Software Engineering - 14:332:424
Group 1

Blockchain-secured Voting Application

Website: sites.google.com/view/vote4me
8th December 2019

Members:
Rahul Katyal
Louis Moccia
Parth Patel
Rahul Patel
Alec Rodrigues
Rani Sayed
Hari Shetty
Vancha Verma

Contents

Individual Contributions 3

Summary of Changes 4

Section 1
- Problem Statements 5

Section 2
- Glossary of Terms 9

Section 3
- System Requirements 10

Section 4
- Functional Requirements Specification
  - Stakeholders 17
  - Actors & Goals 18
  - Use Cases 20
  - System Sequence Diagrams 24

Section 5
- User Effort Estimation 26

Section 6
- Domain Analysis
  - Domain Model 27
  - System Operations Contract 28

Section 7
- Interaction Diagrams 31

Section 8
- Class Diagram and Interface Specification
  - Class Diagram 34
  - Data Types and Operation Signatures 35
  - Traceability Matrix 36

Section 9
- System Architecture and System Design
  - Architectural Styles 37
  - Identifying Subsystems 38
  - Mapping Subsystems to Hardware 39
  - Persistent Data Storage 40
  - Network Protocol 41
  - Global Control Flow 42

Hardware Requirements 43
Section 10
Data Structures and Algorithms
Algorithms 44
Data Structures 44
Section 11
User Interface Design and Implementation 46
Section 12
Design of Tests 48
Section 13
History of Work, Current Status, and Future Work 51
Section 14
References 53

Individual Contributions:

Contributions	Team Member Name							
	Rahul K.	Louis M.	Parth P.	Rahul P.	Alec R.	Rani S.	Hari S.	Vancha V.
Report 1	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
Report 2	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
Report 3	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
Demo 1	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%
Demo 2	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%	12.5%

All group members contributed equally.

Summary of Changes

Section 1: Added a conclusion from that summarized all of the interviewees opinions, and how our system will alleviate these issues.

Section 2: Some relevant words were added to the glossary.

Section 3: System Requirements were reorganized into proper categories (functional vs nonfunctional) and reprioritized based on what was more relevant to the system

Section 4: Removed use cases that have not been implemented and marked them as *FUTURE IMPLEMENTATIONS*

Section 5: The effort estimation for the use cases has been added in.

Section 6: The description has been updated to coincide with the use cases that the application now covers.

Section 7: Short descriptions were added to include the failed cases to give the reader a more thorough description of how the higher valued use cases work on the software.

Section 8: Certain data types have been edited for more clarity.

Section 9: Clarified certain inconsistencies (no database and explained how the system will bypass it, clarified the execution order and made it an event driven system)

Section 10: The data structures section was clarified and made more inclusive. The arraylist was not clear in the run time of adding an element.

Section 11: The Description for the UI log-in page has been updated to be more clear about what the user will be entering to log-in to their account.

Section 12: Added use case for logging in, added UI testing

Section 13: New Section

Section 14: Added more information for some of the resources

Customer Problem Statements

“As a voter it can be pretty hard to get myself to the ballot box on election day. I mean how am I supposed to find the time to physically go and vote, you know its on a Tuesday, right? What, I’m supposed to go up to my boss and ask for almost 2 hours off so I can drive to my polling district, vote, and come back? He’d laugh in my face and tell me ‘America was built on hard work, so lets see some of that instead’... I know, i’ve tried. Now, if you're telling me I could do it from my phone while on break...”

- **Disgruntled Voter working 2 low-income jobs**

“That last election wasn't even worth trying. They already knew they were gonna win, system was rigged to start with. Between the hacking, and spying, and the money involved, it's impossible to think that our vote could have made a difference. It’s not like we can ask for proof, not that they wouldn’t have changed ‘em or thrown ‘em out, whatever they do. We need accountability, a way to make sure our votes aren't played with. Otherwise, I can't see how this is gonna end.”

- **Anxious Voter**

“Yes sir, I have been a poll worker for the past 4 elections. Most people that come through on election day are older folk, the kids usually have other priorities. I tried to bring my grandson last time too, but the boy would rather sit on his phone all day!”

