



TED UNIVERSITY

CMPE 492

Senior Project

Gesture Guide: Virtual Assistant for the Hearing-Impaired

Final Report

Submission Date: 06.06.2024

Team Members:

- Berke Lahna
- Canberk Aydemir
- Ceyhun Toker
- Mehmet Fatih Ülker

Advisor: Venera Adanova

Jurors: Gökçe Nur Yılmaz and Eren Ulu

Web Site: <https://berkelahna.github.io/Pages>

TABLE OF CONTENTS

| | |
|--|-----------|
| 1. Introduction | 1 |
| 1.1 Overview..... | 1 |
| 1.2 Definitions, Acronyms, and Abbreviations | 1 |
| 2. Problem Statement..... | 2 |
| 3. Implementation | 3 |
| 3.1 Machine Learning Models..... | 3 |
| 3.2 Backend Server | 5 |
| 3.3 User Interfaces..... | 6 |
| 4.Tools and Technologies..... | 11 |
| 5.Testing Results | 12 |
| 5.1 User Interface Test Cases | 12 |
| 5.2 Backend Test Cases..... | 16 |
| 5.3 Machine Learning Models | 18 |
| 6. Testing Outcomes..... | 20 |
| 7. Discussion and Possible Flaws | 22 |
| 8. Future Iterations Of The Project..... | 22 |
| 9. Impact of Engineering Solutions | 23 |
| 10. Conclusion | 23 |
| 11. Appendix..... | 24 |
| 12. References | 26 |

1. Introduction

1.1 Overview

Throughout two semesters that we have worked on the Gesture Guide, we have had the chance to conceptualize every small detail of the Gesture Guide project by constructing the project as a software application from scratch. Now that the development and testing phase of Gesture Guide is coming to an end, the time has come to go over every aspect of the Gesture Guide application to demonstrate what we have done for the final product, the outcomes of the testing phase, possible flaws and impact of the project. Throughout this document we will be diving deep into both implementation and testing phase of the project. But more than that we will be talking about flaws of the project and how the project can be improved upon in the next iterations of the project. Overall, we will be finalizing our project by means of writing this report and trying to demonstrate the importance of Gesture Guide.

1.2 Definitions, Acronyms, and Abbreviations

Abbreviations

- UML: Unified Modelling Language
- ISO: International Organization for Standardization
- ID: Identification Number
- OOP: Object Oriented Programming
- ANN: Artificial Neural Networks
- CNN: Convolutional Neural Networks
- CPU: Central Processing Unit

For detailed definitions please refer to the glossary section located at the end of this document.

2. Problem Statement

Gesture Guide project has been defined as the virtual assistant for the hearing impaired since the idea for it was born. The modern virtual assistants like Siri or Alexa are software applications that create an illusion of having an assistant in your pocket that will listen to your commands and will create a response to these commands. In this cycle of communication there is one prerequisite, the user should speak or formally use their vocal chords to interact with these virtual assistants. This is where the idea of Gesture Guide comes to play. The hearing impaired cannot interact with these virtual assistants due to their inability to use their vocal chords as the primary source of communication. Primary source of communication for the hearing impaired is by the means of sign languages and hand gestures. In order to construct this communication environment for the hearing impaired, the virtual assistant needs to be taking hand gestures as input. So, Gesture Guide project has made it its mission to create a virtual assistant that takes hand gestures as input and tries to create the same effect as the modern virtual assistants. As a secondary goal we have also chosen to integrate a sign language interpreter into the gesture guide so that it would also help to create a better communication environment for the target users. Looking at the big picture we have tried to develop both a communication environment in the form of virtual assistant and a tool for communication between the hearing impaired and normal people. With the launch of GPT 4-O the idea of helper virtual assistants has become significantly more relevant compared to when we have first thought of Gesture Guide.

The Gesture Guide also needed to offer users personalized accounts which would provide users with features such as security, personalization and confidentiality. So, the whole virtual assistant and sign language interpreter was designed to be one small component of a big software application that has very familiar patterns to applications offering personalized accounts. The problems that the hearing impaired go through in their daily lives are proposed to be remedied to an extent by the features provided by Gesture Guide.

3. Implementation

Now that we have defined the problem and the motivation behind Gesture Guide, we can better analyze the implementation of the project. This section will be divided into 3 parts all diving deep into different component of the implementation. These components will be machine learning models, backend server and user interfaces.

3.1 Machine Learning Models

For a virtual assistant that works with hand gestures you need a machine learning model that can make predictions on frames containing gestures. This problem is a classification problem revolving hand gestures. First thing that we did for this purpose was to find a dataset of hand gesture images. The dataset that we eventually found has 2000 images of people demonstrating some kind of gesture which consisted of 7 classes. These classes being gestures thumbs up, thumbs down, up, down, right, left and stop. For this problem of gesture detection and object detection tasks the contemporary way of approach is through training a CNN model. The CNN model is a powerful neural network that applies different filters(dot product with matrix) and tries to extract meaningful features from images. The filters (matrices) are learned and improved upon in each step using backpropagation and gradient descent. Since training a CNN by constructing different layers and experimenting with different activation functions is hard and very time consuming, we have used the YOLO model. YOLO model is a CNN based object detection model that can be trained for custom datasets. YOLO is fast and way much easier to train, so our ultimate choice was using YOLO. Our dataset was compatible with YOLO meaning all images had labels stating their class and the location of the bounding box, so we did not have hard times preprocessing the data. We have trained for 300 epochs using batch size 32 and tried different optimizers such as Adam, stochastic gradient descent and gradient descent with momentum. The best optimizer was Adam at the end of the day. After long training processes, the YOLO model eventually learns to make predictions on both class of an image and the box that contains the image. Overall, we have trained a successful gesture model which we will be evaluating more deeply in the next section. Figure 1 bellow shows the model making predictions on webcam feed.

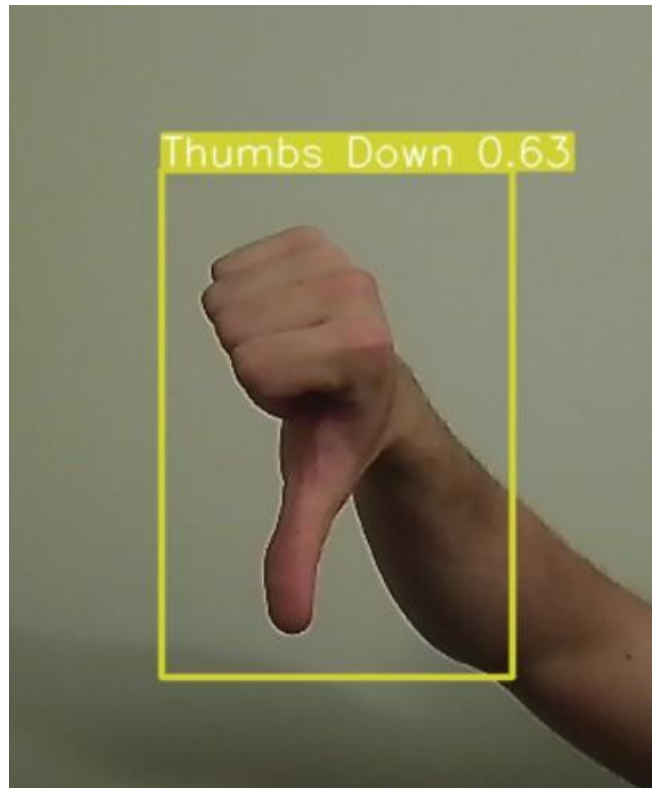


Figure 1: The Hand Gesture Recognition

Sign language interpretation was another task that required machine learning models. In essence hand gesture recognition and sign language detection are very similar tasks both requiring a spatial understanding of images. So, for this purpose as well we needed another CNN based model with fine-tuned parameters. For this case the images that will be used for classification are images showing a sign language character and where in the picture the sign lies in (bounding box). The dataset that we have worked on this task had 5,200 images of size 416×416 which belong to 26 classes. Using this dataset we have trained another YOLO model for 300 epochs, batch size set to 64 and optimizer Adam. Here we have tried many models and many datasets but eventually constructed the model that outperforms others. Figure 2 below shows the sign language interpreter in action.

