uBSaaS: A Unified Blockchain Service as a Service Framework for Streamlined Blockchain Services Integration

Huynh Thanh Thien Pham[1][a], Frank Jiang[1], Lei Pan[1][b], Alessio Bonti[2][c] and Mohamed Abdelrazek[3][d]

[1] *School of IT, Deakin University, Burwood, Victoria, Australia*

[2] *IBM Australia, Melbourne, Australia*

[3] *A2I2D, Deakin University, Burwood, Victoria, Australia*

{htpham, frank.jiang, l.pan}@deakin.edu.au, alex.bonti@ibm.com, mohamed.abdelrazek@deakin.edu.au

Keywords: Blockchain Application Development, Blockchain Service as a Service, Microservices Architecture, Cross-Chain Interoperability, Empirical Study.

Abstract: Blockchain application development remains complex and costly due to specialized cryptographic requirements and platform-specific protocols. Existing solutions often provide only isolated services, hindering cross-chain interoperability and limiting broader adoption. This paper addresses these gaps by introducing *SChare*, a platform founded on a unified Blockchain Service as a Service (*uBSaaS*) framework that abstracts blockchain-intensive tasks into microservices. This architecture enables developers to integrate blockchain features into applications as seamlessly as any third-party service, while supporting orchestration across multiple blockchain networks for enhanced flexibility. We evaluate the platform through an experimental cross-chain application to demonstrate feasibility and scalability. Additionally, a developer study involving hands-on usage and post-study assessments highlights SChare's effectiveness in reducing both the steep learning curve and overall development overhead. The results indicate that SChare facilitates more accessible blockchain development, thereby encouraging wider adoption. This approach advances the state of the art by unifying platform-specific capabilities, fostering interoperability, and offering a scalable, microservices-based solution for blockchain application development.

1 INTRODUCTION

The emergence of blockchain technology, initiated by the publication of the Bitcoin whitepaper (Nakamoto, 2008), has revolutionized digital transactions by promoting decentralization, trust, immutability, and transparency (Pilkington, 2016). Building on these principles, smart contracts (SCs) (Wood, 2014) enable secure, automated agreements among untrusted parties, thereby transforming traditional business processes. Despite this potential, blockchain development (BD) remains complex due to high costs, platform-specific skill requirements, and limited interoperability (Antonucci et al., 2019; Virmani and Singh, 2024; Prewett et al., 2020; Li et al., 2021; Bosu et al., 2019).

A key challenge lies in bridging the gap between diverse blockchain platforms—such as Ethereum (Solidity), Algorand (Teal), and Solana (Rust)—where each employs different languages and requires specialized expertise (Antonopoulos and Wood, 2018; Algorand, 2024b; Solana, 2024). Additionally, the immutable nature of blockchain amplifies the consequences of defects, while inefficient code can incur high transaction fees (Wessling et al., 2019; Marchesi et al., 2020; Chen et al., 2017). Existing approaches address certain facets of BD, but a notable gap persists in offering a unified, cross-chain development framework that abstracts technical complexities, reduces the learning curve, and standardizes core blockchain services.

In this paper, we address the gap by introducing a unified Blockchain Service as a Service (uBSaaS) framework, exemplified by its implementation, the SChare platform. This platform leverages microservices to encapsulate blockchain functionality, easing integration and fostering innovation.

[a] https://orcid.org/0000-0001-5427-6940
[b] https://orcid.org/0000-0002-4691-8330
[c] https://orcid.org/0000-0003-2240-0454
[d] https://orcid.org/0000-0003-3812-9785

Pham, H. T. T., Jiang, F., Pan, L., Bonti, A. and Abdelrazek, M.
uBSaaS: A Unified Blockchain Service as a Service Framework for Streamlined Blockchain Services Integration.
DOI: 10.5220/0013335800003928
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2025)*, pages 107-119
ISBN: 978-989-758-742-9; ISSN: 2184-4895
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

The remainder of the paper is organized as follows: Section 2 surveys related work; Section 3 discusses blockchain application development and corresponding challenges; Section 4 details the uBSaaS framework design; Section 5 describes its implementation; Section 6 examines its evaluation and results; Section 7 outlines potential threats to validity; and Section 8 concludes the paper.

2 RELATED WORK

The concept of Smart Contract as a Service (SCaaS) has gained traction, it was outlined in (Sun et al., 2024), which presents a framework to facilitate SC reuse within the Web3 ecosystem, thereby potentially streamlining development efforts. The study identified several key use cases where SCaaS could add significant value, including digital assets, liquidity pools, exchanges, and various types of decentralized applications (DApps). There are several projects, such as SEIF (Seif.org, 2023) and BitRegalo (Bitregalo.com, 2023), that have explored SCaaS offerings with varied effectiveness and operational statuses. SEIF, while conceptually promising, lacks comprehensive information on its current operational state. BitRegalo, launched in August 2023, offers core SC templates designed for customization and deployment on the Ethereum blockchain. However, its reliance on the factory SC pattern presents cost challenges (CSIRO's Data61, 2024), and its focus on Ethereum limits cross-platform applicability. To address these constraints, our proposed framework encapsulates use-case-specific templates within packaged services, deploying SCs through SDKs or libraries provided by respective blockchain platforms, thereby avoiding excessive deployment costs.

Blockchain as a Service (BaaS) solutions like IBM Blockchain (IBM Blockchain, 2016), Ethereum Blockchain on Azure (Microsoft, 2018), Microsoft Azure Blockchain (Microsoft, 2017), and R3 Corda (R3, 2016) offer foundational functionalities for blockchain management and deployment. These traditional BaaS models operate within the established three-tier cloud architecture—Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Platforms like Kaleido (Kaleido.io, 2024) extend these capabilities by providing enterprise-focused solutions that support large-scale blockchain applications through customized network deployment. However, BaaS solutions often have limited flexibility, constrained by dependencies on a single cloud provider and lacking development tools for building on deployed networks. To address these limitations, a unified blockchain as a service (uBaaS) platform was proposed in (Lu et al., 2019), introducing a more flexible service model that supports blockchain network creation and deployment alongside application development on these networks. The feasibility of uBaaS was validated through a quality-tracing use case, demonstrating its effectiveness in practical applications. Similarly, NBF BaaS (Grandhi et al., 2023) supports blockchain e-governance with the bring your own infrastructure (BYOI) feature, which allows developers to deploy networks on their chosen infrastructure. Additionally, the Functional Blockchain as a Service (FBaaS) framework, introduced in (Chen and Zhang, 2018), leverages a serverless architecture to enhance blockchain performance and abstraction, representing a promising shift from traditional BaaS models.

