National College of Ireland

Configuration Manual

MSc Research Project
Cybersecurity

Abbas Abdur Rahman
Student ID: x23162821

School of Computing
National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Abbas Abdur Rahman
Student ID:	X23162821
Programme:	MSc Cyber Security **Year:** 2023/24
Module:	Research in Computing
Supervisor:	Niall Heffernan
Submission Due Date:	12/08/2024
Project Title:	BehavioGuard: A Gesture-Based Authentication System for Mobile Applications.
Word Count:	4098 **Page Count** 19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Abbas Abdur Rahman
Date:	11/08/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Abbas Abdur Rahman
Student ID: X23162821

1 Introduction

Thus, the BehavioGuard project marks a major evolution in the concept of securing the mobile applications as it uses behavioral biometrics to identify the gesture patterns of the user. This unique solution aims at the problem of increasing requirements for safe and unobtrusive identification, especially in cases where simple measures, such as passwords or fingerprints, will not suffice. This configuration manual will enable you to understand and implement all the settings of the BehavioGuard system in its entirety. It covers such areas as installation of development environment, data gathering and preparation, feature engineering, model training, and inclusion of a real-time machine learning model in a basic mobile application. The goal is thus to make all aspects of the system effective, while delivering security measures to an application, as well as making it integrate with the rest of the components without negative effects on user-friendliness. This is a **configuration manual** that provides extensive details of all the steps that need to be followed in order to deploy and incorporate BehavioGuard system in a real-time mobile application. Starting from the initial state and establishment of the development environment up to the fine-tuning of the integrated machine learning model all the aspects of the configuration are included. This manual ensures that every single part of Firebase such as the protection of data to the TensorFlow Lite model for real time, gesture recognition functions in perfect coordination to deliver a stable, fluid and highly useful user experience. Thus, by following the given guide, you will be prepared to install high-quality security for the application that will increase user confidence and set a high bar for mobile application security.

2 System Requirements

In order to obtain effective functionality of the BehavioGuard system, several requirements both in the hardware and in the software should be met.

2.1 Architecture:

The accent in the design of the BehavioGuard system is made on unobtrusive integration of mobile application protection with the usage of machine learning for identification of users by behavioral biometrics. A few of them are largely integrated to each other with the aim of recording, deciphering, and analyzing user gesture data. The front end of the system is the mobile application through which the user communicates and the interaction data is captured real-time using the gesture detection modules. These interactions are optimized locally before data transmission for efficiency to the cloud services such as Firebase for secure data storage and synchronization. The core of the solution is the machine learning layer that is fed with the acquired data to authenticate users with the help of a neural network model trained in TensorFlow and optimized in TensorFlow Lite. This model processes in real-time on the mobile device hence the latency is kept to a minimum. There is also data security in the architecture through data encryption and secure access to the data by the users in the system. Consequently, with constant tracking and immediate decision making, the system is capable of identifying behavioral shifts and immediately establish and maintain users' authentication and security. Sca-ability directly addresses the problems of construction through the use of cloud

services of Firebase cloud services to provide the needed capacity when users of the system increases or add more features to the system.

2.2 Hardware Requirements:

As for the specifications of the device which should run the mobile application containing BehavioGuard, the application is best used on a device with at least 2GB of RAM, and a CPU speed of at least the ARM Cortex-A53. By such arrangement, it becomes easy and efficient to perform the gesture detection operations as well as the model inference operations. However, if the user plans to push the performance of his device especially for the most demanding and intricate tasks, it is recommended that he has a device that has at least 4GB of RAM and Qualcomm Snapdragon 855. These specifications will allow the app to process data input in real-time, and make model prediction without lagging or stalling while performing the authentication by keeping the integrity and responsiveness high.

2.3 Software Requirements:

In particular, the. AndroidStudio, TensorFlow, and Firebase are the fundamental pillars of BehavioGuard for they build the solid software base for it. Android Studio IDE must be the latest stable build, to support newer released Android SDKs and tools. Python 3.7 or later is required to execute the machine learning scripts which the neural network models in the system are derived from. Additionally, TensorFlow 2. x is needed to build the model and as well as for converting it into TensorFlow Lite whereas Keras which is provided within TensorFlow, makes the process of creating deep learning models faster. Firebase Services and Authentication through FirebaseAuth and FirebaseStore is important for the app mostly when it comes to signing in or signing up a user. Finally, the last but not the least, data preprocessing and selection can be performed only if good preparation is done on the input data and for this purpose libraries like Scikit-learn, NumPy and Pandas have been helpful.

