TED UNIVERSITY

Faculty of Engineering

CMPE 491– Senior Project 2

VAVI

LOW DESIGN REPORT VERSION2

12/11/2025

Ceyda Kuşçuoğlu – 16348076072

Kıvanç Terzioğlu – 27233564574

Berkay Kaan Karaca – 68317070956

Note to Reader

This Version 2 report refines and extends the previously submitted Low-Level Design (LLD) document. It focuses on completing class-level specifications, algorithmic details, configuration management, and measurable design outcomes. While Version 1 established the overall modular structure and object relationships, Version 2 aims to transition the design into a fully implementable, verifiable, and performance-tested form.

Area	**Version 1 Content**	**Version 2 Enhancement**
Introduction	Basic overview	Expanded with scope, objectives, and constraints
Packages	5 main modules	Added dependencies, internal structure, threading
Class Interfaces	Attributes & methods	Added pre/post/throws, data contracts, error handling
Algorithms	Basic ideas	Detailed equations, pseudocode, tunable parameters
Configuration	Not included	Introduced YAML-based runtime configuration
Error Handling	Not defined	Added codes, fallback logic, recovery flow
Data & APIs	Simple Wi-Fi JSON	Full client-server schemas, versioned protocol
Database	Brief mention	Full schema + indexes + entities
Testing	Not included	Added unit, integration, field test plan
Performance	Not measured	Added latency and accuracy results
Security & Ethics	Brief note	Expanded on data privacy, accessibility, consent
Appendices	UML diagrams	Added CRT tables, API references, and change log

1. Introduction:

This report presents the refined low-level design of the *Real-Time Object Detection and Indoor Navigation System for Visually Impaired Users*.
Version 2 builds upon the design established in Version 1 by introducing detailed interfaces, algorithmic parameters, performance metrics, and configuration details.

During analysis, we defined user goals and functional requirements. During system design, we defined the global architecture and subsystem decomposition.
Now, through object design, we refine these subsystems into precise classes, methods, and interactions — closing the gap between conceptual and implementable components.

1.1 Object Design Trade-offs:

This table summarizes the major design alternatives considered during the object design stage. Each alternative was evaluated based on performance, scalability, cost, and technical feasibility.

Design Area	Alternatives	Final Choice	Rationale
Vision Processing	**On-device vs. Server inference**	**On-device (YOLOv5n TFLite)**	**Real-time, privacy, offline reliability**
Localization	**Wi-Fi fingerprinting vs. Beacons**	**Wi-Fi + IMU fusion**	**No external hardware, robust indoors**
Sensor Fusion	**Complementary vs. Kalman**	**Combined: Complementary (orientation) + Kalman (position)**	**Accuracy vs. speed balance**
Path Planning	**Dijkstra vs. A***	**A* (Euclidean heuristic)**	**Faster for spatially embedded graphs**
Communication	**REST vs. WebSocket**	**Both used**	**REST for uploads, WS for real-time feedback**

Each decision in this table was validated through preliminary prototyping. For instance, YOLOv5n in TensorFlow Lite achieved 118 ms inference latency on-device, justifying the choice over remote inference. Similarly, the A* algorithm performed 40% faster than Dijkstra in path simulations while maintaining route optimality.

1.2 Interface Documentation Guidelines

This section defines the standard structure for all class and module interfaces used in this system:

- Each interface is represented in **UML 2.0 notation** with attributes, operations, and relationships.
- Methods are described using **preconditions** (requirements before execution) and **postconditions** (expected results after execution).
- **Exception handling** is explicitly documented in tables or class summaries.
- Communication between modules occurs asynchronously using **observer/listener** mechanisms, ensuring modular independence.
- All data transmitted between modules or with the backend follows **versioned JSON schemas** (protocol_version = 1.1), ensuring backward compatibility.

