big data and cognitive computing

MDPI

Article

SmartMinutes—A Blockchain-Based Framework for Automated, Reliable, and Transparent Meeting Minutes Management

Amir Salar Hajy Sheikhi [1], Elias Iosif [2,3], Oleg Basov [4] and Ioanna Dionysiou [1,*]

[1] Department of Computer Science, School of Sciences and Engineering, University of Nicosia, Nicosia 2417, Cyprus
[2] Department of Digital Innovation, School of Business, University of Nicosia, Nicosia 2417, Cyprus
[3] Institute for the Future, University of Nicosia, Nicosia 2414, Cyprus
[4] Faculty of Digital Transformations, ITMO University, Saint Petersburg 197101, Russia
* Correspondence: dionysiou.i@unic.ac.cy

Abstract: The aim of this research work was to investigate the applicability of smart contracts in the context of automating the process of managing meeting minutes. To this end, *smartMinutes*, a proof-of-concept prototype of automating meeting minutes was designed, implemented, and validated with test cases. The *smartMinutes* framework improves current practices related to the meeting minutes process by providing automation in areas where possible, and doing so in a transparent, flexible, reliable, and tamper-proof manner. The last feature is of paramount importance due to the fact that meeting minutes offer legal protection, as they are considered official records of the actions taken by an organisation. Additionally, *smartMinutes* supports meeting agendas with non-voting items as well as voting items, offering a pool of three voting schemes, executing under three different configurations. A particular configuration, the hidden mode, provides for secrecy while at the same time guaranteeing transparency.

Keywords: smart contracts; Ethereum; meeting minutes; blockchain; voting; legal protection

Citation: Sheikhi, A.S.H.; Iosif, E.; Basov, O.; Dionysiou, I. *SmartMinutes*—A Blockchain-Based Framework for Automated, Reliable, and Transparent Meeting Minutes Management. *Big Data Cogn. Comput.* **2022**, *6*, 133. https://doi.org/10.3390/bdcc6040133

Academic Editor: Michele Melchiori

Received: 6 October 2022
Accepted: 8 November 2022
Published: 10 November 2022

1. Introduction

Meeting minutes are, in their very essence, records of everything that was discussed and decided during a particular meeting [1]. They play an important role within the professional world as they are proof that a meeting did indeed take place and, along with it, the decisions that were made. Meeting minutes can be used and referred to in future meetings to identify progress of any previously discussed topics. This could enhance meeting productivity, as a documented starting point would be already available. Additionally, if one is absent from a particular meeting, the meeting minutes can be retrieved to find out details on the topics discussed. Meeting minutes could also be used in case of conflicts related to any past agreements on decisions. Last but not least, meeting minutes offer legal protection, as they are considered official records of the actions of a particular committee or board. Therefore, it is of paramount importance to safeguard and verify their authenticity.

Voting or polling to determine the stance of meeting attendees on a matter is standard practice. Items on the meeting agenda may require voting by members, not necessarily done during the meeting time. The post-meeting voting could introduce challenges to the smooth process of minute management, as it will be explained in a later section. Conducting this process in a non-automated manner may prove to be nontrivial, especially if it is done online using, for example, email: members may forget to submit their vote by the agreed deadline or may claim they cast a vote when they did not. Specialised online voting systems do exist; however, to the best of our knowledge, there is no single platform that integrates online voting with an automated minutes management system.

This paper introduces *smartMinutes*, an approach that aims to automate the management of agenda minutes in a transparent, flexible, non-reputable and unforgettable manner. It provides solutions to existing shortcomings of current practices within this field, as well as supports automation to fast-track the process of recording and voting on agenda minute items. Furthermore, *smartMinutes* is a proof-of-concept prototype that uses blockchain technology along with smart contracts to manage several aspects of the meeting minutes process, allowing participants to vote on agenda items not only during the meeting but also in a post-meeting manner. Voting items are configured separately, specifying their voting deadlines, as well as the voting scheme to be deployed for each item. Thus, the paper contributions are twofold:

- Present the *smartMinutes* framework that aims to improve the overall experience involved with the management of meeting minutes, as well as provide an integrated voting platform. This unique combination ensures that committee members do not resort to a secondary platform for their agenda-related voting needs.
- Discuss the financial feasibility of the proposed approach by analysing the costs involved during the execution of the embedded *smartMinutes* smart contracts. Additionally, the time complexity of the framework is estimated to further assess the overall performance of the proposed system. As a matter of fact, these two key metrics are utilised to determine the efficiency of the smart contract code, identifying its strengths and weaknesses.

The rest of the paper is organised as follows: Section 2 presents challenges related to the meeting minutes process, suggests the future direction towards an automated approach, and presents existing approaches to computerised meeting minutes. Section 3 introduces the design principles and inner workings of the *smartMinutes* framework, including the pool of voting schemes and voting configurations. The overall framework performance in terms of time complexity is analysed in Section 4. The experimental test cases, along with the financial cost of the smart contract utilisation are described in Section 5. Possible alternate uses of the *smartMinutes* framework in other fields, given its modular design, are identified in Section 6. Section 7 concludes.

2. Review of Literature

2.1. Existing Approaches on Meeting Minutes Systems

Meeting minute related systems typically share the same goal of improving the overall experience when it comes to the process of creating accurate official records of meetings; the focus is on the contents discussed and the outcome of the discussion.

An investigation into the published content related to the meeting minutes domain revealed that the main objective of the existing systems is to provide some level of automation, which would then streamline a part of the meeting minute lifecycle. With an effective solution, this would mean that less manual work would be done, and then it could be argued that there would be a lower chance for human error in areas that are automated. The overall accuracy of the meeting minutes records would increase, given that the automation is performed correctly. Below, details on several existing systems are given.

Starting with the system in [2], the authors presented an approach using natural language processing (NLP) techniques to automate parts of the meeting minute process. The presented solution is split into three phases, namely, speech recognition, speaker recognition, and summarisation. The field of speech recognition was developed by big technology-oriented companies, such as Google, and API access is granted to the public. The speaker recognition phase involves the system to be able to use speaker diarisation to identify the key person who is giving the speech based on a particular phrase. The summarisation phase involves the system making use of networks that utilise bi-directional long short-term memory (bi-LSTM) to comprehend context based on previous and following phrases. Based on the analysis, it was found that abstractive summarisation yielded better results when compared to extractive summarisation. In addition, it was found that

speaker diarisation is more suitable for their system when compared to other methods, such as naive logic. It was also found that speaker diarisation increases load and complexity.

In [3], the authors proposed a system that summarises meeting minutes given an input, which is aimed to be used in the Indonesian parliament. This system would make use of rule-based data extraction in conjunction with regular expressions. ROUGE, or recall-oriented understudy for gisting evaluation [4], is a metric used to identify the quality of auto-generated summaries. This method of evaluation identifies the degree of overlapping content between the system and the human-made references. More specifically, ROUGE-1 indicates every word overlapping, whereas ROUGE-2 indicates overlapping bigrams. The authors concluded that summary output produced by this system has an average of ROUGE-2 alongside a recall of 0.703 and ROUGE-1 with a recall value of 0.803. Additionally, it is suggested that possible future improvements could potentially reduce errors, as well as improve the overall performance of the summarisation module.

