**ML-Enhanced Behavioral & Anomaly Detection**

**Web Application Firewal**l

#### *Submitted by*

#### B K SIBIKARTHIK

#### 142224621053

#### Of

**VALLIAMMAI ENGINEERING COLLEGE**

##### A MINI PROJECT REPORT

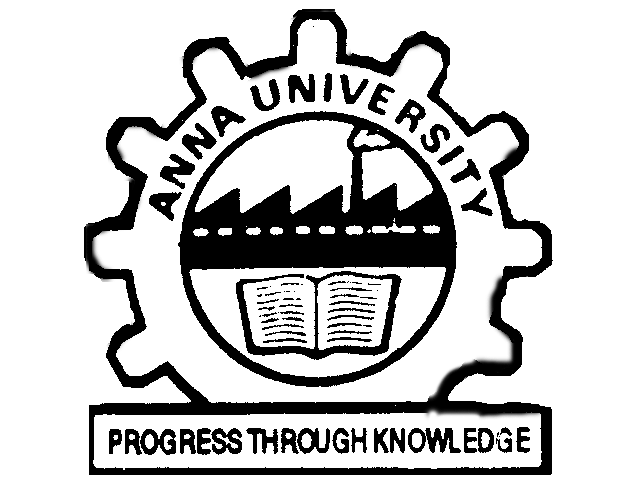
**Submitted to the**

**Faculty of Information and Communication Engineering**

***In partial fulfillment for the award of the degree***

***of***

##### MASTER OF COMPUTER APPLICATIONS



**ANNA UNIVERSITY, CHENNAI**

**MAY, 2025**

**BONAFIDE CERTIFICATE**

Certified that this project report **“**Certified that this project report **“ML-Enhanced Behavioral & Anomaly Detection Web Application Firewal**l**”** is the bonafide work of **“B K SIBIKARTHIK (142224621053)”** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Signature of the Lab In-Charge Signature of the Head of the Department

[Name and Designation] [Name and Designation]

Submitted to the University Examination held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ at Valliammai Engineering College, Kattankualthur.

Internal Examiner External Examiner

# CONTENTS

|  |  |
| --- | --- |
| **CHAPTER** | **Page No.** |
| **ACKNOWLEDGEMENT** |  |
| **SYNOPSIS** |  |
| **I INTRODUCTION** |  |
| 1.1 About the Project |  |
| 1.2 Objective |  |
| 1.3 Problem Statement |  |
| **II Literature Survey** |  |
| **III SYSTEM ANALYSIS** |  |
| 3.1 Existing System |  |
| 3.2 Disadvantages Of Existing System |  |
| 3.3 Proposed System |  |
| 3.4 Advantages Of Proposed |  |
| 3.5 Feasibility Studies |  |
| **IV SYSTEM SPECIFICATION** |  |
| 4.1 Hardware Requirements |  |
| 4.2 Software Requirements |  |
| **4.3 Project Description** |  |
| **4.4 Modules** |  |
| **V SYSTEM DESIGN** |  |
| 5.1 Architecture Design |  |
| 5.2 Structural Design |  |
| 5.3 Behavioural Design |  |
| 5.4 Table Design |  |
| 5.5 User Interface Design |  |
| 5.6 Deployment Design |  |
| 5.7 Navigation Design |  |
| 5.8 Code Design |  |
| **VI SYSTEM IMPLEMENTATION** |  |
| 6.1 Framework Of Demo Implementation |  |
| **VI TEST PLAN** |  |
| **6.1. Testing Strategies** |  |
| **6.1.1 Functional Testing** |  |
| **6.1.2 Verification Testing** |  |
| **6.1.3 Verification Testing** |  |
| **6.1.4 Unit Testing** |  |
| **6.1.5 Integration Testing** |  |
| **6.1.6 System Testing** |  |
| 6.2 Test Cases |  |
| **VII CONCLUSION AND FUTURE ENHANCEMENT** |  |
| 7.1 Conclusion |  |
| 7.2 Future Enhancement |  |

**ACKNOWLEDGMENT**

Apart from our efforts, the success of any project depends largely on the encouragement and guidelines of many others. we take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.

We respect and thank **Dr. M. MURUGAN,** Principal, SRM Valliammai Engineering College, Kattankulathur**,** for providing this wonderful opportunity to work on this project.

We express our sincere gratitude to **Dr. S. PARTHASARATHY,** Professor and Head, Department of Computer Application, SRM Valliammai Engineering College, for his encouragement throughout this project work.

We are indebted to our internal guide**, Mr.M.Asan Nainar,** Associate Professor, (Senior Grade) Department of Computer Application, for her constant support and guidance throughout the project work.

**SYNOPSIS**

This project implements a Web Application Firewall (WAF) enhanced with Machine Learning and rule-based detection to identify and block suspicious web traffic in real time. The system combines an Isolation Forest ML model with custom logic to detect SQL injection, XSS, and brute-force attacks.

Key Components:

ML Anomaly Detection: Uses features from user inputs to detect anomalies via Isolation Forest.

Rule-Based Engine: Detects common attacks using regex patterns (SQLi, XSS, brute-force).

Access & Anomaly Logging: All activities are stored in a SQLite database and log files for auditing.

Secure Login & Registration: Includes password hashing and validation mechanisms.

Admin Dashboard: Displays alerts and encourages testing via simulated attack inputs.

Technologies Used:

Python, Flask, SQLAlchemy, Scikit-learn, SQLite, Joblib, Bcrypt

Outcome:

A real-time, intelligent WAF capable of detecting abnormal behaviors and attack attempts, significantly improving web application security beyond traditional firewalls.

.

## CHAPTER I INTRODUCTION

This chapter deals with the application study of the project domain, software specifications, and a brief overview of the technologies used for the project titled “ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall”..

### 1.1 About the Project

The project titled **“ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall”** is a modern cybersecurity solution aimed at protecting web applications from a wide range of threats using a combination of **machine learning techniques** and **traditional rule-based detection**. With the increasing frequency and sophistication of web-based attacks, such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and brute-force login attempts, there is a growing need for security systems that can not only detect known threats but also adapt to new, unknown attack patterns.

Traditional Web Application Firewalls (WAFs) primarily depend on signature-based detection and fixed rule sets, which are effective for previously identified attacks but fail against novel or obfuscated ones. This project overcomes these limitations by integrating an **Isolation Forest** algorithm to analyze and learn normal behavior patterns from input data. The system then flags any deviations from this pattern as potential anomalies, allowing it to catch suspicious activity even if it doesn’t match predefined rules.

The firewall also includes a **rule-based detection module** that uses regular expressions to scan for specific keywords and patterns associated with known attack vectors, such as SELECT FROM, <script>, and suspicious payloads in HTTP requests. In addition, the system monitors user behavior for signs of brute-force attacks by tracking login attempts over time.

A key feature of this project is the real-time logging of both normal access and anomalies into a secure **SQLite database**, allowing for post-analysis and auditing. The system is built using Python and Flask, with secure password handling via **bcrypt** hashing. A simple web interface demonstrates the working of the system and allows users to interact with login and registration forms, test various inputs, and observe the firewall's responses.

By combining machine learning and rule-based detection, this project presents a **hybrid approach to web security**, offering greater adaptability, lower false positive rates, and better protection against both common and evolving threats. It provides a practical demonstration of how intelligent systems can enhance cybersecurity in real-world web applications

### 1.2 Objective

### The main objective of the project titled **“ML-Enhanced Behavioral & Anomaly Detection Web Application Firewall”** is to develop a smart and adaptive Web Application Firewall (WAF) that goes beyond traditional rule-based systems by integrating Machine Learning (ML) techniques for improved security. In today’s digital world, web applications are increasingly exposed to sophisticated threats such as SQL Injection, Cross-Site Scripting (XSS), and brute-force attacks. Conventional WAFs often fail to detect such threats due to their reliance on static signatures and fixed patterns.

### This project aims to address those limitations by using behavioral analysis and anomaly detection to identify unusual activities and block potentially harmful requests. The core of the system uses an Isolation Forest ML model that can learn from normal user behavior and detect deviations that may indicate an attack. It evaluates features such as input length, presence of suspicious keywords (like <script> or SQL terms), and the frequency of login attempts to make decisions.

### The system also includes a rule-based detection layer to instantly identify well-known attack patterns. These include specific regular expressions to detect SQL keywords, embedded scripts, and JavaScript-based payloads. Together with ML, this hybrid approach enhances both accuracy and response speed.

### Another important objective is to ensure real-time logging and monitoring. Every access and detected anomaly is recorded in an SQLite database, allowing administrators to track and analyze threat patterns. The project also focuses on user authentication, using bcrypt hashing to ensure passwords are stored securely and login attempts are protected against brute-force attacks.

### In summary, the key objectives of this project are:

### To improve the ability to detect unknown threats using machine learning.

### To reduce false positives by analyzing behavioral patterns.

### To combine ML and rule-based techniques for higher accuracy.

### To implement real-time logging for security analysis.

### To provide a secure and user-friendly interface for testing and monitoring.

### Through these goals, the project delivers a modern, intelligent firewall that enhances the safety of web applications in a dynamic threat environment.

### **1.3 Problem Statement**

In the rapidly evolving digital landscape, web applications have become critical platforms for communication, commerce, and data exchange. However, they are also frequent targets for cyberattacks such as **SQL Injection (SQLi)**, **Cross-Site Scripting (XSS)**, and **brute-force attacks**. These threats exploit vulnerabilities in user input fields, authentication mechanisms, and request handling logic. Traditional Web Application Firewalls (WAFs), which rely on static rule sets and signature-based detection, are often inadequate in addressing these evolving attack methods.