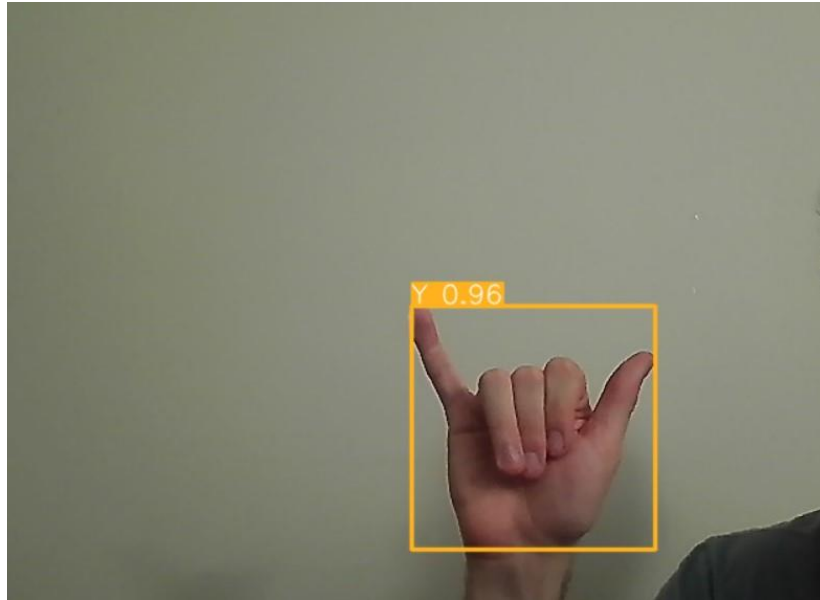


Figure 2 : The Sign Language Interpretation

3.2 Backend Server

Gesture Guide project is so much more than just 2 machine learning models working on their own as we have stated earlier that the aspect of Gesture Guide as a personalized software application requires a backend server that will handle user actions and persistent data management. The 2 entities that are required to be represented as persistent data objects are users and virtual assistant queries. User entity in backend server represents the real users of the application and are required for constructing the user workflows such as registration, authentication, authorization. Whereas the virtual assistant queries are entities to store the past virtual assistant queries performed by users and provide them a shortcut to access these queries using the user interface. For every persistent data object, we have created the repository-service-controller architecture. In this architecture repository layer is responsible for interacting with the database, manipulating the database and handling ORM tasks such as converting Java objects to database entities of user and assistant queries. The service layer interacts with the repository layer to create business logic involving both users and queries.

Finally, the controller is the layer that receives HTTP request from user interface and propagates them to service layer for obtaining/manipulating persistent data. The whole architecture was built for both users and virtual assistant queries. This architecture was responsible for creating business logic as well as persistent data management through database.

For authentication/authorization we have implemented JWT based authorization where every request that is sent to the backend server should also include a JWT token as a HTTP header for user to access that specific backend route on the server. JWT token is an encrypted JSON text that encapsulates the user that sends the request and the role of the user. With the appended authorization header, the backend server checks if the users JWT is valid, and the user has authority to access that specific route. The first instance that user logs in, if the user does exist in the database and credentials are correct, a JWT is created and this JWT should be provided by the user interface in each of the HTTP request to backend. After login, every time a request is made to the backend, the backend validates both user and role of the user through the JWT token before giving permission, creating both authentication and authorization.

3.3 User Interfaces

User interfaces are the layers where users interact with overall system. The user interface that we have created are the desktop application written in Python and mobile application written in Kotlin for android. Both user interfaces possess functionality for users to interact with the backend server without explicitly interacting with it. Both interfaces offer a different approach to visuals as well as the virtual assistant component. First, we would like to go through the desktop application. Every personalized data that is displayed on the user interface comes from the database whether it is a table of assistant queries or profile information of users. In a way the user interface is the layer that encapsulates the machine learning models and the backend server at the same time. Some basic views of the user interface are registration page, login page, my profile page and past assistant queries which are implemented in both interfaces. Figure 3 below shows the dashboard page of the user interface.

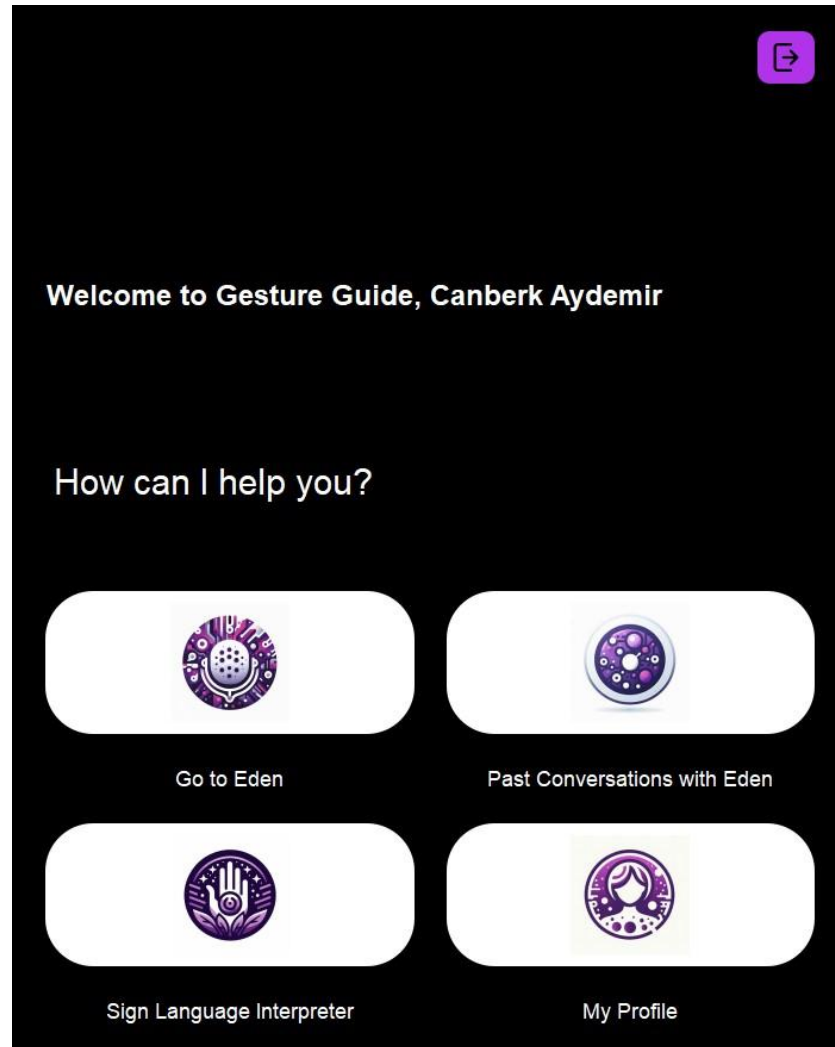


Figure 3: The Dashboard Page

Virtual Assistant Page is the view where users interact with the virtual assistant and the hand gesture recognition model. This view allows users to open their webcams and display one of the gestures to transition to a widget just like the response the modern virtual assistant creates for a query. These widgets being weather widget, coin toss, unit converter, calculator, quote of the day, dad joke, and plate code guessing game. Each frame the model will try to predict the gesture and based on the classified gesture; different widgets will be opened. For gesture Thumbs Up the weather widget is displayed where users can get weather forecast for a city of their choice which is powered by an external API. For gesture Thumbs Down the unit converter is displayed where users

