A PROJECT REPORT

On

Autonix- Rider Safety Assistance and Monitoring System

Submitted by

Maizah Shaikh

And

Yatin Anchan

in partial fulfillment for the award of the degree

of

BACHELOR OF SCIENCE

in

COMPUTER SCIENCE

under the guidance of

Mr. Hasan Phudinawala

Department of Computer Science

Royal College of Arts, Science and Commerce

(Autonomous)

Semester V

2025-2026

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1. Introduction

Autonix is an intelligent, smartphone-based driver safety and monitoring system designed to reduce road accidents by actively detecting signs of drowsiness, reckless driving, or sudden crashes. Unlike traditional systems that rely on expensive in-car hardware or are limited to high-end vehicles, Autonix brings advanced safety features to any vehicle using just a smartphone.

The system uses the phone's front camera, GPS, and built-in motion sensors to continuously monitor the driver's face and behavior. It identifies signs of fatigue using facial landmarks and calculates the EAR (Eye Aspect Ratio) to measure eye closure levels. If prolonged drowsiness is detected, the system instantly triggers alerts through sound or vibration to wake the driver and prevent potential accidents.

In addition to drowsiness detection, Autonix tracks real-time driving patterns, detects sudden jerks or harsh braking, and simulates crash logic based on acceleration spikes. If a possible crash is detected, it automatically sends emergency alerts to predefined guardian contacts via SMS or calls, sharing location details to enable immediate assistance.

Autonix also offers a Smart Trip Planner where admins can create trips and assign them to drivers. Drivers can view their trip summaries, receive alerts, and track performance throughout the journey. With built-in navigation powered by OSMdroid, users can follow real-time routes and search locations seamlessly.

By combining safety, communication, and route management in a single mobile solution, Autonix aims to make roads safer for everyone ; not just those with access to luxury vehicle systems. It's a powerful step toward responsible driving, guardian connectivity, and smarter transport monitoring using accessible technology.

1.1 Background and Project Overview

In today's digital era, mobile applications are increasingly expected to deliver not only functionality but also adaptability and responsiveness to user needs. Conventional applications often lack the integration of smart features and real-time support, which limits user engagement and long-term impact. Addressing this gap, the Autonix system was developed as a next-generation mobile application designed to provide a seamless, interactive, and reliable digital experience.

Autonix is built on the Android runtime environment and powered by Firebase backend services, enabling secure authentication, efficient data storage, and real-time synchronization across devices. Its modular architecture ensures scalability, while its intuitive interface offers users a smooth and engaging interaction flow. By prioritizing accessibility and performance, Autonix creates a responsive mobile environment that adapts to diverse user requirements.

The system architecture also supports future enhancements, ensuring long-term sustainability and adaptability as user expectations evolve.

In summary, Autonix represents a significant advancement in mobile application design. By merging a user-friendly interface with real-time cloud-backed functionality, it empowers users with an engaging experience while providing developers with a scalable and maintainable solution. Autonix exemplifies how modern mobile applications can bridge the gap between innovation and usability, preparing digital ecosystems for the challenges of a rapidly changing world.

1.2 Objectives

Autonix's main aim is to fill in the missing parts of current driver assistance systems, especially those that are costly, too intrusive, or only work on high-end cars. With this project, we're building a smart, affordable, and easy-to-use safety tool that brings intelligent monitoring directly to your smartphone.

The main goals of Autonix are:

1. To improve drowsiness detection by looking at several facial signs, like how long the eyes are closed, how often the person blinks, and signs of facial fatigue, rather than just one thing.
2. To offer safety features that work on any vehicle, without the need for expensive hardware. By using built-in phone features like GPS, accelerometer, and gyroscope, we can detect crashes and track driving behavior on the go.
3. To create a multi-step alert system. It doesn't just warn the driver ; it can also reach out to a guardian or emergency contact if the driver is extremely tired or unresponsive.
4. To test for real-world crashes by looking at sudden changes in speed. If a crash is detected, the system gives the driver a final chance to cancel the alert through a countdown before activating emergency protocols.
5. To provide real-time performance data that shows how often the driver triggers alerts, how safely they're driving, and where they can improve.
6. To keep the system simple and affordable, so that people with regular Android phones can use features that are usually only found in expensive cars.

1.3 Purpose and Scope

1.3.1 Purpose

Autonix is developed with a clear and focused purpose to offer an affordable, non intrusive, and intelligent driver assistance system that enhances road safety, particularly for everyday vehicle users. In today's world, many advanced safety systems are either too costly, limited to premium vehicles, or intrusive in their design. Autonix aims to fill this gap.

The system is designed to detect driver drowsiness and vehicle crashes in real-time using simple camera and sensor inputs. Upon detecting any critical event, Autonix immediately raises alerts, stores essential data, and ensures the driver or caregiver is notified without delay. The idea is to prevent accidents before they happen, and respond quickly if they do.

By eliminating the need for high-end equipment and keeping the process seamless for the user, Autonix brings real-time safety within reach for a much broader audience , making roads safer and responses smarter.

1.3.2 Scope

Autonix is a mobile application focused on helping users stay safe during travel by detecting signs of drowsiness and crash-like movements. It's designed to be simple, lightweight, and accessible, especially for people who don't use high-end vehicles or expensive safety systems. Since it works directly through the user's mobile phone, there's no need for any extra hardware or car-based integration.

The app begins monitoring automatically when the user starts a trip. Throughout the journey, it quietly checks for warning signs like the user dozing off or the phone experiencing a sudden jolt that may indicate a fall or impact. If something unusual is detected, the app immediately alerts the user and logs the event details for future reference. This way, the user can take action before a situation becomes serious.

· A feature to detect drowsiness by using the phone's front camera to monitor facial cues like eye closure.

· A way to identify crash-like movements using the phone's internal motion sensors.

· A trip start mechanism that activates monitoring when the user begins a journey.

· Alerts in the form of sounds or pop-ups that notify the user in case of drowsiness or sudden motion.

· A simple local log that saves event data during the trip.

1.4 Phase Title

Phase Title	**Expected Date of Completion**	**Actual Time of Completion with Guide's Signature**	**Remarks**
I. Introduction			
(i)Background and Project Overview			
(ii) Objectives			
(iii) Purpose & Scope			
(iv) Phase Title			
(v) Gantt Chart			
II. System Analysis			
(i) Existing System			
(ii) Proposed System			
(iii) Requirement Analysis			
III. System Design			
(i) Data Flow Diagram			
(ii) Activity Diagram			
(iii) Sequence Diagram			
(iv) Class Diagram			
(v) Component Diagram			
(vi) Use Case Diagram			
(vii) ER Diagram			
IV. Implementation and Testing			
(i) System Coding			
(ii) Testing Approach			

V. Results			
VI. Conclusion And Future work			
VII. References			
(i) Plagiarism Report			
(ii) Appendix			
(iii) Approved Project Proposal			

1.5 Gantt Chart

TYBSc Computer Science Semester 5 Project Gantt Chart	Time Requirements		Year 2025-2026											
		June	July				August				September			
		W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
Requirement Gathering & Feasibility Study	Estimated													
	Actual													
Project Planning	Estimated													
	Actual													
System Design	Estimated													
	Actual													
Implementation / Coding	Estimated													
	Actual													
Testing	Estimated													
	Actual													
Deployment Preparation	Estimated													
	Actual													

CHAPTER 2
SYSTEM ANALYSIS

2.1 Existing System

The landscape of driver safety assistance has seen significant advancements, with various systems developed to address drowsiness detection and accident monitoring. Existing solutions rely on different methodologies, including sensor-based, vision- based, and vehicle-integrated approaches, each contributing uniquely to road safety but also presenting challenges.

Many existing drowsiness detection systems depend on physiological sensors (EEG, ECG) or vehicle-based signals such as steering patterns and lane deviation. While effective in controlled conditions, these approaches are intrusive, uncomfortable for long-term use, and expensive due to specialized hardware requirements.

Camera-driven approaches analyze facial cues such as eye closure, yawning, or head tilt to detect fatigue. Systems like the Driver Drowsiness Shield (DDSH) have demonstrated high accuracy using eye aspect ratios and facial landmarks. However, performance degrades under poor lighting, when the driver wears glasses/masks, or in cases of rapid head movements, reducing their reliability in real-world environments.

Lightweight Android applications using Haar cascades, blink detection, or Google's Mobile Vision API have been proposed as low-cost alternatives. They eliminate the need for external sensors, relying only on a smartphone camera. Although these applications are portable and user-friendly, they are often limited in scope, focusing only on eye closure or blink rate. This single-metric reliance increases false alarms and decreases robustness.

2.2 Proposed System

1. System Architecture

Autonix is built as a modular, layered system that combines real-time responsiveness with scalability. Operating within the Android runtime environment using Java/Kotlin, the app provides a simple, driver-focused interface for starting trips, viewing logs, and receiving notifications. Critical processing occurs locally, with the front camera monitoring signs of drowsiness such as eye closure, blinking, and yawning, while the accelerometer and gyroscope detect shocks or crash-like movements. Lightweight algorithms ensure these computations run instantly on-device without requiring internet connectivity, allowing immediate alerts in safety-critical situations.

The system is supported by a Firebase cloud backend for secure authentication, data storage, and log synchronization. Events recorded locally are synced to the cloud for persistent storage and review, enabling scalability and future enhancements like analytics or machine learning. Alerts are delivered via visual, audio, and haptic cues, ensuring rapid driver response. Privacy is maintained by processing sensitive sensor data locally, while only essential event information is stored securely in the cloud. This combination of on-device monitoring and cloud support ensures Autonix is efficient, reliable, and adaptable for current and future safety features.

2. Modules

1. *Splash Activity*

This serves as the application entry point, displaying the Autonix branding and initializing core system components. The splash screen provides a professional introduction while loading essential services in the background, including Firebase authentication, sensor initialization, and permission checks for camera and location access.

2. *Welcome Activity*

Once the application loads, users are directed to the WelcomeActivity which serves as the main navigation hub. This module provides options to:

- Login/Register: Navigate to user authentication flows
- Guest Mode: Access basic safety features without registration
- Emergency Contacts Setup: Quick access to configure guardian contacts
- About: Information about Autonix features and usage guidelines

3. *Login Activity and Register Activity*

These modules handle comprehensive user authentication and profile creation:

- Registration: Users provide essential details including name, phone number, emergency contacts, and create secure credentials. All data is stored securely in Firebase with encrypted passwords and verified emergency contact information.
- Login: Existing users authenticate using email/phone and password, with session management through Firebase Authentication.

- Password Recovery: Integrated forgot password functionality with email/SMS verification for account recovery.

4. *Main Activity*

This is the core activity that coordinates all safety monitoring features once a user is authenticated. It includes navigation to:

- Trip Monitoring: Start/stop trip tracking and safety monitoring
- Profile Management: Update user information and emergency contacts
- Trip History: View past trip summaries and safety events
- Fleet Management: For admin users to manage multiple drivers
- Navigation: Access to integrated route planning and GPS navigation

5. *Trip Monitoring*

The central safety monitoring module that activates during active trips:

- DrowsinessDetectionService: Continuous monitoring using front camera and MediaPipe FaceMesh
- CrashAlertActivity: Real-time accelerometer and gyroscope data analysis for crash detection
- Emergency Alert System: Automated emergency contact notification with location sharing
- Performance Logging: Records safety events, alert frequency, and driver behavior patterns

6. *Drowsiness Detector and Drowsiness Detection Service*

These modules implement the core drowsiness detection functionality:

- Facial Landmark Detection: Uses MediaPipe to identify eye landmarks and calculate Eye Aspect Ratio (EAR)
- Real-time Analysis: Continuous monitoring of eye closure patterns, blink frequency, and facial orientation
- Alert Escalation: Multi-level warning system from subtle notifications to emergency contact alerts
- Calibration Support: Personalized threshold adjustment based on individual baseline measurements

7. *Crash Alert Activity*

Implements comprehensive crash detection and emergency response:

- Sensor Fusion: Combines accelerometer, gyroscope, and GPS data for accurate crash detection
- Impact Analysis: Advanced algorithms to distinguish between actual crashes and false triggers (potholes, sudden braking)
- Emergency Protocol: 30-second countdown system allowing user cancellation before automatic emergency contact notification
- Location Services: Real-time GPS coordinates shared with emergency contacts and services

8. *Emergency Contacts Activity*

Manages emergency contact configuration and communication:

- Contact Management: Add, edit, and prioritize emergency contacts with verification
- Communication Protocols: SMS and call integration for multi-channel emergency alerts
- Guardian Dashboard: Optional web interface for emergency contacts to monitor trip status
- Escalation Procedures: Hierarchical contact system with automatic escalation if primary contacts are unresponsive

9. *Navigation Dashboard Activity and Navigation Planner Activity*

Integrated navigation and route planning system:

- Route Planning: Smart trip planner with rest stop suggestions and route optimization
- Real-time Navigation: Turn-by-turn directions with traffic updates
- Safety Integration: Route modifications based on driver alertness and safety events
- Fleet Coordination: For fleet managers to plan and assign routes to multiple drivers

10. *Activity Profile and Activity Profile Edit*

Comprehensive user profile management:

- Personal Information: Driver details, contact information, and medical alerts if applicable
- Driving Preferences: Customizable alert thresholds, notification preferences, and safety settings
- Performance Analytics: Historical data showing improvement trends and safety scores
- Fleet Integration: Professional driver profiles with employer information and compliance tracking

11. *Fleet Control Activity , Fleet Driver and Fleet Driver Adapter*

Advanced fleet management capabilities for commercial use:

- Driver Assignment: Assign trips and routes to specific drivers with real-time tracking
- Performance Monitoring: Centralized dashboard showing all drivers' safety metrics and current status
- Compliance Tracking: Monitor driving hours, mandatory rest periods, and safety compliance
- Alert Management: Fleet-wide emergency alert system with management notifications

12. *Trip History Adapter , Trip History List Activity and Trip Summary Activity*

Comprehensive trip tracking and analysis:

- Trip Logging: Detailed records of each journey including duration, distance, safety events, and performance scores

- Visual Analytics: Graphs and charts showing performance trends, drowsiness patterns, and improvement areas
- Comparative Analysis: Compare performance across different time periods, routes, and conditions
- Report Generation: Exportable reports for personal review or fleet management purposes

13. *Search Results Adapter and Smart Map View*

Enhanced location services and mapping:

- Location Search: Intelligent search for destinations with real-time traffic and safety considerations
- Interactive Mapping: Custom map interface optimized for safety monitoring and route visualization
- Points of Interest: Rest stops, hospitals, and emergency services highlighted along routes
- Offline Capability: Cached maps and essential data for areas with poor connectivity

14. *Safety Events Adapter and Shared Preferences Helper*

Data management and event tracking:

- Event Logging: Detailed recording of all safety events including drowsiness alerts, crashes, and near-misses
- Data Persistence: Local storage management with cloud synchronization for data backup

- Privacy Controls: User-controlled data sharing settings with secure local storage options
- Analytics Processing: Background data analysis for pattern recognition and personalized recommendations

3. Data Handling

Autonix leverages Firebase for comprehensive data management, providing real- time synchronization, secure authentication, and reliable cloud storage. The system implements multiple layers of data security including end-to-end encryption for sensitive information, secure biometric authentication options, and GDPR- compliant data handling practices. Local data caching ensures functionality during poor connectivity while maintaining synchronization when connection is restored.

User profiles, trip histories, and safety events are stored with appropriate privacy controls, allowing users to maintain ownership of their data while enabling essential safety features. Emergency contact information is encrypted and verified to ensure reliability during critical situations.

4. Deployment and Technology Stack

Autonix is developed using Android Studio with Java/Kotlin, providing native Android performance and full access to device sensors and capabilities. The application integrates Firebase for backend services, Android's built-in Camera2 API for facial detection and analysis, and CameraX library for efficient camera operations. The frontend utilizes Material Design principles for intuitive user interaction, while background services ensure continuous safety monitoring without impacting device performance.

The technology stack includes real-time database synchronization, cloud-based user authentication, and robust offline capabilities. Integration with device sensors (accelerometer, gyroscope, GPS, camera) provides comprehensive safety monitoring while maintaining optimal battery efficiency through intelligent resource management.

