Politecnico di Torino

Computer Engineering

A.a. 2023/2024

Graduation Session November 27, 2025

A Blockchain-Enabled Web Application for Hospital Infrastructure Management Integrating Voice of Customer in Healthcare

Supervisors:
Valentina Gatteschi
Marco Domaneschi
Valentina Villa

Candidate:
Shayan Khalighi

Abstract

This thesis presents the design and implementation of a blockchain-enabled web application for hospital infrastructure management that addresses two critical challenges: the communication gap between visitors/patients and hospital staff, and the inefficiencies of paper-based task management systems.

Problem Statement and Motivation

Modern hospitals operate as complex organizations with extensive infrastructure requiring continuous maintenance and monitoring. The efficient management of maintenance requests and periodic safety inspections is critical for ensuring patient safety, operational continuity, and regulatory compliance. However, traditional approaches face significant challenges.

Communication Gap

One primary challenge is the lack of an effective communication channel between visitors, patients, and hospital staff. When visitors or patients encounter broken equipment, maintenance issues, or safety concerns, they often have no straightforward way to report these problems. This communication gap results in delayed reporting of critical issues, safety risks that go unnoticed, and frustration among visitors who cannot easily communicate problems.

Paper-Based Task Management

Many hospitals still rely on paper-based systems for managing periodic maintenance tasks and safety inspections. Technicians perform daily, weekly, or monthly checks and record findings on paper forms. At the end of each month, these forms are collected and manually processed, introducing significant delays: if a door breaks on the 4th of the month, the problem may not be discovered until the end of the month when forms are reviewed. This delay can result in extended equipment downtime, safety hazards that persist for weeks, and compliance issues.

Solution Overview

The system provides a comprehensive digital solution that integrates ticketing and task management capabilities:

- **Digital Ticketing Portal**: Enables visitors, patients, and staff to report maintenance issues through web interface and QR code scanning, with support for both authenticated and anonymous ticket creation, including image attachments. Authenticated users can track the status of their tickets, ensuring transparency.
- **Digital Task Management System**: Replaces paper-based periodic inspection workflows with a digital system supporting recurring tasks (daily, weekly, monthly, semestral) and one-time tasks, enabling real-time tracking and immediate problem identification.
- **Task-Ticket Integration**: When problems are detected during task inspections, tickets can be created immediately with pre-filled information, eliminating month-end delays.
- **Three-Level Hierarchy**: Implements a Department → Location → Equipment structure for precise equipment identification and efficient technician assignment.
- **Blockchain Integration**: Provides immutable audit trails through cryptographic hashing while preserving privacy by storing only hashes on-chain, not personally identifiable information.
- **Analytics and Reporting**: Provides administrators with insights into task completion rates, ticket resolution times, equipment maintenance patterns, and system performance metrics.

Implementation

The system was implemented as a full-stack web application using modern technologies and best practices.

Technology Stack

The application was built using:

- **Frontend**: Next.js 14.2.5 (Pages Router), React 18.3.1, TypeScript 5.4.5, Tailwind CSS, Radix UI components

- **Backend**: Next.js API routes, Node.js 18+, TypeScript
- **Database**: PostgreSQL with Prisma ORM 5.18.0 for type-safe database access
- **Authentication**: NextAuth.js 4.24.7 with JWT tokens and bcryptjs for password hashing
- **Blockchain**: Solidity 0.8.24 smart contracts, Foundry development framework, Anvil local node, Viem for JavaScript client integration
- **Email Service**: Resend API for transactional email delivery
- **QR Codes**: `qrcode` npm package for generation

Database Design

The database schema was defined using Prisma, providing type-safe database access. Key components include:

- **User Model**: Includes role (Admin, Technician, User), authentication fields, optional blockchain address, and hourly rate for technicians
- **Ticket Model**: Comprehensive ticket representation with status tracking, time metrics (startTime, endTime), and anonymous creator fields
- **Hierarchy Models**: Department, Location, and Element models with proper foreign key relationships and unique constraints
- **Task Models**: Task, TaskItemCompletion, and TaskPeriodCompletion models for periodic inspection management
- **Task-Ticket Integration**: TaskItemCompletion includes optional ticketId field linking NOT OK items to created tickets

Backend Implementation

Next.js API routes were organized following RESTful principles, with each route handler implementing authentication, authorization checks, request validation using Zod schemas, business logic execution, database operations via Prisma, and blockchain interactions where applicable.

The ticket creation process validates request payload, verifies hierarchy relationships, handles anonymous creation, generates canonical JSON representation for blockchain hashing, computes keccak256 hash, creates ticket record in database, and if blockchain enabled, calls smart contract's `createTicket()` function.

A technician recommendation algorithm retrieves ticket's associated element, finds all roles associated with that element, queries technicians whose customRole matches those roles, and sorts results with recommended technicians first.

Task Management Implementation

The task management system implements a flexible period-based architecture for recurring tasks:

- **Task Creation**: Supports different scope types (general, element, location, department) and recurrence patterns (daily, weekly, monthly, semestral)
- **Period Management**: Calculates periods on-demand based on recurrence patterns, supporting multiple independent periods (past, current, future)
- **Item Checking**: Technicians check items with OK/NOT OK status and optional notes, with unique constraint preventing duplicate checks per period
- **Task-Ticket Integration**: When item marked NOT OK, system redirects to ticket creation with pre-filled data (department, location, element, title, description)
- **Period Completion**: System validates all items checked before allowing completion, creates TaskPeriodCompletion record with timestamps
- **Access Control**: Technicians can only check current period, administrators can check past periods, future periods are read-only

QR Code Implementation

QR code functionality enables quick ticket creation:

- QR codes encode URLs with query parameters: `/tickets/new?departmentId=X&locationI`
- QR codes can be generated for departments, locations, or specific equipment
- When scanned, browser navigates to ticket creation page with pre-filled form
- Cascading dropdowns automatically populate based on pre-filled values

Email Notification System

Email notifications were implemented using Resend service:

- **Task Assignment**: Sent immediately when task assigned, includes task details and direct link
- **Ticket Assignment**: Sent when ticket assigned, includes ticket information and link
- **Problem Report**: Sent to task creator when NOT OK item reported
- **Daily Reminders**: Scheduled job runs weekdays at 9:00 AM, sends personalized reminders for incomplete/overdue tasks

Blockchain Integration

The TicketRegistry smart contract was developed using Foundry and tested with Anvil for local development. The contract implements a minimal on-chain storage approach, storing only cryptographic hashes rather than full ticket data, optimizing gas costs while providing verifiable audit trails.

Key operations include:

- `createTicket()`: Creates new ticket with hash, severity, department (85,000 gas)
- `assignTicket()`: Assigns ticket to technician address (45,000 gas)
- `updateStatus()`: Updates ticket status (35,000 gas)
- View functions (getTicket, exists, totalTickets): Consume no gas as read-only operations

At typical Ethereum mainnet gas prices (e.g., 30 gwei), the cost per ticket creation would be approximately €0.05-0.10 EUR, making the blockchain audit trail economically viable for hospital operations.

Frontend Implementation

The frontend was built with React and Next.js, implementing:

- Cascading dropdowns for hierarchy selection (Department → Location → Equipment)
- Real-time filtering based on selections

- Image upload with client-side compression (60-80% size reduction)
- Form validation and error handling
- Support for both authenticated and anonymous ticket creation
- Task management interface with item checklists and period tracking
- Administrative dashboard with analytics and reporting
- Real-time updates via 30-second polling for list pages

Results and Evaluation

Comprehensive testing was performed to ensure system reliability, functionality, and performance across all components.

Functional Testing Results

All system components were thoroughly tested:

- **Ticket Management**: Verified ticket creation (authenticated and anonymous), assignment, status updates, editing, deletion, search/filter functionality. All operations function correctly with proper authorization checks.
- **Hierarchy Management**: Tested department, location, and equipment operations, cascade deletes, unique constraints. Three-level hierarchy enables precise equipment identification and efficient filtering.
- **Task Management**: Verified task creation with different scope types and recurrence patterns, period management, item checking, task-ticket integration, period completion and submission. System successfully eliminates month-end delays by enabling immediate problem reporting.
- **QR Code Functionality**: Tested QR code generation for departments, locations, and elements, URL parsing from scanned codes, pre-filling functionality. QR codes successfully enable quick and accurate ticket creation for visitors.
- **Email Notifications**: Verified task assignment, ticket assignment, problem report, and daily reminder emails. All notifications delivered correctly with proper content and links.
- **Analytics**: Tested task completion rate calculations, ticket resolution time calculations, administrative dashboard displays. Analytics provide accurate insights into system performance.

- **Blockchain Integration**: Tested ticket creation, assignment, status updates on-chain, event parsing, and fallback mode when blockchain unavailable. System maintains full functionality even without blockchain, demonstrating practical deployment flexibility.

Performance Evaluation

Performance optimizations were implemented and evaluated:

- **Database Performance**: Indexes added on frequently queried fields (departmentId, locationId, elementId, createdById). Average API response times under 200ms for most operations. Pagination implemented for large ticket lists.

- **Frontend Performance**: Client-side image compression reduces upload size by 60-80%. Next.js automatic code splitting reduces initial bundle size. React memoization prevents unnecessary re-renders.

- **Blockchain Performance**: Minimal on-chain storage reduces transaction costs. Blockchain operations are asynchronous to avoid blocking API responses. Fallback mode ensures system remains functional when blockchain unavailable.

Security Testing

Security measures were verified:

- **Authentication**: Verified bcrypt password hashing, JWT token generation and expiration, session management. Password requirements enforced.

- **Authorization**: Tested role-based access control (admin, technician, user permissions), resource ownership checks, assignment authorization. All protected routes verified for proper authorization.

- **Data Privacy**: Blockchain stores only cryptographic hashes, not personally identifiable information. Anonymous ticket creation preserves privacy while enabling reporting.

- **SQL Injection Prevention**: Prisma ORM prevents SQL injection attacks through parameterized queries.

Key Achievements

The implementation successfully achieved all research objectives:

1. **Digital Ticketing Portal**: Successfully implemented accessible ticketing system supporting authenticated and anonymous creation, QR code scanning, and status tracking for authenticated users.

2. **Digital Task Management**: Developed comprehensive replacement for paper-based workflows, supporting recurring and one-time tasks with real-time tracking and immediate problem identification.

3. **Task-Ticket Integration**: Implemented seamless integration enabling immediate ticket creation from task inspections, eliminating month-end delays.

4. **Three-Level Hierarchy**: Successfully implemented Department → Location → Equipment structure enabling precise tracking and efficient assignment.

5. **Blockchain Integration**: Demonstrated practical application with immutable audit trails while preserving privacy through hash-based storage.

6. **Analytics and Reporting**: Developed comprehensive analytics providing insights into task completion, ticket resolution, and system performance.

Conclusion

This thesis successfully developed a comprehensive blockchain-enabled web application for hospital infrastructure management that addresses the communication gap between visitors/patients and hospital staff, and eliminates inefficiencies of paper-based task management systems. The system integrates modern web technologies with blockchain verification to provide a practical, scalable solution for healthcare infrastructure management.

The implementation demonstrates how blockchain technology can enhance traditional web applications by providing verifiable audit trails and data integrity guarantees in healthcare contexts, while solving practical problems of communication and workflow efficiency. The hybrid architecture combining traditional database storage with blockchain verification shows how modern applications can leverage blockchain benefits while maintaining performance.

The complete source code for this project is publicly available on GitHub at: `https://github.com/ShayanKh76/hospital-ticketing.git`.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Prof. Valentina Gatteschi, Prof. Marco Domaneschi, and Prof. Valentina Villa, for their invaluable guidance, support, and feedback throughout the development of this thesis. Their expertise and encouragement have been instrumental in shaping this work.

I am also grateful to the Politecnico di Torino for providing the resources and academic environment that made this research possible. The knowledge and skills acquired during my studies have been fundamental to the completion of this project.

Finally, I would like to thank my family and friends for their unwavering support and understanding during the long hours dedicated to this work.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Problem Statement

Modern hospitals operate as complex organizations with extensive infrastructure requiring continuous maintenance and monitoring. The efficient management of maintenance requests and periodic safety inspections is critical for ensuring patient safety, operational continuity, and regulatory compliance. However, traditional approaches to infrastructure management in hospitals face significant challenges that can lead to safety issues and equipment downtime.

1.1.1 Communication Gap Between Visitors and Hospital Staff

One of the primary challenges in hospital infrastructure management is the lack of an effective communication channel between visitors, patients, and hospital staff. When visitors or patients encounter broken equipment, maintenance issues, or safety concerns during their stay, they often have no straightforward way to report these problems. This communication gap results in:

- Delayed reporting of critical issues, leading to prolonged equipment downtime
- Safety risks that go unnoticed until discovered by staff during routine inspections
- Frustration among visitors and patients who cannot easily communicate problems
- Missed opportunities for preventive maintenance that could be identified by users

The absence of a digital ticketing portal means that maintenance issues may remain unreported for extended periods, potentially compromising patient safety and hospital operations.

1.1.2 Paper-Based Task Management System

Many hospitals still rely on paper-based systems for managing periodic maintenance tasks and safety inspections. In such systems, technicians (such as fire safety officers or preposti) perform daily, weekly, or monthly checks of equipment and record their findings on paper forms. These forms typically include:

- The date of inspection
- The status of each item checked (e.g., OK or NOT OK)
- Additional notes or observations
- Identification of the equipment or location inspected

At the end of each month, these paper forms are collected by administrative staff and manually processed. This workflow introduces significant delays: if a door breaks on the 4th of the month, the problem may not be discovered or addressed until the end of the month when the forms are reviewed. This delay can result in:

- Extended equipment downtime affecting patient care
- Safety hazards that persist for weeks before being addressed
- Compliance issues with regulatory requirements for timely maintenance
- Inefficient resource allocation due to lack of real-time visibility

Furthermore, paper-based systems lack the traceability, searchability, and audit capabilities required in modern healthcare environments where accountability and data integrity are paramount.

1.1.3 Need for Digital Transformation

The limitations of traditional approaches highlight the need for a comprehensive digital solution that:

- Provides an accessible ticketing portal for visitors, patients, and staff to report issues

- Digitizes the task management workflow, enabling real-time tracking and immediate problem identification
- Ensures data integrity and provides immutable audit trails for compliance purposes
- Supports the complex hierarchical organization of hospital infrastructure (departments, locations, equipment)
- Enables efficient assignment of maintenance tasks and tickets to appropriate technicians
- Provides administrators with real-time visibility into the status of all tasks and tickets, enabling better oversight and resource management
- Offers analytical capabilities and reporting features that provide insights into task completion rates, ticket resolution times, equipment maintenance patterns, and system performance metrics

1.2 Research Objectives

This thesis aims to design and implement a comprehensive web application that addresses the communication gap and paper-based workflow challenges in hospital infrastructure management. The main objectives are:

1. **Digital Ticketing Portal**: Develop an accessible system that allows visitors, patients, and staff to report maintenance issues and safety concerns, with support for both authenticated and anonymous ticket creation, including image attachments and QR code-based quick reporting. Authenticated users can track the status of their tickets through the system, ensuring transparency that their concerns are being addressed.
2. **Digital Task Management System**: Replace paper-based periodic inspection workflows with a digital system that enables real-time tracking of daily, weekly, monthly, and semestral maintenance tasks, with immediate problem identification and reporting capabilities.
3. **Three-Level Hierarchical Organization**: Implement a precise equipment identification system (Department → Location → Equipment) that reflects the complex structure of hospital infrastructure and enables accurate task and ticket assignment.