Building on these advancements, our proposed framework adopts the MSA to introduce a uBSaaS model. The uBSaaS model abstracts complex blockchain functionalities, allowing developers to integrate blockchain services into applications without extensive blockchain expertise. By leveraging MSA, uBSaaS supports flexible deployment of blockchain microservices, optimized operation, and reduced computational overhead, while also enabling seamless orchestration of services across multiple blockchain platforms.

3 CHALLENGES IN BLOCKCHAIN DEVELOPMENT

This section provides an overview of the blockchain application development process, which is illustrated in Fig. 1. Accordingly, it involves writing and compiling SCs before transmitting them to the blockchain via transactions. Once deployed, SCs are governed by the network's consensus mechanism and can be invoked using certain APIs and SDKs, referencing their unique identifiers. Throughout this process, composing SCs and configuring deployment procedures require a profound understanding of blockchain technology, encompassing programming languages, tools, and SC SDKs. Furthermore, given the considerable variance among these components across different blockchain networks, the learning curve is significantly exacerbated. This causes difficulties even for blockchain developers (Sharma et al., 2023), thus, the time and effort that non-blockchain developers need to commit would be significant, not to mention security risks in SCs due to limited domain experience.

Table 1: Related work.

Source	Architecture	Blockchain(s)	Use cases
(Lu et al., 2019)	uBaaS	Permissioned blockchains	Platform addressing cloud vendor lock and enabling application development
(Sun et al., 2024)	SCaaS	Ethereum	Service for SC reusing
(Seif.org, 2023)	N/A	N/A	Legal SC as a service platform
(Bitregalo.com, 2023)	SCaaS	Ethereum	SCaaS using factory SC pattern
(IBM Blockchain, 2016)	BaaS	Hyperledger Fabric	Generic cloud-based BaaS
(Microsoft, 2018)		Ethereum	
(Microsoft, 2017)		Quorum	
(R3, 2016)		Corda	
(Kaleido.io, 2024)	BaaS	Various blockchains	Cloud-based BaaS with high level of customization
(Grandhi et al., 2023)	BaaS	Hyperledger Fabric & Sawtooth	Services for blockchain e-Governance use cases enabling the BYOI feature
(Chen and Zhang, 2018)	FBaaS	Various blockchains	Enhanced consortium blockchain performance with serveless architecture
This study	uBSaaS	Public blockchains	MSA-based unified blockchain services as a service framework

Imagine a project that involves multiple blockchain platforms, the integration of services from multiple blockchain networks would incur notable challenges in terms of complexity and required resources.

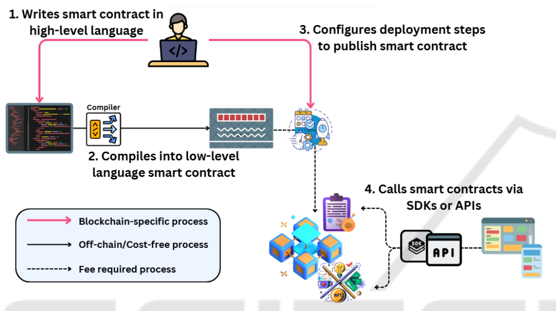
1. Writes smart contract in high-level language
3. Configures deployment steps to publish smart contract
Compiler
2. Compiles into low-level language smart contract
4. Calls smart contracts via SDKs or APIs
API
Blockchain-specific process
Off-chain/Cost-free process
Fee required process

Figure 1: Smart contract development process.

Suboptimal SCs, often due to limited experience, can incur extra execution costs in Ethereum (Chen et al., 2017). There is also a lack of interoperability between blockchain projects, which use various programming languages, consensus mechanisms, and protocols (Monrat et al., 2019). This further exacerbated the complexity of BD considering the difficulty in implementing and executing SCs remained challenging according to a critical literature analysis (Upadhyay, 2020). Extending on that, an empirical survey highlighted that non-blockchain developers identified higher defect costs, decentralized environments, and significant technological complexity among their main challenges, despite their experience in conventional software development (Bosu et al., 2019). Additionally, SC immutability complicates application design, requiring detailed planning from the outset. Limited knowledge of blockchain programming languages exacerbates execution failures, security vulnerabilities, and privacy risks, which often attract hackers. The steep learning curve, coupled with insufficient supporting tools, discourages both industrial managers and developers from transitioning to blockchain (Gurzhii et al., 2023). To address these challenges, developers need comprehensive libraries to reduce code redundancy and avoid "reinventing the wheel". Frameworks are required to facilitate reusable components and organize them into user-friendly classes and methods, easing BD (Zou et al., 2021). Furthermore, many SC developers lack domain expertise, emphasizing the need for a platform that supports the creation and management of domain-specific SC templates or similar solutions. Such a platform should provide reliable, pre-validated templates that are easy to integrate into applications, reducing the burden on developers. By abstracting the complexities of BD, this platform could empower conventional software developers to concentrate on building impactful and functional applications without needing extensive blockchain expertise. Improved tools are also necessary to support blockchain integration with existing systems, fostering wider adoption (Vacca et al., 2021). Current software development tools designed for traditional development are inadequate for blockchain systems, creating a pressing need for new or enhanced tools tailored to blockchain environments (Bosu et al., 2019).

4 THE uBSaaS FRAMEWORK

The objectives of the proposed uBSaaS framework are (1) to streamline the integration of blockchain services into any application, removing the requirement for specialized BD skills; and (2) to facilitate the simple orchestration of multiple services built on different blockchain platforms. This approach is supported by the MSA, which ensures flexibility, availability, and scalability (Shadija et al., 2017; Yu et al., 2018) in providing multiple blockchain services. Furthermore, it enables the division of development tasks, eliminating the necessity for application developers to possess extensive blockchain expertise. Subsequent subsections provide comprehensive insights into the uBSaaS framework.

4.1 Overview

In the MSA, each microservice functions as an autonomous process dedicated to a specific aspect of the application, creating a distributed architecture known for its precision and loose coupling (Bakshi, 2017; Knoche and Hasselbring, 2018). These microservices communicate asynchronously, typically via HTTP REST or message buses, facilitating the continuous, efficient, and autonomous deployment of application functionalities (Yu et al., 2018). Leveraging these traits, MSA has been widely adopted to bolster the scalability and security of applications. The autonomy of each microservice enables greater flexibility in selecting development platforms, while communication protocols are simplified without requiring middleware (Krylovskiy et al., 2015). This study advocates for the integration of MSA with blockchain technology, proposing the utilization of blockchain-enabled microservices.

