0. Introduction.

Continuous monitoring of patients' vital signs is a fundamental requirement in modern healthcare systems, where early identification of abnormal conditions can significantly impact patient safety and clinical outcomes. Despite the availability of monitoring technologies, many solutions remain limited in their ability to provide timely, reliable, and continuous detection of clinically relevant events under real-world operating conditions.

MedAlert is a medical monitoring system designed to support early detection of physiological anomalies by continuously analyzing patient vital signs and generating alerts when abnormal behavior is identified. The system focuses on reliability, responsiveness, and operational continuity, aiming to support medical staff in making timely decisions while reducing unnecessary alarms and manual supervision.

This document presents the design and development of the MedAlert system. It introduces the problem context and system objectives, reviews relevant technological approaches, defines system requirements, and describes the architectural and design decisions underlying the solution. The document further outlines the planned validation approach used to assess system behavior against defined performance and reliability criteria.

Table of Contents

1. Problem description:

In healthcare environments, continuous monitoring of patients' vital signs is critical for detecting clinical deterioration and preventing adverse events. Vital parameters such as heart rate, oxygen saturation, and blood pressure may change rapidly, and delayed identification of abnormal patterns can significantly increase medical risk. In practice, medical staff often rely on a combination of periodic checks and monitoring systems that do not always provide timely or reliable alerts.

Existing monitoring solutions commonly face several challenges. These include delays caused by centralized processing, reduced reliability during network disruptions, and high false-alarm rates that contribute to alert fatigue among medical personnel. In addition, noisy or low-quality sensor data may lead to incorrect detections if not properly validated, further reducing trust in automated monitoring systems.

The problem addressed in this project is the lack of a reliable and responsive software system that can continuously analyze patient vital signs, identify clinically meaningful anomalies in real time, and notify medical staff within a strict time constraint. Such a system must operate effectively under real-world conditions, including unstable connectivity, varying signal quality, and the need to monitor multiple patients simultaneously, while minimizing unnecessary alerts and maintaining high availability.

2. Technological aspect:

2.1. Technological survey:

Continuous patient monitoring systems integrate physiological sensors, data processing components, and alerting mechanisms to identify abnormal medical conditions. Existing technological approaches differ mainly in the location of data processing, the analysis methods applied, and their dependence on network connectivity.

Many solutions rely on centralized processing, where sensor data is transmitted to a central server or cloud platform for analysis. While this enables simplified management and data aggregation, it may introduce latency and reduce system reliability in the presence of network disruptions.

Edge-based processing has emerged as an alternative approach, enabling initial signal validation and analysis close to the data source. This reduces response time and allows continued operation when connectivity is unstable, which is particularly important in time-critical medical environments.

From an analytical perspective, monitoring systems commonly employ rule-based techniques such as threshold detection, sometimes combined with trend analysis, to identify abnormal behavior in physiological signals. In addition, reliability mechanisms such as temporary data buffering and alert prioritization are used to support consistent system operation under real-world conditions.

2.2. Current solutions survey:

Patient monitoring systems are widely used in clinical environments to track vital signs and support early detection of medical deterioration. Commercial solutions such as **Philips IntelliVue**, **Masimo Root**, and **EarlySense** provide continuous measurements, alarm mechanisms, and integration with hospital information systems.

Most existing solutions rely on centralized architectures, in which sensor data is transmitted to central monitoring stations or cloud-based platforms for processing. While this approach enables centralized management, it introduces dependency on network availability and may affect response time in critical situations.

Some systems attempt to reduce false alarms using adaptive thresholds or proprietary analytics. However, these mechanisms are often tightly coupled to specific hardware ecosystems and provide limited flexibility or transparency. Support for continued operation during network disruptions is typically constrained.

The strengths and limitations of current solutions emphasize the importance of low alert latency, reliability, and operational continuity, which serve as key factors for evaluating alternative system designs.

2.3. Parameters for solution comparison:

To enable a systematic comparison between different patient monitoring solutions, the following evaluation parameters are defined:

- **Alert Latency**
  - Measures the time between detection of an abnormal physiological condition and notification of medical staff.
  - Critical in clinical environments where delayed response may increase medical risk.
- **System Availability and Reliability**
  - Reflects the ability of the system to operate continuously over time.
  - Includes fault tolerance and recovery from system or component failures.
- **Operational Continuity under Network Disruptions**
  - Evaluates whether monitoring and alerting can continue when network connectivity is unstable or unavailable.
  - Important for real-world healthcare environments where connectivity cannot be always guaranteed.
- **False-Alarm Handling**
  - Relates to the system's ability to reduce unnecessary or incorrect alerts.
  - Directly affects alert fatigue and the level of trust medical staff place in the system.
- **Scalability**
  - Represents the capability of the system to support multiple monitored patients simultaneously.
  - Important for deployment in clinical settings with varying patient volumes. These parameters form the basis for comparing existing solutions and for evaluating architectural and design alternatives discussed in subsequent sections.

