TED UNIVERSITY

Detailed Design Report

VAVI
(Voice Assistan for Visually Impaired)

Berkay Kaan Karaca(68317070956)

Ceyda Kuşçuoğlu(16348076072)

Kıvanç Terzioğlu(27233564574)

1. Introduction

1.1 Purpose of the Document

The purpose of this Detailed Design Document is to describe the internal architecture, component interactions, and software design of the *Voice Assistant for Visually Impaired Users* project. It provides a comprehensive explanation of how each subsystem is implemented, the technologies used, and how the system meets its functional and non-functional requirements.
This document serves as a reference for developers, supervisors, and maintainers to understand the detailed mechanisms behind object detection, indoor navigation, and user interaction within the system.

1.2 Scope

The project focuses on developing a mobile-based voice assistant application that improves the mobility and safety of visually impaired individuals within indoor environments such as schools or office buildings.
The system integrates **real-time computer vision**, **indoor localization**, and **voice interaction** to guide users through their surroundings and help them avoid obstacles.

The two primary modules of the system are:

1. **Computer Vision Module:**
   The application employs a **TensorFlow Lite (TFLite)** model that performs on-device object detection. While the base model can detect humans, it has been further trained using a **custom dataset** collected from the school environment to recognize additional critical elements such as **stairs**, **elevators**, and **tactile walking surfaces (paths for visually impaired individuals)**.
   The model processes live camera input in real-time to identify these structures and generates audio alerts to assist the user during movement.
2. **Navigation and Localization Module:**
   The navigation system provides both **online** and **offline** modes to ensure continuous operation in environments where GPS signals may be weak or unavailable.
   - In **online mode**, the system uses the **Geolocator** Flutter package, which internally relies on **Google's Fused Location Provider** to combine data from GPS, Wi-Fi networks, cellular towers, and onboard sensors (accelerometer, gyroscope) for accurate location estimation.
   - In **offline mode**, when GPS or network signals are absent, the system performs **dead reckoning** using inertial sensor data to estimate movement and direction.

   Indoor maps of the school were obtained with authorization and converted into a **graph-based representation**. This structure allows efficient pathfinding between rooms, corridors, and key locations using algorithms such as **Dijkstra** or **A***.

To maintain high performance on mobile devices, all computationally heavy operations (e.g., image recognition, navigation calculations) are executed concurrently using **Dart Isolates**, ensuring that the user interface remains smooth and responsive even during continuous processing.

1.3 References

- IEEE Std 1016-2009, *IEEE Standard for Information Technology—Systems Design—Software Design Descriptions*
- TensorFlow Lite Documentation: https://www.tensorflow.org/lite
- Flutter Geolocator Package: https://pub.dev/packages/geolocator
- Flutter Official Documentation: https://docs.flutter.dev
- Ultralytics YOLOv8 Model Documentation: https://docs.ultralytics.com
- Android Accessibility Guidelines

---

2. System Overview

2.1 System Purpose and Goals

The main purpose of the system is to provide a **real-time, voice-assisted navigation and awareness tool** for visually impaired individuals in indoor environments.
The application combines **AI-based visual perception** and **sensor-assisted navigation** to guide users and alert them to environmental hazards.

Primary Goals:

- To detect and recognize humans, stairs, elevators, and tactile paths in real time.
- To provide accurate indoor navigation using a hybrid localization approach (GPS, Wi-Fi, sensors).
- To offer reliable offline navigation when network or GPS signals are unavailable.
- To deliver accessible and natural voice-based feedback and interaction.
- To maintain high performance and low latency through multi-threaded execution with Dart Isolates.

2.2 High-Level Architecture Recap

At a high level, the system consists of three major layers:

1. **Perception Layer (Computer Vision):**
   - Captures live camera input using the Flutter Camera package.
   - Processes each frame using the TFLite model trained to recognize humans, stairs, elevators, and tactile walking surfaces.
   - Operates asynchronously in a separate Dart Isolate to prevent frame drops and UI lag.
   - Sends detection results to the Interaction Layer for corresponding voice feedback.
2. **Localization and Navigation Layer:**
   - Obtains the device's geographic and indoor position through the **Geolocator** package.
   - Combines GPS, Wi-Fi, and cellular signals in online mode for high-accuracy location data.