Yanji [5] is proposed as a mobile solution to automating meeting minutes. The authors outline the importance of having a portable automated solution in an age of intelligent mobile devices. The proposed system is Android-based and can record speeches, generate text, identify speakers, and generate meeting minutes. Yanji makes use of cloud services, such as IBM Cloud, in real time and has the capability of reducing storage sizes of recordings. The authors conclude with exploring future improvements to their system, which include optimised data retrieval, expanding the system to be compatible with online meetings, as well as the ability to synchronise with participants.

A system is proposed in [6], which uses meeting data to produce web pages that outline the sequence of the meeting. The meeting data that are automatically collected can consist of text, video, and audio. The proposed system could retrieve key points of the meeting. Each participant's meeting data are individually recorded and uploaded to a server if a connection is available, otherwise it is saved locally on their device. The authors outline a search module for this system, which allows users to look for a specific instance of a phrase or word. The main goals are to allow users to look through important points after the meeting and to provide an interface to search for key terms or phrases used within the meeting. The authors conclude with suggesting future improvements to their system, such as a "Slide Synchronisation Tool", which would allow users to share presentations using just their web browsers.

The automatic generation of meeting minutes in a parliamentary setting in given in [7]. The authors of the paper state that the presented system makes use of rhetorical structure modelling (RST) [8], which is a process that is conducted on given text that then can be used for analysis. Through an effective implementation of this model, the authors aim to transform audio-based recordings of parliament speeches into automatically generated summaries of meeting minutes. Additionally, they propose a one-step system for chunking, parsing, and extracting summarised texts. The paper concludes by stating the results of the experiments that were conducted on the proposed system. It was found that both the acoustic and N-gram features of the system ranked similarly, where the former was found to be 66.7% ROUGE-L F-measure and the latter 68% ROUGE-L F-measure. For reference, ROUGLE-L [4] is essentially the largest common sequence of words that is found between the system output and the relative reference.

An online multimedia meeting minutes system called LiteMinutes [9] is proposed, which has the necessary capabilities for creating, reviewing, and distributing meeting minutes using the internet. The authors state that the system also supports email-based interaction, as well as media types such as text, images, and video recordings. Once a meeting is concluded, any textual notes taken on the specialised application are uploaded from the user's computer to the proposed system where they are formatted in HTML. On top of that, the system automatically links text items to corresponding images (of slides), as well as any relevant video recordings. The authors state that future improvements of LiteMinutes include support of more complex interactions with support of multiple users taking notes.

Continuing with the ProMETheus system [10], it is designed as a mobile solution to automatically generating meeting minutes from audio recordings. The speech recognition module of the system is capable of transcribing audio into text data. Additionally, a speaker recognition algorithm is used to equip the system with the capability of identifying different speakers. The authors state that the system can identify key points from the meeting in its converted text form using a text summarisation algorithm. The authors performed experimentations in order to identify the best speaker recognition and text summarisation algorithms. It was found that the unsupervised speaker recognition algorithm performed the best. As for the text summarisation, all algorithms were evaluated and the one with the highest F-score is used. The authors state that the proposed system was tested in an academic setting. The authors state that ProMETheus can save a significant amount of time by automating the meeting minute generation process, however, some limitations it may have include an inability to function accurately in cases of overlapping conversation.

A meeting minutes summarisation system, which utilises a novel approach based on a two-step sentence extraction methodology, is described in [11]. The authors explain that the first step involves the system first identifying sentences that are connected to the main topic of discussion in a meeting. Afterwards, the second step consists of evaluating extracted sentences from the first step and continuing with further extraction of content related to the main topic. The result of the two-step extraction process is a tree structure made up of the relevant extracted sentences that link to the main topic. After some experimentation, the authors conclude that their methodology is plausible and that the proposed system demonstrates increased performance. Lastly, they propose a new tree generation methodology, as they describe their current implementation as not robust in terms of an increasing summarisation ratio.

A system called the minutes retrieval system, or MRS, is proposed in [12]. The authors emphasise their aim with this system is to implement a flexible data retrieval system that is to be used in the context of meeting minutes and to be able to identify links or relationships between meeting participants and their interests. The system takes meeting minutes in a custom XML format as input and makes use of latent semantic analysis (LSA). LSA [13] is used to perform analysis on a given input in order to determine relationships between terms. As it is presented in the paper, MRS treats each set of meeting minutes independently. The authors suggest implementing a mechanism that possesses the capability of analysing meeting minutes spanning from different sets. This can then outline the relationship between the subjects across different sets of minute data.

Another system that can automatically recognise conversations in a meeting setting and produce a summary is proposed in [14]. In addition to this functionality, the authors state that a browser interface is implemented, which allows users access to the transcribed highlights of the meeting minutes, as well as the ability to search for specific key points. Some experimentation is conducted in order to determine the best speech recognition system for the proposed system. On top of that, it was found that using already existing acoustic models for this use case may be a viable strategy. During testing of the system, the authors came across a degree of recognition errors due to the segmentation methodology used. It is stated that by improving the segmentation methodology, the degree of recognition errors will improve as well. The authors state that other possible future works may include improving the system to the point where minimal supervision would be needed, as well as an expansion of the system to incorporate other meeting-related tracking needs, and the ability for the system to combine different data sources, such as improved speech recognition and speaker recognition using face identification.

A sliding window methodology in developing a system that can automatically generate meeting minutes is given in [15]. The authors state that, one of the main goals for this system is to take on issues with long transcripts of meetings. The presented system utilises the sliding window approach in combination with a neural abstractive summariser to identify key points from a given meeting transcript. The testing process of the system involves comparing two automatically generated transcripts with a human generated one

and reviewing the differences in the written key points. After testing, the authors conclude that their novel approach shows promise and that it has the potential to be utilised in different domains. The authors state that potential future work on this system may involve developing a look-ahead mechanism, which would allow the system to skip to more relevant sections of the transcript.

The last system presented is the one in [16], which addresses the issue of identifying and extracting relationships within meeting minutes that are generated by a speech recognition system. The approach taken by the authors involves using collective entity resolution (CER). CER [17] is an algorithm or technique that can be used to improve resolution accuracy by determining references of entities as a joint rather than an independent occurrence. As seen in other works, speech recognition systems can be used to generate transcripts of meetings in the form of unmanaged minutes. The authors of the paper make the argument that if the generation of minutes using speech recognition software is not managed, then there would be substantial errors in the output transcript and thus the retrieval of key topics would be more of a challenge. In this regard, the paper proposes a methodology that makes use of minute relationships to aid in the process of information retrieval at a later stage. The developed methodology involves using stop word elimination, as well as similarity calculations conducted on CER-generated clusters. After experimentation, the authors conclude that their technique proves to have better performance when compared to standard techniques.

Several similarities can be identified among the systems discussed above. First, many of the featured works have the main goal of automating the generation of meeting minutes. This goal seems to be motivated by the idea of reducing manual work needed in this domain, and in some cases, fully automating the process by implementing a system that can record audio of the meeting, generate transcripts in text format, and then finally generate the meeting minutes. Second, a few systems aim to generate a summary of key points as deduced by the meeting minutes. Third, many of the featured systems also could search for specific terms or even phrases found in the summary. The main goal for these types of systems appears to make the information retrieval process related with meeting minutes easier.

With these works identified and evaluated, it is evident that currently, there are no blockchain-based solutions to the management of meeting minutes. More importantly, there does not seem to be any comprehensive solution that meets all the needs of such a system. A complete solution may mean that users will not have to rely on multiple platforms for meeting management and voting, which could then simplify the process as a whole.