5. Evaluation

The system is tested in different driving conditions and lighting situations to ensure it works reliably. Safety features are checked through controlled tests and monitoring of accuracy rates. User feedback helps improve the detection algorithms and overall user experience.

Performance testing ensures the app works smoothly on various Android devices while maintaining consistent safety monitoring. Regular updates include user feedback, algorithm improvements, and new safety features to keep the system current and effective.

2.2 Requirement Analysis

2.2.1 Hardware Requirements & its specifications:

Component	**Recommended Specifications**
Android Smartphone	Android 10 or above, Minimum 3 GB RAM
Mobile Camera	Minimum 720p resolution
Internet Connectivity	Stable broadband (>= 5 Mbps)

2.2.2 Software Requirements & its specifications:

Software Components:	**Required Version:**
Android Studio	Latest stable version (>= Flamingo)
Firebase	Blaze plan (for scalable usage)
Programming Languages	Java 17 / Kotlin 1.9 or above, Python 3.10 or above

CHAPTER 3
SYSTEM DESIGN

3.1 Use Case Diagram

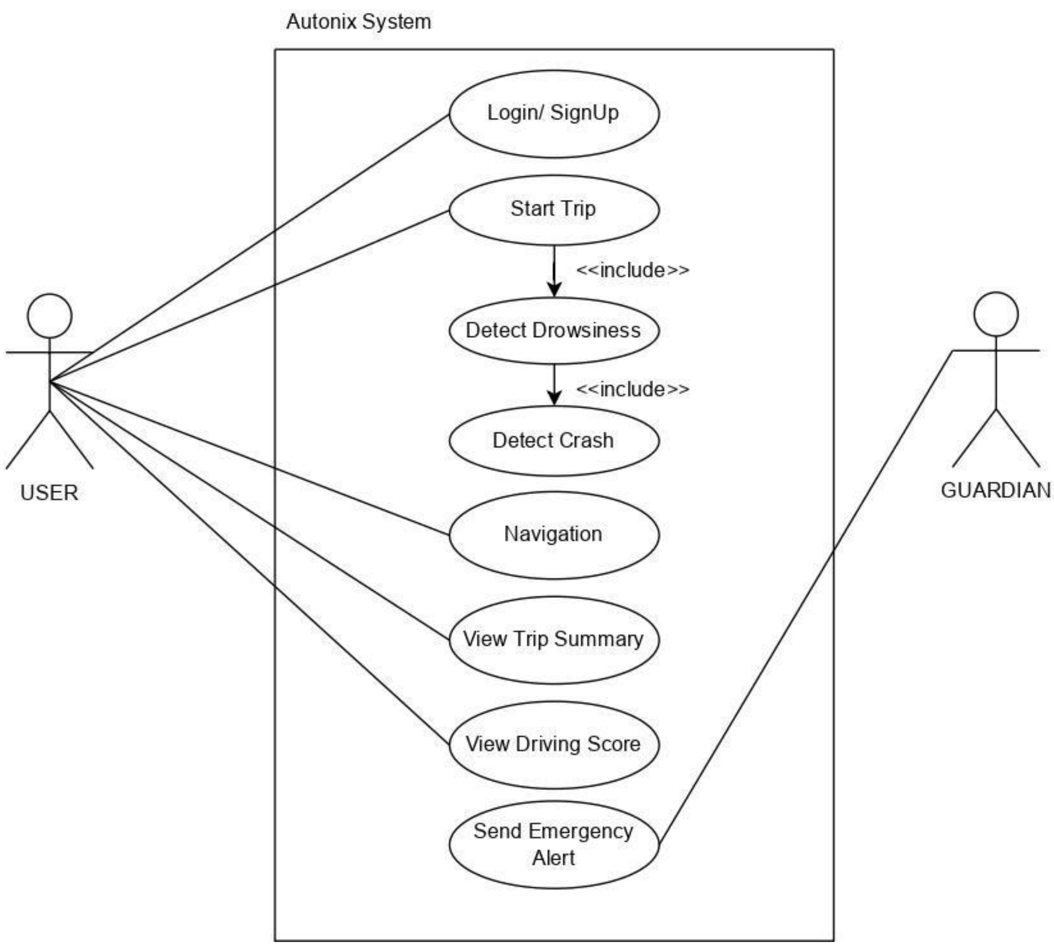


Fig 3.1 Use Case Diagram

3.2 Activity Diagram

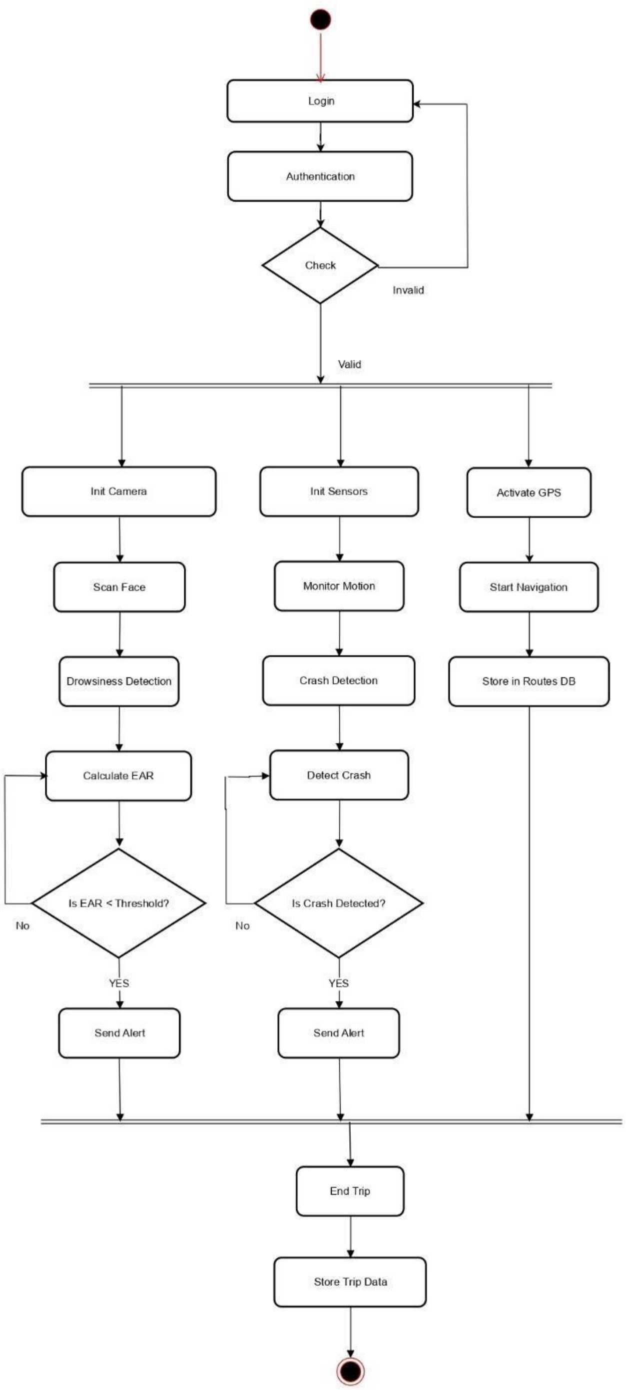


Fig 3.2 Activity Diagram

3.3 Sequence Diagram

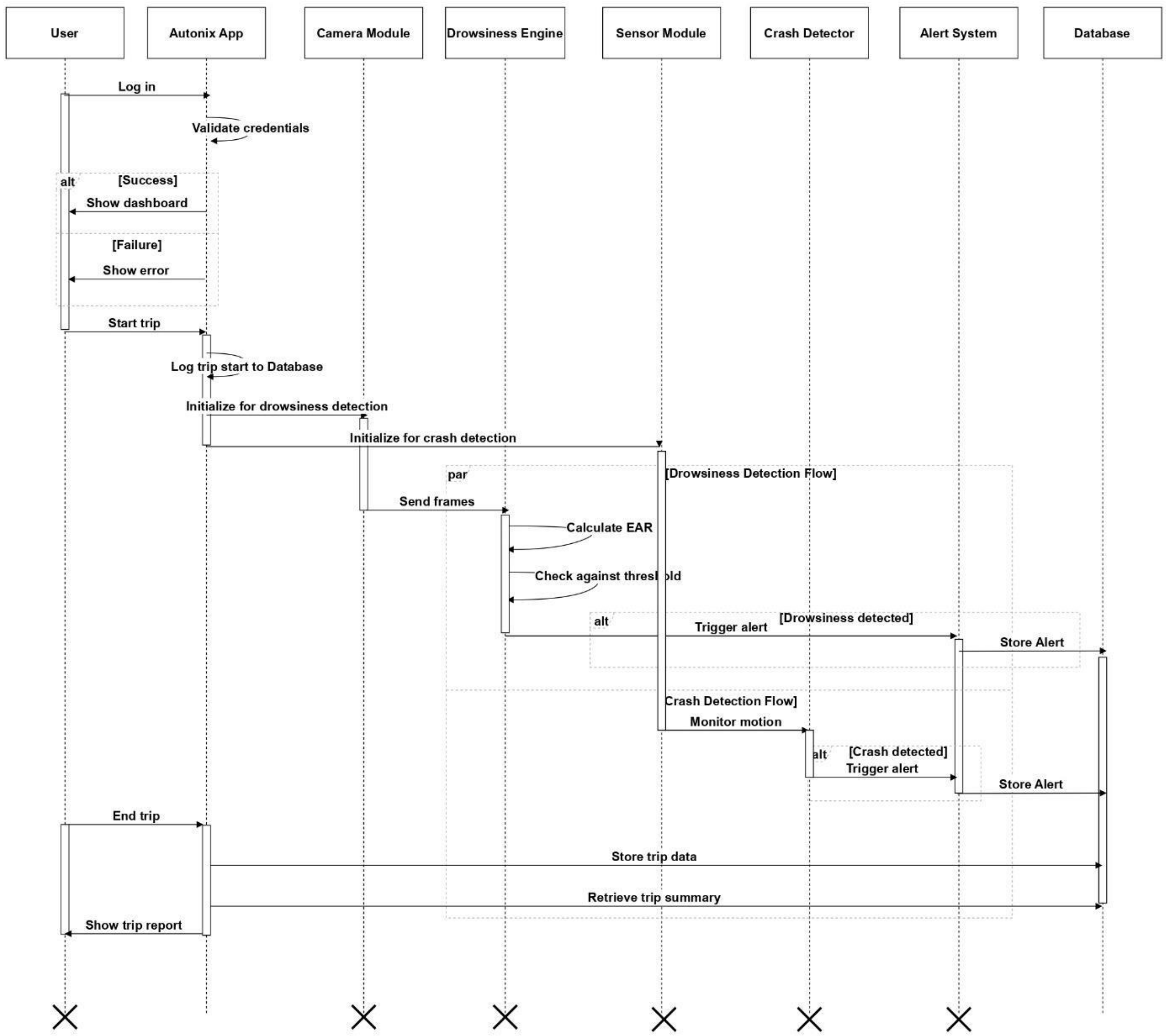


Fig 3.3 Sequence Diagram

3.4 Class Diagram

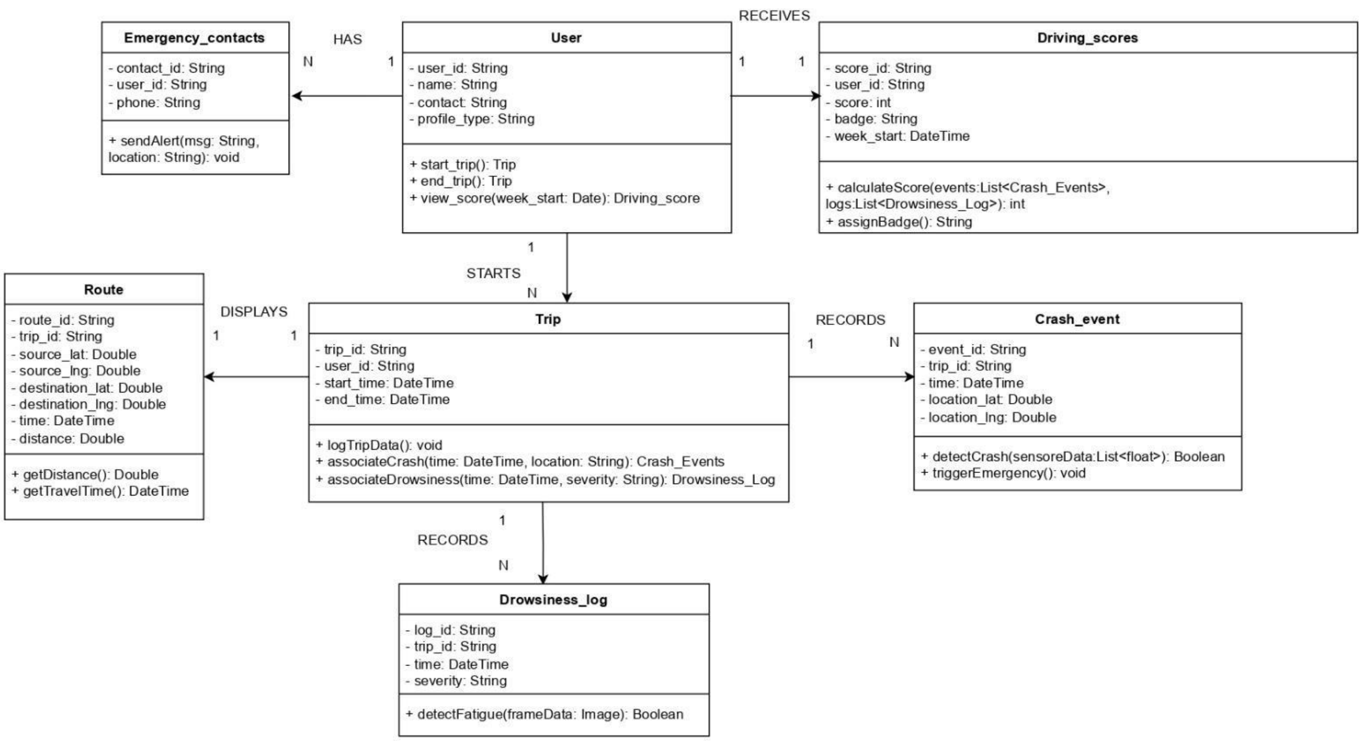


Fig 3.4 Class Diagram for Overall System

3.5 ER Diagram

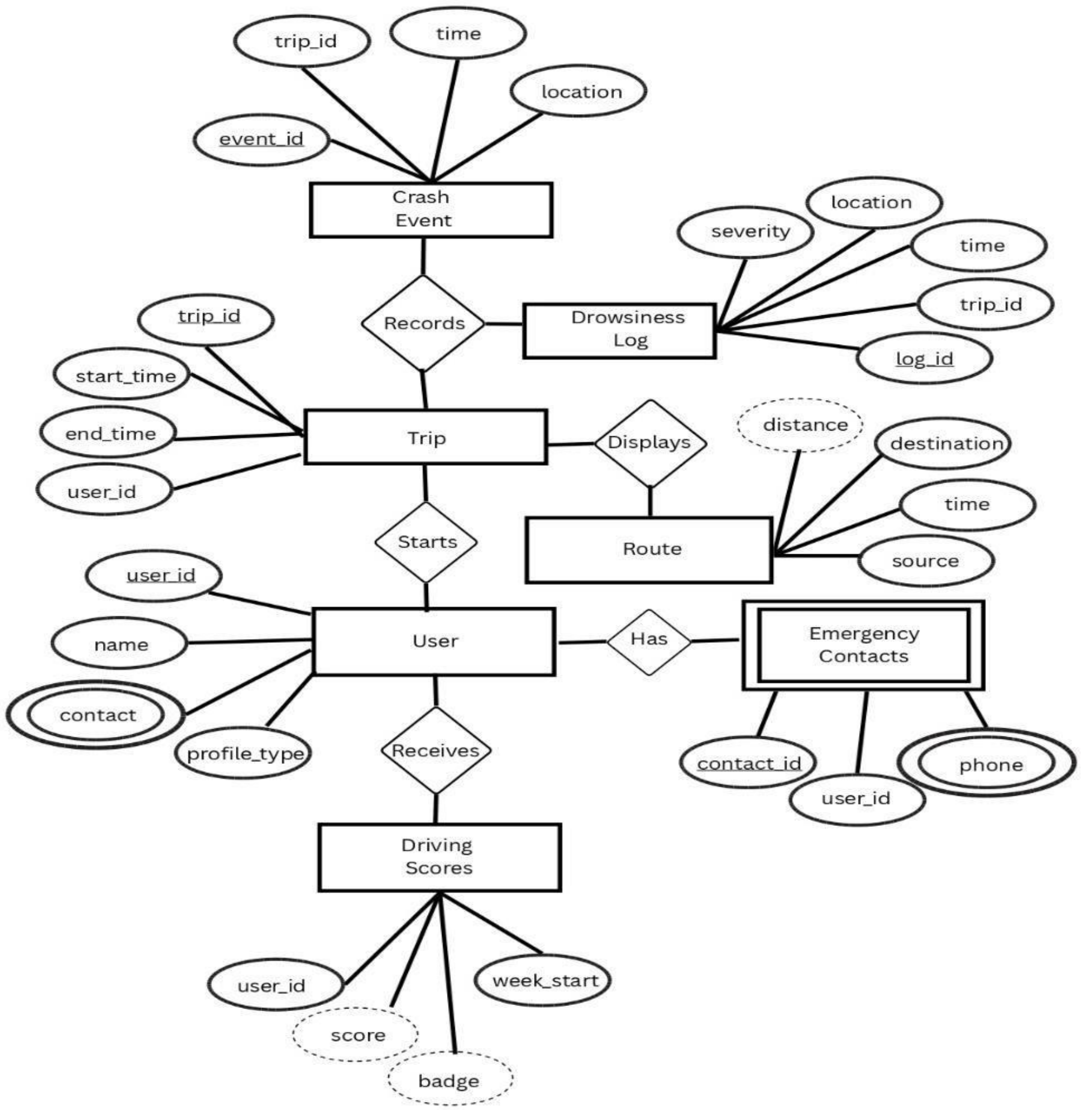


Fig 3.5 ER Diagram

3.6 Data Flow Diagram

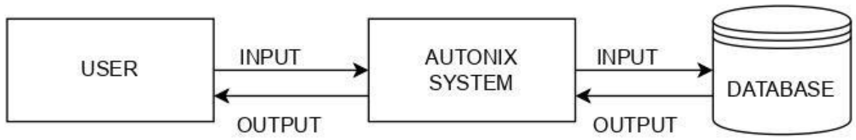


Fig 3.6.1 Data Flow Level 0

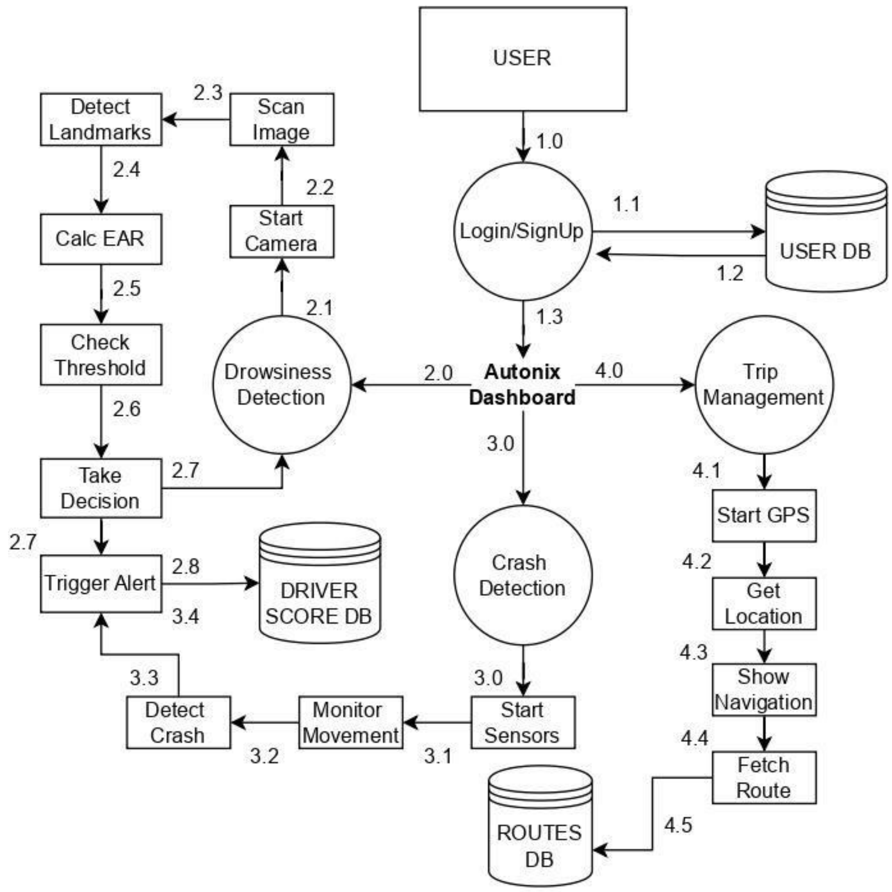


Fig 3.6.2 Data Flow Level 1

3.7 Component Diagram

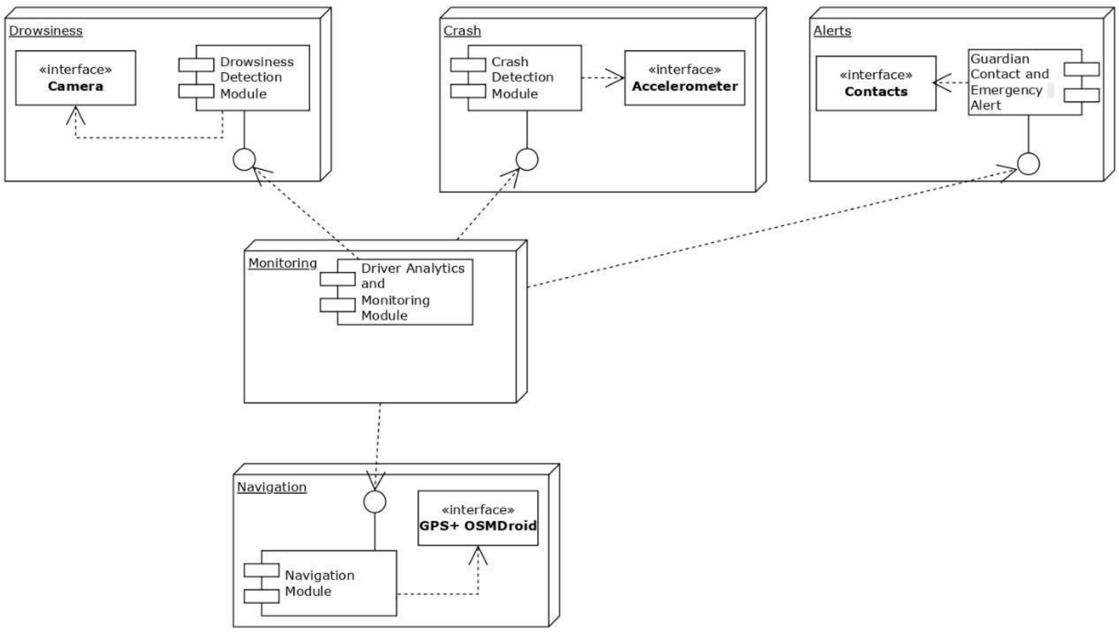


Fig 3.7 Component Diagram

CHAPTER 4

IMPLEMENTATION AND TESTING

4.1 Code

- *WelcomeActivity.kt*

```
package com.example.autonix_work_in_progress
import android.content.Intent
import android.os.Bundle
import android.widget.Button
import androidx.appcompat.app.AppCompatActivity
class WelcomeActivity : AppCompatActivity() {
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_welcome)
setupButtons()}
private fun setupButtons() {
val btnCreateAccount = findViewById<Button>(R.id.btn_create_account)
val btnSignIn = findViewById<Button>(R.id.btn_sign_in)
btnCreateAccount.setOnClickListener {
startActivity(Intent(this, RegisterActivity::class.java))}
btnSignIn.setOnClickListener {
startActivity(Intent(this, LoginActivity::class.java))}}}
```

- *DrowsinessDetector.kt*

```
package com.example.autonix_work_in_progress
import android.graphics.PointF
import android.os.SystemClock
import android.util.Log
import kotlin.math.abs
import kotlin.math.hypot
class DrowsinessDetector(
private val listener: DrowsinessListener){
companion object {
private const val TAG = "DrowsinessDetector"}
interface DrowsinessListener {
fun onEyesClosedDetected(closedMs: Long, description: String)
fun onYawnDetected(ratio: Float, description: String)
fun onHeadNodDetected(angleX: Float, angleZ: Float, description: String)
fun onNormal() // called when face present and no condition}
```

```
// Tunable thresholds
private val EAR_THRESHOLD = 0.20f // Eye Aspect Ratio below -> eye considered closed
private val CLOSED_TIME_THRESHOLD_MS = 1500L // how long eyes must remain closed
to trigger event
private val YAWN_RATIO_THRESHOLD = 0.55f // vertical/horizontal mouth ratio for a yawn
private val HEAD_NOD_ANGLE_THRESHOLD_X = 18f // head nod (pitch) threshold degrees
(x-axis)
private val HEAD_TILT_ANGLE_THRESHOLD_Z = 20f // head tilt threshold degrees (z-axis)
// State
private var eyesClosedSince: Long = -1L
private var isAlerting = false
fun processFace(face: com.google.mlkit.vision.face.Face) {
val now = SystemClock.elapsedRealtime()
// EAR (eye aspect ratio)
Val leftEyeContour =
face.getContour(com.google.mlkit.vision.face.FaceContour.LEFT_EYE)?.points
val rightEyeContour =