In our framework, each microservice is built to abstract the blockchain-intensive development tasks delineated in Section 3. Blockchain application builders only need to supply the requisite parameters and access the desired blockchain-enabled service via standard REST API calls. This streamlined approach enables the integration of blockchain services without extensive blockchain expertise such as SC writing, compilation, and deployment. Consequently, this methodology substantially simplifies the process for developers with limited BD experience, fostering a more efficient, secure, and streamlined development trajectory. It is also promising that such a solution could effectively reduce barriers to blockchain technology adoption (Marengo and Pagano, 2023). Moreover, the framework endeavors to minimize the security risks in SCs written by inexperienced developers by advocating for the reuse of contract-level code. Seasoned blockchain developers suggested this approach and reported this as one of the most effective practices to prevent security issues in SCs (Wu, 2019; Wan et al., 2021; Sharma et al., 2022). To facilitate the analysis and evaluation of the uBSaaS framework, we have implemented it into the SChare platform whose architecture is presented in the following section.

4.2 Architecture

SChare is architected upon the microservices paradigm, with blockchain services operating independently from the main server. The main server of SChare, as depicted in Fig. 2, encompasses several integral components, working cohesively to deliver on-demand blockchain services to client applications. These components include:

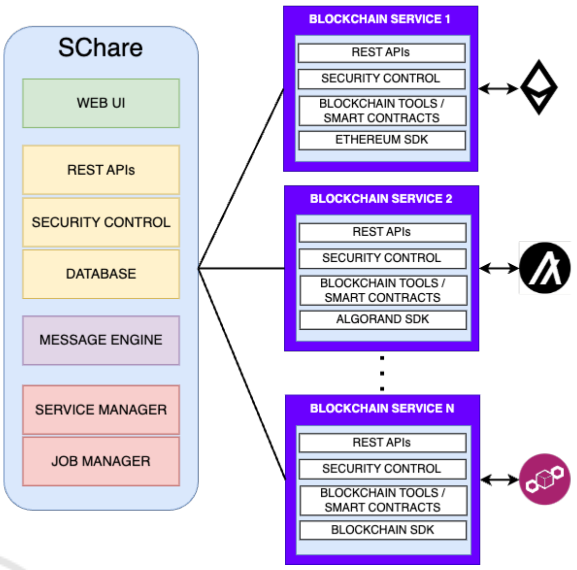

Figure 2: Architectural design of the SChare platform.

- **Web UI:** Providing an intuitive and user-friendly front-end interface, the Web UI simplifies service discovery and basic profile-related tasks for service consumers. Emphasizing ease of use, this interface ensures efficient navigation and interaction with services, while also offering insights into underlying blockchain SCs and service consumption guidelines.
- **API Gateway:** SChare features a robust API Gateway, proficiently exposing available services on the platform. Serving as a pivotal conduit, it facilitates the smooth integration of blockchain services into client applications.
- **Security Control:** Combining the API Gateway with authentication and authorization functions, security control is implemented using a basic key-based mechanism. This regulates consumer access to different services and functionalities.
- **Database:** A dedicated database stores platform user data, service information, and job records.
- **Service Manager:** This manager handles the registration and connection of blockchain microservices. It facilitates the addition of services to the platform, making them available for consumption by client applications.
- **Job Manager:** Responsible for processing and managing jobs generated when services are consumed, this manager ensures seamless integration of available blockchain services. It operates us-

ing an asynchronous mechanism, facilitated by the Message Engine.

- **Message Engine:** Employed to communicate with consuming applications, the Message Engine facilitates asynchronous operations. Leveraging a publish-and-subscribe channel for event-based messaging mitigates service locks caused by the prolonged waiting time for blockchain operations.

4.3 Blockchain Microservice

While SChare simplifies BD for seamless integration into client applications, underlying blockchain microservices play a critical role in configuring optimal setups and executing necessary blockchain transactions. To foster consistency and adherence to best practices, we advocate for a standardized architectural configuration of blockchain microservices. This proposed setup, presented as comprehensive guidelines, assists potential blockchain developers in constructing services aligned with a uniform structure, ensuring compatibility with SChare for seamless consumption.

As depicted in Fig. 2, each microservice should offer accessibility through a well-documented REST API, specifying required parameters and expected data returns to enable smooth consumption and error handling by client applications. Complementing this API, a carefully crafted deployment script should oversee the testing, compilation, and deployment of SCs and/or other blockchain tools. These microservices ought to incorporate pre-designed SC templates or blockchain service procedures, featuring predefined functions and dynamic variables. This standardized yet flexible approach facilitates consistent implementation across diverse consumer scenarios, streamlining integration and utilization of blockchain services for various use cases. For instance, a digital asset service should support default actions like creation and transfer, while also allowing users to define attributes such as asset name, owner, and metadata, enabling its adoption in contexts like non-fungible token marketplaces, digitalized intellectual property, and certifications.

Effective interaction with blockchain networks necessitates utilizing appropriate Software Development Kits (SDKs), which facilitate connection to blockchain platforms and management of transaction-related activities. SDKs typically assist in constructing, signing, and sending transactions to the blockchain network, and often include features for network discovery and specialized tools for BD.

4.4 System Workflow

The system workflow of the SChare platform, illustrated through a sequence diagram (Fig. 3), outlines how a client application utilizes a blockchain service. This section discusses the processes involved.

Initially, developers should utilize the *WebUI* of the SChare platform to browse available services, understand essential details provided by the *ServiceManager*, and acknowledge necessary parameters and usage guidelines. Furthermore, they can generate an API key crucial for subsequent service authorization. The technical interactions start with the client application sending an API request to SChare, equipped with predefined parameters that include both platform-specific and blockchain-specific elements gathered during the initial exploration. Upon receipt, the *SecurityControl* component validates these parameters and verifies the caller's authorization. Depending on the validation outcome, a response is sent back. If validation fails, the request is denied and the process terminates. Conversely, successful validation prompts the creation of a job record in the database, and a unique *jobId* is generated and returned to the client, enabling subscriptions to the cloud messaging channel initiated by SChare's *MessageEngine*.

Concurrently, the *JobManager* activates the job by calling the required blockchain service with the blockchain inputs provided by the client. During its operation, the blockchain microservice validates the parameters and updates the job record with the request's status and data if any. Invalid parameters result in an error message being dispatched to the client via the cloud messaging channel, terminating the process with a failure. On the other hand, valid parameters proceed with database updates on SChare's job record. The *MessageEngine* then subscribes to a topic identified by the *jobId* on a predefined pub/sub channel that links SChare to the blockchain microservice. The blockchain service processes the request and sends the required transactions to the blockchain network.