2.4 Cloud Services:

One of the key elements of the system, thus, lies in the employment of the cloud services. Firebase is also very vital in this kind of architecture; a user sign-in process with FirebaseAuth is scorable, while real-time high quality data storage with FirebaseFirestore is scalable. Also, these cloud-based solutions help to provide synchronization and updates for the data complex and inputs coming from the users such as the gesture data across the devices. Therefore, using the elements of Firebase, BehavioGuard has time-tested secure and fast authentication that does not degrade even in conditions that can be critical in terms of network performance.

2.4.1 Table 1: Hardware Components

Component	Specification	Purpose
Smartphone/Tablet	Android OS (7.0 or higher), Minimum 4GB RAM	Device for running the BehavioGuard mobile application
Computer/Workstation	Intel i5/i7 Processor, 8GB RAM, 500GB HDD/SSD	Used for development, model training, and debugging
External Storage Device	64GB USB Drive or External HDD	Backup and transfer of datasets and application files
Network Router	Standard Wi-Fi Router (2.4GHz/5GHz bands)	Provides internet connectivity for cloud services integration
Fingerprint Scanner	USB or Integrated (optional)	Comparison testing against traditional biometric systems

2.4.2 Table 2: Software Components

Component	Version	Purpose
Android Studio	4.1 or higher	IDE for developing the BehavioGuard mobile application
Firebase SDK	Latest	Enables cloud-based authentication and data storage
TensorFlow	2.4 or higher	Machine learning framework used for training the model
TensorFlow Lite	Latest	Optimized version of TensorFlow for mobile applications
Python	3.8 or higher	Programming language used for scripting and model development
Keras	2.4 or higher	Deep learning API for building the neural network
Pandas	Latest	Data manipulation library used in preprocessing
NumPy	Latest	Numerical computing library used in model training
Firebase Authentication	Latest	Manages user sign-in and sign-up operations
Firebase Firestore	Latest	Stores user data and gesture patterns securely in the cloud
Jupyter Notebook	Latest	Interactive environment for running code and documenting experiments

2.4.3 Table 3: Cloud Services

Service	Provider	Purpose
Firebase Authentication	Google Firebase	Secure user authentication and management
Firebase Firestore	Google Firebase	Cloud-based database for storing gesture data and user profiles
Firebase Cloud Messaging	Google Firebase	Push notifications and real-time updates
TensorFlow Model Hosting	Google Cloud	Deployment and management of TensorFlow models
Cloud Storage	Google Cloud	Backup and archival of user data and model versions

2.4.4 Table 4: System Specifications

Specification	Requirement
Minimum Android Version	7.0 (Nougat) or higher
Minimum RAM	4GB (Device), 8GB (Development Workstation)
Storage Requirement	200MB (App Installation), 2GB (Model Storage and Data)
Internet Connection	Required for cloud synchronization and authentication
Processor	ARM Cortex-A53 (Mobile Device), Intel i5 (Workstation)
Battery Life	Minimum 8 hours (for mobile device testing)
Screen Resolution	720p or higher (Mobile Device)

2.4.5 Table 5: Model Specifications

Component	Specification	Purpose
Neural Network Layers	3 Layers: Input, Hidden (128, 64 neurons), Output	Structure of the model used for user authentication
Activation Function	ReLU for hidden layers, Softmax for output	Introduces non-linearity and computes class probabilities
Optimizer	Adam	Optimizes model parameters during training
Loss Function	Sparse Categorical Cross-Entropy	Measures model performance
Training Data	80% of collected data	Data used to train the model
Validation Data	20% of collected data	Data used to validate the model
Model Format	TensorFlow Lite (.tflite)	Format optimized for mobile devices

3 Installation and Setup

The general implementing plan scope comprises the installation of the BehavioGuard system, which is made of several procedures, including the configuration of Firebase, the development environment, and the final feeding and training of the final Machine learning model.

3.1 Setting Up Firebase:

First and foremost, you need to create a Firebase project that will act as the backend of the developed mobile app called BehavioGuard. This project is built in the Firebase's web interface, here you will authenticate your Android application thus allowing it to access Firebase services. The next significant step is activating Firebase Authentication services to sign in and register users without local storing of sensitive data. Firestore, Firebase's NoSQL

database, are used to store user information and shared array of gesture patterns data. When faced with data such as users, gestures, as well as auth_logs, collections are made to ensure the data is organized properly. Last but not the least, configuration file (google-services. json) is downloaded from firebase and pasted in app folder in android studio to integrate the app with the firebase service.