convert between different units of length, mass and volume. “Down” makes the transition to coin toss simulation. “Up” gesture opens the plate code guessing game where users try to guess the plate code of a city whereas “Left” gesture opens the dad jokes widget which displays jokes on a page. Finally, the “Right” gesture opens the quote of the day component which uses an external API to display a quote on the screen. A simpler scheme is implemented for android as well, but actions are switched with sending messages, making calls etc. Figure 4 bellow shows the weather widget (response of thumbs up) from the user interface.

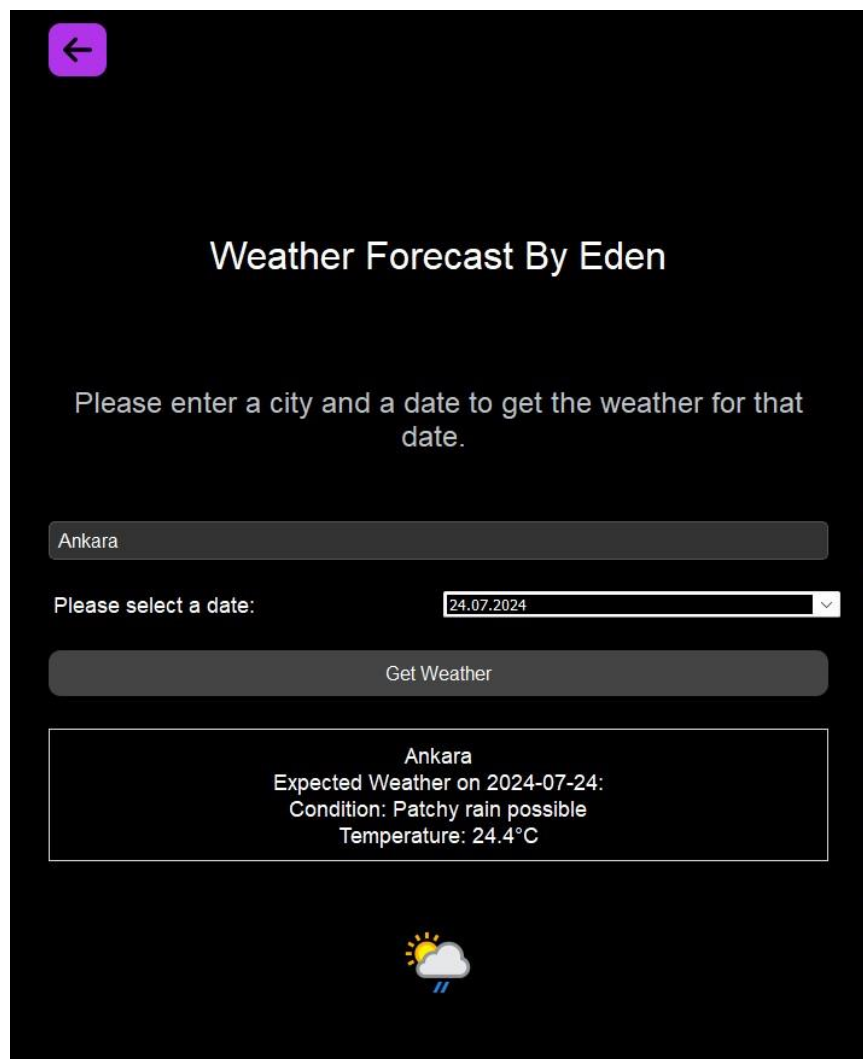


Figure 4: The Weather Widget

Sign Language Interpreter component is a similar component to the virtual assistant component. In this component users can start webcam feed and start displaying sign language characters. On these frames the model makes predictions and if there is 90% confidence it displays the character on the screen. In this way users can sequentially build the message they are trying to express through this medium. Users sequentially show the characters they would like to express and can even add spaces in between sequences to build the complete sequence. Figure 5 shows the sign language interpreter in action with the sequence “Hello World”.

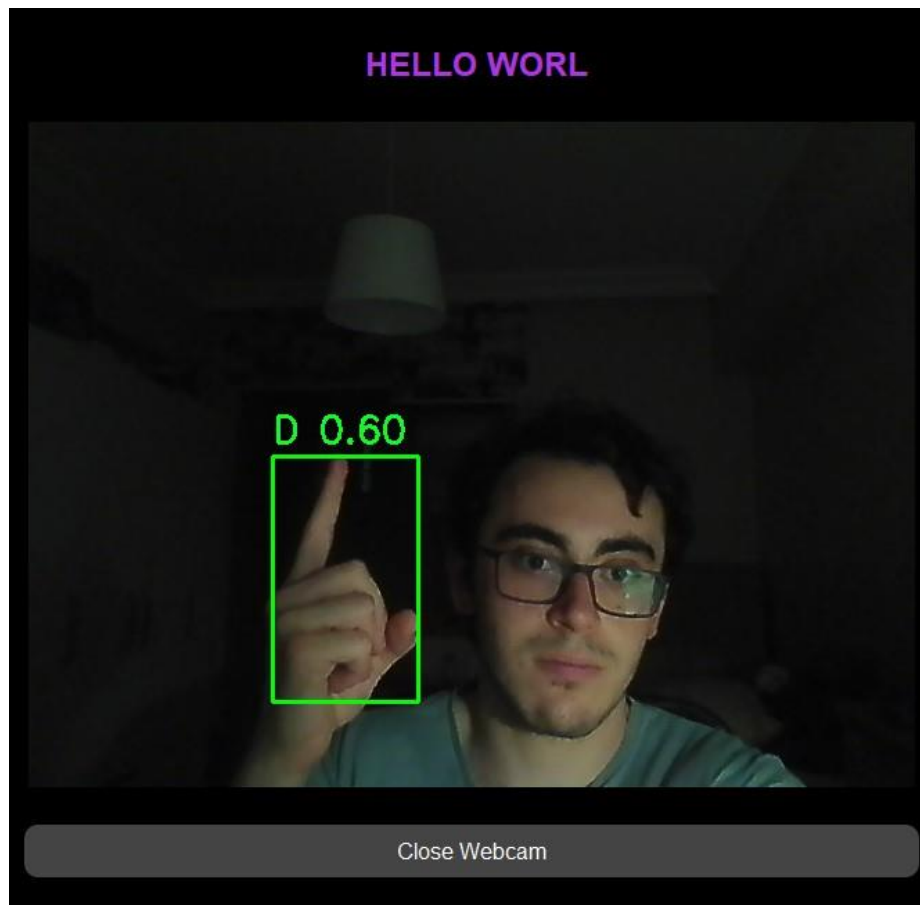


Figure 5: The Sequence” Hello World” in Sign Language Interpreter

The android application offers a completely different aesthetic and a different approach at the virtual assistant. The color theme of the android is of the tones of blue rather than the dark/purple tones of the desktop application. Overall, the interaction with the database is pretty similar and it has very similar components to the desktop application on views such registration, login and profile page. The virtual assistant on the other hand works with a secondary backend that is only used to leverage the machine learning model interactions of the android application. Each frame that is read by the device camera is forwarded to the secondary backend that only uses the models to predict the given frame and returns the prediction with the class of the predicted class. This mechanism is built for both hand gesture recognition and sign language interpreter. The response of the virtual assistant is different than the desktop application as it focuses on internal android functions such as sending an “SMS” rather than displaying cool widgets like the weather widget of the desktop application. Even though the gestures are the same, the functionality is designed specifically for an android application. Figure 6 below displays various views from the android application.

