DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

E-Voting Application using Blockchain

CS 258: Software Engineering Lab

Name	**Roll Number**
Krish Agrawal	210001034
Nishkarsh Luthra	210001045
Jha Rohan	210002041
Rishika Sharma	210002063
Hrishesh Sharma	210003037

Course Instructor: Dr. Puneet Gupta

May 2, 2023

Contents

1 Introduction

1.1 Purpose

Traditional voting methods fail to ensure voter convenience and the authenticity of the election results. Voters need to stand in long queues to cast their votes, which makes the process long and cumbersome. Also, avoiding and penalising rigging during elections is a climb uphill. Election-conducting bodies often fail to do so. Online elections using traditional databases and GUI fail to provide authenticity and required security to satisfy the needs.

To mitigate the issues mentioned above, the project aims at building a Blockchain-Based Voting System capable of decentralisation that will enable users to vote from their mobiles and laptops securely. This software has been exclusively designed for College Gymkhana Elections at IIT Indore. These elections are characterised by concurrent voting for multiple positions. It leverages the power of blockchain to make the election system secure and safe. Blockchain makes use of cryptographic hashing, thereby making the system more robust. This avoids tampering with results and ensures that the results are safely verified, making the complete procedure of election rigging free.

1.2 Intended Audience

This documentation will serve as a guide for the developer, stakeholders(college administration) and users (students who will cast their votes in the college gymkhana elections and the students who are contesting for various positions). It will also serve the college's administration by conducting safe and secure Gymkhana Elections.

1.3 Project Scope

Blockchain-based E-Voting system will create a platform for students and admins wherein secure and transparent Gymkhana Elections can be conducted. The software developed can accommodate up to 5000 students (voters) and store the votes cast by them on the chain.

We sincerely hope this software will make the college elections safe, secure and rigging-free. Using this project, we aim to conduct the 2024 Student Gymkhana Elections of the Indian Institute of Technology Indore on our platform.

2 Developer Documentation

2.1 Overall Description

2.1.1 User-Classes and Characteristics

The "E-Voting application using blockchain" has the following categories of users:

1)Administration or Election officers: They are responsible for conducting safe and secure elections in a smooth manner.

2)Voters

2.1.2 Product Functions

1)The "E-Voting Application using Blockchain" consists of a back-end which will store the information of all the eligible, non-blacklisted voters. They can log in (the software makes use of Google oauth authenticator for creating the back-end of login) using their institute email id and will be able to access the page to vote.

2)There, the voters will be able to see all the candidates contesting for different posts and will be able to cast their votes. The contesting candidates will also be provided with a vote in line with the election rules of the student gymkhana. For example, for Junior senator CSE, only first and second-year CSE UG students can cast a vote.

3) Once the user casts his/her votes, it is first sent to a queue containing unconfirmed transactions. Then, the POW is calculated, the block hash is calculated, and the block is added to the chain.

4) After the election concludes, the chain is verified, and the votes are counted to declare the winner. The application thus enables the hosting of secure elections, with the security of blockchain and appropriate user-friendly GUI.

2.1.3 Operating Environment

The website can function on any browser or operating system(Mac, Windows or Linux). The website will be responsive for all screen sizes.

2.2 System Features

The "E-Voting Application using Blockchain" has some necessary features, and the others are optional for this software.

2.2.1 Description and Priority

The features with priority up to down are:

1)Hosting Elections: This is the pivotal feature of this software. It will be operated by the college's administration or election officers. They will input the candidates contesting for each position in the election.

2)Voting: This is yet another pivotal feature of this software. This feature will be exercised by college students, who will cast their votes in the elections for multiple positions.

3)Result Creation: This will automatically be executed after the successful completion of the voting process. Here, the complete chain will be verified, and if any anomalies are found, they will be reported to the administration or the election officers. It will tabulate the total votes tally of all the contestants for all the listed positions. Visualisation of data can be done too.

