Software Implementation Documentation

# Table of Contents

1. Introduction

2. System Overview

3. Functional Requirements

4. Non-functional Requirements

5. System Architecture

6. Detailed Design

7. Detailed Process

8. External Interfaces

9. Error Handling

10. Security Considerations

11. Conclusion

# Introduction

This document provides a comprehensive and detailed overview of the software implementation for the AI-Save-Life application. It is meticulously designed to cover various critical aspects of the software development lifecycle, including system architecture, design, data storage, external interfaces, error handling, and security considerations.

The primary purpose of this documentation is to ensure that the software development process is not only transparent but also reproducible and aligned with the overarching project goals. Transparency in the development process means that every step, decision, and component of the system is clearly documented and accessible to all stakeholders. This transparency helps in building trust and ensures that all parties involved have a clear understanding of how the application works, what it aims to achieve, and how it intends to achieve those goals.

Reproducibility is another key objective of this documentation. By providing detailed descriptions and specifications of each component and process, other developers can reproduce the software implementation accurately. This is crucial for maintaining consistency, especially when updates, modifications, or scaling of the system are required. It also aids in debugging and troubleshooting, as developers can follow the documented procedures to identify and resolve issues.

Aligning the development process with the project goals ensures that every effort and resource invested in the software contributes directly to the intended outcomes of the AI-Save-Life application. This alignment is achieved by clearly defining the functional and non-functional requirements, setting clear objectives for each component, and continuously verifying that the implementation meets these objectives.

In summary, this document serves as a foundational blueprint for the development of the AI-Save-Life application. It ensures that the development process is transparent, reproducible, and aligned with the project's objectives, thereby facilitating the creation of a robust, reliable, and effective solution for saving lives.

# 2. System Overview

The AI-Save-Life application is meticulously designed to detect accidents through a comprehensive system that employs various threads to monitor multiple data streams. These threads are responsible for capturing and analyzing images, detecting vehicle number plates, updating location data, monitoring speed, and detecting gyroscopic effects that indicate sudden movements or impacts. This multi-threaded approach allows the application to gather and process a wide range of data simultaneously, enhancing the accuracy and reliability of accident detection.

One of the key features of the AI-Save-Life application is the ability for users to manually start and stop the monitoring process. This provides users with the flexibility to control when the application is active, ensuring it can be used in a way that best suits their needs. Data collected during the monitoring process is stored both locally on the user's device and in the cloud, where a Cassandra database is employed to manage and organize the data efficiently. The use of cloud storage not only provides scalability but also ensures that data is backed up and accessible from anywhere, adding an extra layer of reliability and security.

The system is structured to handle multiple tasks concurrently, leveraging the power of multi-threading to ensure efficient processing and timely response to detected incidents. This concurrent processing capability is crucial for the application to function effectively in real-time scenarios. In the event of an accident detection, the system is designed to send immediate notifications to a predefined list of friends and family members, ensuring that help can be dispatched quickly. These notifications include critical information such as the location and nature of the incident, helping responders to act swiftly and accurately.

# 3. Functional Requirements

The main functional requirements of the application include:

**Android Permissions**: The application must request and manage necessary Android permissions to function correctly. This includes permissions for accessing the device's camera, location services, storage, and other relevant features. The system should handle permission requests gracefully, informing users why these permissions are needed and guiding them through the process of granting them.

**User Authentication:** The application must facilitate user login and registration processes. This includes the ability to sign up new users and log in existing ones, with support for One-Time Password (OTP) email verification to ensure secure and verified access.

**Image Capturing:** The application must be capable of capturing images at regular intervals of every 10 seconds. This functionality is essential for ongoing monitoring, providing a continuous stream of visual data for further analysis.

**Number Plate Detection:** The system should be equipped with the capability to detect and recognize vehicle number plates from the images captured. This involves advanced image processing and recognition algorithms to accurately identify and extract number plate information.

**Location Updates:** The application must continuously update the user's location and monitor for any significant falls or movements. This real-time tracking is crucial for assessing user activity and ensuring safety.

**Video Storage:** The system must support local storage of video data. This feature allows the application to keep a record of video footage for future reference, analysis, or evidence.

**Accident Detection:** The application should be able to detect accidents based on the processed data. This involves analyzing various inputs, such as speed, angular velocity, and movement patterns, to determine if an accident has occurred.