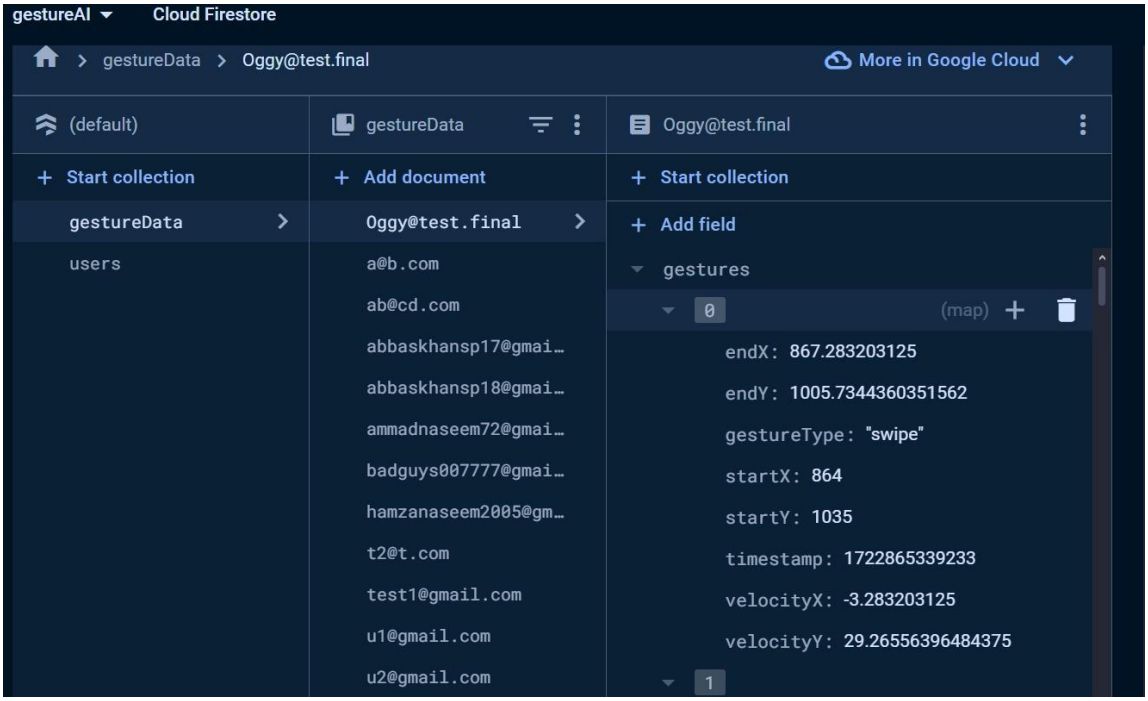

Figure 3.1.1 Code snippet Model Training

3.2 Development Environment Setup:

The most important step towards the setup of development environment is done through downloading of Android Studio in which all SDK tools and emulator images has to be downloaded in order to promote the development and testing. Python is installed, along with all necessary libraries, with the help of which I organize dependencies in the virtual environment. The BehavioGuard project is then checked out from the above URL and brought into the local computing environment through Git, including all the files related to Android programming and the Python scripts. Gradle which is the build tool used by Android Studio is used to handle the dependencies of projects. It also involves the correct library management, that is, correct referencing of the necessary libraries including the Firebase SDK and TensorFlow Lite in the build configuration files including the build.gradl.

3.3 Dataset Collection:

The dataset for gesture recognition was gathered in a methodical approach where the use of a mobile application which can record, different kinds of touch based gestures was installed. The respective collection process was initiated by installing the application under test on a range of devices and checking the permissions for touch and motion sensors. A further pool of participants was then chosen to carry out a set of purposely designed swipes, taps and other custom moves on the specific mobile devices.

Every participant was told to perform the gestures several times for them to capture the fact that people may display differences in their movements. While making each gesture, the application recorded a variety of information concerning the type of gesture, the time stamp

of the gesture, the starting and ending coordinates of gesture, velocities in X and Y directions. This data was stored in a structured format of a table where each gesture was a record in the database. The suite of collected data was further checked for the quality of data completeness and accuracy before being finalized for training from the machine learning model.

3.3.1 Basic Dataset Parameters

The dataset consists of the following parameters for each recorded gesture: The dataset consists of the following parameters for each recorded gesture:

Gesture Type: Enumerates the kind of gesture made, (for instance, swipe or tap).