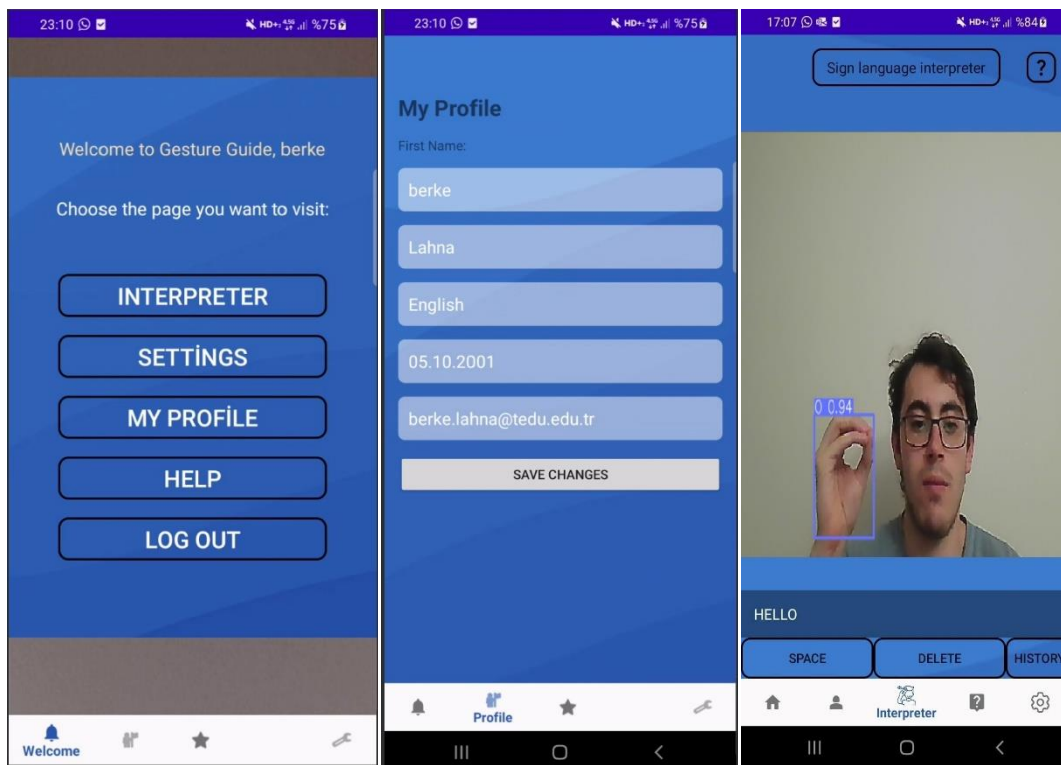


Figure 6: Login, Profile and Sign Language Interpreter views of the Android Application

4.Tools and Technologies

We have used a lot of external libraries for the final product of the Gesture Guide. This libraries allowed us to implement the system. Let us go through all of these libraries and for what reasons they were selected and utilized.

- **Machine Learning Models:** For model training we have used the ultralytics package of python. Ultralytics is the go-to library for training YOLO models with different parameters, optimizers etc. The flexibility the ultralytics library provides is a reason why we have selected this library as it allows users to train models in any of the past Yolo models from v5 to YOLO-Nas. It also allows users to train models using GPU which makes everything faster.
- **Backend Server:** The backend was completely written in Java Spring-boot which is probably the one of the best choices for constructing enterprise level applications that require great attention. We have also used Spring data-jpa for object relational mapping and spring security for constructing JWT based authorization. Also, for the database we have used PostgreSQL through Docker which has sped up the configuration of the database a lot.
- **Desktop Application:** Desktop application is primarily coded in PyQt which is the QT library's counterpart in Python used for creating highly functional desktop applications. From tables to buttons the library provides small components that can be used to create complex application. Since the desktop application runs the machine learning model it also uses ultralytics module. And request module is used for creating HTTP requests.
- **Android Application:** The android application is coded in android studio using the Kotlin language. The helper backend for the android application is written in fastAPI and uses ultralytics module to make predictions on frames. The fast development environment provided by Google really helped us to implement a good-looking interface very fast.

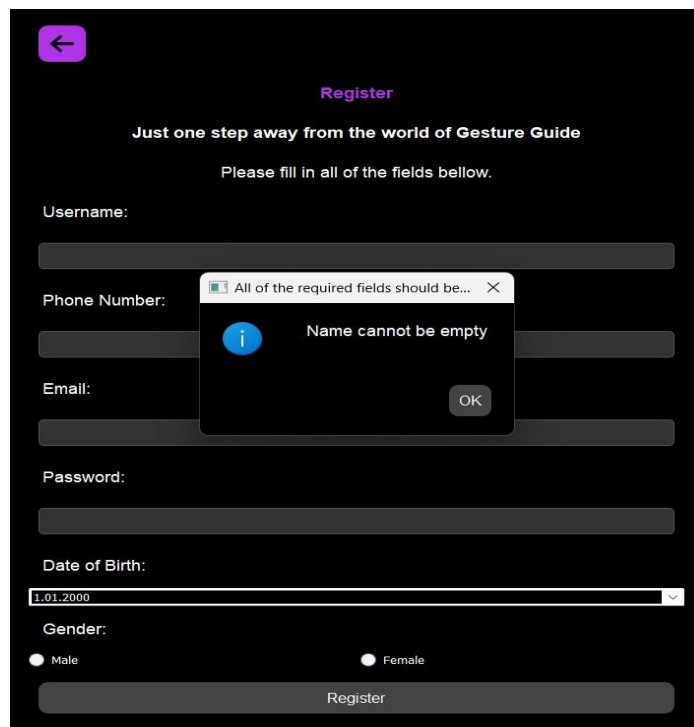
5. Testing Results

In the testing plan document, we have talked about every aspect of testing phase of the project ranging from test cases to testing methodology. We have completed the testing phase as of June 1 being complaint to the schedule that was delivered in the testing plan. This section will focus on testing results and the testing phase as a whole. First let us walk through the test cases and their outcomes.

5.1 User Interface Test Cases

a) User Registration (Test Case 1.1-1.3)

In these test cases we have tested the registration function of the user interface where users enter their information, and a HTTP request is sent to the backend and the user is registered. Test cases 1.1 and 1.2 simulated registration when users did not fill in the form. For this cases everything worked as expected as can be seen in Figure 7 bellow. Test case 1.3 tested registration when the user has filled every field correctly and this test also passed.