    - Switches to **offline mode** when GPS is unavailable, using accelerometer and gyroscope data to estimate displacement (dead reckoning).
    - Uses a **graph-based indoor map**, created from official building plans, to determine optimal routes and directions.
3. **Interaction Layer:**
    - Provides a voice interface for communication between the user and the system.
    - Uses **Speech-to-Text (STT)** for command recognition and **Text-to-Speech (TTS)** for spoken guidance and alerts.
    - Delivers proximity-based alerts and navigation instructions in real-time.

2.3 Assumptions and Dependencies

- The system is designed for **Android** devices with active camera, microphone, and motion sensors (accelerometer, gyroscope).
- The **TFLite model** is pre-trained and deployed locally on the device.
- The **Geolocator** package provides location data through underlying Fused Location Provider mechanisms.
- Building maps and Wi-Fi fingerprints are preloaded and stored locally.
- The system assumes stable lighting conditions for reliable object recognition.
- Internet connectivity is only required for model updates or map synchronization; core functionality operates offline.

3. Detailed System Architecture

The proposed system follows a **modular and layered architecture** that separates concerns between perception, navigation, and user interaction. Each subsystem communicates through well-defined interfaces to ensure scalability, maintainability, and parallel development. The architecture was specifically designed to support **real-time computer vision processing**, **indoor localization**, and **responsive user feedback** without overloading the device's main thread.

3.1 Subsystem Decomposition

The system is composed of three primary subsystems:

1. **Image Processing Subsystem** – Responsible for detecting humans and environmental features (stairs, elevators, tactile paths) using a fine-tuned TensorFlow Lite model.
2. **Navigation Subsystem** – Handles route planning and localization using a graph-based map of the school building and sensor data from the device.
3. **User Interface Subsystem** – Provides auditory and visual feedback to the user, displaying navigation instructions and detection alerts in an accessible format.

These subsystems operate concurrently but independently, leveraging **Dart Isolates** to isolate computationally intensive tasks such as image recognition and pathfinding from the main UI thread.

3.2 Component Descriptions

- **Camera Handler:** Captures live video frames and sends them to the TFLite inference engine for processing.
- **TFLite Inference Engine:** Executes the deep learning model to identify objects and humans in the scene.
- **Map and Graph Manager:** Stores the spatial representation of the school layout, where each node represents a key location (e.g., classroom, elevator, stairway) and edges represent navigable paths.
- **Localization Module:** Utilizes the **Geolocator** Flutter package to obtain location data from multiple sources, including GPS, Wi-Fi, and cellular networks, providing a fused position estimate.
- **Sensor Fusion Unit:** Processes accelerometer, gyroscope, and magnetometer data to estimate movement when GPS signals are weak or unavailable (offline mode).
- **Pathfinding Engine:** Computes optimal routes using algorithms like Dijkstra or A*, depending on the graph's structure.
- **Feedback Manager:** Communicates navigation directions and detection results to the user via voice and vibration cues.

3.3 Layered Architecture

The system is organized into the following layers:

- **Presentation Layer:** Implements the Flutter-based user interface, responsible for accessibility and user interaction.
- **Application Logic Layer:** Coordinates communication between navigation, vision, and feedback modules.
- **Data and Processing Layer:** Handles the core computational tasks such as image recognition, localization, and pathfinding.
- **Hardware and Sensor Layer:** Interfaces with the device's sensors (camera, accelerometer, gyroscope) and external location sources.