Timestamp: The specific time at which the action was filmed.

Start X: The Y coordinate of the gesture's beginning.

Start Y: The Y coordinate of the first point where the gesture begins.

End X: in 'the End_X' The X coordinate of the gesture's ending point.

End Y: The Y coordinate of the gesture's ending point.

Velocity X: The component of the velocity in the X direction of the gesture.

Velocity Y: The magnitude of the speed of the gesture in the Y direction.

3.4 Model Training and Conversion:

Training of the machine learning model is an important part in the overall aspect of the BehavioGuard system. This is facilitated by the model. Bat file that runs the respective py script in the Python environment that is defined and configured. The script fits the neural network to the gesture data that has been preprocessed through a set of parameters to yield the best performance. After training is done, the model is saved in the TensorFlow format The following is an example of the model architecture: The converter. After that, using the py script the TensorFlow model is converted into TensorFlow Lite model format. This conversion is necessary to prepare the model to be deployed for mobile apps, making it lightweight, and less resource-intensive, and enables the mobile application to perform real-time predictions adequately.

model.py

```python
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

# Load your gesture data
data = pd.read_csv('gesture_data.csv')

# Preprocess the data
X = data[['startX', 'startY', 'endX', 'endY', 'velocityX', 'velocityY']].values
y = data['gestureType'].values

# Encode the labels
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# Split the data
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

model.py

```python
y = encoder.fit_transform(y)

# Split the data
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the model
model = keras.Sequential([
    keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(len(np.unique(y)), activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val))

# Save the model
model.save('gesture_model.h5')
```

Figure 3.3.3 Code snippet Model Training

3.5 Android Project Configuration:

The subsequent step for the Android project setting is done after the machine learning model is set up. This requires the project from which the BehavioGuard application is to be copied into the Android Studio, and confirm the right dependencies are resolved. The settings of the project are also Subject to revision, as the build options. Gradle files, to match by the current version of the SDK. Layout files are adjusted to fit the design and give the interface a friendly look while meeting the project's objectives. This a smooth flow through the different components of UI to enable the user easily to go through the gesture input and authentication.

4 Application Configuration

This mainly includes initial Firebase setting up which consists of the Firebase Authentication app, organizing the Firestore data management, including the TensorFlow Lite model, and the advanced UI polishing.

4.1 Firebase Authentication:

It harnesses the Firebase Authentication System which enables the use to register and also sign in the application securely. This is done in MainActivity. java file, where FirebaseAuth is declared and setup. In addition, error control for this user authentication process will also be embedded top-notch so that even if there is network problem or false input the system will be

able to manage it. The authentication flow is heavily integrated with Firebase Firestore which stores user credential as well as gesture data where each user gets their own data and thus can be easily retrieved and is secured.

4.2 Firestore Configuration:

Firestore also has a capability of storing the gesture data and user profile data in a structured and optimized way. Collections are maintained with respect to the user and each field also identifies the user information such as gesture_type, coordinates, timestamp and user_id. This can be to test real-time authentication that involves large amount of data and thereby indexes are created on important fields to enhance efficient data searches. To protect this data, security rules are put in place meaning that no other person can interact with or change another person's data without a proper identity. These rules are, therefore, reviewed and modified from time to time based on the current need in the field of security.

```
// Store captured gesture data to Firestore database
1 usage
private void storeGestureData() {
    if (gestureDataList.isEmpty()) {
        Toast.makeText( context: this,  text: "No gestures captured", Toast.LENGTH_SHORT).show();
        return;
    }

    Map<String, Object> gestureDataMap = new HashMap<>();
    gestureDataMap.put("gestures", gestureDataList);

    db.collection( collectionPath: "gestureData").document(userEmail).set(gestureDataMap)
            .addOnSuccessListener(aVoid -> Toast.makeText( context: MainActivity.this,
                    text: "Behaviour Captured! Training AI Model..", Toast.LENGTH_LONG).show())
            .addOnFailureListener(e -> Toast.makeText( context: MainActivity.this,
                    text: "Error storing gesture data", Toast.LENGTH_SHORT).show());
}
```

Figure 4.2.2 code snippet for storing gestures data to firebase

4.3 TensorFlow Lite Model Integration:

The TensorFlow-Lite model is included in the integrated model for mobile deployment which has been optimized for the same. This integration is performed in the MainActivity as it is defined in the previous section. java file, where the model is downloaded and it further uses TensorFlow Lite interpreter to load the model. Loading of a model should be made as asynchronous to avoid any hindrance that may be caused to the main UI thread hence causing the application to hang. Real-time processing of gestures is provided by a separate function which takes user input data and passes it through the model and the results of the prediction are returned. By integrating such a system, the latencies have been optimized and determined to cause minimum delays hence parameterizing the app to be able to authenticate its users in real time.

