.1 Dart Code Listings

.1.1 Firebase Configuration

```
[language=Dart, numbers=left, numberstyle=\tiny,
   frame=single, caption=Firebase Configuration]
// File generated by FlutterFire CLI.
// ignore_for_file: type=lint
import 'package:firebase_core/firebase_core.dart' show
   FirebaseOptions;
import 'package:flutter/foundation.dart'
show defaultTargetPlatform, kIsWeb, TargetPlatform;
/// Default [FirebaseOptions] for use with the Firebase
   apps.
///
/// Example:
/// dart /// import 'firebase_options.dart'; /// // ... ///
   await Firebase.initializeApp( ///   options:
   DefaultFirebaseOptions.currentPlatform, /// ); ///
class DefaultFirebaseOptions {
static FirebaseOptions get currentPlatform {
if (kIsWeb) {
return web;
}
switch (defaultTargetPlatform) {
case TargetPlatform.android:
return android;
case TargetPlatform.iOS:
return ios;
case TargetPlatform.macOS:
return macos;
case TargetPlatform.windows:
return windows;
case TargetPlatform.linux:
throw UnsupportedError(
'DefaultFirebaseOptions have not been configured for linux
   - '
'one can reconfigure this by running the FlutterFire CLI
   again.',
);
default:
throw UnsupportedError(
'DefaultFirebaseOptions are not supported for this
   platform.',
);
}
}
static const FirebaseOptions web = FirebaseOptions(
apiKey: 'AIzaSyBF1fTptjn0WttfTXI1PUGZRP8nMDungqk',
appId: '1:775148552695:web:00aea9ea25db636fb26ef2',
messagingSenderId: '775148552695',
projectId: 'beast-one',
authDomain: 'beast-one.firebaseapp.com',
storageBucket: 'beast-one.appspot.com',
measurementId: 'G-8T0FPR81JT',
);
static const FirebaseOptions android = FirebaseOptions(
apiKey: 'AIzaSyDVVXLkh10GoXpdF3YsJEvygMoiptkx1BQ',
appId: '1:775148552695:android:726ea5633a899007b26ef2',
messagingSenderId: '775148552695',
projectId: 'beast-one',
storageBucket: 'beast-one.appspot.com',
);
static const FirebaseOptions ios = FirebaseOptions(
apiKey: 'AIzaSyDtypi69lMQSzPp-sSMe-nM1eoVMvyC1bE',
appId: '1:775148552695:ios:9d8151028a8b65e5b26ef2',
messagingSenderId: '775148552695',
projectId: 'beast-one',
storageBucket: 'beast-one.appspot.com',
androidClientId:
  '775148552695-\\fbaopaq359abpmkgb6kojjaaqvu54vfe.
\\apps.googleusercontent.com',
iosClientId:
  '775148552695-\\9ibs7ggv41tsh32bfm6465ukbs44p2gi.apps.
\\googleusercontent.com',
iosBundleId: 'com.example.sensordata',
);
static const FirebaseOptions macos = FirebaseOptions(
apiKey: 'AIzaSyDtypi69lMQSzPp-sSMe-nM1eoVMvyC1bE',
appId: '1:775148552695:ios:9d8151028a8b65e5b26ef2',
messagingSenderId: '775148552695',
projectId: 'beast-one',
storageBucket: 'beast-one.appspot.com',
androidClientId: 'id.googleusercontent.com',
iosClientId: 'id.apps.\\googleusercontent.com',
iosBundleId: 'com.example.sensordata'\\,
);
static const FirebaseOptions windows = FirebaseOptions(
apiKey: 'AIzaSyBF1fTptjn0WttfTXI1PUGZRP8nMDungqk',
appId: '1:775148552695:web:78337568dbfee89fb26ef2',
messagingSenderId: '775148552695',
projectId: 'beast-one',
authDomain: 'beast-one.firebaseapp.com',
storageBucket: 'beast-one.appspot.com',
measurementId: 'G-78D65ET41M',
);
}
```

Explanation:

- Lines 1-5: Import necessary packages for Firebase and platform detection.
- Lines 7-14: Documentation for using the DefaultFirebaseOptions class.
- Lines 15-38: Define the DefaultFirebaseOptions class with a getter for the current platform's options.

