big data and cognitive computing

MDPI

Article

Blockchains for Data Management: The DIGI4ECO Use Case and Practical Lessons Beyond Theory

Andreas Polyvios Delladetsimas, Elias Iosif *, Stamatis Papangelou and George Giaglis

Department of Digital Innovation, School of Business, University of Nicosia, 2417 Nicosia, Cyprus; delladetsimas.a@unic.ac.cy (A.P.D.); papangelou.m@unic.ac.cy (S.P.); giaglis.g@unic.ac.cy (G.G.)
* Correspondence: iosif.e@unic.ac.cy

Abstract

This article examines blockchain as an enabling technological component for data management tasks that are independent of currency-related functionality, a less-discussed aspect of a technology commonly associated with cryptocurrencies and decentralized finance (DeFi). Drawing on empirical findings from the DIGI4ECO project as a case study, we present a structured literature review and cross-domain analysis of blockchain-based data management systems (BDMSs), examine a representative permissioned BDMS implementation, and synthesize practical design guidelines and implementation insights for BDMS development. This perspective is motivated by core blockchain properties such as immutability and transparency, as well as by the observation that existing resources for BDMS development, including methods, tools, and best practices, remain fragmented and less developed than those available for more mature technologies.

Keywords: blockchain; blockchain-based data management systems; permissioned blockchains; blockchain architectures; hybrid storage; distributed data management

1. Introduction

Blockchain technology has emerged as a paradigm for secure and transparent data exchange. Although initially popularized by cryptocurrencies and DeFi [1,2], its application to data management has attracted growing attention in research and industry [3–5]. This has led to the development of Blockchain-based Data Management Systems (BDMSs), which combine blockchain with database technologies to ensure data integrity and traceability, particularly in large-scale, multi-source implementations.

BDMSs have been explored across domains such as healthcare, the Internet of Things (IoT), and personal data management. In this context, we focus on DIGI4ECO (Digital Twin-Sustainable 4D Ecological Monitoring of Restoration in Fishery-Depleted Areas), a European Commission-funded project (2024–2028) that integrates heterogeneous data sources within a digital twin ecosystem for marine monitoring [6]. In this setting, the blockchain component is used for data verification, and this paper analyzes its architecture and design as a representative BDMS implementation.

Common design choices in BDMSs include hybrid architectures combining on-chain and off-chain storage, the use of permissioned platforms such as Hyperledger Fabric (HLF) [7] and private or consortium-based Ethereum deployments [8]. Lightweight consensus mechanisms are also adopted to balance security and performance. Based on our interpretation of the reviewed literature, most implementations remain at intermediate Technology Readiness Levels (TRLs) [9].

Academic Editor: Giuseppe Maria Luigi Sarnè

Received: 3 April 2026
Revised: 11 May 2026
Accepted: 14 May 2026
Published: 18 May 2026

Against this background, we identify a lack of practical guidance for the development and deployment of BDMSs. The main contributions of this paper are as follows. First, a structured literature review and cross-domain analysis of BDMS architectures, consolidating recurring design patterns across application domains. Second, a representative BDMS case study based on the DIGI4ECO project, covering architecture, implementation, and performance evaluation. Third, practical design guidelines and implementation insights for BDMS development, complemented by an overview of relevant development tools and platforms. Together, these contributions are intended to support practitioners and researchers in navigating the design and deployment of blockchain-based data management components, rather than proposing an architecture that advances the state-of-the-art.

The remainder of this paper is structured as follows. Section 2 introduces DIGI4ECO. Section 3 reviews BDMS literature and compares existing approaches. Section 4 summarizes developer tools and platforms for smart contracts and blockchain applications. Section 5 presents the high-level system architecture of DIGI4ECO. Sections 6 and 7 discuss setup, deployment, operational considerations, and performance evaluation. Section 8 presents a discussion of the findings and concludes the paper.

2. Background: The DIGI4ECO Project

DIGI4ECO leverages blockchain for data verification, alongside digital technologies for real-time ecological and marine monitoring [6]. The project collects data from four fishery-depleted areas—OBSEA (Spain), Ancona (PLACE observatory, Italy), Galway (SmartBay, Ireland) and Kristineberg/Gullmarsfjorden (Sweden)—and integrates it into a digital twin platform for monitoring and analysis. These inputs are gathered through networks of robotic platforms, sensors, and video systems, including cabled observatories and mobile platforms such as seafloor crawlers, Autonomous Surface Vehicles (ASVs), and Autonomous Underwater Vehicles (AUVs). As Physical Twins, these platforms collect omics data (eDNA), image-derived biological information, and environmental monitoring data.

Once collected, data are standardized through integration with libraries and repositories (e.g., FathomNet, EMODnet, Copernicus) as well as genetic sequence databases. They are subsequently combined with historical scientific and socioeconomic records, deriving from both online repositories and previously non-digitized “sleeping” data. The resulting datasets are processed using artificial intelligence (AI) and machine learning (ML) tools to extract ecological, biological, and socioeconomic insights, which are then incorporated into Distributed Twin Observatories (DTOs) to support monitoring and decision-making in marine areas [6].

DIGI4ECO integrates IoT devices for environmental data collection, AI and ML techniques for analyzing imagery and other datasets, blockchain for security and traceability, and big data analytics for processing large datasets. Figure 1 illustrates how data flow through the system, highlighting the interaction between the blockchain and big data components.

Blockchain integration presents unique challenges in marine blockchain–IoT (BIoT) applications, commonly referred to as the Marine Internet of Things (MIoT) or the Internet of Underwater Things (IoUT). These challenges stem both from the demanding nature of underwater settings and the complexity of integrating blockchain into these systems. On the marine side, issues include low data rates, high latency, limited power and storage resources of deployed devices, and poor standardization and interoperability. Blockchain, in turn, introduces additional challenges, such as compatibility issues, high energy consumption, and costly migration from legacy systems [10]. To address these concerns, lightweight

consensus mechanisms are increasingly explored. Section 5 provides further details on the DIGI4ECO blockchain architecture.

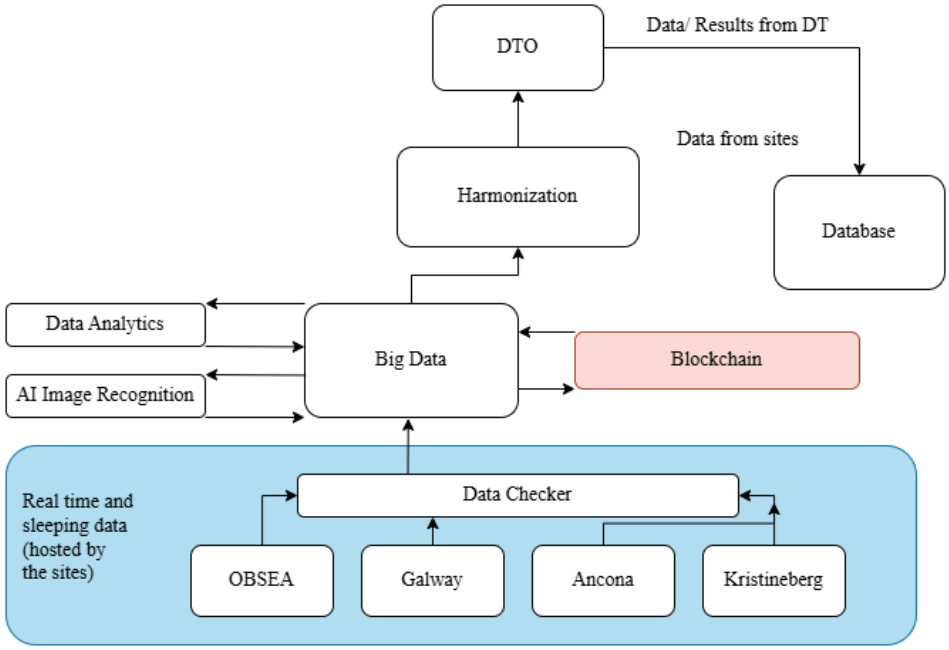

Figure 1. DIGI4ECO system overview.

In summary, DIGI4ECO follows a stepwise approach to collecting, processing, and integrating ecological data, combining multiple digital technologies to support monitoring and decision-making. While developed for marine environments, the architectural patterns and design choices presented here are applicable to BDMS implementations in other large-scale contexts [6].

3. Literature Review

This section reviews BDMSs from an architectural perspective, outlining their structural layers and data organization, followed by an examination of permissioned implementations and a comparison with conventional database systems.

3.1. Architectural Foundations of BDMSs

Before addressing use cases and implementations, the underlying architecture of a BDMS is examined. Paik et al. [3] distinguish two main categories of blockchain data storage, each with distinct trade-offs in performance, cost, and privacy. The first category, fully on-chain storage, maintains all data directly on the blockchain. The second category, hybrid storage (on-chain and off-chain), combines blockchain and external databases by storing large and personal data off-chain, while keeping key records or references of them (data pointers) on-chain, thereby balancing efficiency and security. In the context of such hybrid systems, Truong et al. [11] identify three main types of off-chain storage: conventional databases, cloud storage services, and distributed storage systems.

Paik et al. [3] further propose a conceptual framework that identifies four common architectural layers of blockchain-based applications and interprets each layer from a data management perspective, drawing parallels with conventional database systems.

The logical data layer defines how applications represent and interact with on-chain information. While conventional databases expose data through query interfaces over predefined schemas, blockchain systems manage assets (states), such as tokens or digital records, and smart contracts that automate actions under specified conditions. At this

layer, blockchains typically store data as simple key–value pairs or documents representing accounts, assets, and contract states.The physical data layer specifies how data is structured and organized. Instead of indexing records, blockchains maintain transactions in the form of blocks and a ledger that represents the complete chronological history of the system. The internal structures used to organize this data vary across implementations. The data access layer defines how operations interact with the stored data. Conventional database systems support CRUD (Create, Read, Update, Delete) operations via SQL. In contrast, blockchains express create and update operations as transactions that modify their state. In order to preserve immutability, delete operations are not supported; instead, assets or states are invalidated or replaced through new transactions. Furthermore, read operations are typically slower, since blockchains do not support direct queries and offer limited read access compared to conventional databases. Finally, the data processing layer coordinates transaction execution and validation. In blockchain systems, this is achieved through consensus protocols that establish transaction ordering and finality.

