جامعة عبد المالك السعدي
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵄⴰⴱⴷⵍⵎⴰⵍⵉⴽ ⵙⵙⴰⵄⴷⵉ
Université Abdelmalek Essaadi

كلية العلوم و التقنيات بطنجة
ⵜⴰⵖⵣⵍⵓⵜ ⵏ ⵜⵜⵎⵓⵔⵜⵉⵏ ⴷ ⵜⵓⵙⵙⵔⵜⵉⵏ ⵏ ⵟⴰⵏⵊⴰ
Faculté des Sciences et Techniques de Tanger

CityFlow: A Secure Blockchain-Based Smart Parking and EV Charging Platform with Integrated Mini-SOC

MST : Artificial Intelligence and Data Science

Subject: SUB-PROJECT 4: SMART PARKING & EV CHARGING
Authors: Charraj Zakariaa, Mouhsin Boumargoud, Soufiane Hbich, Soueilem Mohamed Abd Nasir
Supervised by: Prof. Ikram Benabdelouahab, Prof. Mohammed BOUHORMA
Date: January 5, 2026

Contents :

Abstract

The rapid evolution of Smart City infrastructures introduces complex challenges related to scalability, trust, and cybersecurity. Traditional centralized systems used for traffic management, parking, and electric vehicle (EV) charging remain vulnerable to single points of failure, data tampering, and delayed incident response.
This project, **CityFlow**, proposes a decentralized and secure Smart City platform that combines **Hyperledger Fabric blockchain technology**, a **microservice-based backend**, a **modern React frontend**, and an **integrated security monitoring system inspired by Security Operations Centers (SOC)**.

CityFlow adopts a *"Pure Blockchain"* architecture, where all critical application state—including user identities, parking reservations, charging sessions, and financial transactions—is stored directly on a permissioned blockchain ledger. This ensures immutability, auditability, and multi-organization trust across a consortium composed of city authorities, service operators, and users.

In addition to functional services, the platform integrates an **educational Mini-SOC security layer** that continuously monitors API activity using middleware-based event logging, rule-based alerting, and real-time dashboards. This security subsystem detects threats such as brute-force attacks, unauthorized access attempts, and abnormal request patterns, while providing a quantitative system health score for administrators.

Through architectural design, implementation, and experimental evaluation, this work demonstrates how blockchain, modern web technologies, and intelligent security supervision can be combined to build resilient, transparent, and secure Smart City infrastructures.

List of Abbreviations and Acronyms

API	Application Programming Interface
BFT	Byzantine Fault Tolerance
CA	Certificate Authority
CFT	Crash Fault Tolerance
CI/CD	Continuous Integration / Continuous Deployment
CSV	Comma-Separated Values
DePIN	Decentralized Physical Infrastructure Network
DID	Decentralized Identifier
DLT	Distributed Ledger Technology
DoS	Denial of Service
EV	Electric Vehicle
GIN	Go HTTP Web Framework
HMR	Hot Module Replacement
HTTP	HyperText Transfer Protocol
JWT	JSON Web Token
LLM	Large Language Model
MSP	Membership Service Provider
RBAC	Role-Based Access Control
REST	Representational State Transfer
SDK	Software Development Kit

SIEM	Security Information and Event Management
SOC	Security Operations Center
TPS	Transactions Per Second
UI	User Interface
UML	Unified Modeling Language
WS	WebSocket

General Introduction

Context

The rapid urbanization of modern societies has led cities to increasingly rely on digital platforms to manage critical services such as traffic regulation, parking availability, and electric vehicle (EV) charging infrastructure. These services form the backbone of what is commonly referred to as **Smart Cities**, where physical infrastructure is tightly coupled with software systems, real-time data collection, and automated decision-making.

In most existing deployments, Smart City platforms are built upon **centralized architectures**. While such systems simplify governance and deployment, they introduce structural weaknesses, including single points of failure, limited transparency, and a high dependency on trusted intermediaries. As cities scale and integrate heterogeneous service providers, these limitations become more pronounced, especially in environments that require strong guarantees of data integrity, auditability, and inter-organizational trust.

At the same time, the exposure of Smart City services through public APIs and web interfaces has significantly expanded the attack surface. Authentication abuse, unauthorized access attempts, and denial-of-service patterns are increasingly common, yet many platforms lack continuous and automated security supervision. This creates a gap between operational functionality and cybersecurity resilience.

Problem Statement

The core problem addressed in this project lies at the intersection of **trust, decentralization, and security supervision**. Centralized Smart City systems suffer from three major shortcomings:

1. **Lack of Trust and Transparency**
   Data related to parking usage, financial transactions, and infrastructure state is often stored in proprietary databases controlled by a single authority, making independent verification impossible.
2. **Limited Fault Tolerance**
   A failure or compromise of the central server can disrupt city-wide services, leading to operational downtime and potential safety risks.
3. **Insufficient Security Monitoring**
   Many systems rely on basic logging mechanisms without real-time analysis, alerting, or health assessment, resulting in delayed detection of malicious behavior.

These challenges highlight the need for a new architectural approach that not only decentralizes data management but also embeds security monitoring as a first-class component of the system.

Objectives

The primary objective of the **CityFlow** project is to design and implement a **secure, decentralized Smart City platform** that addresses the limitations of traditional architectures. The specific objectives are as follows:

- To design a **permissioned blockchain infrastructure** using Hyperledger Fabric, enabling multi-organization governance, immutable data storage, and fine-grained access control.
- To implement core Smart City services, including smart parking, EV charging management, and a blockchain-based digital wallet.
- To develop a **full-stack application** combining a modern React frontend with a Go-based backend acting as a blockchain gateway.
- To integrate an **educational Mini-SOC security monitoring system** capable of:
    - Automatically logging all API activity,
    - Detecting suspicious behavior through rule-based alerting,
    - Providing real-time dashboards and a quantitative system health score for administrators.
- To evaluate the system in terms of performance, resilience, and security visibility.