The major limitation of existing WAFs is their **inability to detect zero-day attacks** or **intelligent evasion techniques**, where attackers subtly modify their payloads to bypass fixed rules. Moreover, relying solely on predefined rules can lead to a high number of **false positives**, disrupting normal user activities and degrading the user experience. As threats become more sophisticated and dynamic, there is an urgent need for a **more adaptive and intelligent security solution** that can evolve alongside attack patterns.

Another significant issue is the lack of **behavioral analysis** in traditional systems. These firewalls do not learn from historical data or user behavior patterns, making them blind to abnormal usage trends that might indicate malicious intent. For example, repeated login attempts from the same IP in a short span may indicate a brute-force attack, but a rule-based system might not flag it unless explicitly configured.

This project addresses the problem by introducing a **Machine Learning-enhanced Web Application Firewall** that can **analyze user behavior**, detect anomalies, and classify potential threats in real-time. By using algorithms like **Isolation Forest**, the system can distinguish between normal and suspicious traffic based on features such as input length, special characters, and keyword presence. The firewall also incorporates pattern-matching techniques to detect known attack signatures while learning from new data to improve accuracy over time.

Thus, the problem this project solves can be summarized as:

"How can we enhance traditional Web Application Firewalls using machine learning and behavioral analysis to detect both known and unknown web threats more accurately and with fewer false positives?"

By solving this problem, the project contributes to a **smarter, adaptive, and more secure** web environment, capable of protecting against both conventional and emerging threats.

## CHAPTER II

## **LITERATURE SURVEY**

### ****2.1 Introduction****

With the growing reliance on web-based systems, security has become a central concern. Traditional Web Application Firewalls (WAFs) primarily use static rule sets to filter out malicious requests. However, modern threats are increasingly dynamic and sophisticated, requiring more intelligent detection mechanisms. This literature survey explores prior research and technologies related to web application security, machine learning in anomaly detection, and hybrid WAF systems. It forms the foundation for understanding the scope and innovations of the proposed system.

### ****2.2 Traditional Web Application Firewalls****

Traditional WAFs use **signature-based** and **rule-based** detection techniques to identify threats. These systems scan incoming requests and match them against predefined patterns to detect attacks such as SQL injection, XSS, and CSRF. Tools like **ModSecurity** and **Cloudflare WAF** are widely used and can block known threats effectively.

However, their limitations include:

* Inability to detect **zero-day** or **evasive attacks**.
* High **false positive** and **false negative** rates.
* Need for **constant updates** to rules/signatures.

**Reference:**

Curphey, M., & Arkin, B. (2002). Web application security assessment tools. IEEE Security & Privacy.

### ****2.3 Behavior-Based Intrusion Detection****

Behavior-based detection focuses on monitoring and learning normal user behavior to identify anomalies. This approach does not rely on known attack signatures but instead flags deviations from expected behavior as potential threats. It is especially useful for detecting **novel attacks** and **insider threats**.

Machine Learning models such as **k-means clustering**, **One-Class SVM**, and **Isolation Forest** are commonly used in this area. Isolation Forest, in particular, is effective for detecting anomalies in high-dimensional data.

**Reference:**

Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. ICDM 2008.

### ****2.4 Machine Learning in Web Security****

ML-based anomaly detection has proven effective in cybersecurity applications. These techniques can classify user inputs based on features such as input length, frequency of special characters, use of suspicious keywords, etc.

Key techniques used:

* **Supervised Learning**: Requires labeled data, used for classification of known attack types.
* **Unsupervised Learning**: Suitable for detecting unknown or zero-day attacks (e.g., Isolation Forest).
* **Feature Engineering**: Key to extracting meaningful patterns from user inputs.

**Reference:**

Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. IEEE Communications Surveys & Tutorials.

### ****2.5 Hybrid Detection Systems****

Hybrid systems combine rule-based and machine learning techniques for more accurate detection. The rule-based component ensures known threats are immediately blocked, while the ML component detects previously unseen patterns. This approach reduces the **false positive rate** and improves the system’s adaptability.

Some existing research shows that hybrid models outperform single-approach methods in terms of detection accuracy and scalability.

**Reference:**

Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. IEEE Symposium on Security and Privacy.

### ****2.6 SQL Injection and XSS Detection****

SQL Injection and XSS are two of the most common attacks on web applications.

* **SQL Injection** involves injecting malicious SQL code through user inputs to manipulate databases.
* **XSS** allows attackers to inject malicious scripts into web pages viewed by other users.

Regular expressions and pattern-matching techniques are commonly used for detection. However, they are static and often bypassed. Combining them with ML allows dynamic detection.

**Reference:**

Halfond, W. G., Viegas, J., & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. ACM.

### ****2.7 Logging and Analysis in Security Systems****

Logging is essential in tracking user activities and suspicious events. Most modern systems store access logs and anomaly logs in databases for post-analysis. This helps in auditing, real-time monitoring, and forensic investigation after an attack.

Database logging also enables the system to learn from past anomalies and improve its detection over time.

**Reference:**

Kent, K., & Souppaya, M. (2006). Guide to Computer Security Log Management. NIST Special Publication.

### ****2.9 Brute Force Attack Detection****

Brute-force attacks attempt to gain unauthorized access to systems by systematically trying many combinations of usernames and passwords. Traditional systems often struggle to detect such attacks in real-time, especially when attackers use **distributed IPs** or slow down their attempts to avoid detection.

Modern systems implement **rate-limiting**, **login attempt tracking**, and **IP blacklisting** as countermeasures. However, these static measures can be bypassed. Machine learning can track behavioral patterns and detect unusual login activity over time. For example, if a user attempts to log in multiple times in a short period, the system can flag it as an anomaly even if each attempt is valid on its own.

**Reference:**

Chakrabarti, A., & Manimaran, G. (2002). Internet infrastructure security: A taxonomy. IEEE Network.

### ****2.10 Real-World Use Cases of ML in WAFs****

Several organizations and startups are now incorporating AI/ML into their WAF solutions:

**Imperva** and **Cloudflare** have started integrating behavior analytics into their enterprise-level products.

**Microsoft Azure WAF** includes anomaly detection powered by machine learning to identify deviations in API usage.

Open-source tools such as **WAF-FLE** have introduced basic ML-driven traffic filtering.

These tools demonstrate the real-world effectiveness of machine learning models in detecting previously unseen attacks and improving the overall security posture.

**Reference:**

Cloudflare (2023). Cloudflare WAF Documentation. [cloudflare.com]

### ****2.11 Challenges in Machine Learning-based WAFs****

While ML-powered WAFs offer improved detection capabilities, they also come with certain challenges:

* **Feature Engineering**: Identifying relevant features from raw HTTP requests is non-trivial.
* **Labeling Data**: Supervised models require large datasets of labeled traffic (normal vs. malicious), which are hard to obtain.
* **Real-Time Performance**: ML models must process requests in milliseconds to avoid latency.
* **False Positives**: Incorrectly flagging legitimate users can cause loss of trust or user abandonment.

To address these, hybrid models are preferred where rule-based detection handles known threats quickly, and ML-based detection works in the background for anomaly analysis.

### ****2.12 Importance of Logging and Monitoring****

Monitoring access logs and maintaining a history of anomaly detection events is critical for:

* **Post-breach investigation**
* **Compliance and audits (e.g., GDPR, HIPAA)**
* **Model retraining and tuning**

Storing logs in a structured database like SQLite and analyzing trends over time enables long-term threat intelligence gathering.

In the proposed system, the logs are separated into:

* **Access Logs**: Record all normal and suspicious access attempts.
* **Anomaly Logs**: Store detected threats with metadata for analysis.

### ****2.13 Role of Password Security in Web Apps****

User authentication is the primary barrier against unauthorized access. Many breaches result from weak or reused passwords. The use of **bcrypt hashing** ensures that passwords are not stored in plaintext, mitigating risks in the event of a data breach.

The project enforces a **minimum password length policy** and uses **bcrypt** for salting and hashing passwords securely. This aligns with best practices recommended by OWASP and NIST.

**Reference:**

OWASP Foundation. (2023). Authentication Cheat Sheet.

### ****2.8 Summary****

The literature reviewed above highlights the evolving nature of web security threats and the corresponding shift toward intelligent, behavior-based solutions. While traditional WAFs still serve as a foundation, they must be enhanced with adaptive technologies like **machine learning** to effectively counter sophisticated attacks. The proposed system in this project builds on this research by integrating both rule-based detection and **Isolation Forest-based anomaly detection** to form a **hybrid, intelligent WAF** capable of protecting modern web applications.