While Paik et al. [3] focus on the conceptual structure of blockchain data management, Wei et al. [4] examine its practical implementation. They propose a four-level data management stack, with each level addressing different aspects of data organization. At the top are blockchain-assisted databases, which operate at the application level and use blockchain to support database functionality. The next level, the blockchain architecture, describes how the blockchain network is organized, including its ledger design and consensus mechanism. Below this, the blockchain data structure layer specifies how transactions are grouped and verified. At the foundational level, the blockchain storage engine handles the data storage techniques used by blockchain nodes to manage ledger data. Together, the two frameworks provide complementary perspectives on blockchain data management, linking high-level conceptual design with the practical organization of BDMSs. Figure 2 summarizes the correspondence between the two sets of layers.

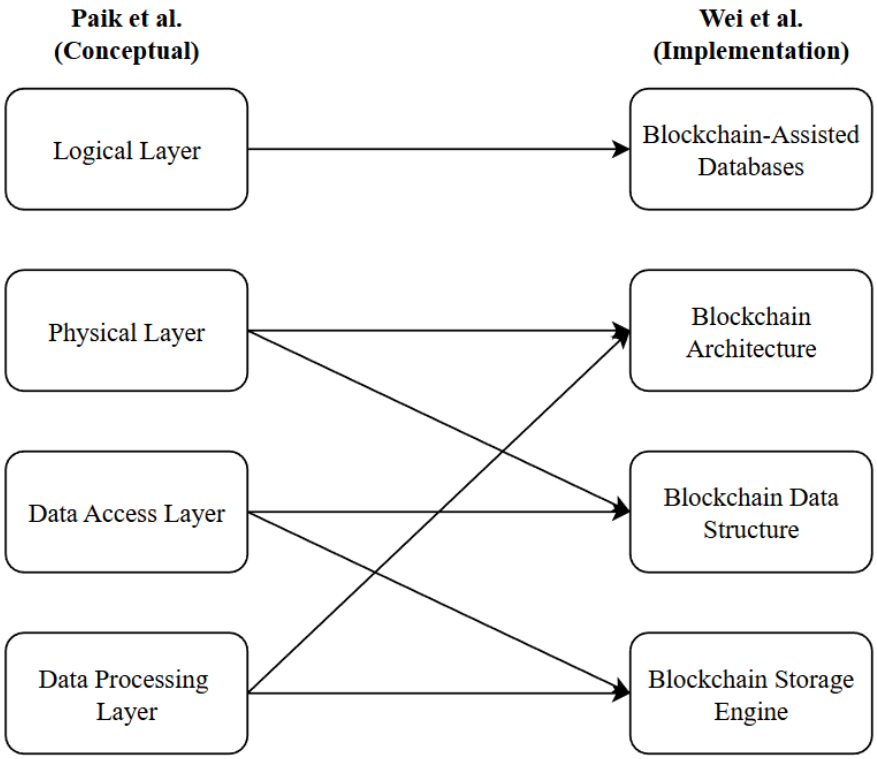

Figure 2. Mapping between the conceptual layers of Paik et al. [3] and the implementation stack of Wei et al. [4].

3.2. Permissioned System Architectures for BDMSs

This subsection reviews representative studies that introduce or implement permissioned BDMS architectures, highlighting key design choices. The studies are organized by application domain: healthcare, IoT data management, personal data management, and other use cases. Systems in each category illustrate key architectural patterns, including off-chain storage solutions, modularity, and access control mechanisms. Collectively, these use cases provide a broad perspective on approaches to designing BDMSs.

Table 1 summarizes the representative systems, which are analyzed in more detail below. To complement the architectural perspective, the maturity of each BDMS is assessed using TRLs [9], which range from TRL 1 (initial concept) to TRL 9 (operational use). This provides an approximate measure of each system's development stage and its proximity to practical deployment.

Table 1. Comparison of permissioned BDMSs across different use cases.

Reference	Application Domain	Blockchain Platform	Consensus Mechanism	TRL	Data Storage Solution
Zaabar et al. (2021), [12]	Healthcare	HLF	PBFT	5	Enc. health data in OrbitDB/IPFS (OC); hashes (OCn)
Azbeg et al. (2022), [13]	Healthcare	Ethereum	PoA (Clique)	5	Enc. health data in IPFS (OC); hashes (OCn)
Han et al. (2025), [14]	Healthcare	Custom system	PoW (assumed setting)	4	Enc. health and personal data in IPFS (OC); hashes + aggregate sigs. (OCn)
Al Asad et al. (2020), [15]	Healthcare	MultiChain	PoA	3–4	General data (OCn); additional records in cloud (OC)
Lee et al. (2020), [16]	Healthcare	Ethereum	PoA	6–7	RSA enc. data in cloud DB (OC); hash ptrs. (OCn)
Tharani et al. (2020), [17]	IoT data	Ethereum	Unspecified (miners referenced)	4	Cloud DB (OC); hash ptrs. (OCn)
Ayoade et al. (2018), [18]	IoT data	Ethereum	Unspecified (tested on the now-deprecated Rinkeby PoA testnet)	4–5	Enc. data in SGX enclaves (OC); hashes (OCn)
Haque et al. (2024), [19]	IoT data	EOSIO (now Antelope)	DPoS	4	IoT data in IPFS (OC); metadata + hash ptrs. (OCn)
Kakarlapudi et al. (2021), [20]	Personal data	HLF	Unspecified (HLF ordering service)	5	Small files in cloud DB, large in IPFS (OC); no hashes (OCn)
Faber et al. (2019), [21]	Personal data	Custom system	Unspecified (permissioned governance by stakeholders)	2	Enc. repos (OC); hash ptrs. (OCn)
Truong et al. (2019), [11]	Personal data	HLF	Kafka-based ordering service (CFT)	4–5	Data in MongoDB (OC); enc. ptrs. + hashes (OCn)
Lomotey et al. (2022), [22]	Personal data	Ethereum	PoA	5	Enc. data in CouchDB (OC); hash ptrs. (OCn)
Zhu et al. (2019), [23]	Documents	Custom system	Voting-based under trusted authority	4	Enc. docs in cloud (OC); version hashes + votes (OCn)
Chen et al. (2019), [24]	Personnel data	HLF	RBFT	4–5	Non-core data in DB (OC); key data + hashes (OCn)
Li et al. (2024), [25]	Shared enterprise data	Custom system	SRL-BFT	5–6	In-memory tables (OC); shared data (OCn)

Note: OC = off-chain; OCn = on-chain; Enc. = encrypted; DB = database. TRL values are approximate and inferred from each study.

3.2.1. Healthcare Data Management

HealthBlock is a permissioned BDMS for remote patient monitoring and electronic health record (EHR) access [12]. The system is structured across six layers: an IoT physical layer with wearable devices capturing patient vitals; a connectivity layer; an off-chain decentralized database layer using OrbitDB with the InterPlanetary File System (IPFS) to store encrypted health data and generate hashes; a blockchain layer built on HLF using PBFT (Practical Byzantine Fault Tolerance) consensus for access control; a decentralized application layer enabling secure EHR access and monitoring; and a user layer for patients, healthcare providers, and administrators. Two permissioned blockchain channels are used:

a device channel for IoT management and a consultation channel for EHR access. The system was benchmarked using Hyperledger Caliper.

BlockMedCare is a BDMS for IoT-enabled healthcare data sharing [13]. Patients register IoT medical devices through a smartphone DApp, which creates blockchain accounts and encrypts collected data using proxy re-encryption before storing it on IPFS. A private Ethereum blockchain using Clique proof-of-authority (PoA) consensus records data hashes and enforces access control and device registration through smart contracts. Hospitals serve as validator nodes and enable authorized physicians to retrieve data through re-encryption. The architecture comprises three components: patients (IoT devices and smartphone gateway), the medical team (physicians, hospitals, and research institutions) and IPFS, deployed on a private Go-Ethereum (Geth) [26] network.

Cross-institutional EHR sharing is also addressed in [14] through a cross-chain architecture based on aggregate BLS (Boneh–Lynn–Shacham) signatures. The system implicitly assumes a proof-of-work (PoW) consensus, leveraging three interconnected blockchains: a patient blockchain storing patient signatures and IPFS hashes of encrypted personal information; a healthcare provider blockchain recording aggregate signatures from multiple institutions as well as dataset metadata linked to encrypted medical records on IPFS; and the social blockchain, which connects the system to external blockchain networks, enabling secure data exchange between institutions. Testing on a local Ethereum deployment reported low latency and reduced storage requirements through signature aggregation.

Al Asad et al. [15] propose a permissioned healthcare data-sharing framework on MultiChain [27] using PoA. Upon registration, each patient is assigned a dedicated blockchain instance initialized with a genesis block, storing general information in an FHIR-compliant format (Fast Healthcare Interoperability Resources). Additional data are offloaded to cloud storage. The architecture includes a blockchain layer, an API service layer that facilitates communication between the blockchain and application components, and an application layer through which stakeholders interact with the system. Access control is hierarchical: patients authorize hospital entities, which may delegate limited rights to physicians through smart contracts. Block creation is restricted to authorized actors based on patient-defined permissions. The system was validated through a simulation of permission delegation and access control.

A comparable platform for EHR exchange is evaluated across hospitals participating in the AeHIN (Asia eHealth Information Network) [16]. It uses a private Ethereum blockchain with PoA consensus. The system includes a record management layer for uploading, viewing, and authorizing access to RSA-encrypted EHRs, alongside a blockchain exchange layer for verification and transaction logging. Records are encrypted and stored in an off-chain database, while a SHA-256 hash and the corresponding database index are recorded on-chain. The prototype was tested with hospitals and users in several Southeast Asian countries, supporting effective cross-organizational record sharing.

3.2.2. IoT Data Management

A two-layer Ethereum-based architecture is proposed in [17] for blockchain-backed database management in IoT-enabled cloud applications. The system separates a blockchain-integrated cloud backend from the application layer. The cloud server includes an on-chain layer that maintains a distributed ledger and an off-chain layer that uses hash pointers to track database changes. Furthermore, smart contracts are used to define database permissions, support application registration, and record database actions in the ledger.