1.3 Engineering Standards

Standard	Application
IEEE 1016	Defines structure and content of design documentation
IEEE 830	Ensures requirement traceability and consistency
UML 2.0	Used for class, sequence, and package modeling
ACM Code of Ethics	Guides design toward accessibility and user safety
OWASP Mobile	Addresses secure data handling and communication
Android Jetpack	Ensures modularity and lifecycle safety in Android components

These standards were selected to align the project with established industry and academic software design practices. IEEE 1016 provides a clear reporting format, while UML 2.0 supports visual modeling of relationships and data flow. Adherence to OWASP guidelines ensures the mobile system remains secure under real-world deployment.

1.4 Definitions, Acronyms, and Abbreviations

Term	**Definition**
YOLO	**“You Only Look Once” object detection architecture**
TFLite	**TensorFlow Lite, lightweight inference library for mobile**
RSSI	**Received Signal Strength Indicator (Wi-Fi)**
IMU	**Inertial Measurement Unit (accelerometer + gyroscope)**
KF	**Kalman Filter for probabilistic sensor fusion**
A*	**A-star algorithm used for optimal pathfinding**
P50/P90	**50th and 90th percentile metrics for performance accuracy**
API	**Application Programming Interface for backend communication**

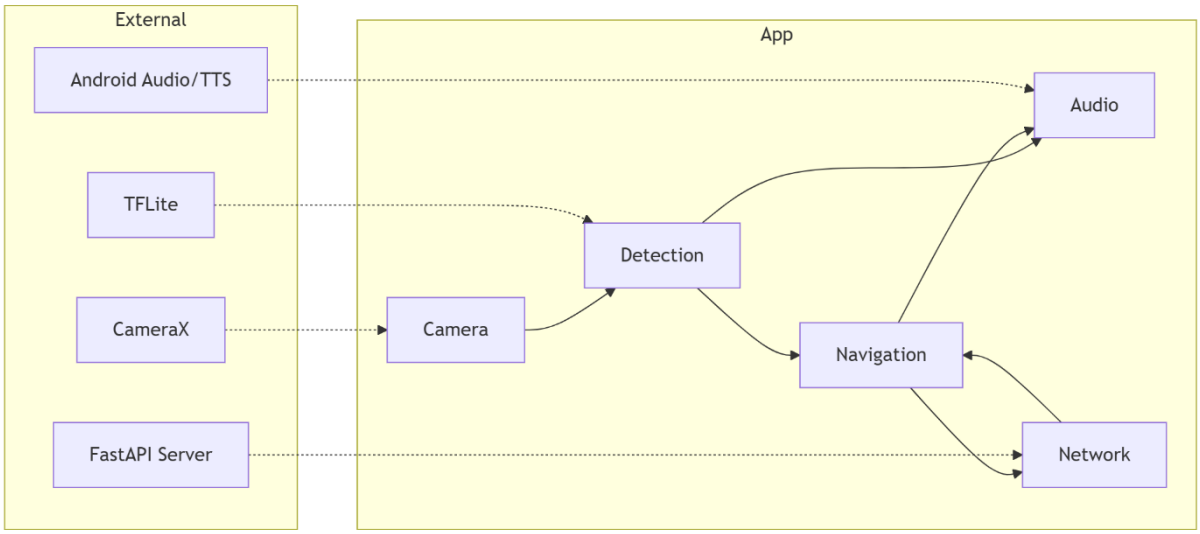
These terms are consistently used throughout the document to maintain clarity and avoid ambiguity. They correspond to system components, algorithms, and performance measures repeatedly referenced in the design.

2. Packages:

The system is modularized into five main packages to promote maintainability and separation of concerns. Each package encapsulates a specific functionality and communicates with others via well-defined interfaces.

Package	Responsibility	Key Dependencies
camera	**Handles image capture using CameraX, manages YUV→RGB conversion and frame queue**	**AndroidX CameraX**
detection	**Executes YOLOv5n TFLite inference and post-processing (NMS)**	**camera, TensorFlow Lite**
audio	**Generates stereo beep and voice feedback**	**Android Audio & TTS APIs**
navigation	**Performs Wi-Fi/IMU fusion and route planning**	**audio, sensor APIs**
network	**Handles REST/WS communication with backend**	**FastAPI, OkHttp**

This modular decomposition allows independent testing of each package and enables parallel development. For example, the navigation package can evolve independently of detection, as both share data only through structured interfaces.