## CHAPTER III

**III SYSTEM ANALYSIS**

### ARCHITECTURAL DIAGRAM

The architecture design model conveys a detailed description and presents a brief overview about the architecture of the application

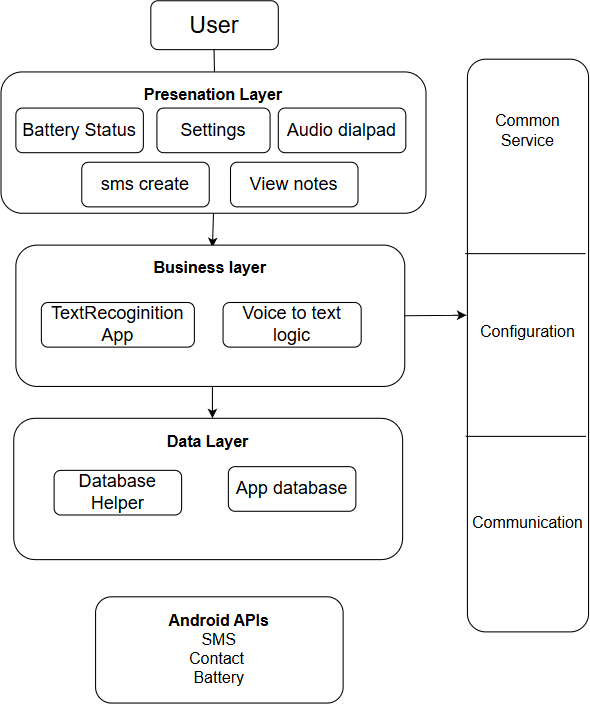


Figure 3.1 (Architectural diagram)

#### User

The User interacts with the app through the Presentation Layer using voice commands or simple UI elements (designed to assist visually impaired users).

#### Presentation Layer

This is the User Interface layer. It includes:

* Battery Status: Speaks out the current battery percentage.
* Audio Dialpad: A voice-navigated number pad to make calls.
* SMS Create: Allows users to create and send messages through voice input.
* View Notes: Users can listen to their saved notes or recordings.
* Settings: Allows users to turn on/off wifi and Bluetooth.

#### Business Layer

This layer handles the core logic and operations:

* TextRecognition App: Processes text from voice inputs or other sources.
* Voice to Text Logic: Converts spoken words into text and routes it to relevant features (SMS, Notes, etc.).

#### Data Layer

This layer is responsible for data storage and retrieval:

* Database Helper: Manages database queries and operations.
* App Database: Stores user notes.

#### Android APIs Used

The **Vision2C** app utilizes essential Android built-in APIs to support key functionalities tailored for visually challenged users:

* SMS API
* Contacts API
* Battery API

### STRUCTURAL DESIGN CLASS DIAGRAM

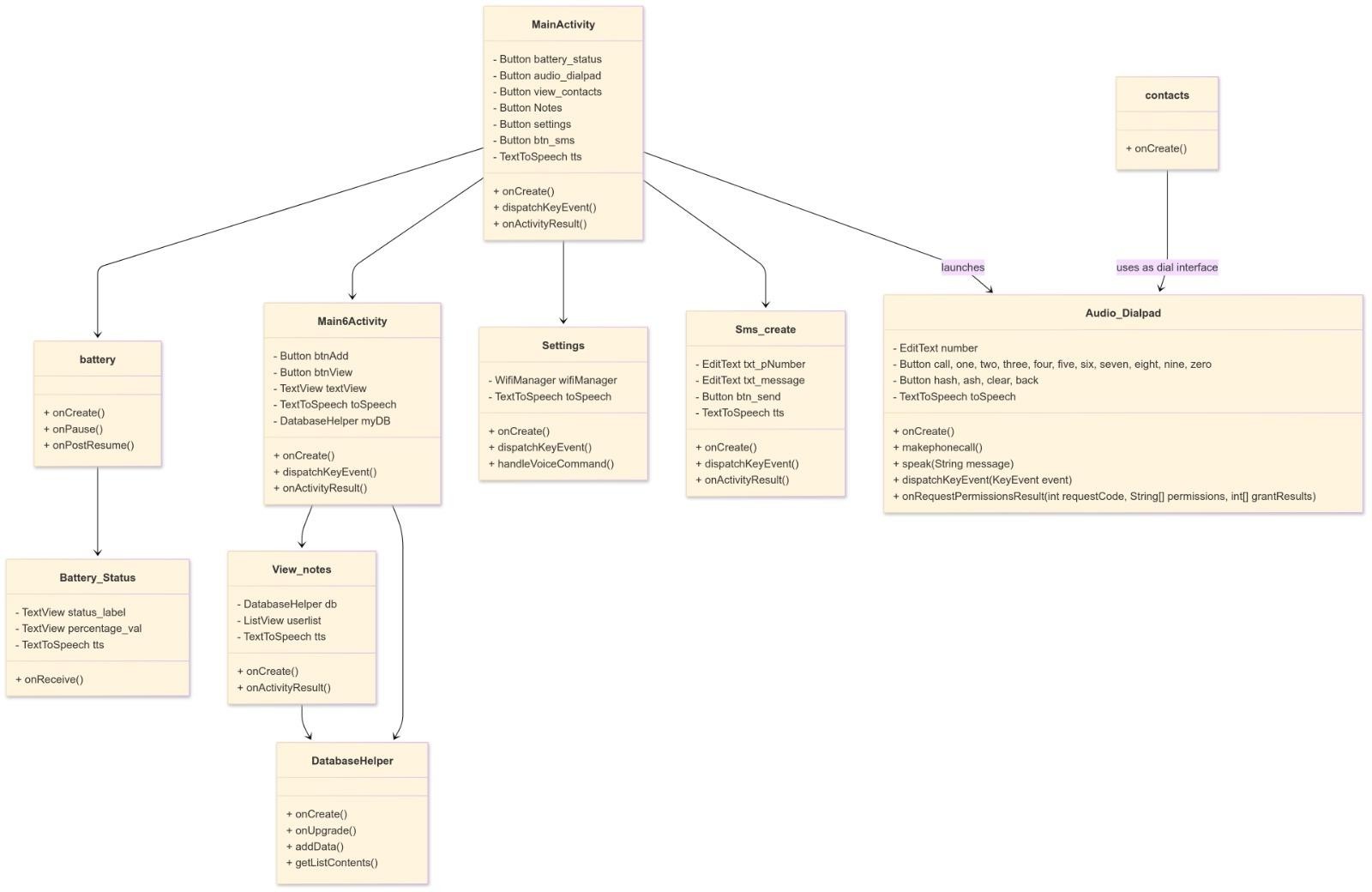
****

Figure 3.2 (Class diagram)

#### Mainactivity:

***Responsibility****:* Acts as the central controller for navigation and launching other modules based on user voice input.

##### Attributes:

* + - battery\_status, audio\_dialpad, view\_contacts, Notes, settings, btn\_sms: Buttons to trigger various activities.
    - tts: TextToSpeech instance for audio feedback.

##### Methods:

* + - onCreate(): Initializes the activity and components.
    - dispatchKeyEvent(): Handles physical key events.
    - onActivityResult(): Receives and processes results from other activities.

#### Create\_note

***Responsibility****:* Handles note creation and storage.

##### Attributes:

* + - * btnAdd, btnView: Buttons for adding/viewing notes.
      * textView: Displays input/output.
      * textToSpeech: Audio output of text.
      * myDB: Instance of DatabaseHelper.

##### Methods:

* + - * onCreate() – Sets up the UI for adding and viewing notes.
      * dispatchKeyEvent() – Manages key input events for this activity.
      * onActivityResult() – Handles responses from child activities.

#### View\_notes

***Responsibility****:* Displays saved notes and reads them aloud.

##### Attributes:

* + - * db: Instance of DatabaseHelper.
      * userlist: ListView of notes.
      * tts: For speaking notes.

##### Methods:

* + - * onCreate() – Loads and displays saved notes with text-to-speech.
      * onActivityResult() – Processes results returned from other activities.

#### Databasehelper

***Responsibility****:* Handles SQLite database interactions.

##### Methods:

* + - * onCreate(): Creates tables.
      * onUpgrade(): Handles DB upgrades.
      * addData(): Adds a note to the database.
      * getListContents(): Fetches all saved notes.

#### Battery

***Responsibility***: Displays battery status via UI and TTS.

##### Methods:

* + - * onCreate() – Sets up the battery monitoring UI.
      * onPause() – Pauses the activity and unregisters the receiver.
      * onPostResume() – Resumes the activity and registers the battery receiver.

#### Battery\_status

***Responsibility****:* Reads and announces real-time battery percentage.

##### Attributes:

* + - * status\_label, percentage\_val: UI elements.
      * tts: Text-to-speech output.

##### Method:

* + - * onReceive(): Triggered when battery state changes.

#### Settings

***Responsibility****:* Handles Wi-Fi settings and voice command processing.

##### Attributes:

* + - * wifiManager: Controls Wi-Fi.
      * textToSpeech: Provides feedback.

##### Methods:

* + - * onCreate() – Initializes settings and TTS components.
      * dispatchKeyEvent() – Handles key events for voice settings navigation.
      * handleVoiceCommand() – Executes actions based on voice input.

#### Audio\_dialpad

***Responsibility****:* Acts as a voice-based phone dialer.

##### Attributes:

* + - * number: Input field for phone number.
      * call, one...zero, hash, ash, clear, back: Buttons for dialpad.
      * toSpeech: TextToSpeech object.

##### Methods:

* + - * onCreate() – Initializes the audio dialpad UI and TTS.
      * makePhoneCall() – Initiates a phone call to the entered number.
      * speak(String message) – Speaks out the provided message via TTS.
      * dispatchKeyEvent(KeyEvent event) – Handles key events to input digits or actions.
      * onRequestPermissionsResult() – Processes the result of phone permission requests.

#### Contacts

***Responsibility****:* Displays contacts (used as interface for dialer).

##### Methods:

* + - * onCreate() – Loads and displays the user’s contacts.