- Lines 40-48: Define Firebase options for web platform.
- Lines 50-56: Define Firebase options for Android platform.
- Lines 58-68: Define Firebase options for iOS platform.
- Lines 70-80: Define Firebase options for macOS platform.
- Lines 82-91: Define Firebase options for Windows platform.

.1.2 Supabase Configuration

```
[language=Dart, numbers=left, numberstyle=\tiny,
   frame=single, caption=Supabase Configuration]
class SupabaseCread {
final String url =
   'https://xnlvzblwtolrtfyrvgta.supabase.co';
final String key = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.\\
eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InhubHZ6Ymx3dG9scnRmeXJ2Z3Rh\\
Iiwicm9sZSI6ImFub24iLCJpYXQiOjE3MTc3MzczNDEsImV4cCI6MjAzMzMxMz\\
M0MX0.quxKIbnkEoCjDdX9K_C7r-4TL25qeGJR17igFSXeuug';
}
```

Explanation:

- Line 1: Define the SupabaseCread class.
- Line 2: Declare a final string variable 'url' with the Supabase project URL.
- Line 3: Declare a final string variable 'key' with the Supabase API key (JWT token).

```
[language=Dart, numbers=left, numberstyle=\tiny,
   frame=single, caption=Firebase Configuration]

import 'dart:async';
import 'dart:io';

import 'package:flutter/foundation.dart';
```

~~~
import 'package:flutter/material.dart';
import 'package:path_provider/path_provider.dart';
import 'package:sensors_plus/sensors_plus.dart';
import 'package:supabase_flutter/supabase_flutter.dart';
import 'package:sensordata/auth_gate.dart';
‘‘‘


- **Imports:**
  - ‘dart:async‘ and ‘dart:io‘: Dart core libraries for
    asynchronous operations (‘Future‘, ‘Timer‘, etc.) and
    file system access (‘File‘, ‘Directory‘).
  - ‘flutter/foundation.dart‘: Flutter library for
    platform-specific functionality (‘kIsWeb‘,
    ‘kDebugMode‘, ‘defaultTargetPlatform‘).
  - ‘flutter/material.dart‘: Flutter library for building
    UI using widgets (‘MaterialApp‘, ‘Scaffold‘, ‘AppBar‘,
    etc.).
  - ‘path_provider/path_provider.dart‘: Flutter plugin for
    accessing device directories
    (‘getExternalStorageDirectory‘).
  - ‘sensors_plus/sensors_plus.dart‘: Flutter plugin for
    accessing device sensors (‘accelerometerEvents‘,
    ‘gyroscopeEvents‘, ‘magnetometerEvents‘).
  - ‘supabase_flutter/supabase_flutter.dart‘: Supabase
    Flutter SDK for interacting with Supabase services.
  - ‘sensordata/auth_gate.dart‘: Assuming it contains
    authentication-related logic.


‘‘‘dart
Future<void> main() async {
  await Supabase.initialize(
      url: SupabaseCread().url, anonKey:
         SupabaseCread().key);
  runApp(const MyApp());
~~~

```
}
‘‘‘


- **Main Function:**
  - ‘main()‘: Entry point of the application.
  - ‘Supabase.initialize‘: Initializes Supabase with
    provided URL and anonymous key.
  - ‘runApp(const MyApp())‘: Runs the Flutter application
    using ‘MyApp‘ as the root widget.

‘‘‘dart
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'SensorData',
      home: MyWidget(),
    );
  }
}
‘‘‘


- **MyApp Class:**
  - ‘MyApp‘: Stateless widget representing the root of the
    application.
  - ‘MaterialApp‘: Root widget of the Flutter application,
    providing material design.
  - ‘MyWidget()‘: Initial screen of the application.

‘‘‘dart
class MyWidget extends StatefulWidget {
  const MyWidget({super.key});
  @override
  State<MyWidget> createState() => _MyWidgetState();
}
‘‘‘

- **MyWidget Class:**
  - ‘MyWidget‘: Stateful widget representing the initial
     screen.
  - ‘State<MyWidget>‘: State class associated with
     ‘MyWidget‘.

‘‘‘dart
class _MyWidgetState extends State<MyWidget> {
  String? _user;

  @override
  void initState() {
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(’Login Portal’),
      ),
      body: _user == null ? const _LoginForm() : HomePage(),
    );
  }
}
‘‘‘

- **_MyWidgetState Class:**
```

```
  - ‘_MyWidgetState‘: State class for ‘MyWidget‘.
  - ‘initState()‘: Lifecycle method called when the
    stateful widget is initialized.
  - ‘Scaffold‘: Provides the basic material design visual
    layout structure.
  - ‘_user‘: Holds the current user’s state.
  - ‘_LoginForm()‘: Displays a login form if ‘_user‘ is
    ‘null‘; otherwise, navigates to ‘HomePage‘.

‘‘‘dart
class _LoginForm extends StatefulWidget {