4) Publishing the result: It is basically a notice. The administration or the Chief Election Officer will execute it. It will make the election result visible to all the stakeholders after verification by the administration or the election officers.

5)Blacklist: It is a function through which the administration or the Chief Election Officer will be able to blacklist a candidate in the election if he/she does not abide by the rules of the election and engages in unfair means.

6)Deny Vote: It is a function through which the administration or the Chief Election Officer can revoke the voting rights of a student.

2.2.2 Functional Requirements

The "E-Voting Application using Blockchain" website has been built on the following tech stack:-

Back-End:
This is the main logic and data unit of the software. It has also been used to encapsulate the chain(s) of the software.

1)Flask framework
2)Python language

Front-End:
It is the the display and visual component of the software, from where the users(administration and voters) will interact with the software.

HTML/CSS

Database:
Used for the data storage for the complete software.

MongoDB Atlas

2.2.3 Non-Functional Requirements

Performance Requirements:
This project will be used to conduct college Gymkhana Elections and can accommodate up to 5000 students. Our software supports various operating systems, so users using different kinds of operating systems can make use of it to cast their votes. The front end is responsive, so it is supported by various devices.

Security Requirements:

Non-registered users cannot cast their votes. A user cannot vote multiple times for the same candidate in order to ensure fair elections. A user cannot change the vote cast by someone else. The system will protect the privacy of each voter by ensuring the anonymity of the vote and will also protect the personal data of the users.

Software Quality Attributes:
The system will be able to handle College Gymkhana elections (i.e. about 5000 voters). It is easy to use and intuitive, with clear instructions and user-friendly interfaces. It complies with all relevant legal and regulatory requirements, including data protection and privacy laws, election laws, and anti-corruption laws.

Business Rules:
Each voter is allowed only one vote per Gymkhana position in the election. The system makes use of a secure user authentication method to ensure that only eligible voters cast their votes. The voting process will be transparent so that all the parties involved can see the results and verify the authenticity of the process.

2.3 UI/UX Description

The front end was developed using HTML(hyper-text markup language) and CSS(cascading style sheets). The website comprises the following pages:

Voting: This page enables a user to vote for his/her preferred candidate.

Voters: This page allows the Admin to add voters to the voting list. The details added include the name of the voter, the institute email id and the branch of the voter.

Candidates: This page enables the Admin to add a candidate and the position he/she is contesting for.

Position: This page facilitates the addition of new positions in the Student Gymkhana by the Admin.

Branch: The admin makes use of this page to add new branches and add categories to the branch.

Dashboard: This is used by the Admin to set the date, the start and end times of the voting process, and publish the result.

Already: This page is shown when a user who has already voted tries to vote again.

Notvoter: When a non-registered voter tries to access the website, he/she is redirected to this page.

Result: This page publishes the result by tabulating the names of the candidates, the position they were contesting and the votes received by them.

Thanks: This page tells the user that his/her votes have been cast.

Notime: This page tells the user that the time for the voting process is over and that votes can no longer be cast.

Admin Template: This page helps the Admin to navigate to the Dashboard, Voters, Candidates, Positions and Branch page.

2.4 Chain Description

2.4.1 Store Transactions into Blocks

The data is stored in the blockchain in JSON format. The data stored in the blockchain will look somewhat like:
{"mail": "ee210002041@iiti.ac.in"
"Votes": "c1000,c2000,c3000....," }

The transactions are packed into blocks. A block contains one transaction each. The blocks containing the transactions are generated frequently and added to the blockchain. Because of the presence of multiple blocks, each block has a unique id.

```
class Block:
    #Initialise block
    def __init__(self,id, transactions, timestamp, previous_hash):
        self.id= id
        self.transactions= transactions #data of the block
        self.timesptamp= timestamp
        self.previous_hash= previous_hash
        self.nonce= 0 #facilitates POW
        self.hash= None
```

2.4.2 Hash the Blocks

To detect if the data in the block is tampered with, we have used cryptographic hash functions. A hash function is a function that takes data of any size and produces data of a fixed size from it (called hash), which is generally used to identify the input. The characteristics of an ideal hash function are:

1)It should be computationally easy to compute.

