`Unosql`: A Secure NoSQL Database for Resource-Constrained Environments

Arman Ghobadi

June 14, 2025

Contents

Abstract

`Unosql` is a lightweight, secure NoSQL database designed for resource-constrained environments, such as those running `MicroPython` on microcontrollers. Optimized for critical industries, it integrates AES-CBC encryption, HMAC-SHA256 integrity checks, and compliance with MISRA and ISO 26262 standards. This article explores the architecture, design principles, and operational logic of `Unosql`, highlighting its features like atomic transactions, efficient indexing, and robust error recovery. We present diagrams illustrating the systems data flow and architecture, discuss its performance considerations, and evaluate its suitability for IoT and embedded systems.

1 Introduction

`Unosql` is engineered to address the need for secure, reliable data storage in resource-constrained environments, such as IoT devices and embedded systems running `MicroPython`. Unlike traditional databases, `Unosql` prioritizes minimal memory usage, robust security, and fault tolerance, making it suitable for critical applications in automotive, healthcare, and industrial sectors. Its compliance with MISRA and ISO 26262 standards ensures safety and reliability, while features like AES-CBC encryption and HMAC-SHA256 integrity checks provide strong data protection.

This article is structured as follows: Section 2 details the system architecture, Section 3 discusses design principles, Section 4 explores key features, Section 5 presents architectural diagrams, Section 6 evaluates performance, and Section 7 concludes with future directions.

2 System Architecture

`Unosql` employs a modular architecture optimized for microcontrollers with limited RAM and storage. The system comprises several core components:

- **Storage Engine**: Manages persistent data storage with encryption and compression.
- **Transaction Manager**: Ensures atomicity and consistency using a context manager.
- **Indexing Module**: Supports single and compound indexes for efficient queries.
- **Error Recovery**: Handles disk full, file corruption, and power failures.
- **Locking Mechanism**: Prevents concurrent access issues with stale lock detection.

The data flow begins with user queries, which are validated and processed through the transaction manager. Data is encrypted, compressed, and stored with checksums for integrity. Indexes are updated atomically to maintain query performance.

3 Design Principles

`Unosql` is built on the following principles:

1. **Security**: AES-CBC encryption and HMAC-SHA256 ensure data confidentiality and integrity.
2. **Fault Tolerance**: Atomic transactions and backups mitigate power failures and corruption.
3. **E**fficiency: Lazy loading and memory unloading optimize RAM usage.
4. **S**tandards Compliance: Adherence to MISRA and ISO 26262 ensures reliability in critical systems.
5. **S**implicity: A lightweight API simplifies integration into embedded applications.

The system handles edge cases such as disk full errors, file corruption, and concurrent access through robust error recovery and locking mechanisms.

4 Key Features

Unosql offers a rich set of features tailored for constrained environments:

- **Encryption and Integrity**: Uses AES-CBC with 256-bit keys and HMAC-SHA256 for secure storage.
- **Atomic Transactions**: Ensures data consistency with rollback capabilities.
- **Efficient Indexing**: Supports single and compound indexes for fast queries.
- **Error Recovery**: Recovers from corruption using backups and checksums.
- **Memory Optimization**: Unloads collections to minimize RAM usage.

Below is an example of using Unosql to insert and query data:

Listing 1: Example Usage of Unosql

```
db = Unosql("my_db", encryption_key=b"32_bytekey12345678901234567890l2")
db.insert("collection", {"name": "Alice", "value": 100})
results = db.find("collection", {"name": "Alice"})
```