IoTSMARTCONTRACT is a system combining blockchain and a Trusted Execution Environment (TEE) for IoT data management, presented in [18]. The system comprises

an IoT client network connecting devices through gateways, an Ethereum blockchain layer managing device registration and access permissions via smart contracts, and an Intel SGX (Software Guard Extensions) module for off-chain storage. Data integrity is maintained using hash-based message authentication codes (HMACs) and SGX enclave sealing, ensuring that only authorized enclaves can access stored information. Experimental evaluation was carried out with five IoT devices, demonstrating efficient operation and secure data storage.

Haque et al. [19] design a scalable IoT data management framework on EOSIO using delegated proof-of-stake (DPoS). The paper uses the legacy EOSIO software stack which was forked and rebranded as Antelope (2022) [28,29]. The architecture separates constrained devices—which rely on gateways to interact with a local blockchain—from streaming devices that can interact directly with smart contracts and off-chain storage. IoT nodes and consensus nodes handle data collection and validation, respectively. The local blockchain temporarily stores metadata and hash references before they are recorded to the public blockchain. The public blockchain runs smart contracts that facilitate node registration and data exchange, storing SHA-256 hashes and URL pointers to the corresponding IoT data. A distributed storage layer uses IPFS for off-chain storage, while a user layer allows users to access IoT data through gateways, which also maintain the local blockchain state. The framework was evaluated through simulation on a local EOSIO node, demonstrating scalability under test conditions.

In IoT environments, alternatives to blockchain such as IOTA have also been proposed [30,31]. IOTA uses a Directed Acyclic Graph (DAG)-based structure, originally designed to enable feeless transactions and higher throughput. It is particularly suited to IoT use cases, although recent developments highlight trade-offs between scalability, decentralization, and IoT suitability.

3.2.3. Personal Data Management

A permissioned HLF-based system that records consent on-chain while storing personal data in cloud databases is described in [20]. In this architecture, the frontend serves as an interface for submitting consent requests and accessing records, interacting with both the blockchain and cloud layers. Consent management and audit functions are implemented in chaincode, enabling organizations to request access that users can approve or revoke. Administrators oversee data sharing, audits, and deletion, with larger deployments requiring multiple administrators. Approved data is shared through AWS (Amazon Web Services) for small files or via IPFS for larger files, with access restrictions and time-limited availability. The authors also avoid storing hash references on-chain, noting that such references could be considered personal data under future privacy regulations. Their evaluation, conducted in both local and cloud-based environments, indicates that the system scales under tested workloads and maintains low transaction latency.

Faber et al. [21] present BPDIMS, a conceptual blockchain-based design for consent-based personal data handling under the General Data Protection Regulation (GDPR). The architecture comprises a smart-contract blockchain to automate consent and data exchanges between users, an access blockchain enforcing time-limited access, and an identity blockchain storing hash pointers to encrypted off-chain data. Data are protected by symmetric keys managed via threshold cryptography across multiple key keepers. The system defines four roles: users, service providers, data purchasers, and data validators, with users interacting through a unified interface to grant or revoke consent, monitor data usage, and manage monetization. Smart contracts define the terms for data access and compensation to users when datasets are purchased.

Another design that emphasizes GDPR compliance uses HLF with a Kafka-based ordering service and features two distributed ledgers: a 3A_ledger for authentication, authorization, and access control, and a log_ledger for access-token validation and logging, each deployed on a dedicated HLF channel [11]. Encrypted data pointers and hashes stored on-chain link blockchain records to personal data stored off-chain in MongoDB. Participants—including data owners, data controllers, and data processors—are enrolled via HLF's Membership Service Provider (MSP) and Certificate Authority using X.509 certificates. Once permissions are verified on-chain, authorized entities can access off-chain data through REST API requests. Evaluation across different network sizes and workloads with Hyperledger Caliper showed efficient access control and secure data sharing across participants.

DTaaS (Data Trusts as a Service) is a cloud-based platform for multi-stakeholder data sharing [22]. The system is built around a cloud-hosted data controller module that acts as middleware connecting stakeholders through HTTPS or APIs. It comprises a presentation layer with a web-based interface, an application layer that manages authentication, identity, user profiles, and consent, a process layer that handles reporting, policy enforcement, and data storage operations, and a PoA Ethereum layer using smart contracts for auditability and consent logging. Data subjects upload their data and define consent parameters, while data controllers request permission to use it. Data are stored off-chain in encrypted repositories within the controller, with references recorded on-chain via smart contracts. Experiments on AWS using CouchDB as the off-chain data lake demonstrated scalability and stable request handling under the evaluated workloads.

3.2.4. Other Use Cases

Zhu et al. [23] propose the Controllable Blockchain Data Management (CBDM) system for document management. It includes a trust authority that validates document changes, supervises identities, and can revoke voting rights; cloud servers that store encrypted documents; and a blockchain layer with voting and counting smart contracts, and a system administrator overseeing registration and voting outcomes. The workflow proceeds through system initialization, document modification, and document management. Users submit modification requests that are verified by the system administrator, while the trusted authority reviews and can override proposed changes. Once approved, the documents are encrypted and stored on cloud servers. Then, users vote via smart contracts to accept or reject the proposed changes, with the blockchain recording hashes of changed documents and associated voting records. Experiments on a simulated Geth environment evaluated time and gas costs under varying conditions.

A prototype personnel information management system built on HLF and employing a hybrid storage model is implemented in [24]. Core data are stored on-chain, while additional information is maintained in a central database and verified via on-chain SHA-256 hash references. The system follows a Model–View–Controller (MVC) architecture comprising a presentation layer, a service layer, and a blockchain network layer, using Redundant Byzantine Fault Tolerance (RBFT) (recent versions of HLF use Raft and BFT [32]) and Chaincode to handle transaction validation and authorization. Membership and access control are enforced through X.509 certificates, elliptic curve cryptography, and a pluggable MSP. Experiments show that the hybrid storage model improves read/write scalability as data volume increases compared to fully on-chain storage.

Finally, CoralDB is a BDMS for document sharing between organizations [25]. It includes a storage layer based on a custom permissioned blockchain (the data chain), which uses a Secret Random Leader-based Byzantine fault-tolerant (SRL-BFT) consensus

among service provider nodes, and a database layer exposed via a gRPC connection pool. A separate permission chain records metadata and access control. The design further incorporates in-memory collaborative tables that store the latest shared data, with client requests processed through transaction pools. To support queries, CoralDB uses a block index to locate older data on-chain when it is not found in memory. Performance testing demonstrates that CoralDB sustains high throughput under load, with similar read speeds to those of standard distributed databases.

3.3. Cross-Domain Analysis

Several architectural patterns emerge from the use cases discussed above. In healthcare and personal data management, architectures typically emphasize privacy and regulatory compliance, leading to the use of off-chain storage, encryption, and access control mechanisms implemented through smart contracts. In contrast, IoT-oriented systems prioritize scalability and efficient data handling, relying on lightweight consensus mechanisms, gateway-based architectures, and distributed storage to manage high data volumes.

A recurring design choice is the adoption of hybrid architectures that combine blockchain with off-chain storage. Following the classification of off-chain storage proposed in [11]—conventional databases, cloud storage services, and distributed storage systems—the reviewed BDMSs use different implementations. For instance, cloud-based storage is used in [17,23]; distributed storage solutions such as IPFS are adopted in [12,13,19]; hybrid approaches combining multiple types of storage are observed in [20]. Conventional database solutions are also used in [11,24].

Although many of the reviewed solutions rely on custom implementations, HLF and Ethereum remain the most commonly used platforms. Furthermore, layered architectures that separate system responsibilities are widely adopted, along with smart contracts (or chaincode in the case of HLF) used for access control, consent management, and, in some cases, device registration and data provenance. With respect to consensus, lightweight byzantine fault tolerance (BFT) and crash fault tolerance (CFT) variants are generally preferred. Most systems (see Table 1) appear to be positioned at TRL 4–5, indicating functional prototypes validated in laboratory or relevant environments, with limited evidence of large-scale deployment.

Some implementations were developed using technologies that have since been upgraded or deprecated. Although miners are mentioned in [17], the consensus mechanism is not specified, making the production setup unclear. Public Ethereum has since transitioned to proof of stake (PoS) with the Merge [33]. Similarly, the Rinkeby testnet used in [18] has been deprecated [34]. The framework in [19] relies on the EOSIO software stack, which was later forked into the Antelope framework [29]. In addition, consensus is implemented using Kafka in [11], while RBFT is employed in [24]. For HLF, Raft and BFT remain the officially supported default ordering services [32]. Despite ongoing platform changes, the underlying architectural principles in these works remain applicable to BDMS design. The following subsection examines the resulting trade-offs in comparison with conventional database systems.

3.4. Comparing Conventional DBMSs and BDMSs

This subsection examines how BDMSs differ from conventional database management systems (DBMSs), focusing on the implications of decentralization compared to traditional systems.

Blockchains differ from conventional databases in both architecture and operational principles across key dimensions such as trust, privacy, robustness, performance, and security [35]. More specifically:

1. Blockchains establish trust through consensus, whereas conventional databases rely on a central authority to manage access and coordinate operations.
2. In terms of privacy, blockchains support transparency and anonymity while databases restrict visibility to authorized users.
3. Blockchains distribute data across nodes, providing fault tolerance, whereas centralized databases remain vulnerable to single points of failure.
4. Conventional databases support high transaction throughput, while blockchains often experience latency due to consensus requirements.
5. Blockchains ensure security through cryptographic mechanisms but remain susceptible to threats such as majority attacks. In contrast, conventional databases rely on access controls that are prone to failure if the administrator is malicious or compromised.

These differences are illustrated in [35] using a decision tree, indicating that blockchain should be selected when immutability and transparency are core requirements. At the same time, conventional databases are preferable in scenarios requiring high performance and confidentiality.

Additional qualitative insights from [5] reinforce these distinctions by highlighting challenges in blockchain systems related to energy-intensive consensus mechanisms, interoperability, regulatory requirements, and standardization. Empirical analyses of real-world implementations [36] identify further difficulties including managing large and heterogeneous datasets, coordinating on-chain and off-chain storage, supporting real-time analytics and IoT integration, and maintaining synchronization across nodes. The study also reports challenges in ensuring ACID (Atomicity, Consistency, Isolation, and Durability) properties, concurrency control, and advanced data management capabilities, which are more effectively supported by conventional DBMSs. Complementing these findings, Wei et al. [4] identify read/write trade-offs, limited query efficiency, and challenges in distributed data access and storage optimization in BDMSs.