2)It should be deterministic, meaning the same data will always result in the same hash.

3)Should be uniformly random, meaning even a single bit change in data should change the hash significantly.

The consequence of this is:

1)It is impossible to guess the input data given the hash (the only way is to try all the possible input combinations).

2)If you know both the input and hash, you can simply pass the input through the hash function to verify the provided hash.

```
def find_hash(self):
    try:
        block_json_string= json.dumps(self.__dict__, sort_keys=True)
        return sha256(block_json_string.encode()).hexdigest()
    except(TypeError): #For handling the set
        if(list(self.transactions)):
            x=(list(self.transactions))
            block_json_string= json.dumps([list(x), self.nonce])
            return sha256(block_json_string.encode()).hexdigest()
        block_json_string= json.dumps([list(self.transactions), self.nonce])
        return sha256(block_json_string.encode()).hexdigest()
```

2.4.3 Chain the Blocks

In order to chain the blocks, we have stored the hash of the previous block in the current block. The first block created is known as the genesis block. The previous hash of the genesis block was set to "0" and the block-id was set to 0.

```
def create_genesis_block(self):
    genesis_block= Block(0,[],time.time(), "0")
    genesis_block.hash= genesis_block.find_hash()
    self.chain.append(genesis_block)
```

2.4.4 Implementing a Proof of Work Algorithm

If we change the previous block, we can re-compute the hashes of all the following blocks quite easily and create a different valid blockchain. To prevent this, we have exploited the asymmetry in efforts of hash functions to make the task of calculating the hash difficult and random. Instead of accepting any hash for the block, we have added some constraints to it. Our constraint is that our hash should start with "n leading zeroes", where n can be any positive integer.

```
@staticmethod
#Method to secure the chain further and reduce concurrency problems
def proof_of_work(self, block):
    block.nonce= 0 #Used to get the required hash. It is changed to satisfy the condition of hash
    curr_hash= block.find_hash()
    while not curr_hash.startswith('0'*Blockchain.difficulty):
        block.nonce+=1
        curr_hash= block.find_hash()
    return curr_hash
```

2.4.5 Add blocks to a Chain

To add a block to the chain, we have verified that:

1) The data has not tampered i.e., the Proof of Work provided is correct.

2) The order of transactions is preserved i.e. the previous_hash field of the block to be added points to the hash of the latest block in our chain.

```
def add_block(self, block, proof):
    previous_hash= self.last_block.hash
    if previous_hash!= block.previous_hash:
        return False
    if not Blockchain.is_valid_proof(block, proof):
        return False
    block.hash= proof
    self.chain.append(block)
    return True
```

2.4.6 Mining

Initially, the transactions are stored as a pool of unconfirmed transactions. The process of putting the unconfirmed transactions in a block and computing the Proof of Work is known as the mining of blocks. Once the nonce satisfying our constraints is figured out, we can say that a block has been mined, and it can be put into the chain.

```
def mine(self):
    if not self.unconfirmed_transactions:
        return False
    last_block= self.last_block
    for i in range(0, len(self.unconfirmed_transactions)):
        new_block= Block(id= last_block.id+1, transactions= self.unconfirmed_transactions[i],
                         timestamp=time.time(), previous_hash= last_block.hash)
        proof= self.proof_of_work(self, new_block)
        self.add_block(new_block, proof)
    self.unconfirmed_transactions= []
    return True
```