4. **Blockchain-Based Audit Trail**: Integrate blockchain technology to provide immutable audit trails for maintenance records, ensuring data integrity and compliance while preserving privacy by storing only cryptographic hashes on-chain.

5. **Automated Workflow Management**: Design systems for automatic ticket assignment, email notifications, and technician recommendations based on equipment specializations and roles.

6. **Real-Time Problem Resolution**: Enable immediate ticket creation from task inspections when problems are detected, eliminating the delays inherent in paper-based monthly reporting cycles.

1.3 Contributions

The main contributions of this work include:

- **Accessible Ticketing Portal**: A user-friendly system that bridges the communication gap between visitors, patients, and hospital staff, enabling quick issue reporting through web interface and QR code scanning, with support for both authenticated and anonymous submissions including image attachments. Authenticated users can track the status of their tickets through the system, ensuring transparency that their concerns are being addressed by hospital staff.

- **Digital Task Management System**: A comprehensive replacement for paper-based inspection workflows, supporting recurring tasks (daily, weekly, monthly, semestral) and one-time tasks, with real-time status tracking and immediate problem reporting capabilities that eliminate month-end delays.

- **Three-Level Hierarchical Organization**: Implementation of a Department $\rightarrow$ Location $\rightarrow$ Equipment structure that accurately reflects hospital infrastructure complexity, enabling precise task scoping and ticket assignment across different organizational levels.

- **Blockchain-Based Audit Trail**: Integration of blockchain technology to provide immutable, verifiable audit trails for maintenance records while preserving privacy through hash-based storage, ensuring compliance and data integrity in healthcare environments.

- **Automated Workflow Integration**: Seamless connection between task management and ticketing systems, where problems detected during inspections automatically generate tickets with pre-filled information, enabling immediate assignment and notification workflows.

- **Hybrid Architecture Design**: A full-stack web application that combines traditional database storage for performance with blockchain verification for immutability, demonstrating practical patterns for blockchain-enabled healthcare applications.
- **Real-Time Administrative Oversight**: Comprehensive dashboard and monitoring capabilities that enable administrators to view the real-time status of all tasks and tickets across the hospital, facilitating better resource allocation, priority management, and operational decision-making.
- **Analytics and Reporting**: Advanced analysis features that provide administrators with insights into task completion rates, ticket resolution patterns, equipment maintenance trends, and system performance metrics, enabling data-driven decision-making and continuous improvement of maintenance operations.

1.4 Thesis Structure

This thesis is organized as follows:

Chapter 2 presents the background and literature review, covering blockchain technology, healthcare information systems, Voice of Customer principles, and related work in infrastructure management systems.

Chapter 3 describes the system requirements, architectural design, database schema, and the three-level hierarchy model. This chapter also details the blockchain integration strategy and security considerations.

Chapter 4 provides a comprehensive description of the implementation, including the technology stack, frontend and backend architecture, API design, blockchain integration, and the user interface components.

Chapter 5 discusses the testing methodology, system evaluation, performance analysis, and user acceptance considerations.

Chapter 6 concludes the thesis with a summary of achievements, limitations, and suggestions for future work.

Chapter 2

Background and Literature Review

2.1 Blockchain Technology in Healthcare

Blockchain technology, originally developed as the underlying technology for cryptocurrencies, has emerged as a promising solution for various industries requiring data integrity, transparency, and immutability [1]. In healthcare, blockchain applications have focused on secure medical records management, supply chain tracking, and audit trail maintenance.

2.1.1 Blockchain Fundamentals

A blockchain is a distributed ledger that maintains a continuously growing list of records, called blocks, which are linked and secured using cryptography. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. This structure makes it computationally infeasible to modify data retroactively without altering all subsequent blocks, providing an immutable audit trail.

Key properties of blockchain technology relevant to healthcare applications include:

- **Immutability**: Once data is recorded, it cannot be altered without consensus from the network
- **Transparency**: All transactions are visible to network participants
- **Decentralization**: No single point of failure or control

- **Cryptographic Security**: Data integrity is ensured through cryptographic hashing

2.1.2 Blockchain Development Tools

Modern blockchain development has been facilitated by advanced toolkits that streamline smart contract development, testing, and deployment. Foundry is a comprehensive toolkit for Ethereum application development that provides fast compilation, testing, and deployment capabilities. Anvil, part of the Foundry suite, serves as a local Ethereum node for development and testing, similar to Ganache or Hardhat Network, enabling developers to test smart contracts locally before deploying to production networks.

For blockchain client interactions, libraries like Viem provide type-safe interfaces for Ethereum operations, supporting transaction signing, contract interactions, and event parsing. These tools have made blockchain integration more accessible for web applications, enabling practical implementations of blockchain-based audit trails in healthcare systems.

2.1.3 Blockchain in Healthcare Applications

Healthcare applications of blockchain have primarily focused on electronic health records (EHR), where blockchain can provide secure, interoperable, and patient-controlled access to medical data [2]. However, blockchain's immutability and audit trail capabilities are equally valuable for administrative and operational systems such as maintenance management.

Research has shown that blockchain can enhance healthcare data management by providing verifiable audit trails, ensuring data integrity, and enabling secure sharing between different healthcare providers [3]. However, challenges remain regarding scalability, privacy concerns, and regulatory compliance. For maintenance management systems, blockchain can provide immutable audit trails for compliance purposes while maintaining privacy by storing only cryptographic hashes rather than sensitive data on-chain.

2.2 Hospital Infrastructure Management

Hospital infrastructure management encompasses the maintenance, tracking, and optimization of physical assets including medical equipment, building systems, and IT infrastructure. Effective infrastructure management is critical for ensuring patient safety, operational efficiency, and cost control.

Traditional Computerized Maintenance Management Systems (CMMS) have been used in hospitals for decades to track maintenance activities, schedule preventive maintenance, and manage work orders [4]. However, these systems often lack the transparency, traceability, and integration capabilities required in modern healthcare environments.

Modern approaches to infrastructure management emphasize:

- Real-time visibility into equipment status and location
- Integration with other hospital information systems
- Data analytics for predictive maintenance
- Mobile accessibility for field technicians
- Compliance with healthcare regulations and standards

2.2.1 Periodic Task Management Systems

Periodic maintenance and inspection tasks are fundamental to hospital infrastructure management. These tasks include daily, weekly, monthly, and semestral inspections of safety equipment, medical devices, and building systems. Traditional approaches to managing these periodic tasks have relied on paper-based systems where technicians record inspection results on forms that are collected and processed at the end of each period.

Paper-based task management systems suffer from several limitations:

- **Delayed Problem Identification**: Issues discovered early in a period may not be addressed until the end of the period when forms are reviewed
- **Lack of Real-Time Visibility**: Administrators cannot monitor task completion or identify problems until forms are manually processed
- **Poor Traceability**: Paper records are difficult to search, archive, and audit
- **No Immediate Action**: Problems cannot trigger immediate ticket creation or technician assignment
- **Data Loss Risk**: Physical forms can be lost, damaged, or misplaced

Digital task management systems address these limitations by providing real-time tracking, immediate problem reporting, and seamless integration with ticketing systems. Modern digital solutions enable technicians to record inspection results immediately, with problems automatically generating maintenance tickets and notifications.

2.2.2 Ticketing and Issue Tracking Systems

Ticketing systems provide structured workflows for reporting, tracking, and resolving maintenance issues. In healthcare environments, effective ticketing systems must accommodate both internal staff and external stakeholders such as visitors and patients who may encounter problems during their hospital visits.

Key requirements for hospital ticketing systems include:

- **Accessibility**: Support for both authenticated users and anonymous submissions
- **Multi-Modal Reporting**: Web-based forms, mobile interfaces, and QR code-based quick reporting
- **Status Transparency**: Allow authenticated ticket submitters to track the status of their reports when logged in
- **Image Attachments**: Enable users to provide visual evidence of problems
- **Automated Workflows**: Automatic assignment, notifications, and escalation
- **Integration with Task Management**: Seamless connection between periodic inspections and issue reporting

Traditional ticketing systems often require user authentication, creating barriers for visitors and patients who need to report issues. Modern approaches emphasize accessibility and transparency, enabling anyone to report problems while providing visibility into resolution status.

2.2.3 QR Code-Based Reporting Systems

Quick Response (QR) codes have become a popular mechanism for enabling quick access to digital services. In the context of hospital maintenance, QR codes can be placed near equipment or in specific locations, allowing visitors and staff to quickly report issues by scanning the code and accessing a pre-configured reporting form.

QR code-based reporting offers several advantages:

- **Rapid Access**: Eliminates the need to navigate complex websites or remember URLs
- **Context Preservation**: QR codes can be linked to specific locations, departments, or equipment, pre-filling relevant information
- **Reduced Errors**: Pre-configured forms reduce the chance of incorrect location or equipment selection

- **Universal Compatibility**: QR codes can be scanned by any modern smartphone

This approach is particularly valuable in hospital settings where visitors and patients may not be familiar with the organizational structure and need guidance in accurately reporting issues.

2.3 Voice of Customer in Healthcare

Voice of Customer (VoC) is a research methodology that captures customers' expectations, preferences, and aversions. In healthcare, VoC principles can be applied to improve patient satisfaction, staff experience, and service delivery quality [5].

For hospital infrastructure management systems, VoC integration extends beyond medical staff to include visitors and patients who interact with hospital facilities. This broader perspective recognizes that anyone using hospital facilities can identify maintenance issues and should have the ability to report them.

Key VoC principles for infrastructure management systems include:

- **Accessibility**: Empowering all stakeholders (visitors, patients, staff) to easily report issues through multiple channels
- **Transparency**: Providing clear visibility into the status of reported issues, ensuring users know their concerns are being addressed
- **Responsiveness**: Ensuring timely response and resolution of reported problems
- **Feedback Loops**: Creating mechanisms for users to track their reports and receive updates on resolution progress
- **Continuous Improvement**: Using reported data and user feedback to improve processes and prevent recurring issues

Digital platforms that incorporate VoC principles have shown improved user satisfaction and operational efficiency in healthcare settings [6]. The ability for users to report issues easily, track their resolution status in real-time, and receive updates creates a feedback loop that enhances service delivery and builds trust between the hospital and its stakeholders.

Transparency is particularly important in VoC implementation: when visitors or patients report an issue, they should be able to see that the hospital is actively working on it. This visibility not only improves user satisfaction but also encourages more reporting, leading to better maintenance coverage and earlier problem detection.

2.4 Real-Time Monitoring and Analytics

Real-time monitoring and analytics capabilities are essential for effective hospital infrastructure management. Administrators need immediate visibility into the status of maintenance tasks, ticket resolution progress, and system performance metrics to make informed decisions about resource allocation and priority management.

2.4.1 Real-Time Status Monitoring

Real-time status monitoring enables administrators to:

- Track the current state of all tasks and tickets across the hospital
- Identify bottlenecks and delays in maintenance workflows
- Monitor technician workload and availability
- Detect patterns in equipment failures or maintenance needs
- Respond quickly to critical issues or emergencies

Unlike paper-based systems where status information is only available after manual processing, digital systems provide continuous visibility, enabling proactive management and faster response times.

2.4.2 Analytics and Reporting

Analytics capabilities transform raw maintenance data into actionable insights:

- **Task Completion Rates**: Track completion rates for periodic inspections across different departments and time periods
- **Ticket Resolution Metrics**: Analyze average resolution times, identify recurring problems, and measure technician performance
- **Equipment Maintenance Patterns**: Identify equipment that requires frequent maintenance or replacement
- **Resource Utilization**: Monitor technician workload and optimize assignment strategies
- **Compliance Reporting**: Generate reports for regulatory compliance and audit purposes

These analytics enable data-driven decision-making, helping administrators optimize maintenance schedules, allocate resources more effectively, and identify areas for process improvement.

2.5 Digital Transformation in Hospital Operations

The shift from paper-based to digital systems represents a fundamental transformation in how hospitals manage their operations. Digital transformation in hospital infrastructure management involves:

2.5.1 Workflow Digitization

Replacing manual, paper-based processes with digital workflows that:

- Eliminate delays between problem identification and resolution
- Enable immediate data capture and processing
- Provide automated notifications and escalations
- Support mobile access for field technicians
- Integrate multiple systems and stakeholders

2.5.2 Data Integration and Connectivity

Digital systems enable seamless integration between:

- Task management and ticketing systems
- Equipment databases and maintenance records
- User management and access control
- Analytics and reporting tools
- External systems and services

This integration creates a unified view of hospital infrastructure, enabling comprehensive management and analysis.

2.5.3 Stakeholder Engagement

Digital transformation enhances engagement with all stakeholders:

- Visitors and patients can easily report issues and track resolution
- Technicians receive immediate task assignments and notifications

- Administrators have real-time visibility and analytics
- Management can access comprehensive reports and insights

This multi-stakeholder engagement creates a collaborative environment where everyone contributes to maintaining hospital infrastructure effectively.

2.6 Related Work

Several systems have been developed for hospital maintenance management, each with different approaches and capabilities.

2.6.1 Traditional CMMS Systems

Commercial CMMS solutions like Maximo, SAP Plant Maintenance, and eMaint provide comprehensive asset management capabilities. However, these systems are often complex, expensive, and designed for industrial settings rather than healthcare-specific needs [4]. They typically lack modern web interfaces, blockchain integration, and the hierarchical precision required for hospital equipment tracking.

2.6.2 Web-Based Maintenance Systems

Web-based maintenance management systems have gained popularity due to their accessibility and ease of use. Systems like Fiix, Hippo CMMS, and UpKeep provide cloud-based solutions with mobile apps for technicians. However, these general-purpose systems do not address healthcare-specific requirements such as:

- Integration with hospital information systems
- Healthcare regulatory compliance
- Equipment specialization tracking
- Blockchain-based audit trails

2.6.3 Blockchain-Based Healthcare Systems

Several research projects have explored blockchain applications in healthcare, but most focus on clinical data management rather than infrastructure maintenance. Systems like MedRec [2] and Gem Health Network have demonstrated blockchain's potential for secure data sharing and audit trails in clinical contexts.

However, the application of blockchain to administrative systems like maintenance management has received less attention, despite the clear benefits of immutable audit trails for compliance and accountability.

2.7 Research Gap

The existing literature reveals significant gaps in addressing the two primary challenges identified in hospital infrastructure management:

2.7.1 Communication Gap Between Stakeholders

While various ticketing and issue tracking systems exist, there is a lack of comprehensive solutions that:

- Enable visitors and patients to easily report maintenance issues without requiring authentication
- Provide transparency by allowing authenticated ticket submitters to track the status of their reports when logged in
- Support multiple reporting channels including QR code-based quick access
- Integrate seamlessly with internal task management workflows
- Maintain immutable audit trails for compliance purposes

Most existing systems either require authentication (creating barriers for visitors) or lack the transparency and integration capabilities needed for effective hospital maintenance management.

2.7.2 Paper-Based Task Management Limitations

Traditional periodic inspection systems rely on paper-based workflows that introduce significant delays. There is a gap in solutions that:

- Digitize periodic task management (daily, weekly, monthly, semestral) with real-time tracking
- Enable immediate problem identification and ticket creation from task inspections
- Provide administrators with real-time visibility into task completion status
- Eliminate month-end delays by enabling immediate reporting and action
- Integrate task management with ticketing systems for seamless workflow

2.7.3 Comprehensive Integration Gap

The existing literature reveals a gap in systems that comprehensively combine:

1. Accessible ticketing portal for visitors, patients, and staff with status transparency
2. Digital task management system replacing paper-based periodic inspections
3. Precise hierarchical equipment organization (three levels: Department, Location, Equipment)
4. Blockchain-based audit trails for maintenance records ensuring data integrity
5. Real-time monitoring and analytics for administrative oversight
6. Voice of Customer integration emphasizing transparency and stakeholder engagement
7. Seamless integration between task management and ticketing systems
8. QR code-based reporting for quick and accurate issue reporting

This thesis addresses these gaps by developing a comprehensive solution that integrates all these elements into a cohesive hospital infrastructure management system. The solution specifically addresses the communication gap between visitors/patients and hospital staff, eliminates the delays inherent in paper-based task management, and provides the transparency, real-time visibility, and analytics capabilities required for modern hospital operations.