2.2. Meeting Minutes Management: Challenges and Future Direction

The process of managing meeting minutes spans the entire meeting lifecycle, where the latter usually comprises of three stages: pre-meeting, meeting, and post-meeting. Figure 1 illustrates these phases.

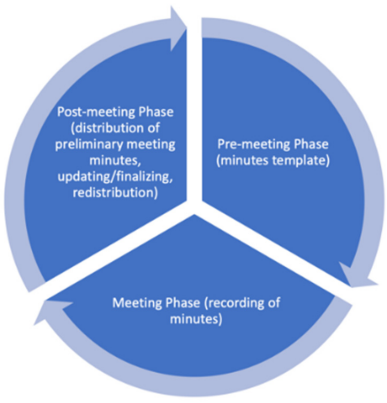

Figure 1. Meeting minutes lifecycle.

During the pre-meeting phase, it is important to become familiar with the way the organisation records its agenda minutes [18]. This can be achieved by reviewing the records of past meetings and how they were recorded, which then, in turn, can be used as a template for future meetings. The next stage, the meeting phase, involves recording the minutes. If sufficient preparation is done, then an effective template should be used that matches the style of the organisation, making the process of recording the meeting easier. During the post-meeting phase, adjustments, or finalisations, to meeting minutes are completed. This usually takes place once the meeting concludes. Final adjustments may include expanding on agenda items that may be vague or making clarifications about some items. During this phase, the finalised meeting minutes are distributed and become official by having them signed by the meeting participants. Protocols set by the organisation must be followed to store the minutes. This also includes making backups.

The various tasks described above can be done either manually, by email, or by using a web-based platform. There is limited automation that can be offered using any of the three options above and safeguarding the meeting minutes authenticity is also nontrivial. In addition, challenges regarding meeting and/or post-meeting voting are difficult to address in an effective and time-efficient manner: meeting participants can forget to digitally send votes, different deadlines for different items can complicate things further, and participants can deny having voted/not voted.

It is evident that there is an opportunity for designing and implementing a better solution to meeting minutes management, one that alleviates the shortcomings of current solutions. As meeting minutes are considered legal documents, the blockchain technology can be considered as the backbone of a proposed system, as it offers transparency while safeguarding against any modifications or attempts to tamper with the authenticity of the stored data. Many sectors already began implementing integrations, or even full blockchain-oriented solutions, to create improved solutions where existing solutions fall short [19]. The proposed system could aim to streamline the steps involved with managing meeting minutes by providing users the ability to add meeting items, vote on existing items, and make the items instantly available to all committee members. The direction that needs to be explored is implementing a blockchain-oriented meeting minutes management system with focus on integrating a voting system that allows attendees to cast their vote on agenda items. An important requirement is to provide the users with the option of anonymity whilst maintain data integrity. A user may want his/her vote to remain hidden, while at the same time participating in the voting process. The aspect of maintaining data integrity is important for a system with a voting aspect to prevent security issues, such vote tampering, a form of result manipulation [20].

3. *SmartMinutes*: Design Principles and Implementation Details

3.1. Design Principles

None of the systems in Section 2.1 support a blockchain-based solution for meeting minutes management. Furthermore, none provide a comprehensive platform that provides for all meeting minutes management tasks while integrating a voting module. To address the shortcomings of the existing practices of managing meeting minutes, a novel approach *smartMinutes* is presented that streamlines several meeting minutes tasks and bundles them into a reliable system that is guaranteed to be transparent and tamper-proof. The *smartMinutes* framework supports the specification of meeting minutes items as voting or non-voting, allows members to know when a vote is cast, simplifies the management of different deadlines of voting items, and creates an immutable record of every vote. These features allow *smartMinutes* to bypass the major shortcomings of current approaches.

Both blockchain technology and smart contracts are used by *smartMinutes* to offer the above guarantees. In the first case, the inherently immutable nature of blockchain provides assurances regarding the integrity, transparency, and nonrepudiation of the items themselves. In the latter case, the automation and configuration of voting items are accommodated and integrated in the blockchain framework. One may claim that smart

contracts are still in their infancy stage [21] and their use cases are still being discovered and theorised. However, they hold several advantages over traditional contracts, such as reduction in risks, limited costs, and improved efficiency [22]. Additionally, they allow for user-defined conditions or specifications, which are then programmed into the smart contract. In the case of *smartMinutes*, organisation-defined protocols or procedures involved in the meeting minutes finalisation can be directly fed into the smart contract which, in turn, enables some level of automation.

SmartMinutes is proof-of-concept prototype designed and developed to automate the procedures involved in approving meeting minutes in the context of departmental meetings in the university setting. The Solidity programming language was used to create the smart contracts [23] that run on Ethereum. Several assumptions were made during the design phase, and these are stated below:

- Applying blockchain technology to facilitate a comprehensive meeting minutes management system was investigated before. Thus, there is no recommendation, based on existing approaches, regarding the type of blockchain to be utilised (public or private). Public blockchains are decentralised and permissionless, as well as accessible by everyone, whereas access in private blockchains must be explicitly granted. The restricted-access characteristic of private blockchains requires centralised management, unlike the public ones. Since the proposed system is a prototype, it was decided to use a public blockchain (Ethereum) as the backbone technology for the platform. This is a suitable choice for application domains, such as e-government, where transparency is a vital democracy element. However, private organisations may opt to deploy a private blockchain, as minutes are not expected to be accessed by the general public. The choice of blockchain type does not affect the overall design of *smartMinutes.*
- The approval process of the agenda minutes is assumed to follow a 3-tier hierarchy (see Figure 2). Each tier in this model represents an academic body involved in the approval of academic minutes. The low level is the department council (where the meeting took place), the mid-level represents the school council, and the top tier stands for the senate. Without loss of generality, the prototype utilises only a 2-level hierarchy.
- The minutes themselves consist of decisions and voting items, with possibly different voting deadlines and voting schemes. It is assumed that there will be two types of agenda items: voting items and non-voting items. Voting items are agenda items that require voting by the committee members and non-voting items are agenda items that do not need a vote.
- A pool of various voting schemes or methodologies should be available, as voting items may have different voting scheme requirements. Majority voting, speedy mode, and veto mode are supported, as shown in Table 1.
- A set of three different voting modes of execution is supported.
    - Live execution mode allows the participants to view the actual votes as they are casted. This is the default setting.
    - Open execution mode reveals the cast votes only after the voting is completed.
    - Hidden execution mode prevents the disclosure of the actual votes during or after the voting completes.

Table 1. Supported voting schemes.

Voting Scheme	Description
Majority voting	Requires all members to vote
Speedy mode	Does not require all members to vote
Veto mode	Ability to veto the item which ends the voting process

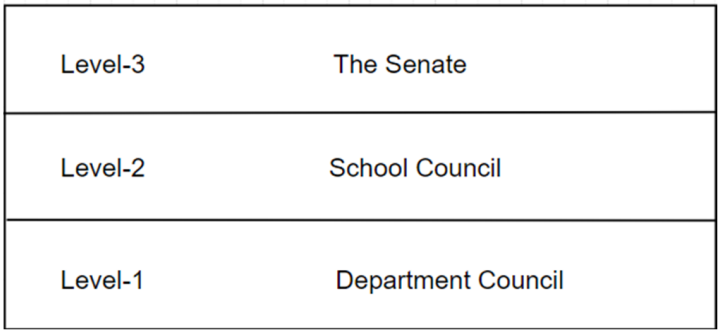

Figure 2. 3-Tier hierarchy.