The screenshot shows a mobile application interface for user registration. At the top, there is a back arrow icon. Below it, the title "Register" is displayed in red. Underneath the title, the text "Just one step away from the world of Gesture Guide" is shown in a smaller font. Below this, a prompt says "Please fill in all of the fields bellow." The registration form consists of several input fields: "Username:", "Phone Number:", "Email:", "Password:", "Date of Birth:" (with a date picker set to "1.01.2000"), and "Gender:" (with radio buttons for "Male" and "Female"). At the bottom of the form is a "Register" button. A modal dialog box is overlaid on the form, displaying an information icon and the message "Name cannot be empty". Above the modal, a toast message reads "All of the required fields should be...".

Figure 7: Failed Register Scenario

b) User Login (Test Case 2.1-2.4)

In these test cases we have tested the login function of the user interface where users enter their credentials that they have used in registration, and a HTTP request is sent to the backend and the user is logged in to the system. Test cases 2.1 and 2.3 simulated login when users did not fill in the form or entered wrong credentials and all of these tests passed thanks to form validation and authentication we have implemented. The outcome of test case 2.3 can be seen in Figure 8 bellow. Test case 2.4 tested login when the user has filled correct credentials and this test also passed successfully.

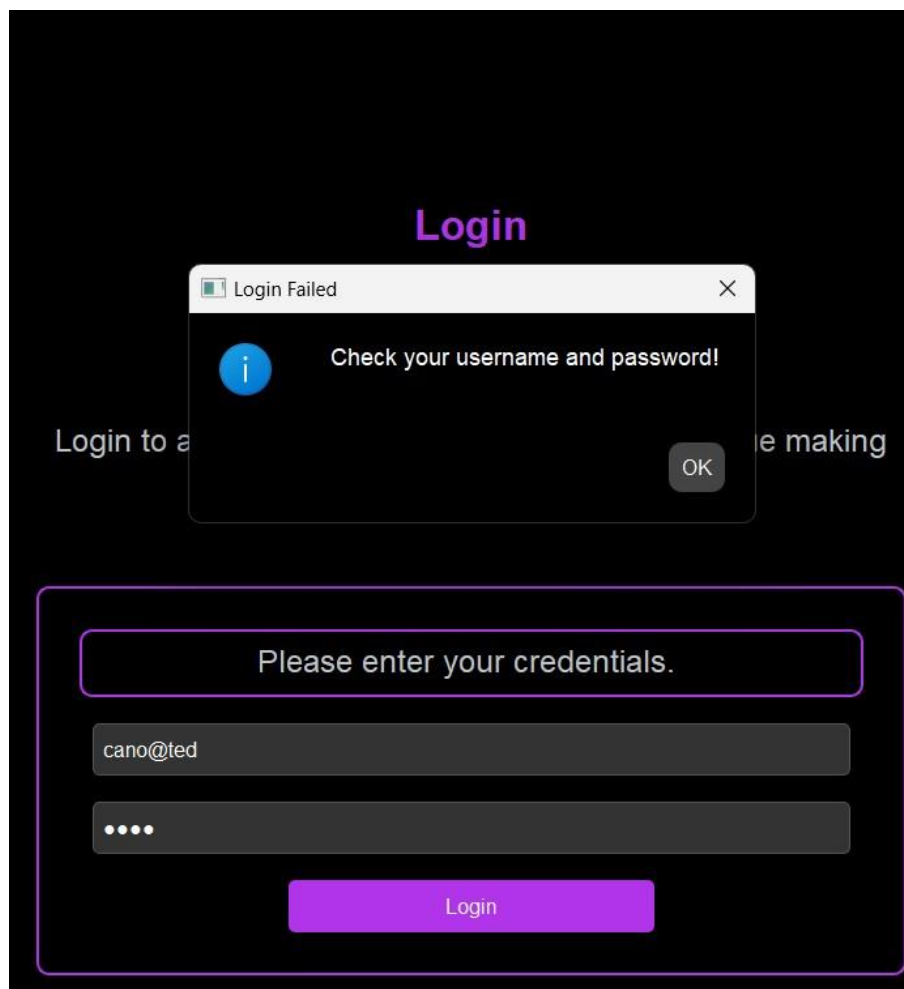


Figure 8: Failed login with wrong credentials

c) Viewing Past Assistant Queries (Test Case 3.1-3.2)

In these test cases we have tested the viewing past assistant queries functional requirement of the Gesture Guide. In Gesture Guide User entities have their assistant queries in a one to many relationship meaning each user possesses many assistant queries of the past. These test cases have tested if the queries were updated and displayed correctly. Test case 3.1 simulated viewing past assistant queries when users did not have any query to be shown and this test passed successfully. Test case 3.2 the same functionality but when the user has a collection of past virtual assistant queries and this test also passed meaning that the backend also works correctly. Figure 9 bellow shows the outcome for test case 3.2.

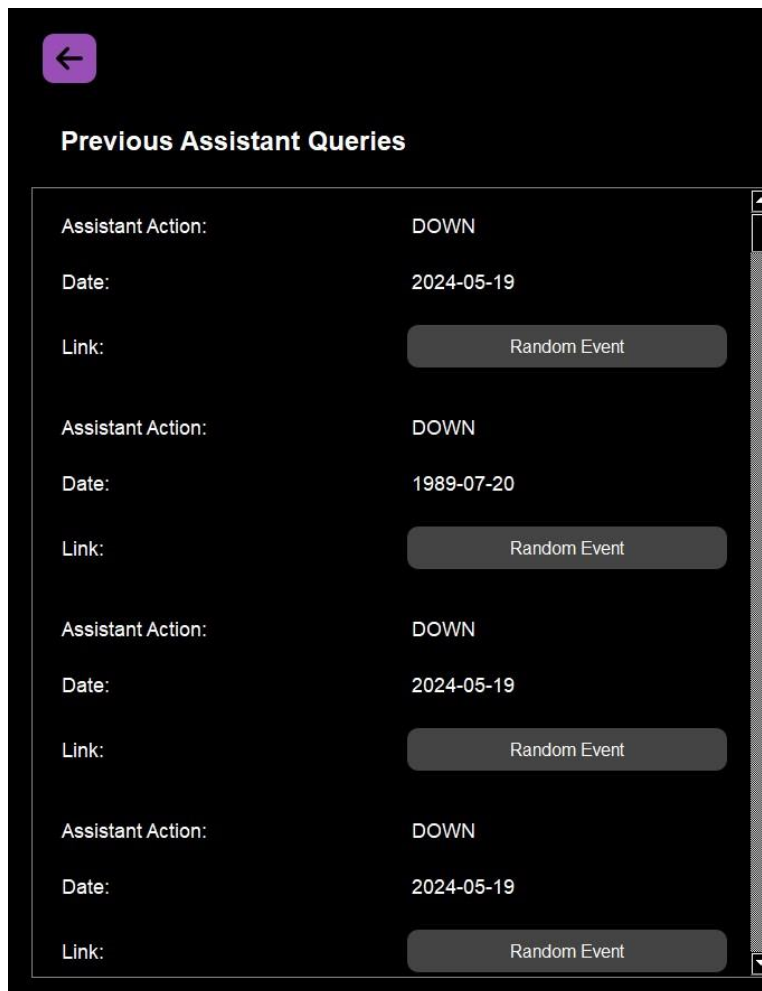


Figure 9: The simulation for test case 3.2

d) Accessing Virtual Assistant/Sign Language Interpreter (Test Case 4.1-4.2)

In these test cases we have tested the accessing of the virtual assistant/sign language interpreter functionality of Gesture Guide. After login users can access both the virtual assistant and the sign language interpreter through the dashboard. Both of these pages prompt users to open the webcam, if the user does not press the required buttons than the machine learning model does not start making predictions. These test cases were simple requirements for the application and application passes both of these cases due to design requirements.

e) Virtual Assistant Functionality (Test Case 5.1-5.2)

In these test cases we have tested the virtual assistant and the responses created for each and every gesture displayed on the virtual assistant. As you already know the virtual assistant is an abstraction under the hand gesture recognition model. Every gesture detected by gesture detection model creates a transition to one of the 7 widgets which are counterparts to 7 different classes the gesture recognition model can classify. The expected behavior is that when “Thumbs Up” is displayed the weather widget should be opened and when “Left” is shown the “Quote of the Day” component should be opened etc.