3.4 Technology Stack Details

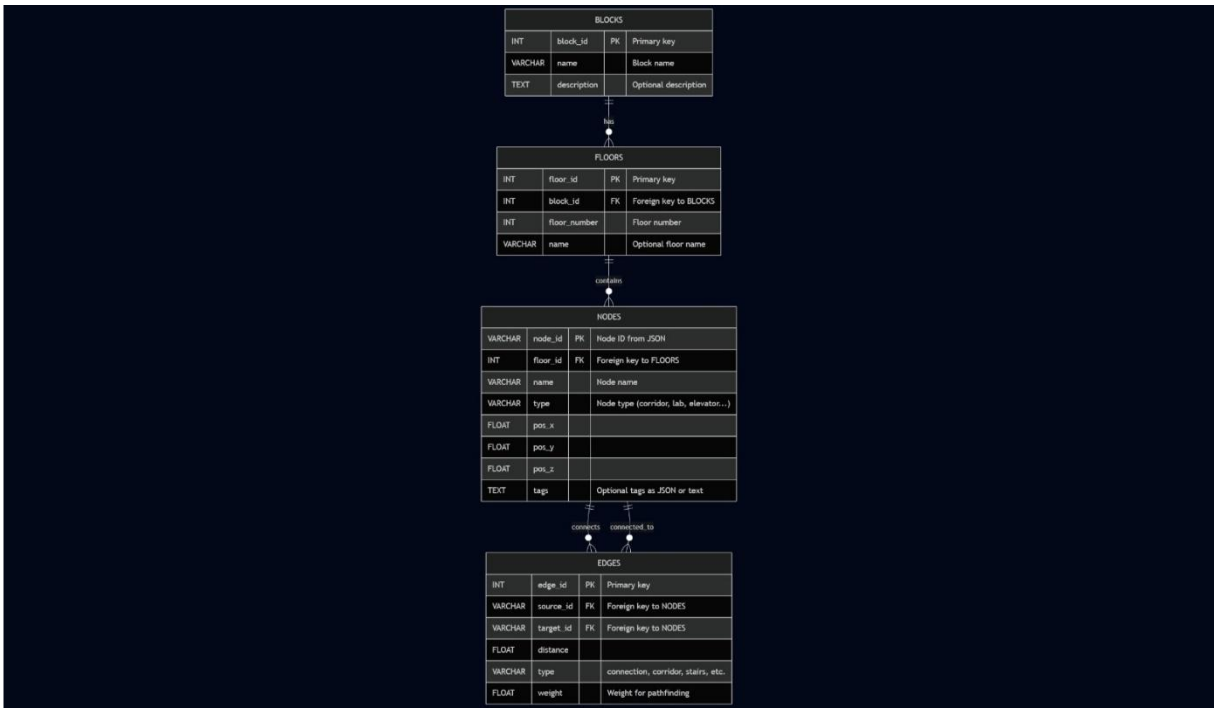
- **Frontend:** Flutter SDK (Dart)
- **AI/ML Framework:** TensorFlow Lite for on-device inference
- **Localization Library:** Geolocator (Flutter plugin for fused positioning)
- **Concurrency:** Dart Isolates for multithreading and performance optimization
- **Data Representation:** Graph-based map model derived from actual school floor plans
- **Platform:** Android OS (fully native execution without external servers)

4. Data Design

This section describes the design of the data structures and the flow of information within the indoor navigation system. The system manages building blocks, floors, nodes, and connections, while also incorporating real-time object detection using YOLO/TensorFlow. The design ensures that user location, environmental data, and detected objects are efficiently organized for navigation and guidance purposes.

4.1 Database Schema & Data Entities and Relationships

The system's database design revolves around four main entities: **BLOCKS, FLOORS, NODES, and EDGES**. These entities capture the structural layout of the building and the relationships necessary for pathfinding.