Chapter 3

System Requirements and Design

3.1 System Requirements

3.1.1 Functional Requirements

The system must fulfill the following functional requirements:

Ticketing System Requirements

1. **User Management**: Support multiple user types (Administrators, Technicians, Users) with role-based access control

2. **Ticket Creation**: Allow authenticated and anonymous users to create maintenance tickets with detailed information including title, description, severity, location, equipment selection, and image attachments

3. **Anonymous Ticket Creation**: Enable visitors and patients to create tickets without authentication, with optional contact information (email, name) for follow-up

4. **QR Code-Based Reporting**: Generate QR codes for departments/locations that, when scanned, pre-fill ticket creation forms with the correct hierarchy (department, location, equipment)

5. **Status Tracking for Authenticated Users**: Allow authenticated ticket submitters to view the status of their tickets when logged in, ensuring transparency that their concerns are being addressed

6. **Equipment Hierarchy**: Maintain a three-level hierarchy (Department → Location → Equipment) for precise equipment tracking and ticket creation
7. **Ticket Assignment**: Enable administrators to assign tickets to specialized technicians based on equipment types and technician roles
8. **Status Tracking**: Track ticket lifecycle through multiple states (Open, InProgress, Resolved, Closed, Canceled) with automatic time tracking (startTime, endTime)
9. **Search and Filtering**: Provide search and filter capabilities for tickets based on status, severity, department, location, date range, and other criteria
10. **Email Notifications**: Automatically send email notifications when tickets are assigned, status changes, or important updates occur
11. **Blockchain Integration**: Store ticket hashes on blockchain for immutable audit trails while preserving privacy
12. **Technician Recommendations**: Automatically recommend technicians based on equipment specializations and role assignments

Task Management System Requirements

1. **Task Creation**: Enable administrators to create maintenance and inspection tasks with flexible scoping (general, element-level, location-level, department-level)
2. **Recurring Tasks**: Support recurring tasks with patterns: daily, weekly, monthly, and semestral (twice per year)
3. **One-Time Tasks**: Support non-recurring tasks for ad-hoc maintenance or inspections
4. **Task Assignment**: Assign tasks to technicians with automatic email notifications
5. **Period Management**: For recurring tasks, track multiple periods independently (past, current, future periods)
6. **Item Checking**: Enable technicians to mark items as OK or NOT OK during task execution, with optional notes
7. **Immediate Ticket Creation**: When an item is marked NOT OK, allow immediate ticket creation with pre-filled information from the task

8. **Period Completion**: Track when periods are completed, recording who completed it and when
9. **Period Submission**: Support submission of completed periods to services (e.g., SS Prevenzione, SC Tecnico), tracking submission separately from completion
10. **Real-Time Status Visibility**: Provide administrators with real-time visibility into task completion status across all tasks and periods
11. **Task History**: Maintain complete history of item completions, period completions, and associated tickets
12. **Access Control for Periods**: Restrict item checking to current periods for technicians, while allowing administrators to check past periods for corrections

Analytics and Reporting Requirements

1. **Task Analytics**: Provide analytics on task completion rates, period completion status, and non-conformity patterns
2. **Ticket Analytics**: Analyze ticket resolution times, ticket volume by department/location, and recurring problem identification
3. **Performance Metrics**: Track technician performance, equipment maintenance patterns, and resource utilization
4. **Administrative Dashboards**: Provide real-time dashboards showing status of all tasks and tickets across the hospital
5. **Compliance Reporting**: Generate reports for regulatory compliance and audit purposes

3.1.2 Non-Functional Requirements

- **Security**: Implement secure authentication, password hashing, and role-based authorization
- **Privacy**: Ensure patient data privacy by storing only hashes on-chain, not PII
- **Performance**: Support concurrent users with acceptable response times
- **Usability**: Provide intuitive user interface accessible to non-technical medical staff

- **Scalability**: Design architecture to accommodate growth in users and tickets
- **Reliability**: Ensure system availability and data consistency

3.2 System Architecture

3.2.1 Architectural Overview

The system follows a three-tier architecture pattern:

1. **Presentation Layer**: Next.js React-based frontend providing user interface
2. **Application Layer**: Next.js API routes handling business logic and request processing
3. **Data Layer**: PostgreSQL database for persistent storage and blockchain for audit trails

The system architecture consists of three main layers: the presentation layer (Next.js React frontend), the application layer (Next.js API routes), and the data layer (PostgreSQL database and blockchain).

Figure 3.1 illustrates the overall system architecture, showing the three-tier structure and the flow of data between layers.

3.2.2 Technology Stack

Frontend:

- Next.js 14.2.5 (Pages Router) for server-side rendering and routing
- React 18.3.1 for UI components
- TypeScript for type safety
- Tailwind CSS for styling
- Radix UI for accessible component primitives

Backend:

- Next.js API Routes for RESTful endpoints
- Prisma ORM for database access
- NextAuth.js for authentication

- Zod for data validation

Database:

- PostgreSQL for relational data storage
- Prisma Migrate for schema management

Blockchain:

- Solidity 0.8.24 for smart contracts
- Foundry for development and testing
- Viem for Ethereum client interactions
- OpenZeppelin Contracts for security standards

3.3 Database Design

3.3.1 Entity Relationship Model

Figure 3.2 shows the complete database schema with all entities and their relationships.

The database schema implements the following core entities:

User and Authentication Entities

- **User**: Stores user accounts with roles (admin, technician, user), authentication information, optional blockchain addresses, and hourly rates for technicians
- **Role**: Custom roles defining technician specializations (e.g., "Medical Equipment Technician", "Radiology Specialist")
- **RoleElement**: Many-to-many relationship between roles and equipment, enabling equipment specialization tracking

Hierarchy Entities

- **Department**: Top-level organizational units (e.g., Emergency, Radiology, ICU)
- **Location**: Physical spaces within departments (e.g., Room 204, Operating Theater 3, Ward A)
- **Element**: Equipment items located in specific locations (e.g., Defibrillator, MRI Scanner, Ventilator)

Ticketing Entities

- **Ticket**: Represents maintenance requests with full lifecycle tracking, including status, severity, assignments, timestamps, and optional anonymous creator information
- **Ticket Relationships**: Tickets reference users (creator, assignee), department, location, and element through foreign keys

Task Management Entities

- **Task**: Represents maintenance or inspection tasks with flexible scoping (general, element-level, location-level, department-level), recurrence patterns, and assignment information
- **TaskItemCompletion**: Records individual item checks within tasks, storing status (OK/NOT OK), notes, checker information, period date (for recurring tasks), and optional linked ticket ID
- **TaskPeriodCompletion**: Tracks period completion for recurring tasks, recording completion and submission timestamps and user information
- **Task Relationships**: Tasks reference users (creator, assignee), department, location, and element through foreign keys, with JSON arrays for multi-location/element scoping

3.3.2 Three-Level Hierarchy

The hierarchical structure enables precise equipment identification:

Level 1 - Department: Represents organizational divisions within the hospital. Each department can contain multiple locations. Examples include:

- Emergency Department
- Intensive Care Unit (ICU)
- Radiology Department
- Laboratory Services

Level 2 - Location: Represents physical spaces within departments. Each location belongs to exactly one department and can contain multiple equipment items. Examples include:

- Room 204 (Emergency Department)

- Operating Theater 3 (Surgery Department)
- Ward A (General Medicine)

Level 3 - Equipment (Element): Represents specific medical devices or equipment. Each equipment item belongs to exactly one location. Examples include:

- Defibrillator (Room 204, Emergency Department)
- MRI Scanner (Imaging Suite, Radiology Department)
- Ventilator (ICU Room 5, Intensive Care Unit)

This three-level hierarchy provides several advantages:

- Precise equipment identification for accurate ticket creation
- Efficient technician assignment based on location and equipment type
- Better cost tracking and analytics at each organizational level
- Support for location-based filtering and reporting

3.3.3 Key Relationships

Hierarchy Relationships

- **Department-Location**: One-to-many relationship with cascade delete (deleting a department deletes all its locations)
- **Location-Element**: One-to-many relationship with cascade delete (deleting a location deletes all its equipment)
- **Unique Constraints**: Location names are unique within a department; Element names are unique within a location

User and Role Relationships

- **User-Role**: Many-to-one relationship (users can have a custom role for specialization)
- **Role-Element**: Many-to-many relationship via RoleElement join table, enabling equipment specialization tracking
- **User-Ticket**: Users can create multiple tickets and be assigned to multiple tickets (as technicians)

Ticket Relationships

- **Ticket-User**: Tickets reference creator (createdBy) and assignee (assignedTo) users

- **Ticket-Hierarchy**: Tickets reference department, location, and element through foreign keys (normalized references) while maintaining legacy string fields for backward compatibility

- **Ticket Status Tracking**: Tickets track lifecycle with status enum, timestamps (startTime, endTime), and soft delete capability

Task Relationships

- **Task-User**: Tasks reference creator (createdBy) and assignee (assignedTo) users

- **Task-Hierarchy**: Tasks can reference department, location, and/or element depending on scope type, with JSON arrays for multi-location/element scoping

- **TaskItemCompletion-Task**: One-to-many relationship (tasks have multiple item completions)

- **TaskItemCompletion-Element**: Item completions reference specific elements being checked

- **TaskItemCompletion-User**: Records who checked each item

- **TaskItemCompletion-Ticket**: Optional link to tickets created from NOT OK items

- **TaskPeriodCompletion-Task**: One-to-many relationship (recurring tasks have multiple period completions)

- **TaskPeriodCompletion-User**: Records who completed and submitted each period

- **Unique Constraints**: TaskItemCompletion has unique constraint on (taskId, elementId, periodDate) to prevent duplicate checks; TaskPeriodCompletion has unique constraint on (taskId, periodDate)

3.4 Blockchain Integration Design

3.4.1 Smart Contract Architecture

The TicketRegistry smart contract implements a minimal on-chain representation of tickets:

- **Storage**: Only stores cryptographic hashes (keccak256) of ticket canonical JSON, not full data
- **Privacy**: No personally identifiable information (PII) stored on-chain
- **Immutability**: Once recorded, ticket hashes cannot be altered
- **Audit Trail**: All operations emit events for off-chain indexing

3.4.2 Canonical JSON Generation

To ensure consistent hashing, ticket data is serialized in a canonical format:

1. All object keys are sorted alphabetically
2. Consistent stringification ensures deterministic output
3. Includes: title, description, location, severity, department, departmentId, locationId, elementId, attachments, timestamps, creator information

The hash of this canonical JSON is computed using keccak256 and stored on-chain, providing a verifiable fingerprint of the ticket data.

3.4.3 Blockchain Operations

The system performs the following blockchain operations:

- **Ticket Creation**: Backend relayer calls `createTicket()` with hash, severity, and department string
- **Ticket Assignment**: `assignTicket()` links tickets to technician Ethereum addresses
- **Status Updates**: `updateStatus()` records status changes on-chain
- **Query Operations**: `getTicket()` retrieves on-chain ticket data for verification

3.4.4 Blockchain Fallback Mechanism

The system includes a fallback mechanism for development and deployment flexibility:

- **Mock Mode**: When blockchain environment variables are not configured, the system operates in mock mode with console logging instead of actual blockchain transactions
- **Graceful Degradation**: System remains fully functional without blockchain, enabling development and testing without blockchain infrastructure
- **Configuration-Based**: Blockchain integration is enabled/disabled based on environment variable presence (RPC_URL, CONTRACT_ADDRESS, SERVER_PRIVATE_KEY)

This design ensures the system can be deployed in various environments, from local development (using Anvil) to production networks, with or without blockchain integration.

3.5 Task-Ticket Integration Design

A key design feature is the seamless integration between task management and ticketing systems, enabling immediate problem reporting from periodic inspections.

3.5.1 Integration Workflow

1. **Task Execution**: Technician performs periodic inspection, checking items as OK or NOT OK
2. **Problem Detection**: When item is marked NOT OK, system offers immediate ticket creation option
3. **Automatic Pre-filling**: Ticket creation form is pre-filled with:
    - Department, location, and element from task
    - Task description and item notes as ticket description
    - Default severity level (medium)
    - Link back to originating task item
4. **Ticket Creation**: Technician completes ticket creation, ticket is automatically linked to task item via `ticketId` field
5. **Status Tracking**: Both task item and ticket maintain their status independently, with cross-references for traceability

3.5.2 Benefits of Integration

- **Eliminates Delays**: Problems identified during inspections immediately become actionable tickets
- **Reduces Data Entry**: Pre-filled information reduces errors and saves time
- **Maintains Traceability**: Link between task items and tickets enables complete audit trail
- **Streamlines Workflow**: Single interface for both inspection and problem reporting

3.6 Security Design

3.6.1 Authentication

- Password-based authentication using bcrypt hashing
- JWT tokens for session management
- Secure password requirements (minimum 6 characters)

3.6.2 Authorization

Role-based access control (RBAC) with three primary roles:

- **Administrator**: Full system access, user management, system configuration
- **Technician**: View assigned tickets, update ticket status, access role-specific equipment
- **User**: Create tickets, view own tickets, limited read access

3.6.3 Data Privacy

- Patient data never stored on blockchain
- Only cryptographic hashes on-chain
- PII stored securely in encrypted database
- Anonymous ticket creation supported with optional contact information (email, name) for follow-up only
- Anonymous ticket submitters provide optional contact information for follow-up communications

3.7 Email Notification Design

The system implements automated email notifications to keep stakeholders informed about task assignments, ticket updates, and important events.