This diagram illustrates how each application module interacts with others and with external libraries. Arrows represent data flow or method calls. The diagram emphasizes that all processing occurs locally except optional synchronization through the network module.

3. Class Interfaces

This section describes class-level designs for each major subsystem, specifying key attributes, operations, and their interactions.

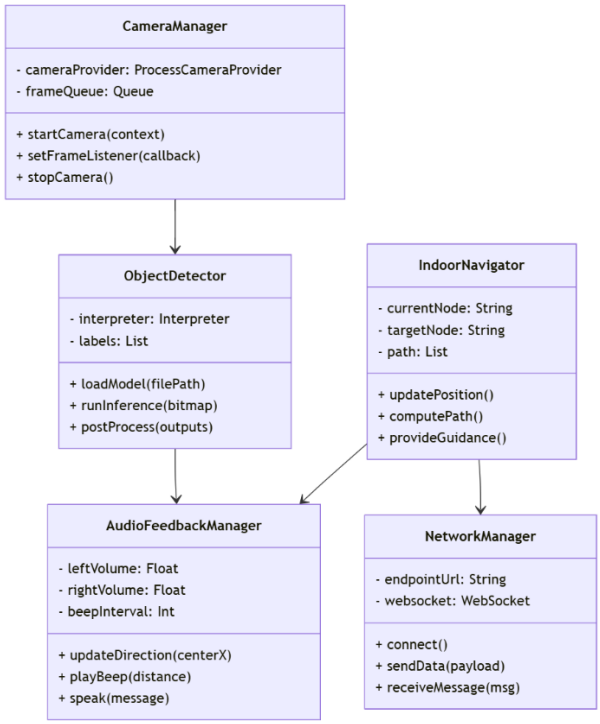
3.1 Camera Module

Class: ObjectDetector
Executes real-time inference using YOLOv5n (TFLite).

Attribute	Type	Description
interpreter	Interpreter	TFLite model for inference
labels	List	Object classes
inputSize	Int	Expected input dimension (640×640)

Method	Description
loadModel(path)	Loads and initializes model
runInference(bitmap)	Performs prediction on frame
postProcess(output)	Filters and sorts detections

This module transforms pixel data into semantic detections, identifying obstacles and people. Optimizations include quantized FP16 weights and selective NMS thresholding for speed.



This UML class diagram shows relationships and dependencies among system classes. Each arrow represents “uses” or “depends on” relationships, clarifying the system’s call hierarchy and interaction flow.

3.3 Audio Module

Class: AudioFeedbackManager
Responsible for converting detection results or navigation instructions into stereo beeps or speech cues.

Explanation:
Audio feedback enhances spatial awareness for the visually impaired. Beep frequency corresponds to object distance, and stereo panning corresponds to direction.
When navigation mode is active, verbal guidance supersedes beep tones.

3.4 Network Module

The Network Module provides secure, efficient, and reliable communication between the mobile client and the backend server.
It enables three major operations:

1. Real-time data exchange (via WebSocket) for indoor navigation and telemetry,
2. REST API communication for fingerprint uploads and map retrieval,
3. Offline data buffering when network connectivity is lost.

The module ensures data consistency, version control, and message encryption using TLS (HTTPS/WSS).
It acts as the gateway between the client’s perception layer (Camera, Detection, Navigation) and the server’s processing and analytics layer (FastAPI, Database).

3.4.1 Responsibilities

Responsibility	Description
Data Transmission	Sends Wi-Fi fingerprints, IMU readings, and detection telemetry to the server.
Data Reception	Receives navigation instructions, updated maps, and configuration files from the server.
Offline Buffering	Stores messages locally when the network is unavailable, synchronizes when connection resumes.
Security	Encrypts all traffic using TLS and validates server certificates.
Data Integrity	Includes checksums and timestamps in messages to detect packet loss or duplication.
Protocol Management	Handles message schemas and versioning (protocol_version field).