- **Elderly Poll Worker**

“We need the younger generation to vote. We need the older generation to vote. We need every American to vote. Simply put, they’ll never see the changes they want if they continue to not make their voices heard. As my constituents have agreed with me, there is a fundamental issue in our voting system. It is susceptible to manipulation and as a result a danger to the free will that every person that stands on this soil is entitled. Security measures are crucial to ensure the integrity of our system of power, as having the right to choose who leads us is the backbone of our Constitution.”

- **Unnamed Senator**

“Heard about blockchain before? It’s neat, same tech as that bitcoin stuff. My niece was telling me to invest in it back in the day, that it’d ‘revolutionize money’. Smart kid. Now she’s talking about how it can be used in Healthcare to track medical records. She says it’s crazy secure, makes me think it can be used in other places too. Maybe this voting system needs to upgrade to something more secure like this.”

- **Unknown**

“After the catastrophic failure of the ability of the United States to manage the security of the election system in the previous 2016 election. It is quite necessary for a newer, more advanced system to be put in place. There are too many existing vulnerabilities with the current system. If no action is taken to rectify the immediate issues we have the ability of the country to maintain its own election, and protect the democratic rights of the people will be severely undermined. A solution to this issue is urgently needed!"

- **National Security Analyst**

"Ya know I've been working this land for a long time and I know it ain't much but it's honest work. But I really feel like I have ta do my job in this country, and vote. Unfortunately, I find it really hard sometimes I can't just go to a pollin' station, cause I got a lotta work to do. And trust me I know about the early votin' mail stuff. But drivin' to the post office, can be real tough too. Nearly an hour away, might as well not vote at that point. I just wish there was an easier way to vote for somebody like me"

- **Farmer**

"Honestly I want to vote, I truly do. But what's the point anymore, there's such a huge fuss to get to the polling station, and I have to make sure I pick up my kids from school everyday. Plus I work two jobs, and barely have any time. So to hear this news about our election being compromised, and for my vote not to matter. What's the point any more, I just wish we had a way to vote that can't be hacked, some sort of way that would just make it so much easier for me to vote, and vote with a peace of mind.... Ugh"

- **Upset Mom**

"I feel really frustrated these days, I really want to vote. But I have no convenient way to get to the polling station, ever since I broke my spine in the accident 12 years ago. I know a lot of people would tell me to just go and vote early. But they don't allow that in my state. I mean what am I supposed to do? I really feel like it's my responsibility to go and speak my mind. I think it's everybody's responsibility. But of course the way the system is set up it's just too hard for me. I really wish there was an easier way, just some sort of method that I could vote at home or something…"

- **Disabled Voter**

"There's been so much fraud done to those who just want to exercise their civic responsibility. The level of voter disenfranchisement is unimaginable, and the current system must change. But I feel that we are at a difficult position in our current climate. The number one most important challenge I face in getting my fellow citizens to vote, is ease of access. So many get intimidated by the lack of protection they have when they go to vote, that they are scared off. If there was a simpler solution, just some sort of method in place to enable these voters to vote more anonymously. I truly feel like we could increase voter turnout, in many disenfranchised communities if we could just find an easier solution…"

- **Prominent Community leader**

Conclusion: It is clear that a large number of society finds it difficult to make it to the polling station, and has gripes with the sense of security of their vote, based on the comments of a number of the interviewees. In our proposed system, we propose a solution to this. We give users the ability to cast their vote from their couch at home, or their desk at work, or anywhere in between. By giving the users the ability to vote in such a remote way, we are increasing the possible turnout, since users are able to vote without having to take off from work on election day, or having to invest a large amount of time to go and cast their vote. Also, in our proposed system, we are giving the users a sense of security that their vote cannot be changed, by implementing our system on the basis of a blockchain network, that is decentralized, making the data stored on *every* user's device, making it almost impossible to alter with the result set.

Glossary

51% Attack - the scenario when more than half of the computing power of a blockchain network is controlled by one entity (single or group), and they experience a conflicting transaction, that can harm the network, if the intent is malicious

Block - packages of data (similar to the value in a Linked List) that carries permanently recorded data in the chain

Blockchain - a line of blocks in which the pointers pointing to the next block is encrypted, keeping the data private and being unable to hack the values in the blocks unless directly accessed

Block Height - the number of blocks in the chain

Confirmation - act of successfully hashing a value into the chain

Decentralized Application - an open source application that operates on its own, and has its data stored on a blockchain, instead of the usual database approach

Difficulty - refers to how easily the data in a block can be mined (or retrieved)

Fork - creates an alternate version of the current chain (duplicate), which leaves two chains running on 2 different networks

Genesis Block - the first block or first few blocks in the chain

Hard Fork - reverts the status of each of the transactions in the chain (successful becomes unsuccessful, and vice versa)

Peer to Peer Network - refers to the decentralized network between multiple parties in a network that is intertwined. Usually, there is only one mediation point in which participants are able to communicate to the other networks in.