After being processed by the blockchain network, the outcomes, which may include transaction hashes, SC addresses, or digital asset identifiers, are published back to the *jobId* topic on the pub/sub channel. These results update the job's records in the *Database* on SChare. SChare's *MessageEngine* then informs the client application of the results via the cloud messaging channel. This communication, along with the pub/sub channel, is temporarily closed post-operation, readying the system for future requests. By integrating cloud messaging and pub/sub channels,

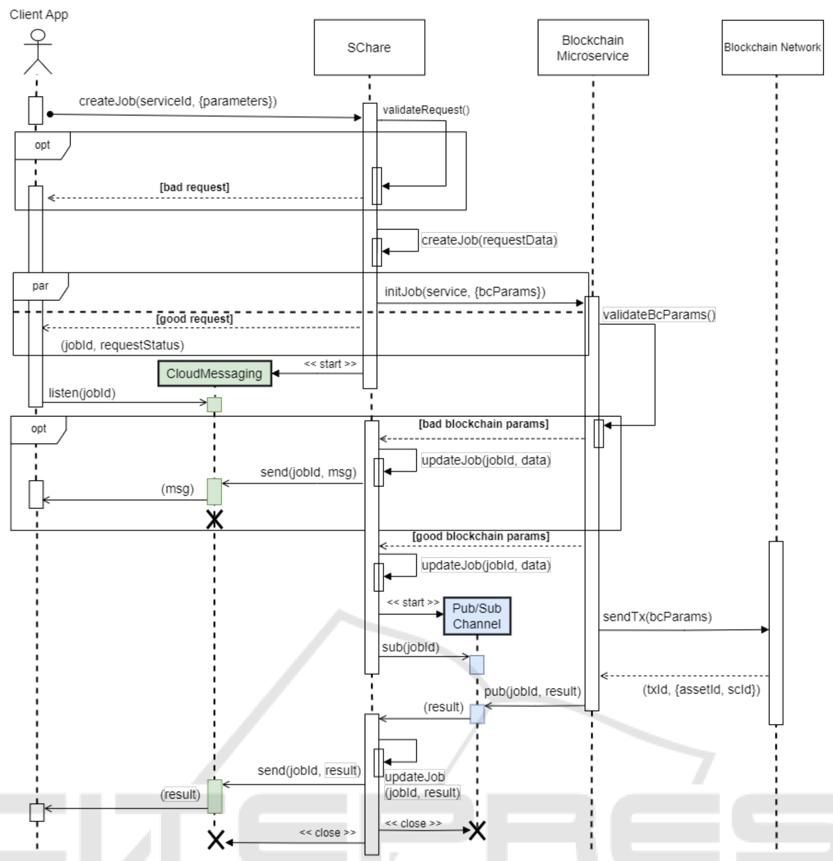

Figure 3: The workflow of a client application consuming a blockchain microservice on SChare.

SChare leverages an event-based model to facilitate asynchronous operations. This approach aims to significantly reduce wait times associated with executing multiple blockchain transactions, thereby enhancing the user experience for client applications.

5 FRAMEWORK IMPLEMENTATION

Fig. 4 presents a class diagram showing the implementation design of the uBSaaS, equivalent to the SChare platform. *JobManager* maintains and manages each created *Job*, while *ServiceManager* stores information and maintains available *BlockchainService* on the platform. The *JobManager* creates each *Job* from a combination of a *Request* sent in by a client application and a *BlockchainService* specified by its identifier with relevant parameters. *JobManager* and *BlockchainService* co-establish a *MessageChannel* for each created *Job* until it is completed or failed. For each *BlockchainService*, there may be one or more pre-built *SmartContract*s ready for deployment, one or more *BlockchainTool*s configured in a specific manner to perform blockchain-related tasks, or, in some cases, a combination of both.

We developed the SChare platform and its blockchain microservices using Node.js[1] for its robust asynchronous capabilities and extensive support in multiple blockchain ecosystems. Docker was employed to containerize the microservices, ensuring portability and consistent deployment across diverse environments. The platform runs on a cloud-based VM infrastructure, with SChare on an instance featuring 1 CPU and 1 GB RAM, the Web UI on 1 CPU and 256 MB RAM, and each blockchain microservice on 1 CPU and 256 MB RAM. Multiple Ethereum[2] and Algorand[3] services are provided, such as NFT Minter (Algorand), NFT Standard Minter (Ethereum), and NFT Express Minter (Ethereum). To contain costs and preserve realism, all interactions occur on test networks: Sepolia for Ethereum and Algorand Testnet for Algorand. SCs for Ethereum-based services

[1]https://nodejs.org/
[2]https://ethereum.org/
[3]https://developer.algorand.org/

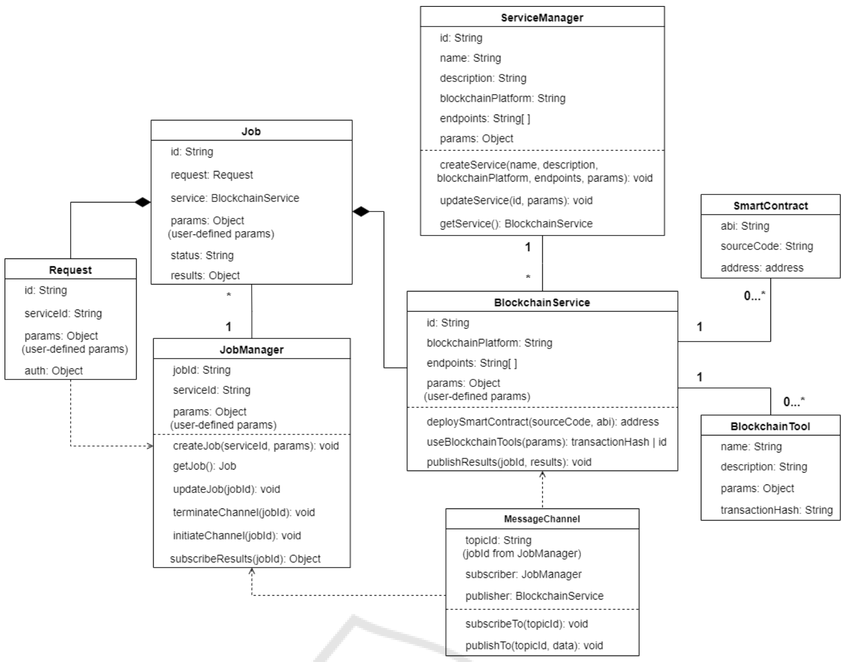
ServiceManager
id: String
name: String
description: String
blockchainPlatform: String
endpoints: String[ ]
params: Object
createService(name, description, blockchainPlatform, endpoints, params): void
updateService(id, params): void
getService(): BlockchainService
Job
id: String
request: Request
service: BlockchainService
params: Object (user-defined params)
status: String
results: Object
Request
id: String
serviceId: String
params: Object (user-defined params)
auth: Object
JobManager
jobId: String
serviceId: String
params: Object (user-defined params)
createJob(serviceId, params): void
getJob(): Job
updateJob(jobId): void
terminateChannel(jobId): void
initiateChannel(jobId): void
subscribeResults(jobId): Object
BlockchainService
id: String
blockchainPlatform: String
endpoints: String[ ]
params: Object (user-defined params)
deploySmartContract(sourceCode, abi): address
useBlockchainTools(params): transactionHash | id
publishResults(jobId, results): void
SmartContract
abi: String
sourceCode: String
address: address
BlockchainTool
name: String
description: String
params: Object
transactionHash: String
MessageChannel
topicId: String (jobId from JobManager)
subscriber: JobManager
publisher: BlockchainService
subscribeTo(topicId): void
publishTo(topicId, data): void
1
*
1
0...*
1
0...*