### MODULE DIAGRAM

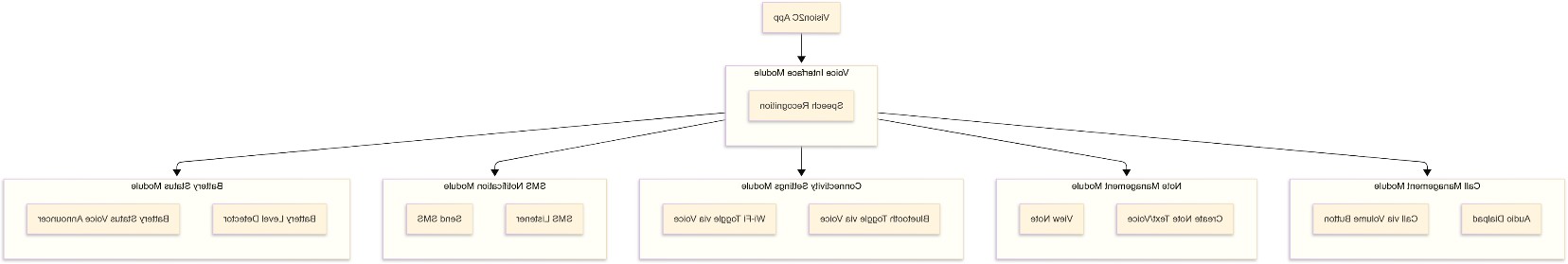
****

Figure 3.3 (Module diagram)

#### Voice Interface Module

* ***Speech Recognition****:* Core component for converting voice input into actions across all modules.

#### Call Management Module

* ***Audio Dialpad****:* Allows users to dial numbers through a spoken or audio interface.
* ***Call via Volume Button****:* Enables making calls by long-pressing physical volume buttons, acting as a shortcut.

#### Note Management Module

* ***Create Note Text/Voice***: Users can create notes via text input or voice commands.
* ***View Note****:* Retrieves and reads saved notes aloud using text-to-speech.

#### Connectivity Settings Module

* ***Bluetooth Toggle via Voice****:* Enables/disables Bluetooth through voice commands.
* ***Wi-Fi Toggle via Voice****:* Enables/disables Wi-Fi via spoken instructions.

#### SMS Notification Module

* ***SMS Listener****:* Listens for incoming messages and reads them out loud.
* ***Send SMS****:* Allows users to compose and send text messages using voice input.

#### Battery Status Module

* ***Battery Level Detector****:* Continuously monitors battery levels.
* ***Battery Status Voice Announcer****:* Announces the battery percentage and charging state vocally.

### BEHAVIOURAL DESIGN

**ACTIVITY DIAGRAM**

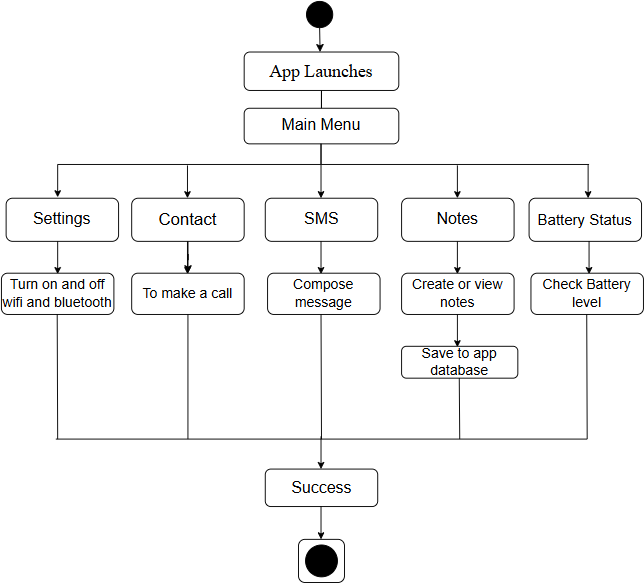
****

Figure 3.4 (Activity diagram)

Once the app is launched, the user is presented with a main menu containing five key modules:

#### Settings

WiFi and Bluetooth can be turned on or off through voice commands.

#### Contacts

Enables users to dial numbers using voice input.

### SMS

Allows users to compose and send messages.Users record the message using voice, and pressing the volume up button sends the message to a predefined contact.

#### Notes

Users can create or view notes, which are saved to the app’s internal database.

#### Battery Status

Displays the current battery level of the device.All module operations lead to a common end state labeled as **“Success”**, indicating that the user action was completed successful

### SEQUENCE DIAGRAM

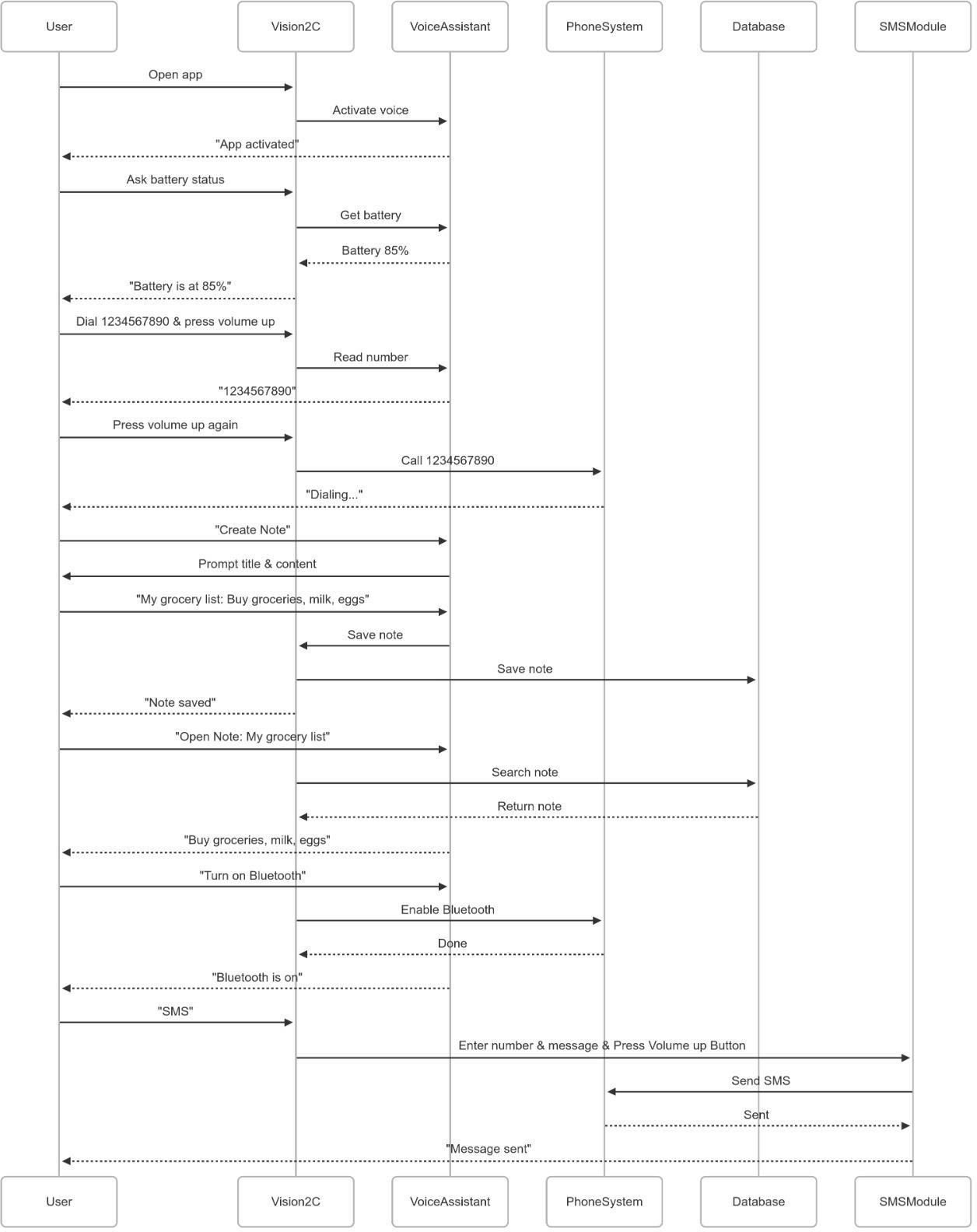
****

Figure 3.5 (Sequence diagram)

#### Voice Activation:

* On app launch, voice assistant is activated to allow hands-free control.

#### Battery Status Check:

* User can ask for current battery percentage.
* VoiceAssistant fetches and announces battery level (e.g., "Battery is at 85%").

#### Voice-Based Calling:

* User speaks a phone number and confirms with volume-up button.
* VoiceAssistant reads the number back for confirmation.
* Vision2C triggers a phone call through the PhoneSystem.

#### Note Creation:

* User says “Create Note”.
* VoiceAssistant prompts for note title and content.
* Vision2C saves the note to the App database.

#### Note Retrieval:

* User requests a saved note by title.
* Vision2C fetches and reads the note aloud.

#### Bluetooth Activation:

* User says “Turn on Bluetooth”.
* Vision2C sends a command to PhoneSystem to enable Bluetooth.
* Confirmation message: “Bluetooth is on”.