2.4. Decision justifications (using above parameters):

The **comparison parameters** defined in the previous section were selected to reflect the most critical limitations observed in existing patient monitoring solutions, as well as the operational needs of real-world clinical environments. While commercial monitoring systems generally provide comprehensive measurement and alerting capabilities, they often emphasize **centralized processing** and infrastructure integration over real-time responsiveness and operational robustness. As a result, aspects such as **alert latency**, **behavior under network disruptions**, and consistency of operation are not always explicitly addressed or evaluated as primary criteria.

The selected parameters therefore focus on **system behavior under non-ideal conditions**, including unstable connectivity, high workload, and the requirement for continuous monitoring. These conditions are common in practice, yet many existing solutions address them only partially and may degrade significantly when assumptions regarding connectivity or infrastructure availability are violated. Furthermore, current solutions frequently rely on **proprietary hardware** and closed analytics mechanisms, limiting transparency and flexibility in handling **false alarms** and signal quality issues.

Taken together, the selection of these parameters highlights clear gaps between the capabilities of existing solutions and the requirements of a **robust and responsive monitoring system**. Addressing these gaps provides the justification for exploring alternative system designs and motivates the development of a new monitoring platform rather than relying solely on existing commercial solutions.

3. Stakeholders' description:

Stakeholder	Role / Involvement	Main Needs / Scenarios
Doctors	Decision makers	Real-time anomaly alerts, patient history, clinical trends, dashboards.
Nurses	Primary operators	Live monitoring, alerts, patient management, documentation.
Hospital/Clinic Managers	Supervisors	Operational efficiency, reduced workload, scalability, compliance.
IT/Engineering Staff	Deployment & maintenance	Hardware integration, network management, troubleshooting.
Patients	End users	Accurate monitoring, early warning, preserved privacy.
Regulatory Authorities	Oversight	Compliance with GDPR/HIPAA and medical data privacy.

Table 1: Stakeholders

The MedAlert system involves multiple stakeholders with distinct clinical and operational roles, as summarized in **Table 1**. These stakeholders influence system requirements, design priorities, and operational constraints.

Doctors and **nurses** are the primary users of the system. Doctors rely on timely anomaly alerts, access to patient history, and clinical trends to support medical decision-making, while nurses interact with the system continuously for live monitoring, alert handling, and documentation. Their shared priorities include accuracy, low response time, and minimizing false alarms.

Hospital or clinic managers act as supervisory stakeholders, focusing on operational efficiency, workload management, scalability, and regulatory compliance. **IT and engineering staff** are responsible for system deployment, maintenance, and integration with existing infrastructure, emphasizing system stability, network reliability, and ease of troubleshooting.

Patients are indirect end users who depend on accurate monitoring, early warning of abnormal conditions, and protection of their medical data. **Regulatory authorities** provide oversight and ensure compliance with medical and data protection regulations.

Considering the needs of these stakeholders supports effective system design and adoption in clinical environments.

4. Requirements:

4.1. Functional requirements (mapped stakeholders' description above):

The functional requirements of the MedAlert system define the core capabilities required to support continuous patient monitoring, real-time anomaly detection, and alert generation. These requirements are derived from stakeholder needs and clinical use scenarios and are formally specified in **Table 2**.

The system supports continuous collection of physiological data, edge-level signal validation and processing, offline operation, and the generation of medical alerts based on detected anomalies and deteriorating trends. All functional requirements are uniquely identified to support traceability across system design, implementation, and validation phases.

ID	Name	Description
F1	Continuous Data Collection	Collect continuous streams of heart rate, SpO_2, temperature, and blood pressure.
F1.1	Multi-Sensor Support	The system shall support multiple sensors per patient and receive measurements at defined intervals (ΔT).
F1.2	Signal Quality Check	The Edge device shall validate signal quality (noise filtering, missing-data detection) and flag low-quality measurements.
F1.3	Timestamping & Logging	All measurements shall be timestamped and stored locally (offline) or centrally (online) with patient ID, sensor ID, and quality metadata.
F2	Edge Processing	All incoming data shall be analyzed locally on the Edge device to ensure real-time detection even without network connectivity.
F2.1	Threshold Detection	The Edge shall compare each measurement to medical threshold ranges and trigger anomaly events within ≤ 2 seconds.
F2.2	Trend Analysis	The system shall analyze historical data to detect deterioration trends and generate early-warning alerts.
F2.3	Offline Operation	The Edge shall continue full operation during network outages and securely cache ≥ 24h of data.
F3	Alert Generation	The system shall generate alerts for medical anomalies, deterioration trends, and system/sensor faults.
F3.1	Critical Alerts	The system shall generate a *Critical Alert* when a value exceeds a medical threshold. The alert must be produced and visible on the dashboard within ≤ 2 seconds, including patient ID, value, expected range, timestamp, and severity.

Table 2: Functional requirements.

The requirements listed above form the functional foundation of the system and serve as the basis for architectural design and subsequent validation activities.