face.getContour(com.google.mlkit.vision.face.FaceContour.RIGHT_EYE)?.points
val leftEAR = if (!leftEyeContour.isNullOrEmpty()) computeEARFromContour(leftEyeContour)
else -1f
val rightEAR = if (!rightEyeContour.isNullOrEmpty())
computeEARFromContour(rightEyeContour) else -1f
val avgEAR = when {
leftEAR > 0 && rightEAR > 0 -> (leftEAR + rightEAR) / 2f
leftEAR > 0 -> leftEAR
rightEAR > 0 -> rightEAR
else -> -1f}
// Mouth ratio (yawn)
val upperLipTop =
face.getContour(com.google.mlkit.vision.face.FaceContour.UPPER_LIP_TOP)?.points
val lowerLipBottom =
face.getContour(com.google.mlkit.vision.face.FaceContour.LOWER_LIP_BOTTOM)?.points
val mouthRatio = if (!upperLipTop.isNullOrEmpty() && !lowerLipBottom.isNullOrEmpty()) {
computeMouthRatio(upperLipTop, lowerLipBottom)
} else 0f
// Head pose (use ML Kit provided angles)
val headX = face.headEulerAngleX // pitch (nodding)
val headY = face.headEulerAngleY // yaw (left/right)
val headZ = face.headEulerAngleZ // roll (tilt)
```

```
// Evaluate eye closure using EAR
if (avgEAR > 0f && avgEAR < EAR_THRESHOLD) {
if (eyesClosedSince < 0L) eyesClosedSince = now
val closedFor = now - eyesClosedSince
if (closedFor >= CLOSED_TIME_THRESHOLD_MS && !isAlerting) {
isAlerting = true
listener.onEyesClosedDetected(
closedFor,
"Eyes closed for ${closedFor}ms (EAR=${"%.2f".format(avgEAR)})")}}
else {
// Eyes open
if (isAlerting) {
// Clear alert state once reopened
isAlerting = false}
eyesClosedSince = -1L}
// Evaluate yawn
if (mouthRatio > YAWN_RATIO_THRESHOLD) {
listener.onYawnDetected(
mouthRatio,
"Yawn detected (ratio=${"%.2f".format(mouthRatio)})")}
// Head nod/tilt detection
if (abs(headX) > HEAD_NOD_ANGLE_THRESHOLD_X || abs(headZ) >
HEAD_TILT_ANGLE_THRESHOLD_Z) {
listener.onHeadNodDetected(
headX, headZ,
"Head nod/tilt detected (X=${"%.1f".format(headX)}°, Z=${"%.1f".format(headZ)}°)")}
// If none triggered and face present, inform normal
if (!isAlerting &&
!(mouthRatio > YAWN_RATIO_THRESHOLD) &&
abs(headX) <= HEAD_NOD_ANGLE_THRESHOLD_X &&
abs(headZ) <= HEAD_TILT_ANGLE_THRESHOLD_Z) {
listener.onNormal()}
Log.d(TAG, "EAR L=${"%.2f".format(leftEAR)} R=${"%.2f".format(rightEAR)} " +
"avg=${"%.2f".format(if (avgEAR>0) avgEAR else 0f)} " +
"mouthRatio=${"%.2f".format(mouthRatio)} " +
"headX=${"%.1f".format(headX)} headZ=${"%.1f".format(headZ)}")}
private fun computeEARFromContour(eye: List<PointF>): Float {
// ML Kit eye contour contains many points around the eye; strategy:
// - find left-most and right-most points for width (horizontal)
// - estimate top average and bottom average vertical positions for height (vertical)
```

```
if (eye.size < 4) return -1f
// Leftmost and rightmost by x
val left = eye.minByOrNull { it.x }!!
val right = eye.maxByOrNull { it.x }!!
// Top average (points with y < median y)
val sortedByY = eye.sortedBy { it.y }
val topAvg = averagePoint(sortedByY.take(eye.size / 3))
val bottomAvg = averagePoint(sortedByY.takeLast(eye.size / 3))
val vert = distance(topAvg.x, topAvg.y, bottomAvg.x, bottomAvg.y)
val hor = distance(left.x, left.y, right.x, right.y)
if (hor <= 0f) return -1f
return (vert / hor)}
private fun computeMouthRatio(upper: List<PointF>, lower: List<PointF>): Float {
val top = averagePoint(upper)
val bottom = averagePoint(lower)
// Width estimate: find leftmost & rightmost from combined lower points
// (often lower lip bottom contains full width)
val combined = upper + lower
val left = combined.minByOrNull { it.x } ?: top
val right = combined.maxByOrNull { it.x } ?: bottom
val vert = distance(top.x, top.y, bottom.x, bottom.y)
val hor = distance(left.x, left.y, right.x, right.y)
if (hor <= 0f) return 0f
return (vert / hor)}
private fun averagePoint(points: List<PointF>): PointF {
if (points.isEmpty()) return PointF(0f, 0f)
var sx = 0f
var sy = 0f
for (p in points) {
sx += p.x
sy += p.y}
return PointF(sx / points.size, sy / points.size)}
private fun distance(x1: Float, y1: Float, x2: Float, y2: Float): Float {
return hypot((x2 - x1).toDouble(), (y2 - y1).toDouble()).toFloat()}}
```