3.7.1 Notification Types

- **Task Assignment**: Sent to technician when task is assigned, includes task details and link to task page
- **Ticket Assignment**: Sent to technician when ticket is assigned, includes ticket details and link to ticket page
- **Problem Report**: Sent to task creator when problem is reported during task execution, includes problem details and links to task and ticket
- **Daily Reminders**: Sent to technicians on weekdays at 9:00 AM for incomplete or overdue tasks, with overdue tasks highlighted

3.7.2 Email Design Principles

- **Hospital-Friendly Design**: Professional, clean email templates suitable for hospital environment
- **Actionable Links**: Direct links to relevant pages (tasks, tickets) for quick access
- **Clear Information**: Concise summaries of relevant information (task/ticket details, status, deadlines)
- **Personalization**: Personalized per recipient, showing only their assigned tasks/tickets

3.8 API Design

3.8.1 RESTful Architecture

The system provides RESTful API endpoints organized by resource:

Ticketing Endpoints

- `/api/tickets`: List tickets with filtering and search
- `/api/tickets/create`: Create new ticket (supports authenticated and anonymous)
- `/api/tickets/[id]`: Get ticket details
- `/api/tickets/[id]/edit`: Edit ticket (owner/admin only)
- `/api/tickets/[id]/status`: Update ticket status
- `/api/tickets/[id]/assign`: Assign ticket to technician
- `/api/tickets/[id]/close`: Close ticket
- `/api/tickets/[id]/delete`: Soft delete ticket
- `/api/tickets/[id]/audit`: Get blockchain audit trail
- `/api/tickets/[id]/related`: Get related tickets (same department/location)
- `/api/tickets/suggest-technicians`: Get technician recommendations

Hierarchy Management Endpoints

- `/api/departments`: List/create departments
- `/api/departments/[id]/update`: Update department
- `/api/departments/[id]/toggle`: Toggle visibility
- `/api/departments/[id]/delete`: Delete department
- `/api/locations`: List/create locations (filtered by department)
- `/api/locations/[id]/update`: Update location
- `/api/locations/[id]/toggle`: Toggle visibility
- `/api/locations/[id]/delete`: Delete location
- `/api/elements`: List/create elements (filtered by location)
- `/api/elements/[id]/update`: Update element
- `/api/elements/[id]/toggle`: Toggle visibility
- `/api/elements/[id]/delete`: Delete element
- `/api/elements/[id]/roles`: Get roles assigned to element

Task Management Endpoints

- `/api/tasks`: List tasks with filtering
- `/api/tasks/create`: Create new task
- `/api/tasks/[id]`: Get task details
- `/api/tasks/[id]/edit`: Edit task
- `/api/tasks/[id]/delete`: Delete task
- `/api/tasks/[id]/duplicate`: Duplicate task
- `/api/tasks/[id]/items`: Get task items (elements to check)
- `/api/tasks/[id]/items/check`: Mark item as OK/NOT OK
- `/api/tasks/[id]/items/batch-ok`: Mark all items as OK and complete
- `/api/tasks/[id]/periods`: Get period information for recurring tasks
- `/api/tasks/[id]/periods/complete`: Complete current period
- `/api/tasks/[id]/periods/submit`: Submit completed period to services
- `/api/tasks/[id]/reopen`: Reopen completed task/period
- `/api/tasks/history`: Get task completion history

Administrative Endpoints

- `/api/admin/roles`: Manage custom roles
- `/api/admin/roles/[id]/elements`: Assign roles to elements
- `/api/admin/technicians/pricing`: Get technicians with pricing
- `/api/admin/cost-analysis`: Get cost analysis data
- `/api/admin/system-analysis`: Get system metrics
- `/api/admin/task-analysis`: Get task analytics and metrics

QR Code and Utility Endpoints

- `/api/qr/generate`: Generate QR code for department/location/element
- `/api/qr/scan`: Process QR code scan and return pre-filled ticket data

Authentication Endpoints

- `/api/auth/[...nextauth]`: NextAuth.js authentication
- `/api/auth/register`: User registration
- `/api/me`: Get current user
- `/api/me/password`: Change password

3.8.2 Request/Response Format

- JSON request/response format
- Consistent error response structure: `{ok: boolean, error?: string}`
- Validation using Zod schemas
- HTTP status codes following REST conventions

3.9 User Interface Design

3.9.1 Design Principles

- **Usability**: Intuitive interface accessible to non-technical users including visitors, patients, and hospital staff
- **Accessibility**: WCAG-compliant components using Radix UI, ensuring accessibility for all users
- **Responsiveness**: Mobile-friendly design with Tailwind CSS, enabling use on smartphones and tablets
- **Internationalization**: Support for multiple languages (English/Italian) with language switcher
- **Transparency**: Clear status visibility for ticket submitters, ensuring users know their concerns are being addressed
- **Real-Time Updates**: Immediate feedback on actions, status changes visible without page refresh where possible

3.9.2 QR Code Design

QR codes serve as a quick access mechanism for ticket creation, particularly valuable for visitors and patients who may not be familiar with the hospital's organizational structure.

QR Code Generation

- QR codes can be generated for departments, locations, or specific equipment
- Each QR code encodes a URL with query parameters specifying the hierarchy (departmentId, locationId, elementId)
- QR codes are displayed as downloadable images that can be printed and placed near equipment
- Codes are static and do not expire, enabling long-term placement

QR Code Scanning Flow

1. User scans QR code with smartphone camera
2. Browser opens ticket creation page with pre-filled hierarchy information
3. User completes remaining form fields (title, description, severity, optional images)
4. User submits ticket, receiving ticket ID for status tracking

This design eliminates navigation complexity and reduces errors in location/equipment selection, making it particularly valuable for visitors and patients unfamiliar with hospital structure.

3.9.3 Key User Flows

Ticketing Flows

1. **Standard Ticket Creation**: User navigates to ticket creation page, selects department → location → equipment (cascading dropdowns), fills form with title, description, severity, optional images, submits
2. **QR Code Ticket Creation**: Visitor/patient scans QR code placed near equipment, QR code pre-fills department/location/element, user completes form and submits
3. **Anonymous Ticket Creation**: Visitor/patient creates ticket without login, provides optional email/name for follow-up
4. **Status Tracking**: Authenticated users can view status of their tickets when logged in, seeing updates in real-time

5. **Ticket Assignment**: Admin views ticket, sees technician recommendations based on equipment specialization, selects technician, assigns (email notification sent)

6. **Status Update**: Technician updates ticket status (Open → InProgress → Resolved → Closed), system automatically tracks startTime and endTime

Task Management Flows

1. **Task Creation**: Admin creates task, selects scope type (general/element/location/department), configures recurrence pattern if needed, assigns to technician (email notification sent)

2. **Task Execution**: Technician views assigned tasks, opens task detail page, sees list of items to check, marks each item as OK or NOT OK with optional notes

3. **Immediate Problem Reporting**: When item marked NOT OK, technician can immediately create ticket with pre-filled information (department, location, element, description)

4. **Period Completion**: For recurring tasks, technician completes all items for current period, system marks period as complete, records completion timestamp

5. **Period Submission**: After period completion, technician submits period to services (e.g., SS Prevenzione, SC Tecnico), system records submission separately

6. **Task History Review**: Technician or admin views period history, sees completion status for all periods, can review past item completions and associated tickets

Administrative Flows

1. **Real-Time Monitoring**: Admin views dashboard showing real-time status of all tasks and tickets across hospital

2. **Analytics Review**: Admin accesses analytics pages to view task completion rates, ticket resolution metrics, equipment maintenance patterns, and performance data

3. **QR Code Generation**: Admin generates QR codes for departments/locations, prints and places near equipment for quick visitor reporting

4. **Hierarchy Management**: Admin manages departments, locations, and equipment through catalog interface with search, create, update, delete, and visibility toggle capabilities

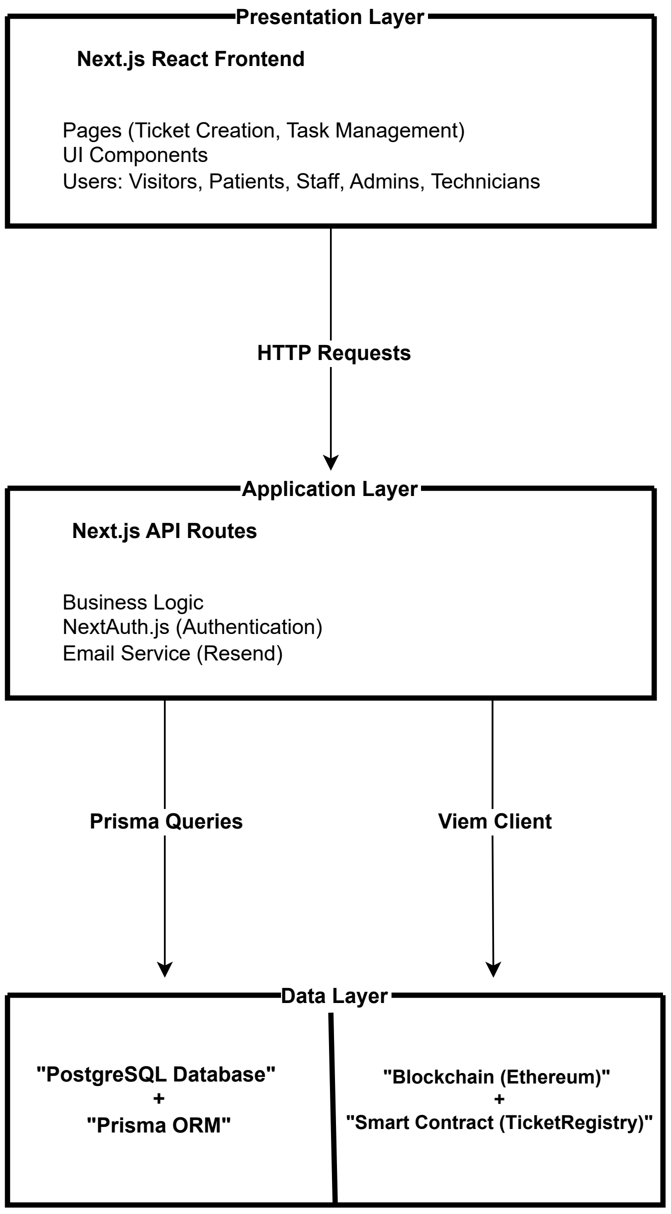

Presentation Layer
Next.js React Frontend
Pages (Ticket Creation, Task Management)
UI Components
Users: Visitors, Patients, Staff, Admins, Technicians
HTTP Requests
Application Layer
Next.js API Routes
Business Logic
NextAuth.js (Authentication)
Email Service (Resend)
Prisma Queries
Viem Client
Data Layer
"PostgreSQL Database" + "Prisma ORM"
"Blockchain (Ethereum)" + "Smart Contract (TicketRegistry)"

Figure 3.1: System Architecture Overview

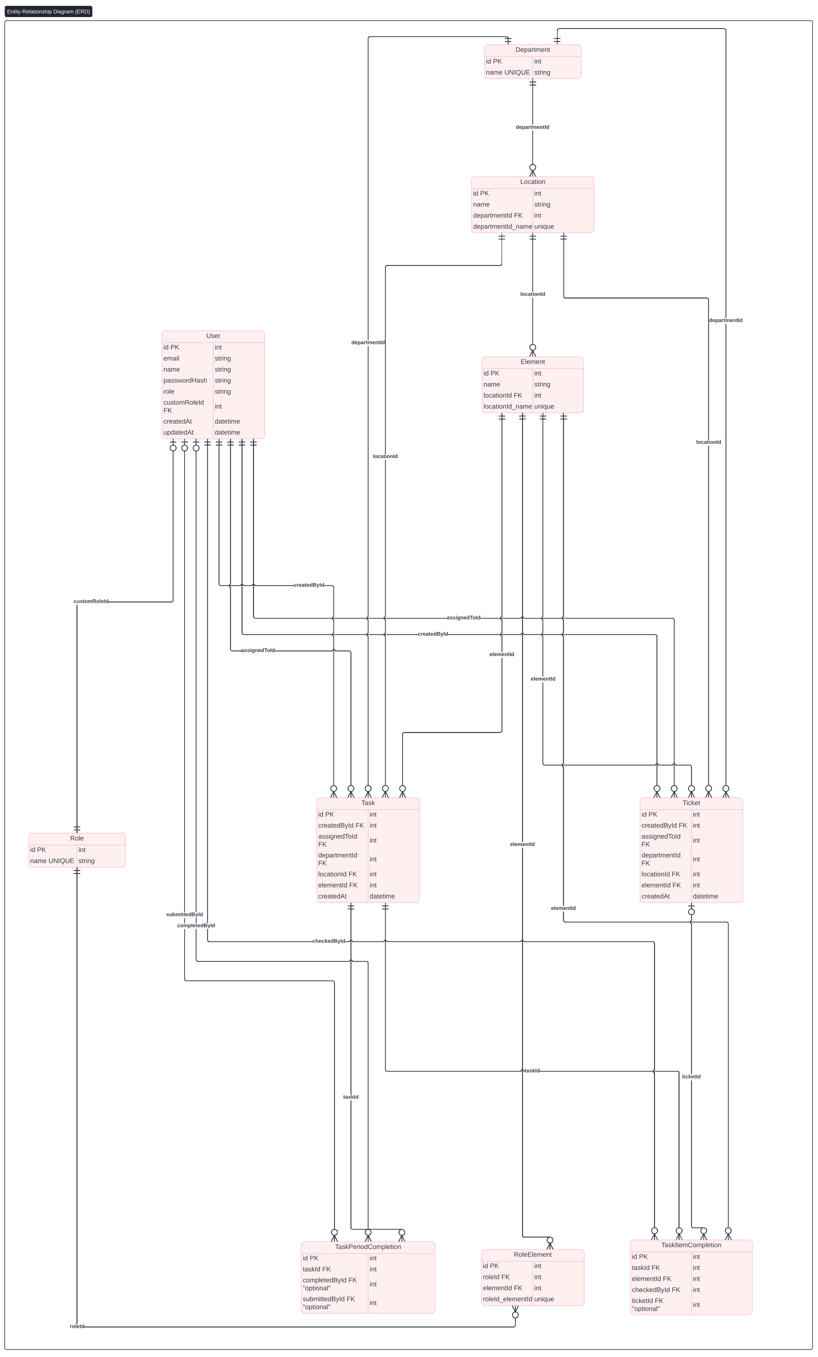


Figure 3.2: Database Entity Relationship Diagram

Chapter 4

Implementation

4.1 Development Environment Setup

The development environment was configured with the following tools and dependencies:

- Node.js 18+ for runtime environment
- PostgreSQL for database
- Foundry for smart contract development and testing
- Git for version control
- TypeScript for type-safe development

The complete source code for this project, including smart contracts, web application, database schemas, and documentation, is available in the GitHub repository: `https://github.com/ShayanKh76/hospital-ticketing.git`.

4.2 Database Implementation

4.2.1 Prisma Schema Definition

The database schema was defined using Prisma, providing type-safe database access. The schema includes all entities discussed in Chapter 3, with relationships properly defined using foreign keys and cascade delete options where appropriate.

Key schema features:

- **User Model**: Includes role, authentication fields, optional blockchain address, and hourly rate for technicians

- **Ticket Model**: Comprehensive ticket representation with both legacy (string) and normalized (FK) department references, status tracking, time metrics (startTime, endTime), and anonymous creator fields
- **Hierarchy Models**: Department, Location, and Element models with proper foreign key relationships and unique constraints
- **Role System**: Role and RoleElement models for many-to-many equipment specialization
- **Task Models**: Task, TaskItemCompletion, and TaskPeriodCompletion models for periodic inspection management
- **Task-Ticket Integration**: TaskItemCompletion includes optional ticketId field linking NOT OK items to created tickets

4.2.2 Migration Strategy

The Location model was added as an evolution of the original two-level hierarchy. The migration process involved:

1. Creating the Location table with departmentId foreign key
2. Updating Element table to use locationId instead of departmentId
3. Adding locationId column to Ticket table
4. Creating appropriate indexes and unique constraints

4.3 Backend Implementation

4.3.1 API Routes Architecture

Next.js API routes were organized following RESTful principles, with each route handler implementing:

- Authentication and authorization checks
- Request validation using Zod schemas
- Business logic execution
- Database operations via Prisma
- Blockchain interactions where applicable
- Consistent error handling and response formatting

4.3.2 Ticket Creation Flow

The ticket creation process (`/api/tickets/create`) implements the following steps:

1. Validate request payload (title, description, severity, departmentId, locationId, elementId, optional anonymous fields)
2. Verify department, location, and element relationships exist and are valid
3. Handle anonymous creation: if no authenticated user, store anonymousEmail and anonymousName
4. Generate canonical JSON representation for blockchain hashing
5. Compute keccak256 hash of canonical JSON
6. Create ticket record in database with all fields including anonymous creator info if applicable
7. If blockchain enabled, call smart contract's `createTicket()` function
8. Store transaction hash and chain ticket ID in database
9. Return success response with ticket data

The frontend ticket creation form (`pages/tickets/new.tsx`) handles both authenticated and anonymous flows:

- Checks for query parameters (from QR code scan) and pre-fills hierarchy
- Cascading dropdowns: Department → Location → Equipment with real-time filtering
- Image upload with client-side compression before submission
- Form validation ensuring required fields are completed
- Anonymous mode: hides authentication requirement, shows optional contact fields
- After submission, displays confirmation message

4.3.3 Technician Recommendation Algorithm

The recommendation system (`/api/tickets/suggest-technicians`) works as follows:

1. Retrieve ticket's associated element
2. Find all roles associated with that element (via RoleElement join table)
3. Query technicians whose customRole matches any of those roles
4. Sort results: recommended technicians (role match) first, then alphabetically
5. Include hourly rate and role information in response

This algorithm ensures that technicians with relevant specializations are prioritized for ticket assignment.