4.2. General functional requirements:

The general functional requirements define cross-cutting system behaviors and operational constraints that apply to the functional capabilities specified in Section 4.1. These requirements ensure controlled usage, safe operation, and effective system management in clinical environments.

- **GFR-1: User Authorization and Access Control:**
  The system shall restrict access to monitoring data, alerts, and administrative functions to authorized users only, according to their roles.
- **GFR-2: Alert Acknowledgment and Handling:**
  The system shall allow authorized medical staff to acknowledge, review, and manage active alerts.
- **GFR-3: Audit Logging:**
  The system shall record system events, alerts, and user actions for traceability, auditing, and operational review.
- **GFR-4: Fault Handling and System Recovery:**
  The system shall detect operational faults and support recovery without compromising patient monitoring.
- **GFR-5: Configuration Management:**
  The system shall allow authorized administrators to update monitoring policies and system configurations.
- **GFR-6: Data Privacy Support:**
  The system shall support functional mechanisms that protect patient data and prevent unauthorized exposure.

These general functional requirements apply across all system functions and support safe, controlled, and reliable operation.

4.3. Non-Functional requirements:

The non-functional requirements define measurable quality attributes and operational constraints that govern the performance, reliability, security, and scalability of the MedAlert system. These requirements complement the functional requirements presented in Sections 4.1 and 4.2 and are derived directly from system goals and stakeholder needs.

The non-functional requirements are formally specified in **Table 3**, including quantitative targets that enable objective validation of system behavior.

ID	Category	Requirement	Metric / Target
NFR1	Performance	Alert generation speed	$\leq$ 2 seconds
NFR2	Performance	Data refresh rate	Once per ΔT (configurable)
NFR3	Availability	System uptime	$\geq$ 99%
NFR4	Reliability	False alarm rate	$\leq$ 5%
NFR5	Reliability	Offline operation	$\geq$ 24 hours of cached data
NFR6	Scalability	Patient capacity	20–50 per Edge unit
NFR7	Security	Secure communication	TLS/SSL + role-based access
NFR8	Privacy	Data protection	GDPR/HIPAA compliant

Table 3: Non-Functional requirements.

The requirements listed above define clear performance and quality targets and serve as the basis for system evaluation and validation, as described in the validation chapter.

5. Architecture and high-level design.

This chapter presents the overall architecture and high-level design of the MedAlert system. The architecture is designed to support continuous patient monitoring, real-time anomaly detection, and reliable alert generation, while meeting the performance, availability, and reliability requirements defined in the SRS and refined in the SDD and LLD.

The system adopts a distributed architecture that separates time-critical processing at the edge from centralized services responsible for coordination, storage, and user interaction.

5.1. Architectural Overview:

The MedAlert system is organized into three main architectural layers:

- **Sensors Layer** – Wearable sensors that continuously measure patient vital signs, including heart rate, SpO_2, temperature, and blood pressure.
- **Edge Layer** – A local processing unit responsible for data acquisition, signal quality validation, anomaly detection, and offline operation.
- **Central Services and User Layer** – Central backend services, medical databases, and user interfaces for clinical and administrative stakeholders.

This layered separation enables low alert latency, resilience to network disruptions, and scalable system growth.

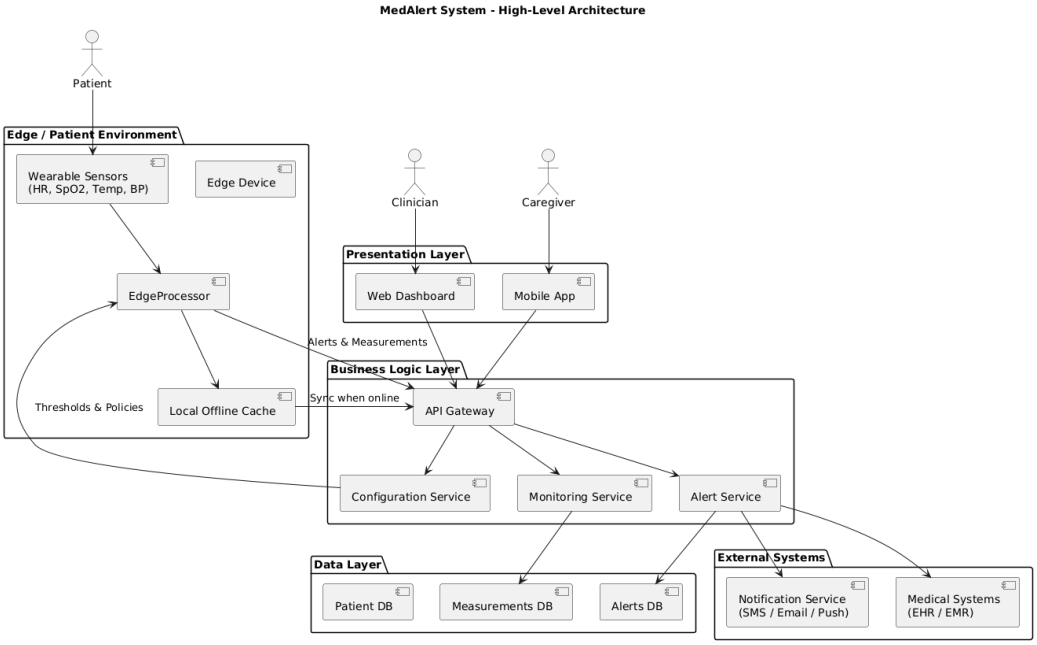

Figure 1: High-Level System Architecture.