```
// Load TensorFlow Lite model from file
1 usage
private MappedByteBuffer loadModelFile(Context context) throws IOException {
    FileInputStream fileInputStream = new FileInputStream(context.getAssets().openFd( fileName: "model.tflite").getFileDescriptor());
    FileChannel fileChannel = fileInputStream.getChannel();
    long startOffset = context.getAssets().openFd( fileName: "model.tflite").getStartOffset();
    long declaredLength = context.getAssets().openFd( fileName: "model.tflite").getDeclaredLength();
    return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset, declaredLength);
}
```

Figure 4.3.3 TensorFlow lite Model Loading code snippet

4.4 User Interface Configuration:

Some components of the application include, and are not limited to, the user interface, which should be straightforward and interactive for the BehavioGuard application. MainActivity is created in a way of initializing Firebase components and real-time gesture capture. The UI provides interface components to instruct the users about how to make the gestures with downstream visual feedback information of the recognition status. Such feedback is critical for enhancing the user experience because users get a real-time confirmation of successful or unsuccessful authentication efforts.

5 Running the Application

Running the BehavioGuard application involves a series of tests to ensure that all components function as expected and that the system can accurately authenticate users based on their gesture inputs.

5.1 Testing the Application:

The first thing which you have to do in order to test this application is to place BehavioGuard. apk on some test device or, if you do not have physical device for testing, use Android Emulator integrated in Android Studio. The application at first Setup should not show any error with Firebase and any of the services such as Authentication, FirebaseFirestore etc should be fully functional when the application is launched. Sign up and sign in features are tested so as to confirm that multiple users can sign up and whether the data will be stored in Firestore in the correct way. Once the user signs in, the app goes to the gesture capture screen where all the user's gestures are logged in real-time.

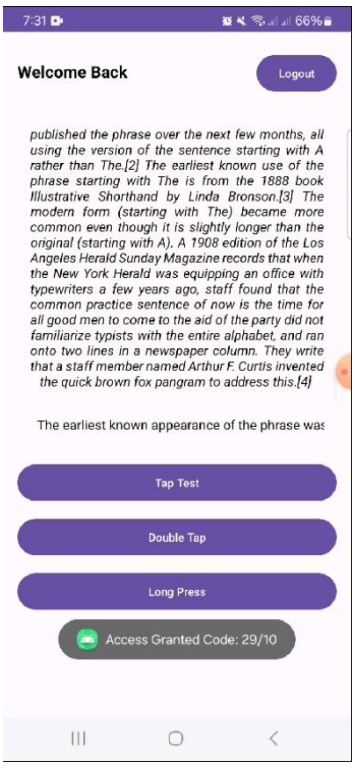

Figure 5.1.1 Authorized User Access Granted

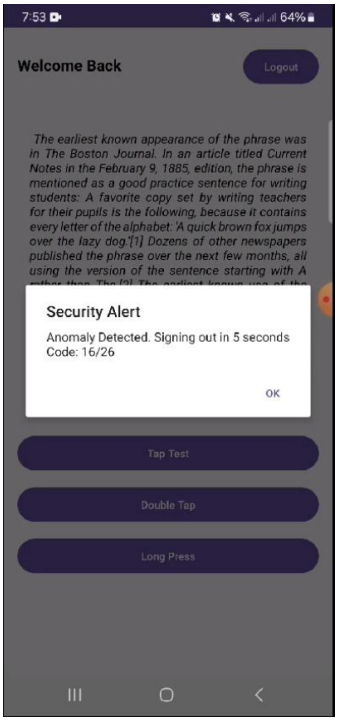

Figure 5.1.1 Anomaly detected UI Automatically Logging Out in 5 sec

5.2 User Authentication Flow:

They drew the user authentication flow starting from time when the user logs in. I precise touching movements as swiping, tapping and long pressing of your fingers, after which the TensorFlow Lite model authenticates it. The concepts of identity comparison of such gestures with the data previously saved in the app remains the key priority of the app as well. The match involves calculation of a match percentage and the decision as to whether the current gestures correspond to the stored pattern. The switching between the different states that include idle state, capturing state, and the authentication state is intended to be swift, and without much delay so that when the user is at the point of authentication, there will not be a long waiting period.

6 Model Saving and Conversion

6.1 Saving the Model:

Once in a while in the process of successful training of the neural network model, the model should be saved in TensorFlow format. This step saves the structure of the model and floats for it, such that the trained model can be used each time without needed to be trained again. Saving the model also helps in tweaking and improving the model so that developers can make adjustments to the particular model with their different logging configurations without loss of the core model.

6.2 Conversion to TensorFlow Lite:

The trained TensorFlow model is then converted to TensorFlow Lite which is pivotal for deploying a model on a mobile phone. TensorFlow Lite is designed to be used in phones and other similar devices and it has smaller models and faster conclusions. This procedure applies quantization, whereby the precision of the model weights is lowered and results in a considerable reduction of the model's size, though accuracy is also slightly affected. The

resulting . tflite file is further incorporated into the load asset of the applications and can be implemented in real time.

converter.py