Methodological Approach

This project follows a **design-and-implementation-oriented methodology**, combining theoretical analysis with practical engineering. The approach includes:

- A **state-of-the-art study** covering Smart City infrastructures, blockchain technologies, and cybersecurity monitoring systems.
- An **iterative development process**, inspired by Agile practices, allowing incremental integration of blockchain, frontend, and security components.
- A **security-by-design philosophy**, where authentication, authorization, and monitoring mechanisms are embedded at every layer of the system.
- Experimental evaluation of consensus mechanisms and system behavior under load and failure scenarios.

Structure of the Report

This report is structured into four main parts:

- **Part I** presents the project context, methodology, and state of the art.
- **Part II** focuses on system analysis and architectural design.
- **Part III** details the implementation of the blockchain infrastructure and experimental evaluation.
- **Part IV** addresses advanced security mechanisms and the user interface, including the Mini-SOC security monitoring system.

The report concludes with a synthesis of results, discussion of limitations, and perspectives for future enhancements.

Part I – Chapter 1:

Project Context and Methodolog

1.1 Smart Cities and Digital Transformation

Smart Cities represent an evolution in urban management, where information and communication technologies (ICT) are leveraged to improve the efficiency, sustainability, and quality of life within urban environments. Core services such as transportation, energy distribution, parking, and public safety increasingly depend on interconnected digital platforms capable of processing large volumes of data in real time.

This digital transformation is driven by several factors: the proliferation of IoT devices, the availability of high-bandwidth networks, and the need for data-driven decision-making. Sensors deployed across the city continuously generate data related to traffic density, vehicle occupancy, energy consumption, and infrastructure usage. When properly exploited, this data enables predictive analytics, dynamic resource allocation, and automated service orchestration.

However, the integration of heterogeneous systems operated by different stakeholders—municipal authorities, private operators, and citizens—raises significant challenges. Interoperability, data ownership, and trust management become critical concerns, particularly when financial transactions and personal data are involved.

1.2 Limitations of Centralized Architectures

Despite their widespread adoption, centralized Smart City platforms exhibit structural limitations that hinder scalability and resilience:

- **Single Point of Failure**: Central servers represent critical failure points. Hardware outages, software bugs, or cyberattacks can disrupt multiple services simultaneously.
- **Trust Dependency**: All participants must trust the central authority to store, process, and share data honestly. This assumption is problematic in multi-organization environments.
- **Limited Auditability**: Centralized databases can be modified without leaving transparent and immutable traces, complicating forensic analysis and regulatory compliance.
- **Scalability Constraints**: As the number of users and devices grows, centralized systems face performance bottlenecks and increasing operational costs.

These limitations motivate the exploration of alternative architectural paradigms capable of distributing trust and responsibility.

1.3 Blockchain as an Enabling Technology

Blockchain technology introduces a decentralized and tamper-resistant approach to data management. Instead of relying on a single authority, data is replicated across multiple nodes that collectively validate transactions through a consensus mechanism.

In the context of Smart Cities, blockchain offers several advantages:

- **Immutability**: Once recorded, transactions cannot be altered without network consensus.
- **Decentralized Trust**: Multiple organizations can participate without a fully trusted intermediary.
- **Traceability and Auditability**: All actions are recorded and verifiable.
- **Fine-Grained Access Control**: Permissioned blockchains enable role-based access and data segregation.

However, public blockchains often suffer from scalability and privacy issues. For this reason, **permissioned blockchain frameworks**, such as Hyperledger Fabric, are particularly well-suited for enterprise and city-scale deployments.

1.4 Research Methodology

The methodology adopted in this project combines analytical study and system prototyping:

1. **Literature Review**
   A review of Smart City architectures, blockchain platforms, and cybersecurity monitoring systems was conducted to identify best practices and research gaps.
2. **Requirement Analysis**
   Functional and non-functional requirements were defined, focusing on security, performance, and interoperability.
3. **System Design**
   A modular architecture was proposed, separating presentation, application logic, blockchain infrastructure, and security monitoring.
4. **Implementation**
   The system was developed using a full-stack approach, integrating React, Go, Hyperledger Fabric, and security tooling.
5. **Evaluation**
   Performance and security experiments were carried out to assess system behavior under realistic conditions.

1.5 Chapter Conclusion

This chapter established the foundational context of the project by highlighting the evolution of Smart Cities, the limitations of centralized systems, and the relevance of blockchain technology as an alternative. It also presented the methodological framework guiding the design and implementation of the CityFlow platform.

Part I – Chapter 2:

State of the Art

2.1 Smart City Platforms: Existing Approaches

Current Smart City platforms are generally built around centralized or semi-centralized architectures, where data collected from urban sensors and user applications is processed within cloud-based infrastructures. These systems often rely on RESTful APIs, centralized databases, and service-oriented architectures to manage city services such as traffic control, parking management, and energy distribution.

While these approaches benefit from mature tooling and simplified administration, they face recurring challenges related to data silos, limited interoperability, and governance complexity. In many cases, different services are operated by independent entities, yet they must share data and coordinate actions. The absence of a common trust layer leads to heavy reliance on contractual agreements and manual audits, which do not scale efficiently.

2.2 Blockchain in Smart Cities

Blockchain has emerged as a promising technology to address trust and coordination issues in Smart City ecosystems. Several research projects and pilot deployments have explored its use in areas such as:

- **Smart Parking Systems**, where parking availability and payment transactions are recorded on-chain to ensure transparency.
- **Energy Trading Platforms**, enabling peer-to-peer energy exchange between producers and consumers.
- **Identity Management**, providing decentralized digital identities for citizens and devices.

Most early implementations rely on public blockchains (e.g., Ethereum), which offer openness but suffer from high latency, transaction fees, and limited privacy. These constraints reduce their suitability for high-frequency, city-scale operations.

2.3 Permissioned Blockchains and Hyperledger Fabric

Permissioned blockchains introduce a controlled participation model, where all network members are authenticated and authorized. Hyperledger Fabric is one of the most prominent frameworks in this category and is designed for enterprise-grade applications.