5.2. Edge-Level Design (High-Level)

The Edge layer is responsible for all time-critical operations and local decision-making. Its main responsibilities include:

- Receiving continuous streams of physiological data from multiple sensors per patient
- Performing signal filtering and quality validation
- Comparing measurements to medical thresholds
- Detecting anomalies and deterioration trends
- Generating alert events
- Supporting offline operation with secure local data caching

By processing data locally, the system ensures timely detection and alert generation even when connectivity to central services is unavailable.

5.3. Central Services and User Interaction

Central services provide system-wide coordination and long-term data management, including:

- Aggregation and storage of medical measurements and alerts
- Patient profile and configuration management
- Alert distribution and escalation
- Audit logging and compliance support

The user interaction layer includes dashboards for doctors and nurses, allowing real-time visualization of patient status, active alerts, and recent measurements, as well as administrative views for system configuration and monitoring.

5.4. High-Level System Flows

To complement the architectural description, this section presents the main high-level system flows. These flows illustrate how architectural components interact during key operational scenarios, without detailing internal module implementations.

SF1 – High-Level Monitoring and Alert Flow

Wearable sensors continuously measure patient vital signs and transmit data to the Edge unit. The Edge performs signal filtering, quality checks, and compares measurements to predefined medical thresholds. When an anomaly is detected, an alert event containing the alert type, severity, timestamp, and patient ID is generated and securely transmitted to the central backend. Central services update patient status, store data in the medical history database, and update the monitoring dashboard in near real time.

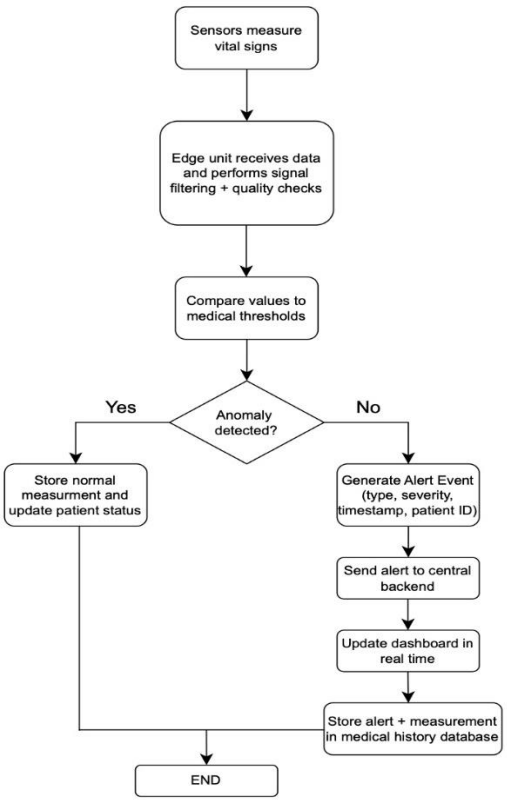

Figure 2: High-Level Monitoring and Alert Flow.

SF2 – Patient Registration and Sensor Assignment Flow

Medical staff create or select a patient profile via the dashboard. Patient identifiers are validated and stored in the central database. Sensors and monitoring parameters are assigned to the patient and securely transmitted to the corresponding Edge unit, which confirms sensor pairing and begins continuous monitoring.

SF3 – Error and Degradation Handling Flow

When network connectivity between the Edge unit and central services is lost, the Edge switches to offline mode, continues local processing, and securely caches all events. The dashboard displays the last known patient state and indicates degraded operation. Once connectivity is restored, cached events are transmitted to the central backend, patient records are updated, and synchronization issues are logged for administrative review.

5.5. Architectural Alignment with Requirements

The architecture and system flows described in this chapter directly support the functional and non-functional requirements defined in Chapter 4. Edge-level processing ensures low alert latency and offline operation, while centralized services support scalability, security, and regulatory compliance.

6. Low level design.

The purpose of this chapter is to refine the system architecture into a detailed, implementation-ready design, focusing on the internal structure, responsibilities, and interactions of the system's core components.

Based on the high-level architecture presented in Chapter 5, the **EdgeProcessor** module was selected for detailed low-level design, as it encapsulates the system's real-time processing logic and operates under strict performance and reliability constraints.

6.1. EdgeProcessor Module – Overview

The **EdgeProcessor** is the core processing component deployed at the system edge. It receives raw physiological measurements from connected sensors and processes them locally to meet real-time performance and reliability requirements.