#### SMS Sending via Voice:

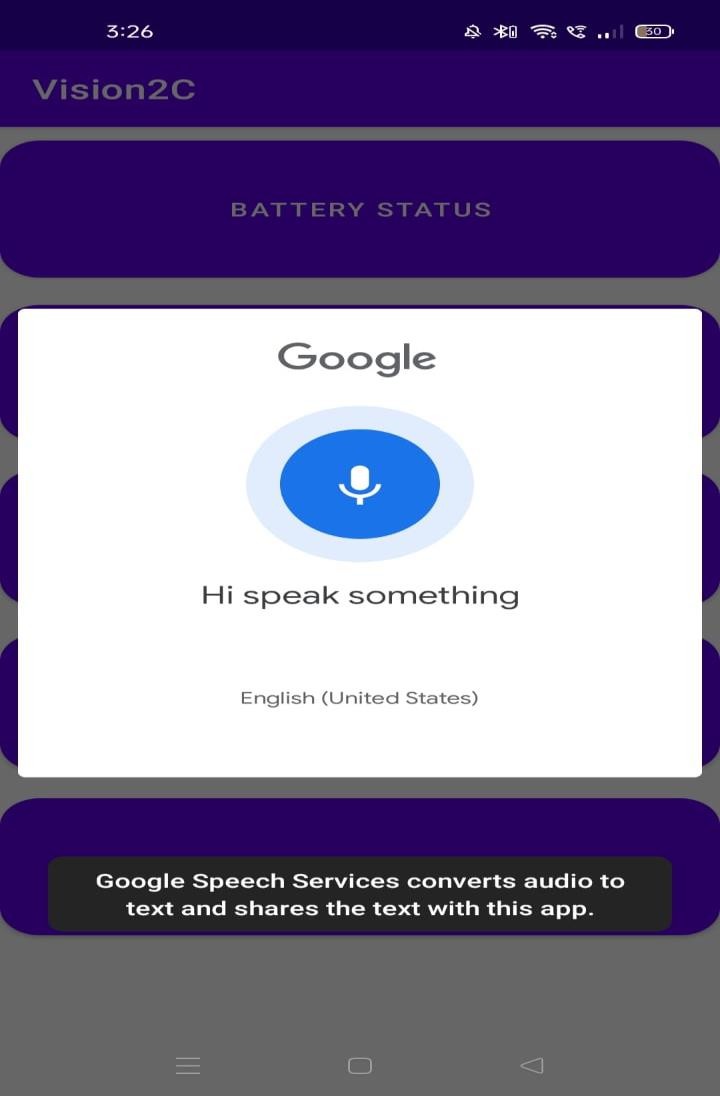
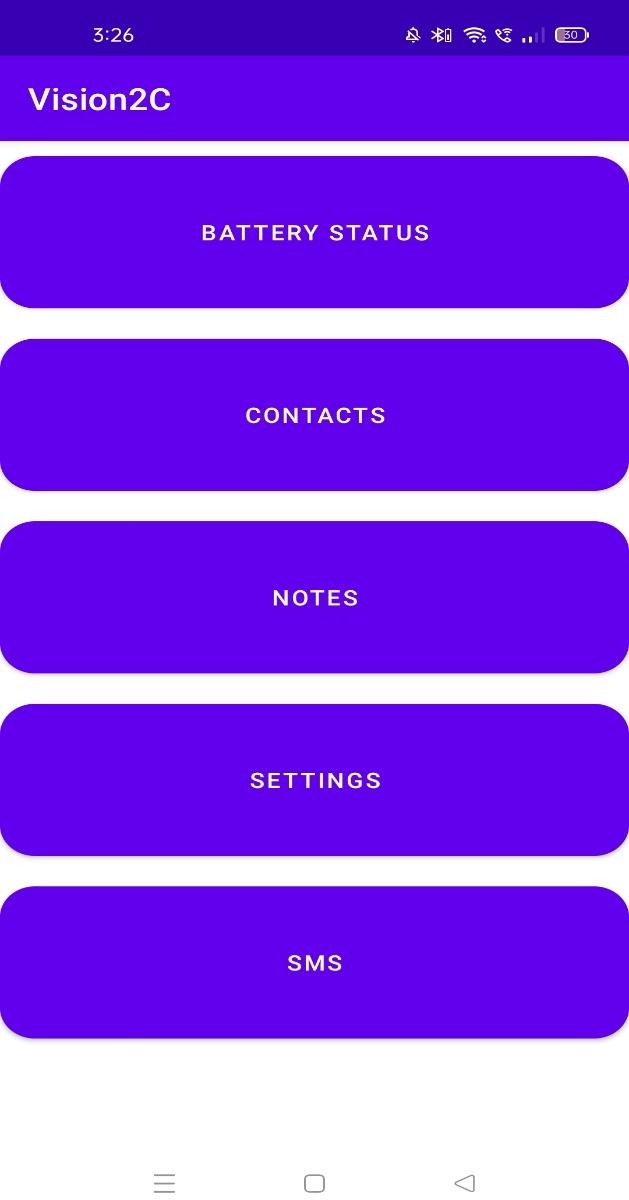
* User says “SMS”, provides the phone number and message.
* Confirms by pressing volume-up.
* SMS is sent via the SMSModule.
* Confirmation message: “Message sent”.

### TABLE DESIGN

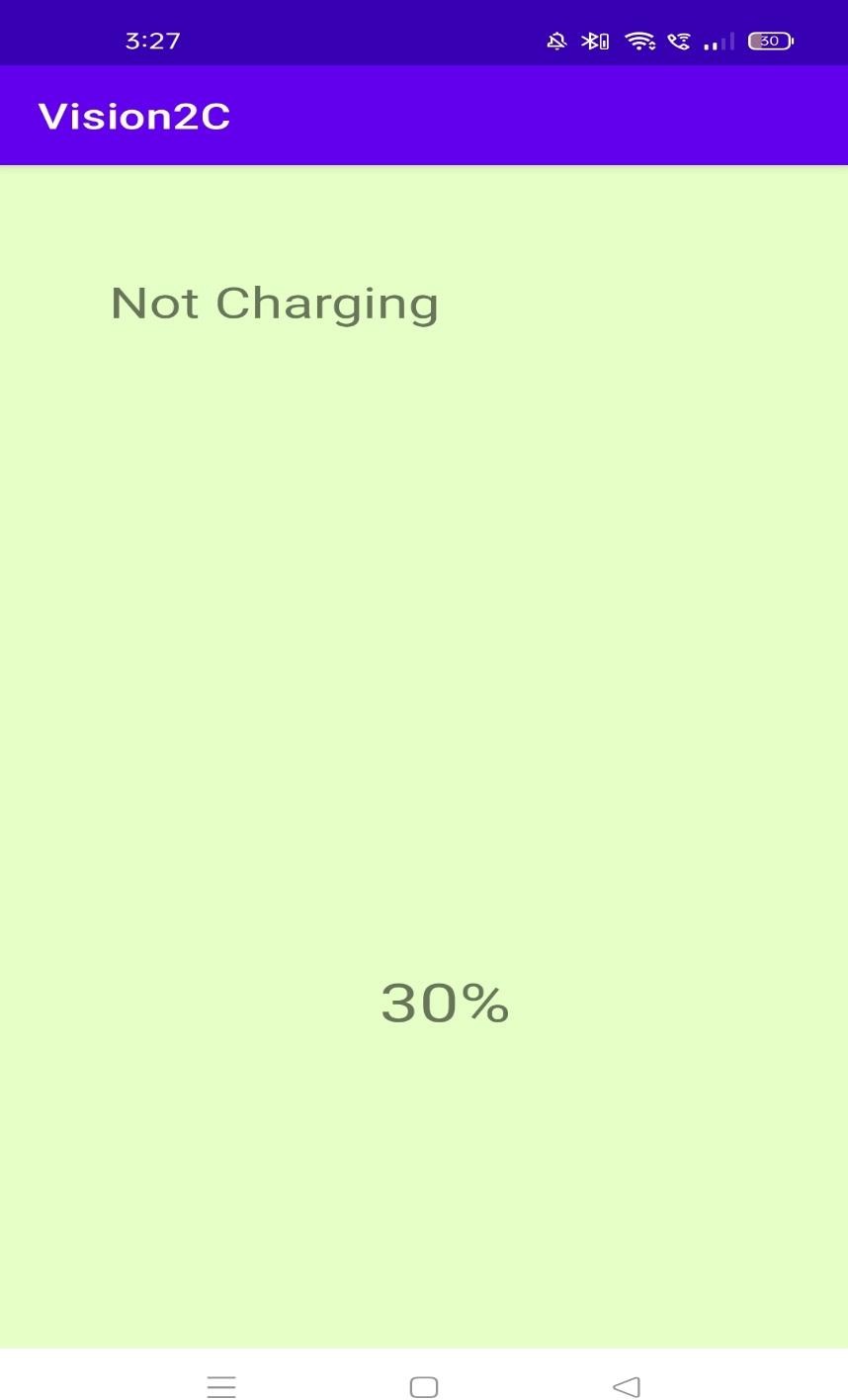
Table 3.1 (Table design)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.NO** | **COLUMN NAME** | **DESCRIPTION** | **DATA TYPE** | **CONSTRAINT** |
| 1 | ID | Unique identifier for each row | Integer | Primary key Auto increment |
| 2 | Title | Title for each text items | Text | Not Null Unique |
| 3 | Item 1 | Stores message/note/text items | Text | Not Null |