- CrashAlertActivity.kt

```
package com.example.autonix_work_in_progress
import android.Manifest
```

```
import android.content.pm.PackageManager
import android.hardware.camera2.CameraManager
import android.location.Location
import android.location.LocationListener
import android.location.LocationManager
import android.media.AudioManager
import android.media.ToneGenerator
import android.os.*
import android.telephony.SmsManager
import android.view.WindowManager
import android.widget.Button
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import com.google.android.gms.location.*
import com.google.android.gms.tasks.CancellationTokenSource
import kotlin.random.Random
import kotlinx.coroutines.*
import java.text.SimpleDateFormat
import java.util.Calendar
import java.util.Date
import java.util.Locale
class CrashAlertActivity : AppCompatActivity() {
private lateinit var auth: FirebaseAuth
private lateinit var db: FirebaseFirestore
private lateinit var tvName: TextView
private lateinit var tvAge: TextView
private lateinit var tvDob: TextView
private lateinit var tvBloodGroup: TextView
private lateinit var tvCrashTime: TextView
private lateinit var tvContacts: TextView
private lateinit var btnStop: Button
private var sosJob: Job? = null
private var gradientJob: Job? = null
private var torchOn = false
```

```
private var cameraManager: CameraManager? = null
private var cameraId: String? = null
private val LOCATION_PERMISSION_CODE = 2001
private val SMS_PERMISSION_CODE = 2002
private lateinit var locationManager: LocationManager
private lateinit var fusedLocationClient: FusedLocationProviderClient
private var currentLocation: Location? = null
private var pendingSendEmergency = false
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_crash_alert)
// Firebase
auth = FirebaseAuth.getInstance()
db = FirebaseFirestore.getInstance()
window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)
// Views
tvName = findViewById(R.id.tvName)
tvAge = findViewById(R.id.tvAge)
tvDob = findViewById(R.id.tvDob)
tvBloodGroup = findViewById(R.id.tvBloodGroupIcon)
tvCrashTime = findViewById(R.id.tvCrashTime)
tvContacts = findViewById(R.id.tvContacts)
btnStop = findViewById(R.id.btnStop)
// Torch
cameraManager = getSystemService(CAMERA_SERVICE) as CameraManager
cameraId = cameraManager?.cameraIdList?.getOrNull(0)
// Location
locationManager = getSystemService(LOCATION_SERVICE) as LocationManager
fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
checkLocationPermission()
// Populate user info
loadUserInfo()
// Start SOS + gradient
startEmergencyLoop()
btnStop.setOnClickListener {
finish()}}
private fun checkLocationPermission() {
val permissions = arrayOf(
Manifest.permission.ACCESS_FINE_LOCATION,
Manifest.permission.SEND_SMS)
```

```
if (permissions.any { ContextCompat.checkSelfPermission(this, it) !=
PackageManager.PERMISSION_GRANTED }) {
ActivityCompat.requestPermissions(this, permissions, LOCATION_PERMISSION_CODE)
} else {
startLocationUpdates()}}
private fun startLocationUpdates() {
try {
locationManager.requestLocationUpdates(
LocationManager.GPS_PROVIDER,
2000, 1f, locationListener)
locationManager.requestLocationUpdates(
LocationManager.NETWORK_PROVIDER,
2000, 1f, locationListener)
} catch (_: SecurityException) {}}
private val locationListener = object : LocationListener {
override fun onLocationChanged(location: Location) {
currentLocation = location}
override fun onProviderEnabled(provider: String) {}
override fun onProviderDisabled(provider: String) {}
override fun onStatusChanged(provider: String?, status: Int, extras: Bundle?) {}}
private fun loadUserInfo() {
val user = auth.currentUser ?: return
// Read user profile
db.collection("users").document(user.uid).get()
.addOnSuccessListener { doc ->
val fullName = doc.getString("fullName") ?: "Unknown"
val phone = doc.getString("phone") ?: "Unknown"
val blood_group = doc.getString("bloodGroup")
val createdAt = doc.getLong("createdAt") ?: 0L
tvName.text = "Name: $fullName"
tvDob.text = "Phone: $phone" // you can map DOB later if you store it
tvBloodGroup.text = "$blood_group"
tvAge.text = "Joined: ${SimpleDateFormat("dd/MM/yyyy",
Locale.getDefault()).format(Date(createdAt))}"
val time = SimpleDateFormat("HH:mm:ss dd/MM/yyyy", Locale.getDefault()).format(Date())
tvCrashTime.text = "Crash Occurred: $time"}
// Read emergency contacts from subcollection
db.collection("emergency_contacts").document(user.uid)
.collection("contacts")
.get()
```

```
.addOnSuccessListener { snapshot ->
val phones = snapshot.mapNotNull { it.getString("phone") }
if (phones.isNotEmpty()) {
tvContacts.text = "Emergency Contacts: ${phones.joinToString()}"
} else {
tvContacts.text = "Emergency Contacts: None"}}}
private fun calculateAge(dob: String): Int {
return try {
val parts = dob.split("-")
val calDob = Calendar.getInstance().apply {
set(parts[0].toInt(), parts[1].toInt() - 1, parts[2].toInt())}
val now = Calendar.getInstance()
var age = now.get(Calendar.YEAR) - calDob.get(Calendar.YEAR)
if (now.get(Calendar.DAY_OF_YEAR) < calDob.get(Calendar.DAY_OF_YEAR)) age--
age
} catch (e: Exception) {0}}
private fun startEmergencyLoop() {
gradientJob = CoroutineScope(Dispatchers.Main).launch {
val colors = arrayOf(
intArrayOf(0xFFFF0000.toInt(), 0xFF800000.toInt()),
intArrayOf(0xFF800000.toInt(), 0xFFFF0000.toInt()))
var i = 0
while (isActive) {
val gd = android.graphics.drawable.GradientDrawable(
android.graphics.drawable.GradientDrawable.Orientation.TOP_BOTTOM,
colors[i % 2])
window.decorView.background = gd
i++
delay(500)}}
sosJob = CoroutineScope(Dispatchers.IO).launch {
val toneGen = ToneGenerator(AudioManager.STREAM_ALARM, 100)
val morse = "... --- ..."
while (isActive) {
for (ch in morse) {
if (!isActive) break
when (ch) {
'.' -> {
toneGen.startTone(ToneGenerator.TONE_CDMA_ALERT_CALL_GUARD, 200)
toggleFlash(200)}
'-' -> {
```

```
toneGen.startTone(ToneGenerator.TONE_CDMA_ALERT_CALL_GUARD, 600)
toggleFlash(600)}
else -> delay(200)}
delay(200)}
delay(800)

getLastLocation { loc ->
loc?.let { sendEmergencySMS(it.latitude, it.longitude) }}}}}
private suspend fun toggleFlash(duration: Long) {
cameraId?.let {
try {
cameraManager?.setTorchMode(it, true)
delay(duration)
cameraManager?.setTorchMode(it, false)
} catch (_: Exception) {}}}
private fun getLastLocation(callback: (Location?) -> Unit) {
if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION)
!= PackageManager.PERMISSION_GRANTED) {
callback(null)
return}
try {
fusedLocationClient.lastLocation
.addOnSuccessListener { loc ->
if (loc != null) callback(loc) else {
val cts = CancellationTokenSource()
fusedLocationClient.getCurrentLocation(
LocationRequest.PRIORITY_BALANCED_POWER_ACCURACY,
cts.token)
.addOnSuccessListener { callback(it) }
.addOnFailureListener { callback(null) }}}
.addOnFailureListener { callback(null) }
} catch (e: SecurityException) {
callback(null)}}
private fun sendEmergencySMS(lat: Double, lon: Double) {
val user = auth.currentUser ?: return
val contactsRef = db.collection("emergency_contacts")
.document(user.uid)
.collection("contacts")
contactsRef.get().addOnSuccessListener { snapshot ->
```

```
val contacts = snapshot.mapNotNull { it.getString("phone") }.filter { it.isNotEmpty() }
if (contacts.isEmpty()) return@addOnSuccessListener
val smsBody = "EMERGENCY! Crash detected. My location:
https://www.google.com/maps?q=$lat,$lon&z=18"
if (ContextCompat.checkSelfPermission(this, Manifest.permission.SEND_SMS)
== PackageManager.PERMISSION_GRANTED) {
val smsManager = SmsManager.getDefault()
contacts.forEach { phone ->
try {
val parts = smsManager.divideMessage(smsBody)
if (parts.size > 1) {
smsManager.sendMultipartTextMessage(phone, null, parts, null, null)}
else {
smsManager.sendTextMessage(phone, null, smsBody, null, null)}}
catch (_: Exception) {}}}
else {
pendingSendEmergency = true
ActivityCompat.requestPermissions(
this,
arrayOf(Manifest.permission.SEND_SMS),
SMS_PERMISSION_CODE}}}}
private fun stopEmergencyLoop() {
sosJob?.cancel()
gradientJob?.cancel()
cameraId?.let { cameraManager?.setTorchMode(it, false) }
}
override fun onRequestPermissionsResult(
requestCode: Int,
permissions: Array<out String>,
grantResults: IntArray
) {
super.onRequestPermissionsResult(requestCode, permissions, grantResults)
if (requestCode == LOCATION_PERMISSION_CODE) {
val allGranted = grantResults.isNotEmpty() &&
grantResults.all { it == PackageManager.PERMISSION_GRANTED }
if (allGranted) startLocationUpdates()
} else if (requestCode == SMS_PERMISSION_CODE &&
grantResults.isNotEmpty() &&
grantResults[0] == PackageManager.PERMISSION_GRANTED) {
if (pendingSendEmergency) {
```

```
pendingSendEmergency = false
currentLocation?.let { sendEmergencySMS(it.latitude, it.longitude) }}}}
override fun onDestroy() {
super.onDestroy()
stopEmergencyLoop()
locationManager.removeUpdates(locationListener) }}
```

4.2 Testing Approach

4.2.1 Unit Testing

Unit testing is a type of software testing where individual components or units of a software application are tested independently to ensure that each part functions correctly. A unit is the smallest testable part of any software system, often a function or method within a module. The primary goal of unit testing is to isolate each part of the program and show that the individual parts are correct in terms of requirements and functionality.

TEST CASEID	TESTCASE DESCRIPTION	OBJECTIVE	EXPECTED RESULTS	ACTUAL RESULTS	REMARK
TC01	Register a new user with valid details	To verify that the system allows new user registration	User is registered successfully; confirmation message shown	As Expected	**Pass**
TC02	User login with correct credentials	To verify login functionality with correct credentials.	Successful login and redirection to user dashboard	As Expected	**Pass**
TC03	User login with incorrect password	To validate error handling for wrong password	Error message displayed; no login	As Expected	**Pass**

TC04	Verify trip start feature	To test that when the user starts a trip, the monitoring begins.	Camera and sensors activate, monitoring begins	As Expected	**Pass**
TC05	Verify drowsiness detection	To check if system accurately detects drowsiness by eye closure and issues alerts.	Alert triggered	As Expected	**Pass**
TC06	Verify normal eye state does not trigger alert	To validate that normal driving behavior (eyes open) does not generate false alerts.	No alert displayed	As Expected	**Pass**
TC07	Verify crash detection	To ensure that sudden and strong motion values detected by accelerometer/gyroscope are correctly identified as crash events, triggering alerts.	Crash alert triggered immediately	As Expected	**Pass**
TC08	Verify alert cancellation	To check that users can dismiss false alerts manually by tapping "Cancel," stopping alert immediately.	Alert stops immediately	As Expected	**Pass**
TC09	Verify emergency message delivery	To confirm that when a crash is detected and not canceled within a time limit, an emergency SMS is sent automatically to the registered guardian with the trip details and location.	An emergency SMS is automatically sent to the guardian.	As Expected	**Pass**

TC10	Verify false motion does not trigger crash alert	To test that small bumps, normal speed breakers, or minor motions do not produce unnecessary crash alerts or emergency messages.	No crash alert triggered	As Expected	**Pass**