The main responsibilities of the EdgeProcessor include:

- Ingestion of sensor measurements
- Validation of signal quality and measurement plausibility
- Local signal processing and sliding-window aggregation
- Detection of threshold-based and trend-based anomalies
- Generation and emission of alert events
- Local buffering and synchronization during offline operation

6.2. EdgeProcessor – Internal Sub-Modules

To improve modularity, testability, and future extensibility, the EdgeProcessor is decomposed into focused internal sub-modules.

6.2.1 Signal Validation Sub-Module

Responsible for validating signal quality, verifying value plausibility, and detecting missing, duplicated, or out-of-order measurements using timestamp analysis.

6.2.2 Signal Processing Sub-Module

Applies lightweight noise filtering, aggregates measurements using sliding windows, and prepares processed data for anomaly detection.

6.2.3 Anomaly Detection Sub-Module

Detects abnormal conditions using threshold-based analysis and identifies early warning signs through trend-based analysis.

6.2.4 Alert Management Sub-Module

Classifies anomaly severity, generates alert events with contextual metadata, and applies debouncing and suppression rules to reduce false alerts.

6.2.5 Offline Cache and Synchronization Sub-Module

Stores measurements and alert events locally during connectivity loss and synchronizes cached data with backend services when connectivity is restored.

6.3. Data Structures and Internal Models

The EdgeProcessor uses implementation-agnostic internal data models that represent physiological measurements, processing windows, anomalies, alerts, and cached items.

Key models include:

- Measurement Model
- Measurement Window Model
- Anomaly Model
- Alert Event Model
- Offline Cache Entry Model

These models are used consistently across validation, processing, detection, alerting, and synchronization stages.

6.4. Processing Flows (Sequence-Level Design)

The EdgeProcessor executes several well-defined processing flows that describe the interaction between internal sub-modules:

- Measurement Ingestion and Validation Flow
- Local Processing and Window Update Flow
- Anomaly Detection Flow
- Alert Generation and Handling Flow
- Offline Storage and Synchronization Flow

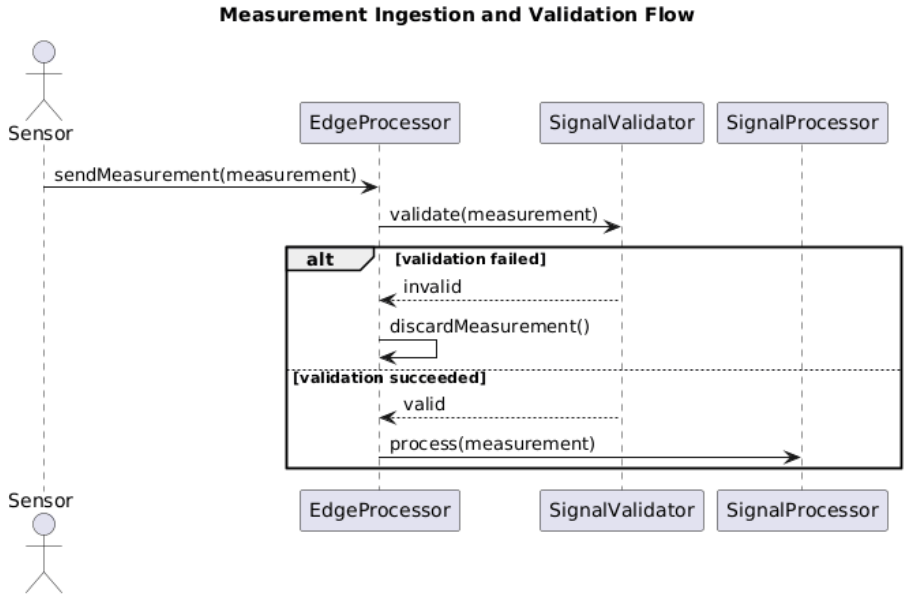

Figure 3:Measurement Ingestion and Validation Flow.

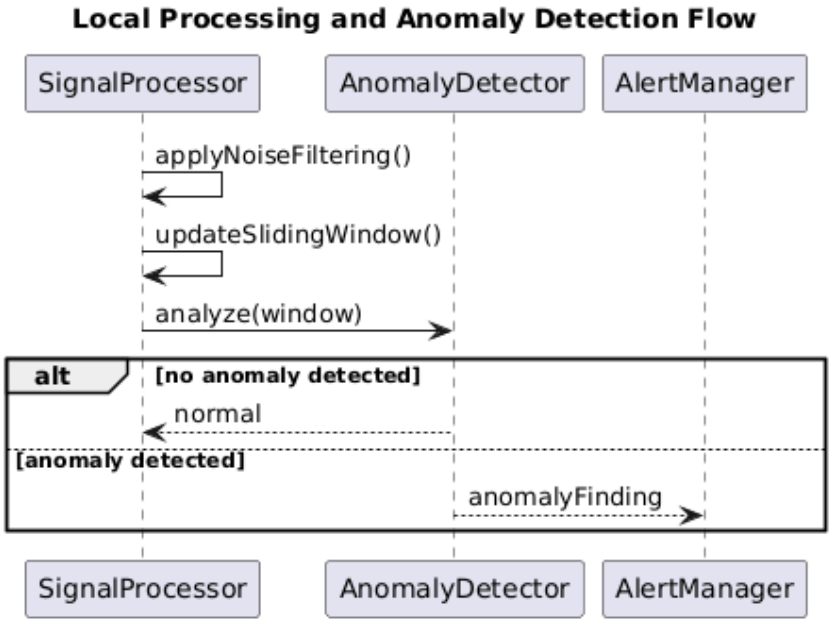

Figure 4:Local Processing and Anomaly Detection Flow.