```python
# Load the Keras model
model = tf.keras.models.load_model('gesture_model.h5')

# Convert the model to TFLite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the TFLite model
with open('gesture_model.tflite', 'wb') as f:
    f.write(tflite_model)
```

Figure 5.1.1 TensorFlow Model conversion code snippet

7 Application Integration

7.1 SignIn and SignUp Activities:

The SignIn and SignUp activities are developed using FirebaseAuth as it is easy to implement and also it is secured. FirebaseAuth is responsible for handling of the user credentials such as the email and password as well as Firebase authentication service for integration with FirebaseFirestore where the user information and gesture patterns shall be kept. This setup ensures that all the users' data are well secured while at the same time ensuring that such data can be easily retrieved especially when it comes to the authentication processes.

```java
// Sign in user with Firebase Auth
1 usage
private void signInUser(String email, String password) {
    progressBar.setVisibility(View.VISIBLE);

    mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener( activity: this, task -> {
                progressBar.setVisibility(View.GONE);
                if (task.isSuccessful()) {
                    FirebaseUser firebaseUser = mAuth.getCurrentUser();
                    if (firebaseUser != null) {
                        fetchUserData(firebaseUser.getUid(), email);
                    }
                } else {
                    Toast.makeText( context: this,  text: "Authentication failed: "
                            + task.getException().getMessage(), Toast.LENGTH_SHORT).show();
                }
            });
}
```

Figure 7.1.1 Authentications Code Snippet

```java
// Sign up user with Firebase Auth
1 usage
private void signUpUser(String user, String email, String password) {
    progressBar.setVisibility(View.VISIBLE);

    mAuth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener( activity: this, task -> {
                progressBar.setVisibility(View.GONE);
                if (task.isSuccessful()) {
                    FirebaseUser firebaseUser = mAuth.getCurrentUser();
                    if (firebaseUser != null) {
                        saveUserData(firebaseUser.getUid(), user, email);
                    }
                } else {
                    Toast.makeText( context: this,  text: "Authentication failed: "
                            + task.getException().getMessage(), Toast.LENGTH_SHORT).show();
                }
            });
}
```

Figure 7.1.1 Auth Code Snippet

7.2 MainActivity Implementation:

MainActivity is the main part of actual BahavioGuard application; it contains all initialization of Firebase components and detecting/processing gestures. This activity records the user's gestures as they are performed and analyses them using the TensorFlow Lite model. The gestures are then compared with stored data in FirebaseFirestore to check if the user is valid or not. This process is made in a way that they optimise a lot of time so that users can have little delay time while they are authenticating.

```java
// Start capturing gestures for training the model
1 usage
private void startGestureCapturing() {
    gestureDataList.clear();
    isCapturingGestures = true;
    Toast.makeText( context: this,  text: "Gesture capturing started! Use App Normally for 30 Seconds.", Toast.LENGTH_LONG).show();

    // Stop capturing gestures after 30 seconds
    stopGestureCaptureRunnable = this::stopGestureCapturing;
    gestureCaptureHandler.postDelayed(stopGestureCaptureRunnable,  delayMillis: 30000); // Stop after 30 seconds
}
```

Figure 7.2.1 Code Snippet for capturing gestures for training the model

```
// Gesture listener for capturing various gestures
1 usage
private class GestureListener extends GestureDetector.SimpleOnGestureListener {
    @Override
    public boolean onDown(MotionEvent e) {
        if (isCapturingGestures) {
            gestureDataList.add(new GestureData( gestureType: "down", System.currentTimeMillis(),
                    e.getX(), e.getY(),  endX: 0,  endY: 0,  velocityX: 0,  velocityY: 0));
        }
        return true;
    }