CHAPTER 5

RESULTS

Fig 5.1 Splash Screen

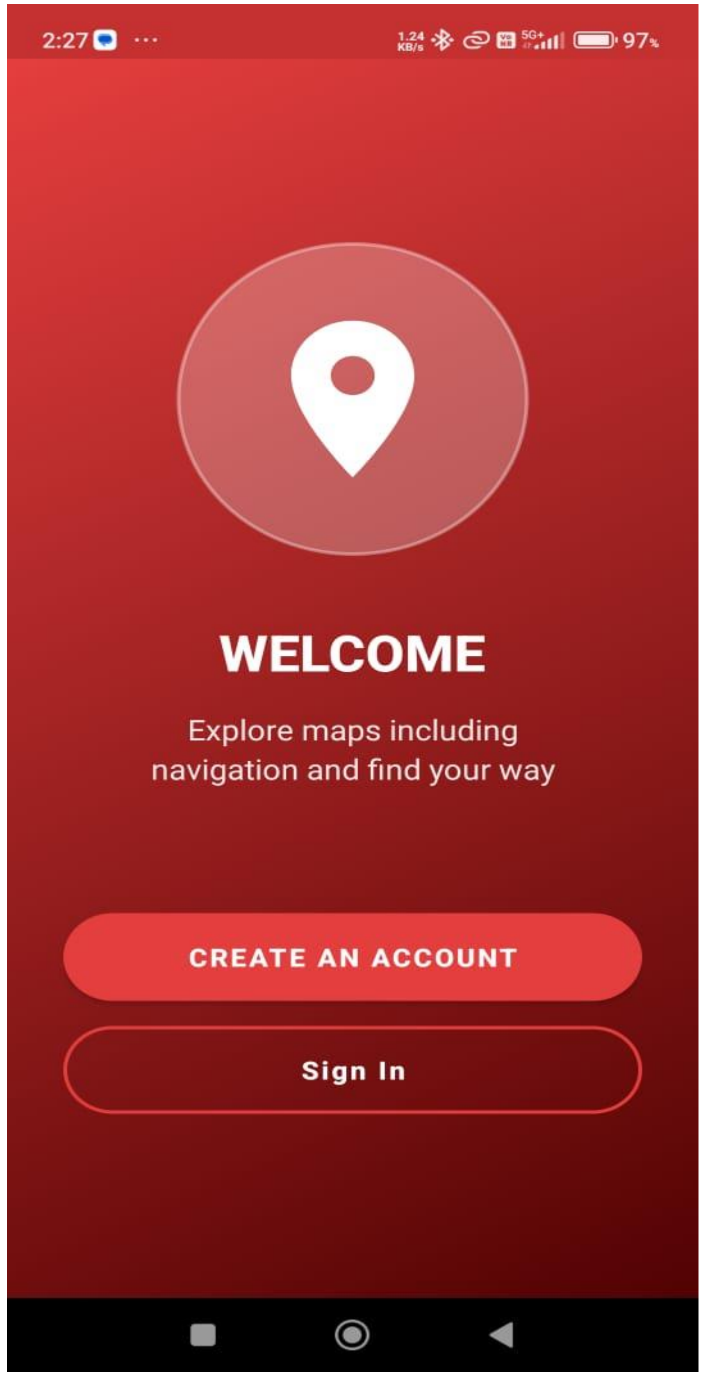


Fig 5.2 Welcome Screen

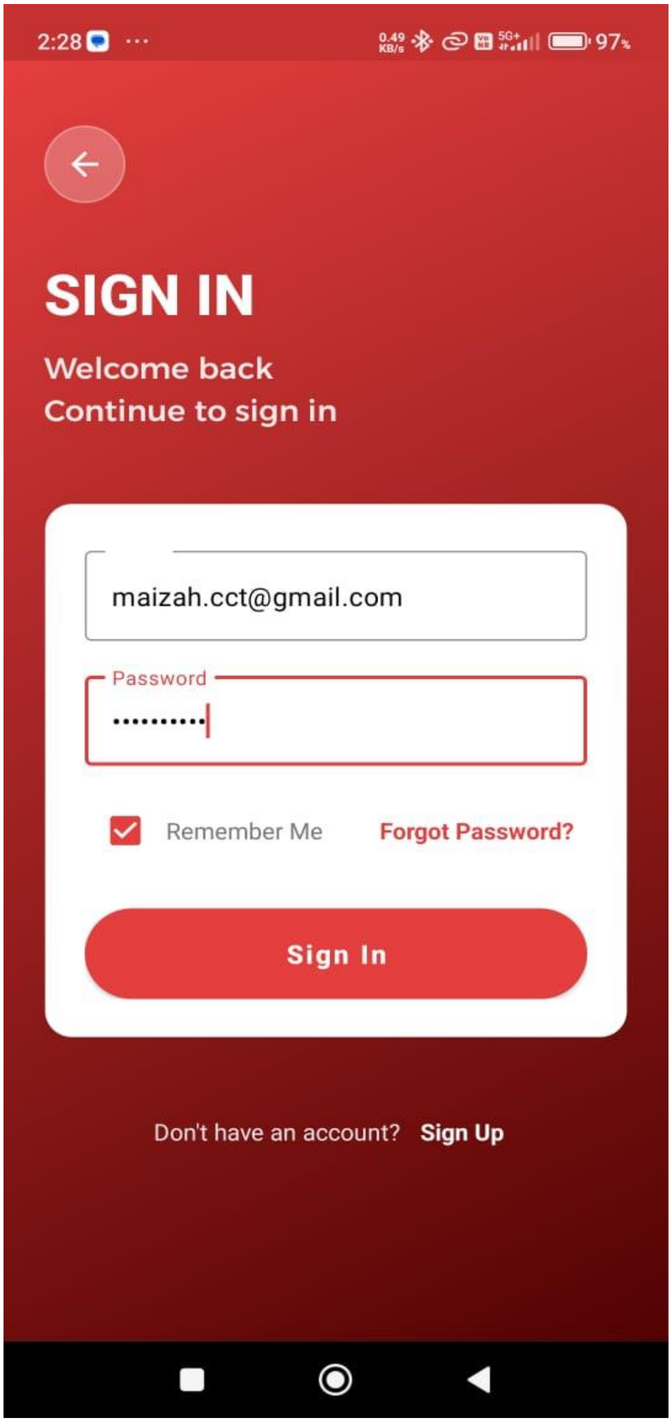


Fig 5.3 Login/Sign Up Page

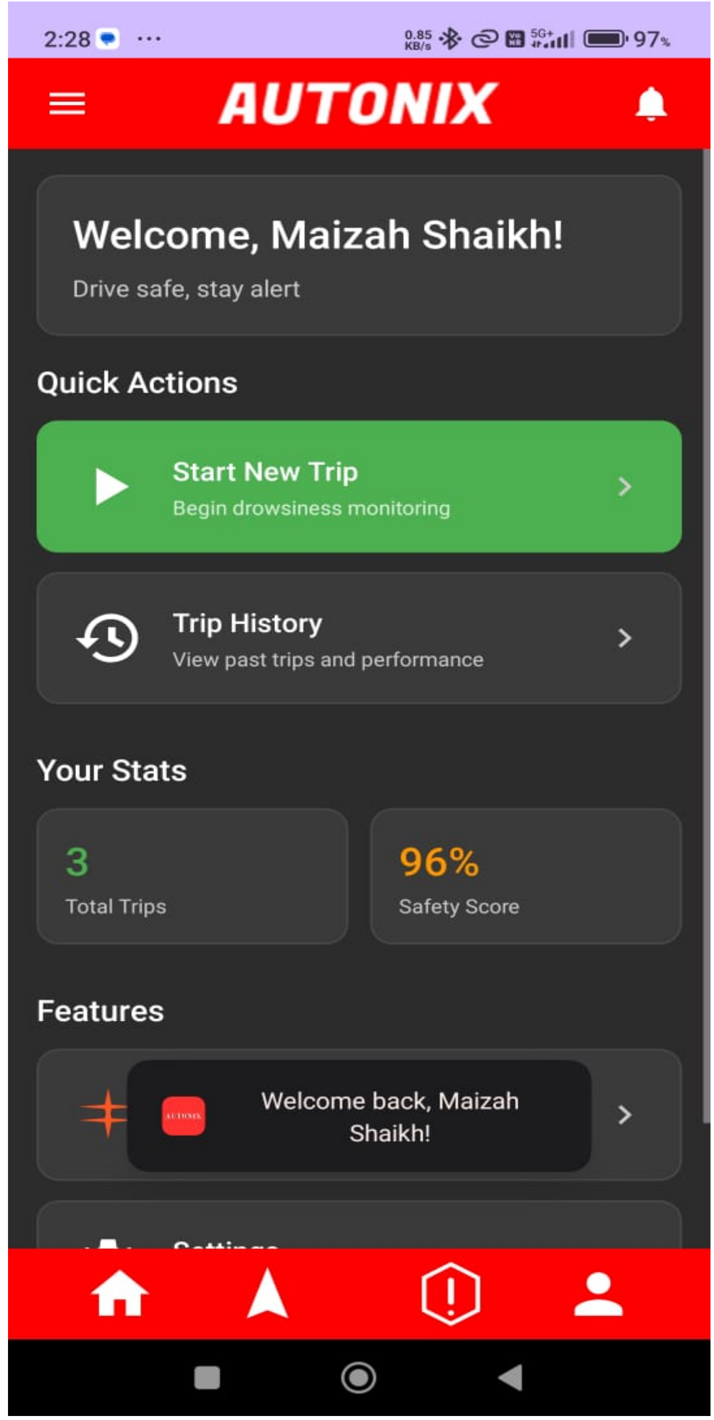


Fig 5.4 Dashboard

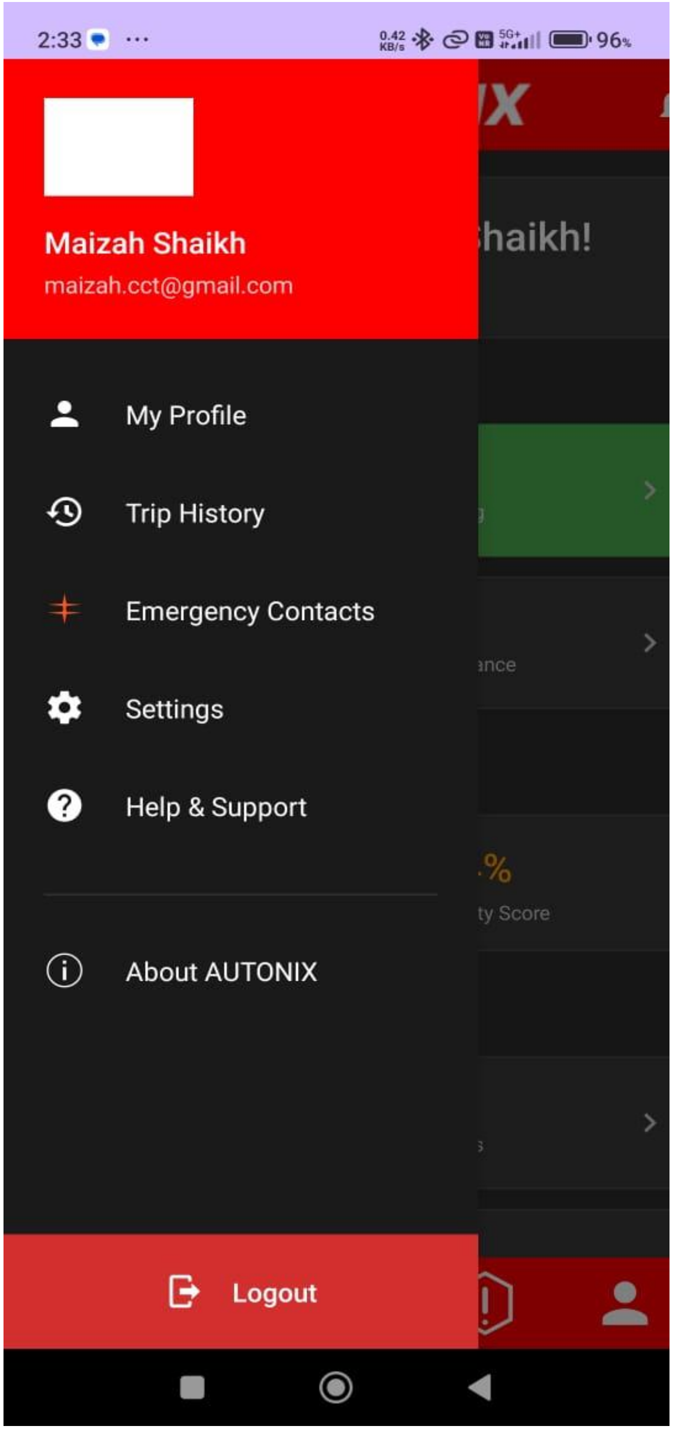


Fig 5.5 Navigation Drawer

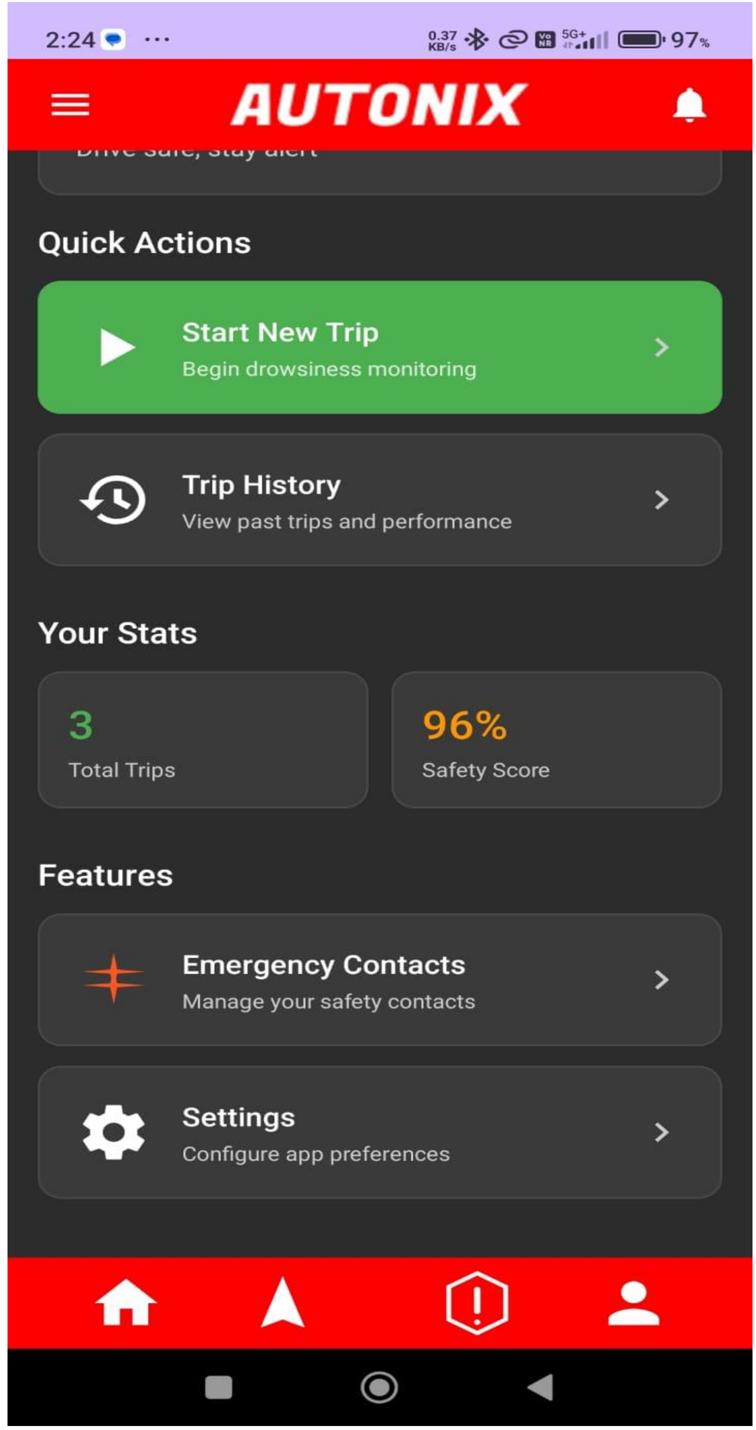
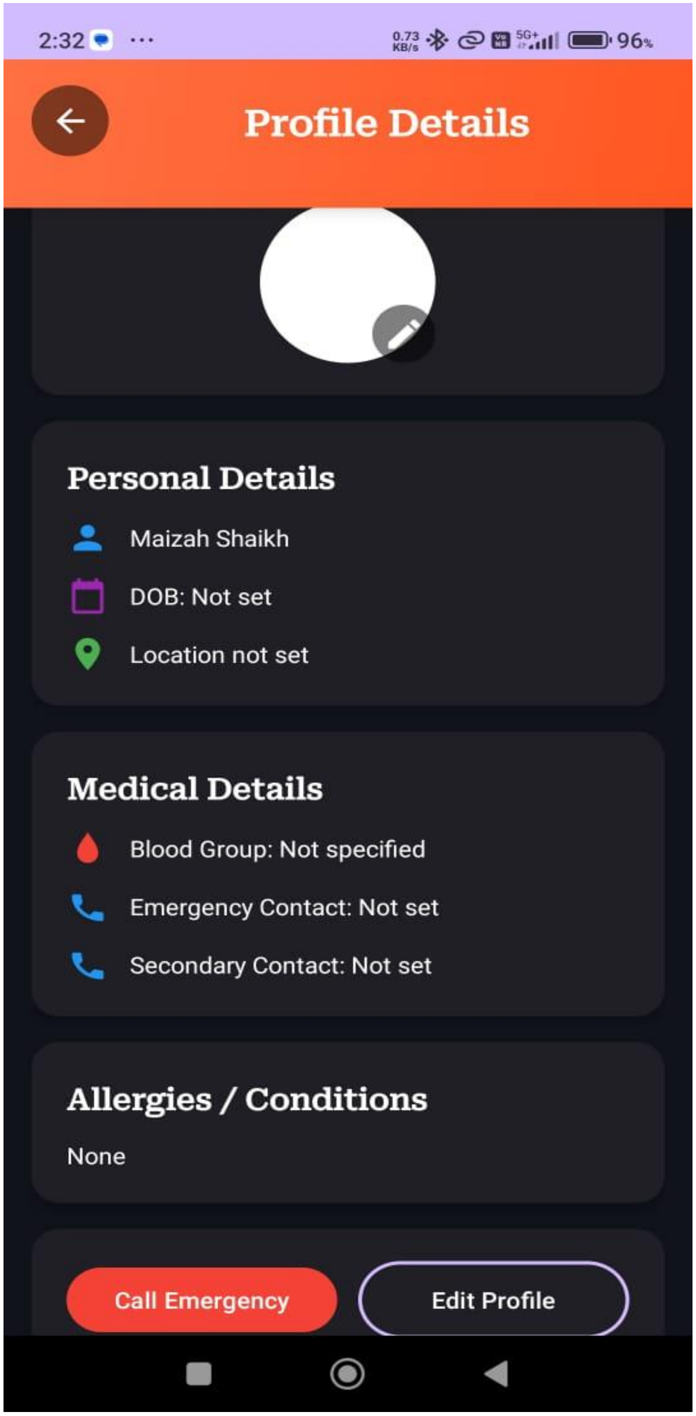