Figure 4: Class diagram of the SChare platform.

are written in Solidity using Remix IDE[4], whereas Algorand services leverage PyTeal[5] with Visual Studio Code.

6 FRAMEWORK EVALUATION

In this section, we evaluate the feasibility, scalability, and effectiveness of the proposed uBSaaS framework. First, we assess the utility and performance of the SChare platform by building an application using available blockchain-enabled services running on it. Then, we measured the time it took to process each bridging request to analyze its scalability. This experiment focused on the possibility of creating a cross-blockchain NFT bridging application, utilizing multiple services on SChare, and practically demonstrated the SChare platform's utility and scalability to enable multiple blockchain-enabled services integration in building cross-chain applications. In addition, we also assess the effectiveness of the SChare platform in assisting builders in developing blockchain applications through a user study. In the user study, participants were enabled to interact with the running SChare platform as well as assess and analyze the implemented code base, before we asked them to attempt to replicate the NFT bridging application, from which the evaluation of the SChare platform's effectiveness was derived. For transparency and reproducibility, the SChare platform is made available online[6].

6.1 Sample Application Development

We developed an experimental application to showcase SChare's capability to integrate multiple blockchain-enabled services across multiple blockchain platforms. In this study, we developed a cross-chain NFTs bridging application. We named the experimental application as NFTs Bridge. The application took the effort to simulate the 'burn-and-mint' approach in offering cross-chain NFTs[7] on two blockchains, Ethereum and Algorand. The process of the NFTs Bridge application is illustrated in Fig. 5. Accordingly, the NFTs Bridge application utilized two services from SChare for token creation while the token-burning process, a part of the swap mechanism, was managed directly on the application.

The first service, based on Ethereum, implemented an NFT SC conforming to the ERC-721 standard suggested by OpenZeppelin[8]. The second service used the Algorand Standard Asset (ASA)(Algorand, 2024a), a unique feature of the Algorand blockchain for digital assets, following the

[4]https://remix.ethereum.org/

[5]https://pyteal.readthedocs.io/en/stable/

[6]https://baas-db.fly.dev

[7]https://chain.link/education-hub/cross-chain-nft#how-do-cross-chain-nfts-work

[8]https://docs.openzeppelin.com/contracts/3.x/erc721

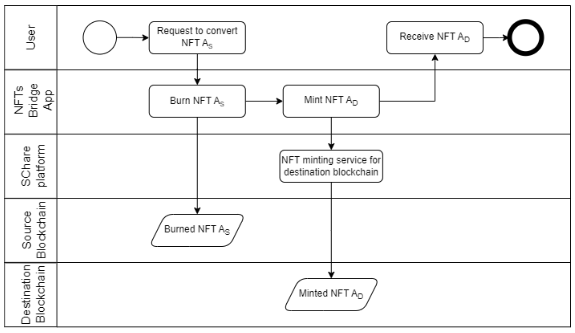

Figure 5: Burn-and-mint NFTs bridging process.

ARC3 standard provided by the Algorand Foundation[9]. These services enabled the NFTs Bridge application to create NFTs on Ethereum and Algorand networks. Before minting a relevant NFT on the destination blockchain network, the converted token was burned via direct transactions constructed with the source blockchain's SDKs and the token owner's signature on the NFTs Bridge application. Building a backend middleware could be beneficial for efficiently managing interactions with SChare. However, to emphasize SChare's seamless integration capabilities and simplify the implementation of the experiment, the NFTs Bridge application was developed using a serverless model. Listing 1 shows the functions implemented in the NFTs Bridge application for the burning and minting of NFTs on Ethereum and Algorand.

After building the application, we measured its performance in terms of processing time to convert an NFT in both directions, from Ethereum to Algorand, and vice versa. Fig. 6 presents the amount of time taken in each attempt from the start to the end of the process depicted in Fig. 5. Starting from an NFT with one attribute, we increased an additional attribute in every consecutive attempt. In general, the performance was reasonable with the average execution time for conversions from Ethereum to Algorand being 24.52 seconds, while the average for the reverse direction was 23.94 seconds. However, the execution time for both directions of conversion increased linearly with the number of attributes of the NFT, showing a good level of scalability. Additionally, considering that the bridging process involves the burning of an NFT on the source blockchain, which might need multiple transactions on the source blockchain, and the creation of an NFT on the destination blockchain, which might be equivalent to the deployment of a SC, the performance of the implemented NFTs Bridge application showed comparatively scalability in reference with the framework proposed in (Lu et al.,

[9]https://github.com/algorandfoundation/ARCs

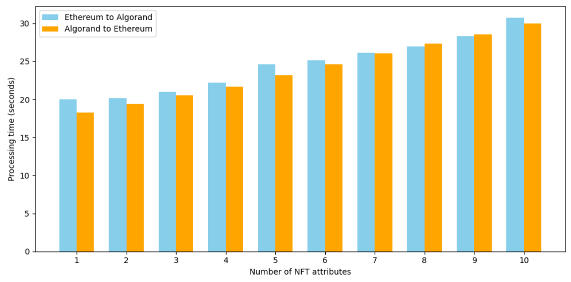

Figure 6: NFTs Bridge application's performance.