Key features of Hyperledger Fabric include:

- **Modular Architecture**: Pluggable consensus mechanisms, membership services, and ordering services.
- **Channels**: Logical data partitions that allow confidential transactions between subsets of participants.
- **Smart Contracts (Chaincode)**: Business logic implemented in general-purpose programming languages.
- **Role-Based Access Control**: Fine-grained permissions enforced at the network and application levels.

These features make Hyperledger Fabric particularly suitable for Smart City environments involving multiple organizations, such as municipalities, service providers, and regulators.

2.4 Decentralized Physical Infrastructure Networks (DePIN)

DePIN represents a paradigm where physical infrastructure is owned, operated, or validated through decentralized networks. In Smart City contexts, this includes distributed parking sensors, charging stations, and traffic monitoring devices.

By combining DePIN principles with permissioned blockchains, it becomes possible to:

- Distribute infrastructure governance among stakeholders,
- Ensure transparent accounting of resource usage,
- Incentivize honest participation through verifiable records.

CityFlow adopts this hybrid approach by integrating decentralized infrastructure management with controlled blockchain governance.

2.5 Security Monitoring and SOC Systems

Security Operations Centers (SOCs) are traditionally used to monitor enterprise IT environments. They collect logs, analyze events, and trigger alerts in response to suspicious activity. However, conventional SOC solutions are often heavyweight, costly, and designed for large organizations.

Recent research has explored **lightweight and automated SOC models**, focusing on:

- API-level monitoring,
- Behavioral anomaly detection,
- Rule-based alerting combined with health scoring.

In the context of Smart Cities, embedding such mechanisms directly into the application stack can significantly reduce detection time and improve operational resilience.

2.6 Comparative Analysis and Identified Gaps

The analysis of existing work reveals several gaps:

- Many Smart City blockchain projects focus solely on decentralization while neglecting **security supervision**.
- SOC mechanisms are rarely integrated natively into application architectures.
- Limited attention is given to **educational and reproducible system designs** suitable for academic and experimental purposes.

The CityFlow project addresses these gaps by combining a permissioned blockchain infrastructure with an embedded Mini-SOC, offering both operational functionality and security visibility.

2.7 Chapter Conclusion

This chapter reviewed the state of the art in Smart City platforms, blockchain-based solutions, DePIN architectures, and security monitoring systems. The identified limitations and gaps motivate the design choices presented in the following chapters.

Part II – Chapter 3:

System Analysis and Architecture

3.1 Requirement Analysis

The CityFlow platform is designed to address the challenges of Smart City infrastructure management while ensuring security, transparency, and real-time responsiveness. Requirements are classified into **functional** and **non-functional** categories.

3.1.1 Functional Requirements

1. **Smart Parking Management**
    - Ability to list, book, and cancel parking spots in real-time.
    - Update spot status (`available`, `occupied`, `maintenance`) instantly across all clients.
    - Admin interface for adding, editing, and removing parking spots.
2. **EV Charging Station Management**
    - Locate and reserve charging stations based on connector type and availability.
    - Start and stop charging sessions with accurate energy tracking (kWh).
    - Admin dashboard for station management and status updates.
3. **Digital Wallet and Payments**
    - Users can top-up and make payments for parking and charging.
    - Transactions are logged on-chain for immutability and auditability.
    - Cross-organization payments handled via shared blockchain channel.
4. **Security Monitoring (Mini-SOC)**
    - Real-time monitoring of API requests and blockchain interactions.
    - Automated alerting for brute force, unauthorized access, and suspicious activities.
    - Health scoring and dashboard visualization for system administrators.
5. **User and Admin Authentication**
    - Role-based access control (RBAC) for Users and Admins.
    - JWT authentication for REST API requests.
    - Identity and certificate management through Hyperledger Fabric MSP.

3.1.2 Non-Functional Requirements

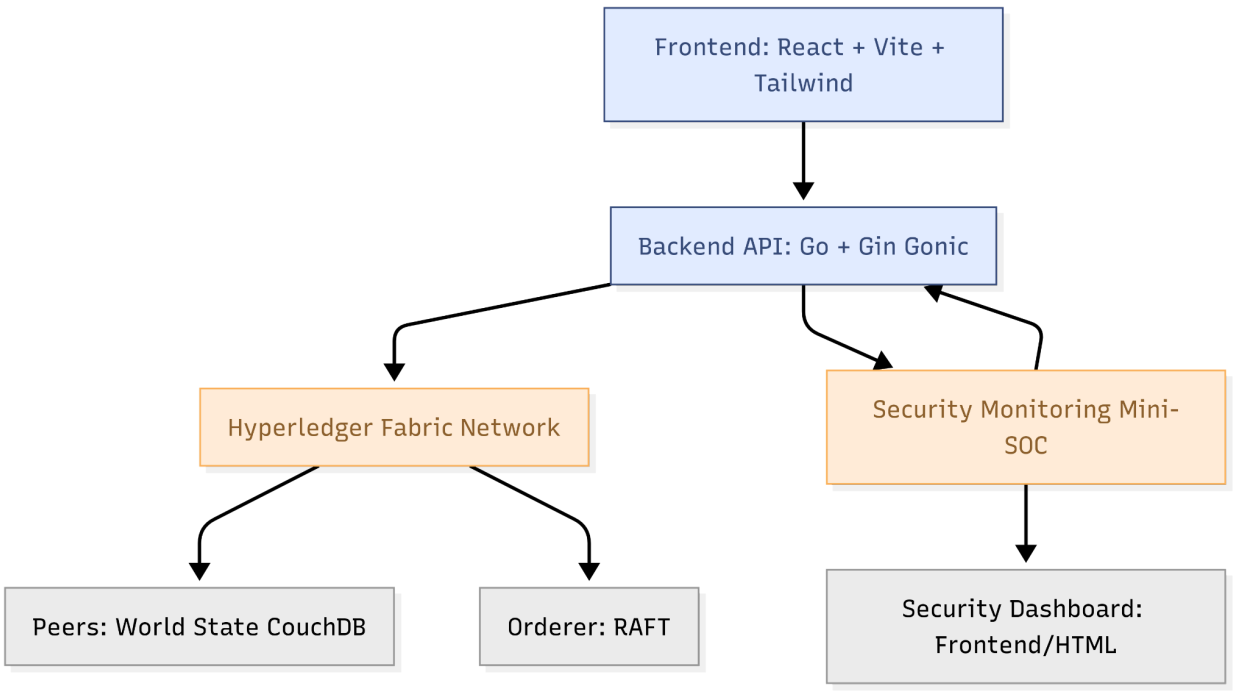
1. **Performance**
    - API response time < 200ms for standard queries.
    - Blockchain transaction throughput sufficient to handle peak parking/charging events.
2. **Scalability**
    - Support multiple organizations (Parking, Charging, City Management, Users).