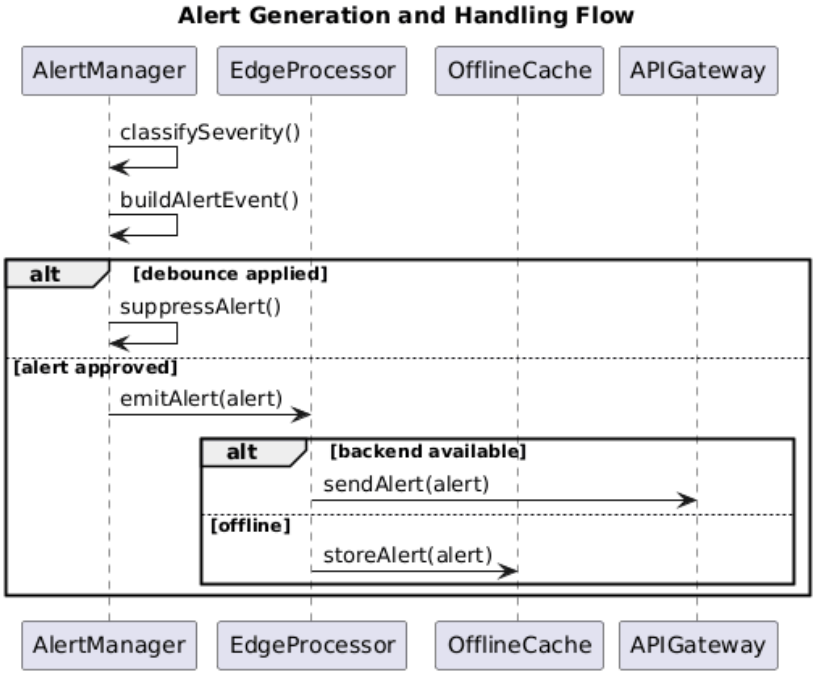

Figure 5: Alert Generation and Handling Flow.

6.5. Error Handling and Edge Cases

The low-level design explicitly addresses potential error conditions and edge cases, including:

- Invalid or corrupted measurements
- Missing, delayed, or out-of-order data
- False positives and alert flooding
- Backend unavailability
- Resource constraints at the edge

Handling strategies are designed to ensure predictable system behavior and prevent cascading failures.

6.6. Assumptions and Design Decisions

The EdgeProcessor design is based on assumptions such as limited edge resources, sensor-agnostic input, intermittent connectivity, and externally configurable detection policies.

Key design decisions include modular decomposition, edge-centric processing, offline-first reliability, and configuration-driven behavior.

7. Additional information,

7.1. Algorithms for Anomaly Detection

The MedAlert system employs **lightweight and deterministic algorithms** designed for **real-time execution on edge devices**. The design emphasizes **low latency**, **predictable behavior**, and **operational reliability**, rather than complex or opaque analytical models.

Anomaly detection is based on two complementary approaches: **threshold-based detection** and **trend-based detection**.
Threshold-based detection serves as the primary mechanism for identifying **acute physiological abnormalities**. For each incoming measurement, the most recent value in the **sliding window** is compared against **predefined medical thresholds** configured externally. When a measurement violates its allowed physiological range, an anomaly is generated **immediately**, enabling rapid identification of critical medical conditions.

To support **early detection of gradual deterioration**, the system also applies **trend-based anomaly detection**. Trend analysis is implemented using **linear regression slope computation** over a sliding window of recent measurements. The calculated slope represents the **direction and magnitude of change** in the monitored signal. Increasing trends are considered critical for measurements such as **heart rate** and **temperature**, while decreasing trends are considered critical for **SpO_2**. When the slope exceeds a **measurement-specific configurable threshold**, a trend anomaly is generated.

7.2. Sliding Window Management and Signal Processing

Physiological measurements are managed using **fixed-size sliding windows** maintained **per patient and per measurement type**. **Raw measurements are preserved unchanged** to ensure **data integrity** and **traceability**.

Signal preprocessing is applied **only to derived analytical data**, not to stored measurements. A **simple moving average** is used to generate **smoothed measurement windows**, reducing sensitivity to transient noise while maintaining low computational overhead. This design improves detection stability without altering original medical data.

