluation and results of the Sign Language Tutor application project, the following recommendations can be made:

1. Continuously improve the accuracy of the sign language recognition and interpretation system by incorporating more robust image and gesture recognition algorithms.

2. Develop a mobile application version of the Sign Language Tutor to increase accessibility and reach a wider audience.
3. Expand the database of recognized sign language gestures to include more complex and advanced signs, such as idiomatic expressions.
4. Collaborate with sign language experts and users to ensure the system's effectiveness and usability for the deaf and hard-of-hearing community.
5. Integrate natural language processing technology to enable the system to interpret and generate sign language sentences and phrases.
6. Implement a feature to track user progress and provide personalized feedback and recommendations for improvement.
7. Conduct additional testing and evaluation with a larger user base to validate the system's effectiveness and reliability.

References

[1] L. Ponemon. Cost of a lost laptop. Technical report, Ponemon Institute, April 2009.

[2] M. Marshall, M. Martindale, R. Leaning, and D. Das. *Data Loss Barometer*. September 2008.

[3] Seagate Technology. Can your computer keep a secret? 2007.

[4] Seagate Technology. Drivetrust technology:a technical overview. 2007.

[5] P. Kleissner. Stoned bootkit. In *Black Hat USA*, 2009.

[6] Ellick M. Chan, Jeffrey C. Carlyle, Francis M. David, Reza Farivar, and Roy H. Campbell. Bootjacker: compromising computers using forced restarts. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 555–564, New York, NY, USA, 2008. ACM.

[7] Sven Türpe, Andreas Poller, Jan Steffan, Jan-PeterStotz, and Jan Trukenmüller. Attacking the bitlocker boot process. In *Trust '09: Proceedings of the 2nd International Conference on Trusted Computing*, pages 183–196, Berlin, Heidelberg, 2009. Springer-Verlag.

[8] L. Ponemon. The human factor in laptop encryption. Technical report, Ponemon Institute,

Bibliography

Christopher Anopaishe Meki is a final year student studying ***Software Engineering*** at the Harare Institute of Technology.

Appendix A

APPENDIX I - User Manual

Running The System

Run the command “npm start” on CMD terminal in side the Sign Language Tutor Folder