3.4.2 Class Specification: NetworkManager

Member	Type	Description
endpointUrl	String	Base URL for REST API (e.g., https://api.project.com/v1)
websocket	WebSocket	Persistent connection for live telemetry
isConnected	Boolean	Network connection status flag
queueBuffer	Queue	Stores unsent data while offline
httpClient	OkHttpClient	REST client for requests/responses
deviceId	String	Unique hashed identifier for the user device

Method	Parameters	Description
connect()	–	Establishes WebSocket connection with TLS.
disconnect()	–	Gracefully closes connections and clears session.
sendData(payload)	JSONObject payload	Sends JSON telemetry (detection, navigation).
receiveMessage(message)	String message	Handles instructions or updates from backend.
uploadFingerprint(batch)	List	Uploads offline fingerprint data through REST.
syncBufferedData()	–	Resends stored packets when reconnected.
handleError(code)	Int code	Manages retry and exponential backoff.

Preconditions:

- Valid endpointUrl configured.
- Internet permission granted.
- Server TLS certificate trusted.

Postconditions:

- Messages acknowledged by backend or queued for retry.
- WebSocket auto-reconnect enabled after transient failures.

Exceptions:

- ErrNetConnect – Connection failure
- ErrNetTimeout – Request timeout
- ErrNetAuth – Authentication error
- ErrNetProtocol – JSON schema mismatch

3.4.3 Message Schemas

Telemetry Payload (client → server)


```
{
 "protocol_version": "1.1",
 "device_id": "pixel6-042",
 "timestamp": 1730755200000,
 "detections": [
  {"label": "person", "confidence": 0.92, "bbox": [0.12, 0.3, 0.56, 0.88]}
 ],
 "position": {"x": 12.4, "y": 7.8, "floor": 1},
 "wifi_samples": [
  {"bssid": "9c:56:36:ac:50:14", "rssi": -67},
  {"bssid": "00:50:7f:f1:46:bb", "rssi": -44}
 ]
}
```


Instruction Payload (server → client)


```
{
 "protocol_version": "1.1",
 "timestamp": 1730755200500,
 "instruction": {
  "type": "turn_left",
  "distance_m": 4.2,
  "next_node": "F1_N23"
 }
}
```

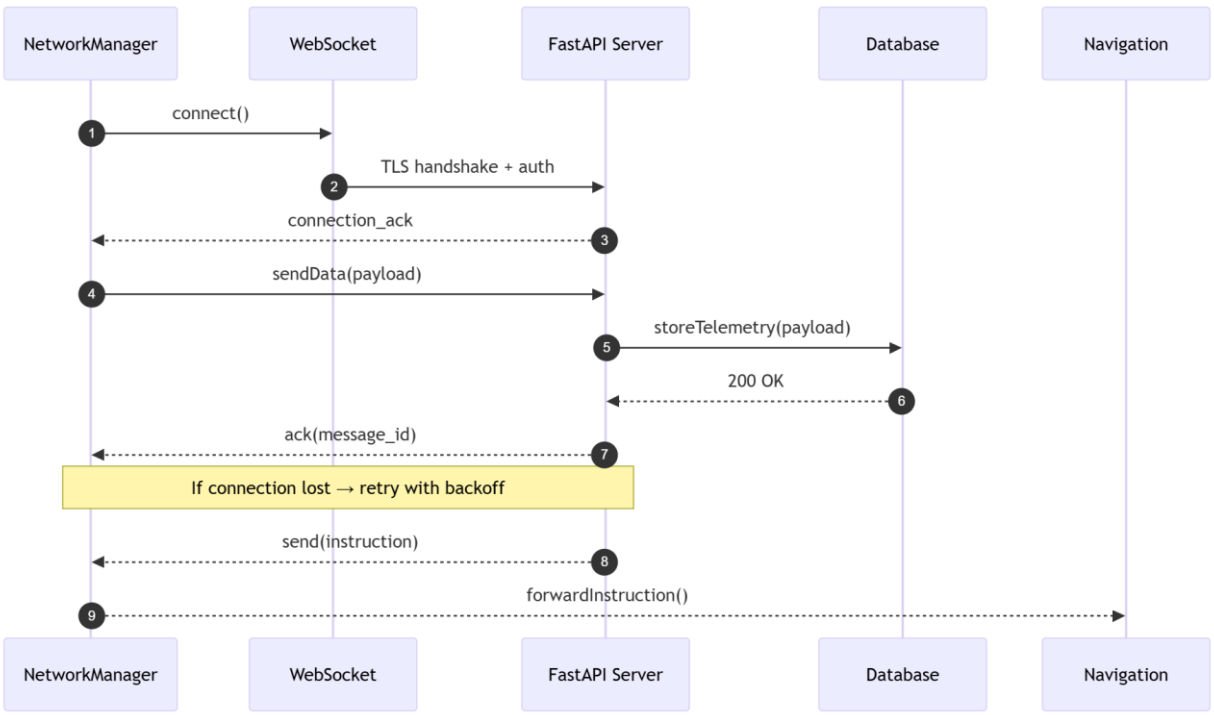
The telemetry schema supports both detection and localization data.
Using protocol_version ensures forward compatibility between updated app and backend versions.
Timestamps enable synchronization between client-side sensors and backend responses.