2019). In addition, the NFTs Bridge application was connected to the Sepolia testnet and Algorand testnet on which network traffic was similar to the mainnets, thus, showing the feasibility of building scalable cross-chain applications on public blockchains using the SChare platform.

```
const mintAlgoNFT = async () => {
    await burnAsset();
    tokenDataObj.jobName = "mintAlgoNFT_fromNFTsBridge";
    tokenDataObj.serviceID = process.env.REACT_APP_ALGO_NFT_SERVICE;
    tokenDataObj.datafileURL.json = props.tokenData;
    const result = await API.createJob(tokenDataObj);
    if (result.success) {
      props.setTokenData({})
    }
}
const mintEthNFT = async () => {
    await burnAsset();
    tokenDataObj.jobName = "mintEthNFT_fromNFTsBridge";
    tokenDataObj.serviceID = process.env.REACT_APP_ETH_NFT_SERVICE;
    tokenDataObj.datafileURL.json = props.tokenData;
    const result = await API.createJob(tokenDataObj);
    if (result.success) {
      props.setTokenData({})
    }
}
```

Listing 1: SChare usage in NFTs Bridge application.

6.2 User Study

To further validate the effectiveness of the SChare platform, we invited developers to interact with it. Participants were asked to test and analyze the platform. Our user study involved developers with various levels of public BD experiences to gather diverse insights into the platform's efficacy. The user study design took previous studies as references (Chakraborty et al., 2018; Bosu et al., 2019; Sharma et al., 2022), and referred to (Ko et al., 2015) as a guideline for recruiting participants.

Sampling Strategy. We specifically target participants who can develop with JavaScript and the React framework. In terms of experience, the participants could be those ranging from developers and researchers with at least basic experience in API usage and service integration, extending to those specializing in the blockchain sector. This diverse group ensures a comprehensive evaluation of the SChare platform from a wide variety of perspectives. To recruit suitable participants, we employed various strategies: (1) reaching out to developers on LinkedIn with relevant skills, (2) leveraging snowball sampling from industry contacts, and (3) posting in developer forums on platforms like Discord and Telegram. After reaching out, 15 participants agreed to join the study.

Platform Interaction. Each participant attended an in-person interactive session to explore and engage with the SChare platform. The session began with an introduction to the study, followed by a demonstration of the NFTs Bridge application. We explained the burn-and-mint mechanism of the NFT bridge without revealing the full application implementation. Next, we presented the template code for the application, which includes all features of the NFTs Bridge application except the two NFT minting functions. Participants were then asked to estimate how long it would take to replicate the NFTs Bridge application using their own approaches and the provided template. These responses were recorded for later analysis, as shown in Table 2, along with each participant's development experience. Participants' estimates took into account the time needed to learn and understand the process of minting an NFT on both Ethereum and Algorand. Those with prior experience on either platform estimated shorter completion times, while those with less BD experience generally estimated longer completion times.

After recording the estimations, we introduced the SChare platform and its features to the participants, presenting them with the SChare platform's web interface, detailing its features including service and job instantiation, and API key generation. This aimed to provide a hands-on understanding of SChare's features and usage similar to the process where developers have to go through the documentation before integrating a third-party service. After that, the participants were asked to test and interact with the SChare platform for a maximum of 30 minutes and raise any questions to the research team before continuing to the next phase of the study.

After the participants engaged with the platform, we asked them to replicate the NFTs Bridge application utilizing the SChare platform. To simplify the task and preserve the consistency of the study, we provided them with the identical template code presented earlier. Therefore, the assigned task was to use the template and available services on the live SChare platform to implement the rest of the NFTs Bridge application. This approach intended to isolate other tasks that were not necessary for participants to perform, allowing us to focus on evaluating the core functionality of the platform rather than troubleshooting unrelated issues. All participants were asked to use the same computer provided by the research team and connect to the same network in the research team's lab room. Once the participant completed the task, they informed the research team who would record the actual completion time to measure the Efficiency Realization Ratio (ERR). ERR is a simple metric designed to assess the impact of using SChare on implementing the blockchain functionalities required in our study. It is calculated as the ratio between the delta of estimated and actual time taken by participants to complete the task using the SChare platform and their estimated time to complete the same task without awareness of the SChare platform. The ERR serves as an indicator of the estimated efficiency gains enabled by SChare, with a value close to 1 indicating a significant reduction in task completion time compared to the estimated effort without the use of SChare. This metric helps demonstrate the practical benefits of SChare in simplifying and expediting complex BD tasks.

Post-Interaction Assessment. After the interaction session, we asked the participants to provide comments on their experience after using the SChare platform to complete the task. This aimed to seek participant feedback on usability or error rates in implementation, to create a more holistic evaluation of platform effectiveness.

6.3 Results

At the end of the interactive session, all participants managed to complete the task independently, leading to a 100% completion rate. However, their completion times varied. Table 3 details the actual time taken by each participant to complete the assigned task. Based on their anticipation before being introduced to the SChare platform, the ERR was also calculated.

The actual completion time of the participants varied from 18 to 57 minutes, however, the ERR was

Table 2: Participants' development experiences and estimation of implementation time.

Participant	Profession	SE exp.	Blockchain dev exp.	Estimated completion time
1	Software engineer	$>$ 5 years	$<$ 2 years	2 - 3 hours
2	Software engineer	2 - 5 years	$<$ 1 year	3 - 4 hours
3	Researcher	$>$ 5 years	$<$ 1 year	4 - 5 hours
4	Software engineer	$>$ 5 years	$>$ 5 years	1.5 - 2 hours
5	Blockchain developer	2 - 5 years	2 - 5 years	2 - 3 hours
6	Software engineer	2 - 5 years	$<$ 1 years	3 - 4 hours
7	Software engineer	$>$ 5 years	$>$ 5 years	1 - 2 hours
8	Software engineer	$<$ 2 years	$<$ 1 year	5 - 6 hours
9	Researcher	$>$ 5 years	$<$ 1 year	3 - 4 hours
10	Researcher	$>$ 5 years	$<$ 1 year	4 - 5 hours
11	Project manager	$>$ 5 years	$>$ 5 years	2 - 3 hours
12	Software engineer	$>$ 5 years	2 - 5 years	2 - 3 hours
13	Blockchain developer	2 - 5 years	$<$ 2 years	3 - 4 hours
14	Software developer	$>$ 5 years	$<$ 1 year	2 - 3 hours
15	Software engineer	$>$ 5 years	$>$ 5 years	1.5 - 2.5 hours

Table 3: Task completion time and ERR.