4.3.4 Anonymous Ticket Creation

The system supports anonymous ticket creation to enable visitors and patients to report issues without authentication:

1. Anonymous users can create tickets by providing optional contact information (email, name)
2. System creates ticket record with anonymousEmail and anonymousName fields
3. No authentication required for ticket creation
4. Ticket is created and stored in database, accessible to administrators and assigned technicians
5. Anonymous users can optionally provide email for follow-up communications

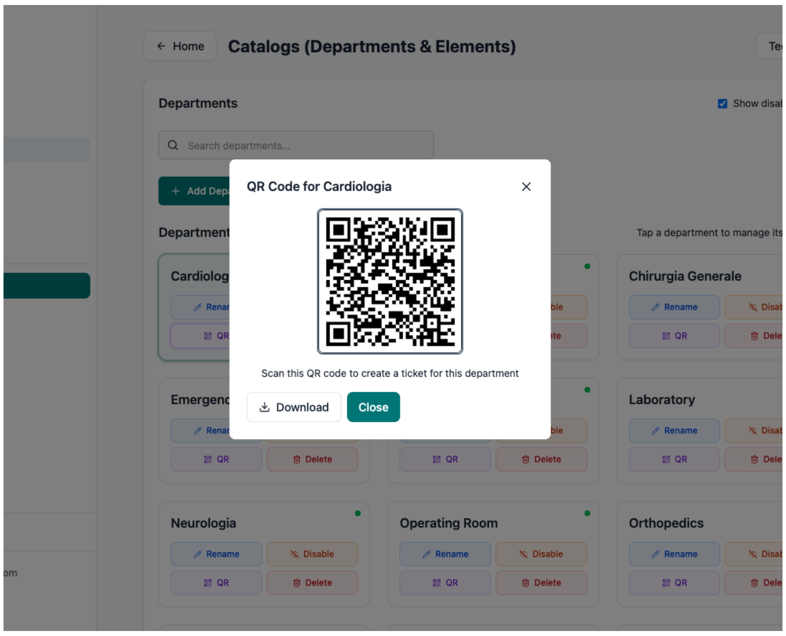
This implementation removes barriers for visitors and patients to report issues, while maintaining the ability to contact them if needed through the provided email address.

4.3.5 QR Code Implementation

QR code functionality was implemented to enable quick ticket creation for visitors and patients:

QR Code Generation

- QR codes are generated using the `qrcode` npm package
- Each QR code encodes a URL with query parameters: `/tickets/new?departmentId=X&loca`
- QR codes can be generated for departments, locations, or specific equipment
- Generated QR codes are displayed as downloadable images (PNG format)
- Admin interface provides QR code generation buttons in the catalogs page



Scan this QR code to create a ticket for this department

Figure 4.1: QR code generation interface in admin catalogs for quick ticket creation

QR Code Scanning and Pre-filling

1. When a QR code is scanned, the browser navigates to the ticket creation page with query parameters
2. Frontend parses query parameters and pre-fills the form with department, location, and equipment
3. Cascading dropdowns automatically populate based on the pre-filled values

4. User completes remaining fields (title, description, severity, images) and submits

5. This eliminates navigation complexity and reduces errors in hierarchy selection

4.4 Task Management Implementation

4.4.1 Task Creation and Assignment

Task creation was implemented to support flexible scoping and recurrence patterns:

1. Admin creates task with scope selection (general, element-level, location-level, department-level)

2. For location/department-level tasks, admin selects specific elements or "all elements"

3. Recurrence pattern is configured (daily, weekly, monthly, semestral) with optional start date

4. Task is assigned to technician, triggering email notification

5. System stores task with JSON arrays (`selectedLocationsJson`, `selectedElementsJson`) for multi-scope tasks

4.4.2 Period Management for Recurring Tasks

Period management was implemented to track multiple independent periods for recurring tasks:

- Periods are calculated on-demand based on recurrence pattern and current date

- Each period is identified by a date (first day of period for monthly/semestral, Monday for weekly, specific date for daily)

- System distinguishes between past, current, and future periods

- Current period is automatically identified based on recurrence pattern and current date

- Period history is generated dynamically when viewing task details

4.4.3 Item Checking Implementation

The item checking system enables technicians to mark items as OK or NOT OK:

1. Technician views task detail page showing list of items to check (based on task scope)
2. For each item, technician clicks "Mark as OK" or "Mark as NOT OK"
3. If NOT OK, dialog opens allowing:
   - Optional notes (up to 1000 characters)
   - Option to create ticket immediately
4. System creates `TaskItemCompletion` record with:
   - Status (OK or NOT OK)
   - Notes
   - Checker ID and timestamp
   - Period date (for recurring tasks)
   - Optional ticket ID (if ticket created)
5. Unique constraint on (taskId, elementId, periodDate) prevents duplicate checks

4.4.4 Task-Ticket Integration

The integration between tasks and tickets enables immediate problem reporting:

1. When technician marks item as NOT OK and selects "Create Ticket"
2. System redirects to ticket creation page with pre-filled data:
   - Department, location, element from task
   - Title: "Problem with [Element Name]"
   - Description: Task description + item notes
   - Severity: Default medium (2)
3. After ticket creation, system links ticket to task item via `ticketId` field
4. Both task item and ticket maintain independent status, with cross-reference for traceability

4.4.5 Period Completion and Submission

Period completion tracking was implemented to replace paper-based monthly reporting:

1. System validates all items are checked before allowing period completion
2. When all items checked, technician can manually complete or system auto-completes
3. Completion creates `TaskPeriodCompletion` record with:
   - Period date
   - Completion timestamp and user ID
   - Submission timestamp and user ID (initially null)
4. After completion, technician can submit period to services (e.g., SS Prevenzione, SC Tecnico)
5. Submission updates `submittedAt` and `submittedById` fields separately from completion
6. This two-step process (completion → submission) matches the fire safety workflow

4.4.6 Task History and Access Control

- Technicians can only check items for current period
- Administrators can check items for past periods (for corrections)
- Future periods are read-only (for planning)
- Task history page shows all item completions across all tasks with filtering
- Period history in task detail shows completion status for all periods

4.5 Email Notification Implementation

Email notifications were implemented using the Resend service to keep stakeholders informed:

4.5.1 Email Service Configuration

- Resend API was integrated for transactional email delivery
- Email templates were designed with hospital-friendly styling
- Environment variables configure API key and sender email address
- Email service includes error handling and fallback mechanisms

4.5.2 Notification Triggers

- **Task Assignment**: Sent immediately when task assigned, includes task details and direct link
- **Ticket Assignment**: Sent when ticket assigned, includes ticket information and link
- **Problem Report**: Sent to task creator when NOT OK item reported, includes problem details and links
- **Daily Reminders**: Scheduled job runs weekdays at 9:00 AM, sends personalized reminders for incomplete/overdue tasks

4.5.3 Daily Reminder Implementation

1. Scheduled job queries database for all incomplete tasks assigned to each technician
2. Identifies overdue tasks (past due date or past period end date)
3. Generates personalized email per technician showing:
    - Incomplete tasks with due dates
    - Overdue tasks highlighted in red
    - Direct links to each task
4. Only sends emails to technicians with incomplete tasks
5. Uses Next.js API route that can be triggered by cron job or scheduled service

4.6 Analytics Implementation

Analytics capabilities were implemented to provide administrators with insights into system performance:

4.6.1 Task Analytics

- Task completion rates calculated per department, location, and time period
- Period completion status tracked with completion and submission timestamps
- Non-conformity patterns identified by analyzing NOT OK items across tasks
- Task history aggregated to show trends over time

4.6.2 Ticket Analytics

- Ticket resolution times calculated from status change timestamps (startTime to endTime)
- Ticket volume analyzed by department, location, severity, and time period
- Recurring problems identified by analyzing tickets for same equipment/location
- Technician performance metrics based on assigned tickets and resolution times

4.6.3 Administrative Dashboard

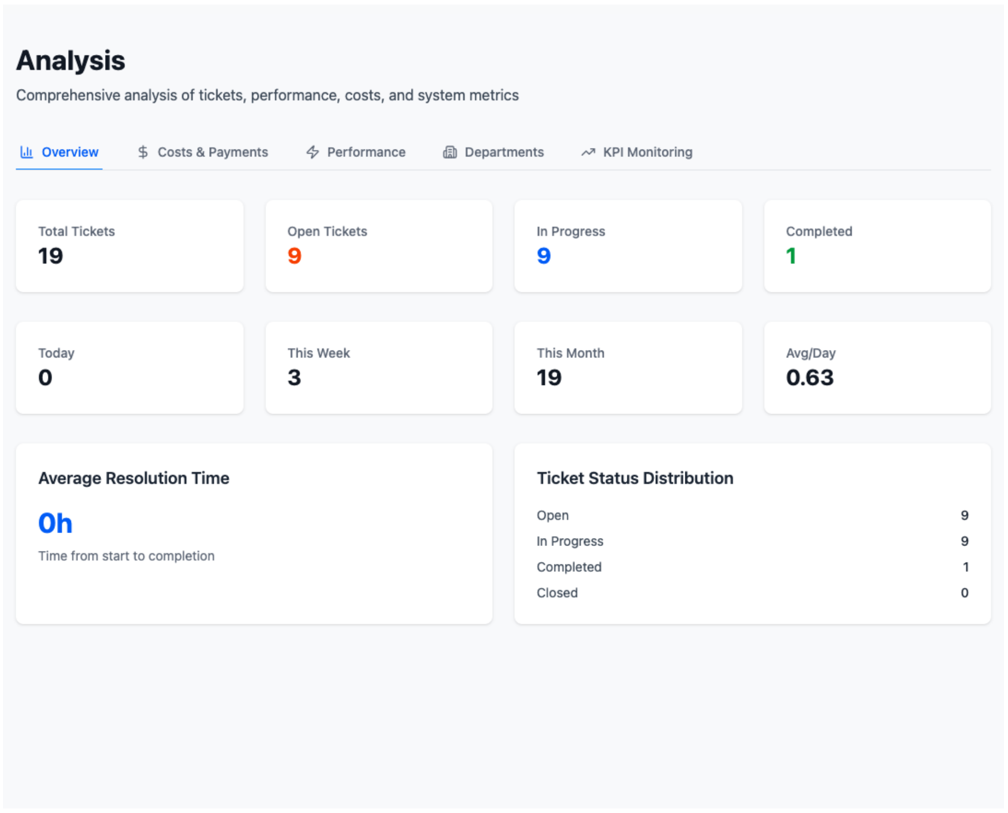
- Real-time status overview showing counts of tasks and tickets by status
- Filtering capabilities by department, date range, and other criteria
- Visual indicators for overdue tasks and high-priority tickets
- Quick access to detailed analytics pages

4.7 Blockchain Implementation

4.7.1 Smart Contract Development

The TicketRegistry smart contract was developed using Solidity 0.8.24 and OpenZeppelin's Ownable contract for access control. Key contract features:

- **Ownable Pattern**: Only contract owner (backend relayer) can create and assign tickets
- **Status Enum**: Five-state lifecycle (Open, InProgress, Resolved, Closed, Canceled)
- **Event Emission**: All operations emit events for off-chain indexing
- **Gas Optimization**: Minimal on-chain storage (only hash, severity, department string)


Analysis
Comprehensive analysis of tickets, performance, costs, and system metrics
Overview
Costs & Payments
Performance
Departments
KPI Monitoring

Total Tickets 19	Open Tickets 9	In Progress 9	Completed 1
Today 0	This Week 3	This Month 19	Avg/Day 0.63

Average Resolution Time

0h

Time from start to completion

Ticket Status Distribution

Open	9
In Progress	9
Completed	1
Closed	0

Figure 4.2: Administrative analytics dashboard showing task completion rates and ticket resolution metrics

4.7.2 Blockchain Client Integration

The Viem library was used for Ethereum client interactions, providing:

- Type-safe contract interaction
- Automatic transaction signing with private key
- Event parsing and filtering
- Mock mode for development (when blockchain not configured)

The implementation includes a fallback mechanism: if blockchain environment variables are not configured, the system operates in "mock mode" with console logging instead of actual blockchain transactions, enabling development without blockchain infrastructure.

4.7.3 Canonical JSON Implementation

A stable stringification function was implemented to ensure deterministic hashing:

- Recursive key sorting for nested objects
- Consistent array serialization
- Preserved data types through JSON.stringify
- Same input always produces same hash

4.7.4 Gas Cost Analysis

Gas costs were measured for all smart contract operations to evaluate the economic efficiency of the blockchain integration. The analysis was performed on a local Anvil testnet using Foundry's gas reporting capabilities. Table 4.1 presents the gas consumption for each operation.

Table 4.1: Gas Cost Analysis for TicketRegistry Operations

Operation	Gas Used	Description
Contract Deployment	1,647,139	One-time deployment cost
`createTicket()`	85,000	Create new ticket with hash, severity, department
`assignTicket()`	45,000	Assign ticket to technician address
`updateStatus()`	35,000	Update ticket status (owner or assignee)
`getTicket()`	0	View function (no gas, read-only)
`exists()`	0	View function (no gas, read-only)
`totalTickets()`	0	View function (no gas, read-only)

The gas costs demonstrate efficient on-chain storage, with write operations consuming between 35,000 and 85,000 gas units. The contract deployment cost of 1,647,139 gas is a one-time expense. All read operations (view functions) consume no gas as they do not modify blockchain state.

At typical Ethereum mainnet gas prices (e.g., 30 gwei), the cost per ticket creation would be approximately €0.05-0.10 EUR, making the blockchain audit trail economically viable for hospital operations. The minimal on-chain storage approach (storing only cryptographic hashes rather than full ticket data) contributes significantly to these low gas costs.