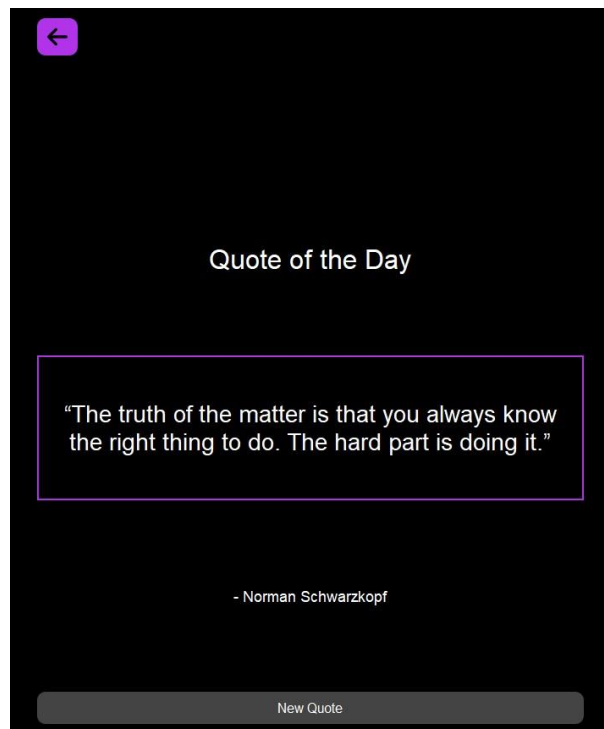


Figure 10: The Quote Of the Day Component

After testing this functionality, we had failed some test cases needing us to reconsider the confidence interval of each gesture, so we have set higher level of confidences for “Thumbs Up” and “Thumbs Down” and set lower levels for the rest of classes and everything worked very well. These test cases were reported as a defect, but we were able to solve the defect and eventually all the test cases were passed. Figure 10 above shows Quote of the day component after gesture “Left” is displayed.

5.2 Backend Test Cases

Backend test case was tested by explicitly coding unit tests using Junit and Mockito to mock how classes would work under certain circumstances. In this section we will be talking about these unit tests and their results. In this section we have so much test cases to dive deep into so we will be separating this section to 2 parts testing of UserService class’s functionality and testing of VirtualAssistantService class’s functionality.

a) UserService Class (Test Cases 6-11)

In these test cases we have tested the methods of UserService class methods using unit tests. Unit tests test the smallest unit of the software with different conditions and boundaries. Starting from test case 6 we have tested the backend functionality to create users, we have tested scenarios such as creating users with missing required fields and users with all of the required fields set. The results were expected, if the user had one missing field the users would not have been created. For test case 7 we have tested fetching all of the users when there is multiple user’s vs when there were no users registered in the database and all the scenarios have been passed for this case as well. Test case 8 tested fetching a specific user with userId given as parameter. Recall that this function is needed for authentication and profile page. This test case was crucially important, but this test case passed as well just like fetching all the users case. In Test case 9 we have tested user deletion capabilities - which is not so relevant for the bigger application of the backend and all the variations of the test case were successfully passed. In test cases 10 and 11 we have tested the functionality to update/get user’s past virtual assistant queries given a virtual assistant query or userId as parameter. This test cases were also crucial part of the application compliant to viewing past assistant history functional requirement. The results were great as all the tests passed successfully reflection of the strong backend server we have implemented.

b) AssistantQueryService Class (Test Cases 12-16)

For test cases between 12 and 16 we have tested backend functionalities of the AssistantQueryService class which is the brain behind the business logic created for assistant queries. Test Case 12 tested the functionality to create new assistant queries with a given assistant query object either with empty fields or all of the fields set. This test case was successful just like test case 13 which tested fetching all of the virtual assistant queries when there is no assistant query to be find or there is a collection of assistant queries. Case 14 was just like case 8 but for the case of queries where we are trying to fetch an assistant query based on a query id that does exist or one that does not exist. This test case was successful as well. With case 15 we have tested deletion of assistant queries and tested both valid and invalid assistant queries and the results were as expected the query was deleted if the id was valid whereas no deletion occurred for the invalid id case. Case 16 was the final test case we have simulated on Junit where we were trying to test the obtaining the user related to the assistant query functionality of the backend server. At first this test case did not work, but when we modified the assistant query creation process this test case was successful. Overall, the backend of the system was pretty successful almost passing all of the test cases for a sample unit test you can see figure 11 bellow.

```
@Test new *
void getAllUsers() {

    List<Users> allUsers = List.of(
        new Users( name: "Jane Doe",   phoneNumber: "41567789543",   email: "jane.doe@tt.com",   password: "password"),
        new Users( name: "John Doe",   phoneNumber: "42567789543",   email: "john.doe@tt.com",   password: "passwor"),
        new Users( name: "Joe Doe",     phoneNumber: "43567789543",   email: "joe.doe@tt.com",    password: "passwo"),
        new Users( name: "Jack Doe",    phoneNumber: "44567789543",   email: "jack.doe@tt.com",   password: "passw"),
        new Users( name: "Jones Doe",   phoneNumber: "45567789543",   email: "jones.doe@tt.com",  password: "pass"));

    Users userN = new Users( name: "Jones Doe",   phoneNumber: "45567789543",   email: "jones.doe@tt.com",  password: "pass");

    Mockito.when(usersRepository.findAll()).thenReturn(allUsers);
    Mockito.when(modelMapper.map(userN, UsersDto.class)).thenReturn(UsersDto.of(userN));

    List<UsersDto> testAll = allUsers.stream().map(user -> modelMapper.map(user, UsersDto.class)).toList();

    List<UsersDto> getAll = usersService.getAllUsers();

    assertEquals(testAll, getAll);
}
```

Figure 11: The Unit code for test case 7

5.3 Machine Learning Models

In the implementation section we talked about how we have trained the models and which hyperparameters we have used. Now we should check how they perform on the validation sets as well as the webcam inference. The Hand Gesture Model and Sign Language Model performs great on their validation sets. The sign language model reaches class losses of 0.14 and box losses of 0.32 on the final epoch. At the same time achieving almost 0.99 precision, recall and Map-50 metrics on the validation set. These metrics can be seen from figure 12 bellow.