Participant	Actual completion time	ERR
1	$\approx$ 31 minutes	0.79
2	$\approx$ 42 minutes	0.80
3	$\approx$ 36 minutes	0.86
4	$\approx$ 23 minutes	0.78
5	$\approx$ 29 minutes	0.82
6	$\approx$ 33 minutes	0.84
7	$\approx$ 23 minutes	0.74
8	$\approx$ 57 minutes	0.83
9	$\approx$ 31 minutes	0.85
10	$\approx$ 39 minutes	0.86
11	$\approx$ 28 minutes	0.81
12	$\approx$ 30 minutes	0.80
13	$\approx$ 34 minutes	0.84
14	$\approx$ 31 minutes	0.79
15	$\approx$ 27 minutes	0.78
	Mean	0.81
	Std Dev	0.03
	Variance	0.001
	SEM	0.009

quite consistent across participants. The ERR starting from 0.74 to 0.86 across participants showed that SChare could help builders reduce from 74% to 86% development time. The calculated standard deviation of 0.03 indicates moderate variability in the ERR among participants. This level of variability suggests that while SChare helped most participants to reduce a significant amount of effort in developing the functionalities, there were some differences based on individual experiences. The high values of ERR could be seen in most participants with less experience in BD. This could be attributed to the fact that less experienced blockchain developers might benefit more from the SChare platform, which helped abstract the tasks requiring deep blockchain expertise that they would otherwise need to spend much more effort to learn and implement. Interestingly, two participants (5 and 13) with intermediate experience in BD also demonstrated high ERR values of 0.82 and 0.84, respectively. This may be due to their specialization in developing on the Ethereum blockchain, combined with a lack of familiarity with NFT development on the Algorand blockchain. As a result, they initially estimated a longer building time, but with the assistance of the SChare platform, they were able to complete the task in a much longer duration. Overall, participants, on average, realized an ERR of 0.81, with a standard deviation of 0.03. This suggests that the SChare platform was perceived as a valuable tool for significantly improving task efficiency, as most participants demonstrated similar levels of efficiency gains. The low variance (0.001) and standard error of the mean (SEM) of 0.009 further indicate consistency in participants' efficiency improvements, reinforcing the platform's capability to reduce complexities and implementation time effectively.

In addition to the survey data in Table 4, participants shared positive personal comments on SChare's potential impact on BD. BD is notoriously complex, particularly for less-experienced developers (Sharma et al., 2023), and SChare addresses this by reducing the need to engage with complex blockchain concepts. One researcher noted, "*This makes blockchain integration as simple as integrating any standard third-party service [...] I might not have to learn a lot about a new blockchain if I want my application to shift [to a new platform].*" Given the high costs and security risks of inexperienced implemen-

Table 4: Participant responses on SChare's effectiveness in addressing blockchain development challenges.

Statements	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	Addressed challenges
SChare enables non-blockchain software developers to easily integrate blockchain features into their applications without learning in-depth blockchain development	0	0	1	12	2	Steep learning curve
SChare helps minimize the developer's interactions with blockchain complex concepts to enable simpler integration.	0	0	0	10	5	Technological complexity
SChare might help avoid unwanted costs caused by inexperienced implementation on blockchain	0	1	1	9	4	High cost of defects
SChare streamlines the development process by allowing non-blockchain developers to focus on more impactful aspects of application building, rather than on blockchain-specific tasks.	0	0	2	3	10	Technological complexity
SChare could be useful for both researchers and practitioners to easily experiment new blockchain application and rapidly prototype blockchain application on any blockchain network.	0	0	1	5	9	Technological complexity, steep learning curve

tations (Chen et al., 2017), SC reuse is a common strategy among developers (Khalid and Brown, 2023; Sharma et al., 2022), which SChare facilitates. A senior engineer and project manager further remarked, "*[SChare] might reduce the need to hire expensive blockchain developers who are hard to find.*"

BD can be demanding even for seasoned developers due to stark differences from traditional software development (Bosu et al., 2019). SChare alleviates these challenges by allowing developers to focus on core value creation while relying on existing blockchain services. A blockchain developer stated, "*SChare could help more developers get into building blockchain-based applications without worrying about the steep learning curve associated with programming for different blockchains...*" Another researcher, with significant software engineering experience but limited BD skills, affirmed, "*[I] would definitely use [SChare] if I were building an application that relied heavily on some blockchain[s]...*" Additional comments underscored the platform's efficiency and ability to shorten the learning curve, with one participant noting, "*SChare ... cuts down [the] time needed to learn the prerequisites for blockchain development.*"

7 THREATS TO VALIDITY

In this study, we employed rigorous measures to ensure reliable evaluation, yet certain threats to validity persist. For internal validity, participant diversity in the user study, while valuable for broad insights, may introduce bias. To mitigate this, we standardized task conditions and collected initial completion time estimates before introducing SChare, ensuring objective measurements. While the ERR metric effectively highlights efficiency gains, it does not account for factors like ease of learning or adaptability, which were explored through qualitative feedback for a more comprehensive evaluation. On the other hand, for external validity, our sample may not fully represent all potential users, and the focus on an NFT bridging application, though practical, may limit generalizability to other blockchain functionalities. Future studies will address these limitations by broadening use cases and participant diversity to strengthen the evaluation of SChare's utility.

8 CONCLUSION

Developing blockchain applications demands specialized expertise and carries a steep learning curve, hindering broader adoption. Limited interoperability among different blockchain systems adds further complexity. To address these challenges, this study proposes a unified Blockchain Service as a Service (uBSaaS) framework that abstracts blockchain-specific elements into lightweight microservices, enabling easier integration with minimal blockchain expertise. The framework also employs asynchronous operations for seamless interactions with complex functionalities.

We evaluated this approach through the SChare platform via an NFT bridging experiment and a user study, demonstrating feasibility, scalability, and improved developer efficiency. Positive feedback highlights SChare's effectiveness in tackling high development costs, complex integrations, and skill barriers. These findings underscore the platform's potential to streamline BD and promote broader adoption.

Although the proposed uBSaaS framework and the SChare platform have demonstrated promising results, several open challenges remain. First, ensuring robust security and privacy across multiple blockchain networks is crucial, especially as cross-chain interactions increase the risk of vulnerabilities. Advanced cryptographic techniques or standardized security protocols could further safeguard transactions and data integrity in decentralized environments. Additionally, optimizing performance and scalability remains an ongoing endeavor. As the platform supports an expanding range of blockchain systems and use cases, efficiently managing resource allocation and latency-sensitive operations will be critical.

Future research may also explore extending the framework's capabilities in areas like on-chain governance, automated compliance checks for regulatory requirements, and compatibility with permissioned blockchains. Incorporating AI-driven techniques to automate or guide key decisions, for instance, selecting optimal blockchains for specific use cases, could further streamline development processes. Evaluating the framework with larger, more diverse developer groups or within industry-scale applications may yield deeper insights into how best to enhance adoption and interoperability. Ultimately, refining and broadening SChare's capabilities can help advance blockchain's promise of secure, transparent, and decentralized application development.

REFERENCES

Algorand (2024a). Algorand Developer Documentation: Algorand Standard Assets (ASAs). Accessed: 2024-11-06.