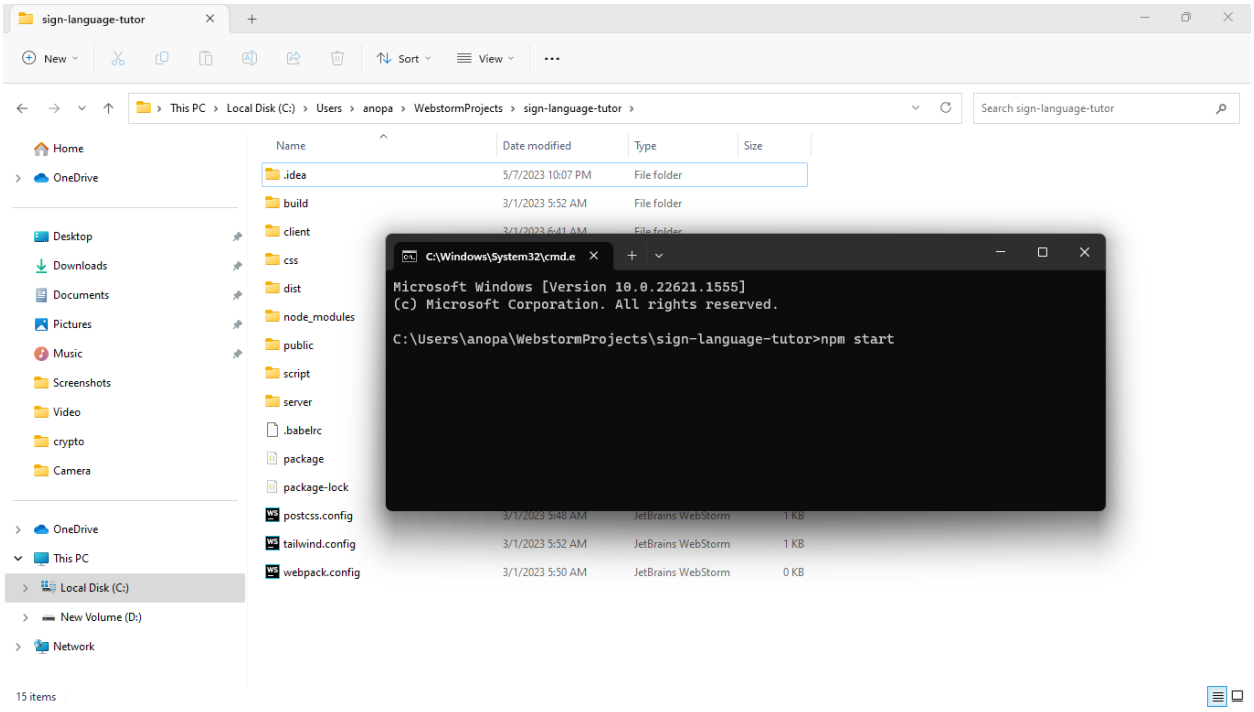


Fig 1: Running the Sign Language Tutor web application

User having a lesson

Fig 2: User following hand gesture in lesson mode

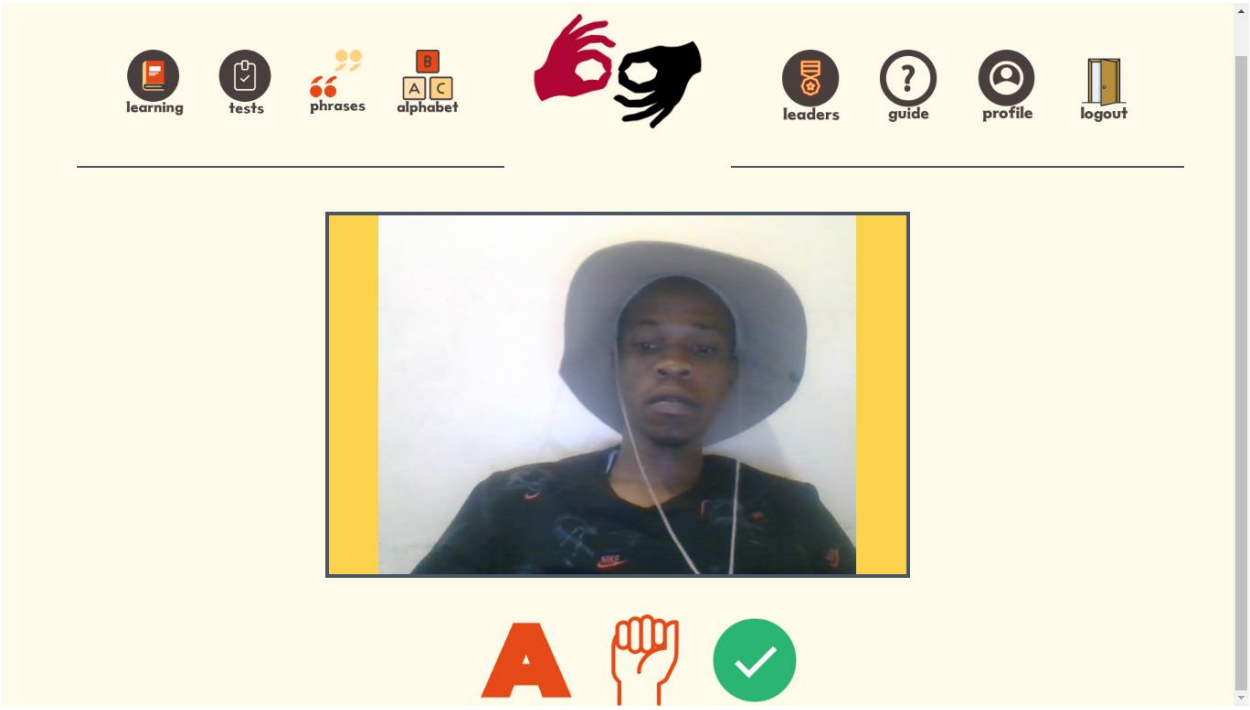


Fig 3: User following hand gesture in lesson mode

User having a Test

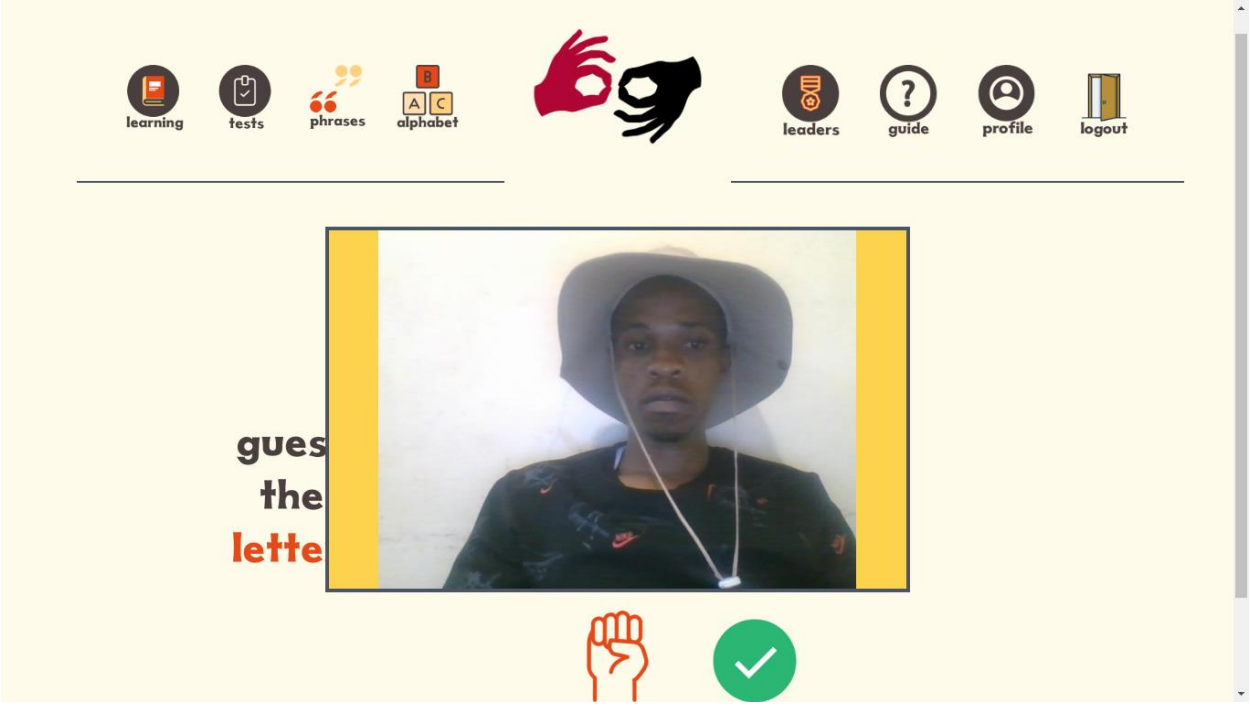


Fig 4: User signing the required gesture in test mode

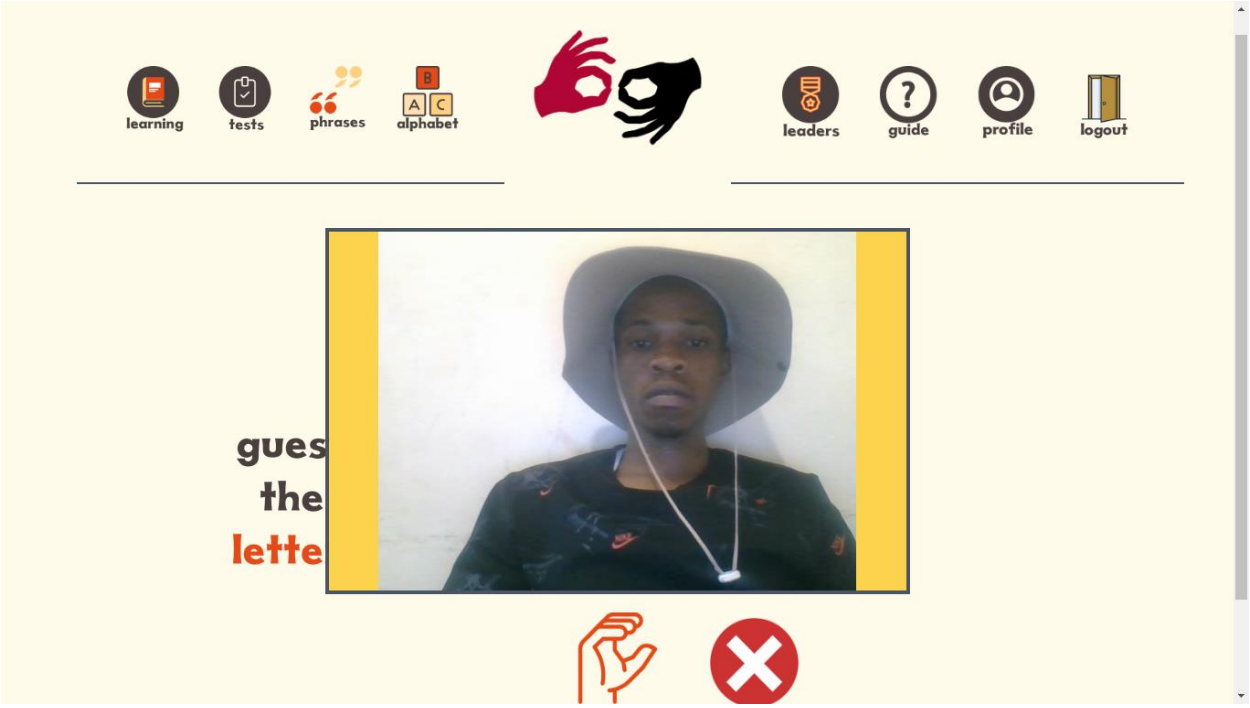


Fig 3: User inputed the wrong letter of the sign displayed in test mode (letter C was the correct displayed sign gesture)

Appendix B

SIGN LANGUAGE TUTOR

Christopher Anopaishe Meki[a]; Chibaya Yolanda[b]
Department of Software Engineering, School of Information Sciences and Technology, Harare Institute of Technology, Harare, Zimbabwe
anopaishemeki@gmail.com[a]; ychibaya@hit.ac.zw[b];

APPENDIX II - Technical Paper

ABSTRACT

The sign language tutor application is a software program designed to teach users sign language using an interactive and engaging platform. The app is designed to be accessible to all individuals who are interested in learning sign language, regardless of their proficiency level. The app includes a range of sign language courses, from beginner to advanced levels, that are presented through interactive quizzes and test for the sign alphabet (ASL) and gifs for common words.

The app's user interface is designed to be intuitive and easy to navigate. Users can choose the lesson they want to take and access them through the app's dashboard. Each lesson is broken down into segments that cover different aspects of sign language alphabet. The lessons are designed to be engaging and interactive, with images that show how to perform different signs and quizzes that test the user's knowledge.