Several studies compare BDMSs and conventional databases from a performance perspective. HLF and Apache Cassandra are compared in [37]. Their experiments measure latency across varying network sizes and read/write ratios, while accounting for architectural differences. The results show that HLF achieves lower read latency in smaller networks, whereas Cassandra scales more effectively under higher loads. These findings suggest that performance trade-offs depend on the application context, with permissioned blockchains better suited to scenarios prioritizing fast reads and controlled access.

Similarly, BigchainDB is compared against HadoopDB and Hive in [38], where it is reported that BigchainDB systematically outperforms the other systems in transaction creation and point queries. This advantage is partly attributed to BigchainDB's BFT consensus and MongoDB backend, which avoid the performance overheads of fully decentralized systems. More broadly, private blockchain systems are benchmarked in [39] using the BLOCKBENCH framework, showing that they still differ substantially in performance compared to conventional databases, mainly due to consensus mechanisms and limitations in execution engines and data storage strategies. The study indicates that improving blockchain performance requires clearer architectural separation between components, more optimized storage layers, and more efficient smart-contract execution.

Overall, BDMSs are best suited to scenarios requiring trust and coordinated data access, but integrating blockchain introduces additional complexity and overhead. Consequently, such systems are often most effective when implemented as hybrid solutions alongside conventional databases, which highlights the need for careful evaluation of architectural trade-offs [40]. One approach in this direction is the Whatever-Ledger Consensus (WLC) model, which enables heterogeneous databases to coordinate through a shared blockchain by agreeing on transaction outcomes [41]. Building on WLC, ChainifyDB implements this

model by integrating blockchain functionality into existing databases without requiring changes to their underlying infrastructures. Experimental evaluations show that ChainifyDB maintains strong consistency, supports failure recovery, and achieves significantly higher throughput than established permissioned blockchain platforms such as HLF. This illustrates an alternative pathway for enabling conventional databases to participate in permissioned networks while preserving their core performance characteristics.

4. Development Tools and Platforms

This section provides an overview of commonly used tools, clients, and platforms for developing, testing, and deploying smart contracts on Ethereum Virtual Machine (EVM)-based and permissioned blockchain systems, summarized in Table 2. The ecosystem can be viewed across different roles within the development process: execution clients, development environments, enterprise platforms, and tools for local testing and network deployment. Tool selection depends on system requirements, including network type (public vs. permissioned), performance constraints, and integration needs.

Table 2. Comparison of blockchain development tools, clients and platforms.

Tool/Platform	Type	Development Language(s)	Key Features	Primary Use Case	Status
Geth [26]	Ethereum client	N/A (no contract authoring)	Execution client; JSON-RPC and tracing tools	Running Ethereum nodes; smart contract execution	Active
Besu [42]	Ethereum client	N/A (no contract authoring)	Execution client; supports public and permissioned Ethereum; configurable consensus algorithms; private transactions; JSON-RPC APIs	Running Ethereum nodes; permissioned Ethereum networks	Active
Hardhat [43]	Development environment	Solidity, TypeScript	Mainnet forking; stack traces; Solidity and TypeScript testing; network simulation; plugin ecosystem	Smart contract development and testing	Active
Foundry [44]	Development toolkit	Solidity	Mainnet forking; Solidity REPL; framework integration	Smart contract development and testing	Active
GoQuorum [45,46]	Permissioned Ethereum platform	Solidity	Public and private transactions; pluggable consensus; access control; JSON-RPC extensions	Enterprise blockchain applications	Active
Truffle [47]	Development environment	Solidity, JavaScript	Automated testing; deployment and migration framework; network management	Smart contract development and testing	Archived
Ganache [48]	Personal blockchain	Solidity, JavaScript	Mainnet and testnet forking; state snapshots; JSON-RPC interface	Local dApp testing and deployment	Archived
HLF [7]	Permissioned blockchain framework	Go, Java, Node.js	Chaincode execution; privacy and identity management; pluggable consensus; SDK and API support	Enterprise blockchain applications	Active
Minifabric [49]	Network management tool	N/A (no contract authoring)	Automated network setup; chaincode lifecycle management	Fabric network deployment and management	Archived

Note: The "Development Language(s)" column indicates the languages developers use to create smart contracts and apps, not the languages in which the tools themselves are implemented.

Geth is one of the original and most widely used Ethereum clients, written in Go. It functions as an execution client, handling transactions and smart contract deployments while also maintaining the EVM. Running Geth alongside a consensus client enables a system to operate as a full Ethereum node. The client provides JSON-RPC APIs and libraries for contract interaction and account management, as well as a developer mode for safe testing [26]. Unlike the other tools listed below, Geth functions as the underlying execution client rather than a dedicated development framework.

Hyperledger Besu (Besu) is a Java-based Ethereum execution client developed under the Hyperledger project [42]. It operates on both public and permissioned networks and provides a command-line interface and standard JSON-RPC APIs for node management, transaction processing, and smart contract interaction. Within DIGI4ECO, the blockchain component is implemented using Hyperledger Besu [42], combined with the QBFT consen-

sus mechanism [50]. This configuration is well suited to permissioned environments with controlled participation and reliable performance [51].

Hardhat (Nomic Foundation) is a Node.js-based Ethereum development environment for writing, testing, debugging, and deploying Solidity smart contracts. It offers stack traces, supports testing in Solidity and TypeScript, and allows local network forking and testing on Layer 2 environments. It also provides a plugin-based, TypeScript-extensible architecture for customizing workflows and managing contract deployment [43].

Foundry is a Rust-based toolkit for Ethereum development that provides tools for compiling, testing, deploying, and interacting with smart contracts. It comprises four main components: Forge for contract compilation and testing, Cast for command-line blockchain interaction, Anvil for running local Ethereum-compatible networks, and Chisel, a Solidity REPL for experimentation and debugging. Foundry also supports local network simulation, mainnet forking, and integration with other frameworks [44].

GoQuorum is an enterprise Ethereum platform maintained by ConsenSys for deploying permissioned Ethereum networks. It provides an environment supporting both private and public transactions, multiple consensus algorithms, and extended JSON-RPC APIs for privacy and network management. Developers can integrate GoQuorum with existing Ethereum tools such as Hardhat and web3 libraries for smart contract deployment and monitoring [45,46].

Truffle Suite (Truffle Suite tools are no longer actively maintained [52]), developed by ConsenSys, provides an integrated development framework for EVM-based blockchains [53], including Truffle [47] for contract development and Ganache [48], a personal Ethereum blockchain, for local testing. Ganache supports mainnet and testnet forking, state snapshots and reversion, and provides an interface compatible with Node.js for programmatic use.

HLF is a permissioned blockchain framework for enterprise applications that enables the development of chaincode using general-purpose programming languages. In HLF, privacy is achieved through channels, private data collections, and MSPs. Its modular architecture, including pluggable consensus mechanisms, ordering services, and endorsement policies, allows customization for different trust models and performance requirements. The framework also provides APIs, SDKs, and documentation to support application development, testing, and deployment [7].

Hyperledger Minifabric is a lightweight automation tool for deploying and managing HLF networks. It simplifies operations such as channel creation, chaincode lifecycle management, and configuration artifact generation, providing an accessible environment for testing and rapid deployment of Fabric networks [49] (the Hyperledger Minifabric repository was archived in November 2023 and is no longer maintained).

5. System Description

This section outlines the blockchain component currently under development for the DIGI4ECO project. Although the project is ongoing and subject to further refinement, the fundamental design and analysis presented here are not expected to diverge significantly from the final implementation. This stability is supported by a central design principle: maximizing modularity to ensure portability and seamless integration within broader and more complex data management architectures.

5.1. Design Goals

The following design goals reflect the specific requirements of the DIGI4ECO project, while also serving as general guidelines for the development of blockchain-based data management components in similar contexts.

- Type of blockchain: The term "blockchain technologies" is generic, as in practice it refers to a whole spectrum spanning from public/permissionless to private/permissioned systems. For DIGI4ECO, the latter end of the spectrum is the suitable choice. Furthermore, there is no need for the use of a digital currency, as there is no incentive scheme associated with the underlying data management.
- On-chain vs. off-chain operations: Blockchains are not intended by design to handle massive amounts of data. Based on this fundamental consideration, we need to let the blockchain component process and store the absolute minimum information (e.g., in the form of metadata), while the actual data are stored off-chain by other components with which the blockchain component is integrated. This choice is of particular importance for DIGI4ECO, where massive amounts of raw data are acquired and transmitted through several sensors.
- Data pipelines: In general, the project involves workflows for ingestion, transforming, and delivering acquired data. Taking into account the points mentioned above, the blockchain component can play a supportive role in an asynchronous manner rather than in real time.
- No direct exposure of blockchain input/output (I/O): To separate the core blockchain I/O operations from the high-velocity data dynamics of the project (i.e., massive data pipelines), we have introduced a frontend layer that acts as the communication endpoint between the blockchain and the non-blockchain components.
- Easy integration with other components: This relates to the previous point regarding the addition of a frontend layer. While integrating blockchains with other components designed for intensive data processing is a relatively recent area, we believe that the appropriate approach is to utilize a well-established method (e.g., API), likely originating from the non-blockchain part of the technological spectrum. Naturally, this protocol should be supported by tools compatible with the underlying blockchain.

From a high-level perspective, the above design goals serve as a paradigm that challenges several common stereotypes about blockchains, which often convey a partial view of the technological reality, especially when the analysis is superficial. Indicative stereotypical misconceptions include assumptions that blockchain-native currency is always required and that all associated data must be stored directly on the blockchain. Furthermore, one key consideration, which underpins the overall design goals, is that a purely blockchain component cannot, by itself, address all data management tasks; instead, multiple technologies must work together.

In a concise manner, the intended system can be characterized by the following functional and non-functional requirements.

Functional Requirements:

- Frontend–Backend Separation: The system shall separate frontend and backend components to ensure safe and controlled handling of blockchain I/O operations.
- Asynchronous Processing: The system shall support asynchronous data processing through a queue-based mechanism.
- RESTful API Communication: The system shall provide RESTful APIs to enable interaction between external components and the blockchain subsystem.
- Data Validation and Filtering: The system shall validate and filter incoming data (off-chain) prior to submission to the blockchain.
- Data Hashing and Metadata Storage: The system shall store only hashed representations and associated metadata of the data on the blockchain.