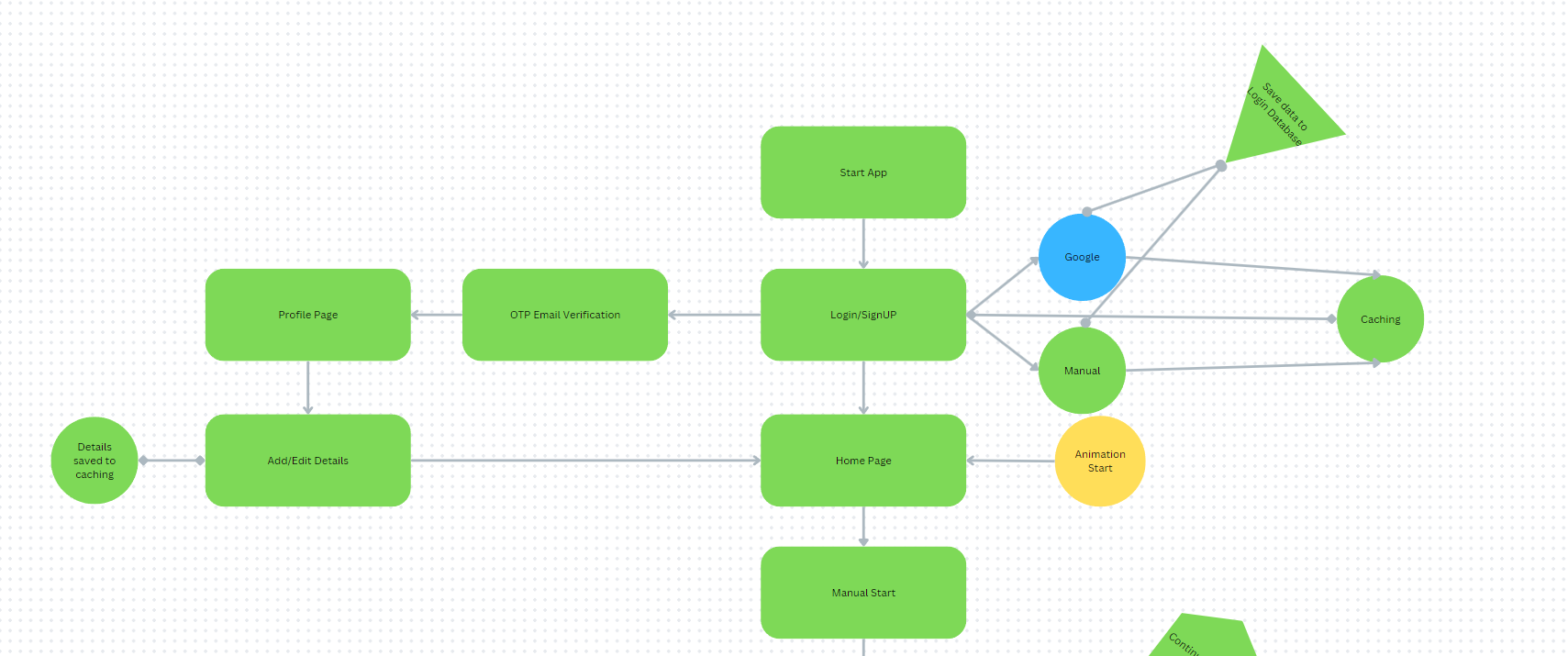
**Notification:** Upon detecting an accident, the system must send SMS notifications to a predefined list of emergency contacts. Additionally, it should upload relevant data to the cloud for further processing and storage, ensuring that all critical information is securely backed up and accessible.

# 4. Non-functional Requirements

Non-functional requirements describe the system's operational capabilities and constraints. These include:

* **Performance Efficiency:** The application should process data and respond to incidents in real-time with minimal latency.
* **Reliability:** The system must be reliable, with high availability and minimal downtime.
* **Scalability:** The application should handle an increasing number of users and data without performance degradation.
* **Security:** The system must ensure data confidentiality, integrity, and availability through robust security measures.
* **Usability:** The application should be user-friendly, with an intuitive interface and easy navigation.

# 5. System Architecture



A diagram of a company

Description automatically generated

The system architecture is composed of multiple interconnected components designed to perform specific tasks efficiently. Key components include:

* **Start/Stop App:** Initiates/Ends the application and navigates the user to the authentication process.
* **Login/Sign Up:** Manages user authentication, including OTP email verification.
* **Home Page:** Serves as the main interface for users to start the monitoring process and access other functionalities.
* **Threads:** The application employs multiple threads to handle tasks such as image capturing, number plate detection, location updates, and video storage concurrently.