Algorand (2024b). Algorand developer documentation: Smart contracts. Accessed: 2024-11-06.

Antonopoulos, A. and Wood, G. (2018). *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly.

Antonucci, F., Figorilli, S., Costa, C., Pallottino, F., Raso, L., and Menesatti, P. (2019). A review on blockchain applications in the agri-food sector. *Journal of the Science of Food and Agriculture*, 99:6129–6138.

Bakshi, K. (2017). Microservices-based software architecture and approaches. In *2017 IEEE Aerospace Conference*, pages 1–8.

Bitregalo.com (2023). BitRegalo Whitepaper.

Bosu, A., Iqbal, A., Shahriyar, R., and Chakraborty, P. (2019). Understanding the motivations, challenges and needs of blockchain software developers: a survey. *Empirical Software Engineering*, 24(4):2636–2673.

Chakraborty, P., Shahriyar, R., Iqbal, A., and Bosu, A. (2018). Understanding the software development practices of blockchain projects: a survey. ESEM '18, New York, NY, USA. Association for Computing Machinery.

Chen, H. and Zhang, L. J. (2018). *FBaaS: Functional Blockchain as a Service*, volume 10974. Springer, Cham.

Chen, T., Li, X., Luo, X., and Zhang, X. (2017). Under-optimized smart contracts devour your money. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 442–446.

CSIRO's Data61 (2024). Factory contract.

Grandhi, J., Patil, M. U., and P R, L. E. (2023). Automation of blockchain network setup in offering blockchain as a service (baas). In *2023 Fifth International Conference on Blockchain Computing and Applications (BCCA)*, pages 635–642.

Gurzhii, A., Islam, N., and Marella, V. (2023). Understanding the challenges surrounding decentralized applications: An empirical study. In Janssen, M., Pinheiro, L., Matheus, R., Frankenberger, F., Dwivedi, Y. K., Pappas, I. O., and Mäntymäki, M., editors, *New Sustainable Horizons in Artificial Intelligence and Digital Solutions*, pages 277–293, Cham. Springer Nature Switzerland.

IBM Blockchain (2016). IBM blockchain.

Kaleido.io (2024). Blockchain, Digital Assets & Tokenization Radically Simple. Enterprise-Grade.

Khalid, S. and Brown, C. (2023). Software engineering approaches adopted by blockchain developers. In *2023 Tenth International Conference on Software Defined Systems (SDS)*, pages 1–6.

Knoche, H. and Hasselbring, W. (2018). Using microservices for legacy software modernization. *IEEE Software*, 35(3):44–49.

Ko, A. J., LaToza, T. D., and Burnett, M. M. (2015). A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering*, 20(1):110–141.

Krylovskiy, A., Jahn, M., and Patti, E. (2015). Designing a smart city internet of things platform with microservice architecture. In *2015 3rd International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 25–30.

Li, Y., Yin, H., Gai, K., Zhu, L., and Wang, Q. (2021). Blockchain-as-a-service powered knowledge graph construction. In Qiu, H., Zhang, C., Fei, Z., Qiu, M., and Kung, S.-Y., editors, *Knowledge Science, Engineering and Management*, pages 500–511, Cham. Springer International Publishing.

Lu, Q., Xu, X., Liu, Y., Weber, I., Zhu, L., and Zhang, W. (2019). ubaas: A unified blockchain as a service platform. *Future Generation Computer Systems*, 101:564–575.

Marchesi, L., Marchesi, M., Destefanis, G., Barabino, G., and Tigano, D. (2020). Design patterns for gas optimization in ethereum. In *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 9–15.

Marengo, A. and Pagano, A. (2023). Investigating the factors influencing the adoption of blockchain technology across different countries and industries: A systematic literature review. *Electronics*, 12(14):3006.

Microsoft (2017). Microsoft Azure blockchain solutions.

Microsoft (2018). Ethereum blockchain as a service on Azure.

Monrat, A. A., Schelén, O., and Andersson, K. (2019). A survey of blockchain from the perspectives of applications, challenges, and opportunities. *IEEE Access*, 7:117134–117151.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Pilkington, M. (2016). *Blockchain Technology: Principles and Applications*. Edward Elgar. Available at SSRN: https://ssrn.com/abstract=2662660.

Prewett, K., Prescott, G., and Phillips, K. (2020). Blockchain adoption is inevitable—barriers and risks remain. *Journal of Corporate Accounting & Finance*, 31:21–28.

R3 (2016). R3 Corda.

Seif.org (2023). SEIF - Smart contract for LegalTech.

Shadija, D., Rezai, M., and Hill, R. (2017). Towards an understanding of microservices. In *2017 23rd International Conference on Automation and Computing (ICAC)*, pages 1–6.

Sharma, T., Zhou, Z., Miller, A., and Wang, Y. (2022). Exploring security practices of smart contract developers. *arXiv preprint arXiv:2204.11193*.

Sharma, T., Zhou, Z., Miller, A., and Wang, Y. (2023). A mixed-methods study of security practices of smart contract developers. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, SEC '23, USA. USENIX Association.

Solana (2024). Solana developer documentation: Introduction for developers. Accessed: 2024-11-06.

Sun, J., Saddik, A. E., and Cai, W. (2024). Smart contract as a service: A paradigm of reusing smart contract in web3 ecosystem. *IEEE Consumer Electronics Magazine*, pages 1–9.

Upadhyay, N. (2020). Demystifying blockchain: A critical analysis of challenges, applications and opportunities. *International Journal of Information Management*, 54:102120.

Vacca, A., Di Sorbo, A., Visaggio, C. A., and Canfora, G. (2021). A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges. *Journal of Systems and Software*, 174:110891.

Virmani, N. and Singh, R. (2024). Analysis of barriers for adopting blockchain in agri-food supply chain management: a decision support framework. *International Journal of Quality & Reliability Management*, 41(8):2122–2145.

Wan, Z., Xia, X., Lo, D., Chen, J., Luo, X., and Yang, X. (2021). Smart contract security: A practitioners' perspective. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1410–1422.

Wessling, F., Ehmke, C., Meyer, O., and Gruhn, V. (2019). Towards blockchain tactics: Building hybrid decentralized software architectures. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 234–237.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32.

Wu, K. (2019). An empirical study of blockchain-based decentralized applications.

Yu, D., Jin, Y., Zhang, Y., and Zheng, X. (2018). A survey on security issues in services communication of microservices-enabled fog applications. *Concurrency and Computation: Practice and Experience*, page e4436.

Zou, W., Lo, D., Kochhar, P. S., Le, X.-B. D., Xia, X., Feng, Y., Chen, Z., and Xu, B. (2021). Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering*, 47(10):2084–2106.