Soft Fork - makes ONLY valid transactions invalid, does not change the older invalid transactions, and therefore the outcome is a ledger of invalid transactions

Solidity - Programming Language used to instantiate and maintain blockchain network

System Requirements

Functional Requirements

On a scale of 1-5 since it is easiest to split up into what is necessary and what is extraneous from these priority weights (PW) for each requirement (REQ-x)

REQ-x	PW	Descriptions
REQ-1	4	Each registered voter with a valid State ID, Driving Permit, or State Issued Driving License can login/create an account.
REQ-2	2	Each voter account should be associated with only one State ID.
REQ-3	2	No voter should be allowed to vote more than once.
REQ-4	2	Voters are not allowed to modify their votes after submission.
REQ- 5	5	Voting should be handled by a trusted third party.
REQ-6	1	Voters should not have to remake their accounts every year.
REQ-7	5	Voters should be allowed to vote in given time frames based on their location.
REQ-8	3	Voters have to verify their submission.
REQ-9	4	Voters are allowed to reset their password using your a valid State ID, Driving Permit, or State Issued Driving License.

Nonfunctional Requirements

FURPS Requirements

Functionality - Every individual account should be secured with a user's SSN and license ID and can optionally have extra biometric security measures at the time of voting. Furthermore, the application should allow users to choose their candidate, either by the provided candidates or by write-in.

Usability - Everything visible to the user should be shown in a manner that comes off as clear and concise to avoid any errors from occuring. Multiple confirmations will be given when a user has cast their vote so that the user has reassurance that their data has been stored for the correct candidate.

Reliability - The app should be thoroughly tested before being released to the public to fix any major errors. Any account may take a minute to update once a vote has been casted (taken into consideration how large scale the app could be), however, updating an account should not take longer than two to three minutes at the network's busiest to ensure everything is completed in a timely manner. Furthermore, the application should securely transport the data to other devices without fear of tampering from third-parties in order to provide accurate results and secure data.

Performance - The app itself should not be very tasking on hardware. Excellent internet speed for the user should not necessarily be needed since all people logged in by a certain time will receive the ability to vote and all users should be allowed to vote regardless of the quality of internet connection.

Supportability - For the purpose of completing this within time constraints, the Vote4Me app will be made to work on Android devices, which allows us to provide one uniform app that can be presented. Moveover, a service phone number can be created to assist users with issues they encounter as well when using the app.

Vital Requirements

The most important requirement that must be met would be reliability. The system must be able to be used by a large amount of users without any crashing or slow downs that hinder functionality. Without reliability, such a product would not be viable in its intended setting since it would be unrealistic to authorize this system in a real election with no assurance that it would be successful and secure.

Equally as important of a requirement is functionality due to the fact that registered voters should be able to vote without issue on the app and they should be able to vote for any candidate they want to without issue. Ensuring functionality is paramount to ensuring our application could be used in future elections with the utmost confidence that the users are registered voters that can vote for whomever they please.

High Priority Requirements

Usability is a very important requirement due to how much of an impact usability has on the average user. Although not much time or effort needs to go into visual design, it is important that there be no confusion when it comes to how the user navigates the system. A user should be able to navigate the system knowing the meaning behind what they select with no confusion. With this in mind, usability is a high priority because it is a goal that can be easily achieved and is highly important to making sure users have confidence in the application.

Medium Priority Requirements

Supportability is of medium priority because in the long-term, all devices would eventually be supported, however, major phones can be targeted to ensure higher efficiency when comparing workload to coverage. In other words, as long as we ensure more common devices like Samsung Galaxys, LG G4s, or Google Pixels are supported, we can ensure that we cover the majority of the population with little changes to the application.