BLOCKS			
INT	block_id	PK	Primary key
VARCHAR	name		Block name
TEXT	description		Optional description

FLOORS			
INT	floor_id	PK	Primary key
INT	block_id	FK	Foreign key to BLOCKS
INT	floor_number		Floor number
VARCHAR	name		Optional floor name

NODES			
VARCHAR	node_id	PK	Node ID from JSON
INT	floor_id	FK	Foreign key to FLOORS
VARCHAR	name		Node name
VARCHAR	type		Node type (corridor, lab, elevator...)
FLOAT	pos_x		
FLOAT	pos_y		
FLOAT	pos_z		
TEXT	tags		Optional tags as JSON or text

EDGES			
INT	edge_id	PK	Primary key
VARCHAR	source_id	FK	Foreign key to NODES
VARCHAR	target_id	FK	Foreign key to NODES
FLOAT	distance		
VARCHAR	type		connection, corridor, stairs, etc.
FLOAT	weight		Weight for pathfinding

1. **BLOCKS**
   - **Description:** Represents individual building blocks.
   - **Attributes:**
     - `block_id`: Primary Key (PK), uniquely identifies each block.
     - `name`: Name of the block.
     - `description`: Optional textual description.
   - **Relationships:** Each block contains one or more **FLOORS** (1:N).
2. **FLOORS**
   - **Description:** Represents floors within a block.
   - **Attributes:**
     - `floor_id`: Primary Key (PK).
     - `block_id`: Foreign Key (FK) referencing the parent block.
     - `floor_number`: Numerical floor identifier.
     - `name`: Optional floor name.
   - **Relationships:** Each floor contains multiple **NODES** (1:N).
3. **NODES**
   - **Description:** Represents key points within a floor, such as corridors, labs, or elevators.
   - **Attributes:**
     - `node_id`: Primary Key (PK).
     - `floor_id`: Foreign Key (FK) referencing the corresponding floor.
     - `name`: Node name.
     - `type`: Node type (e.g., corridor, lab, elevator).
     - `pos_x`, `pos_y`, `pos_z`: 3D coordinates for accurate positioning.

        - `tags`: Optional metadata stored in JSON or text format.
    - **Relationships:** Each node is connected to one or more **EDGES**.
4. **EDGES**
    - **Description:** Represents connections between nodes, including distances and weights for navigation.
    - **Attributes:**
        - `edge_id`: Primary Key (PK).
        - `source_id`, `target_id`: Foreign Keys (FK) referencing connected nodes.
        - `distance`: Physical distance between nodes.
        - `type`: Connection type (e.g., corridor, stairs).
        - `weight`: Weight used by pathfinding algorithms.
    - **Relationships:** Connects two nodes in the building graph.

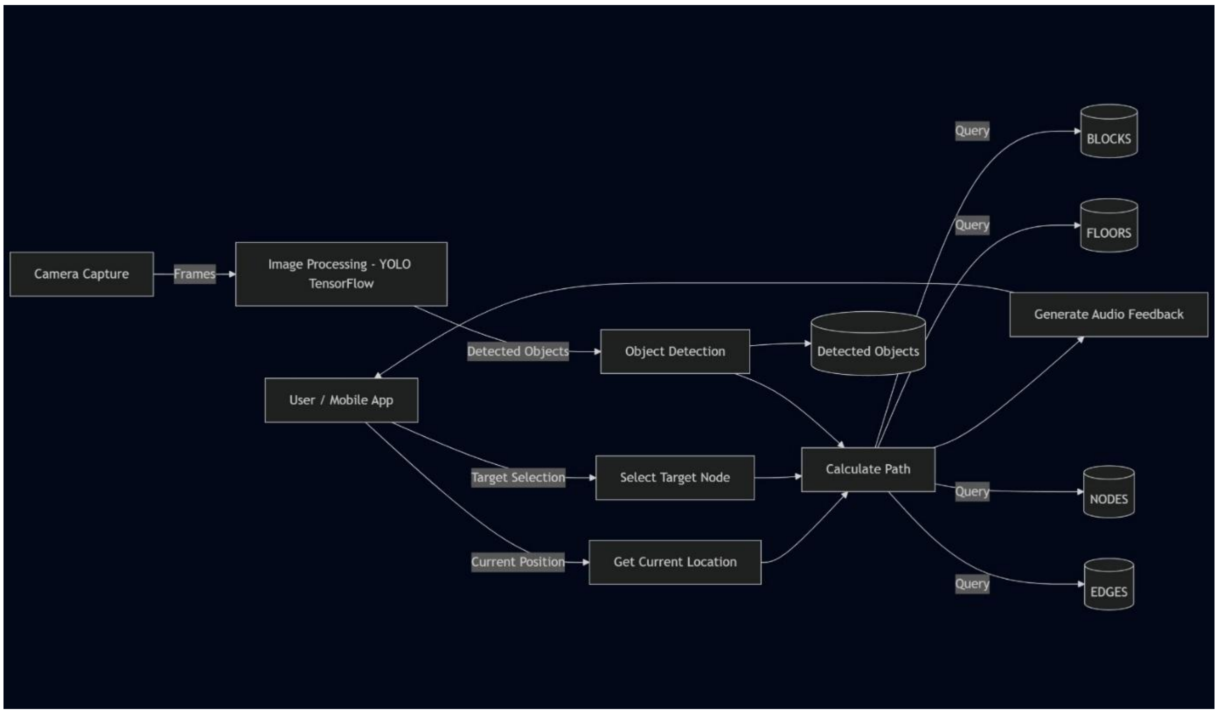
Summary of Relationships:

- **BLOCKS → FLOORS:** One block has multiple floors (1:N).
- **FLOORS → NODES:** One floor contains multiple nodes (1:N).
- **NODES → EDGES:** Nodes are connected through edges (1:N).

This schema ensures data integrity, organizes building layout efficiently, and supports accurate pathfinding.

4.2 Data Flow Diagrams

The Data Flow Diagram (DFD) illustrates how information moves through the system, including both navigation and real-time object detection components. The diagram shows processes, data stores, external entities, and the flow of data between them.



Key Components:

1. **External Entities:**
   - **User / Mobile App:** Provides current location and selects a target destination.
2. **Processes:**
   - **Camera Capture:** Captures real-time video frames.
   - **Image Processing (YOLO/TensorFlow):** Preprocesses frames for object detection; runs in a separate Dart Isolate for parallel execution.
   - **Object Detection:** Detects humans and other objects using YOLO/TensorFlow; results are stored in `DETECTIONS_DB`.
   - **Get Current Location:** Retrieves the user's GPS or indoor position.
   - **Select Target Node:** Allows the user to choose a navigation target.
   - **Calculate Path:** Computes optimal routes using building data and detected objects.
   - **Generate Audio Feedback:** Converts path and obstacle information into directional audio cues.
3. **Data Stores:**
   - **BLOCKS, FLOORS, NODES, EDGES:** Store building structural and spatial information.
   - **DETECTIONS_DB:** Stores objects detected in real time.
4. **Data Flows:**
   - User inputs location and target.
   - Camera frames are processed in parallel for object detection.
   - Detected objects influence the path calculation.
   - Audio instructions are generated and delivered to the user.