Non-Functional Requirements:

- Modularity and Portability: The system shall be modular to enable portability and seamless integration.
- Scalability: The system shall handle large-scale data indirectly by storing only minimal metadata on-chain.
- Performance (Non-Real-Time Operation): The system shall operate in a non-real-time, batch-oriented manner, consistent with blockchain constraints.
- Interoperability: The system shall integrate seamlessly with external components using standard communication protocols (e.g., RESTful APIs).
- Robustness and Fault Isolation: The system shall isolate faults through the frontend layer to prevent disruptions to core blockchain operations.
- Maintainability and Upgradability: The system shall support independent deployment and updates of frontend and backend components.

Together, these requirements provide a clear specification of the system's capabilities, constraints, and design priorities, guiding both development and evaluation.

5.2. System Architecture and Implementation

The architectural overview of the DIGI4ECO blockchain component is presented in Figure 3 (This represents the current baseline version, subject to adjustment as both the overall project and this specific task are ongoing). The diagram distinguishes between two categories of elements: (1) the components that collectively form the blockchain subsystem, and (2) an abstract visual representation of the external DIGI4ECO component(s) with which the blockchain subsystem interacts. Integration between these components is achieved through structured information exchange. To represent this interaction, two types of arrows are used: gray arrows indicate internal information flows within the blockchain subsystem, while blue arrows represent the exchange of information between the blockchain subsystem and external DIGI4ECO components. The direction and semantics of each information flow are clarified using numbered labels associated with the arrows.

The blockchain subsystem consists of three core components: the data-pipeline controller, the frontend, and the backend. The data-pipeline controller is responsible for acquiring data from external DIGI4ECO components (referred to collectively as the external infrastructure). Once retrieved, the data are transmitted to the blockchain for processing. After blockchain operations are completed, the corresponding results are returned to the external infrastructure. This component includes two main mechanisms: a data fetcher and a data sender. The fetcher retrieves data from the non-blockchain environment, while the sender transmits this data to the blockchain. The fetch-and-send process operates asynchronously, with a queue mechanism temporarily storing retrieved data between the two stages. Coordination between fetching and sending is configurable.

The frontend acts as a buffer between the blockchain backend and the data source, providing increased robustness and control. This separation helps isolate faults or irregularities in the incoming data stream, thereby preventing them from directly affecting core blockchain operations. In addition, the frontend also supports filtering and validation of data prior to their entry into the underlying blockchain backend. Both the frontend and backend are equipped with RESTful APIs.

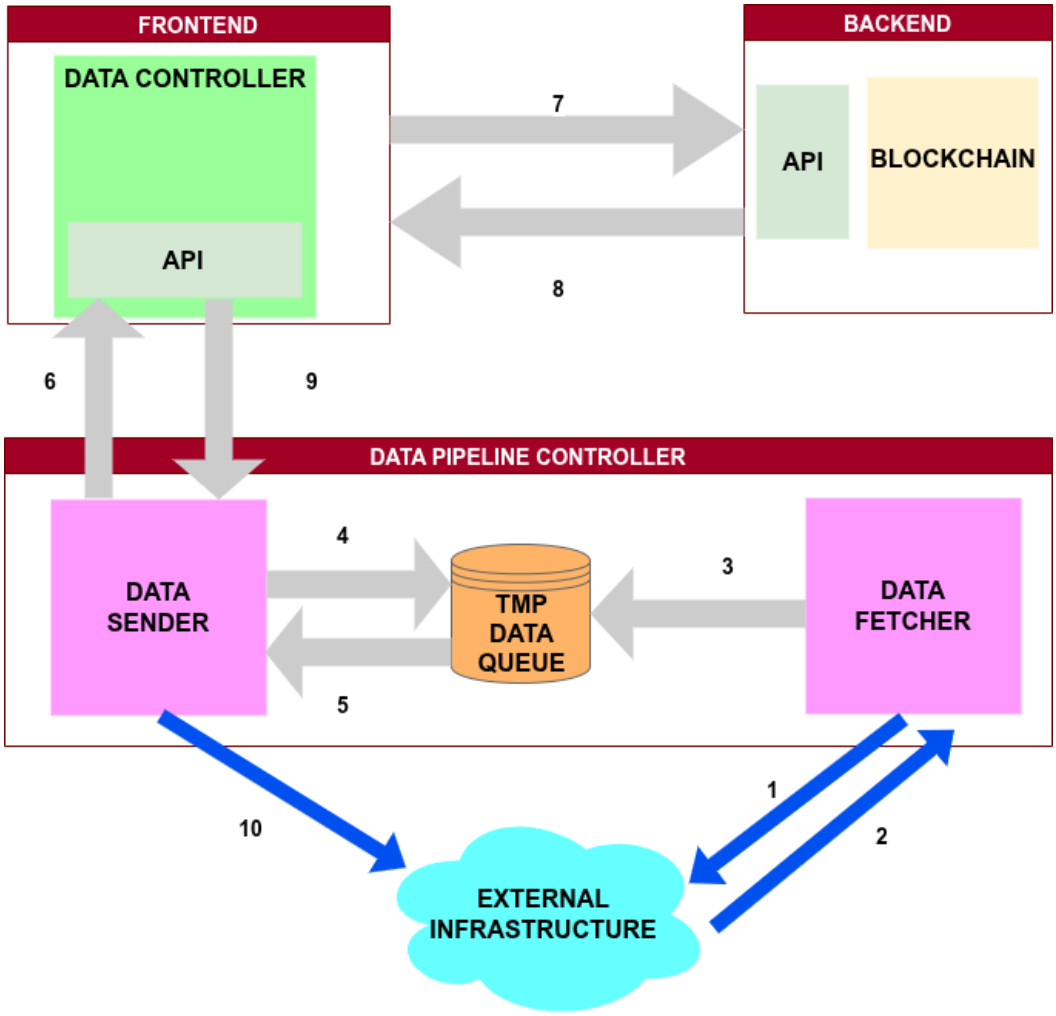

Figure 3. Architectural overview of DIGI4ECO's blockchain component.

Overall, key characteristics of the architecture include:

1. No storage of actual data in the blockchain: Only minimal information is stored on-chain in the form of hashed values accompanied by concise descriptive metadata.
2. No continuous or real-time data streaming into the blockchain: Data are not ingested in real time from the external infrastructure.

The decision not to store actual data on-chain follows from the design constraints of blockchains, which are not intended to manage large-scale datasets. As an indicative background, as of May 2026, the Bitcoin blockchain occupies approximately 0.7 TB of storage [54], while a pruned Ethereum execution-layer full node requires approximately 1.5 TB of disk space [55].

Similarly, the absence of real-time streaming is in line with the design principles of blockchains, which are not optimized for low-latency processing at scale. This design choice also reflects the specific requirements of the DIGI4ECO use case, where the blockchain component operates on summarized data rather than raw streams. This setup introduces a trust layer that can function at slower timing, compared to the high-velocity data pipelines handled by other components.

From an implementation perspective, the system incorporates several additional elements. Data validation is performed off-chain using the Pydantic library, ensuring that all incoming data conform to predefined schemas before being submitted to the blockchain. With respect to access control, the adopted strategy reflects the specific requirements

of the DIGI4ECO project: data producers, i.e., entities authorized to store data on the blockchain, are restricted to project partners, whereas data consumers operate in read-only mode and may include any interested scientific entity. Given this controlled yet open-access model, access control mechanisms are implemented at the application layer rather than within the smart contract itself, thereby keeping the on-chain logic lightweight. Communication between off-chain components and the blockchain is realized through RESTful APIs, implemented using FastAPI, enabling efficient and modular interaction. Furthermore, off-chain data handling, including temporary storage (queuing), is supported by a lightweight SQLite database.

6. Practical Considerations: Setup and Deployment

This section focuses on a very practical aspect of the blockchain component—namely, the setup of the underlying infrastructure and the corresponding deployment process. In general, the uniqueness of each case and the specific configurations it may require are acknowledged. However, in the spirit of this work, it is considered useful to report the specific considerations adopted for the present case. In doing so, this section aims to help address a gap frequently observed in the literature, where practical details of this nature are often left undocumented, even though, at the end of the day, what is needed is an operational system. While these details may not push the state-of-the-art frontiers, they provide valuable guidance for practitioners and researchers seeking to translate blockchain design principles into reliable, reproducible implementations.

- Modularity between backend and frontend: As mentioned above, this design choice provides robustness. From a deployment point of view, the separation of backend and frontend enables independent deployment cycles. This is particularly convenient for the blockchain part (i.e., the backend), since changes to smart contracts typically involve a more complex process compared to updates in an API-based frontend.
- Handling bursts of requests: Blockchains, in general, are not designed to serve real-time requests, so operating periodically is a practical choice. In such cases, buffering mechanisms play a supportive role. This is the approach adopted, as shown in Figure 3. Furthermore, although not utilized in this setup, auto-scaling and load-balancing technologies constitute relevant techniques when the bursts of data are substantial.
- Threading: Having the two blockchain layers (i.e., frontend and backend), as well as numerous I/O-bound tasks, threading is an excellent approach for developing and deploying an efficient component. In brief, independent tasks can run concurrently while waiting for I/O operations.
- Virtual environments: Virtual environments are extremely helpful for isolating specific dependencies across different applications. Given that numerous blockchain-related tools and libraries require specific versions, this isolation helps prevent conflicts. In this case, Python's virtual environment is used primarily for operations dealing with the compilation and deployment of smart contracts.
- Data management: It is sometimes assumed that blockchain systems operate independently of conventional database technologies. In practice, however, many blockchain implementations inherently rely on underlying database components to manage data efficiently. Therefore, deployment decisions should take into account the presence and requirements of these databases, such as storage capacity, backup policies, and access control.
- Fail-safe blockchain: This refers to scenarios in which the blockchain component temporarily ceases operations. In such cases, catastrophic outcomes such as data loss are to be avoided. Queued off-chain data is preserved rather than lost, and the

decoupling of the backend from the frontend plays an important role in ensuring this fail-safe behavior.