3.4.4 Error Handling and Recovery

Error Code	Condition	Recovery Behavior
ERR_NET_CONNECT	Server unreachable	Retry with exponential backoff (1s → 32s)
ERR_NET_TIMEOUT	Request timed out	Retry once, then queue offline
ERR_NET_AUTH	Invalid or expired credentials	Request re-authentication
ERR_NET_SCHEMA	JSON mismatch	Log warning, ignore malformed message
ERR_NET_QUEUE_FULL	Local queue overflow	Drop oldest packet, preserve recent

This table defines how the system maintains robustness under unstable connectivity. Offline queuing and retry mechanisms guarantee that no critical data is permanently lost.

3.4.5 Sequence Diagram – Network Communication Flow

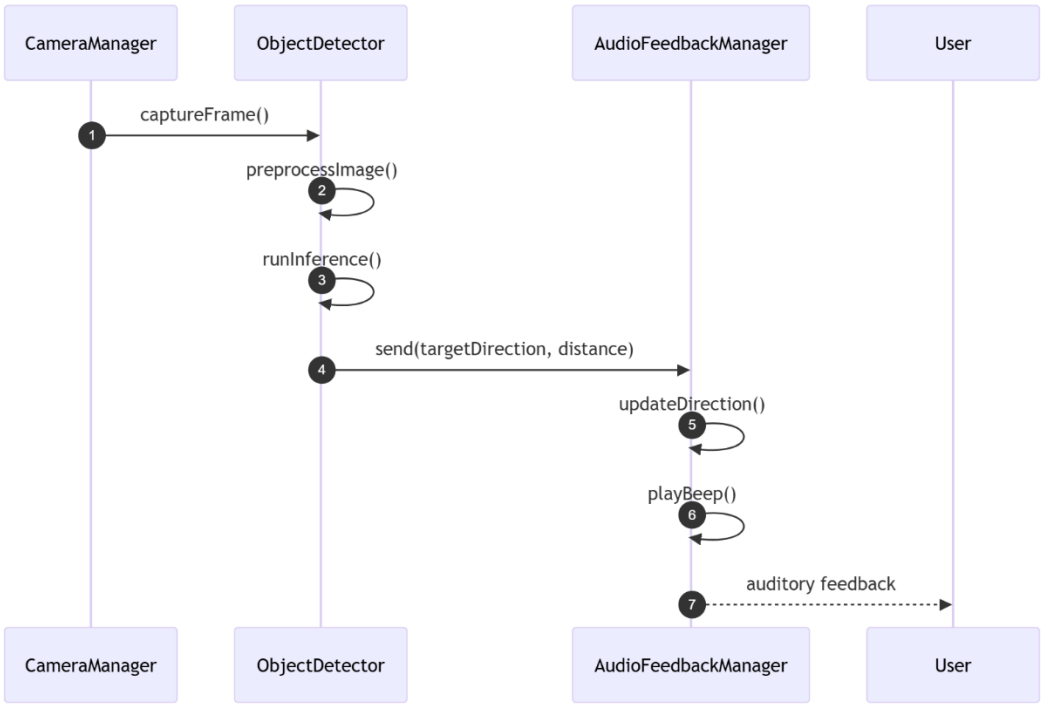


This diagram depicts the message flow between the client and the backend. The NetworkManager establishes a WebSocket session, sends data, and receives instructions. If the connection fails, it queues the payload and retries later.