# 6. Detailed Design and Implementation

**Thread Implementations**

The application employs multiple threads to handle specific tasks concurrently, enhancing performance and efficiency. These include:

* **Thread 1:** Handles number plate detection using image processing techniques. It scans the captured images for vehicle number plates and extracts relevant information.
* **Thread 2:** Manages location updates and fall detection. It continuously monitors the device's location and detects any significant falls or movements, which might indicate an accident.
* **Thread 3:** Takes care of video storage, ensuring that video data is stored locally and uploaded to the cloud as necessary.

**Number Plate Detection**

The process begins with an initial check to determine if the processing queue is empty or not. This step is crucial as it dictates the subsequent actions of the system. If the queue is found to be empty, the system proceeds to verify whether the main process has started. If the main process is active, the system then sends the data of each detected number plate, stored in an array, to the server. This data is subsequently stored in the Cassandra database, ensuring it is safely and systematically archived for future reference and analysis.

On the other hand, if the queue is not empty, the system retrieves (or pops out) an image from the cached queue for further processing. This image undergoes initial object detection using a TensorFlow Lite model, which helps in identifying potential areas within the image that might contain number plates. Following this, the same image is processed through a number plate detection model, which precisely detects the fragments of number plates, delineating them with bounding boxes.

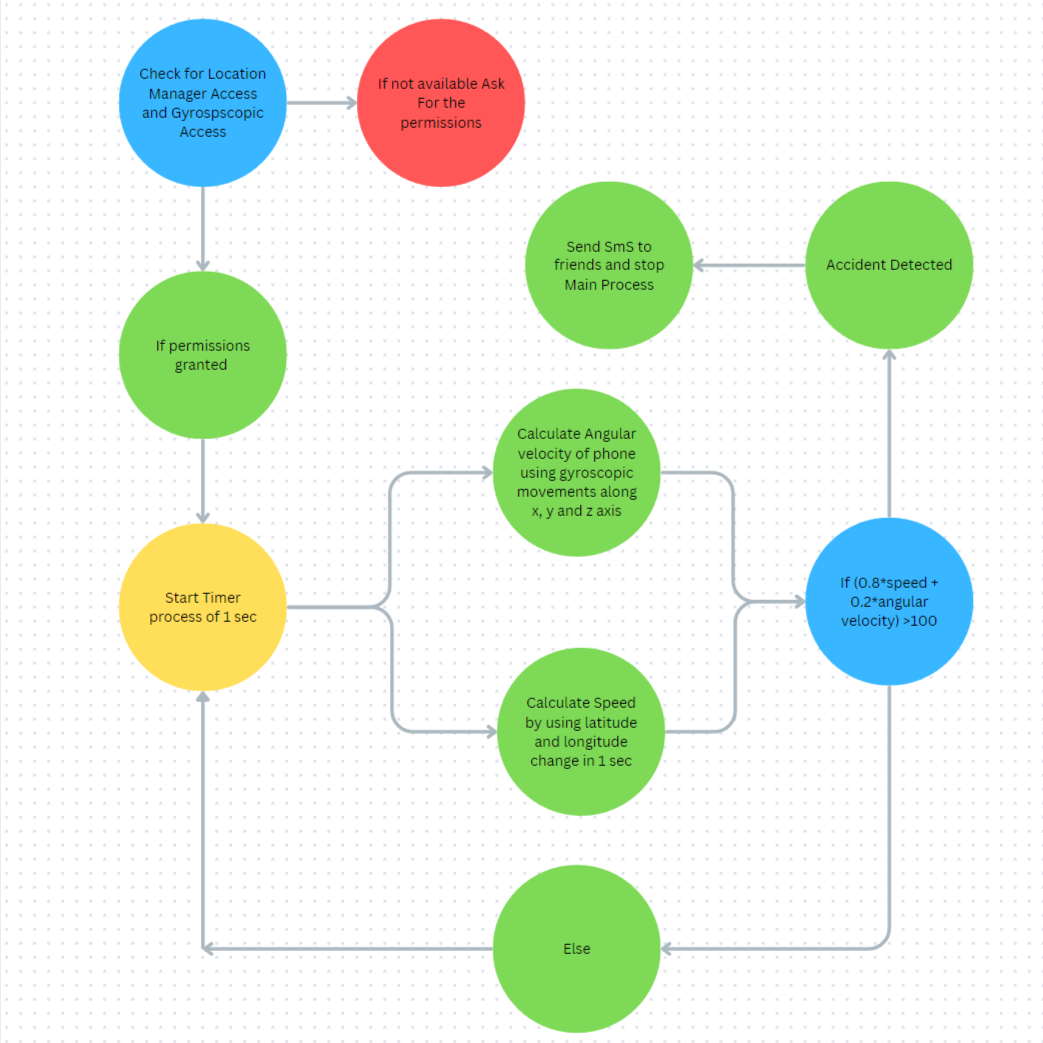
The detected number plates are then subjected to further analysis through the Tesseract OCR (Optical Character Recognition) model. This model extracts the textual information from the detected number plates, converting the visual data into readable text. Each piece of detected text is meticulously stored in an array, ensuring that all relevant information is captured and organized. This comprehensive process ensures that number plates are accurately identified, processed, and stored efficiently, facilitating robust data management and retrieval capabilities.