  const _LoginForm();

  @override
  State<_LoginForm> createState() => _LoginFormState();
}
‘‘‘

- **_LoginForm Class:**
  - ‘_LoginForm‘: Stateful widget for the login form.
  - ‘_LoginFormState‘: State class associated with
    ‘_LoginForm‘.

‘‘‘dart
class _LoginFormState extends State<_LoginForm> {
  bool _loading = false;
  final _usernameController = TextEditingController();
  final _passwordController = TextEditingController();

  @override
  void dispose() {
    _usernameController.dispose();
    _passwordController.dispose();
    super.dispose();
```

```
}

Future<void> _login() async {
  setState(() {
    _loading = true;
  });
  final username = _usernameController.text;
  final password = _passwordController.text;

  try {
    final response = await Supabase.instance.client
        .from('users')
        .select('username')
        .eq('username', username)
        .eq('password', password);

    if (kDebugMode) {
      print(response);
    }

    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Login successful'),
        backgroundColor: Colors.green,
      ),
    );

    Navigator.pushReplacement(
      context,
      MaterialPageRoute(builder: (context) => HomePage()),
    );
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
          content: Text('Login failed: $e'),
          backgroundColor: Colors.red,
        ),
      );
    } finally {
      setState(() {
        _loading = false;
      });
    }
  }

  Future<void> _signUp() async {
    setState(() {
      _loading = true;
    });
    final username = _usernameController.text;

    try {
      final existingUserResponse = await
         Supabase.instance.client
          .from('users')
          .select()
          .eq('username', username);

      if (kDebugMode) {
        print(existingUserResponse);
      }

      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('Username already exists'),
          backgroundColor: Colors.red,
        ),
      );
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Signup failed: $e'),
        backgroundColor: Colors.red,
      ),
    );
  } finally {
    setState(() {
      _loading = false;
    });
  }
}

@override
Widget build(BuildContext context) {
  return _loading
      ? const Center(child: CircularProgressIndicator())
      : ListView(
          padding: const EdgeInsets.symmetric(horizontal:
             16, vertical: 20),
          children: [
            TextFormField(
              controller: _usernameController,
              decoration: const
                 InputDecoration(labelText: 'Username'),
            ),
            const SizedBox(height: 16),
            TextFormField(
              obscureText: true,
              controller: _passwordController,
              decoration: const
                 InputDecoration(labelText: 'Password'),
            ),
              const SizedBox(height: 16),
              ElevatedButton(
                onPressed: _login,
                child: const Text('Login'),
              ),
              const SizedBox(height: 16),
              TextButton(
                onPressed: _signUp,
                child: const Text('Signup'),
              ),
              TextButton(
                onPressed: () async {
                  await Supabase.instance.client.auth
                      .signInWithOAuth(OAuthProvider.google);
                },
                child: const Text('Google sign'),
              ),
            ],
          );
  }
}
‘‘‘


- **_LoginFormState Class:**
  - ‘_LoginFormState‘: State class for ‘_LoginForm‘.
  - ‘_loading‘: Tracks whether the login/signup process is
     ongoing.
  - ‘_usernameController‘ and ‘_passwordController‘:
     Controllers for username and password input fields.
  - ‘_login()‘: Attempts to log in using Supabase
     credentials.
  - ‘_signUp()‘: Attempts to sign up a new user using
     Supabase credentials.
  - ‘build(BuildContext context)‘: Builds the UI for the
```

```
     login form.

‘‘‘dart
class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  List<String> accelerometerData = [];
  List<String> gyroscopeData = [];
  List<String> magnetometerData = [];
  Timer? _timer;

  @override
  void initState() {