5 Architectural Diagrams

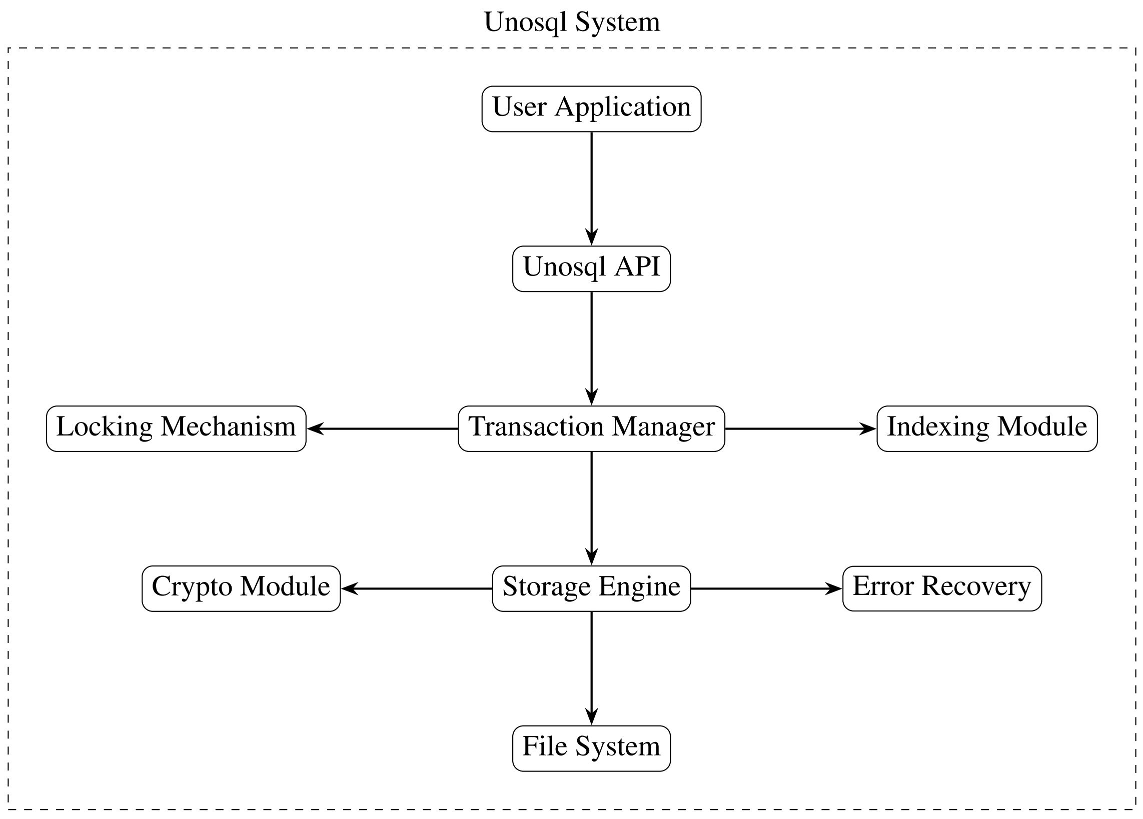

Figure 1: System Architecture of Unosql

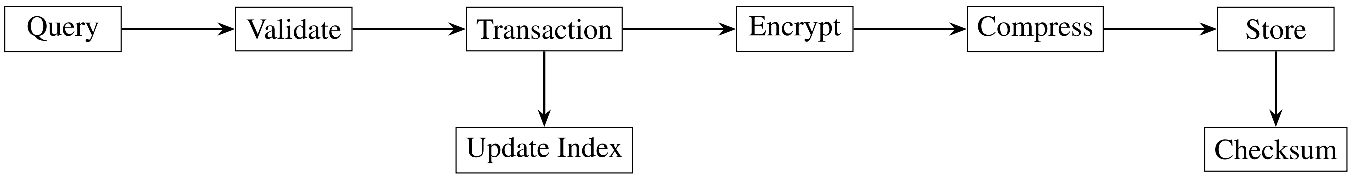

Figure 2: Data Flow in `Unosql`

Figure 1 illustrates the modular architecture, showing interactions between components like the storage engine and crypto module. Figure 2 depicts the data flow from query validation to storage, highlighting encryption and indexing steps.

6 Performance Considerations

`Unosql` is optimized for low-memory environments, with features like lazy loading and compression reducing resource usage. However, encryption and indexing may introduce overhead, particularly on devices with limited CPU power. The following table summarizes performance trade-offs:

Table 1: Performance Trade-offs in `Unosql`

Feature	**Benefit**	**Overhead**
Encryption	Data security	CPU usage
Compression	Reduced storage	CPU and memory
Indexing	Faster queries	Storage and update time
Atomic Transactions	Data consistency	I/O operations

`Unosql` is ideal for applications requiring secure, reliable data storage with minimal resources, such as IoT sensors and automotive control units.

7 Conclusion

`Unosql` provides a robust solution for secure data management in resource-constrained environments. Its architecture, combining encryption, atomic transactions, and efficient indexing, ensures reliability and performance. Compliance with MISRA and ISO 26262 makes it suitable for critical industries.

Future enhancements could include support for distributed systems, advanced query optimization, and integration with cloud-based IoT platforms.

References

1. MISRA Guidelines, `https://www.misra.org.uk`
2. ISO 26262 Standard, `https://www.iso.org`
3. `MicroPython` Documentation, `https://micropython.org`