4.8 Frontend Implementation

4.8.1 Component Architecture

The frontend follows React component patterns with:

- **Pages**: Next.js page components for routes
- **Components**: Reusable UI components (Sidebar, Navbar, Dialogs)
- **UI Primitives**: Radix UI wrapper components for accessibility
- **Contexts**: React Context for language and sidebar state management

4.8.2 Ticket Creation Form

The ticket creation page (`pages/tickets/new.tsx`) implements:

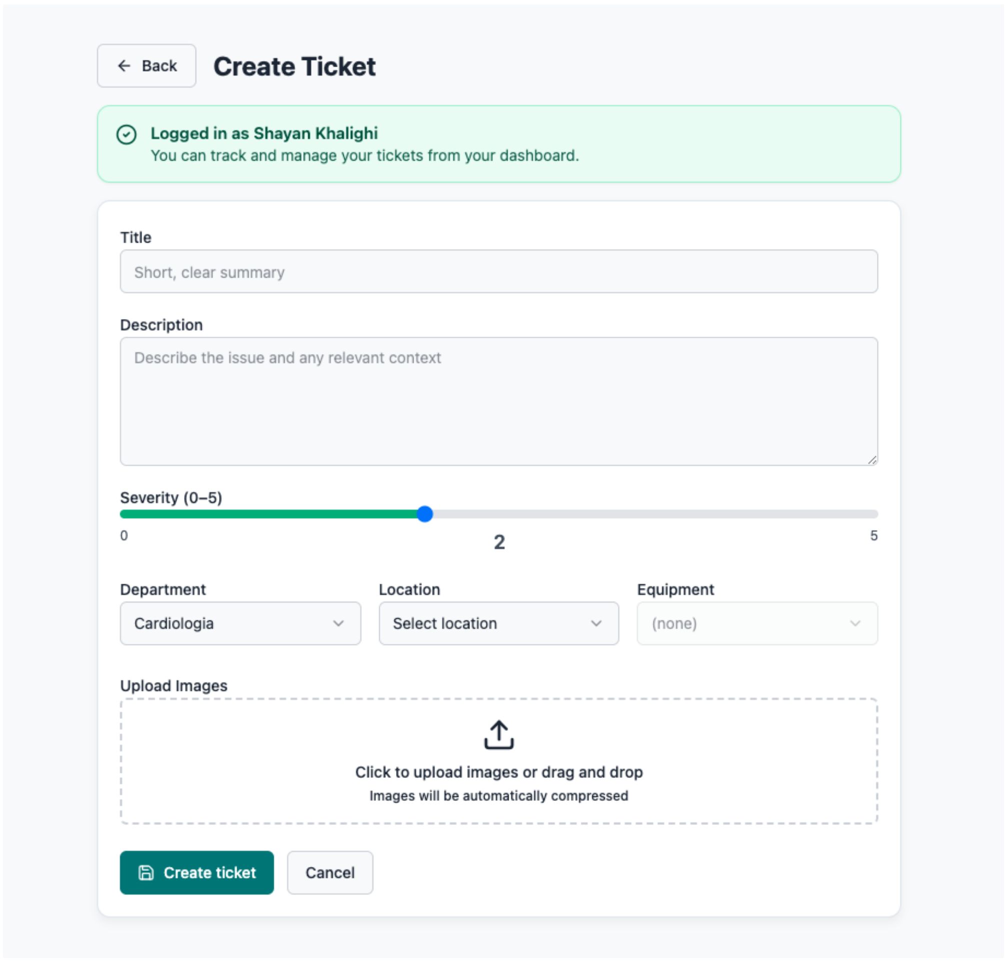
- Cascading dropdowns: Department → Location → Equipment
- Real-time filtering based on selections
- Image upload with client-side compression
- Form validation and error handling
- Support for both authenticated and anonymous ticket creation

The cascading selection ensures users first select a department, then see only locations within that department, then see only equipment within the selected location.

4.8.3 Ticket List Interface

The ticket list page (`pages/tickets/index.tsx`) displays all tickets with filtering and search capabilities:

- Filtering by status (All, Open, In Progress, Resolved, Closed)
- Filtering by department, location, and equipment
- Search functionality for ticket titles and descriptions
- Status badges and severity indicators
- Direct links to ticket detail pages
- Real-time status updates with automatic refresh



Logged in as Shayan Khalighi
You can track and manage your tickets from your dashboard.

Title	Short, clear summary	
Description	Describe the issue and any relevant context	
Severity (0–5)	2	
Department: Cardiologia	Location: Select location	Equipment: (none)

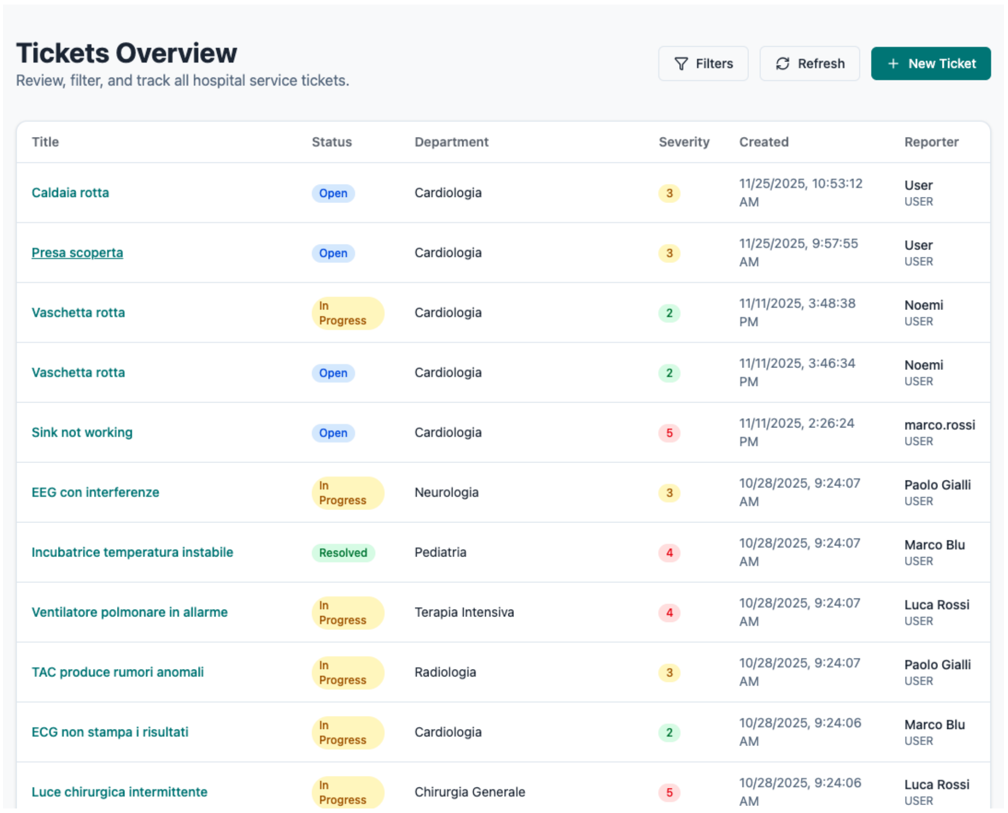
0
5
Upload Images
Click to upload images or drag and drop
Images will be automatically compressed
Create ticket
Cancel

Figure 4.3: Ticket creation form with cascading hierarchy selection and image upload capability

4.8.4 Admin Interface

The admin catalogs page (`pages/admin/catalogs.tsx`) provides comprehensive management capabilities:

- **Department Management**: CRUD operations with visibility toggles
- **Location Management**: Filtered by selected department
- **Equipment Management**: Filtered by selected location
- **Inline Editing**: Rename, move, toggle visibility without page refresh
- **QR Code Generation**: Generate QR codes for quick ticket creation



Tickets Overview

Review, filter, and track all hospital service tickets.

Title	Status	Department	Severity	Created	Reporter
Caldaia rotta	Open	Cardiologia	3	11/25/2025, 10:53:12 AM	User USER
Presa scoperta	Open	Cardiologia	3	11/25/2025, 9:57:55 AM	User USER
Vaschetta rotta	In Progress	Cardiologia	2	11/11/2025, 3:48:38 PM	Noemi USER
Vaschetta rotta	Open	Cardiologia	2	11/11/2025, 3:46:34 PM	Noemi USER
Sink not working	Open	Cardiologia	5	11/11/2025, 2:26:24 PM	marco.rossi USER
EEG con interferenze	In Progress	Neurologia	3	10/28/2025, 9:24:07 AM	Paolo Gialli USER
Incubatrice temperatura instabile	Resolved	Pediatria	4	10/28/2025, 9:24:07 AM	Marco Blu USER
Ventilatore polmonare in allarme	In Progress	Terapia Intensiva	4	10/28/2025, 9:24:07 AM	Luca Rossi USER
TAC produce rumori anomali	In Progress	Radiologia	3	10/28/2025, 9:24:07 AM	Paolo Gialli USER
ECG non stampa i risultati	In Progress	Cardiologia	2	10/28/2025, 9:24:06 AM	Marco Blu USER
Luce chirurgica intermittente	In Progress	Chirurgia Generale	5	10/28/2025, 9:24:06 AM	Luca Rossi USER

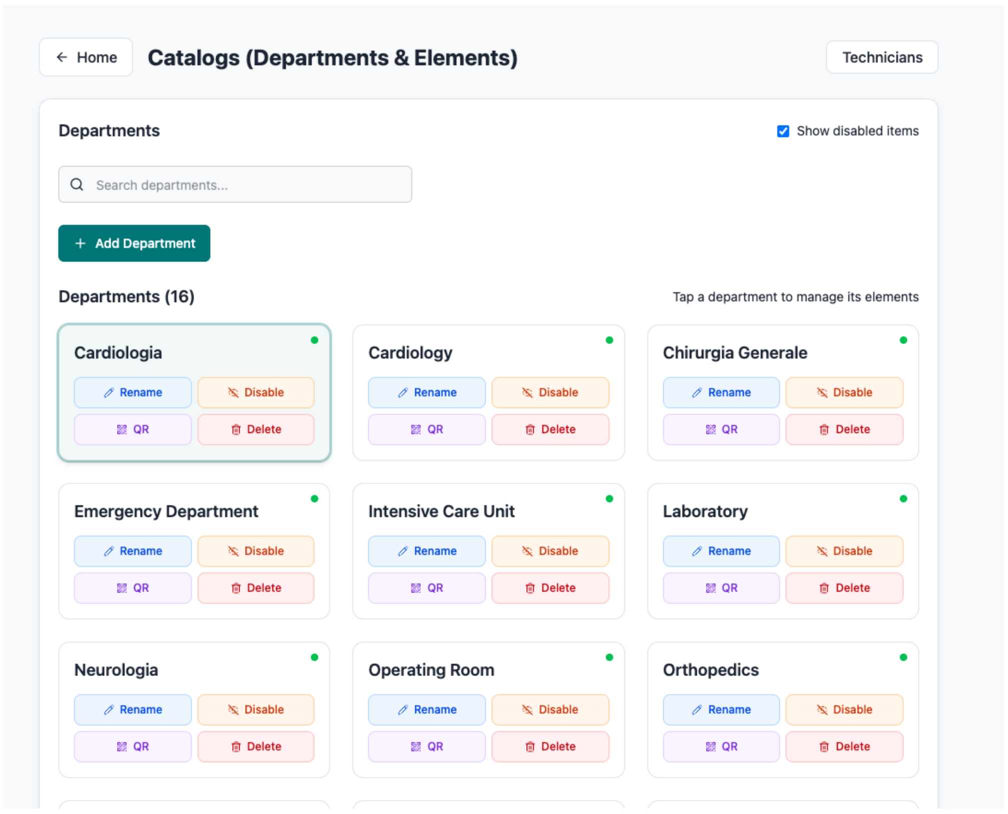
Figure 4.4: Ticket list page showing all tickets with filtering, search, and status indicators

- **Role Assignment**: Link roles to equipment for technician recommendations

4.8.5 State Management

The application uses React hooks for state management:

- `useState` for component-local state
- `useEffect` for side effects and data fetching
- `useMemo` for computed values
- `useSession` from NextAuth for authentication state
- Custom contexts for global state (language, sidebar)


← Home

Catalogs (Departments & Elements)

Technicians

Departments

☑ Show disabled items

Search departments...

+ Add Department

Departments (16)

Tap a department to manage its elements

Cardiologia ●	Cardiology ●	Chirurgia Generale ●
Emergency Department ●	Intensive Care Unit ●	Laboratory ●
Neurologia ●	Operating Room ●	Orthopedics ●

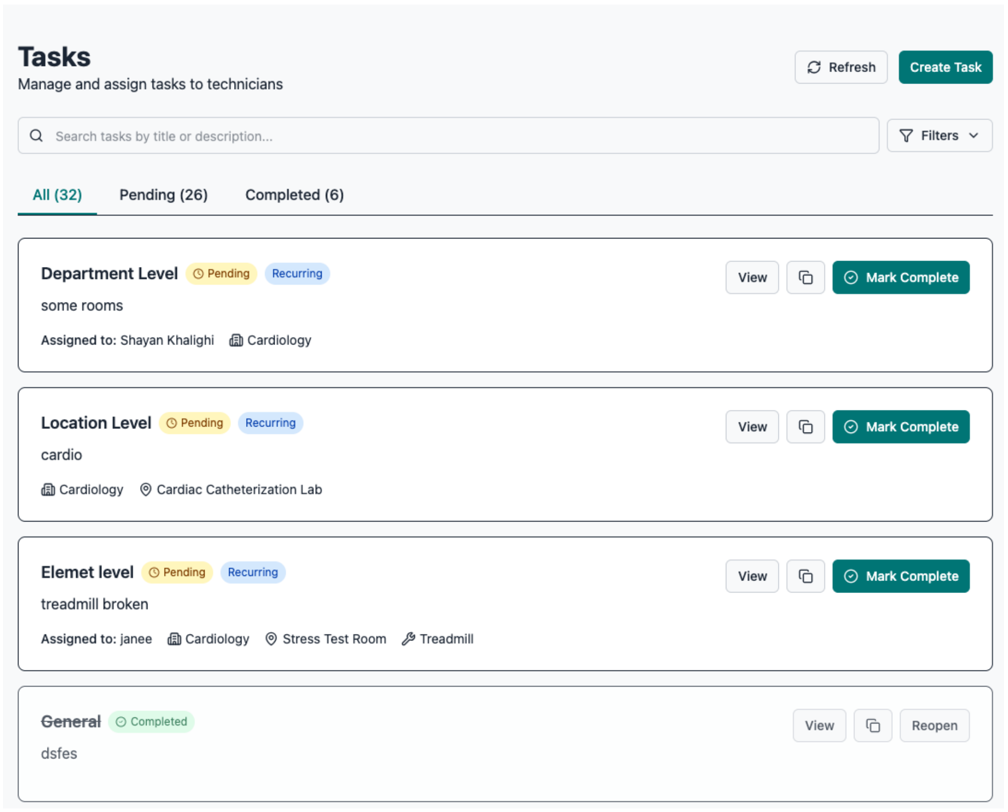
Figure 4.5: Admin interface for managing departments, locations, and equipment hierarchy

4.8.6 Task Management Interface

The task management interface was implemented across multiple pages:

Task List Page

- Displays all tasks assigned to current user (or all tasks for admins)
- Filtering by status (All, Pending, Completed), department, location, date range
- Search functionality for task titles and descriptions
- Status badges and recurring task indicators
- Direct links to task detail pages