    @Override
    public boolean onSingleTapUp(MotionEvent e) {
        if (isCapturingGestures) {
            gestureDataList.add(new GestureData( gestureType: "tap", System.currentTimeMillis(),
                    e.getX(), e.getY(),  endX: 0,  endY: 0,  velocityX: 0,  velocityY: 0));
        }
        return true;
    }
```

Figure 7.2.2 Code Snippet for capturing gestures of different gestures

```
// Compare new gesture data with stored gesture data
1 usage
private void compareGestureData(List<GestureData> newGestureData) {
    db.collection( collectionPath: "gestureData").document(userEmail).get()
            .addOnSuccessListener(documentSnapshot -> {
                if (documentSnapshot.exists()) {
                    List<Map<String, Object>> storedGestureDataMap = (List<Map<String, Object>>) documentSnapshot.get("gestures");
                    List<GestureData> storedGestureData = new ArrayList<>();
                    for (Map<String, Object> map : storedGestureDataMap) {
                        storedGestureData.add(new GestureData(
                                (String) map.get("gestureType"),
                                (long) map.get("timestamp"),
                                ((Number) map.get("startX")).floatValue(),
                                ((Number) map.get("startY")).floatValue(),
                                ((Number) map.get("endX")).floatValue(),
                                ((Number) map.get("endY")).floatValue(),
                                ((Number) map.get("velocityX")).floatValue(),
                                ((Number) map.get("velocityY")).floatValue()
                        ));
                    }
```

Figure 7.2.3 Code Snippet for compares the new data with stored gestures data

```java
// Calculate match percentage between stored and new gesture data
1 usage
private float calculateMatchPercentage(List<GestureData> storedData, List<GestureData> newData) {
    if (newData.isEmpty()) { return 0.0f; // Avoid division by zero
    }
    int matchCount = 0;
    for (GestureData newGesture : newData) {
        boolean isMatched = false;
        for (GestureData storedGesture : storedData) {
            if (isGestureMatch(storedGesture, newGesture)) {
                matchCount++;
                Log.d( tag: "GESTURE MATCHED", msg: "Matched");
                isMatched = true;
                break; // Move to the next newGesture once a match is found
            }
        }
        // Optionally, handle cases where no match was found for a newGesture
        if (!isMatched) {
            Log.d( tag: "MATCHEDGESTURE",  msg: "Gesture Not Matched!");
        }
    }

    return ((float) matchCount / newData.size()) * 100;
}
```

Figure 7.2.4 Code Snippet for calculating match percentage between stored and new gestures