7.3. Alert Debouncing and Reliability Mechanisms

To reduce **false alarms** and prevent **alert flooding**, the system implements **alert debouncing mechanisms**. Debounce rules are applied **per patient, measurement type, and anomaly type**.

If the same anomaly is detected within a configurable **debounce interval**, the alert is suppressed. This approach ensures that **persistent abnormal conditions** do not result in repeated alerts over short time intervals, thereby reducing **alert fatigue** and improving **system usability** in clinical environments.

7.4. Offline Operation and Configuration-Driven Behavior

The system supports **offline operation** to maintain continuous monitoring during **network disruptions**. During offline periods, measurements and alerts are stored locally in an **in-memory unified event queue**, preserving their **chronological order**. When connectivity is restored, cached events are **synchronized and flushed** in their original temporal sequence. The current implementation uses **in-memory storage (v1)** for offline caching, while persistent offline storage is considered **out of scope**.

Key system behaviors, including **medical thresholds**, **trend detection parameters**, **debounce intervals**, and **alert severity policies**, are defined through **external configuration files**. This **configuration-driven design** allows system behavior to be adapted without code changes, supporting **maintainability**, **flexibility**, and **future extensibility**.

8. Development environment description.

Such as programming language selection justification, cloud storage vs local storage and the likes.
The MedAlert system is implemented as a **modular Node.js application**, following the architecture and low-level design described in previous chapters. The implementation emphasizes separation of concerns, configurability, and support for edge-level execution.

8.1 Programming Language and Runtime

The system is implemented in **JavaScript (Node.js runtime)**. This choice enables rapid prototyping, modular development, and portability across different execution environments, including edge-capable devices.

8.2 Modular Code Structure

The EdgeProcessor is implemented as a coordinating component that orchestrates a set of dedicated sub-modules, each implemented as a separate module:

- Signal validation (signalValidator)
- Signal processing and sliding-window aggregation (signalProcessor)
- Anomaly detection (anomalyDetector)
- Alert generation and debouncing (alertService)
- Offline caching and synchronization (offlineCache)

This structure directly reflects the Low-Level Design decomposition.

8.3 Configuration-Driven Behavior

System behavior is controlled through external configuration files (e.g., medical thresholds, trend parameters, debounce intervals). This allows runtime tuning of detection logic without code changes.

8.4 Simulation and Testing Environment

A dedicated simulation component is used to inject synthetic measurement streams, simulate connectivity changes, and validate system behavior under both normal and degraded conditions.

9. Full validation report.

9.1. Validation Objectives

The objective of the validation phase is to verify that the MedAlert system behaves according to its defined functional and non-functional requirements under realistic operating conditions.
Validation focuses on correctness, reliability, availability, and robustness of the system, with particular emphasis on edge-level processing and real-time anomaly detection.

Two complementary validation approaches were applied:

1. **Unit-level validation**, used to verify the correctness of internal logic and module behavior.
2. **System-level (end-to-end) validation**, used to validate runtime behavior, operational flows, and non-functional requirements.

This combined approach ensures both implementation correctness and realistic system behavior.

9.2. Unit-Level Validation

Unit-level validation was performed using automated tests targeting the **EdgeProcessor** module.
These tests verify compliance with the Low-Level Design (LLD) and ensure that individual components behave correctly in isolation.

Validated Aspects

The unit tests validate the following behaviors:

- **Threshold-based anomaly detection**
  Verification that abnormal physiological values (e.g., high heart rate, low SpO_2, high temperature) generate the correct anomaly types.
- **Input validation and rejection**
  Verification that invalid measurements are rejected, including:
  - Low signal quality
  - Missing mandatory fields
  - Unsupported measurement types
  - Non-numeric or implausible values
  - Out-of-order timestamps
- **Sliding window management**
  Verification that sliding windows enforce the configured window size and discard older measurements.
- **Alert debouncing**
  Verification that repeated alerts of the same type are suppressed within the configured debounce interval.
- **Multi-patient isolation**
  Verification that measurements from different patients are handled independently without shared state.

- **Offline caching and recovery logic**
  Verification that alerts and measurements are cached during offline operation and correctly flushed when connectivity is restored.

Unit-level validation confirms that the EdgeProcessor implementation correctly follows the defined logic and internal design assumptions.

9.3. System-Level Validation

System-level validation was performed using an end-to-end simulation script that injects synthetic measurement streams into the system and observes runtime behavior. This approach validates the interaction between components and the system's behavior under realistic operating scenarios.

Each validation scenario is explicitly labeled, includes an expected outcome, and records the observed behavior through runtime logs.

9.3.1. Validated Scenarios

Test 1 – Normal Operation
Valid physiological measurements are processed successfully without generating alerts.
This scenario verifies baseline system behavior and absence of false positives.

Test 2 – Threshold-Based Anomaly Detection
Measurements exceeding configured medical thresholds generate immediate alerts.
This scenario validates real-time alert generation for acute medical conditions.

Test 3 – Alert Debouncing
Repeated abnormal measurements of the same type do not generate multiple alerts within a short time window.
This scenario validates alert flooding prevention and reliability mechanisms.