- Security consideration: The proposed blockchain infrastructure is deployed within a private blockchain environment. As a result, the system is not exposed to the same threat model or level of exposure typically associated with public blockchain deployments. Furthermore, the deployment environment can incorporate conventional cybersecurity safeguards and best practices, including access control mechanisms, network isolation, authentication procedures, monitoring, and secure communication protocols, similarly to any other distributed software application.

In the present use case, a cloud-based deployment on AWS is used, specifically a `t4g.2xlarge` EC2 instance with 32 GB RAM, 8 vCPUs, and 1 TB of storage. According to AWS (https://aws.amazon.com/ec2/instance-types/t4/, accessed on 9 May 2026), this instance type provides a baseline level of CPU performance together with the ability to burst CPU usage when needed. This aligns well with the system requirements, as the platform operates a private, permissioned blockchain that does not rely on compute-intensive consensus mechanisms such as proof of work (PoW), nor is it required to scale to thousands of transactions per second in real time. Furthermore, the burst capability is consistent with the aforementioned operational characteristics related to handling transient spikes in workload. This ensures that periodic increases in demand can be effectively accommodated.

Given that the implementation handles minimal units of data rather than full data records, 1 TB of storage is sufficient for maintaining the corresponding on-chain state. Each unit is approximately under 1 KB in size—though this may compound depending on burst density. Based on this estimated payload size, the available RAM is also adequate for supporting the associated data flows. Moreover, the 8 vCPUs enable efficient multithreading for asynchronous, I/O-bound operations, allowing the blockchain component to fetch, queue, and process the relevant data effectively.

The primary tools used for the development of each component are presented in Table 3.

Table 3. Primary tools for each component.

Component	Primary Tools Used
Data pipeline controller	Python, SQLite
Frontend	FastAPI, Python
Backend	FastAPI, Python, Geth, Solidity

Regarding the data pipeline controller, Python 3.14 was used to implement the data fetcher and sender components, including the integration of SQLite for the development of the temporary data queue. For the frontend, FastAPI within Python was employed. Similarly, the backend was also developed using FastAPI and Python; however, the primary focus was on the utilization of Geth and Solidity for blockchain integration and smart contract development.

7. Practical Considerations: Operational Factors and Performance Evaluation

This section extends the previous discussion by focusing on factors that significantly affect the operation of the blockchain component as well as evaluating its performance. These factors often go unnoticed and do not directly determine the high-level design decisions—especially when no prior experience exists. Moreover, only limited analysis of such operational issues can be found in technical documentation or the research literature, and even then, practicalities and heuristics are rarely included.

The present analysis is not meant to be exhaustive in the sense of complete coverage. After all, this is nearly impossible, as such factors are use case-specific, while a vast number of variations can exist. However, empirical findings are documented in a way that supports broader applicability; that is, they can inform similar types of applications.

7.1. Operational Factors

- Fees: For data management purposes—especially for small networks compared to major public blockchains—there is often no need for transaction fees, which are typically used to incentivize validators/miners and to prevent network abuse. In such cases, one should first check whether the underlying blockchain supports or requires this feature, and, second, how it is configured. If fees are required (e.g., in a private Ethereum network), they can be practically eliminated by (1) setting the gas price to zero, (2) using a PoA consensus where incentives are unnecessary, or (3) pre-allocating unlimited funds of zero monetary value.
- Latency due to block creation: The process of block creation introduces some latency, which depends on the exact configuration (e.g., consensus mechanism, block size); however, in general, it does not allow time-sensitive operations at the level of real-time or near–real-time data processing. This consideration determines the type of actions that can take place between the initial data ingestion from external sources through the frontend and the creation of the respective blocks. Such actions can include data aggregation as well as validation.
  In particular, the decision to aggregate data opens a broader discussion on how the data of interest should be processed, for example, individually or in batches. In [56], a related approach was proposed, where the metadata of batches of academic certificates was managed, rather than the corresponding PDF versions. This aspect has a significant impact on the overall architecture of the system and requires careful coordination with external components. In the DIGI4ECO-specific case, a batch-based paradigm is adopted to efficiently manage data and accommodate the latency introduced by block creation.
- On-demand block creation: Some blockchains create blocks according to regular time periods (often a configurable parameter), while others follow the so-called on-demand block creation. This refers to the creation of blocks when specific conditions are met, such as a smart contract being invoked or certain thresholds (e.g., the size of a transaction queue) being reached.
  The former approach aligns with the characteristics of major public networks, including Bitcoin and Ethereum. However, this operational mode is less well-suited to smaller private/permissioned networks, where data requests occur at irregular intervals. In such cases, following a periodic block creation strategy is likely to occupy storage unnecessarily. Thus, it is preferable for the selected network to support on-demand block creation. If this is not possible, a middle ground can be achieved by appropriately adjusting (i.e., fine-tuning for the specific application) the block creation period, which can be combined with off-chain data aggregation prior to blockchain operations—especially when the block creation periods are relatively long.
- No ARM64 for Solidity compiler: Solidity constitutes, by many measures, the most mature programming language for (EVM-based/compatible) smart contracts. Unfortunately, the official precompiled binaries of the Solidity compiler, `solc`, are not available for ARM64 architectures (common in energy-efficient CPUs). This can create challenges, as compiling from source on ARM64 is not always straightforward (e.g., dependency issues, environment-specific compilation errors), particularly in cloud environments.

The JavaScript version of the Solidity compiler, `solc-js`, addresses this issue by relying on WebAssembly, which is portable across platforms. This portability also facilitates automatic deployment and more efficient CI/CD pipelines.

- Backward compatibility: In general, blockchains exhibit backward compatibility between versions. However, they are sophisticated systems composed of multiple protocol layers, including data-related mechanisms such as the databases responsible for maintaining the blockchain state. Therefore, despite this compatibility, any major upgrade across different versions (assuming a network of nodes) may trigger a cascade of errors (e.g., database inconsistencies), because many components and layers are involved.

  A similar situation was observed when exploring Geth, where newer versions introduced changes to some protocol rules compared to version `1.9.22-stable-c71a7e26`, upon which an early development cycle relied. Thus, the overall recommendation is to use the same version for all participating nodes whenever possible. This approach is more feasible in private or permissioned networks, where the network size is typically small, and version coordination is easier to enforce.

Additional factors are already widely known in other non-blockchain domains; however, their importance can be redefined when considered within the blockchain framework. Indicative examples include data validation (e.g., data types and formats) prior to processing by smart contracts, as well as the need to monitor resources such as storage—especially when block creation cannot be fully configured, as discussed above in the context of on-demand block creation.

Furthermore, containerization is another general aspect that enhances portability; however, it should be considered in relation to the specific blockchain implementation in order to avoid nested containerization, since several implementations rely heavily on containers. Such nested containerization may introduce challenges in the form of increased operational complexity, performance overhead, and potential security issues.

The primary objective of the above analysis is to raise awareness of specific directions and to set a broader context within which such matters can be practically handled, rather than to be regarded as a cookbook.

7.2. Performance Evaluation

In order to quantify the operational characteristics of the backend (i.e., blockchain), several performance measurements are presented in Table 4. The reported metrics include transactions per second (TPS), average transaction latency (in seconds), and memory usage (in MB). These measurements were obtained for different numbers of workers, where each worker corresponds to a simultaneous parallel thread. Parallelization was applied to read transactions because, according to the project requirements, this type of transaction, unlike write operations, constitutes the dominant workload, as the blockchain component is primarily intended to support audit trails for the relevant data.

Table 4. Performance evaluation under varying number of workers (read).

Number of Workers	TPS	Average Latency (s)	Memory Usage (MB)
1	42.00	0.0237	326.60
2	42.27	0.0471	326.48
3	41.88	0.0713	326.49
5	41.98	0.1178	326.55
10	41.04	0.2367	326.72

It is observed that TPS is not affected by the number of workers. Unlike TPS, the average latency increases as the number of workers increases. Memory measurements

correspond to the client-side application footprint, whereas the underlying blockchain node consumes approximately 326 MB of memory, remaining unaffected by the number of workers. The memory usage remains under 350 MB for all configurations, which makes the present deployment sustainable even in low-end infrastructure.

Write transaction performance is evaluated in Table 5 by focusing on the time required for a node to validate the transaction format and admit it into its transaction pool. This design choice isolates the measurement from the mining time, which is strongly dependent on the underlying computational resources. First, it is observed that TPS for write operations is lower than that of read operations. Second, the average latency is higher compared to read operations, while memory consumption remains approximately identical.

Table 5. Performance evaluation under varying number of workers (write).

Number of Workers	TPS	Average Latency (s)	Memory Usage (MB)
1	17.90	0.0558	326.83
2	17.65	0.1130	326.70
3	16.92	0.1764	326.78
5	16.68	0.2973	326.94
10	16.53	0.5962	327.37

Furthermore, three distinct experimental scenarios were employed to assess the behavior of the system (using one worker) under both read and write operations: (1) Steady Baseline Scenario (low stress level): Simulates low and consistent traffic; (2) Sustained Poisson Scenario (mid stress level): Simulates moderate traffic characterized by stochastic request arrivals and occasional traffic clustering; (3) High-Pressure Burst Scenario (high stress level): Simulates a relatively heavy workload with bursty traffic patterns. The results are presented in Table 6.

Table 6. Performance evaluation for different stress levels for read/write operations.

Type of Operation	Stress Level of Scenario	Average Latency (s)
read	low	0.0311
read	mid	0.0265
read	high	0.0189
write	low	0.0780
write	mid	0.0638
write	high	0.0603

In terms of average latency, read operations consistently outperform write operations across all scenarios. In the Steady Low scenario, write latency is approximately 2.5 times higher than read latency, mainly due to the additional overhead associated with transaction validation and insertion procedures. Under the Stress High scenario, write latency rises to approximately three times that of reads, while the system exhibits stable performance despite the increased workload.

Overall, the observed performance aligns with the computational demands of the system's intended scope. Specifically, according to the respective requirements, the achieved TPS is satisfactory for the intended use case. The reported TPS and latency could be significantly improved by deploying more powerful infrastructure and leveraging distributed parallelization; however, such considerations fall beyond the requirements of this project, as the goal is not to propose a new architecture that exceeds the current state-of-the-art. In general, comparing the reported TPS and latency with other blockchain systems designed and deployed for different purposes (i.e., different contexts) is not always meaningful.