```java
// Check if two gestures match
1 usage
private boolean isGestureMatch(GestureData storedGesture, GestureData newGesture) {
    // Define increased tolerance values for matching criteria
    final float COORDINATE_TOLERANCE = 400.0f; // Tolerance for coordinates
    final float DIRECTION_TOLERANCE = 250.0f;  // Tolerance for direction in degrees
    final float LENGTH_TOLERANCE = 400.0f;     // Tolerance for length
    final float VELOCITY_TOLERANCE = 5000.0f; // Tolerance for velocity
    boolean isMatch = storedGesture.getGestureType().equals(newGesture.getGestureType());
    if (isMatch) {
        switch (storedGesture.getGestureType()) {
            case "down":
            case "tap":
                // Coordinate-based matching
                isMatch = isWithinBoundingBox(storedGesture.getStartX(), storedGesture.getStartY(),
                        newGesture.getStartX(), newGesture.getStartY(), COORDINATE_TOLERANCE);
                break;
            case "long_press":
                // Coordinate-based matching with a longer press
                isMatch = isWithinBoundingBox(storedGesture.getStartX(), storedGesture.getStartY(),
                        newGesture.getStartX(), newGesture.getStartY(), COORDINATE_TOLERANCE);
                break;
            case "swipe":
                // Coordinate-based matching for start and end points, and feature-based matching
                isMatch = isWithinBoundingBox(storedGesture.getStartX(), storedGesture.getStartY(),
```

Figure 7.2.4 Code Snippet whether two datasets matches

7.3 Real-Time Authentication:

This last point is one of the most important because through the real-time authentication the app is constantly checking and validating user gestures. The Android system employs a GestureDetector to capture input related to various gestures including measures like swipes, taps, and long press to be processed. The TensorFlow Lite model is then employed for post-

gesture analysis, and in real time, the application determines the status of being authenticated or not. It means that clients are monitored in real time and there is no possibility for an unauthorized person to have access to the application, which is very significant.

8 Final Testing and Validation

8.1 Extensive Testing:

Substantial testing is thus made prior the last deployment of the BehavioGuard application, testing that seeks to identify and correct any component faults, as well as testing that seeks to determine the overall efficiency of the system in meeting the set accuracy and performance standards. The testing process consists of checking the ability of the app to record and recognize gestures, the accuracy of the TensorFlow Lite model predictions and functionality of the Firebase support. In case there is a problem with a program or application during testing, it is fixed by debugging and finally optimized so that users can have the best experience with the final product.

8.2 Validation of Accuracy and Performance:

The effectiveness of the solutions given by the BehavioGuard system can be recognized when the outcomes of the model are compared to the real user information. The validation process to confirm such system involves the use of as many users as possible and numerous gestures to confirm that the system can be able to identify the user well. Other key factors including system response time, usability, and security are also audited to guarantee the system corresponds to real-time mobile application systems' stringent standards.

9 User Data Comparison and Security

9.1 User Data Management:

User data is stored and accessed when needed from FirebaseFirestore storage with the focus on the date of registration. To address this issue, the system is built to warrant protection of user data such that the unauthorized users cannot access any users' data, but only the users with Firebase Auth credentials. This setup guarantees that the system follows the right practices concerning data management and security offering a surety of the privacy of the individual data.

9.2 Gesture Comparison and Authentication:

This goes on as the system analyzes new gestures against the stored data with regard to the tensorflow lite model to determine whether the gestures are in sync with the user's defined patterns. The system then assigns each gesture a match percentage, on which the identification of the user, based on tolerance, is done. This comparison process is intended to be very precise so that the system will be able to reliably separate the legitimate users from the impostors.

10 Maintenance and Updates

10.1 Ongoing Maintenance:

The general management of the BehavioGuard system includes constant updating of the machine learning model, firebase settings and the app modules as well. As new data is acquired, the model may have to be updated for the system to reflect the changes in the use patterns and increase the efficiency of its performance. Firebase rules and permission settings should also have to be set appropriately and have to be checked from time to time with the current security standards.

10.2 Future Enhancements:

Possible future development of BehavioGuard includes having more inputs from several other biometric indicators like gait, face or voice files, to enhance the protection level. Other possible enhancements include, the adaptation of the system on low-end devices, the updating of the UI/UX, and extension of the guidelines for more intricate authentication solutions.

11 Glossary

Behavioral Biometrics: The type of user identification that takes into account the user's signature in the behaviour such as click, mouse movments, typing pattern on the keyboard.

Gesture Recognition: Communicating with the help of devices in the informal language with materials that can imitate the human gestures.

Neural Network: An imitation of the human mini-brains called artificial neural networks specially designed to acquire patterns.

TensorFlow: A machine learning toolbox, initiated by Google and is opensource.

TensorFlow Lite: The new version TensorFlow mobil specially designed for the mobile and portable operating systems the most familiar of which is Android.

ReLU (Rectified Linear Unit): Non-linear function which is commonly used in the hidden layers of the neural networks to present non-linearity.

Adam Optimizer: A method that is used in training of neural networks and in instance of adjusting learning rates.

Firebase: Mainly involve in the provision of company and service mainly in the provision of cloud solutions for mobile and web applications.

Firebase Authentication (FirebaseAuth): An application that relates to the processes of users' identification in the course of operating the service.

Firebase Firestore: An Example: A NoSQL cloud database for data depository and real time synchronisation of the data.

Preprocessing: The procedure in data preprocessing where a noble feature is removed, or missing values are filled in the raw data.

Model Inference: Where there is the use of a trained model on new data in other to predict something or make some form of a forecast.

Latency: This is the time to complete a round in a system from the input and to the output.

Swipe Gesture: A move with the hand and a finger on looking at the touch screen as a controller.

Tap Gesture: A light touch on an object usually of a screen with an aim of choosing or changing something.

12 Acronyms

ML: Machine Learning

AI: Artificial Intelligence

ReLU: Rectified Linear Unit

API: Application Programming Interface

SDK: Software Development Kit

ID: Identification

NoSQL: Not Only SQL

CSV: Comma-Separated Values

UI: User Interface

RAM: Random Access Memory

HDD: Hard Disk Drive

SSD: Solid State Drive

OS: Operating System

IDE: Integrated Development Environment

JSON: JavaScript Object Notation