It is important to emphasise an important security feature integrated in the framework, which is that of access control. The smart contract is configurable and interactable only by the committee members at their respective hierarchical level. For example, a committee member from Level-1 voting cannot interact with the smart contract in Level-2 and vice versa. This is so in order to provide voting integrity and prevent tampering with the votes.

The three voting schemes, as described in Table 1, provide users with a level of flexibility. For example, if an item is bound to an important decision, then, all users may be required to participate in the voting process. This can be enforced by using the majority voting scheme. Another scenario could involve the committee members having a busy schedule, thus they could utilise the speedy mode voting scheme, which aims to conclude the voting process as soon as enough votes are cast. Additionally, the option to reject a particular voting item is also provided in the form of the veto mode scheme. This reserves the user's right to immediately conclude the voting process for an item.

3.2. System Design Details

In this section, design diagrams depicting the inner workings of the *smartMinutes* framework are presented. To be more specific, the core operations of *smartMinutes* and the voting module in greater detail are outlined.

The general operation of *smartMinutes* can be summarised with the following components: the minute management module, voting module, item deadline module, and lastly, the user registration module. The meeting minutes module is responsible for handling user inputs (agenda items) and storing them correctly for later use. Committee members can enter agenda items into the system. Afterwards, this module will store the input item in memory to be used at a later stage. The voting module requires two core conditions to continue its operations. Firstly, the agenda item in question needs to already be input into the system and marked to require voting. Secondly, the vote would need to take place before the deadline specified for the item. The voting module performs these two critical checks before proceeding with the voting procedure. The voting procedure is further described later in this section. The item deadline module expects user input in the form of Unix time and assigns this deadline to the specified agenda item. The final core component, user registration module, handles registering the user as committee members. The user must specify the hierarchical level in which they want to be assigned. This module then whitelists the user's wallet address, thus promoting them to a committee member. These operations can be seen in Figure 3.

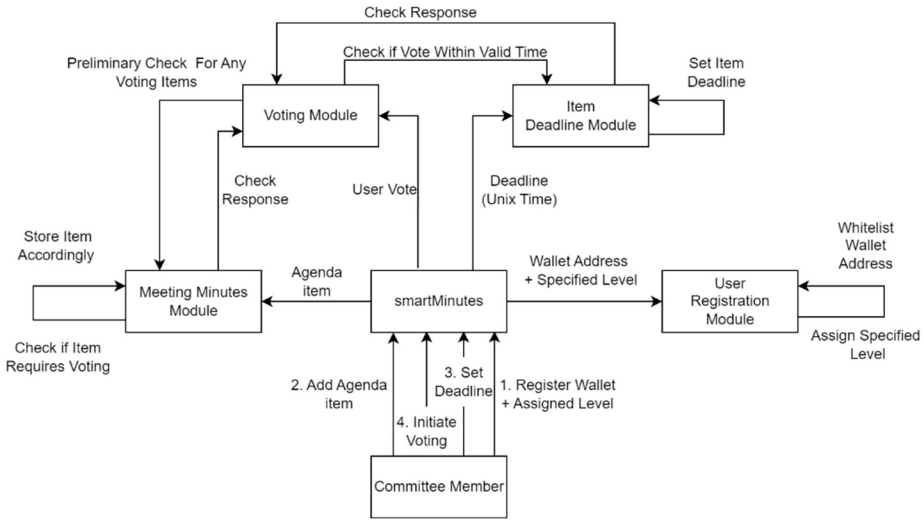

Figure 3. SmartMinutes operations diagram.

The voting module handles all aspects of the voting process. Once this process begins, *smartMinutes* starts to accept votes from committee members. When a vote is cast, the voting module performs a check to determine whether the minimum number of votes required for the selected voting scheme was reached. It does this by invoking the vote determination module. This module first checks the selected voting scheme by invoking a call to the vote scheme handler and then determines whether the voting process is concluded. The result of the vote is then passed onto the voting module, where it can be accessed by the committee members. The described process is depicted in Figure 4.

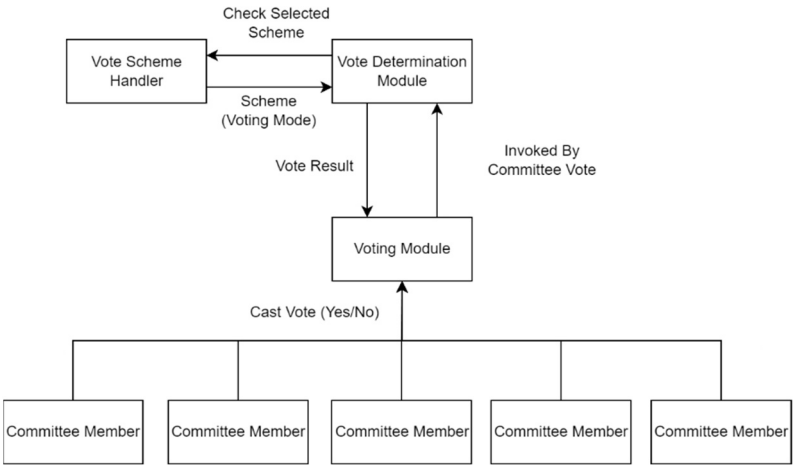

Figure 4. Voting module diagram.

3.3. Implementation Details

The prototype was developed using the Ethereum Remix IDE. Remix offers a graphical user interface (GUI) for deploying, testing, and interacting with the contract. For ease of development, the Visual Studio Code (VS Code) text editor was used alongside the Remix IDE extension. This approach enables the full use of vs. Code's arsenal of useful extensions for development, whilst making use of the compilation, deployment, and testing features of Remix IDE. For testing the smart contract locally, *Ganache* was used. This tool acts as an emulator, which simulates the Ethereum blockchain for development purposes. A major benefit of using Ganache over other similar tools is the user-friendly interface that allows for block inspection. This tool provides the user with the necessary resources to test smart contracts locally.

Note that some *experimental* features were enabled. Given that *smartMinutes* is a proof-of-concept prototype, these features allow for easier string manipulation, as these features are not readily available with the compiler version. The experimental features are the only dependency of the smart contract.

As outlined in Figure 5, the available functions and accessible variables are: *addItemNeedsVoting*, *addItemNoVoting*, *level1*, *level2*, *majority*, *proposeDocument*, *setDeadline*, *speedy*, *veto*, *vote*, *addressCount*, *allHaveVoted*, *documentResult*, *getActualVote*, *getItemResults*, *getItems*, *getWalletArrayDEBUG*, *level*, *proposedDocument*, *voteCount*, and *votingDeadline*. Details on these functions and variables are given below.

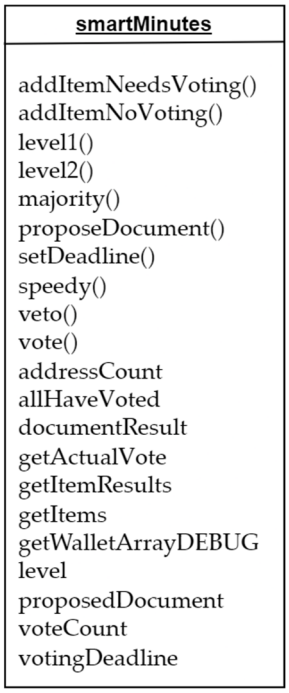

Figure 5. SmartMinutes operations.

The *addItemNeedsVoting* function expects an input from the user. This function will take the user input and store it locally. The program accepts any string, but the intended use is to add agenda items that require voting. Additionally, this function will signal to the program that the added item is qualified for the voting process.