5. Interface Design

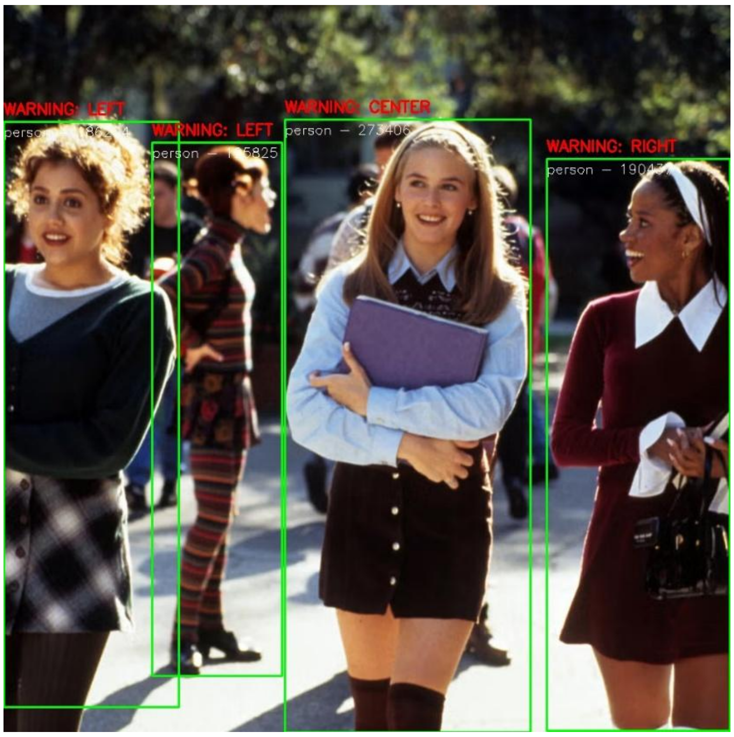
The **user interface (UI)** of the application was developed with a focus on **accessibility**, **clarity**, and **real-time responsiveness**, tailored specifically for visually impaired users. The design follows a minimal yet functional structure, consisting of two main interfaces: one for **object detection and proximity awareness**, and the other for **indoor navigation**.

5.1 Object Detection Interface

The **Object Detection Interface** provides real-time visual and auditory feedback based on the camera input. The user's device camera captures the environment, and detected entities such as **humans, elevators,** and **staircases** are enclosed within **bounding boxes** and labeled with corresponding text identifiers on the screen.

The detection model processes each frame independently in a **Dart Isolate**, ensuring that the intensive image processing tasks do not affect the main UI thread or cause frame drops. The system uses **auditory cues** to communicate proximity information to the user. When a person is detected, the application emits a **beeping sound** whose **frequency increases** as the detected person approaches. Additionally, **directional sound cues** are provided — the sound is louder in the left or right ear depending on the direction of the detected person relative to the user's orientation.

This interface is designed to enhance **spatial perception** for visually impaired individuals, enabling them to sense nearby obstacles and moving entities without requiring visual confirmation.

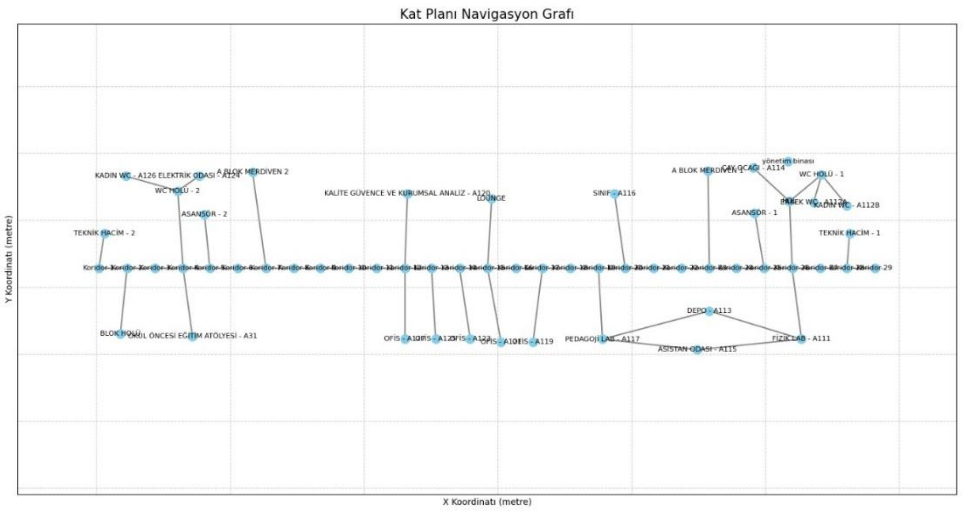


5.2 Navigation Interface

The **Navigation Interface** presents a **graph-based indoor map** representing the layout of the building, generated from the **AutoCAD drawings** of the school. Each **node** on the graph corresponds to a physical location (e.g., corridor intersection, elevator, or stair entrance), and **edges** define traversable paths between these nodes.