Fig 5.6 Home Screen



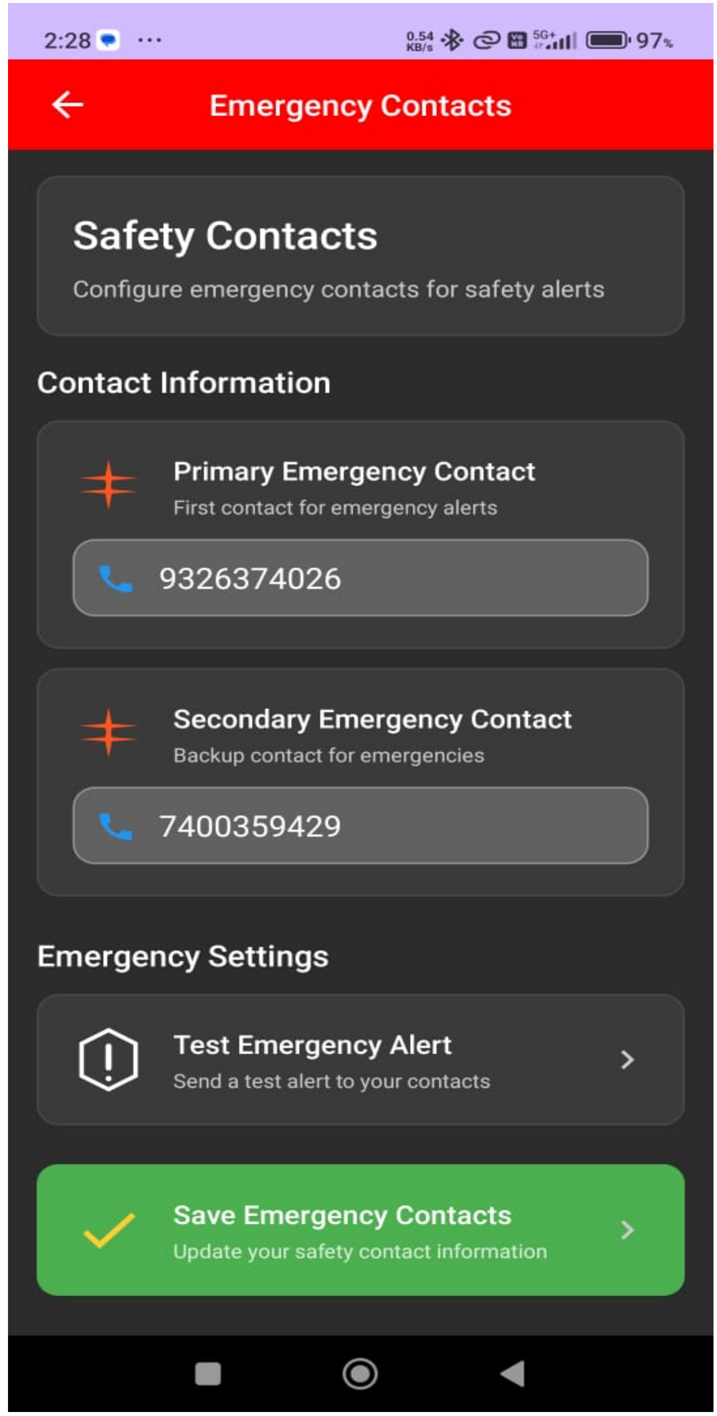
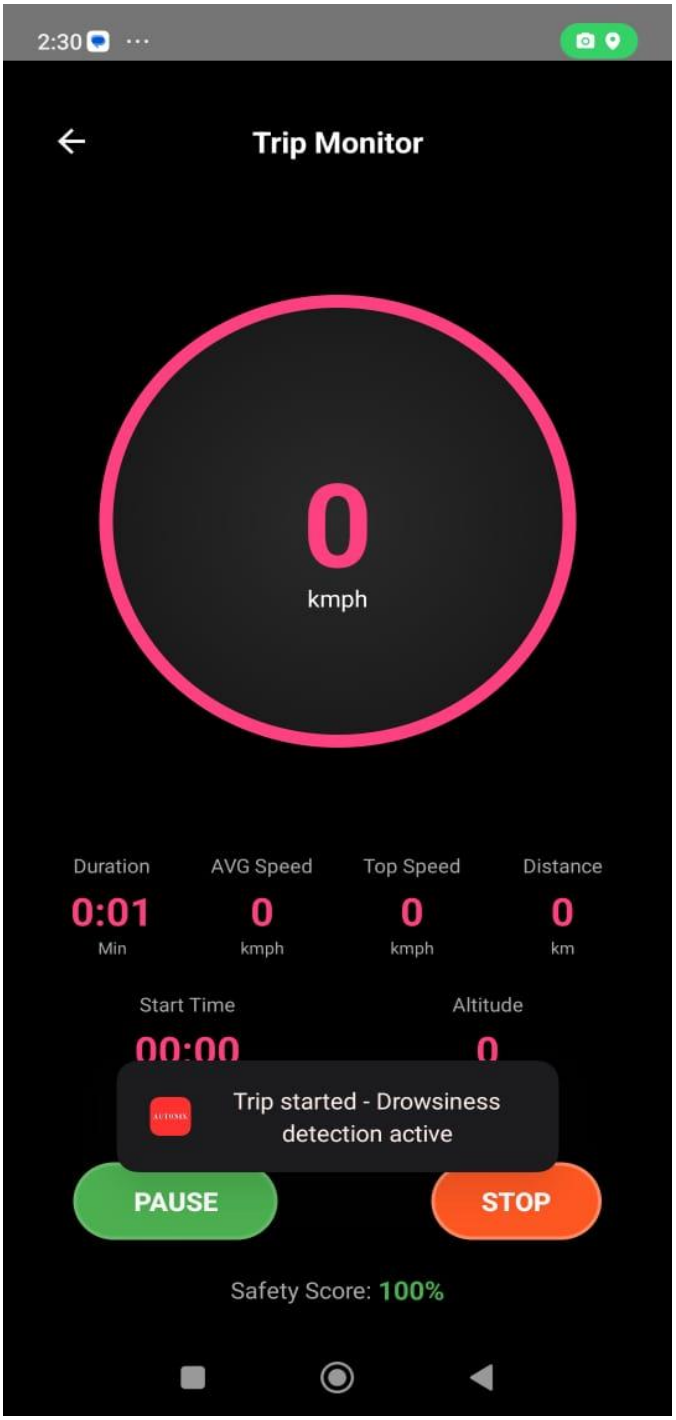