The following *addItemNoVoting* function is similar to the previous function, however, it is intended to be used for agenda items that do not require voting. The function expects the user to enter an input that would be the agenda item, however, unlike the *addItemNeedsVoting* function, does not signal the program for voting.

The following two functions are *level1* and *level2*, and as their names suggest, they will change the hierarchical level of the contract. For example, if it is in level-1 mode, the *level2* function will put the contract into level-2 mode and vice versa. This is of course, assuming that the program is in a "safe" state to change levels. This means that all items that require voting must be voted on before the level change and that that voting is not currently in progress.

The next function is called *majority*, and when it is executed, it will switch the voting configuration to majority voting. By default, the program is set to this voting method.

The next function is *proposeDocument* and it expects an input from the user. This is used for specifying the document or item that the committee should vote on. This function stores the name of the document, which is to be voted on and also triggers the voting functionality of the contract.

The *setDeadline* function expects the user to specify the deadline for the document or item that is to be voted on. The format the program expects the input deadline to be is in

Unix time (Epoch). For example, the long integer *1651932954* represents the time and date of *17:15 on Saturday 7th of May 2022*.

The following function, called *speedy*, executes a function that sets the voting methodology to speedy mode.

Afterwards, the next function is called *veto* and is the last voting configuration of the specified voting schemes. This function sets the voting methodology to veto mode.

The next function is also the final user-interactable function. This function is called *veto* and expects a user input, which is in this case the user's actual vote. The program expects 1 for yes, 0 for no, and -1 for a veto vote if applicable. Note that the option to abstain is not considered, however, can be easily implemented at a later date. Upon execution, the function records the user's vote.

The remaining functionality available to the user has to do with being able to check the value(s) of public variables. To be more specific, *addressCount* will reveal the number of committee wallet addresses registered for the current hierarchical level. Then, *allHaveVoted* will reveal whether all registered committee members have cast a vote. Afterwards, *documentResult* will reveal whether the item passed voting, as in it was determined by the committee members. Next, *getActualVote*, given a wallet address, will return a committee member's actual vote if they voted. Then, *getItems* will reveal a list of agenda items input by committee members where the list specifies whether an item requires voting. Afterwards, *getWalletArrayDEBUG* will reveal the list of registered committee member's wallet addresses, which is of course only for proof-of-concept demonstration purposes only. Then, *level* will reveal the current hierarchical level the contract is in. Next, *voteCount* will show how many committee members casted a vote for the current item. Finally, *votingDeadline* will reveal the voting deadline that was input to the contract in Unix time format.

3.4. SmartMinutes Graphical User Interface

Figure 6 illustrates what the presented smart contract solution, *smartMinutes*, will present to the user upon deployment. As shown, 23 interactable buttons appear. These buttons are split into two different categories. The first category of buttons includes ones with an orange background and the second category includes buttons with a blue background. The orange ones signify that upon triggering the button (by clicking it), a function will execute. The blue buttons represent variables within the contract and triggering them will simply display the variable's current value.

Either orange or blue buttons with a field next to them signify that the function expects an input from the user in order to execute the function or to reveal the current value of the variable.

Through the interaction with the buttons, the user interacts with the smart contract. Each button is labelled with the name of the function it executes or variable it shows. For example, the blue button labelled "level" will display the hierarchical level (level-1, level-2) that the contract is currently set to. Note that for purposes of demonstrating this proof-of-concept project, some critical variables were left to be accessible by the user. In actual live deployment, these variables should not be visible as their free access is a cybersecurity concern.

When the user is required to submit an input, they can utilise the default input GUI provided by Remix. This interface even allows the user to see what type of input the smart contract is expecting. As shown in Figure 7, the selected function requires a user to input a string.

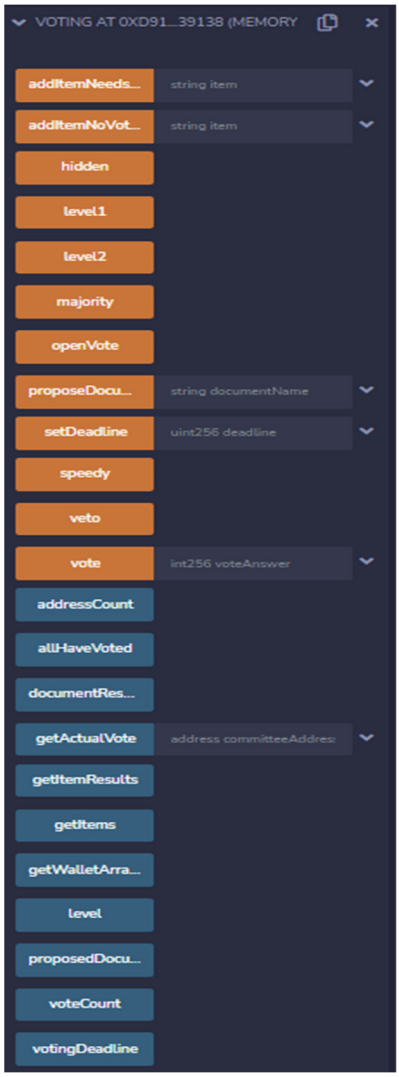

Figure 6. SmartMinutes initial deployment output GUI.

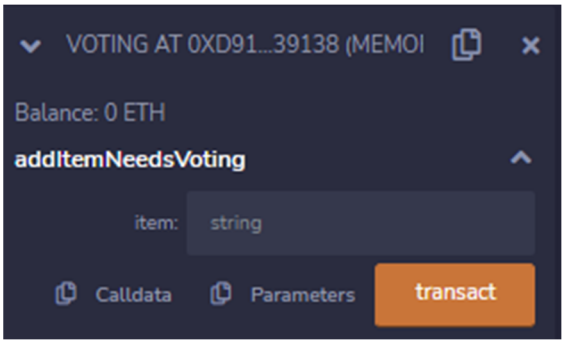

Figure 7. User input field GUI.

4. *SmartMinutes* Time Complexity Performance

The importance of determining the time complexity of a program comes from the fact that there are many different types of devices with their own hardware architecture. On top of that, devices may run different operating systems, which can be a major factor when it comes to determining the overall runtime of executed code. These factors can affect the speed in which the device's processor can operate, thus affecting the overall runtime of programs. The determination of time complexity in any program is important

in understanding its efficiency in terms of both execution time and resources, regardless of the hardware and operating system of the device.

Table 2 showcases the overall runtimes of each function used within the *smartMinutes* contract in terms of BigO notation. Please note that these functions were already described in Section 3.

Table 2. *SmartMinutes* BigO of functions.

Function Name	BigO	Resultant T
addItemNoVoting	O(1)	O(1)
addItemNeedsVoting	O(1)	O(1)
getItems	O(1)	O(1)
getItemResults	O(1)	O(1)
getActualVote	O(1)	O(1)
setDeadline	O(1)	O(1)
containsAddress	O(1)	O(1)
committeeHasVoted	O(1)	O(1)
actualCommitteeVote	O(1)	O(1)
setAddress	O(1)	O(1)
initialise	O(1)	O(1)
reset	O(2N)	O(N)
l1Reset	O(4N)	O(N)
l2Reset	O(4N)	O(N)
proposeDocument	O(2N)	O(N)
majority	O(1)	O(1)
speedy	O(1)	O(1)
veto	O(1)	O(1)
checkAllVoted	O(N)	O(N)
determineDocument	O(N)	O(N)
majorityVoting	O(N)	O(N)
speedyVoting	O(N)	O(N)
vote	O(N)	O(N)

The *smartMinutes* contract uses arrays of committee wallet addresses at its core. This keeps track of the committee members and their interactions with the smart contract. The need to loop through these addresses is kept at a minimum.