2.4.7 Create Interfaces

Finally, we created interfaces for our blockchain node to interact with the application we built. We have used Flask to create a REST-API to interact and invoke various operations in our blockchain node.

```
from flask import Flask, request
import time
import blockchain
import json
app= Flask(__name__)
Blockchain= blockchain.Blockchain()
Blockchain_voter= blockchain.Blockchain()
Blockchain.create_genesis_block()
Blockchain_voter.create_genesis_block_set()
@app.route('/')
def home():
    return "Hello, Flask!"


@app.route('/login', methods=['POST'])
def chk_double():
    prev= Blockchain_voter.last_block.transactions
    trans_data= request.get_json()
    if trans_data["mail"] in prev:
        return "Chtya samjha hai kya?", 404
    prev.add(trans_data["mail"])
    trans_data["timestamp"]= time.time()
    trans_data["mail"]= prev
    Blockchain_voter.add_new_transaction(trans_data)
    return "Sahi hai", 201
```

```
@app.route('/new_transaction', methods=['POST'])
def new_transaction():
    trans_data= request.get_json()
    required_fields= ["mail","votes"]
    for field in required_fields:
        if not trans_data.get(field):
            return "Chtya hai kya", 404
    trans_data["timestamp"]= time.time()
    Blockchain.add_new_transaction(trans_data)
    return "Sahi hai", 201

@app.route('/chain', methods=['GET'])
def get_chain():
    chain_data= []
    for block in Blockchain.chain:
        chain_data.append(block.__dict__)
    return json.dumps({"length": len(chain_data), "chain": chain_data})

@app.route('/mine', methods=['GET'])
def mine_unconfirmed_transaction():
    result= Blockchain.mine()
    if not result:
        return "Khali hai"
    return "Block #{} is mined".format(result)



@app.route('/pending_trans')
def get_pending_trans():
    return json.dumps(Blockchain.unconfirmed_transactions)

if __name__ == "__main__":
    app.run()
```

3 User Documentation

3.1 Installation Guide

1) Install the latest version of python along with pip using the url `https://www.python.org/downloads/`.

2)Install venv using the command
pip install venv

3)Create a new directory and navigate to the directory using the command
cd name_of_the_directory

4)Clone the project by typing
git clone `https://github.com/Krish2208/HashVote`
in the terminal.

5) Create a virtual environment for the cloned project using the command
python -m venv my_venv

6) Activate the virtual environment using the command
source ./my_venv/bin/activate

7)Install the dependencies by typing
pip install -r requirements.txt
in the terminal.

8) To launch the app, type
python main.py
in the terminal.

3.2 User Manual

1)Login using your institute Email-ID.
After logging in successfully, you will be able to look at the candidates contesting for every position.

2)You are required to cast your votes for various positions. For every position, you can cast only one vote for a candidate contesting for the respective position.

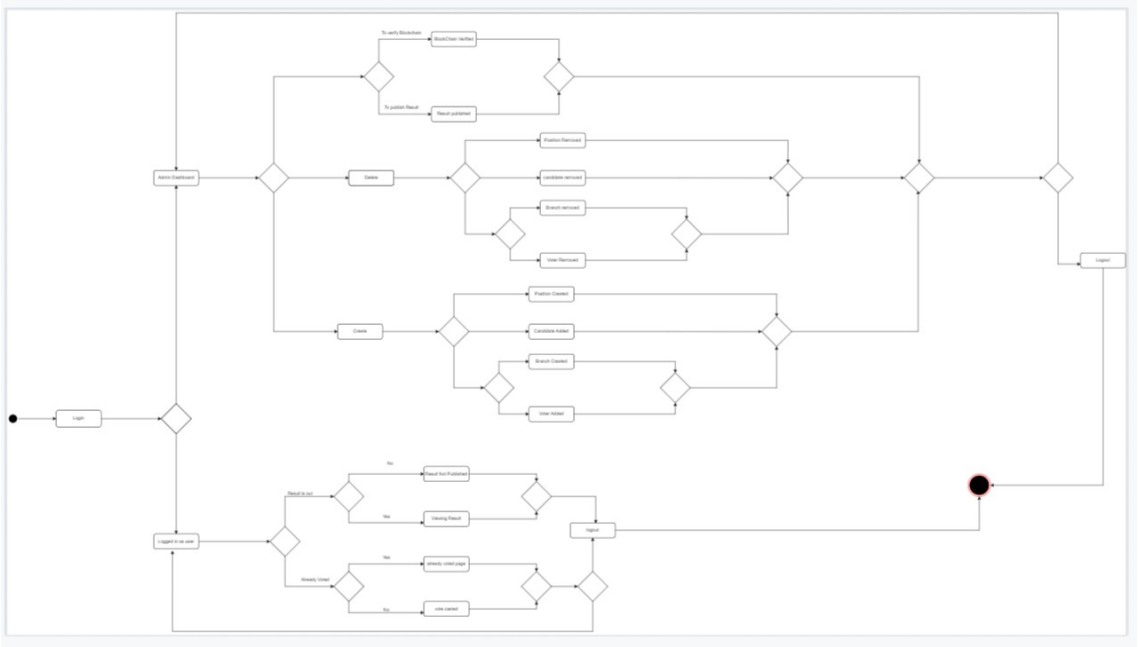
3)Your preferred choice of candidates will be uniquely identified using a hash (which will be generated using a hash function) and timestamp-based protocol.