Test 4 – Offline Operation
The system continues processing measurements during network disconnection and caches alerts and measurements locally.
This scenario validates operational continuity under degraded connectivity.

Test 5 – Online Recovery and Cache Synchronization
Cached measurements and alerts are flushed and synchronized in chronological order once connectivity is restored.
This scenario validates correct recovery behavior.

Test 6 – Signal Quality Validation
Measurements with insufficient signal quality are rejected and do not enter the processing pipeline.
This scenario validates early noise filtering and false-alarm reduction.

Test 7 – Timestamp Consistency Validation
Measurements with out-of-order timestamps are rejected to preserve temporal correctness.
This scenario validates robustness against inconsistent sensor data.

Test 8 – Trend-Based Anomaly Detection
Gradual physiological deterioration triggers an alert based on trend analysis, even when no fixed threshold is violated.
This scenario validates early warning detection capabilities.

9.4. Validation of Non-Functional Requirements

The system-level validation scenarios address the following non-functional requirements:

- **Performance**
  Alerts are generated immediately upon anomaly detection, supporting low-latency response requirements.
- **Availability and Reliability**
  Offline operation and recovery scenarios demonstrate continuous system operation under network disruptions.
- **False Alarm Reduction**
  Signal quality validation and alert debouncing mechanisms reduce unnecessary alerts.
- **Scalability**
  Validation confirms independent handling of multiple patients without shared state.

Security and privacy requirements are addressed at the architectural and design level and are therefore not validated through runtime simulation in this project.

9.5. Validation Summary

The validation results demonstrate that the MedAlert system behaves as specified under both normal and degraded conditions.
Unit-level validation confirms correctness of internal logic and design compliance, while system-level validation confirms reliable runtime behavior and satisfaction of critical non-functional requirements.

Together, these validation activities provide strong evidence that the system is robust, reliable, and suitable for real-time edge-based medical monitoring.

10. Summary.

This project presented the design and implementation of **MedAlert**, a software-based medical monitoring system focused on real-time detection of physiological anomalies and reliable alerting in clinical environments. The system was designed using a structured software engineering approach, progressing from requirement analysis through architectural design, low-level design, and validation planning.

A key contribution of this project is the emphasis on **edge-level processing**, enabling early detection of abnormal conditions with low alert latency and continued operation during network disruptions. The system architecture clearly separates time-critical processing at the edge from centralized services responsible for data management, visualization, and system administration. This separation supports scalability, robustness, and maintainability.

The Low-Level Design focused on the **EdgeProcessor** module, detailing its internal sub-modules, data structures, processing flows, and error-handling strategies. This design directly supports the functional and non-functional requirements defined in the SRS, including reliability, offline operation, and reduction of false alerts. Validation planning and simulation-based testing demonstrate how the system behavior can be evaluated under both normal and degraded operating conditions.

While the current implementation provides a solid foundation, several directions for future work remain. These include integration with real medical devices, deployment on constrained edge hardware, extension of anomaly detection algorithms using adaptive or learning-based techniques, and enhanced security and privacy mechanisms. In addition, large-scale clinical evaluation would be required to assess system performance in real-world healthcare settings.

Overall, MedAlert demonstrates how systematic software engineering methods can be applied to the design of a safety-critical, real-time monitoring system, bridging the gap between conceptual design and practical implementation.

11. Documentation.

The MedAlert project is documented primarily through its **source code, configuration files, and validation artifacts**.
Documentation is embedded directly within the implementation using clear module headers, inline comments, descriptive naming conventions, and structured test scenarios.
This approach ensures that the system can be understood, executed, validated, and extended without reliance on extensive external manuals, which is appropriate for the scope of an academic software engineering project.

In addition to inline documentation, the hierarchical organization of the GitHub repository reflects the system architecture and serves as the main navigation and documentation mechanism for the project.

Repository Hierarchical Structure

The project repository is organized according to a clear separation of concerns, where each directory corresponds to a specific architectural responsibility:

- **src/**
  Main system implementation.
  - **modules/** – Core edge-processing logic (signal validation, processing, anomaly detection, alert management, offline caching).
  - **models/** – Data model definitions for measurements, anomalies, and alert events.
  - **repositories/** – Local persistence and history management components.
  - **utils/** – Shared utilities (time handling, logging, statistical helpers).
  - **ui/** – Lightweight demonstration dashboard for runtime visualization.
  - **edgeProcessor.js** – Central orchestration component of the edge system.
- **test/**
  Automated unit-level validation tests for internal logic verification.
- **config/**
  External configuration files defining thresholds, trend parameters, debounce intervals, and severity policies.
- **data/**
  Locally persisted measurement and alert data used for demonstration and validation purposes.

This hierarchical structure mirrors the architectural decomposition described in earlier chapters and enables intuitive understanding of system responsibilities and data flow.

The complete source code and validation artifacts are available in the project's GitHub repository: MidAlert_repo.