The *addItemNoVoting* function is O(1), as the only action it will take is to simply push a given user input to an array. Therefore, this function has constant complexity. Similarly, following function *addItemNeedsVoting* is O(1), as it performs a similar operation with the addition of string concatenation. This function also has constant time.

The next function *getItems* is O(1). This function does not have a dependency on an input size, as its purpose is to return the contents of an array without having to access each item individually. With this in mind, this getter function has constant time complexity. The *getItemResults* function performs the same operation with another array, which stores data about the results of the voting process of an item. This getter function is O(1) and has constant time complexity.

The *getActualVote* function is O(1). This function has constant time complexity, as it returns the value of vote cast by a particular committee member. This is accomplished by directly accessing a particular array index and returning its contents. The following function *setDeadline* stores user input for later use. Since this function is O(1), it has constant time complexity.

The next function *containsAddress* is O(1). It accesses a particular key of a mapping (dictionary) and returns its paired value. This function has constant time complexity. The *committeeHasVoted* function performs a similar operation on a different mapping. It is O(1) which means it has constant time complexity. The *actualCommitteeVote* function is a helper that retrieves the vote value of a committee member. It accesses a particular

key in a mapping and returns the value pair. This function is O(1), which is constant time complexity.

The following function *setAddress* is O(1) and uses a helper function to initialise the wallet address of a new committee member. This is done by appending the new address to the relevant arrays/mappings. This function has constant time complexity. The helper function is called *initialise* and is set as a private function. The *setAddress* function is set as public and uses modifiers to control user access.

The *reset* function is O(2N). Its purpose is to perform a reset and re-initialise all relevant committee wallet data. Since the function has to individually reset the state of the wallets, it is dependent on the number of committee members (N). This reset is performed for both the presence of the committee member as well as re-initialising the mapping used to keep track of their votes. With this in mind, this function has linear time complexity.

The following functions *l1Reset* and *l2Reset* are both O(4N). Their main purpose is to switch the contract's hierarchical level. This is accomplished by resetting the committee wallet address-related arrays/mappings. It re-initialises variables to adjust the presence of committee members based on the particular level the contract is switching to, as well as makes use of the *reset* function. These two functions have linear time complexity.

The *proposeDocument* function is O(2N). This function keeps track of the document or item that is currently being voted on. Additionally, upon submitting a new item to be voted on, this function uses the *reset* function to prepare the contract for the voting process. With this in mind, the function has linear time complexity.

The following functions *majority*, *speedy*, and *veto* set the contract's voting mode to their respective modes. Each of the three of these functions are O(1), meaning they have constant time complexity.

The next function *checkAllVoted* is O(N). This function has to perform a check on a mapping, which contains whether the committee member voted. Since the function is dependent on the number of wallet addresses, it has linear time complexity. The *determine-Document* function performs a check on an array, which holds all committee member votes to determine the voting outcome. This function is O(N) and has linear time complexity, as it is dependent on the number of committee wallet addresses.

The *majorityVoting* function is O(N) and makes use of the *determineDocument* function. This function first performs a check to see if all members voted. It has linear time complexity. The next function *speedyVoting* operates in a similar manner, however, this function does not perform a check as per the speedy voting scheme. It still makes use of the *determineDocument* function, therefore it is O(N), and therefore linear time complexity.

Lastly, the *vote* function is responsible for registering the user's vote. It makes use of *majorityViting* or *speedyVoting* functions depending on the selected voting scheme. With this in mind, this function is O(N), which means it has linear time complexity.

It can be determined that the overall time complexity of *smartMinutes* is O(N)—linear time. The main factor, which would linearly increase runtime, would be the number of committee members present. Furthermore, *smartMinutes* was designed to accommodate five committee members per level for up to two hierarchical levels. For the support of up to 10 committee members, the determined overall runtime is acceptable. If the system is to be expanded to fit a much larger committee, then further optimisations to the code of *smartMinutes* should be made to reduce the overall runtime complexity.

5. Experimental Findings

The objective of the experimentation is to determine overall costs in terms of fees [24]. Two use case scenarios were run on *smartMinutes*, and the findings are described below. More details on the testing findings can be found in [25]. Measuring the total costs involved with the usage and operation of *smartMinutes* is crucial in determining its feasibility to be deployed as a valid solution. Additionally, viewing the total gas costs as a key metric can help determine the smart contract's cost-effectiveness, which can then be compared to other available meeting minute management solutions.

The aim of use case Scenario 1 is to investigate the overall gas costs for *smartMinutes* functions and to determine if its deployment has a cost. Use case Scenario 2 is designed to determine the Ethereum gas cost-effectiveness of the speedy mode voting configuration compared to majority voting and veto mode. This experiment involves recording the gas costs of the executed functions using speedy mode, majority voting, and veto mode voting configurations and then comparing the overall costs.

The general methodology, which was used for conducting these experiments, is as follows: First, the combination of voting scheme and simulated voting scenario is identified. Then, the locally simulated wallet(s) are reset to be able to keep track of transactions and their fees for a particular instance. Next, *smartMinutes* is configured to the desired voting scheme. Last, the committee votes are simulated, and their respective gas costs are recorded. These experiments were conducted on a Windows 10 machine using an 8-core Ryzen 7 2700x processor clocked at 4.0 GHz.

5.1. Use Case Scenario 1

Consider an agenda that contains six items. This agenda is then input into *smartMinutes*, and the gas costs are tracked and recorded per item. Table 3 illustrates the agenda item configuration along with the gas costs.

Table 3. Use case Scenario 1 total gas cost per item.

Item	Requires Voting	Total Gas Cost (ETH)
Item #1	Yes	7.8390×10^{-13}
Item #2	No	5.0445×10^{-14}
Item #3	Yes	7.7219×10^{-13}
Item #4	No	9.5108×10^{-14}
Item #5	No	9.5132×10^{-14}
Item #6	No	5.0553×10^{-14}

For this particular simulated scenario, out of the six total items, two of them (items #1 and #3) required the use of the voting functionality of *smartMinutes*, whereas the remaining four items were simply input in order to keep track of them. It was calculated that for item #1 the gas cost was about $\mathbf{7.8390 \times 10^{-13}}$ **ETH** and for item #3, it was calculated at $\mathbf{7.7219 \times 10^{-13}}$ **ETH**. It was found that for the non-voting items, the recorded gas costs were as low as $\mathbf{5.0445 \times 10^{-14}}$ **ETH**.

As is evident by the findings in Table 3, items that require voting have a significantly higher overall cost than non-voting items. This is due to the difference in the number of critical functions that are executed when it comes to the implemented voting procedure. It was also found that deploying the smart contract itself does indeed have an initial cost. The Ethereum gas fee for deploying *smartMinutes* is around $\mathbf{4.0886 \times 10^{-12}}$ **ETH**, given Remix IDE's [23] default local deployment settings.

5.2. Use Case Scenario 2

For this experiment, consider a committee of five members. Table 4 depicts simulated votes by the committee members as well as their relative gas costs. For this run, the speedy mode voting configuration was used.

Table 4. Speedy mode votes and gas costs.