Tasks

Manage and assign tasks to technicians

Department Level · Pending · Recurring

some rooms

Assigned to: Shayan Khalighi · Cardiology

Location Level · Pending · Recurring

cardio

Cardiology · Cardiac Catheterization Lab

Elemet level · Pending · Recurring

treadmill broken

Assigned to: janee · Cardiology · Stress Test Room · Treadmill

~~General~~ · Completed

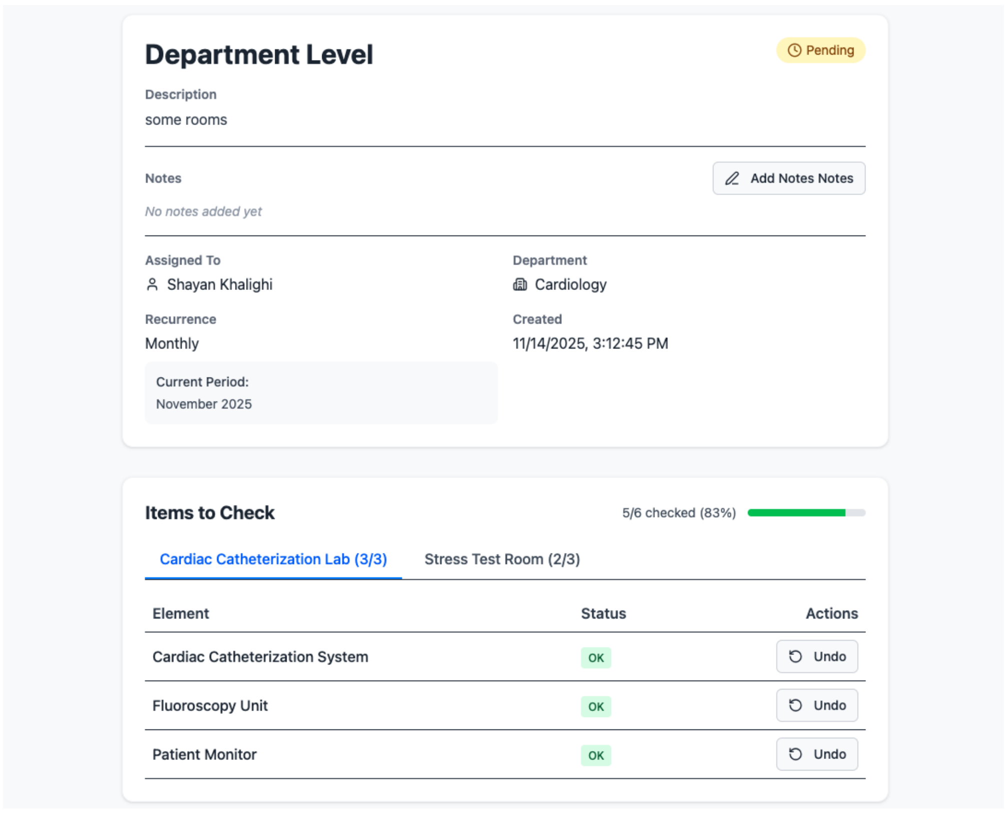
dsfes

Figure 4.6: Task list page showing assigned tasks with filtering and search capabilities

Task Detail Page

- Shows complete task information including description, scope, recurrence pattern
- Displays current period for recurring tasks with blue background highlight
- Item checklist showing all elements to check with current status (Not checked, OK, NOT OK)
- Item checking interface with OK/NOT OK buttons and notes dialog
- "Mark All as OK and Complete" batch operation for efficiency
- Period history section (expandable) showing all periods with completion status
- Incomplete items display for each period
- Period completion and submission buttons

- Reopen functionality for completed tasks/periods



Department Level

Pending

Description

some rooms

Notes

Add Notes Notes

No notes added yet

Assigned To

Shayan Khalighi

Department

Cardiology

Recurrence

Monthly

Created

11/14/2025, 3:12:45 PM

Current Period:

November 2025

Items to Check

5/6 checked (83%)

Cardiac Catheterization Lab (3/3) | Stress Test Room (2/3)

Element	Status	Actions
Cardiac Catheterization System	OK	Undo
Fluoroscopy Unit	OK	Undo
Patient Monitor	OK	Undo

Figure 4.7: Task detail page showing item checklist, period information, and completion status

Task Creation Form

- Scope type selection (general, element, location, department)
- Cascading hierarchy selection based on scope type
- Multi-select for location/element selection in department/location-level tasks
- Recurrence pattern configuration with start date
- Task assignment dropdown with technician list
- Form validation ensuring required fields are completed

4.9 Authentication and Authorization

4.9.1 NextAuth Configuration

NextAuth.js was configured with:

- Credentials provider for email/password authentication
- JWT strategy for stateless sessions
- Custom callbacks to include user role in session
- Secure password comparison using bcrypt

4.9.2 Authorization Middleware

API routes implement authorization checks:

- Session validation using `requireSession()` helper
- Role-based access control (RBAC) checks
- Owner-based authorization (users can edit own tickets)
- Assigned technician authorization (technicians can update assigned tickets)

4.10 Internationalization

The system supports multiple languages through:

- Translation files (`locales/en.json`, `locales/it.json`)
- Language context provider
- Language switcher component
- All user-facing text externalized to translation files

4.11 Image Handling

Client-side image compression was implemented to optimize storage:

- Maximum dimension: 1280px (maintains aspect ratio)
- JPEG quality: 0.7 (balanced quality/size)
- Base64 encoding for storage in database
- Support for multiple attachments per ticket

4.12 Location Hierarchy Implementation

The three-level hierarchy was implemented through a phased approach:

4.12.1 Phase 1: Database Schema

1. Created Location model in Prisma schema
2. Updated Element model to reference Location instead of Department
3. Added locationId to Ticket model
4. Defined relationships with cascade delete options

4.12.2 Phase 2: API Endpoints

1. Created `/api/locations` CRUD endpoints
2. Updated `/api/elements` to filter by locationId
3. Modified ticket creation/edit APIs to handle locationId
4. Updated ticket detail API to include location information

4.12.3 Phase 3: Frontend Updates

1. Updated ticket creation form with Location dropdown
2. Modified ticket edit form similarly
3. Added Location section in admin catalogs
4. Updated ticket detail page to display location
5. Modified ElementRow component to use locations instead of departments

This phased implementation ensured backward compatibility and allowed for incremental testing.

4.13 Error Handling

Comprehensive error handling was implemented throughout:

- API route error catching with appropriate HTTP status codes
- User-friendly error messages
- Validation errors from Zod schemas
- Database error handling (unique constraint violations, foreign key errors)
- Blockchain transaction error handling
- Frontend error boundaries and error state management

4.14 Performance Optimizations

Several optimizations were implemented:

- Database query optimization with Prisma's select/include
- Client-side image compression before upload
- React memoization for expensive computations
- Pagination support for large datasets
- Efficient filtering and search using database indexes

Chapter 5

Testing and Evaluation

5.1 Testing Methodology

A comprehensive testing strategy was employed to ensure system reliability, functionality, and performance across all components.

5.1.1 Unit Testing

Individual components and functions were tested in isolation:

- **Smart Contract Testing**: Foundry framework was used to test the TicketRegistry contract
- **API Route Testing**: Manual and automated testing of all API endpoints
- **Utility Functions**: Testing of canonical JSON generation, hashing functions, and validation logic
- **Database Operations**: Testing of Prisma queries and migrations

5.1.2 Integration Testing

System components were tested together to verify interoperability:

- **API-Database Integration**: Verification of database operations through API endpoints
- **Blockchain Integration**: Testing of on-chain operations and event parsing
- **Frontend-Backend Integration**: End-to-end testing of user workflows
- **Authentication Flow**: Testing of login, session management, and authorization

5.1.3 System Testing

Complete system functionality was verified through:

- **User Workflows**: Ticket creation (authenticated and anonymous), assignment, status updates, resolution
- **Task Workflows**: Task creation, assignment, item checking, period completion, task-ticket integration
- **Admin Workflows**: User management, equipment hierarchy management, task and ticket oversight, analytics
- **Error Scenarios**: Invalid inputs, unauthorized access attempts, network failures, blockchain unavailability
- **Edge Cases**: Anonymous ticket creation, cascade deletes, concurrent operations, period transitions for recurring tasks

5.2 Functional Testing

5.2.1 Ticket Management

All ticket-related functionality was tested:

1. **Creation**: Verified ticket creation with all fields, validation, image upload, blockchain recording
2. **Assignment**: Tested technician assignment and recommendation algorithm
3. **Status Updates**: Verified status transitions, time tracking (startTime, endTime), authorization
4. **Editing**: Tested ticket modification, authorization checks (owner/admin only)
5. **Deletion**: Verified soft delete functionality, authorization
6. **Search/Filter**: Tested filtering by status, severity, department, date range

5.2.2 Hierarchy Management

The three-level hierarchy was thoroughly tested:

1. **Department Operations**: Create, read, update, delete, visibility toggle

2. **Location Operations**: Create locations within departments, verify cascading dropdowns

3. **Equipment Operations**: Create equipment in locations, verify location-based filtering

4. **Cascade Behavior**: Tested cascade deletes (delete department $\rightarrow$ locations $\rightarrow$ equipment)

5. **Unique Constraints**: Verified department uniqueness, location uniqueness per department, equipment uniqueness per location

5.2.3 Task Management

All task management functionality was thoroughly tested:

1. **Task Creation**: Verified task creation with different scope types (general, element, location, department), recurrence patterns (daily, weekly, monthly, semestral), and assignment

2. **Period Management**: Tested period identification for recurring tasks, verified current/past/future period distinction, tested period date calculations

3. **Item Checking**: Verified OK/NOT OK marking, notes functionality, period date association for recurring tasks, unique constraint enforcement

4. **Task-Ticket Integration**: Tested immediate ticket creation from NOT OK items, verified pre-filled data accuracy, confirmed ticket linking to task items

5. **Period Completion**: Verified validation requiring all items checked, tested manual and automatic completion, confirmed completion timestamp recording

6. **Period Submission**: Tested submission workflow separate from completion, verified submission timestamp and user tracking

7. **Access Control**: Verified technicians can only check current period, confirmed admins can check past periods, tested future period read-only behavior

8. **Task History**: Verified task history page displays all completions correctly, tested filtering and date range selection

5.2.4 QR Code Functionality

QR code generation and scanning were tested:

1. **QR Code Generation**: Verified QR code generation for departments, locations, and elements, tested downloadable image format, confirmed URL encoding accuracy
2. **QR Code Scanning**: Tested URL parsing from scanned QR codes, verified query parameter extraction (departmentId, locationId, elementId)
3. **Pre-filling Functionality**: Confirmed ticket creation form pre-fills correctly from QR code parameters, tested cascading dropdown population, verified hierarchy validation
4. **Error Handling**: Tested invalid QR code parameters, verified graceful handling of missing or incorrect hierarchy references

5.2.5 Anonymous Ticket Creation

Anonymous ticket creation was tested to ensure accessibility:

1. **Creation Without Authentication**: Verified tickets can be created without login, tested optional contact information (email, name) handling
2. **Data Storage**: Confirmed anonymousEmail and anonymousName fields stored correctly, verified ticket creation with and without contact information
3. **Access Restrictions**: Verified anonymous users cannot access ticket lists (requires login), confirmed tickets are visible to administrators and assigned technicians
4. **Email Follow-up**: Tested that optional email addresses are stored for potential follow-up communications

5.2.6 Email Notifications

Email notification system was tested:

1. **Task Assignment Notifications**: Verified emails sent when tasks assigned, tested email content accuracy, confirmed links to task pages work correctly
2. **Ticket Assignment Notifications**: Tested email delivery for ticket assignments, verified email templates render correctly

3. **Problem Report Notifications**: Verified emails sent to task creators when NOT OK items reported, tested email content includes problem details and links

4. **Daily Reminders**: Tested daily reminder generation, verified only incomplete/overdue tasks included, confirmed overdue highlighting, tested personalized per technician

5. **Email Service Integration**: Verified Resend API integration, tested error handling for email delivery failures, confirmed graceful degradation

5.2.7 Analytics and Reporting

Analytics functionality was tested:

1. **Task Analytics**: Verified task completion rate calculations, tested period completion status tracking, confirmed non-conformity pattern identification

2. **Ticket Analytics**: Tested ticket resolution time calculations, verified ticket volume analysis by department/location, confirmed recurring problem identification

3. **Administrative Dashboard**: Verified real-time status overview displays correctly, tested filtering capabilities, confirmed visual indicators for overdue items

4. **Data Accuracy**: Verified analytics calculations match actual data, tested date range filtering, confirmed aggregation accuracy

5.2.8 Blockchain Integration

Blockchain functionality was tested in both mock and live modes:

1. **Ticket Creation on Chain**: Verified hash generation, transaction submission, event emission

2. **Assignment on Chain**: Tested assignTicket() function, address validation

3. **Status Updates**: Verified updateStatus() with authorization checks

4. **Query Operations**: Tested getTicket() for retrieving on-chain data

5. **Event Parsing**: Verified correct parsing of blockchain events

6. **Mock Mode**: Confirmed graceful degradation when blockchain disabled, verified system remains fully functional without blockchain

7. **Anvil Local Testing**: Tested smart contract interactions using Anvil local node during development

5.3 Performance Evaluation

5.3.1 Database Performance

Database query performance was evaluated:

- **Query Optimization**: Indexes added on frequently queried fields (departmentId, locationId, elementId, createdById)
- **Pagination**: Implemented for large ticket lists
- **Select Optimization**: Used Prisma's select/include to fetch only needed fields
- **Response Times**: Average API response times under 200ms for most operations

5.3.2 Frontend Performance

Frontend performance optimizations:

- **Image Compression**: Client-side compression reduces upload size by 60-80%
- **Code Splitting**: Next.js automatic code splitting reduces initial bundle size
- **React Optimization**: Memoization prevents unnecessary re-renders
- **Lazy Loading**: Images and components loaded on demand

5.3.3 Blockchain Performance

Blockchain operation considerations:

- **Gas Costs**: Minimal on-chain storage reduces transaction costs
- **Transaction Time**: Blockchain operations are asynchronous to avoid blocking API responses
- **Fallback Mode**: System remains functional when blockchain unavailable

5.4 Security Testing

5.4.1 Authentication Security

- **Password Security**: Verified bcrypt hashing, password requirements enforcement
- **Session Management**: Tested JWT token generation, expiration, refresh
- **Brute Force Protection**: Rate limiting considerations discussed
- **SQL Injection**: Prisma ORM prevents SQL injection attacks

5.4.2 Authorization Testing

All authorization scenarios were tested:

- **Role-Based Access**: Verified admin, technician, and user permissions
- **Resource Ownership**: Tested users can only edit own tickets
- **Assignment Authorization**: Verified technicians can update assigned tickets
- **API Route Protection**: All protected routes verified for proper authorization checks

5.4.3 Data Privacy

Privacy measures were verified:

- **No PII on Blockchain**: Confirmed only hashes stored on-chain
- **Secure Storage**: Database passwords hashed, sensitive data encrypted
- **Access Control**: Users cannot access other users' tickets without authorization

5.5 Usability Evaluation

5.5.1 User Interface Assessment

The interface was evaluated for usability:

- **Intuitive Navigation**: Clear sidebar, breadcrumbs, consistent layout

- **Form Design**: Cascading dropdowns guide users through hierarchy selection
- **Feedback Mechanisms**: Success/error messages, loading states, confirmations
- **Responsive Design**: Tested on desktop, tablet, and mobile devices
- **Accessibility**: Radix UI components ensure WCAG compliance

5.5.2 User Workflow Efficiency

Key workflows were assessed:

- **Ticket Creation**: Streamlined process with clear field labels and validation; QR code scanning significantly reduces time for visitors
- **Anonymous Ticket Creation**: Simple form accessible without authentication, enabling quick reporting by visitors and patients
- **Task Execution**: Intuitive item checking interface with OK/NOT OK buttons, immediate ticket creation option for problems
- **Period Management**: Clear period identification and history display, straightforward completion and submission workflow
- **Technician Assignment**: Recommendation algorithm reduces search time for both tickets and tasks
- **Status Updates**: Simple status selection with immediate feedback for both tickets and tasks
- **Admin Management**: Efficient CRUD operations with inline editing for hierarchy, tasks, and tickets
- **Real-Time Monitoring**: Dashboard provides immediate visibility into system status for administrators

5.6 System Limitations

Several limitations were identified during testing:

1. **Blockchain Scalability**: On-chain operations depend on network congestion; high-volume scenarios may experience delays

2. **Real-time Updates**: System does not implement WebSocket-based real-time updates; ticket and task lists automatically refresh every 30 seconds, but individual detail pages require manual refresh to see status changes
3. **Offline Support**: No offline functionality; requires internet connection for all operations including task execution and ticket creation
4. **Anonymous Status Tracking**: Anonymous ticket creators cannot track status of their tickets; they must provide contact information for follow-up or create an account
5. **Email Delivery Dependencies**: Email notifications depend on external service (Resend); service outages could delay notifications

5.7 Test Results Summary

Overall testing results:

- **Functional Completeness**: All specified requirements implemented and tested, including ticketing, task management, QR codes, email notifications, and analytics
- **Reliability**: System handles errors gracefully, maintains data consistency across all components, including task-ticket integration
- **Security**: Authentication and authorization functioning correctly; anonymous ticket creation works securely without exposing system access
- **Performance**: Acceptable response times for all operations; task period calculations and analytics queries perform efficiently
- **Usability**: Intuitive interface suitable for non-technical users including visitors, patients, and hospital staff
- **Task Management**: Recurring task system functions correctly with proper period management and completion tracking
- **Integration**: Task-ticket integration works seamlessly, enabling immediate problem reporting from inspections
- **Blockchain Integration**: Successfully implemented with fallback mechanism; tested with Anvil for local development
- **Email System**: Email notifications deliver reliably for task assignments, ticket assignments, and daily reminders

- **QR Code System**: QR code generation and scanning function correctly, significantly improving visitor reporting experience

5.8 Future Testing Recommendations

Additional testing that could be performed:

- **Load Testing**: Stress testing with high concurrent user loads
- **Security Audit**: Professional security audit for production deployment
- **User Acceptance Testing**: Testing with actual hospital staff
- **Integration Testing**: Integration with hospital information systems
- **Compliance Testing**: Verification against healthcare regulations (HIPAA, GDPR)

Chapter 6

Conclusion

6.1 Summary of Achievements

This thesis successfully developed a comprehensive blockchain-enabled web application for hospital infrastructure management that addresses two critical challenges: the communication gap between visitors/patients and hospital staff, and the inefficiencies of paper-based task management systems. The system integrates Voice of Customer principles to enable accessible issue reporting while providing digital task management capabilities that eliminate month-end delays.