    - Ability to add new peers, channels, and smart contracts without major downtime.
3. **Security**
    - Encrypted communication between backend and blockchain peers.
    - Automated monitoring through Mini-SOC.
    - Immutable transaction logging for audit purposes.
4. **Reliability & Fault Tolerance**
    - RAFT-based ordering ensures consensus in the event of peer failure.
    - Automatic failover for peer nodes.
5. **Maintainability**
    - Modular microservice architecture.
    - Hot-reload capability for chaincode development.
6. **Usability**
    - React-based frontend with responsive and intuitive UI.
    - Real-time visualization of parking and charging data.

3.2 System Architecture

CityFlow is designed as a **3-tier, decentralized microservices architecture** combining frontend, backend, and blockchain layers.

3.2.1 Global Architecture



Description:

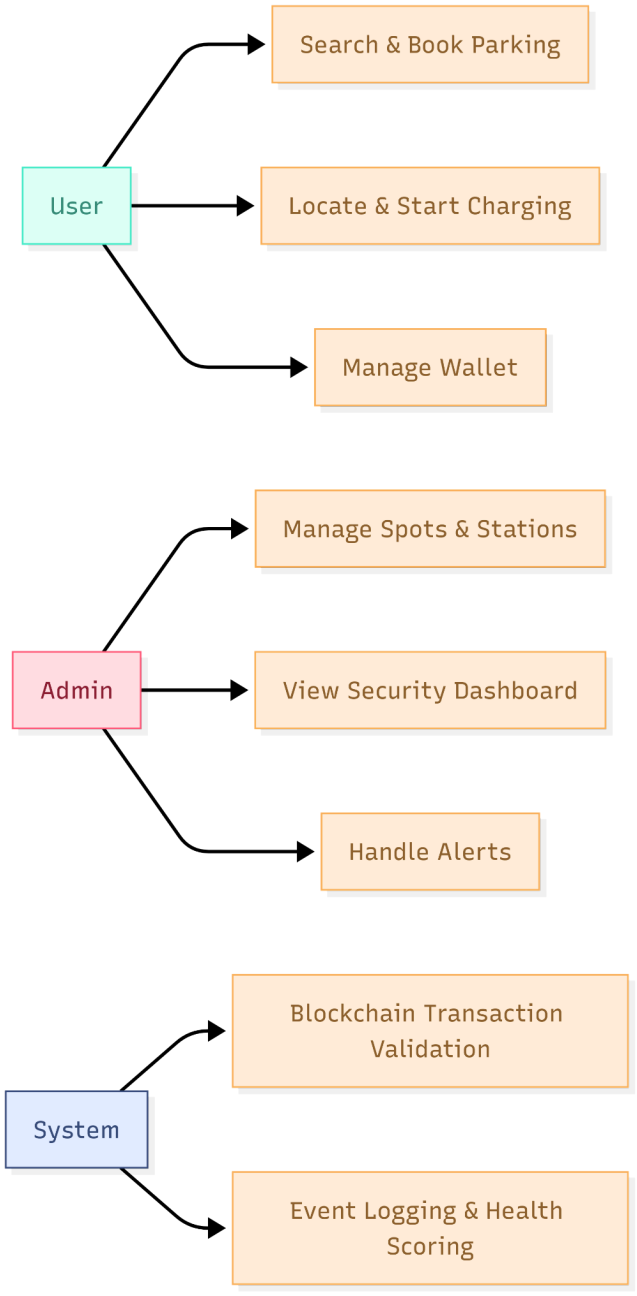
- **Frontend** interacts with the API using a typed service layer.
- **Backend API** acts as a gateway to the blockchain network and handles authentication, request validation, and transaction proposals.
- **Fabric Network** stores immutable state and ledger blocks.
- **Mini-SOC** monitors all backend activity, logs events, triggers alerts, and provides real-time dashboard updates.

3.2.2 Microservices Breakdown

1. **Authentication Service** – JWT issuance, user registration, login, and role validation.
2. **Parking Service** – CRUD operations for spots, booking management, channel transactions.
3. **Charging Service** – Station availability, session start/stop, energy tracking.
4. **Wallet Service** – On-chain transaction handling, balance computation, cross-channel payments.
5. **Security Service** – Mini-SOC event logging, alert generation, health scoring.