Committee Member	Vote Cast	Gas Cost (ETH)
Member #1	1	1.1434×10^{-13}
Member #2	1	9.6680×10^{-14}
Member #3	1	1.7983×10^{-13}
Member #4	N/A	N/A
Member #5	N/A	N/A

The speedy mode configuration concludes the voting process as soon as the necessary number of votes is made to pass the agenda item. This means that, for example, if three out of five committee members cast the same vote (positive or negative vote), then the speedy mode configuration will determine the result of the agenda item and conclude the voting process.

For this simulated scenario, only three of the five committee members cast a vote. The remaining two voters, members #4 and #5 did not need to vote as per the mechanisms of speedy mode described above. This means that members #4 and #5 did not have any gas costs related to this scenario.

The same process is repeated, but using the majority voting configuration. This configuration requires all committee members to cast their vote before determining the result of the agenda item. The simulated votes, as well as the gas costs can be seen in Table 5. The gas costs associated with each member's positive vote are quite similar, with the exception of member #5, which can be regarded as an outlier.

Table 5. Majority voting votes and gas costs.

Committee Member	Vote Cast	Gas Cost (ETH)
Member #1	1	9.7547×10^{-14}
Member #2	1	8.5132×10^{-14}
Member #3	1	8.9817×10^{-14}
Member #4	1	9.4502×10^{-14}
Member #5	1	2.0008×10^{-13}

The process is repeated one last time using the veto mode voting scheme. This voting configuration allows committee members to veto an agenda item by inputting the value of -1. Upon vetoing, the voting process is immediately terminated. Table 6 demonstrates the simulated votes, as well as their gas costs.

Table 6. Veto mode votes and gas costs.

Committee Member	Vote Cast	Gas Cost (ETH)
Member #1	-1	1.8357×10^{-13}
Member #2	N/A	N/A
Member #3	N/A	N/A
Member #4	N/A	N/A
Member #5	N/A	N/A

For this particular scenario, only member #1 cast a vote. The first vote cast in this scenario is a veto vote, which terminates the voting process. The only gas cost associated with this scenario is for member #1's veto vote.

The summation of the overall gas costs for speedy mode, majority voting, and veto mode for a single run can be seen in Table 7. From the findings of this experiment, a few things can be determined about the voting schemes. First, the majority voting configuration has the highest overall gas cost, with $\mathbf{5.6708 \times 10^{-13}}$ **ETH**. Second, veto mode has the lowest cost with $\mathbf{1.8357 \times 10^{-13}}$ **ETH**. Last, speedy mode has the second lowest cost in terms of gas fees with $\mathbf{3.9085 \times 10^{-13}}$ **ETH**.

Table 7. Voting scheme gas costs.

Scheme	Total Gas Cost (ETH)
$\sum$ Speedy Mode Voting	3.9085×10^{-13}
$\sum$ Majority Voting	5.6708×10^{-13}
$\sum$ Veto Mode Voting	1.8357×10^{-13}

It is determined that the veto mode voting scheme can have the lowest possible gas cost. Based on the results of this experiment, when compared to majority voting, the veto mode voting configuration can save up to $\mathbf{3.8351 \times 10^{-13}}$ **ETH** in gas fees. The specific scenario, which allows for this, as shown above, involves the first committee member casting a veto vote, thus terminating the voting process. However, if the veto vote would be entered by the last member to cast his/her vote, then indeed the veto mode voting would have the same cost as the majority voting. Speedy mode would then be the second most cost-effective configuration, as it allows for the voting process to conclude with fewer votes than there are committee members. When compared to majority voting, speedy mode can save up to $\mathbf{1.7623 \times 10^{-13}}$ **ETH** in gas fees. Majority voting, the default voting scheme, is determined to be the least cost-effective from this experiment. A reason for this is that this configuration requires all committee members to cast a vote, with no exceptions.

6. Alternate Uses of *SmartMinutes*

At its core, *smartMinutes* consists of a meeting minute management module, as well as a voting module. With minimal changes, as a possible alternate use case, the meeting minute module can be re-purposed to keep track of important documents, as well as their corresponding status, if applicable. This could make *smartMinutes* suitable for organisations of any size to utilise as a bureaucratic system. With modifiable data access policies, *smartMinutes* can provide users with the ability to track documents and their statuses at an organisation-wide level. This could provide users with a reliable way to track their documents, as well as the bureaucratic process, without the need for any direct input from the organisation.

Electronic voting (e-voting) at an electoral level is a developing field. The key goal of developing this type of technology is to improve the overall experience involved with participating in government elections, as well as make it more accessible for citizens to cast their vote. The existing voting module in *smartMinutes* can be adapted and deployed at a wide-scale level as an e-voting system using blockchain technology. With the correct governmental integration, this would provide citizens with a voting system that is tamper-proof, given the blockchain's immutability core attribute. On top of that, users would be able to maintain their anonymity whilst maintaining voter data integrity. The discussed benefits are a few examples of many more [26]. This implementation can indeed increase accessibility, which could then, in turn, increase overall participation in elections. Additionally, the process of counting the votes or tallying can be automated, which would then mean that election results can be determined much faster. This then could possibly shorten the length of the entire election process.

The adoption of a modified *smartMinutes* or any other blockchain does present some challenges [26]. First, the complexity of implementing the necessary technological infrastructure to support such frameworks might prove to be a difficult task. Second, in their current state, blockchain networks can be quite slow, and considering a scenario where the masses would be making transactions in order to cast their vote may subject the network to further congestion.

With the constant efforts of developers, making improvements may mean that this type of solution would be feasible to implement. The Ethereum Mainnet upgraded from a proof-of-work model to proof-of-stake [27]. This, as it is referred to as "The Merge", is said to reduce the blockchain's energy usage by approximately 99.95%. This example of a major change to the existing technology can be an indication that in the near future, implementations such as blockchain-based e-voting may become more feasible. As for the effects "The Merge", in terms of gas fee means for *smartMinutes*, it is reported [28] that this upgrade to the Mainnet will not affect its overall capacity, as it is a change to the consensus mechanism.

7. Conclusions

In this paper, *smartMinutes*, a framework solution for the management of meeting minutes, is presented. With the combination of an agenda item voting functionality, this framework aims to resolve challenges present within existing solutions. To the best of our knowledge, this is not only the first blockchain-based platform for meeting minutes management, but also the only framework that integrates voting as part of the platform.

The experimental findings show that the veto mode implementation can be the most cost-effective voting scheme in some scenarios when compared to the majority voting and speedy mode schemes. Determining the cost-effectiveness of the supported schemes in the framework can better outline the feasibility of the framework financially. From the total costs of the simulated scenarios, it is found that the highest gas cost for the whole voting process of a particular agenda item is about $\mathbf{5.6708 \times 10^{-13}}$ **ETH**. This value can then be regarded as an approximation of the maximum cost of utilising the voting module of *smartMinutes*. This promising result can indicate that the presented framework is financially feasible, given that the Ethereum Mainnet is not congested.

Possible use cases of *smartMinutes* in other domains may include a transparent organisation-wide documents tracker with adjustable privacy settings, or even, with some adjustments, a standalone e-voting solution. The future directions of *smartMinutes* include the design and implementation of an easy-to-use, user-friendly front-end interface that communicates with *smartMinutes*. As it stands, some technical knowledge is required to utilise the presented framework, however, with the use of a well-designed front-end that uses *smartMinutes* as a back-end, the framework can become more accessible to a greater range of users, regardless of technical background. Another future enhancement may include the support of a configurable multi-tier hierarchy. By implementing a dynamic mechanism to adjust the hierarchy, *smartMinutes* may become easier to be adopted by different organisations, as their company structure may be more complex than a simple 2-tier hierarchy. Last but not least, it is in the future plans to integrate a richer set of attributes for specifying agenda items. This can then help improve the clarity of the nature of agenda items, as well as include a wider range of related tools.