A diagram of a flowchart

Description automatically generated

**Location and Gyroscopic Function**

The process starts by ensuring the app has access to location services and gyroscopic sensors. If these permissions are not granted, the system prompts the user to enable them. Once access is confirmed, a timer initiates a process every second. The system then collects gyroscopic data to measure angular velocity along the x, y, and z axes and calculates the device's speed based on changes in latitude and longitude within each second. A combined value is calculated using the formula `0.8 \* Speed + 0.2 \* Angular Velocity`. If this value exceeds 100, it is interpreted as a potential accident. In such a case, the system sends an SMS to the user’s emergency contacts and emergency services (112) and halts the monitoring process. If the threshold is not met, the system continues monitoring by looping back to the initial checks.



**Number Plate Bounding ML Model**

A diagram of a diagram

Description automatically generated

In this workflow, an image containing a vehicle with an Indian license plate is first fed into a fine-tuned YOLOv8 model. This YOLOv5 model, trained specifically on a dataset of Indian license plate images, detects and localizes the license plate within the image. The model outputs coordinates in the format [ymin, xmin, ymax, xmax], which are used to crop the image to focus only on the license plate region. This cropped license plate image is then passed as input to a Keras-based OCR (Optical Character Recognition) model. The OCR model processes the cropped image and extracts the text, effectively recognizing and outputting the alphanumeric characters on the license plate. This output is the final text representation of the license plate number.

**Object Detection ML Model**

In this workflow, an image of road traffic is fed into the MobileNetSSDv2 model, which is trained on the COCO dataset with 70 classes. The model takes an input image of size 320x320 pixels. Upon receiving the image, the MobileNetSSDv2 model detects objects within the image and forms bounding boxes around them. Each detected object is assigned a label with a specific confidence score. To enhance the efficiency and reduce the computational time for real-time object detection, this model has been converted into the .tflite format for use in an application. The output of the model is a 2D matrix where each row contains six elements: the class ID, confidence score, and the coordinates of the bounding box. This enables effective and swift object detection in real-time scenarios.

A diagram of a flowchart

Description automatically generated

# 7. Data Storage

Data storage is a critical component of the application, playing a vital role in handling the storage of images, videos, and other relevant data generated or used by the system. To optimize performance and ensure quick access and processing, the system initially uses local storage for temporary data. This approach allows the application to operate efficiently, reducing latency and providing rapid data retrieval when needed.

However, for long-term storage and to accommodate larger volumes of data, the system uploads this data to the cloud. In the cloud, Cassandra DB is utilized, chosen for its scalability and reliability. Cassandra's distributed database structure ensures that data can be stored across multiple nodes, providing high availability and fault tolerance. This means that even in the event of hardware failures or other issues, the data remains accessible and secure.

# 8. External Interfaces

The system interfaces with various external services to perform essential functions. These include:

* **Email Servers:** Brevo Mail Service for OTP verification during the signup process.
* **Cloud Storage Services:** Cassandra for uploading and storing data securely and reliably.
* **SMS Services:** Phone SMS Service is utilized for sending notifications to a predefined list of contacts in case of an accident.

# 9. Error Handling

The application includes robust error handling mechanisms to manage and log errors encountered during execution. This includes:

* **Network Failures:** The system handles network disruptions gracefully by saving the data in Work Manager, retrying operations or providing user feedback.
* **Unexpected Exceptions:** General error handling to capture and log unexpected exceptions, preventing application crashes and allowing for troubleshooting and debugging.

# 10. Security Considerations

Security is a paramount concern for the application. Key security measures include:

* **Secure Authentication:** Implementing strong authentication mechanisms, including OTP verification, to prevent unauthorized access.

# 11. Conclusion

This document outlines the comprehensive implementation details for the application. By following the design and requirements specified, developers can ensure a robust, reliable, and scalable solution. The application is designed to meet both functional and non-functional requirements, providing users with a secure and efficient accident detection system.