    super.initState();
    _startSensorDataCollection();
    _timer = Timer.periodic(const Duration(seconds: 1),
       (Timer t) {
      _saveDataToCSV(accelerometerData, 'accelerometer');
      _saveDataToCSV(gyroscopeData, 'gyroscope');
      _saveDataToCSV(magnetometerData, 'magnetometer');
    });
  }

  @override
  void dispose() {
    _timer?.cancel();
    _stopSensorDataCollection();
    super.dispose();
  }

  void _startSensorDataCollection() {
```

```
  accelerometerEvents.listen((event) {
    _addSensorData('accelerometer', event.x, event.y,
       event.z);
  });
  gyroscopeEvents.listen((event) {
    _addSensorData('gyroscope', event.x, event.y,
       event.z);
  });
  magnetometerEvents.listen((event) {
    _addSensorData('magnetometer', event.x, event.y,
       event.z);
  });
}

void _stopSensorDataCollection() {
  accelerometerData.clear();
  gyroscopeData.clear();
  magnetometerData.clear();
}

void _addSensorData(String sensorType, double x, double
   y, double z) {
  final now = DateTime.now().toIso8601String();
  final dataPoint = '$sensorType,$now,$x,$y,$z';
  setState(() {
    if (sensorType == 'accelerometer') {
      accelerometerData.add(dataPoint);
    } else if (sensorType == 'gyroscope') {
      gyroscopeData.add(dataPoint);
    } else if (sensorType == 'magnetometer') {
      magnetometerData.add(dataPoint);
    }
  });
}
```

```
  Future<void> _saveDataToCSV(List<String
> data, String sensorType) async {
    final directory = await _createFolder();
    final file = File('${directory.path}/$sensorType.csv');
    final csvData = data.join('\n');
    await file.writeAsString(csvData);
    if (kDebugMode) {
      print('Sensor data saved to ${file.path}');
    }
  }

  Future<Directory> _createFolder() async {
    final directory = await getExternalStorageDirectory();
    const folderName = 'SensorData';
    final folder =
       Directory('${directory!.path}/$folderName');
    if (!folder.existsSync()) {
      folder.createSync(recursive: true);
    }
    return folder;
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Sensor Data Collector'),
        actions: [
          IconButton(
            icon: const Icon(Icons.logout),
            onPressed: () async {
              Navigator.pushReplacement(
                context,
                MaterialPageRoute(builder: (context) =>
                  const MyWidget()),
              );
            },
          ),
        ],
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
              onPressed: () =>
                  _saveDataToCSV(accelerometerData,
                     'accelerometer'),
              child: const Text('Save Accelerometer Data'),
            ),
            ElevatedButton(
              onPressed: () =>
                 _saveDataToCSV(gyroscopeData, 'gyroscope'),
              child: const Text('Save Gyroscope Data'),
            ),
            ElevatedButton(
              onPressed: () =>
                 _saveDataToCSV(magnetometerData,
                 'magnetometer'),
              child: const Text('Save Magnetometer Data'),
            ),
          ],
        ),
      ),
    );
  }
```

```
}
‘‘‘

- **HomePage Class:**
  - ‘HomePage‘: Stateful widget representing the home page
     after login.
  - ‘_HomePageState‘: State class for ‘HomePage‘.
  - Initializes sensor data collection
     (‘_startSensorDataCollection‘) and periodic data
     saving (‘_timer‘).
  - ‘dispose()‘: Cancels the timer and clears sensor data
     when the widget is disposed.
  - ‘_saveDataToCSV()‘: Saves sensor data to CSV files on
     the device storage.
  - ‘build(BuildContext context)‘: Builds the UI for the
     home page, displaying buttons to save sensor data and
     logout.

### Summary:
This Dart code integrates Supabase for user authentication,
   Flutter for building UI, and device sensors for
   collecting data. It includes login/signup functionality,
   sensor data collection, saving data to CSV files, and
   basic navigation between screens. Each class and method
   contributes to the overall functionality of the
   application, ensuring proper management of state, UI
   rendering, sensor data handling, and data persistence.
```