Using the **Geolocator** package, the system determines the user's position in real time and marks it as a dynamic node on the graph. As the user moves, their location is continuously updated, and the interface visually and audibly guides them along the optimal path to their destination. Audio instructions provide direction changes (e.g., "turn left," "proceed straight") and alerts for approaching points of interest.

To maintain smooth performance, navigation computations and path updates are handled in an **Isolate**, preventing any interference with user interactions or map rendering. The interface uses **clear visual contrasts**, **simple graph animations**, and **focused layouts** to represent position and movement effectively.



6. Algorithm and Logic Design

The core functionality of the system revolves around **real-time object detection** and **indoor navigation**, combined with **directional audio feedback** to guide visually impaired users safely. The algorithms are designed to operate efficiently on mobile devices while providing continuous, accurate, and responsive guidance.

6.1 Object Detection and Distance-Based Feedback

1. **Input Acquisition:**
   - The system continuously captures video frames from the device's camera.
   - Each frame is sent to a **TensorFlow Lite (TFLite) model**, which has been custom-trained to detect humans, stairs, elevators, and tactile walking paths.
2. **Detection Process:**
   - The TFLite model performs inference on each frame to identify objects of interest.
   - The output contains the object type, bounding box coordinates, and detection confidence.
3. **Proximity Estimation:**
   - The system estimates the distance to detected objects based on the size of the bounding box and the camera's intrinsic parameters.
   - Distance is used to **modulate the frequency of the audio alert**: closer objects result in faster beep rates, signaling urgency.
4. **Directional Audio Feedback:**
   - Using the device's orientation data from the magnetometer and accelerometer, the system determines the **relative direction of each detected object**.
   - The audio output is spatially modulated, providing stronger or more frequent signals from the direction of the object.
   - This allows users to intuitively understand **both the proximity and location** of obstacles in their environment.

6.2 Indoor Navigation Algorithm

1. **Graph Representation of the Environment:**
   - The school's floor plans are represented as a **graph**, where nodes correspond to rooms, hallways, staircases, and elevators, and edges represent navigable paths between them.
2. **Localization:**
   - In **online mode**, the system uses the **Geolocator** package to obtain fused location estimates from GPS, Wi-Fi, cellular, and sensor data.
   - In **offline mode**, dead reckoning is performed using accelerometer and gyroscope readings to estimate movement along paths.
3. **Pathfinding:**
   - Given a start and destination node, the navigation engine computes the optimal route using a **shortest-path algorithm** (e.g., Dijkstra or A*).
   - The path is continuously updated as the user moves or as new localization data becomes available.
4. **Guidance Delivery:**
   - The system provides step-by-step navigation instructions via **voice feedback**.
   - The audio cues include distance to the next node, direction, and alerts for obstacles detected along the path.

6.3 Integration of Detection and Navigation

- Object detection and navigation modules operate concurrently in **separate Dart Isolates** to maintain performance.
- As the user moves, the system continuously fuses **location, orientation, and object detection data** to produce dynamic guidance:
  1. Update user position on the graph.
  2. Detect obstacles in proximity.
  3. Generate directional audio cues based on both navigation path and object locations.
  4. Repeat in real-time.

This architecture ensures that users receive **continuous, context-aware, and intuitive guidance**, enabling safe and independent navigation in indoor environments.

7. Security Design

The system is designed to operate entirely **on-device**, without requiring user accounts or external server connections, minimizing potential security risks related to personal data and network exposure. The primary security considerations focus on **data protection, secure storage, and privacy-preserving design**.

7.1 Authentication and Authorization

- The application does not implement user authentication or authorization mechanisms, as all functionalities are accessible directly on the user's device.

- This design choice reduces complexity while eliminating potential attack vectors associated with credential management.