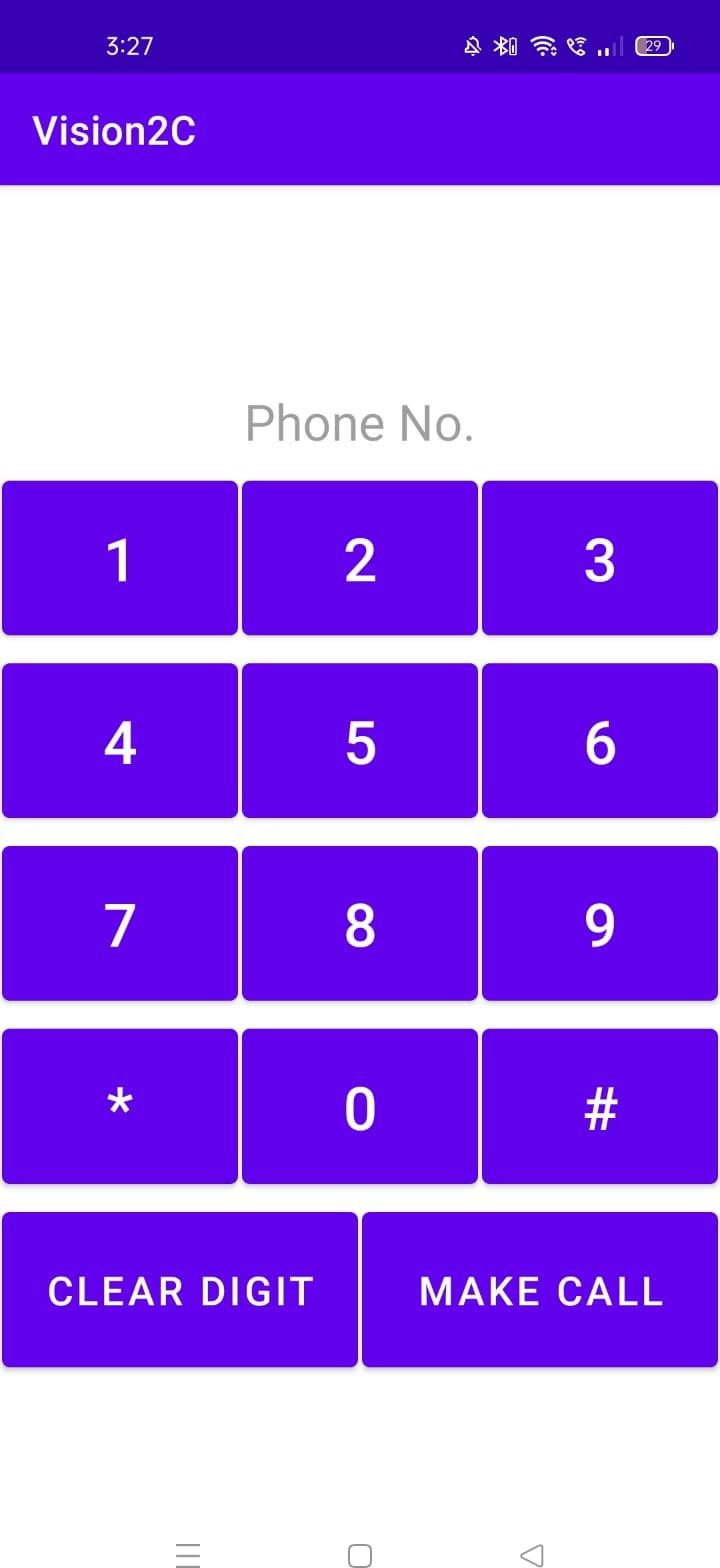
### USER INTEFACE DESIGN

****

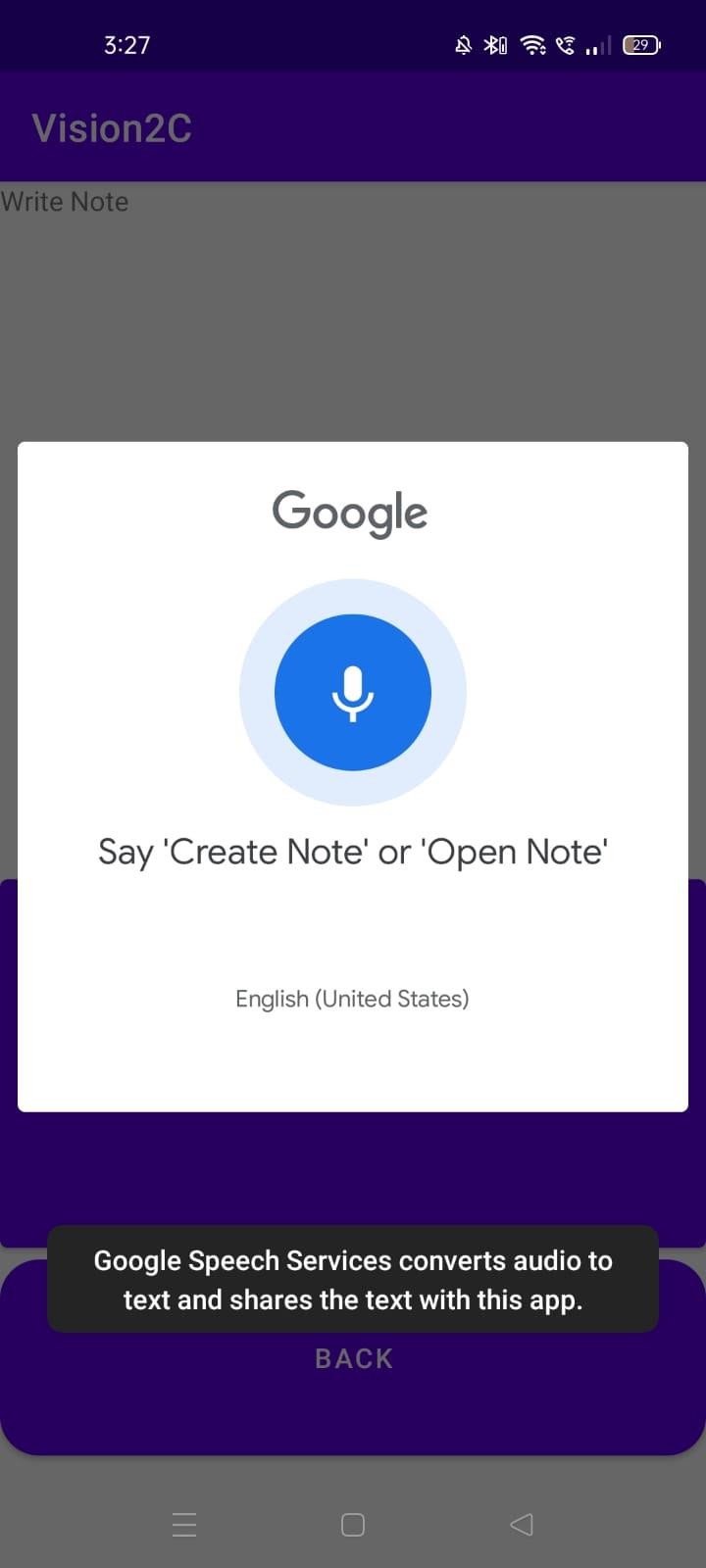
Screen 3.1 (Shows the main menu, where the user can click the volume down button to start using the application through voice commands.)



Screen 3.2 (Shows the battery and charging status, which are conveyed to the user through voice commands.)



Screen 3.3 (Displays the audio dialpad, where the user can enter a number and click the volume up button to make a call.)



Screen 3.4 (Shows the notes section where the user give Create Note or View Note Voice command.)



Screen 3.5 (Shows the notes section where the user give Create Note or View Note Voice

Commands)

### DEPLOYMENT DESIGN

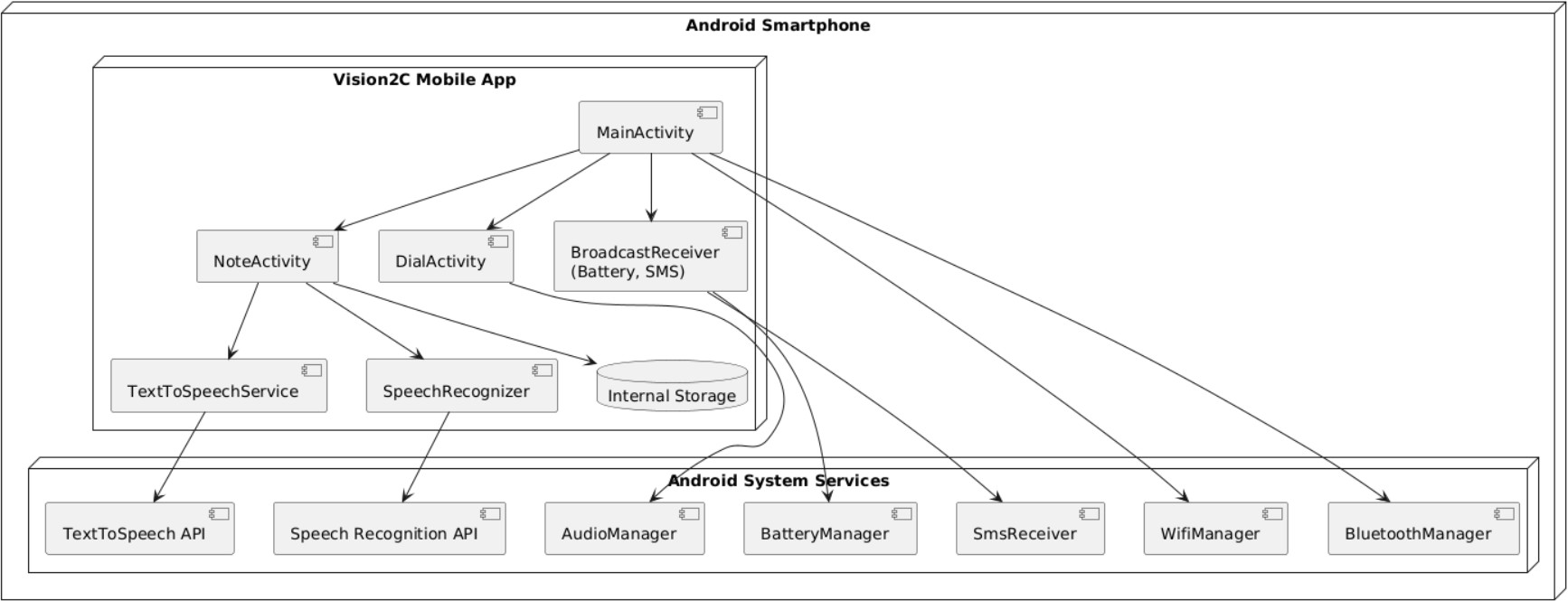
****

Figure 3.6 (Deployment design)

#### Android Smartphone

* + - Platform where the Vision2C app is installed and executed.
    - Contains the Vision2C Mobile App and Android System Services.
    - Represents the execution environment for all components.