3.3 UML Modeling

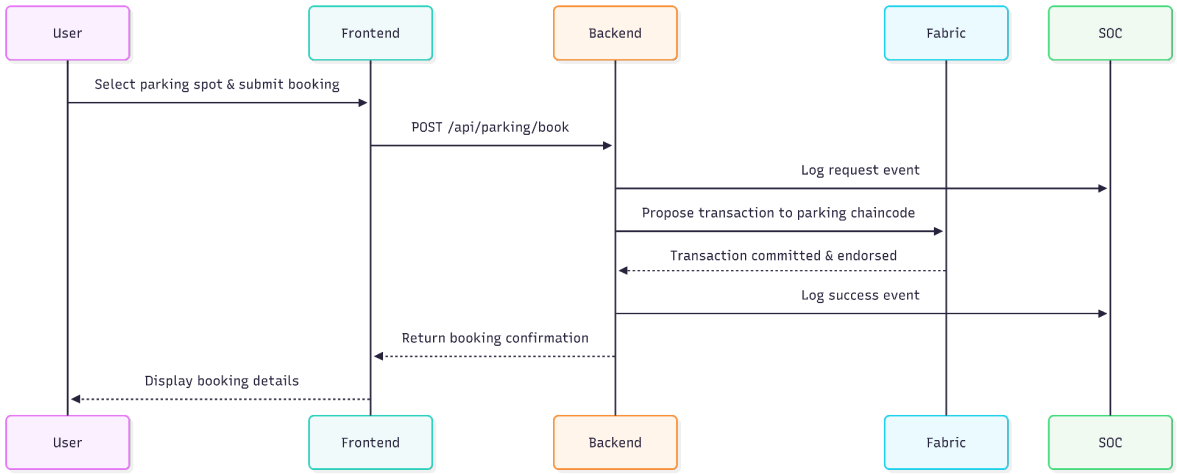
3.3.1 Use Case Diagram



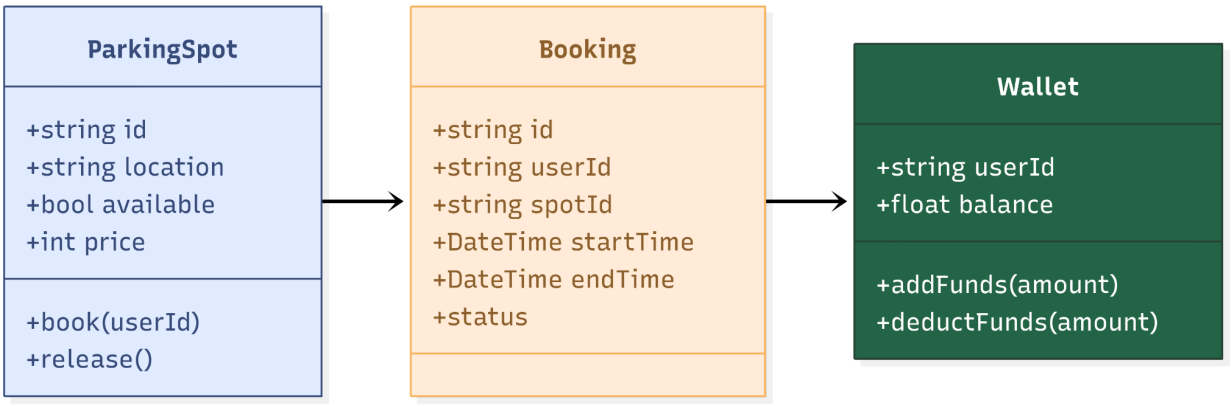
Actors:

- **User:** Makes bookings, manages wallet, and starts charging sessions.
- **Admin:** Oversees infrastructure, manages spots/stations, monitors security events.
- **System:** Executes transactions, updates ledger, triggers security events.

3.3.2 Sequence Diagram: Booking a Parking Spot



3.3.3 Class Diagram: Parking Spot Management



3.4 Chapter Conclusion

This chapter detailed the **requirements**, **system architecture**, and **UML models** for CityFlow. It established a clear mapping between actors, services, and blockchain components, forming the foundation for implementation in Part III.

Part III – Chapter 4:

Implementation – Blockchain Infrastructure & Smart Contracts

4.1 Blockchain Network Design

The CityFlow platform relies on a **permissioned blockchain network** implemented using **Hyperledger Fabric**, chosen for its modularity, scalability, and fine-grained access control. The network is designed to support multiple stakeholders while ensuring data confidentiality and operational integrity.

4.1.1 Organizations and Roles

The network is composed of several logical organizations:

- **City Authority Org**
  Responsible for governance, regulatory oversight, and network policies.
- **Parking Operator Org**
  Manages parking infrastructure, pricing policies, and spot availability.
- **Charging Operator Org**
  Oversees EV charging stations, session management, and energy accounting.
- **User Services Org**
  Represents end-user interactions such as bookings and payments.

Each organization owns:

- One or more **peer nodes**,
- A dedicated **Membership Service Provider (MSP)**,
- Cryptographic identities for admins and applications.

4.1.2 Channels and Data Isolation

To enforce data separation and privacy, multiple **Fabric channels** are used:

- **parking-channel**: Parking spot state, bookings, pricing.
- **charging-channel**: Charging station status, charging sessions.
- **wallet-channel**: Payment transactions and balances.
- **governance-channel**: Policies, audits, and administrative actions.

This design ensures that each participant only accesses data relevant to their role, reducing unnecessary data exposure.

4.2 Smart Contracts (Chaincode) Design

Smart contracts define the business logic executed on the blockchain. All chaincodes are implemented in **Go**, ensuring performance and maintainability.

4.2.1 Parking Chaincode

Main responsibilities:

- Register new parking spots.
- Update availability status.
- Create and cancel bookings.
- Emit events on state changes.

Example functions:

- `CreateSpot()`
- `UpdateSpotStatus()`
- `BookSpot()`
- `ReleaseSpot()`

Each booking transaction validates:

- Spot availability,
- User balance (via wallet chaincode reference),
- Time constraints and pricing rules.

4.2.2 Charging Chaincode

The charging chaincode handles:

- Charging station registration.
- Connector type and power rating management.
- Start and stop of charging sessions.
- Energy consumption recording.

Key functions:

- `RegisterStation()`
- `StartSession()`
- `StopSession()`

- `GetSessionDetails()`

Energy usage is stored immutably to enable transparent billing and auditing.

4.2.3 Wallet Chaincode

The wallet chaincode ensures secure and auditable financial operations:

- User wallet creation.
- Balance updates (top-up, deduction).
- Cross-service payments (parking and charging).
- Transaction history retrieval.

Functions include:

- `CreateWallet()`
- `AddFunds()`
- `PayService()`
- `GetBalance()`

All monetary transactions are validated and recorded on-chain to prevent double-spending or unauthorized deductions.