4)Only admins will have access to the blockchain-based database where the votes cast by students will be stored. Admins can debar a student from casting his/her vote and can debar a candidate from contesting in the election under disciplinary action.

4 System Documentation

4.1 Activity Diagram

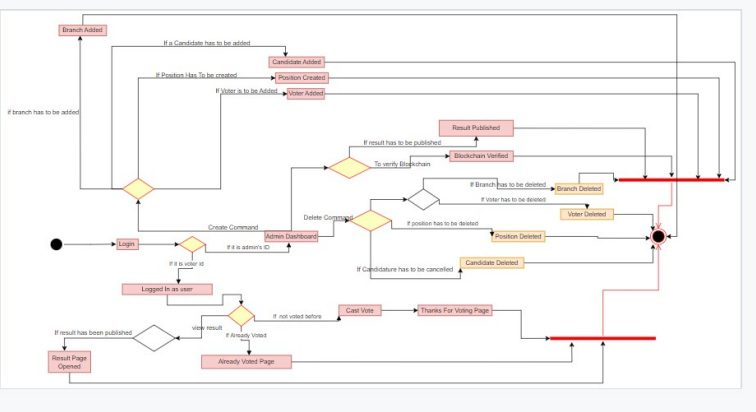
An activity can be described as an operation performed by the system. This activity diagram is basically a flowchart to represent the flow from one activity to another activity.



Activity Diagram

4.2 State Diagram

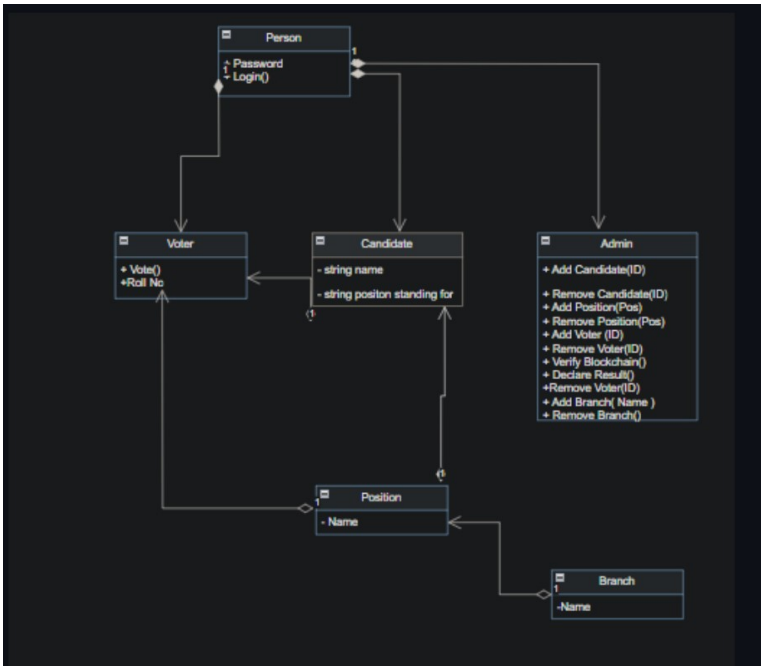
This diagram shows the behaviour of the system in response to external factors. It describes the behaviour of a single object in response to a series of events in a system.