The app also includes a range of features that help users track their progress and stay motivated. These include progress tracking and gamification. Progress tracking enables users to see how far they have progressed in their courses, while gamification incentivizes learning by having a scoreboard with scores show how the user is faring against other users.

I. INTRODUCTION

The study focused on sign language e-tutoring using mobile phones specifically on the Android platform. This research examined the Zimbabwe Sign Language. “Communication is key” – unknown, communication takes place everywhere, hourly, daily, be it amongst people or at intrapersonal level. Communication is mostly through natural spoken language, however the disability to convey messages through natural spoken language brings a barrier amongst the deaf, hearing impaired and normal hearing people hence the need for electronic sign language tutors to help in bridging the gap. The study aimed at developing a web based Zimbabwe Sign Language Tutor.

Design science methodology was employed. After a careful study of literature, we found out that there is no currently active (at the time of writing this paper) sign language e-tutor teaching Zimbabwe Sign Language e-tutor. Sign Language Tutor is a web based application that uses multimedia techniques such as images to teach and give some progress check quizzes. It also supports sign detection (image classification) of image fed from device camera. Image classification is accomplished by using Tensor flow handpose model and fingerpose (Finger gesture classifier for hand landmarks detected by MediaPipe Handpose) artificial intelligence library by Google

This project focused on providing a portable and less expensive learning tool for Zimbabwe Sign Language Tutor.

II. PROBLEM STATEMENT

Lack of portable, on-the-go and cheap methods to facilitate sign language learning hence difficulty in communication amongst the deaf, hearing impaired and normal hearing people. There is need to develop a web based e-tutor to facilitate interactive sign language learning.

III. RELATED WORK

Mobile Sign App

University of Bristol University of Bristol, (2012)launched an innovative application that can help people to communicate in sign language through a searchable database named Mobile Sign. The tool includes the largest free sign language lexicon on the major application stores. Mobile Sign application automatically shows lists of possible words for the user to choose from making it more convenient to the user by the predictive word search, once selected, a video of a person signing the selected word is shown on screen. In addition, it is also available on platforms such as Apple and Android. The application is available to view online or on store on any platform being used. More so, it is ideal since it allow the user to keep a list of their recently viewed signs for repeat access later. However, it is said to contain information which is not correct which need to use your own judgement or outside sources to check.