#### MainActivity

* + - Acts as the entry point of the app.
    - Navigates to NoteActivity, DialActivity, SettingsActivity and BroadcastReceiver.
    - Coordinates user interaction and UI flow.

#### NoteActivity

Core feature to record and manage notes using voice.

##### Communicates with:

* + - * TextToSpeechService to read notes aloud.
      * SpeechRecognizer to convert spoken input to text.
      * Internal Storage to save note data.

#### DialActivity

* + - * Enables users to initiate calls using voice.

#### BroadcastReceiver (Battery, SMS)

##### Listens to:

* + - * Battery status updates via BatteryManager.
      * Sends SMS via SmsReceiver.

#### TextToSpeechService

* + - * Converts text output to spoken speech.
      * Interfaces with the TextToSpeech API.

#### Internal Storage

* + - * Saves potentially audio data.
      * Accessed by NoteActivity for persistence.

### NAVIGATION DESIGN

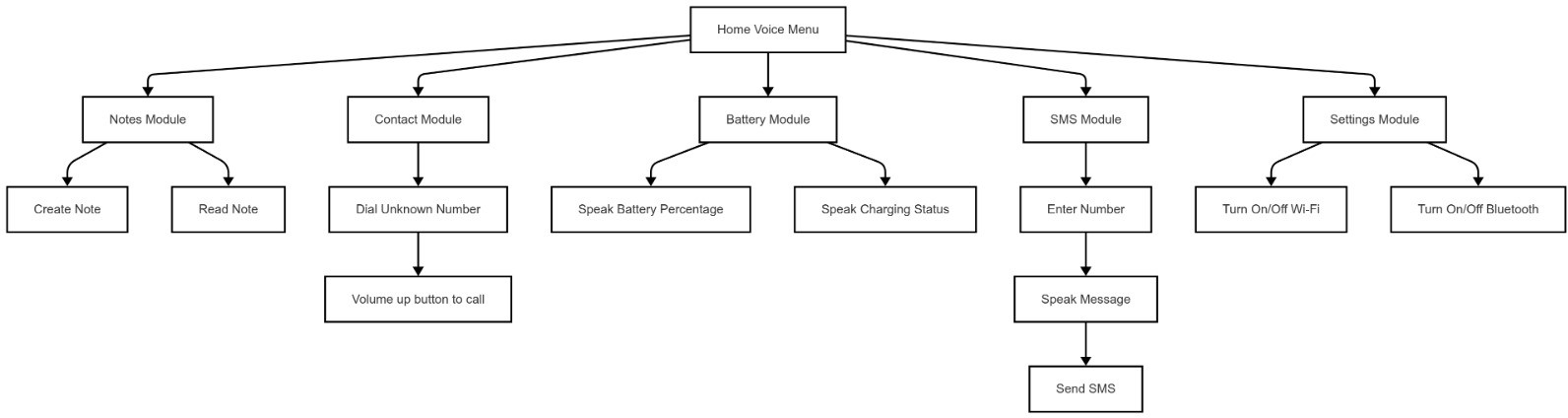
****

Figure 3.11 (Navigation design)

#### Notes Module

Create Note: Allows the user to create a note using voice input.

Read Note: Reads out the stored notes so users can recall important information.

#### Contact Module

Dial Unknown Number: Lets users dial a phone number through audio dialpad

Volume Up Button to Call: Uses the volume-up button as a physical shortcut to place a call.

#### Battery Module

Speak Battery Percentage: Announces the current battery level to the user.

Speak Charging Status: Informs the user whether the phone is currently charging.

#### Settings Module

Turn On/Off Wi-Fi: Voice command to toggle Wi-Fi connectivity. Turn On/Off Bluetooth: Voice command to toggle Bluetooth.

#### SMS Module

Enter Number: The user enters the recipient's phone number using the Audio Dialpad, which gives voice feedback for each digit.

Speak Message: The user dictates the message content through voice input.

Send SMS: Once the message is spoken, the user confirms sending by pressing the Volume Up button.

### CODE DESIGN

package com.example.vision2c;

import android.content.ActivityNotFoundException; import android.content.Intent;

import android.os.Build; import android.os.Bundle;

import android.speech.RecognizerIntent; import android.speech.tts.TextToSpeech; import android.view.KeyEvent;

import android.view.View; import android.widget.Button; import android.widget.TextView; import android.widget.Toast;

import androidx.annotation.Nullable;

import androidx.appcompat.app.AppCompatActivity; import java.util.ArrayList;

import java.util.Locale;

public class Main6Activity extends AppCompatActivity { DatabaseHelper myDB;

Button btnAdd, btnView, back; TextView textView; TextToSpeech toSpeech; boolean isCreatingNote = false;

String intro = "Notes page. Volume down button to give voice command. Say 'Create Note' or 'Open Note'.";

@Override

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.main6activity);

textView = findViewById(R.id.textView); btnAdd = findViewById(R.id.btnAdd); btnView = findViewById(R.id.btnView); back = findViewById(R.id.btn\_back); myDB = new DatabaseHelper(this);

toSpeech = new TextToSpeech(getApplicationContext(), status -> { if (status != TextToSpeech.ERROR) {

toSpeech.setLanguage(Locale.ENGLISH); toSpeech.speak(intro, TextToSpeech.QUEUE\_ADD, null);

}

});

btnAdd.setOnClickListener(v ->

toSpeech.speak("Long press to add note", TextToSpeech.QUEUE\_ADD, null)); btnAdd.setOnLongClickListener(v -> {

String newEntry = textView.getText().toString(); if (!newEntry.isEmpty()) {

AddData(newEntry);

toSpeech.speak("Note added", TextToSpeech.QUEUE\_ADD, null); textView.setText("");

} else {

Toast.makeText(Main6Activity.this, "You must put something in the text field!", Toast.LENGTH\_LONG).show();

}

return true;

});

btnView.setOnClickListener(v ->

toSpeech.speak("View saved notes", TextToSpeech.QUEUE\_ADD, null)); btnView.setOnLongClickListener(v -> {

startActivity(new Intent(Main6Activity.this, View\_notes.class)); return true;

});

back.setOnClickListener(v ->

toSpeech.speak("Back", TextToSpeech.QUEUE\_ADD, null));

back.setOnLongClickListener(v -> { speak("Back");

finish(); onBackPressed(); return true;

});

}

public void AddData(String newEntry) {

boolean insertData = myDB.addData(newEntry); if (insertData) {

Toast.makeText(this, "Data Successfully Inserted!", Toast.LENGTH\_LONG).show();

} else {

Toast.makeText(this, "Something went wrong :(.", Toast.LENGTH\_LONG).show();

}

}

private void speak(String message) {

if (Build.VERSION.SDK\_INT >= 21) {

toSpeech.speak(message, TextToSpeech.QUEUE\_FLUSH, null, null);

}

}

@Override

public boolean dispatchKeyEvent(KeyEvent event) { int action = event.getAction();

int keycode = event.getKeyCode(); switch (keycode) {

case KeyEvent.KEYCODE\_VOLUME\_UP: if (action == KeyEvent.ACTION\_UP) {

if (isCreatingNote) {

String note = textView.getText().toString(); if (!note.isEmpty()) {

AddData(note);

toSpeech.speak("Note saved", TextToSpeech.QUEUE\_FLUSH, null, null); textView.setText("");

isCreatingNote = false;

} else {

speak("Please speak your note first.");

}

}

return true;

}

break;

case KeyEvent.KEYCODE\_VOLUME\_DOWN: if (action == KeyEvent.ACTION\_DOWN) {

Intent intent = new Intent(RecognizerIntent.ACTION\_RECOGNIZE\_SPEECH);

intent.putExtra(RecognizerIntent.EXTRA\_LANGUAGE\_MODEL, RecognizerIntent.LANGUAGE\_MODEL\_FREE\_FORM);

intent.putExtra(RecognizerIntent.EXTRA\_LANGUAGE, Locale.getDefault());

intent.putExtra(RecognizerIntent.EXTRA\_PROMPT, "Say 'Create Note' or 'Open

Note'");

try {

startActivityForResult(intent, 1);