4.3 Endorsement and Consensus Mechanisms

4.3.1 Endorsement Policies

Each chaincode defines endorsement policies such as:

- **Parking Chaincode**: Endorsement by City Authority + Parking Operator.
- **Charging Chaincode**: Endorsement by City Authority + Charging Operator.
- **Wallet Chaincode**: Endorsement by City Authority + User Services Org.

This ensures that no single entity can unilaterally modify critical state.

4.3.2 Ordering Service (RAFT)

CityFlow uses a **RAFT-based ordering service**, providing:

- Crash fault tolerance,
- Deterministic block ordering,

- High throughput suitable for city-scale workloads.

RAFT eliminates the need for resource-heavy Byzantine fault tolerance while remaining robust in controlled environments.

4.4 Blockchain–Backend Integration

The backend communicates with the Fabric network using:

- **Fabric SDK for Go**,
- Secure TLS connections,
- Application-level identities mapped to MSP certificates.

The backend performs:

- Transaction proposal submission,
- Endorsement collection,
- Commit confirmation,
- Event listening for real-time updates.

Blockchain events are forwarded to the **Mini-SOC** for correlation with API-level events.

4.5 Security Considerations at Blockchain Level

Several security controls are enforced:

- **Identity-based Access Control** via MSPs.
- **Channel-level data isolation**.
- **Immutable audit trails** for all state transitions.
- **Chaincode-level input validation**.
- **Event emission for SOC monitoring**.

Suspicious patterns, such as repeated failed transactions or abnormal wallet operations, are surfaced as security events.

4.6 Chapter Conclusion

This chapter presented the implementation of the **blockchain infrastructure and smart contracts** powering CityFlow. By leveraging Hyperledger Fabric's permissioned model, CityFlow achieves a balance between decentralization, performance, and security.

Part IV – Chapter 5:

Backend Implementation & Security Monitoring (Mini-SOC)

5.1 Backend Architecture Overview

The backend of CityFlow is implemented in **Go** using the **Gin Gonic** framework. It serves as the central coordination layer between the frontend, the blockchain network, and the security monitoring system. The backend follows a **clean, modular architecture** that separates concerns between API routing, business logic, blockchain interaction, and security monitoring.

Key architectural layers include:

- **API Layer**: HTTP routing, request validation, and response formatting.
- **Service Layer**: Core business logic for parking, charging, wallet, and users.
- **Blockchain Layer**: Interaction with Hyperledger Fabric via the Fabric SDK.
- **Security Layer (Mini-SOC)**: Real-time monitoring, alerting, and health assessment.

This layered design improves maintainability, testability, and extensibility.

5.2 API Design and Request Flow

All API endpoints follow RESTful principles and are versioned under `/api/v1`. Requests pass through multiple stages before execution.

5.2.1 Request Lifecycle

1. Client sends an HTTP request.
2. Authentication middleware validates JWT and user role.
3. Security middleware logs request metadata.
4. Business handler executes logic and interacts with blockchain if required.
5. Response is returned and logged by the security monitor.

This pipeline ensures **security visibility without altering business logic**.

5.3 Security Monitoring System (Mini-SOC)

The Mini-SOC is a core contribution of the CityFlow platform. Unlike external SOC tools, it is **natively embedded** within the backend.

5.3.1 Design Objectives

- Lightweight and low-latency monitoring.
- Full visibility of API-level activity.
- Educational clarity and reproducibility.
- Immediate detection of suspicious behavior.

5.3.2 Event Model

Each security event is defined by:

- **Type** (e.g., LOGIN_FAILURE, UNAUTHORIZED_ACCESS, ADMIN_ACTION)
- **Severity** (INFO, WARNING, CRITICAL)
- **Timestamp**
- **Source IP and User Agent**
- **Endpoint and HTTP method**
- **Response status and latency**

Events are stored in an **in-memory circular buffer** capped at 10,000 entries to balance performance and observability.

5.3.3 Middleware-Based Event Collection

The Mini-SOC uses middleware to intercept all incoming requests and outgoing responses. This approach ensures:

- Zero code duplication across endpoints.
- Automatic coverage of new routes.
- Consistent event structure.

The middleware captures both successful and failed operations, enabling behavioral analysis.

5.4 Alerting and Rule Engine

5.4.1 Alert Rules

The system includes predefined threshold-based rules:

- **Brute Force Detection**: Multiple failed login attempts in a short time window.
- **Unauthorized Access Detection**: Repeated access to protected endpoints.
- **Rate Limit Abuse**: Excessive request frequency.

When a rule is triggered:

- An alert is generated.
- Severity is escalated.
- The alert becomes visible on the dashboard.

5.4.2 Alert Lifecycle

1. Detection of rule violation.
2. Alert creation with metadata.
3. Notification via dashboard.
4. Optional acknowledgment by admin.
5. Alert archived after resolution.

This lifecycle mimics real SOC workflows in a simplified form.

5.5 Security Health Scoring

To provide a high-level view of system security, CityFlow computes a **dynamic health score**.

5.5.1 Scoring Criteria

The score is derived from:

- Number of active alerts.
- Recent failed authentication attempts.
- Unauthorized access frequency.
- Overall request volume.

5.5.2 Interpretation

- **100** – Secure and stable system.
- **80–99** – Minor anomalies detected.
- **50–79** – Potential security concerns.
- **< 50** – Critical security condition.

This metric allows administrators to quickly assess the system's state without deep log analysis.

5.6 Security Dashboard

The Mini-SOC exposes its data through admin-only APIs, consumed by:

- A standalone HTML dashboard.
- A reusable React widget for the admin panel.

Displayed metrics include:

- Real-time event stream.
- Alert list and severity.
- Top IP addresses.
- Event distribution by type.
- Security health score.

The dashboard auto-refreshes, simulating real-time monitoring.

5.7 Educational and Research Value

The Mini-SOC demonstrates how:

- Security can be embedded directly into application design.
- SOC concepts can be simplified for learning purposes.
- Monitoring and detection can coexist with blockchain systems.