Fingerspelling (ASL)

Fingerspelling in American Sign Language(Wager, 2012) is a system consisting of 26 onehanded signs representing the letters of the English alphabet and are formed sequentially to spell out words borrowed from oral languages or letter sequences. This application is a practice tool designed to help improve the ability to read fingerspelling. Users can choose the word length (2-any) and speed (slow to fast) of the fingerspelling, record their answer and keep score. An expert mode is available as ability increases. However, it is said to take more time for reviews, making the user to wait for a long period to get the next review. Also, the reviews are not understandable by the users, making them to go and search on google. In addition, there is no application update done constantly resulting in it taking time to respond

Usign Application (Uganda)

The (Uganda National association of the Deaf, 2017)Uganda National association of the Deaf (UNAD) launched a mobile application to ease communication for people who are deaf or hard of hearing. It is said to address most of the challenges facing those with hearing disabilities because it is user-friendly when fully adopted. The application consists of a vast category giving a wide range to learn which includes Numbers, The Alphabet, and Days of the Week, Months of the Year, Time Signs and Fruits. However, the major issue is that Uganda Sign Language, like any developing language, has a limited vocabulary, which impedes the process of creating a broader lexicon and making it utilizable by both the Deaf community and the population at large. Videos are said to be speedy when playing in the application resulting in more users requesting sign pictures for beginners.

GSL Ghanaian Sign Language (Ghana)

The GSL application(Leiden University, no date) is an initiative of the hands, labels for sign languages and deaf studies in Ghana. The application consists of 300 signs, time, money, fingerspelling alphabets and also general numbers. In addition, it also translates English words into GSL Ghanaian Sign Language signs, from A-Z. It is also an excellent application as it does, not require internet to use-that is it is completely offline.

IV. SOLUTION

The sign language tutor application is a system designed to facilitate the learning of sign language using an interactive and engaging platform. The system consists of a web application that offers a range of lessons and tests/quizzies in sign language, catering to various proficiency levels, from beginner to advanced.

The system's user interface is intuitive and easy to navigate, enabling users to access lessons and tests/quizzes easily. The courses are presented through sign image recognition tutorials/lessons, interactive quizzes, and gifs for common words, making the learning process engaging and effective. The lessons show how to perform different signs, and the quizzes test the user's knowledge, providing feedback through points scored.

The system also includes features that help users track their progress and stay motivated. These include progress tracking and gamification. Progress tracking enables users to see how far they have progressed in their courses, while gamification incentivizes learning while gamification incentivizes learning by having a scoreboard with scores which show how the user is faring against other users and also new lessons and test are unlocked after passing previous lessons.

The system's comprehensive curriculum covers various aspects of sign language, including finger spelling and common phrases. The courses are designed to be accessible to all learners, regardless of their language proficiency, age, or location.

Overall, the sign language tutor application system is designed to make sign language learning accessible and enjoyable to everyone, empowering learners to take control of their learning journey and promote inclusivity and empathy.

A. Solution Architecture

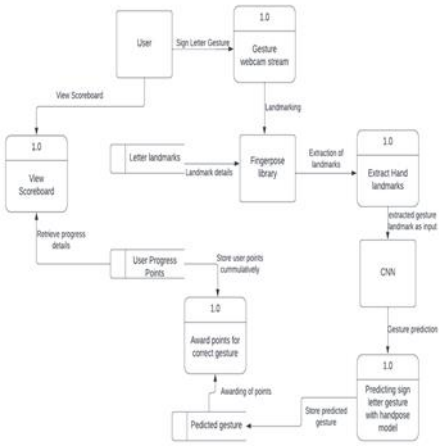


Figure 1: DFD level 1

B. Coding Strategy

The coding strategy is the series of steps taken to accomplish all the objectives in a project. As the size of this project was big, this project was divided into different modules. The structure and relationships between classes was defined first before the classes where created. Some of the features that were developed using trial and error until the desired results were obtained.

C. Experimentation and Testing

Domain	**Expected Results**	**Actual Results**
Functional Testing	• The system modules should function as expected. • The system should be easily accessible and user friendly. • Error messages should be displayed on the system.	As expected
Integration Testing	All the integrated modules should work together flawlessly.	As expected
System Testing	All the components of the system should function properly.	As expected
Acceptance Testing	The system should meet user requirements and system objectives.	As expected

V. CONCLUSION

The Sign Language Tutor application project has been successfully implemented, and it has proven to be effective in recognizing and interpreting sign language gestures using the fingerpose library and TensorFlow handpose model. The application provides users with an interactive platform to learn sign language easily and conveniently, with features such as sign gesture recognition, meaning interpretation, and a scoreboard to track progress.