Fig 5.7 Profile Screen

Fig 5.8 Emergency Contact



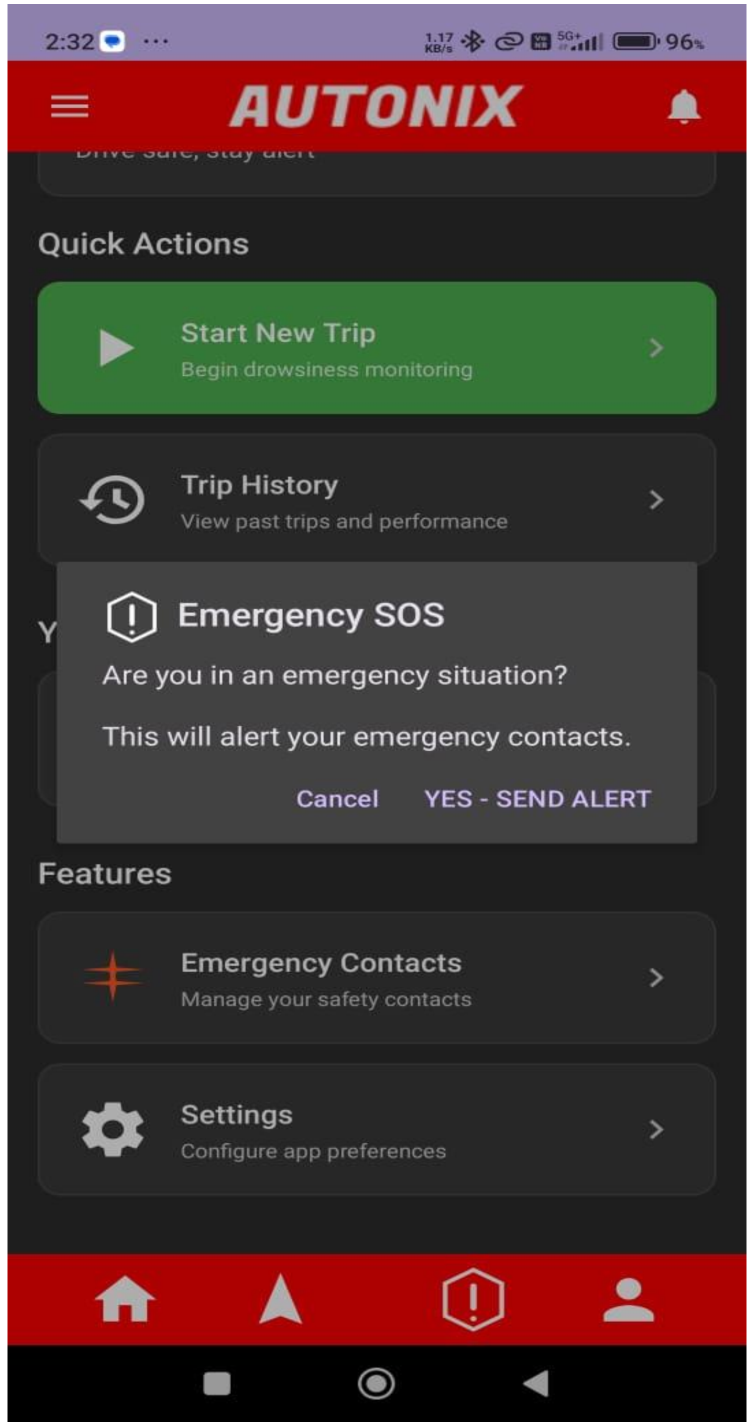


Fig 5.9 Trip Start Screen

Fig 5.10 SOS Screen

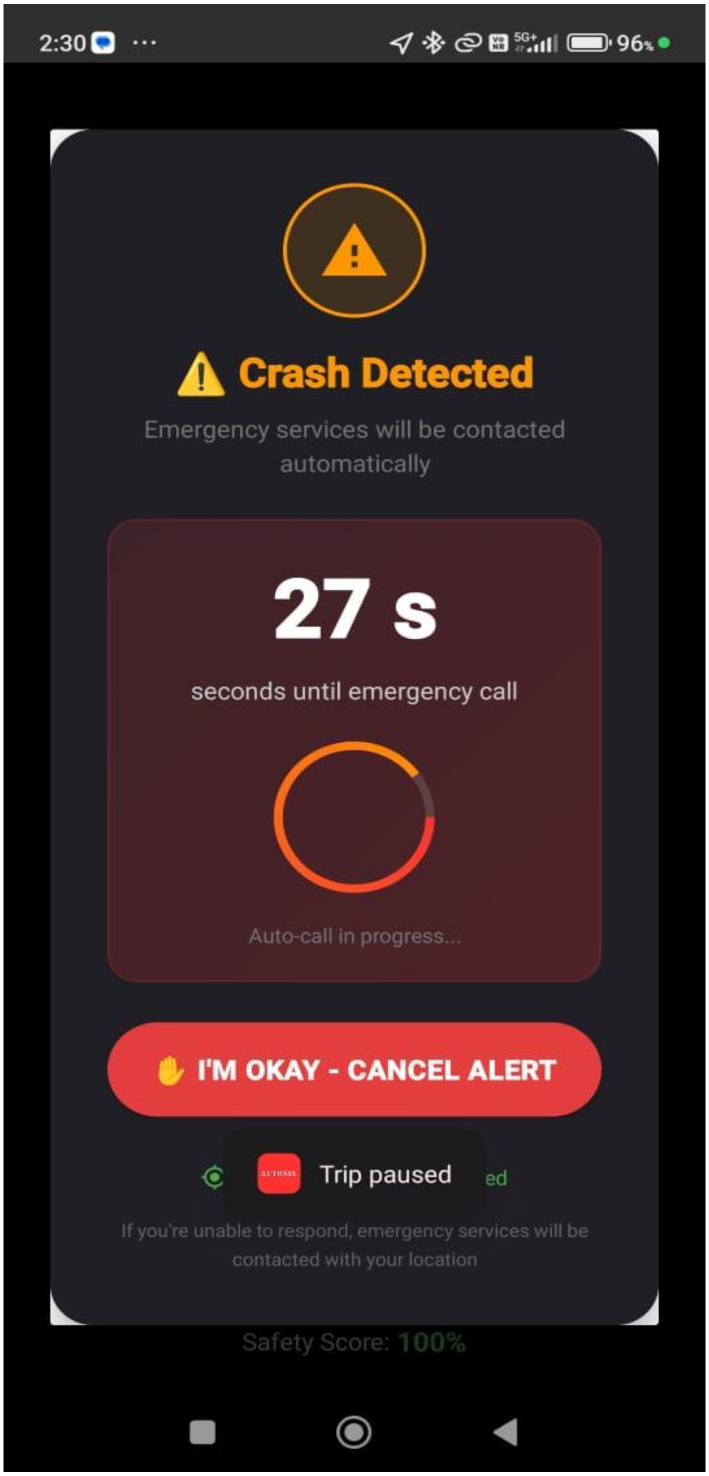


Fig 5.11 Crash Detector Screen

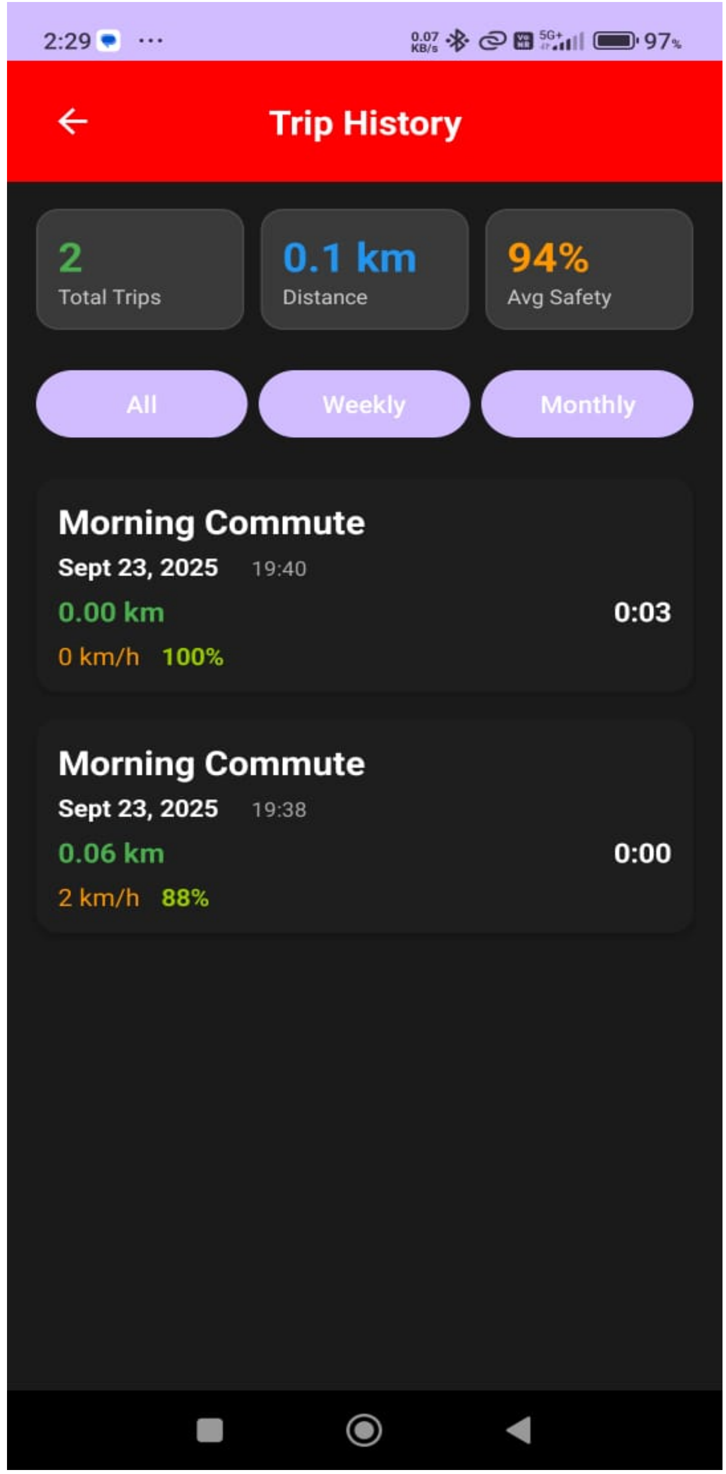


Fig 5.12 Trip Log

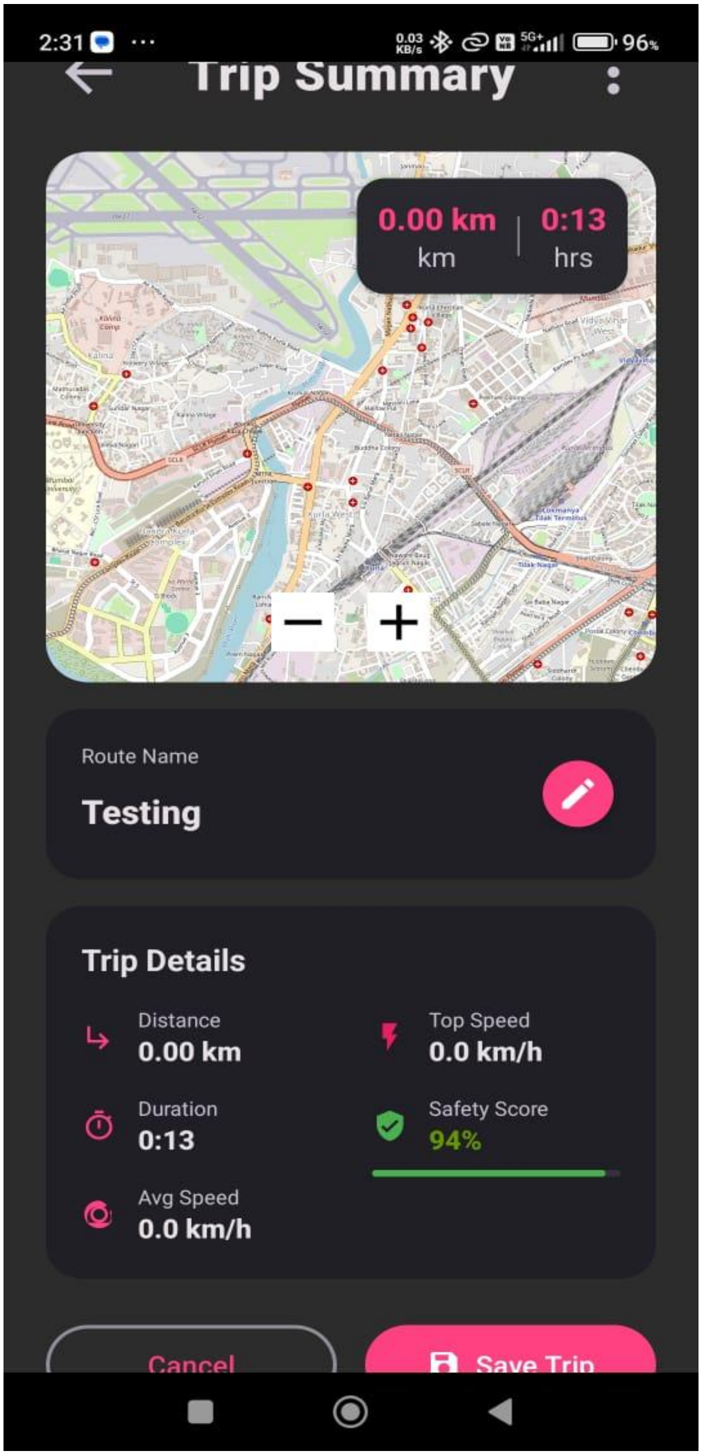


Fig 5.13 Trip Summary Screen

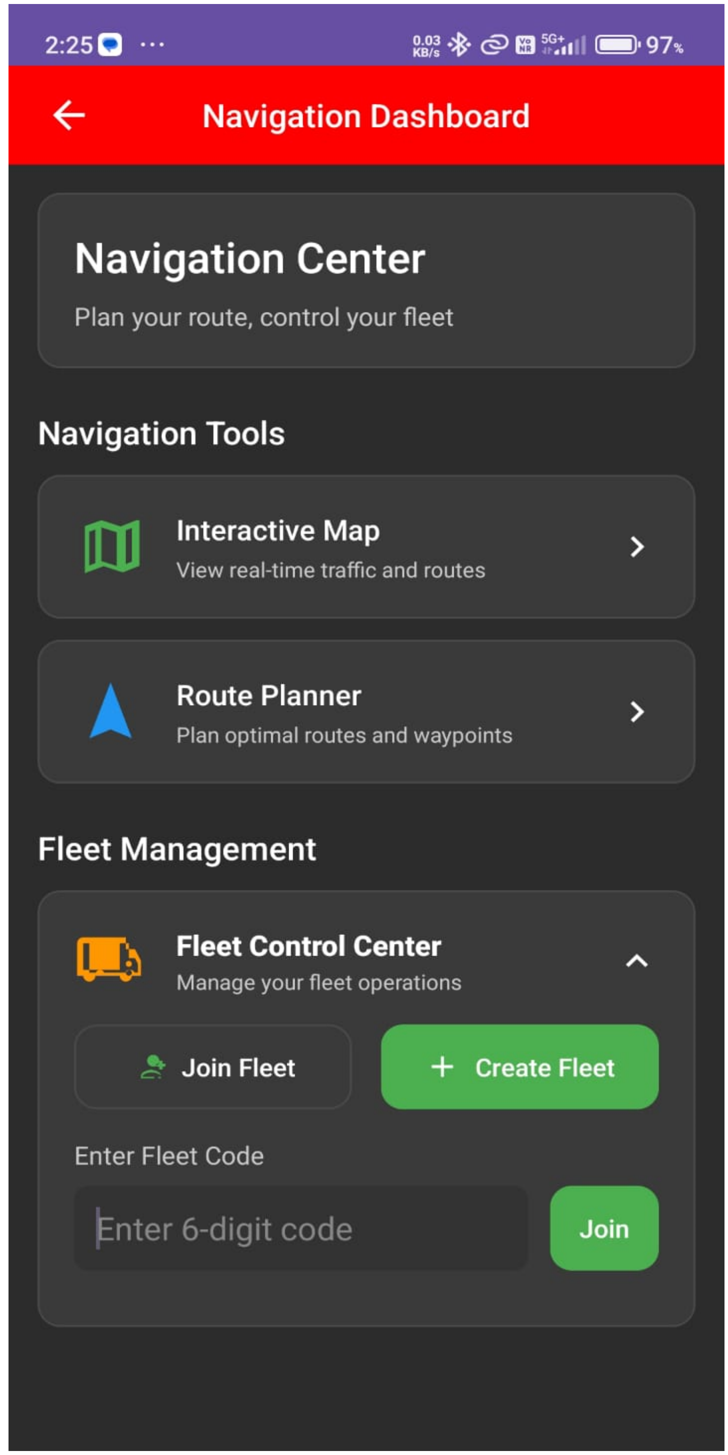


Fig 5.14 Navigation Screeen

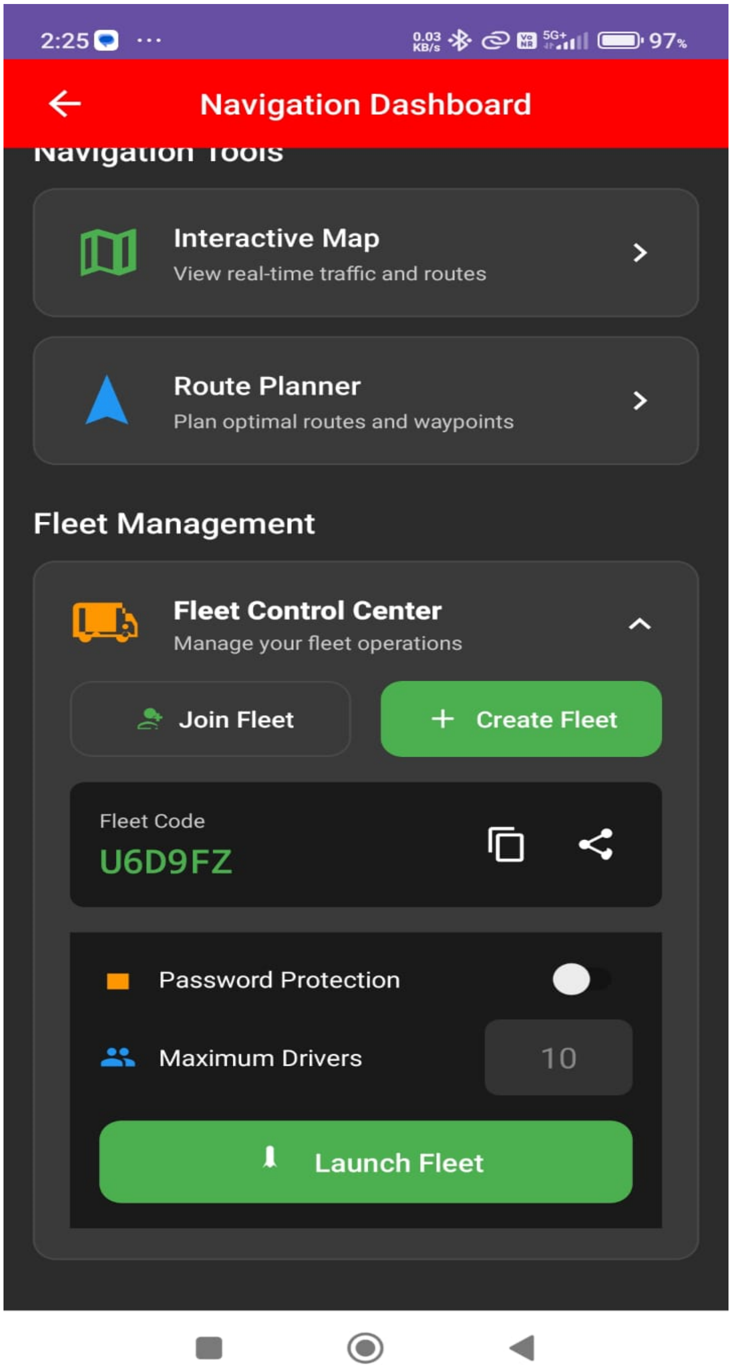


Fig 5.15 Fleet Control

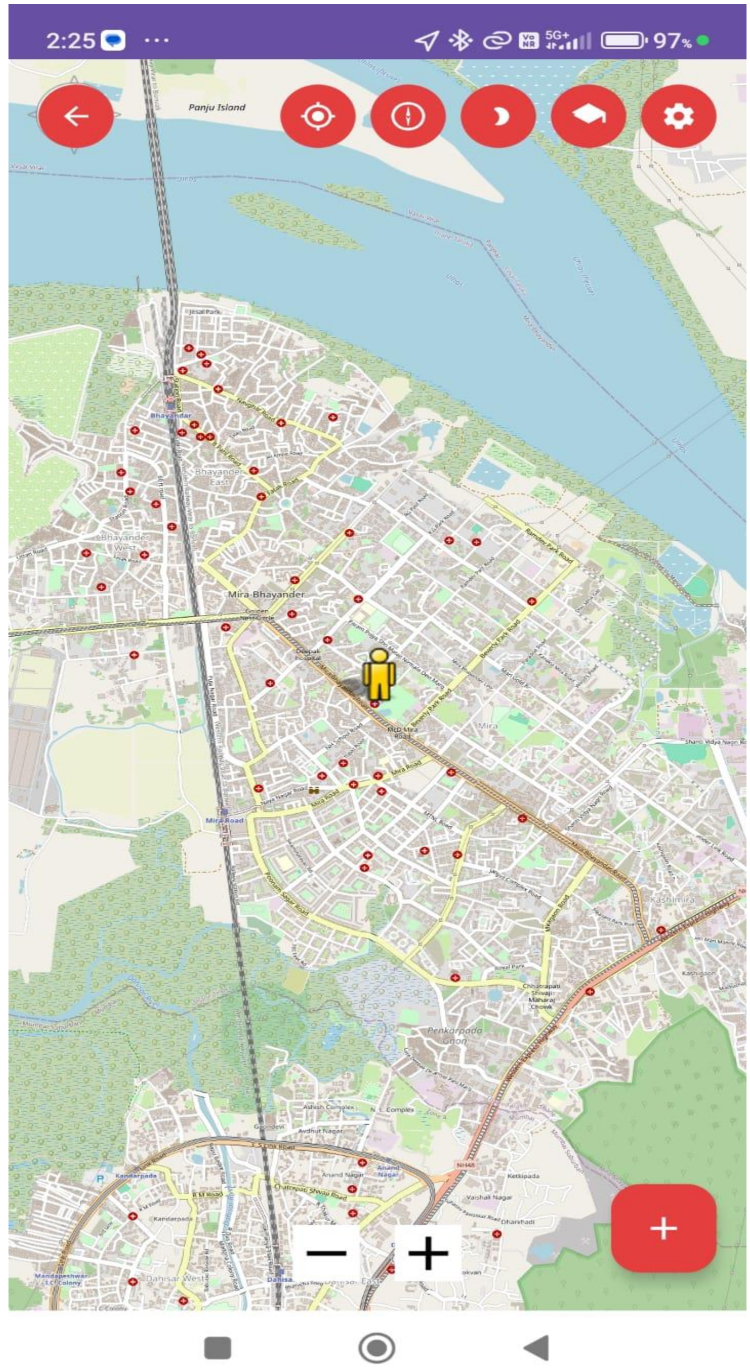


Fig 5.16 Map View

CHAPTER 6
CONCLUSION AND FUTURE WORK

6.1 Conclusion

The Autonix system represents a significant advancement in automotive safety technology, providing real-time drowsiness detection and emergency response capabilities. Autonix empowers drivers and families to monitor driver alertness effectively, facilitating timely interventions to prevent accidents. The integration of comprehensive drowsiness monitoring, automated emergency response, and personalized safety insights creates a proactive safety environment encouraging drivers to take ownership of their road safety.

Autonix's multi-layered approach detects drowsiness indicators including eye closure duration, yawning patterns, and head nodding movements. Firebase integration ensures seamless user management and emergency contact systems, keeping critical safety information readily accessible.

Development revealed improvement areas, particularly reducing false positive alerts and enhancing predictive capabilities. Implementing adaptive threshold adjustments based on individual driver patterns can maintain higher accuracy while minimizing interruptions. Refining Eye Aspect Ratio (EAR) and mouth ratio algorithms to accommodate lighting variations and driver-specific characteristics will enhance reliability and personalization.

The emergency response system provides comprehensive post-incident management through automated SMS alerts with GPS coordinates, SOS signals via flashlight and audio, and real-time location sharing. Multiple emergency contact management ensures help arrives quickly during accidents.

This project demonstrates the transformative potential of AI-driven automotive safety approaches. Through real-time monitoring, predictive analytics, and automated emergency response, Autonix addresses the critical need for advanced driver assistance systems.

6.2 Future Work

As Autonix continues to develop, it is essential to address the identified limitations and expand the system's capabilities to improve driver safety and usability. The following initiatives are planned to strengthen functionality, increase reliability, and make Autonix more impactful across diverse driving contexts. By focusing on advanced detection mechanisms, connectivity, and large-scale adoption, Autonix aims to evolve into a comprehensive and intelligent safety platform.

- **Fleet Management:** Extend Autonix for fleet use, allowing managers to track multiple drivers, monitor safety, and ensure compliance with real-time insights.
- **Advanced Drowsiness Detection:** Use improved computer vision (yawning, head position) for higher accuracy in varied conditions.
- **Emergency Assistance:** Enable direct contact with emergency services in addition to guardian alerts for faster response.
- **Voice Controls:** Add voice commands for starting trips, checking alerts, and dismissing notifications to reduce distraction.
- **Predictive Analytics:** Analyze trip history to predict driver risks and provide preventive feedback.
- **Offline Support:** Enhance offline features so alerts and logging work even without internet.
- **Gamification:** Introduce safe-driving badges, scores, and leaderboards to boost engagement.
- **User Feedback:** Continuously refine features using driver and fleet manager feedback.

CHAPTER 7

REFERENCES AND PLAGIARISM REPORT

References

[1] A. Sahayadhas, K. Sundaraj, and M. Murugappan, *"Detecting driver drowsiness based on sensors: A review,"* Sensors, vol. 12, no. 12, pp. 16937–16953, Dec. 2012, doi: 10.3390/s121216937.

[2] A. Bhanja, D. Parhi, D. Gajendra, K. Sinha, and A. K. Sahoo, *"Driver drowsiness shield (DDSH): A real-time driver drowsiness detection system,"* ROBOMECH Journal, vol. 12, no. 18, pp. 1–15, 2025, doi: 10.1186/s40648-025-00307-4.

[3] N. Irtija, M. Sami, and M. A. R. Ahad, *"Fatigue detection using facial landmarks,"* in Proc. Int. Symp. Affective Sci. Eng. (ISASE), Nov. 2018, pp. C000041-1–C000041 4, doi: 10.5057/isase.2018-C000041.

[4] L. M. E., G. A. Kumar, S. T. Chandru, and H. Rajan, *"Vehicle accident detection and notification system using LoRaWANR,"* Int. Res. J. Modernization Eng. Technol. Sci., vol. 3, no. 6, pp. 1484–1490, Jun. 2021.

[5] M. Jain, S. Jain, and A. Dhawan, *"Driver drowsiness detection and alert system,"* Int. J. Sci. Res. Comput. Sci., Eng. Inf. Technol., vol. 11, no. 2, pp. 3504–3511, Apr. 2025, doi: 10.32628/CSEIT25112815.

[6] S. Essahraui et al., *"Real-time driver drowsiness detection using facial analysis and machine learning techniques,"* Sensors, vol. 25, no. 3, p. 812, Jan. 2025, doi: 10.3390/s25030812.

[7] I. Nasri, M. Karrouchi, K. Kassmi, and A. Messaoudi, *"A review of driver drowsiness detection systems: Techniques, advantages and limitations,"* High School of Technology, Mohammed First University, Oujda, Morocco, 2024/2025.

[8] F. Asim and S. Sunil, *"DrowsyDetect: Android application for driver drowsiness detection using eye closure rate with deep learning,"* Journal of Student Research, vol. 12, no. 2, pp. 1–9, Jun. 2023.

[9] B. Rajkumarsingh and D. Totah, *"Drowsiness detection using Android application and Mobile Vision Face API,"* R&D Journal, vol. 37, pp. 26–34, Nov. 2021.

[10] J. Pansare, R. Deshpande, S. Shingare, H. Deokar, and P. Manwar, *"Real-time driver drowsiness detection with Android,"* International Journal for Research in Applied Science & Engineering Technology (IJRASET), vol. 10, no. 4, pp. 42210 42215, May 2022, doi: 10.22214/ijraset.2022.42210.