State Diagram

4.3 Class Diagram

This class diagram developed using Unified Modeling Language (UML) describes the structure of our system by showing the system's classes, their attributes, operations, and the relationships among objects.

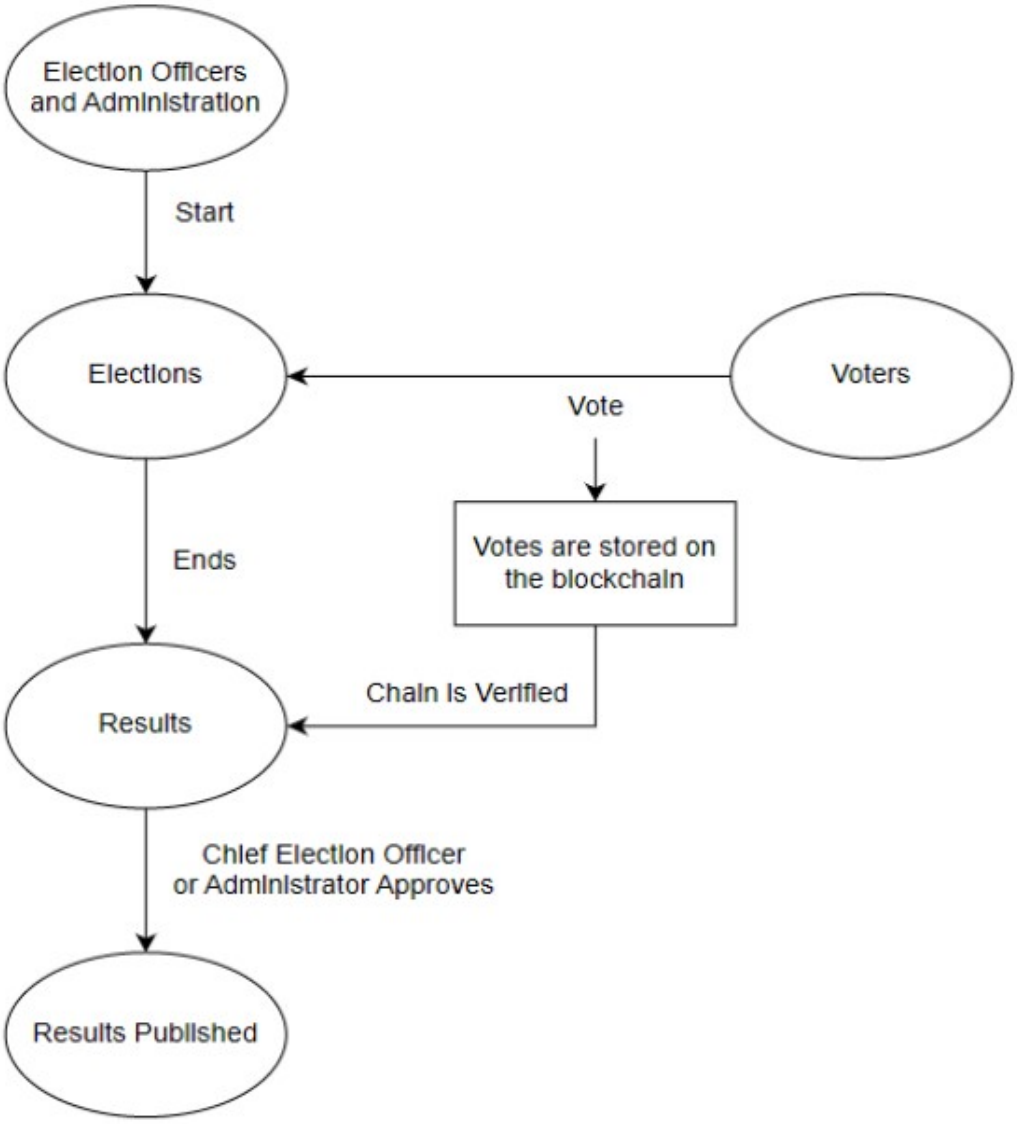


Class Diagram