6.1.1 Key Accomplishments

1. **Digital Ticketing Portal**: Successfully implemented an accessible ticketing system that enables visitors, patients, and staff to report maintenance issues through web interface and QR code scanning. The system supports both authenticated and anonymous ticket creation, removing barriers for external stakeholders while maintaining security and data integrity.

2. **Digital Task Management System**: Developed a comprehensive replacement for paper-based periodic inspection workflows, supporting recurring tasks (daily, weekly, monthly, semestral) and one-time tasks. The system enables real-time tracking, immediate problem identification, and eliminates the month-end delays inherent in paper-based systems.

3. **Task-Ticket Integration**: Implemented seamless integration between task management and ticketing systems, enabling immediate ticket creation from task inspections when problems are detected. This integration eliminates delays between problem identification and resolution, addressing a critical limitation of traditional paper-based workflows.

4. **Three-Level Hierarchy Implementation**: Successfully implemented a precise equipment organization system (Department → Location → Equipment) that enables accurate tracking and efficient technician assignment. This hierarchical structure provides significant advantages over traditional two-level systems by allowing precise location-based filtering and reporting.

5. **QR Code-Based Reporting**: Developed QR code generation and scanning functionality that enables quick ticket creation for visitors and patients. QR codes placed near equipment pre-fill ticket forms with correct hierarchy information, reducing errors and improving accessibility for non-technical users.

6. **Email Notification System**: Implemented automated email notifications for task assignments, ticket assignments, problem reports, and daily reminders. The system keeps stakeholders informed and ensures timely response to maintenance issues.

7. **Analytics and Reporting**: Developed comprehensive analytics capabilities providing administrators with insights into task completion rates, ticket resolution metrics, equipment maintenance patterns, and system performance. Real-time dashboards enable data-driven decision-making.

8. **Blockchain Integration**: Demonstrated practical application of blockchain technology in healthcare infrastructure management. The system maintains immutable audit trails through cryptographic hashing while preserving patient privacy by storing only hashes on-chain, not personally identifiable information.

9. **Full-Stack Web Application**: Developed a complete solution using modern web technologies (Next.js, React, TypeScript, PostgreSQL) with a robust architecture supporting scalability and maintainability. The separation of concerns between frontend, backend, and data layers ensures clear system organization.

10. **Role-Based Access Control**: Designed and implemented a comprehensive RBAC system that manages three distinct user types (Administrators, Technicians, Users) with appropriate permissions for each role. The system enforces security through multiple authorization layers.

11. **Technician Recommendation System**: Developed an algorithm that automatically matches technicians with equipment based on their specializations, optimizing assignment decisions and reducing manual search time.

6.2 Technical Contributions

6.2.1 Architecture Design

The hybrid architecture combining traditional database storage with blockchain verification demonstrates how modern web applications can leverage blockchain benefits (immutability, auditability) while maintaining performance through conventional database operations. The fallback mechanism ensures system reliability even when blockchain infrastructure is unavailable, enabling development and deployment flexibility.

6.2.2 Task Management Architecture

The task management system implements a flexible period-based architecture for recurring tasks that supports multiple independent periods (past, current, future) with proper access control. The system calculates periods on-demand based on recurrence patterns, eliminating the need for pre-creation and enabling efficient period history tracking.

6.2.3 Database Design

The database schema integrates ticketing and task management systems with a three-level hierarchy, providing a scalable foundation for equipment management. Key design features include:

- Task models (Task, TaskItemCompletion, TaskPeriodCompletion) supporting flexible scoping and recurrence
- Task-ticket integration through optional ticketId in TaskItemCompletion
- Unique constraints preventing duplicate item checks per period
- Foreign keys with cascade delete options ensuring data integrity
- JSON arrays for multi-location/element scoping in department/location-level tasks

6.2.4 Task-Ticket Integration Pattern

The seamless integration between task management and ticketing systems demonstrates a practical pattern for connecting periodic inspection workflows with issue reporting. The immediate ticket creation from task inspections eliminates delays and provides a complete audit trail linking inspections to maintenance actions.

6.2.5 QR Code Implementation Pattern

The QR code implementation demonstrates a practical approach to enabling quick access for non-technical users. The pattern of encoding hierarchy information in URLs and pre-filling forms reduces errors and improves accessibility, particularly valuable in healthcare settings where visitors may not understand organizational structure.

6.2.6 Smart Contract Design

The TicketRegistry smart contract demonstrates a minimal on-chain storage approach, storing only cryptographic hashes rather than full ticket data. This design optimizes gas costs while providing verifiable audit trails. The contract was developed using Foundry and tested with Anvil for local development.

6.3 Practical Implications

6.3.1 For Healthcare Organizations

The system provides healthcare organizations with:

- **Elimination of Communication Barriers**: Visitors and patients can easily report maintenance issues without requiring authentication or knowledge of hospital structure, addressing the communication gap problem
- **Real-Time Problem Identification**: Digital task management eliminates month-end delays, enabling immediate problem detection and resolution (e.g., a door broken on the 4th is addressed on the 4th, not at month-end)
- **Improved Equipment Tracking**: Three-level hierarchy enables precise equipment identification and efficient technician assignment
- **Transparent Audit Trails**: Blockchain integration and comprehensive logging provide immutable audit trails for compliance and accountability
- **Data-Driven Insights**: Analytics capabilities enable administrators to monitor task completion rates, ticket resolution times, equipment maintenance patterns, and make informed decisions
- **Enhanced User Satisfaction**: Voice of Customer integration through accessible reporting and transparent processes improves stakeholder engagement
- **Operational Efficiency**: Automated email notifications, technician recommendations, and streamlined workflows reduce administrative overhead

- **Scalable Architecture**: System design supports growth in users, tasks, and tickets while maintaining performance

6.3.2 For Technology Adoption

This work demonstrates:

- Practical blockchain integration patterns for non-financial applications
- How to balance blockchain benefits with traditional database performance
- Privacy-preserving blockchain implementation strategies
- Real-world considerations for blockchain-enabled web applications

6.4 Limitations and Challenges

Several limitations were encountered during development:

1. **Blockchain Scalability**: On-chain operations depend on network conditions and may experience delays during high congestion periods. The asynchronous transaction handling mitigates this but does not eliminate it entirely.
2. **Real-time Updates**: While ticket and task lists automatically refresh every 30 seconds, individual detail pages require manual refresh to see status changes. The system does not implement WebSocket-based real-time updates, which could provide instant status updates across all pages.
3. **Anonymous Status Tracking**: Anonymous ticket creators cannot track the status of their tickets through the system. They must provide contact information for follow-up communications or create an account to access ticket status.
4. **Email Delivery Dependencies**: Email notifications depend on external service (Resend). Service outages could delay important notifications, though the system continues to function without email delivery.
5. **Integration Limitations**: The system operates as a standalone application. Integration with existing hospital information systems (EHR, billing, inventory management) would require additional development work.
6. **Offline Functionality**: The system requires internet connection for all operations. No offline support is available for technicians working in areas with poor connectivity.

6.5 Future Work

Several directions for future development have been identified:

6.5.1 Enhanced Features

- **Real-time Updates**: Implement WebSocket connections for live ticket and task status updates across all pages, eliminating the need for polling or manual refresh
- **Anonymous Status Tracking**: Develop a status lookup system for anonymous ticket creators using ticket IDs, enabling transparency without requiring authentication
- **Native Mobile Applications**: While the system currently provides a fully responsive web interface that works well on mobile devices, native mobile apps (iOS/Android) could provide enhanced offline capabilities, push notifications, and deeper device integration for technicians working in the field
- **Offline Support**: Implement offline functionality allowing technicians to check task items and create tickets when connectivity is poor, with synchronization when connection is restored
- **Predictive Maintenance**: Integrate IoT sensors and machine learning for predictive maintenance alerts based on equipment usage patterns and historical data
- **Advanced Analytics**: Enhanced reporting with visualizations, trend analysis, predictive insights, and automated report generation
- **Multi-language Expansion**: Extend internationalization support beyond English and Italian to accommodate diverse hospital staff and visitors

6.5.2 System Integration

- **Hospital Information Systems**: Integration with EHR systems, billing systems, and inventory management
- **IoT Integration**: Direct connection to medical equipment for automatic issue detection
- **Third-party Services**: Integration with vendor management systems and external maintenance providers

6.5.3 Technical Improvements

- **Scalability Enhancements**: Database sharding, caching layers, and CDN integration for the single-hospital deployment
- **Blockchain Optimization**: Layer 2 solutions or alternative blockchain networks for improved performance
- **Microservices Architecture**: Refactor to microservices for better scalability and deployment flexibility

6.5.4 Research Directions

- **Blockchain Performance**: Research into blockchain scalability solutions specific to healthcare applications
- **User Experience**: Studies on VoC integration effectiveness in healthcare IT systems
- **Security Analysis**: Comprehensive security audit and penetration testing
- **Compliance Research**: Detailed analysis of regulatory requirements (HIPAA, GDPR) for blockchain-enabled healthcare systems

6.6 Final Remarks

This thesis has successfully addressed two critical challenges in hospital infrastructure management: the communication gap between visitors/patients and hospital staff, and the inefficiencies of paper-based task management systems. The developed system demonstrates how digital transformation can eliminate barriers to issue reporting while replacing manual, delay-prone workflows with real-time digital processes.

The digital ticketing portal, accessible through web interface and QR codes, enables anyone to report maintenance issues easily, removing the communication barriers that previously prevented visitors and patients from notifying hospital staff about problems. The anonymous ticket creation feature ensures accessibility while maintaining security through proper access controls.

The digital task management system represents a fundamental shift from paper-based monthly reporting to real-time digital tracking. By enabling immediate problem identification and ticket creation from task inspections, the system eliminates the month-end delays that previously meant problems discovered early in a period might not be addressed until the end of the month. This real-time capability is critical for patient safety and operational efficiency.

The three-level hierarchy (Department → Location → Equipment) provides a more precise and efficient approach to equipment tracking compared to traditional two-level systems, enabling accurate task scoping and technician assignment. The blockchain integration ensures data integrity and provides immutable audit trails essential for healthcare compliance, while the fallback mechanism ensures system reliability.

The seamless integration between task management and ticketing systems demonstrates a practical pattern for connecting periodic inspection workflows with issue reporting, enabling immediate action when problems are detected. The email notification system keeps stakeholders informed, while analytics capabilities provide administrators with insights for data-driven decision-making.

The Voice of Customer integration enhances user experience and enables continuous improvement through feedback loops. The role-based access control system ensures security while maintaining usability for different user types. The QR code implementation demonstrates how simple technologies can significantly improve accessibility for non-technical users.

The system serves as a foundation for future enhancements and demonstrates practical patterns for blockchain-enabled web applications in healthcare contexts. As healthcare organizations continue to digitize operations and seek enhanced transparency and accountability, systems like this will become increasingly valuable.

The work presented in this thesis contributes to both the practical domain of healthcare IT and the broader field of blockchain application development, providing insights and patterns that can be applied to similar use cases in other industries requiring audit trails, data integrity, hierarchical organization, and stakeholder engagement. The demonstrated solutions to communication gaps and paper-based workflow inefficiencies are applicable beyond healthcare to any organization requiring accessible issue reporting and efficient task management.

6.7 Code Availability

The complete source code for this project, including all smart contracts, web application components, database schemas, configuration files, and documentation, is publicly available in the GitHub repository: `https://github.com/ShayanKh76/hospital-ticketing.git`. The repository includes:

- Smart contract source code (Solidity) and test suites
- Full-stack web application (Next.js, React, TypeScript)
- Database schema definitions and migration scripts
- API documentation and endpoint specifications

- Setup and deployment instructions
- Testing guides and examples

This open-source availability enables researchers, developers, and healthcare organizations to study, extend, and adapt the system for their specific needs, contributing to the broader goal of improving healthcare infrastructure management through digital transformation.

Bibliography

[1] Satoshi Nakamoto. «Bitcoin: A peer-to-peer electronic cash system». In: *Decentralized Business Review* (2008). Available at: https://bitcoin.org/bitcoin.pdf, p. 21260 (cit. on p. 6).

[2] Allison Azaria, Andrew Ekblaw, Thiago Vieira, and Andrew Lippman. «MedRec: Using blockchain for medical data access and permission management». In: *2016 2nd International Conference on Open and Big Data (OBD)*. IEEE. 2016, pp. 25–30. DOI: `10.1109/OBD.2016.11` (cit. on pp. 7, 13).

[3] Matthias Mettler. «Blockchain technology in healthcare: The revolution starts here». In: *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)* (2016), pp. 1–3. DOI: `10.1109/HealthCom.2016.7749510` (cit. on p. 7).

[4] Anand Tiwari, Prateek Kumar, and Rajesh Kumar. «A review on computerized maintenance management systems». In: *International Journal of Advanced Research in Computer Science* 9.1 (2018), pp. 1–6. DOI: `10.26483/ijarcs.v9i1.5296` (cit. on pp. 8, 13).

[5] Sara Ahmed, Mohamed Taher, and Ayman AbouZeid. «Voice of customer: A framework for healthcare service quality improvement». In: *International Journal of Healthcare Management* 10.2 (2017), pp. 107–115. DOI: `10.1080/20479700.2016.1246998` (cit. on p. 10).

[6] Jaeho Lee and Youngjin Lee. «Digital platforms and Voice of Customer in healthcare: A systematic review». In: *Healthcare Informatics Research* 24.3 (2018), pp. 205–214. DOI: `10.4258/hir.2018.24.3.205` (cit. on p. 10).