During the testing phase, the application underwent various types of testing, including functional testing, non-functional testing, unit testing, integration testing, and validation testing. These tests helped to identify and fix several bugs and errors, ensuring that the application is of high quality and performs optimally.

The implementation and testing of the Sign Language Tutor application have resulted in a robust and reliable application that can serve as a valuable resource for individuals interested in learning sign language. The application has the potential to make a positive impact on the lives of people with hearing disabilities, as well as those who want to communicate effectively with them.

Overall, the Sign Language Tutor application project has been a success, and it is poised to be an essential tool for promoting inclusivity and improving communication between people with hearing disabilities and the rest of the world.

VI. FUTURE WORK

The Sign Language Tutor application project has opened up many possibilities for future work and improvements. Some of the areas that could be explored include:

1.Adding more sign language gestures: The current version of the application only recognizes a limited number of sign language gestures. More gestures can be added to increase the scope of the application and make it more useful for users.

2.Improving accuracy: While the application has a good level of accuracy, there is still room for improvement. Fine-tuning the hand pose model and improving the image recognition algorithm can help to increase the accuracy of the system.

3.Adding user profiles: The current version of the application does not have user profiles. Adding user profiles can help to personalize the application and make it more engaging for users.

4.Creating a mobile version: The current version of the application is web-based. Creating a mobile version of the application can make it more accessible and user-friendly.

5.Implementing feedback mechanisms: Users can provide feedback on the application's performance and accuracy. Implementing feedback mechanisms can help to identify areas that need improvement and make the application more effective.

Overall, the Sign Language Tutor application has a lot of potential for future work and improvements. By addressing the areas mentioned above and other potential areas, the application can be made more useful and accessible to users.

VII. BIBLIOGRAPHY

Christopher Anopaishe Meki is a final year student studying Software Engineering at HIT

REFERENCES

[1] L. Ponemon. Cost of a lost laptop. Technical report, Ponemon Institute, April 2009.

[2] M. Marshall, M. Martindale, R. Leaning, and D. Das. *Data Loss Barometer*. September 2008. Seagate Technology. Can your computer keep a secret? 2007.

[3] Seagate Technology. Drivetrust technology:a technical overview. 2007.

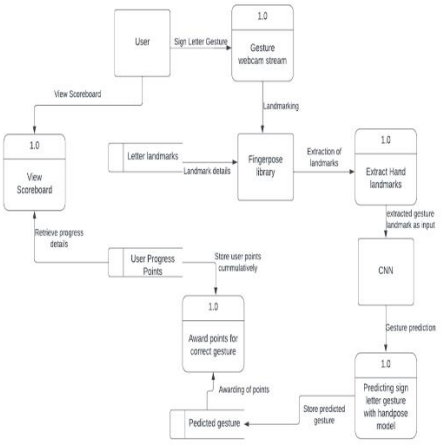
[4] P. Kleissner. Stoned bootkit. In *Black Hat USA*, 2009.

[5] Ellick M. Chan, Jeffrey C. Carlyle, Francis M. David, Reza Farivar, and Roy

H. Campbell. Bootjacker: compromising computers using forced restarts. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 555–564, New York, NY, USA, 2008. ACM.

[6] Sven Türpe, Andreas Poller, Jan Steffan, Jan-PeterStotz, and Jan Trukenmüller. Attacking the bitlocker boot process. In *Trust '09: Proceedings of the 2nd International Conference on Trusted Computing*, pages 183–196, Berlin, Heidelberg, 2009. Springer-Verlag.

[7] L. Ponemon. The human factor in laptop encryption. Technical report, Ponemon Institute,


User
Sign Letter Gesture
1.0
Gesture webcam stream
View Scoreboard
Landmarking
1.0
View Scoreboard
Letter landmarks
Landmark details
Fingerpose library
Extraction of landmarks
1.0
Extract Hand landmarks
extracted gesture landmark as input
Retrieve progress details
User Progress Points
Store user points cummulatively
CNN
1.0
Award points for correct gesture
Gesture prediction
1.0
Predicting sign letter gesture with handpose model
Awarding of points
Store predicted gesture
Predicted gesture