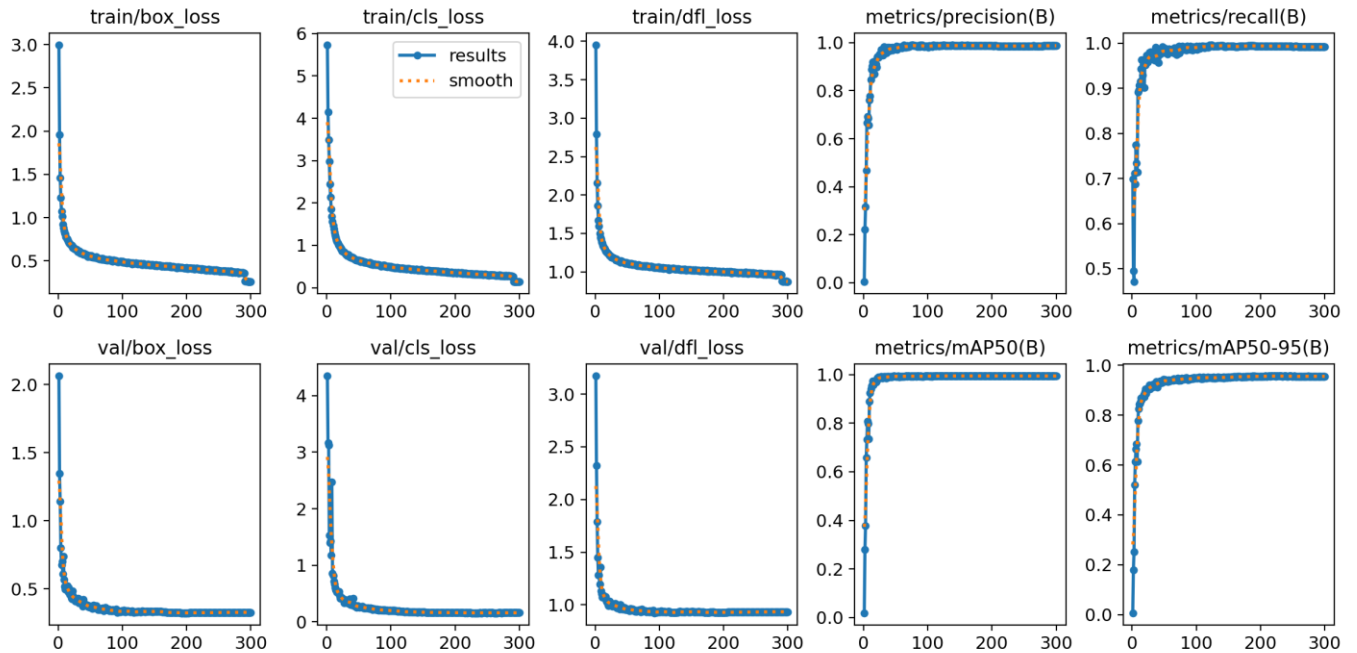


Figure 12: The Classification metrics for sign language model.

The gesture recognition model on the other hand performs very well -slightly worse than sign language model- on the validation set reaching class losses of 1.1 and box losses of 1.7. It performs slightly worse than the sign language model because sign language model has had 2.5 times the training samples of the gesture recognition model. That's why the sign language model works far better even on high confidence levels. The other classification metrics such as precision, recall and

MAP-50 stays around 80% percent which is a reflection of the lower validation losses. Figure 13 bellow shows the classification metrics plot for hand gesture model.

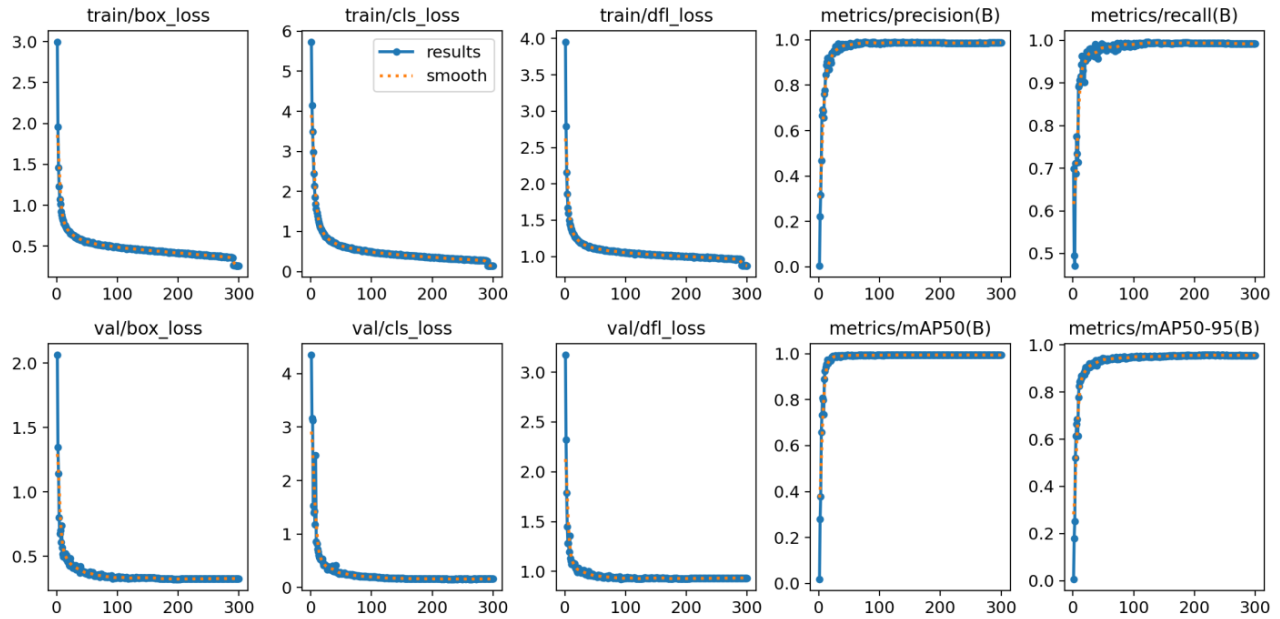


Figure 13: The Classification metrics of hand gesture model for each epoch

Performance comparison of these models show that sign language performs slightly better than hand gesture recognition models. When making predictions via webcam we have seen that with good lighting it can almost predict any of the classes with more than 90 percent confidence. There are some characters that are slightly ambiguous (like another character) so sometimes there are problems classifying these characters such as “m”, “u”, “n”, “q” and “i” but other characters just work very well. The gesture model also works very well but it has inclination to predict “Thumbs Up” over “UP” that’s why we fixed the confidence level of “Thumbs Up” to be 90 percent in the virtual assistant. There is also slight ambiguity between gestures “Right” and “Left” and sometimes two can be mixed up when making predictions. But overall, all of the gestures work very well even on webcam inference. But it should also be noted that the bad lighting (too much sunlight) is a crucial factor for webcam inference as it may completely corrupt the predictions the models make. This phenomenon happens due to inefficiency of webcams not being able to focus properly.