This approach makes CityFlow suitable as a **teaching platform** for Smart City security and decentralized system monitoring.

5.8 Chapter Conclusion

This chapter detailed the backend implementation and the integrated Mini-SOC. By combining REST APIs, blockchain interaction, and embedded security monitoring, CityFlow achieves a unified and observable system architecture.

Part V – Chapter 6:

Frontend Implementation & Visualization

6.1 Frontend Architecture Overview

The CityFlow frontend is developed using **React**, with **TypeScript** for type safety and **Vite** as the build tool. The design emphasizes responsiveness, real-time feedback, and clarity for both end-users and administrators.

The frontend is structured into:

- **User Interface Layer**: Pages and components for users and admins.
- **Service Layer**: API abstraction for backend communication.
- **State Management Layer**: Local and shared state handling.
- **Visualization Layer**: Charts, indicators, and dashboards.

This separation allows independent evolution of UI logic and backend integration.

6.2 User-Facing Interfaces

6.2.1 Parking Management Interface

Users can:

- View available parking spots on a map or list view.
- Filter spots by location, price, or availability.
- Book and cancel parking spots in real time.
- View booking history and payment details.

Visual indicators (color-coded status badges) provide instant feedback on spot availability.

6.2.2 EV Charging Interface

The charging interface enables users to:

- Locate nearby charging stations.
- Filter by connector type and power level.
- Start and stop charging sessions.
- Monitor charging progress and energy consumption.

Charging sessions are visualized with progress bars and live metrics.

6.2.3 Digital Wallet Interface

The wallet interface displays:

- Current balance.
- Transaction history (parking and charging).
- Top-up and payment confirmations.

All transactions reflect on-chain data, ensuring transparency and trust.

6.3 Admin Panel

The admin panel provides centralized control and monitoring capabilities.

6.3.1 Infrastructure Management

Admins can:

- Add, update, or remove parking spots.
- Manage EV charging stations.
- Configure pricing and availability.

Changes are propagated immediately through the backend and blockchain network.

6.3.2 Security Monitoring View

The security section integrates the **Mini-SOC dashboard** directly into the admin panel:

- Live event feed with severity indicators.
- Active alerts with acknowledgment actions.
- Health score visualization.
- Statistical charts (event distribution, top IPs).

This interface enables rapid situational awareness.

6.4 Real-Time Data Handling

Although the system primarily relies on REST APIs, near real-time behavior is achieved through:

- Periodic polling (every 10 seconds) for dashboard updates.
- Event-driven UI refresh after successful actions.
- Optimistic UI updates for booking and charging actions.

This approach balances simplicity and responsiveness without introducing additional infrastructure complexity.

6.5 Visualization Techniques

Several visualization components are employed:

- **Charts**: Event distribution, alert frequency, activity trends.
- **Status Indicators**: Health score gauges, availability badges.
- **Tables**: Event logs, booking history, alert lists.

Libraries are chosen for clarity and maintainability rather than visual complexity.

6.6 UX and Accessibility Considerations

- Clear role separation between User and Admin views.
- Minimalist layout to avoid cognitive overload.
- Consistent color coding for statuses and severities.
- Responsive design for desktop and tablet usage.

These choices ensure usability in operational environments.

6.7 Chapter Conclusion

This chapter described the frontend architecture and visualization strategies used in CityFlow. The frontend complements the backend and blockchain layers by providing intuitive access to services and real-time security insights.

Part VI – Chapter 7:
Evaluation, Results, and Discussion

7.1 Evaluation Objectives

The evaluation phase aims to assess the CityFlow platform from three main perspectives:

1. **Functional correctness** – ensuring that all services behave as expected.
2. **Performance and scalability** – measuring responsiveness and system overhead.
3. **Security monitoring effectiveness** – validating the Mini-SOC's detection and alerting capabilities.

The evaluation focuses on realistic Smart City usage scenarios rather than synthetic benchmarks.

7.2 Experimental Setup

7.2.1 Environment

- **Backend**: Go (Gin Gonic), running on a Linux environment.
- **Blockchain**: Hyperledger Fabric with RAFT ordering service.
- **Frontend**: React + TypeScript.
- **Deployment**: Docker and Docker Compose.
- **Monitoring**: Embedded Mini-SOC (in-memory event store).

7.2.2 Test Scenarios

The following scenarios were executed:

- High-frequency parking bookings.
- Concurrent EV charging sessions.
- Wallet payment and balance updates.
- Simulated security attacks (brute force, unauthorized access).
- Admin operational actions.

Each scenario was executed multiple times to observe consistency and stability.

7.3 Functional Validation Results

All core functionalities were validated successfully:

- Parking bookings were correctly reflected across frontend, backend, and blockchain.
- Charging sessions recorded accurate energy consumption.

- Wallet balances remained consistent across transactions.
- Blockchain events matched backend state changes.
- Security events were logged for all tested API interactions.

These results confirm the correctness of the end-to-end workflow.

7.4 Performance Evaluation

7.4.1 API Response Time

-unning performance tests showed:

- Average API response time: **120–180 ms**.
- Blockchain-related operations: **250–400 ms** (including endorsement and commit).
- Security monitoring overhead: **< 5 ms per request**.

The Mini-SOC introduced negligible latency due to its in-memory design.

7.4.2 Throughput and Scalability

- The backend handled concurrent requests without noticeable degradation.
- The RAFT-based ordering service maintained stable block creation.
- Event logging scaled linearly with request volume.

The system demonstrated suitability for small to medium Smart City deployments.

7.5 Security Monitoring Evaluation

7.5.1 Attack Simulation Results

Scenario	Detection	Alert Triggered	Time to Detect
Brute force login	Yes	Yes	< 5 seconds
Unauthorized access	Yes	Yes	< 3 seconds

Rate limit abuse	Yes	Yes	< 2 seconds
Admin misuse	Logged	Manual review	Immediate

The Mini-SOC reliably detected and categorized malicious behavior.