Plagiarism Report

Dupli Checker

Date: 30-09-2025

Plagiarism Scan Report

0% Plagiarism	
0% Exact Match	
0% Partial Match	
100% Unique	
Words	465
Characters	3370
Sentences	23
Paragraphs	11
Read Time	3 minute(s)
Speak Time	4 minute(s)

Content Checked For Plagiarism

Introduction

Autonix is an intelligent, smartphone-based driver safety and monitoring system designed to reduce road accidents by actively detecting signs of drowsiness, reckless driving, or sudden crashes. Unlike traditional systems that rely on expensive in-car hardware or are limited to high-end vehicles, Autonix brings advanced safety features to any vehicle using just a smartphone.

The system uses the phone's front camera, GPS, and built-in motion sensors to continuously monitor the driver's face and behavior. It identifies signs of fatigue using facial landmarks and calculates the EAR (Eye Aspect Ratio) to measure eye closure levels. If prolonged drowsiness is detected, the system instantly triggers alerts through sound or vibration to wake the driver and prevent potential accidents.

In addition to drowsiness detection, Autonix tracks real-time driving patterns, detects sudden jerks or harsh braking, and simulates crash logic based on acceleration spikes. If a possible crash is detected, it automatically sends emergency alerts to predefined guardian contacts via SMS or calls, sharing location details to enable immediate assistance.

Autonix also offers a Smart Trip Planner where admins can create trips and assign them to drivers. Drivers can view their trip summaries, receive alerts, and track performance throughout the journey. With built-in navigation powered by OSMdroid, users can follow real-time routes and search locations seamlessly.

By combining safety, communication, and route management in a single mobile solution, Autonix aims to make roads safer for everyone ; not just those with access to luxury vehicle systems. It's a powerful step toward responsible driving, guardian connectivity, and smarter transport monitoring using accessible technology.

1.1 Background and Project Overview

In today's digital era, mobile applications are increasingly expected to deliver not only functionality but also adaptability and responsiveness to user needs. Conventional applications often lack the integration of smart features and real-time support, which limits user engagement and long-term impact. Addressing this gap, the Autonix system was developed as a next-generation mobile application designed to provide a seamless, interactive, and reliable digital experience.

Autonix is built on the Android runtime environment and powered by Firebase backend services, enabling secure authentication, efficient data storage, and real- time synchronization across devices. Its modular architecture ensures scalability, while its intuitive interface offers users a smooth and engaging interaction flow. By prioritizing accessibility and performance, Autonix creates a responsive mobile environment that adapts to diverse user requirements.

The system architecture also supports future enhancements, ensuring long- term sustainability and adaptability as user expectations evolve.

In summary, Autonix represents a significant advancement in mobile application design. By merging a user-friendly interface with real-time cloud-backed functionality, it empowers users with an engaging experience while providing developers with a scalable and maintainable solution. Autonix exemplifies how modern mobile applications can bridge the gap between innovation and usability, preparing digital ecosystems for the challenges of a rapidly changing world.

Matched Source

Congratulations !

No Plagiarism Found

Check By: Dupli Checker

Dupli Checker Date: 30-09-2025

Plagiarism Scan Report

0% Plagiarism	0% Exact Match 0% Partial Match	100% Unique	Words 606 Characters 3725 Sentences 42 Paragraphs 23 Read Time 4 minute(s) Speak Time 5 minute(s)

Content Checked For Plagiarism

1.2 Objectives

Autonix's main aim is to fill in the missing parts of current driver assistance systems, especially those that are costly, too intrusive, or only work on high-end cars. With this project, we're building a smart, affordable, and easy-to-use safety tool that brings intelligent monitoring directly to your smartphone.
The main goals of Autonix are:

1. To improve drowsiness detection by looking at several facial signs, like how long the eyes are closed, how often the person blinks, and signs of facial fatigue, rather than just one thing.
2. To offer safety features that work on any vehicle, without the need for expensive hardware. By using built-in phone features like GPS, accelerometer, and gyroscope, we can detect crashes and track driving behavior on the go.
3. To create a multi-step alert system. It doesn't just warn the driver ; it can also reach out to a guardian or emergency contact if the driver is extremely tired or unresponsive.
4. To test for real-world crashes by looking at sudden changes in speed. If a crash is detected, the system gives the driver a final chance to cancel the alert through a countdown before activating emergency protocols.
5. To provide real-time performance data that shows how often the driver triggers alerts, how safely they're driving, and where they can improve.
6. To keep the system simple and affordable, so that people with regular Android phones can use features that are usually only found in expensive cars.

1.3 Purpose and Scope

1.3.1 Purpose

Autonix is developed with a clear and focused purpose to offer an affordable, non intrusive, and intelligent driver assistance system that enhances road safety, particularly for everyday vehicle users. In today's world, many advanced safety systems are either too costly, limited to premium vehicles, or intrusive in their design. Autonix aims to fill this gap.
The system is designed to detect driver drowsiness and vehicle crashes in real- time using simple camera and sensor inputs. Upon detecting any critical event, Autonix immediately raises alerts, stores essential data, and ensures the driver or caregiver is notified without delay. The idea is to prevent accidents before they happen, and respond quickly if they do.
By eliminating the need for high-end equipment and keeping the process seamless for the user, Autonix brings real-time safety within reach for a much broader audience , making roads safer and responses smarter.

1.3.2 Scope

Autonix is a mobile application focused on helping users stay safe during travel by detecting signs of drowsiness and crash-like movements. It's designed to be simple, lightweight, and accessible, especially for people who don't use high-end vehicles or expensive safety systems. Since it works directly through the user's mobile phone, there's no need for any extra hardware or car-based integration.

The app begins monitoring automatically when the user starts a trip. Throughout the journey, it quietly checks for warning signs like the user dozing off or the phone experiencing a sudden jolt that may indicate a fall or impact. If

something unusual is detected, the app immediately alerts the user and logs the event details for future reference. This way, the user can take action before a situation becomes serious.

- A feature to detect drowsiness by using the phone's front camera to monitor facial cues like eye closure.
- A way to identify crash-like movements using the phone's internal motion sensors.
- A trip start mechanism that activates monitoring when the user begins a journey.
- Alerts in the form of sounds or pop-ups that notify the user in case of drowsiness or sudden motion.
- A simple local log that saves event data during the trip.

Matched Source

Congratulations !

No Plagiarism Found

Check By: Dupli Checker

T. Y. B. Sc. Computer Science

Semester V

A.Y. 2025 – 2026

Accepted Project Proposal

ON

Autonix- Rider Safety Assistance and Monitoring System

Maizah Shaikh
Roll No : 20
Yatin Anchan
Roll No : 23

Title : Autonix- Rider Safety Assistance and Monitoring System

Introduction:

Autonix is an intelligent, smartphone-based driver safety and monitoring system designed to reduce road accidents by actively detecting signs of drowsiness, reckless driving, or sudden crashes. Unlike traditional systems that rely on expensive in-car hardware or are limited to high-end vehicles, Autonix brings advanced safety features to any vehicle using just a smartphone.

The system uses the phone's front camera, GPS, and built-in motion sensors to continuously monitor the driver's face and behavior. It identifies signs of fatigue using facial landmarks and calculates the EAR (Eye Aspect Ratio) to measure eye closure levels. If prolonged drowsiness is detected, the system instantly triggers alerts through sound or vibration to wake the driver and prevent potential accidents.

In addition to drowsiness detection, Autonix tracks real-time driving patterns, detects sudden jerks or harsh braking, and simulates crash logic based on acceleration spikes. If a possible crash is detected, it automatically sends emergency alerts to predefined guardian contacts via SMS or calls, sharing location details to enable immediate assistance.

Autonix also offers a Smart Trip Planner where admins can create trips and assign them to drivers. Drivers can view their trip summaries, receive alerts, and track performance throughout the journey. With built-in navigation powered by OSMdroid, users can follow real-time routes and search locations seamlessly.

By combining safety, communication, and route management in a single mobile solution, Autonix aims to make roads safer for everyone — not just those with access to luxury vehicle systems. It's a powerful step toward responsible driving, guardian connectivity, and smarter transport monitoring using accessible technology.

Motto: "Safety for all. Accessible to all."

Objectives:

1. **Help drivers stay awake** and alert by using a smart system that detects signs of drowsiness through their **eye movements**.
2. **Reduce the chances of accidents** by warning the driver when they're too sleepy or losing focus on the road.
3. Automatically **detect crashes or collisions using mobile sensors**, and take quick action like **alerting emergency contacts**.
4. **Guide drivers safely** to their destination by offering simple, reliable **navigation right from within the app**.
5. Support **fleet managers or owners** by allowing them to assign trips, track driver activity, and ensure planned routes are being followed.
6. Make the app usable by **multiple drivers** on the same device, with each having their **own profile, logs, and performance history**.
7. Keep a **record of each trip** including **drowsiness alerts** and **performance scores**, to help analyze driving behavior over time.

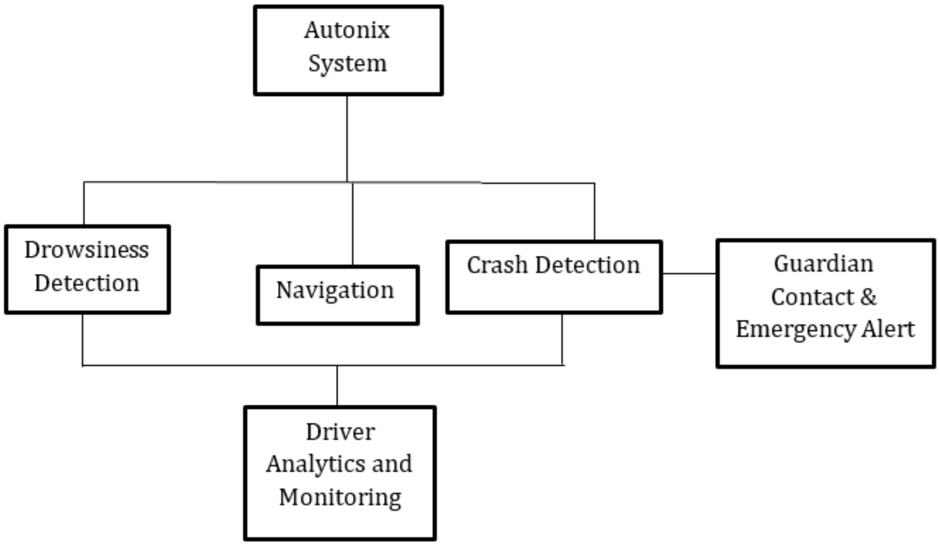
Scope:

The scope of **Autonix** includes the development of a mobile application that enhances **road safety** by **monitoring** driver behavior and responding to **real-time risks** such as **drowsiness**, **crashes**, and **unsafe driving patterns**. The app will use a smartphone's builtin **camera**, **accelerometer**, **gyroscope**, and **GPS** to detect and log critical events.

Key features include:

1. **Real-time drowsiness detection** through eye movement and alertness monitoring.
2. **Crash/accident detection** with emergency contact alerts and countdown-based response.
3. **Driving behavior analysis** using sensor data to assign a safety score per trip.
4. **Guardian and fleet monitoring** with optional live trip status and event notifications.
5. **Smart trip planning** with rest stop suggestions and weather or risk-based route updates.

Methodologies:



1.Development Approach-

For the development of the Autonix system, we adopted the **Iterative and Incremental Development Model**, which allowed us to gradually build and improve individual components of the system. This model was chosen due to the exploratory nature of the project, the requirement for ongoing research integration, and the need for frequent testing and refinement of real-time safety features.

2. System Modules-

The entire system was divided into multiple logical modules for focused implementation:

1. **Drowsiness Detection System**
2. **Rage and Crash Buster Module**
3. **Driving Behaviour Analysis**
4. **Real-time Notifications and Alerts**

Tools and Technologies:

Category	Tools / Technologies	Purpose
Mobile App Development	Android Studio, XML, Java/Kotlin	Building and designing Android application
Backend / Database	Firebase	User authentication, data storage
UI Design	XML (for Android), Canva (for mockups)	Designing and styling UI/UX
Testing Tools	Manual testing, Debugging tools in Android Studio	Module-wise testing and validation
Version Control	Git, GitHub	Code collaboration and version management
Deployment	Android APK build, Play Store (future)	Distributing the app

Timeline

TYBSc Computer Science Semester 5 Project Gantt Chart	Time Requirements	Year 2025-2026												
		June	July				August				September			
		W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
Requirement Gathering & Feasibility Study	Estimated													
Project Planning	Estimated													
System Design	Estimated													
Implementation / Coding	Estimated													
Testing	Estimated													
Deployment Preparation	Estimated													

Resources:

Software Requirements

Software / Platform	**Specification / Version**
Android Studio	Latest stable version (>= Flamingo)
Java / Kotlin	Java 17 / Kotlin 1.9 or above
Firebase	Blaze plan (for scalable usage)
Git & GitHub	Git 2.40+

Hardware Requirements

Hardware	**Specification**
Development Machine (Laptop/PC)	Minimum 8GB RAM, i5 or Ryzen 5, SSD storage
Android Smartphone	Android 10 or above, 3GB RAM minimum
Mobile Camera	Minimum 720p resolution
Internet Connectivity	Stable broadband (>= 5 Mbps)

Expected Outcomes:

A smartphone-based driving assistant system that **monitors the driver's behavior**, detects **drowsiness**, and sends timely **alerts**, with optional support for **crash detection**, **trip summaries**, and **guardian/fleet notifications**. The key focus is **safety**, **real-time alerts**, and **affordability**.

References

[1] A. Sahayadhas, K. Sundaraj, and M. Murugappan, *"Detecting driver drowsiness based on sensors: A review,"* Sensors, vol. 12, no. 12, pp. 16937–16953, Dec. 2012, doi: 10.3390/s121216937.

[2] A. Bhanja, D. Parhi, D. Gajendra, K. Sinha, and A. K. Sahoo, *"Driver drowsiness shield (DDSH): A real-time driver drowsiness detection system,"* ROBOMECH Journal, vol. 12, no. 18, pp. 1–15, 2025, doi: 10.1186/s40648-025-00307-4.

[3] N. Irtija, M. Sami, and M. A. R. Ahad, *"Fatigue detection using facial landmarks,"* in Proc. Int. Symp. Affective Sci. Eng. (ISASE), Nov. 2018, pp. C000041-1–C000041 4, doi: 10.5057/isase.2018-C000041.

[4] L. M. E., G. A. Kumar, S. T. Chandru, and H. Rajan, *"Vehicle accident detection and notification system using LoRaWANR,"* Int. Res. J. Modernization Eng. Technol. Sci., vol. 3, no. 6, pp. 1484–1490, Jun. 2021.

[5] M. Jain, S. Jain, and A. Dhawan, *"Driver drowsiness detection and alert system,"* Int. J. Sci. Res. Comput. Sci., Eng. Inf. Technol., vol. 11, no. 2, pp. 3504–3511, Apr. 2025, doi: 10.32628/CSEIT25112815.

[6] S. Essahraui et al., *"Real-time driver drowsiness detection using facial analysis and machine learning techniques,"* Sensors, vol. 25, no. 3, p. 812, Jan. 2025, doi: 10.3390/s25030812.

[7] I. Nasri, M. Karrouchi, K. Kassmi, and A. Messaoudi, *"A review of driver drowsiness detection systems: Techniques, advantages and limitations,"* High School of Technology, Mohammed First University, Oujda, Morocco, 2024/2025.

[8] F. Asim and S. Sunil, *"DrowsyDetect: Android application for driver drowsiness detection using eye closure rate with deep learning,"* Journal of Student Research, vol. 12, no. 2, pp. 1–9, Jun. 2023.

[9] B. Rajkumarsingh and D. Totah, *"Drowsiness detection using Android application and Mobile Vision Face API,"* R&D Journal, vol. 37, pp. 26–34, Nov. 2021.

[10] J. Pansare, R. Deshpande, S. Shingare, H. Deokar, and P. Manwar, *"Real-time driver drowsiness detection with Android,"* International Journal for Research in Applied Science & Engineering Technology (IJRASET), vol. 10, no. 4, pp. 42210 42215, May 2022, doi: 10.22214/ijraset.2022.42210.