Author Contributions: All authors have made substantial contributions to the submitted work. All authors approved the submitted version and agree to be accountable for its contents. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

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

References

1. Morand, T. How to Write Effective Meeting Minutes (with Templates and Samples). WildApricot Blog. Available online: https://www.wildapricot.com/blog/how-to-write-meeting-minutes (accessed on 6 April 2020).
2. Mohammed Farooq Abdulla, F.M.; Pawankumar, S.; Guruprasath, M.; Jayaprakash, J. Automation of Minutes of Meeting (MoM) using Natural Language Processing (NLP). In Proceedings of the 2022 International Conference on Communication, Computing and Internet of Things (IC3IoT), Chennai, India, 10–11 March 2022; pp. 1–6. [CrossRef]
3. Yulyanto, M.T.; Khodra, M.L. Automatic extractive summarization on Indonesian parliamentary meeting minutes. In Proceedings of the 2017 International Conference on Advanced Informatics, Concepts, Theory, and Applications (ICAICTA), Denpasar, Indonesia, 16–18 August 2017; pp. 1–6. [CrossRef]
4. Lin, C.-Y. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2004; pp. 74–81. Available online: https://aclanthology.org/W04-1013 (accessed on 1 November 2022).
5. Chen, X.; Sheng, F.; He, R.; Chen, S.; Ma, H.; Wu, Y.; Xu, J. Yanji: An Automated Mobile Meeting Minutes System. In Proceedings of the 2021 2nd International Symposium on Computer Engineering and Intelligent Communications (ISCEIC), Nanjing, China, 6–8 August 2021; pp. 425–428. [CrossRef]

6. Hirashima, D.; Tanaka, M.; Teshigawara, Y. Development and evaluation of a minutes system focusing on importance in a meeting. In Proceedings of the 18th International Conference on Advanced Information Networking and Applications, Fukuoka, Japan, 29–31 March 2004; AINA 2004. Volume 2, pp. 293–298. [CrossRef]
7. Zhang, J.J.; Fung, P. Automatic Parliamentary Meeting Minute Generation Using Rhetorical Structure Modeling. *IEEE Trans. Audio Speech Lang. Process.* **2012**, *20*, 2492–2504. [CrossRef]
8. Mann, W.C.; Thompson, S.A. Rhetorical Structure Theory: Toward a functional theory of text organization. *Text Interdiscip. J. Study Discourse* **1988**, *8*, 243–281. [CrossRef]
9. Chiu, P.; Boreczky, J.; Girgensohn, A.; Kimber, D. LiteMinutes: An Internet-based system for multimedia meeting minutes. In Proceedings of the Tenth International Conference on World Wide Web—WWW'01, Hong Kong, China, 1–5 May 2001; pp. 140–149. [CrossRef]
10. Liu, H.; Wang, X.; Wei, Y.; Shao, W.; Liono, J.; Salim, F.D.; Deng, B.; Du, J. ProMETheus: An Intelligent Mobile Voice Meeting Minutes System. In Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, New York, NY, USA, 5–7 November 2018; pp. 392–401. [CrossRef]
11. Lee, J.-K.; Song, H.-J.; Park, S.-B. Two-Step Sentence Extraction for Summarization of Meeting Minutes. In Proceedings of the 2011 Eighth International Conference on Information Technology: New Generations, Las Vegas, NV, USA, 11–13 April 2011; pp. 614–619. [CrossRef]
12. Ito, H.; Miyazato, K.; Ishikawa, K.; Taki, T.; Hasegawa, J.; Raita, K. Structure and retrieval mechanism of a minutes retrieval system. In Proceedings of the 2017 IEEE 21st International Conference on Intelligent Engineering Systems (INES), Larnaca, Cyprus, 20–23 October 2017; pp. 000291–000296. [CrossRef]
13. Evangelopoulos, N.; Zhang, X.; Prybutok, V.R. Latent Semantic Analysis: Five Methodological Recommendations. *Eur. J. Inf. Syst.* **2012**, *21*, 70–86. [CrossRef]
14. Yu, H.; Clark, C.; Malkin, R.; Waibel, A. Experiments in automatic meeting transcription using JRTK. In Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181), Seattle, WA, USA, 15 May 1998; Volume 2, pp. 921–924. [CrossRef]
15. Koay, J.J.; Roustai, A.; Dai, X.; Liu, F. A Sliding-Window Approach to Automatic Creation of Meeting Minutes. *arXiv* **2021**, arXiv:2104.12324. Available online: http://arxiv.org/abs/2104.12324 (accessed on 1 November 2022).
16. Nishita, S.; Itoh, M. Extracting Relationship of Meeting Minutes Generated by Speech Recognition System using Entity Resolution. *IAENG Int. J. Comput. Sci.* **2016**, *43*, 284–289.
17. Bhattacharya, I.; Getoor, L. Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data* **2007**, *1*, 5-es. [CrossRef]
18. Eisenstein, L. How to Take Minutes at a Board Meeting. BoardEffect. Available online: https://www.boardeffect.com/blog/how-to-take-minutes-ata-board-meeting/ (accessed on 15 July 2019).
19. Al-Jaroodi, J.; Mohamed, N. Blockchain in Industries: A Survey. *IEEE Access* **2019**, *7*, 36500–36515. [CrossRef]
20. Bannet, J.; Price, D.W.; Rudys, A.; Singer, J.; Wallach, D.S. Hack-a-vote: Security issues with electronic voting systems. *IEEE Secur. Priv.* **2004**, *2*, 32–37. [CrossRef]
21. Reed, J. *Smart Contracts: The Essential Guide to Using Blockchain Smart Contracts for Cryptocurrency Exchange*; Wydawca Nieznany; CreateSpace Independent Publishing Platform: Scotts Valley, CA, USA, 2016.
22. Zheng, Z.; Xie, S.; Dai, H.-N.; Chen, W.; Chen, X.; Weng, J.; Imran, M. An overview on smart contracts: Challenges, advances and platforms. *Future Gener. Comput. Syst.* **2020**, *105*, 475–491. [CrossRef]
23. Remix—Ethereum IDE & Community. Available online: https://remix-project.org (accessed on 1 November 2022).
24. Peaster, W.M. Ethereum Gas Explained. DeFiprime.com. Available online: https://defiprime.com/gas (accessed on 22 September 2022).
25. Sheikhi, A.S.H. smartMinutes: A Blockchain-Based Approach to Automate the Meeting Minutes Process. Master's Thesis, University of Nicosia, Nicosia, Cyprus, 2022.
26. Kshetri, N.; Voas, J. Blockchain-Enabled E-Voting. *IEEE Softw.* **2018**, *35*, 95–99. [CrossRef]
27. The Merge. Ethereum.Org. Available online: https://ethereum.org/en/upgrades/merge/ (accessed on 21 September 2022).
28. Salvo, M.D. What the Ethereum Merge Means for Ordinary Users—And What It Doesn't. Decrypt. Available online: https://decrypt.co/109724/what-ethereum-merge-means-and-doesnt-users (accessed on 14 September 2022).