} catch (ActivityNotFoundException e) {

Toast.makeText(getApplicationContext(),e.getMessage(), Toast.LENGTH\_SHORT).show();

}

return true;

}

break;

}

return super.dispatchKeyEvent(event);

}

@Override

protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) { super.onActivityResult(requestCode, resultCode, data);

if (requestCode == 1 && resultCode == RESULT\_OK && data != null) {

ArrayList<String> result = data.getStringArrayListExtra(RecognizerIntent.EXTRA\_RESULTS);

if (result != null && !result.isEmpty()) {

String command = result.get(0).toLowerCase(); if (command.contains("create")) {

isCreatingNote = true;

toSpeech.speak("Speak your note. Press volume up to save.", TextToSpeech.QUEUE\_FLUSH, null, null);

} else if (command.contains("open")) {

toSpeech.speak("Opening notes. Say the note title or date.", TextToSpeech.QUEUE\_FLUSH, null, null);

startActivity(new Intent(Main6Activity.this, View\_notes.class));

} else {

toSpeech.speak("Unknown command. Please say 'Create Note' or 'Open Note'", TextToSpeech.QUEUE\_FLUSH, null, null);

}

}

}

}

}

#### View\_sms.java:

package com.example.vision2c; import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity; import androidx.core.app.ActivityCompat;

import androidx.core.content.ContextCompat;

import android.Manifest; import android.content.Context;

import android.content.pm.PackageManager; import android.database.Cursor;

import android.net.Uri; import android.os.Bundle; import android.widget.Toast;

public class View\_sms extends AppCompatActivity { Context context;

@Override

protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.activity\_view\_sms);

// Uri uriSms =Uri.parse("content://sms/inbox");

// Cursor cursor = getContentResolver().query(uriSms, null,null,null,null);

Cursor cursor = getContentResolver().query(Uri.parse("content://sms/inbox"), null, null, null, null);

if (cursor.moveToFirst()) { // must check the result to prevent exception do {

String msgData = "";

for(int idx=0;idx<cursor.getColumnCount();idx++)

{

msgData += " " + cursor.getColumnName(idx) + ":" + cursor.getString(idx);

Toast.makeText(this, msgData , Toast.LENGTH\_SHORT).show();

}

// use msgData

} while (cursor.moveToNext());

} else {

// empty box, no SMS

}

if (ContextCompat.checkSelfPermission(context, Manifest.permission.READ\_SMS) != PackageManager.PERMISSION\_GRANTED)

{

if(ActivityCompat.shouldShowRequestPermissionRationale(View\_sms.this, Manifest.permission.READ\_SMS))

{

ActivityCompat.requestPermissions(View\_sms.this,newString[]

{Manifest.permission.READ\_SMS}, 1);

}

else

{

ActivityCompat.requestPermissions(View\_sms.this,newString[]

{Manifest.permission.SEND\_SMS}, 1);

}

}

else

{

/\* do nothing \*/

/\* permission is granted \*/

}

}

@Override

public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {

super.onRequestPermissionsResult(requestCode, permissions, grantResults);

switch (requestCode)

{

case 1:

if (grantResults.length > 0 &&

grantResults[0] == PackageManager.PERMISSION\_GRANTED)

{

if (ContextCompat.checkSelfPermission(View\_sms.this, Manifest.permission.READ\_SMS) ==

PackageManager.PERMISSION\_GRANTED)

{

Toast.makeText(context, "Permission granted", Toast.LENGTH\_SHORT).show();

}

}

else

{

Toast.makeText(context, "No Permission granted", Toast.LENGTH\_SHORT).show();

}

break;

}

}

}

# CHAPTER IV

**SYSTEM TESTING**

This chapter gives an introduction to the various testing process involved in the project.The testing process focus on the logical internals of the program, ensuring that all the statements have been tested and also the functional externals that is conducting the test to the uncover errors and ensuring that the defined input will produce actual result that agree with the expected results.

* 1. **TEST CASES AND TEST REPORTS**

##### Performance Analysis On Various Testing

There are two general categories of testing. Pre implementation and post implementation.The software testing for the process planning system has been done during the pre-implementation stage using various software testing strategies.

##### Unit Testing

Each module of Vision2C was tested independently to confirm that its functionality meets the expected voice interactions. For example, the Notes module was tested to ensure that it correctly saves notes with a title and content through voice input, and retrieves them accurately using the title. The Battery module was tested to ensure it announces the correct battery level and charging status. Any errors in speech recognition or incorrect voice outputs were identified and debugged at this stage.

##### Interface Testing

After testing modules separately, the next step focused on verifying each voice interface’s usability and response. Since Vision2C is entirely voice-controlled, interfaces were checked for their ability to recognize and respond to user commands clearly and accurately. For instance, the Settings module was tested to make sure that voice commands like "Turn on Wi- Fi" or "Turn off Bluetooth" were executed as expected with appropriate voice confirmation.

##### Black Box Testing

Without looking into the internal logic, the Vision2C application was tested from the user’s perspective to validate if the output matched the intended functionality. Commands such as "Create a note", "Send SMS", or "What is my battery level?" were issued, and the response was evaluated for correctness. Special attention was given to detecting errors like unrecognized commands, incorrect responses, or failure to complete a task, and making sure

the app responds with voice-based error messages like "Unable to process request" or "Please repeat your command".

##### Integrated Testing

All modules were integrated, and the complete application was tested to ensure that every feature works in unison. The workflow between modules was tested—such as creating a note and then accessing it later or switching between the battery module and settings. The focus was on checking for consistency, conflict-free navigation, and the correct sequencing of voice interactions. Voice feedback was tested for clarity and timing, ensuring the user always knows what action is being taken.

Through these testing processes, Vision2C was validated to provide a seamless, voice-based interaction model suited for visually impaired users. The testing ensured that the system is robust, accessible, and delivers the intended functionality effectively.

Table 4.1(Test report)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Objective | Description | Input | Expected output | Actual output | Result |
| TC1 | Test note saving without title | Verify that the app shows an error if the note title is empty | Title: *(empty)*, Content: "Take medicine" | Voice output: Note title cannot be empty | Note title cannot be empty | Passed |
| TC2 | Test battery status feedback | Check that the app speaks battery percentage and charging status. | Charger plugged in, Battery at 65% | Voice output: "Battery is 65% and charging" | App announced: "Battery is 65% and charging" | Passed |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TC3 | Test toggling of Wi-Fi and Bluetooth | To Verify the settings module enables/disables Wi-Fi and Bluetooth | Tap "Turn off Wi-Fi", Tap "Turn on Bluetooth" | Wi-Fi is off; Bluetooth is on | Wi-Fi status: OFF;  Bluetooth status: ON | Passed |
| TC4 | Test SMS  sending | Tests behavior when a valid number is given without message content. | No message content provided | Message content cannot be empty.  Please say your message. | Message content cannot be empty. Please say your message. | Passed |
| TC5 | Test Contact | Invalid Number Entry | User enter number less than 10 digits | Error: Enter the correct number | Enter the correct number | Passed |

# CHAPTER V

**SYSTEM IMPLEMENTATION**

### SOFTWARE INSTALLATION AND PROJECT SETUP

#### Install Android Studio

* + - * Android Studio comes bundled with the required JDK (Java Development Kit), so no separate installation is needed.

#### Gradle and Plugin Setup

* + - * Uses Android Gradle Plugin version 7.0.3**.**
      * Integrated with Google Services Plugin version 4.3.10 for Firebase or related services.

#### SDK Configuration

* + - * compileSdkVersion: 31 (Android 12 APIs)
      * minSdkVersion: 22 (Android 5.1 and above)
      * Ensures compatibility across a wide range of Android devices.

#### Application Metadata

* + - * applicationId: com.example.vision2c
      * versionCode: 1
      * versionName: "1.0"

#### Java Compatibility

* + - * Configured to support Java 1.8**.**
      * Enables usage of modern features like lambda expressions and streams**.**

#### Build and Run the Project

* + - * Open Android Studio.
      * Click the "Run" button (green triangle) to build and launch the application on a connected device or emulator.

# CHAPTER VI CONCLUSION

In conclusion, the *Vision2C* application is a significant step toward empowering visually challenged individuals by providing them with a voice-assisted mobile interface that enhances their independence and usability of smartphones. With features like audio-based battery and charging status updates, a voice-controlled dialpad, note creation and playback, and voice commands for essential settings, the app ensures an intuitive and inclusive user experience. By integrating accessibility, voice recognition, and essential mobile functionalities, *Vision2C* bridges the gap between technology and accessibility, fostering equal digital opportunities for all users.

# BIBOLIOGRAPHY

1. H. Alhussaini, S. Ludi and J. Leone, An evaluation of Access Braille: A tablet-based braille keyboard for individuals with visual impairments, HCI International 2015, vol.529, no.2, pp.369- 374, 2015.
2. Alnfiai and S. Sampalli, Braille Enter: A touch screen braille text entry methodfor the blind, Procedia Computer Science 109C, pp.257-264, 2017.
3. A. S. Brian, J. Haegele and L. Bostick, Perceived motor competence of childrenwith visual impairments: A preliminary investigation, British Journal of Visual Impairment, pp.151-155, 2016.
4. M. V. Gaikwad and R. M. Khaire, Hardware based braille note taker, International Journal of Science, Engineering and Technology Research (IJSETR),vol.4, no.2, pp.3957-3959, 2015.