7.2 Data Encryption and Protection (Revised)

- **Persistent data:** The only data stored locally is the **graph representation of the school's floor plan**, derived from authorized AutoCAD drawings.
- **Access control:** This database is **read-only for the application logic** and **inaccessible to end users**, preventing unauthorized viewing or modification of the indoor map data.
- **Data security measures:** Standard platform-level protections (SQLite permissions and application sandboxing) ensure that even if the device is used by other applications or users, the stored graph remains secure.
- **Privacy-by-design:** By restricting access and storing only non-sensitive mapping data, the system maintains both security and privacy without exposing any personal or environmental information.

7.3 Security Implications for Voice Interaction

- All voice feedback is generated in real-time and is ephemeral, not being recorded or stored on the device.
- Since the system does not transmit any data externally, the risk of interception of user activity or personal information is effectively mitigated.

8. Error Handling and Recovery

The system is designed to ensure **robust operation** under varying conditions, including sensor inaccuracies, hardware limitations, and unexpected runtime exceptions. Error handling and recovery mechanisms have been implemented to maintain **continuous user guidance** and prevent application crashes.

8.1 Exception Handling

- All critical modules, including camera capture, TFLite inference, sensor reading, and navigation computations, are wrapped in **try-catch blocks** to gracefully handle runtime exceptions.
- In case of failures such as camera access denial, sensor malfunction, or inference errors, the system logs the event locally for debugging and continues operating in a **degraded but functional mode**.

8.2 Fallback Mechanisms

- **Navigation Fallback:**
  - When GPS or network signals are unavailable (offline mode), the system automatically switches to **dead reckoning** using accelerometer and gyroscope data.
  - If sensor data temporarily fails, the last known position is maintained until readings stabilize.
- **Object Detection Fallback:**

    - If TFLite inference fails or produces low-confidence outputs, the system temporarily suspends obstacle alerts while keeping navigation instructions active.
    - This prevents false alarms and ensures user safety.

8.3 System Restoration and Continuity Measures

- **Dart Isolates Recovery:**
    - Long-running processes such as image recognition and navigation computations run in separate Dart Isolates.
    - If an isolate crashes, it is automatically restarted without affecting the main UI thread, preserving application responsiveness.
- **User Experience Continuity:**
    - Audio guidance is buffered to prevent gaps in notifications.
    - The system continuously monitors module health and restores interrupted services as soon as possible.

9. Glossary

Term	Definition
Dart Isolate	A mechanism in Dart that allows code to run concurrently on separate threads, isolating heavy computations from the main UI thread.
TFLite (TensorFlow Lite)	A lightweight, mobile-optimized version of TensorFlow used for on-device machine learning inference.
Dead Reckoning	Navigation method that estimates current position based on previous location and motion sensor data when GPS is unavailable.
Geolocator	Flutter package that provides fused location data using GPS, Wi-Fi, cellular, and sensor information.
Graph-Based Map	Digital representation of indoor environments where nodes correspond to locations and edges represent navigable paths.
Object Detection	Process of identifying and locating objects within an image or video frame using machine learning models.
Proximity-Based Audio Feedback	Audio cue system that modulates beep frequency and direction based on distance and orientation relative to obstacles.
Offline Mode	Operation mode in which the system relies solely on device sensors for navigation due to unavailable GPS/network signals.
Online Mode	Operation mode in which the system uses GPS, Wi-Fi, cellular data, and sensors for accurate location estimation.
Fallback Mechanism	Predefined strategy to maintain functionality when a system component fails or produces unreliable data.

---

10. References

1. IEEE Std 1016-2009, *IEEE Standard for Information Technology—Systems Design—Software Design Descriptions*.
2. TensorFlow Lite Documentation: https://www.tensorflow.org/lite
3. Flutter Geolocator Package: https://pub.dev/packages/geolocator
4. Flutter Sensors Package Documentation: https://pub.dev/packages/sensors_plus
5. Ultralytics YOLOv8 Model Documentation: https://docs.ultralytics.com
6. Android Accessibility Guidelines: https://developer.android.com/guide/topics/ui/accessibility
7. Dart Isolate API Documentation: https://dart.dev/guides/libraries/concurrency
8. AutoCAD Official Documentation: https://www.autodesk.com/products/autocad/overview