6. Testing Outcomes

| Test Case Id | Methodology | Importance | Tester | Testing Date | Outcome |
|--------------|---------------------------------|------------|---------|--------------|----------------------|
| 6 | Unit Test Via Junit | Moderate | Canberk | 06/05-12/05 | Passed |
| 7 | Unit Test Via Junit | Moderate | Canberk | 06/05-12/05 | Passed |
| 8 | Unit Test Via Junit | Moderate | Canberk | 06/05-12/05 | Passed |
| 9 | Unit Test Via Junit | Low | Canberk | 06/05-12/05 | Passed |
| 10 | Unit Test Via Junit | Moderate | Canberk | 06/05-12/05 | Passed |
| 11 | Unit Test Via Junit | Moderate | Canberk | 06/05-12/05 | Passed |
| 1 | Unit Test without Explicit Code | Crucial | Ceyhun | 13/05-19/05 | Passed |
| 2 | Unit Test without Explicit Code | Crucial | Ceyhun | 13/05-19/05 | Passed |
| 3 | Unit Test without Explicit Code | Crucial | Ceyhun | 13/05-19/05 | Reported as Defect |
| 4 | Unit Test without Explicit Code | Moderate | Berke | 13/05-19/05 | Reported as Defect |
| 5 | Unit Test without Explicit Code | Moderate | Berke | 13/05-19/05 | Reported as Defect |
| 17 | Validation Testing | Crucial | Fatih | 19/05-31/05 | Passed/Conditionally |
| 18 | Validation Testing | Crucial | Fatih | 19/05-31/05 | Passed/Conditionally |

| | | | | | |
|------------------|------------------------------------|----------|---------|-----------------|--------|
| 12 | Unit Test Via Junit | Moderate | Canberk | 27/05- 31/05 | Passed |
| 13 | Unit Test Via Junit | Moderate | Canberk | 27/05- 31/05 | Passed |
| 14 | Unit Test Via Junit | Moderate | Canberk | 27/05- 31/05 | Passed |
| 15 | Unit Test Via Junit | Low | Canberk | 27/05- 31/05 | Passed |
| 16 | Unit Test Via Junit | Moderate | Canberk | 27/05- 31/05 | Passed |
| 3 (Revisited) | Unit Test without Explicit Code | Crucial | Ceyhun | 27/05- 31/05 | Passed |
| 4 (Revisited) | Unit Test without Explicit Code | Moderate | Berke | 27/05- 31/05 | Passed |
| 5 (Revisited) | Unit Test without Explicit Code | Moderate | Berke | 27/05- 31/05 | Passed |

7. Discussion and Possible Flaws

Throughout this document we have demonstrated how we have implemented the overall system for the Gesture Guide project even talking about the results of the testing phase of the project. Overall we have actually kept our promises and implemented every feature that we have thought of as early as October of 2023. The virtual assistant that works with gestures is implemented, the real time sign language interpreter is implemented and personalized accounts flow is implemented in the final product of the Gesture Guide. But that does not mean we have a perfect project with no flaws. Even in this document we have mentioned couple of these flaws. Like for example the problem with webcam inference and its sensitivity to lighting. Even though our models work very well on the validation set, due to different lighting the models may not work as expected and misclassify a sign/gesture. On the other hand, we have chosen to implement simple and strong user interfaces both in the desktop application and the android application but maybe we should have created more complex and visually aesthetic applications that would look better than our application. All these questions aside we believe that even with its flaws the Gesture Guide has come a long way, and it delivers every functionality that was planned since the beginning.

8. Future Iterations Of The Project

This section is dedicated to expressing how we would continue developing the project in the future iterations of the Gesture Guide application. The first feature we would add to the Gesture Guide application would be gestures as videos rather than images. The current version of Gesture Guide makes predictions on frames alone, disregarding the earlier and the future frames. In future we would like to incorporate gestures such as swiping that is actually multiple frames played at a constant rate. But this would require complex models using 3D convolution and usage of sequential models. The other feature we would add in the future would be classifying sign language words such as “Hello” which are also a sequence of frames. For the word level sign language interpretation, we would need a dataset of probably terabytes and training times of weeks. Other than that the future iterations of the project would work on constructing even more visually aesthetic user interfaces for the application.

9. Impact of Engineering Solutions

Now that we are finalizing the Gesture Guide project, we need to talk briefly about impact of the engineering solutions we have provided along the way. On the bigger picture the Gesture Guide is an accessibility application that aims to make the lives of hearing impaired easier. So, we have no doubt that on a global level it will promote accessibility tools. Since the project also offers aid in communication through the sign language interpreter. The project offers a new communication channel that eliminates the caveats people have when they are trying to communicate with a hearing-impaired person. Overall, the impact of the Gesture Guide is parallel to the communication solutions that it offers to the hearing-impaired community which can be a catalyst of accessibility applications that it can inspire in the future.

10. Conclusion

Throughout the Gesture Guide project, we have not only tried to create a virtual assistant for the hearing impaired but a communication tool that eliminates at least some of the problems the hearing-impaired community comes across in their lives. The combination of virtual assistant and the sign language interpreter was a new approach to the accessibility applications targeted for the hearing impaired. And after a development phase of almost 8 months, we were able to implement the accessibility application Gesture Guide with all of the features we have promised. Gesture Guide was a unique approach at accessibility applications 8 months earlier but in today's world the big tech companies are making advertisements revolving AI applications helping blind people with the surrounding objects. So, we can say the accessibility application using AI will be very popular in the future, we cannot tell if any of them will be an augmented version of this project, but we know that one of the primitive examples of AI based accessibility applications will be the project we have developed in the scope of our graduation project the Gesture Guide.

11. Appendix

- **Authentication:** In a software system, validating if a user exists in the system.
- **Authorization:** In a software system, granting authority to users based on their roles.
- **Classification:** In machine learning, the process of determining which class a sample belongs to.
- **Credentials:** Personal information that users use to register/log in to a system. Credentials are a crucial part of the authentication process.
- **Model (Machine Learning):** An algorithm that is used to predict on regression or classification tasks.
- **Unit:** The smallest possible software component.
- **Recall:** A model classification metric that focuses on number of false negative predictions (Type 2 error also known as Beta).
- **Precision:** A classification metric that focuses on false positive prediction (Type 1 error also known as Alpha).

| UserService Class | |
|--|---|
| public User createUser(User newUser); | The method that creates and save a user entity to the database. |
| public List<User> getAllUsers(); | The method that returns all of the user entities from the database. |
| public User getUser(int userId); | The method to find a user by the user id field of the user. |
| public boolean deleteUser(int userId) | The method to delete the user with the specified user id. |
| boolean updateAssistantQueries(int id, List<AssistantQuery> assistantQueryList) | Update the list of assistant queries for a user with a new list of assistant queries. |
| List<AssistantQuery> getAssistantQueries(int id); | Get the assistant queries of the user with the given user id. |

Appendix A: The UserService Class Methods

| AssistantQueryService Class | |
|--|--|
| public AssistantQueryDto createAssistantQuery(AssistantQuery assistantQuery); | Create new assistant query entity with the given assistant query. |
| public List<AssistantQueryDto> getAllQueries(); | Return the list of all the assistant query entities. |
| public AssistantQueryDto getQuery(int id); | The method for returning a specific assistant query with the given id. |
| boolean deleteAssistantQuery(int id); | Delete a specific assistant query with the specified assistant query id. |
| public User getUser (int id); | Get the user affiliated with the assistant query. |

Appendix B: The AssistantQueryService Class Methods

12. References

- Sommerville, I. (2016). Software Engineering, 10th edition. Pearson Education Limited. ISBN 10: 1-292-09613-6
- Martin, R.C. (2002). UML for Java Programmers. 1'st edition. Prentice Hall
- Bruegge, B et al. (2004). Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition. Pearson
- Crispin, L., & Gregory, J. (2009). Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley Professional.
- Sign Language Interpreter Dataset: <https://data.mendeley.com/datasets/xs6mvhx6rh/1>
- Hand Gestures Dataset: <https://universe.roboflow.com/yolo-zxvpk/hand-gesture-r7qgb/dataset/6>