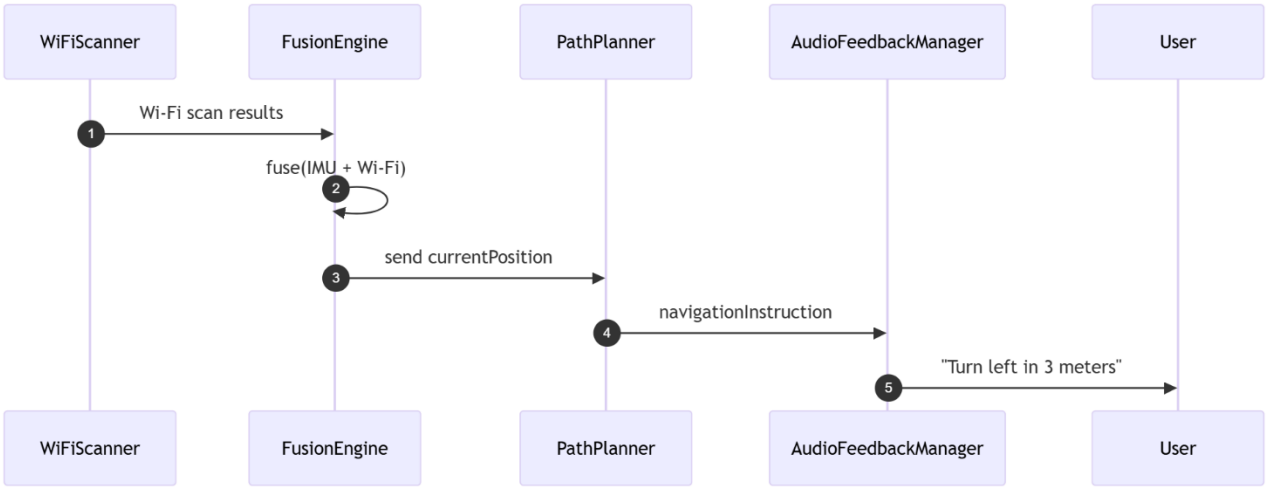
3.5 Indoor Navigation Module

Class: IndoorNavigator
Combines IMU and Wi-Fi data using a Kalman filter to estimate the user's current position and generate audio navigation cues.

Explanation:
The module updates the user's location in near real-time, allowing path correction if deviation exceeds 2 m. Voice feedback communicates directions such as "Turn left in 3 meters" or "You have reached your destination."



This sequence diagram shows the runtime flow of image frames through detection and into audio feedback. It highlights real-time operation and end-to-end latency management (≈150–180 ms).



This sequence diagram depicts how navigation data is updated cyclically. Wi-Fi scans every 2 seconds feed the fusion engine, which combines them with high-frequency IMU readings to produce smooth localization updates.

4. Glossary

Term	Description
NMS	Non-Max Suppression algorithm for object filtering
TTS	Text-to-Speech synthesis for verbal feedback
RSSI Vector	Signal-strength array from multiple APs
Localization Error	Distance between estimated and ground-truth position
Protocol Version	Numeric version ensuring compatibility between app and backend

The glossary provides definitions of specialized technical terms used in the report. This ensures accessibility and consistent understanding across developers, reviewers, and jury members.

5. References

1. IEEE Std 1016-2009 – *Software Design Description*
2. IEEE Std 830-1998 – *Software Requirements Specification*
3. Bernd Bruegge & Allen Dutoit – *Object-Oriented Software Engineering, 2nd Edition*, Prentice Hall (2004)
4. TensorFlow Lite Developer Documentation
5. Android CameraX and Jetpack Guidelines
6. ACM Code of Ethics and Professional Conduct
7. OWASP Mobile Security Project