Appendix: Code for Data Processing and Model Training

The following code demonstrates the process of data loading, preprocessing, and training a deep learning model using TensorFlow's Keras API. This code is designed to be added to the appendix of the thesis for detailed reference.

Listing 1: IoT Forensics Data Processing and Deep Learning Model

```
\begin{verbatim}
import os
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Define paths to CSV files
accel_file = r'G:\My Drive\Ninad data\Projects\2117
    Improving the efficiency of IOT forensics using Deep
    learning\finalcsvfiles\accelerometer.csv'
gyro_file = r'G:\My Drive\Ninad data\Projects\2117
    Improving the efficiency of IOT forensics using Deep
    learning\finalcsvfiles\gyroscope.csv'
mag_file = r'G:\My Drive\Ninad data\Projects\2117 Improving
    the efficiency of IOT forensics using Deep
    learning\finalcsvfiles\magnetometer.csv'

# Load CSV files into Pandas DataFrames
accel_data = pd.read_csv(accel_file)
gyro_data = pd.read_csv(gyro_file)
mag_data = pd.read_csv(mag_file)

# Assuming the data is structured with columns:
    'sensorType', 'timestamp', 'x', 'y', 'z'
# Combine data from all sensors into a single DataFrame
combined_data = pd.concat([accel_data, gyro_data, mag_data])

# Encode categorical labels into numerical classes
```

```python
label_encoder = LabelEncoder()
combined_data['sensorType'] =
    label_encoder.fit_transform(combined_data['sensorType'])

# Split data into features (X) and labels (y)
X = combined_data[['sensorType', 'x', 'y', 'z']].values
y = combined_data['sensorType'].values

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)

# Define the model architecture
model = Sequential([
    Dense(64, activation='relu', input_shape=(4,)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')  # 10 output classes
])

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

# Define early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss',
    patience=3, restore_best_weights=True)

# Train the model
history = model.fit(X_train, y_train, epochs=20,
    batch_size=32, validation_data=(X_test, y_test),
    callbacks=[early_stopping])
```

```

# Evaluate model on test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy * 100:.2f}%')

# Example prediction on new data
new_data = np.array([[0, 0.1, 0.2, 0.3]])  # Example data
    point, adjust as per the data format
predictions = model.predict(new_data)
predicted_class = np.argmax(predictions)
print(f'Predicted Class: {predicted_class}')
\end{verbatim}

\subsection*{Explanation}

\begin{itemize}
    \item \textbf{Import Libraries}: Import necessary
       libraries including pandas, numpy, and TensorFlow's
       Keras API for model building and training.
    \item \textbf{Define Paths to CSV Files}: Specify the
       paths to the CSV files containing accelerometer,
       gyroscope, and magnetometer data.
    \item \textbf{Load CSV Files}: Read the CSV files into
       pandas DataFrames for further processing.
    \item \textbf{Combine Data}: Combine data from all
       sensors into a single DataFrame assuming the
       structure includes columns 'sensorType',
       'timestamp', 'x', 'y', 'z'.
    \item \textbf{Label Encoding}: Encode the categorical
       sensor type labels into numerical classes using
       LabelEncoder.
    \item \textbf{Split Data}: Split the data into features
       (X) and labels (y), then further split into training
       and testing sets using train_test_split.
    \item \textbf{Model Architecture}: Define a Sequential
       model with dense layers and dropout for
       regularization.
    \item \textbf{Compile Model}: Compile the model with
       the Adam optimizer and sparse categorical
       crossentropy loss function.
    \item \textbf{Early Stopping}: Implement early stopping
       to prevent overfitting during model training.
    \item \textbf{Train Model}: Train the model on the
       training data while validating on the test data.
    \item \textbf{Evaluate Model}: Evaluate the model's
       performance on the test data and print the test
       accuracy.
    \item \textbf{Example Prediction}: Demonstrate an
       example prediction using new data, showing the
       predicted class.
\end{itemize}
```