5 Design Documentation

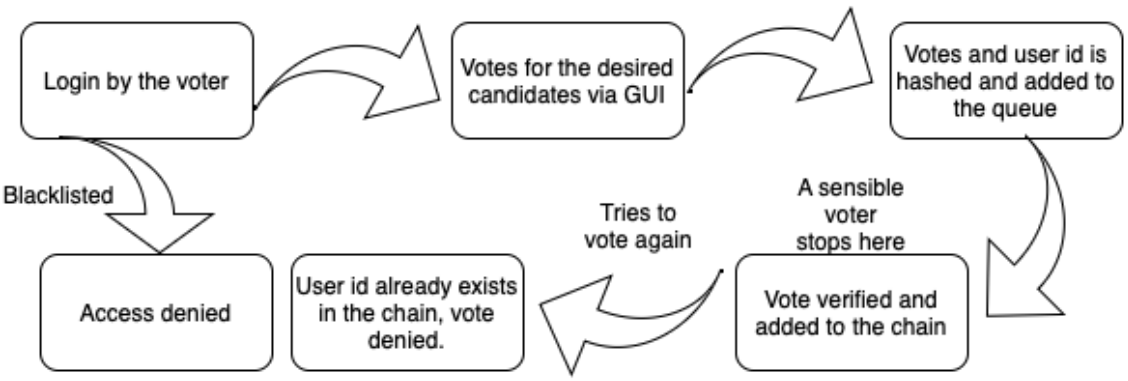
5.1 Flow Diagrams

5.1.1 Software Workflow



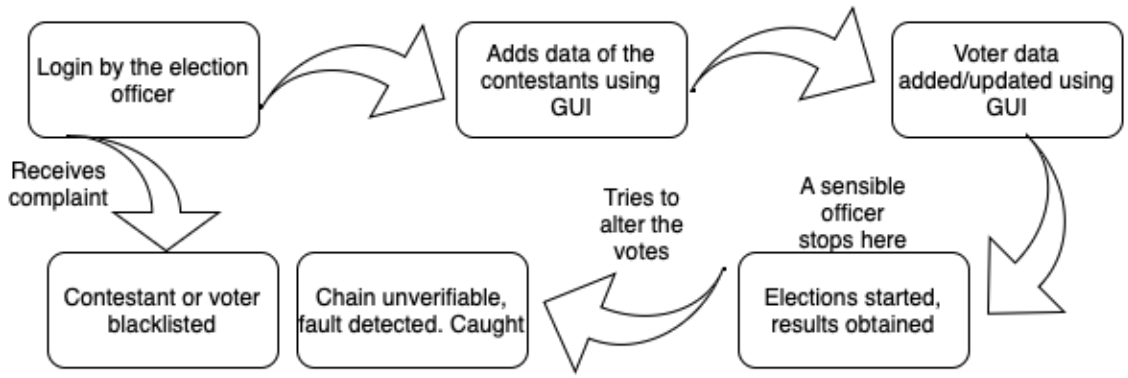
Software Workflow

5.1.2 Voting Flow



Voting Flow

5.1.3 Admin Flow



Flow Diagram describing the role of the Election Officer

5.2 MongoDB Schema