The performance assessment was conducted on a development machine running Ubuntu, equipped with an Intel Core i3-7100T CPU @ 3.40GHz and 16 GB of RAM. This setup differs from the currently active deployment environment, which is based on an EC2 instance on AWS. While the use of the latter could potentially yield improved performance in terms of TPS and latency, as discussed, optimizing for maximum performance is not the primary objective of this work. From this perspective, practical adoption of blockchain technology can be valuable even without ultra-high TPS, as long as the system adequately meets its intended requirements. This illustrates a form of democratization of blockchain adoption, as practical deployments can be achieved even without high-end computational infrastructure.

For the present use case, the transaction payloads are represented as JSON structures containing multiple fields associated with the data of interest. Indicative examples include hash computation results and timestamps corresponding to different stages of the processing pipeline. The relationship between read and write operations is inherently stochastic in the context of the specific use case, as it depends both on the availability of new maritime datasets and on the demand for accessing these datasets. Nevertheless, from an empirical perspective, the volume of read operations is expected to exceed that of write operations. This is attributed to the fact that each dataset insertion typically corresponds to a single write transaction, whereas the same dataset may subsequently be accessed through multiple read operations.

A further advantage of the proposed configuration is that block creation is not performed at fixed time intervals. Instead, block generation is triggered on demand whenever new information is recorded on the blockchain. This approach enables more efficient utilization of the underlying computational resources and contributes to the overall sustainability and operational efficiency of the deployed system. Due to the highly specialized nature of the maritime datasets involved, the system is not expected to experience read/write operations at the scale of hundreds of transactions per day, rendering the proposed configuration well-suited to the operational demands of the use case.

8. Discussion and Conclusions

This section summarizes the key findings of this work and discusses their broader implications for BDMS design and deployment.

Most systems adopt hybrid architectures that combine on-chain and off-chain storage solutions to balance performance, scalability, and user privacy, alongside layered designs and lightweight consensus mechanisms. While individual systems differ in micro-level implementation choices—such as the type of off-chain storage or access control mechanisms—these variations reflect a combination of domain-specific requirements (e.g., privacy-sensitive data) and developer design decisions. Consequently, comparisons across BDMSs are most meaningful at a high level, whereas detailed performance characteristics and implementation trade-offs remain context dependent. In terms of platforms, Ethereum and Hyperledger are frequently utilized, although several works rely on custom implementations. At the same time, although many of the systems reviewed demonstrate technical feasibility through prototypes and experimental evaluations, most remain at intermediate maturity levels, indicating that further work is needed to support broader adoption.

Concerning tool selection and related practical considerations, a system intended for data management should not be based solely, or almost solely, on a blockchain component. A synergy between different technologies is required. This relates to the remark that blockchains, by design, are not meant for handling massive amounts of data. Numerous tools are often promoted amid hype, partly due to rapid release cycles aimed at not missing emerging technological waves. Thus, careful tool selection is required, taking into account

factors such as the degree of maintenance and documentation, popularity within the technical community, and the existence of indicative or mature projects. Even seemingly obvious aspects—such as alignment with the development team's technical profile—should be considered carefully. This alignment can be understood as a spectrum, ranging from leveraging existing knowledge (to accelerate design and development) to maintaining flexibility for experimenting with new architectural designs and tools.

Furthermore, there should be no direct exposure of the blockchain component. At a minimum, a frontend layer is required, acting as the primary interface for receiving data from external components. This, in turn, relates to the selection of appropriate integration technologies. In the general case—excluding specific use cases where incentives exist for data-related operations or services—a blockchain-based application for data management purposes does not require any form of native digital currency or transaction fees. Such configurations should be eliminated or neutralized (e.g., via the proper consensus mechanisms), as they would unnecessarily affect the operational characteristics of the blockchain component. Finally, technical aspects related to core blockchain properties, such as block creation, should be taken into account throughout all major phases, from architectural design to development. These aspects may affect the overall system characteristics.

Drawing on the cross-domain analysis presented in Section 3.3 and the practical considerations discussed in Sections 6 and 7, several lessons emerge that extend beyond the specifics of DIGI4ECO. Recurring patterns in the reviewed literature, including hybrid storage architectures, modular layered designs, and lightweight consensus mechanisms, are consistent with the design choices adopted in the DIGI4ECO blockchain component. At the implementation level, considerations such as on-demand block creation, frontend isolation, off-chain storage coordination, and careful tool selection are not specific to marine ecological monitoring. Rather, they apply to any BDMS aiming to balance data integrity, scalability, and operational efficiency in large-scale, multi-source settings.

Several limitations should be acknowledged: the practical insights are derived from a single case study, the system operates at an intermediate TRL, and the performance benchmarks were conducted on a development machine rather than the production environment. Nevertheless, the architectural patterns and design choices discussed are consistent with those observed across the broader BDMS literature, suggesting that the findings carry relevance beyond the specific application domain.

Future work could focus on the development of a semi-automatic, or ideally fully automatic, framework supported by algorithmic tools to assist system architects and developers in identifying the proper configuration according to the unique characteristics of each use case. Such a framework could guide architectural and technological choices, reducing trial-and-error during system design and initial implementation. In addition, AI-assisted tools could prove practically useful for monitoring blockchain-specific operational characteristics and capturing critical patterns, such as abnormalities in block creation, thereby enhancing system robustness.

Author Contributions: Conceptualization: E.I.; Methodology: E.I.; Literature research: A.P.D. and E.I.; Software: E.I. and S.P.; Writing—original draft preparation: A.P.D. and E.I.; Review: E.I., G.G., S.P. and A.P.D.; Writing—final review and editing: E.I.; supervision of article development: E.I.; Funding supervision: G.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially funded by the European Union—DIGI4ECO project, grant agreement ID: 101112883.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All data presented in this study are contained within the article.

Acknowledgments: During the preparation of this manuscript, the authors used Generative AI tools for the purposes of minor language editing and formatting support, including improving grammar, syntax, and table structure after the manuscript content had been written by the authors. No AI tools were used for generating scientific content, analysis, or conclusions. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ACID	Atomicity, Consistency, Isolation, and Durability
AI	Artificial Intelligence
API	Application Programming Interface
ASV	Autonomous Surface Vehicle
AUV	Autonomous Underwater Vehicle
AWS	Amazon Web Services
BDMS	Blockchain-based Data Management System
BFT	Byzantine Fault Tolerance
BIoT	Blockchain Internet of Things
BLS	Boneh–Lynn–Shacham
CFT	Crash Fault Tolerance
DAG	Directed Acyclic Graph
DApp	Decentralized Application
DBMS	Database Management System
DeFi	Decentralized Finance
DTO	Distributed Twin Observatory
DPoS	Delegated Proof of Stake
EHR	Electronic Health Record
EMODnet	European Marine Observation and Data Network
EVM	Ethereum Virtual Machine
FHIR	Fast Healthcare Interoperability Resources
GDPR	General Data Protection Regulation
HLF	Hyperledger Fabric
HMAC	Hash-based Message Authentication Code
IPFS	InterPlanetary File System
IoT	Internet of Things
IoUT	Internet of Underwater Things
MIoT	Marine Internet of Things
ML	Machine Learning
MSP	Membership Service Provider
MVC	Model–View–Controller
PBFT	Practical Byzantine Fault Tolerance
PoA	Proof of Authority
PoS	Proof of Stake
PoW	Proof of Work
QBFT	Quorum Byzantine Fault Tolerance
RBFT	Redundant Byzantine Fault Tolerance
RSA	Rivest, Shamir, and Adleman
SHA-256	Secure Hash Algorithm 256
SGX	Software Guard Extensions
SRL-BFT	Secret Random Leader-based Byzantine Fault Tolerance
TEE	Trusted Execution Environment

TPS	Transactions Per Second
TRL	Technology Readiness Level
X.509	X.509 digital certificate standard