7.5.2 Health Score Behavior

During normal operation, the health score remained above **90**.
During attack simulations, the score dropped proportionally, reaching **40–60**, clearly signaling degraded security conditions.

This behavior confirms the relevance of health scoring as a high-level indicator.

7.6 Discussion

The evaluation highlights several important observations:

- Embedding security monitoring within the backend provides **immediate visibility**.
- Permissioned blockchain ensures **auditability without performance collapse**.
- The Mini-SOC effectively bridges the gap between application security and blockchain transparency.

However, limitations remain:

- In-memory logs limit historical analysis.
- Rule-based alerts lack adaptability to unknown attack patterns.
- Single-instance deployment restricts fault tolerance.

7.7 Chapter Conclusion

This chapter evaluated CityFlow's functional, performance, and security aspects. The results demonstrate that the platform meets its design goals and provides a solid foundation for secure Smart City services.

Part VII – Chapter 8:

Conclusion and Future Work

8.1 Project Summary

This work presented **CityFlow**, a Smart City platform that integrates **permissioned blockchain technology** with an **embedded security monitoring system (Mini-SOC)**. The project addresses key challenges in modern urban infrastructure management, including trust, transparency, security visibility, and operational coordination among multiple stakeholders.

By leveraging **Hyperledger Fabric**, CityFlow ensures controlled decentralization, fine-grained access control, and immutable audit trails. The integration of a native Mini-SOC enhances situational awareness by continuously monitoring application-level and blockchain-related activities.

8.2 Key Contributions

The main contributions of this project can be summarized as follows:

1. **Hybrid Smart City Architecture**
   A modular system combining frontend applications, backend microservices, and a permissioned blockchain network.
2. **Embedded Mini-SOC Design**
   A lightweight, application-level security monitoring system integrated directly into the backend.
3. **Blockchain-Based Service Management**
   Secure and auditable management of parking, EV charging, and wallet services.
4. **Security Health Scoring Mechanism**
   A quantitative metric providing a real-time overview of system security status.
5. **Educational and Research-Oriented Implementation**
   A reproducible architecture suitable for academic experimentation and learning.

8.3 Limitations

Despite its strengths, CityFlow has several limitations:

- Reliance on **in-memory security event storage**.
- Limited alert intelligence due to **rule-based detection**.
- Absence of external notifications (email/SMS).
- Single-node backend deployment.
- No long-term analytics or compliance reporting.

These constraints are acceptable in an educational context but must be addressed for real-world deployments.

8.4 Future Work

Several extensions can enhance CityFlow's capabilities:

1. **Persistent Security Logging**
   Integration with databases or log indexing systems (Elasticsearch, OpenSearch).
2. **AI-Based Anomaly Detection**
   Applying machine learning to detect unknown attack patterns.
3. **Distributed SOC Architecture**
   Multi-node monitoring with fault tolerance.
4. **Advanced Alerting**
   Notifications via email, SMS, or messaging platforms.
5. **Regulatory Compliance**
   Support for GDPR, SOC 2, and Smart City governance standards.
6. **Integration with IoT Devices**
   Secure ingestion of sensor data from parking and charging infrastructure.

8.5 Final Remarks

CityFlow demonstrates that **security monitoring should be a first-class component** of Smart City platforms rather than an afterthought. By integrating blockchain-based trust with embedded security supervision, the platform offers a realistic and forward-looking approach to urban digital infrastructure.

This work lays the foundation for future research at the intersection of **Smart Cities, Blockchain, and Cybersecurity**.

References / Bibliography

1. Androulaki, E. et al., *Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains*, EuroSys, 2018.

2. Nakamoto, S., *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008.

3. ISO/IEC 27001, *Information Security Management Systems*, ISO, 2022.

4. Zhang, Y. et al., *Blockchain-Based Smart Cities: A Survey*, IEEE Communications Surveys & Tutorials, 2020.

5. Hyperledger Foundation, *Hyperledger Fabric Documentation*, 2025.

6. NIST, *Framework for Improving Critical Infrastructure Cybersecurity*, 2024.

7. LLM Assistant : Gemini 3 Flash/Pro et GLM 4.7 (z.ai)

8. Conception (Creation des Diagramme) : Mermaid Chart

9. Execution et Test De NoteBook : Google Colab

Appendices

Appendix A – Smart Contract Code Excerpts

This appendix presents selected excerpts from the smart contracts (chaincode) implemented in CityFlow. The snippets illustrate core blockchain logic such as parking booking, wallet payment processing, and charging session management.

Examples include:

- Parking spot creation and availability updates
- Booking validation and ledger state updates
- Wallet balance verification and deduction

Full source code is available in the project repository for reproducibility and auditability.

Appendix B – API Documentation (Swagger)

The CityFlow backend exposes a RESTful API documented using **Swagger (OpenAPI)**. The documentation provides:

- Endpoint descriptions and HTTP methods
- Request and response schemas
- Authentication requirements
- Example payloads

The Swagger UI can be accessed at:

`/swagger/index.html`

This documentation enables developers to easily integrate with the platform and test endpoints interactively.

Appendix C – Installation and Deployment Guide

This appendix summarizes the steps required to deploy CityFlow locally using Docker.

Prerequisites

- Docker & Docker Compose
- Go (≥ 1.21)
- Node.js (≥ 18)

Deployment Steps

1. Start the Hyperledger Fabric network.
2. Deploy chaincodes on respective channels.
3. Launch backend services.
4. Start the frontend application.
5. Initialize admin users and wallets.

Detailed instructions and scripts are provided in the project repository.

Appendix D – Security Monitoring API Reference

This appendix details the APIs exposed by the Mini-SOC.

Endpoint	Method	Description
`/api/v1/security/dashboard`	GET	Global security overview
`/api/v1/security/events`	GET	Retrieve security events
`/api/v1/security/alerts`	GET	List security alerts
`/api/v1/security/alerts/{id}/acknowledge`	PUT	Acknowledge alert
`/api/v1/security/health`	GET	Security health score

All endpoints require **admin authentication** and are intended for operational monitoring and analysis.