References

1. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 27 October 2025).
2. Buterin, V. Ethereum White Paper. 2013. Available online: https://ethereum.org/en/whitepaper/ (accessed on 26 September 2025).
3. Paik, H.Y.; Xu, X.; Bandara, H.D.; Lee, S.U.; Lo, S.K. Analysis of Data Management in Blockchain-Based Systems: From Architecture to Governance. *IEEE Access* **2019**, *7*, 186091–186107. [CrossRef]
4. Wei, Q.; Li, B.; Chang, W.; Jia, Z.; Shen, Z.; Shao, Z. A Survey of Blockchain Data Management Systems. *Acm Trans. Embed. Comput. Syst.* **2022**, *21*, 1–28. [CrossRef]
5. Khan, A.N.; Hussain, M.Z.; Nosheen, S.; Hussain, Q. Database Technology for Blockchain: Opportunities and Challenges in Database Technology for Blockchain. *Ucp J. Eng. Inf. Technol.* **2023**, *1*, 32–38. [CrossRef]
6. Aguzzi, J.; Chatzidouros, E.; del Río, J.; Toma, D.M.; Chatzievangelou, D.; Picardi, G.; Masmitja, I.; Clavel-Henry, M.; Grinyó, J.; Francescangeli, M.; et al. *DIGI4ECO: Digital Twin-Sustained 4D Ecological Monitoring of Restoration in Fishery Depleted Areas*; European Union: Bruxelles, Belgium, 2024.
7. Hyperledger Fabric. Hyperledger Fabric Documentation. 2025. Available online: https://hyperledger-fabric.readthedocs.io/en/release-2.5/ (accessed on 10 October 2025).
8. Ethereum Foundation. Ethereum. 2025. Available online: https://ethereum.org/ (accessed on 10 October 2025).
9. European Commission. TRL. EURAXESS. 2015. Available online: https://euraxess.ec.europa.eu/career-development/researchers/manual-scientific-entrepreneurship/major-steps/trl (accessed on 17 November 2025).
10. Delladetsimas, A.P.; Papangelou, S.; Iosif, E.; Giaglis, G. Integrating Blockchains with the IoT: A Review of Architectures and Marine Use Cases. *Computers* **2024**, *13*, 329. [CrossRef]
11. Truong, N.B.; Sun, K.; Lee, G.M.; Guo, Y. GDPR-compliant personal data management: A blockchain-based solution. *IEEE Trans. Inf. Forensics Secur.* **2019**, *15*, 1746–1761. [CrossRef]
12. Zaabar, B.; Cheikhrouhou, O.; Jamil, F.; Ammi, M.; Abid, M. HealthBlock: A secure blockchain-based healthcare data management system. *Comput. Netw.* **2021**, *200*, 108500. [CrossRef]
13. Azbeg, K.; Ouchetto, O.; Andaloussi, S.J. BlockMedCare: A healthcare system based on IoT, Blockchain and IPFS for data management security. *Egypt. Inform. J.* **2022**, *23*, 329–343. [CrossRef]
14. Han, G.; Ma, Y.; Zhang, Z.; Wang, Y. A hybrid blockchain-based solution for secure sharing of electronic medical record data. *Peerj Comput. Sci.* **2025**, *11*, e2653. [CrossRef] [PubMed]
15. Al Asad, N.; Elahi, M.T.; Al Hasan, A.; Yousuf, M.A. Permission-based blockchain with proof of authority for secured healthcare data sharing. In Proceedings of the 2020 2nd International Conference on Advanced Information and Communication Technology (ICAICT), Dhaka, Bangladesh, 28–29 November 2020; pp. 35–40.
16. Lee, H.A.; Kung, H.H.; Udayasankaran, J.G.; Kijsanayotin, B.; B Marcelo, A.; Chao, L.R.; Hsu, C.Y. An architecture and management platform for blockchain-based personal health record exchange: Development and usability study. *J. Med. Internet Res.* **2020**, *22*, e16748. [CrossRef]
17. Tharani, J.S.; Tharmakulasingam, M.; Muthukkumarasamy, V. A blockchain-based database management system. *Knowl. Eng. Rev.* **2020**, *35*, e20. [CrossRef]
18. Ayoade, G.; Karande, V.; Khan, L.; Hamlen, K. Decentralized IoT data management using blockchain and trusted execution environment. In *Proceedings of the 2018 IEEE International Conference on Information Reuse and Integration (IRI), Salt Lake City, UT, USA, 6–9 July 2018*; IEEE: New York, NY, USA, 2018; pp. 15–22.
19. Haque, E.U.; Shah, A.; Iqbal, J.; Ullah, S.S.; Alroobaea, R.; Hussain, S. A scalable blockchain based framework for efficient IoT data management using lightweight consensus. *Sci. Rep.* **2024**, *14*, 7841. [CrossRef]
20. Kakarlapudi, P.V.; Mahmoud, Q.H. Design and development of a blockchain-based system for private data management. *Electronics* **2021**, *10*, 3131. [CrossRef]
21. Faber, B.; Michelet, G.; Weidmann, N.; Mukkamala, R.R.; Vatrapu, R. BPDIMS: A blockchain-based personal data and identity management system. In Proceedings of the The 52nd Hawaii International Conference on System Sciences (HICSS), Grand Wailea, Hawaii, 8–11 January 2019; pp. 6855–6864.
22. Lomotey, R.K.; Kumi, S.; Deters, R. Data Trusts as a Service: Providing a platform for multi-party data sharing. *Int. J. Inf. Manag. Data Insights* **2022**, *2*, 100075. [CrossRef]
23. Zhu, L.; Wu, Y.; Gai, K.; Choo, K.K.R. Controllable and trustworthy blockchain-based cloud data management. *Future Gener. Comput. Syst.* **2019**, *91*, 527–535. [CrossRef]

24. Chen, J.; Lv, Z.; Song, H. Design of personnel big data management system based on blockchain. *Future Gener. Comput. Syst.* **2019**, *101*, 1122–1129. [CrossRef]
25. Li, W.; Tian, W.; Yan, Z.; Li, Z.; Gao, J.; Wu, F.; Ren, J. CoralDB: A Collaborative Database for Data Sharing Based on Permissioned Blockchain. *IEEE Trans. Mob. Comput.* **2024**, *23*, 8886–8901. [CrossRef]
26. Go Ethereum. go-ethereum (Geth)—Official Go Implementation of the Ethereum Protocol. 2026. Available online: https://geth.ethereum.org/ (accessed on 9 May 2026).
27. MultiChain. MultiChain: Enterprise Blockchain Platform. 2025. Available online: https://www.multichain.com/ (accessed on 15 October 2025).
28. EOSIO. EOSIO: An Open-Source Blockchain Platform. 2022. Available online: https://github.com/EOSIO/eos (accessed on 27 October 2025).
29. Antelope. Antelope: Open Framework for Building Fast, Secure, and User-Friendly Web3 Applications. 2025. Available online: https://antelope.io/ (accessed on 27 October 2025).
30. IOTA Foundation. IOTA. 2026. Available online: https://www.iota.org/ (accessed on 3 April 2026).
31. Sedi Nzakuna, P.; Paciello, V.; Lay-Ekuakille, A.; Kuti Lusala, A.; Dello Iacono, S.; Pietrosanto, A. From IOTA Tangle 2.0 to Rebased: A Comparative Analysis of Decentralization, Scalability, and Suitability for IoT Applications. *Sensors* **2025**, *25*, 3408. [CrossRef]
32. Hyperledger Fabric. The Ordering Service. 2025. Available online: https://hyperledger-fabric.readthedocs.io/en/latest/orderer/ordering_service.html (accessed on 30 October 2025).
33. Ethereum Foundation. The Merge. 2025. Available online: https://ethereum.org/roadmap/merge/ (accessed on 29 September 2025).
34. Ethereum Foundation Protocol Support Team. Ropsten, Rinkeby & Kiln Deprecation Announcement. 2022. Available online: https://blog.ethereum.org/2022/06/21/testnet-deprecation (accessed on 30 October 2025).
35. Chowdhury, M.J.M.; Colman, A.; Kabir, M.A.; Han, J.; Sarda, P. Blockchain versus Database: A Critical Analysis. In *Proceedings of the 2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), New York, NY, USA, 1–3 August 2018*; IEEE: New York, NY, USA, 2018; pp. 1348–1353.
36. Vo, H.T.; Kundu, A.; Mohania, M.K. Research Directions in Blockchain Data Management and Analytics. In Proceedings of the Proceedings of the 21st International Conference on Extending Database Technology (EDBT 2018), Vienna, Austria, 26–29 March 2018; pp. 445–448.
37. Bergman, S.; Asplund, M.; Nadjm-Tehrani, S. Permissioned blockchains and distributed databases: A performance study. *Concurr. Comput. Pract. Exp.* **2020**, *32*, e5227. [CrossRef]
38. Wang, Y.; Hsieh, C.H.; Li, C. Research and Analysis on the Distributed Database of Blockchain and Non-Blockchain. In *Proceedings of the 2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), Chengdu, China, 10–13 April 2020*; IEEE: New York, NY, USA, 2020; pp. 307–313. [CrossRef]
39. Dinh, T.T.A.; Liu, R.; Zhang, M.; Chen, G.; Ooi, B.C.; Wang, J. Untangling Blockchain: A Data Processing View of Blockchain Systems. *IEEE Trans. Knowl. Data Eng.* **2018**, *30*, 1366–1385. [CrossRef]
40. Gilbert, C.; Gilbert, M.A. The integration of blockchain technology into database management systems for enhanced security and transparency. *Int. Res. J. Adv. Eng. Sci.* **2024**, *9*, 316–334.
41. Schuhknecht, F.M.; Sharma, A.; Dittrich, J.; Agrawal, D. ChainifyDB: How to Blockchainify Any Data Management System. *arXiv* **2019**, arXiv:1912.04820. [CrossRef]
42. Hyperledger Besu. Hyperledger Besu Documentation. 2025. Available online: https://besu.hyperledger.org (accessed on 8 February 2025).
43. Nomic Foundation. Hardhat—Ethereum Development Environment for Professionals. 2025. Available online: https://hardhat.org (accessed on 4 November 2025).
44. Foundry. Foundry—Ethereum Development Framework. 2025. Available online: https://www.getfoundry.sh (accessed on 4 November 2025).
45. ConsenSys. GoQuorum Documentation. 2025. Available online: https://docs.goquorum.consensys.io (accessed on 4 November 2025).
46. Nelson, M. What is Consensys Quorum? 2021. Available online: https://consensys.io/blog/what-is-consensys-quorum (accessed on 10 February 2026).
47. ConsenSys Software Inc. Truffle Documentation. 2022. Available online: https://archive.trufflesuite.com/docs/ (accessed on 6 November 2025).
48. ConsenSys Software Inc. Ganache Overview. 2022. Available online: https://archive.trufflesuite.com/docs/ganache/ (accessed on 6 November 2025).
49. Hyperledger Labs. Minifabric. 2023. Available online: https://github.com/hyperledger-labs/minifabric (accessed on 4 November 2025).

50. Hyperledger Besu. Configure QBFT Consensus. 2025. Available online: https://besu.hyperledger.org/private-networks/how-to/configure/consensus/qbft (accessed on 9 September 2025).
51. Delladetsimas, A.P.; Papangelou, S.; Iosif, E.; Giaglis, G. Leadership Uniformity in Timeout-Based Quorum Byzantine Fault Tolerance (QBFT) Consensus. *Big Data Cogn. Comput.* **2025**, *9*, 196. [CrossRef]
52. Okonkwo, K. Consensys Announces the Sunset of Truffle and Ganache and New Hardhat Partnership. 2023. Available online: https://consensys.io/blog/consensys-announces-the-sunset-of-truffle-and-ganache-and-new-hardhat (accessed on 4 November 2025).
53. Truffle Suite. Truffle Suite—The Most Comprehensive Suite of Tools for Smart Contract Development. 2022. Available online: https://archive.trufflesuite.com (accessed on 4 November 2025).
54. YCharts. Bitcoin Blockchain Size. 2026. Available online: https://ycharts.com/indicators/bitcoin_blockchain_size (accessed on 8 May 2025).
55. YCharts. Ethereum Chain Full Sync Data Size. 2026. Available online: https://ycharts.com/indicators/ethereum_chain_full_sync_data_size (accessed on 8 May 2025).
56. Karasavvas, K. Revoking records in an immutable ledger: A platform for issuing and revoking official documents on public blockchains. In *Proceedings of the 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), Zug, Switzerland, 20–22 June 2018*; IEEE: New York, NY, USA, 2018; pp. 105–111.