Lower Prioritized Requirements

Performance is an important aspect, however the speed requirements that would be set on the application are not very strict (nor do they need to be). In terms of service, internet is required, but in terms of extremely fast internet, this is unnecessary since updating a user's account should not take too much data.

Requirement Tests

1. A user should be able to create an account with their state ID and SSN from the initial download of the application in no more than 4 button taps and 5 form fields.
2. After login, the user should be prompted to select which election they are voting in, and should be able to select the election they are eligible to vote in within one tap.

3. A user should be able to submit any non write-in votes within 5 taps after election selection.
4. A user should be able to reset their password in 4 button taps and 5 form fields.
5. A write-in ballot submission page should be accessible after selecting an election within 1 tap. Then, a write-in ballot should be able to be submitted and a confirmation notification should be received by the user after no more than 3 taps.
6. Only 1 megabyte of data should be transmitted from the user's phone to the blockchain from a vote.
7. A user should be able to see which candidate they have voted for in 1 button tap.

User Interface Requirements

When the user opens the app, they will be taken to the login page. On the login page, they will have the option to either login with the account they have already created or create a new account. If they choose to create a new account, they will be taken to a form to create an account. The user must register to vote before using the app. We will check the information against the database of registered voters and create an account. If they are not found in the database, there will be a pop up notifying them. If they forgot the password, the user will be asked to enter their SSN # and the identification number they used. If it is correct and matched the username of the account, they will be allowed to create a new password. Once either of the forms are submitted, the user will be taken back to the login page to login with the new credentials. The flow of this user experience can be seen in figure 1 below.

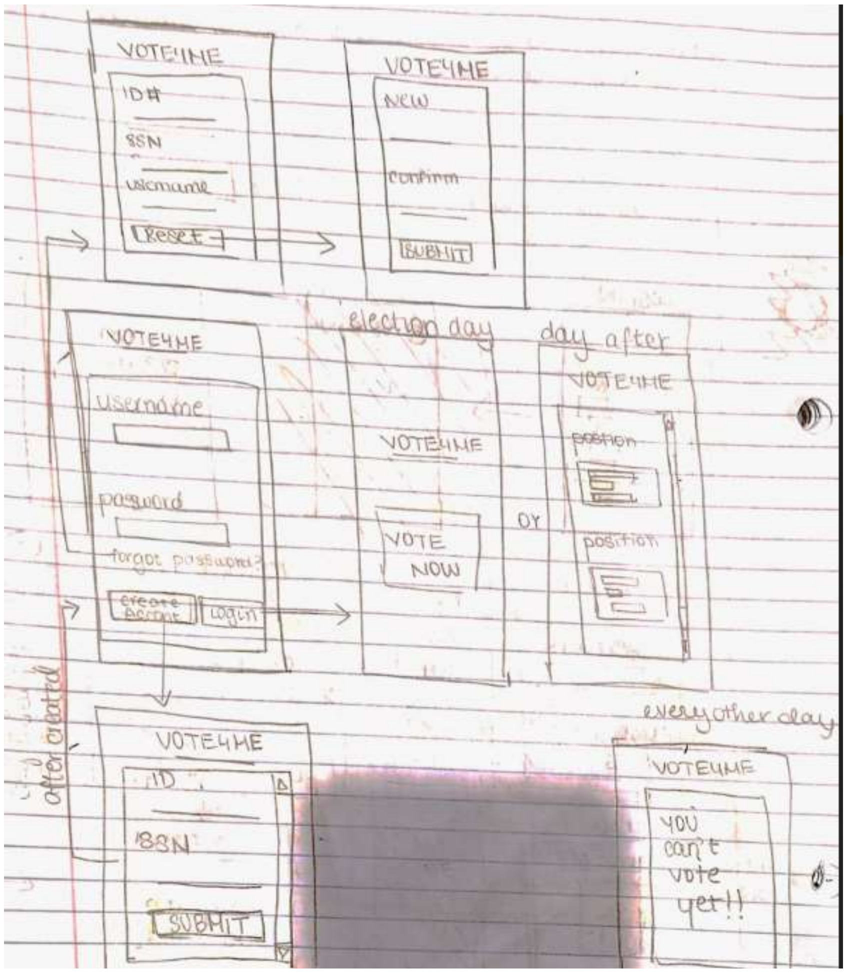


Figure 1

Once the user clicks on the vote button, they will be taken to another form which allows them to pick an option or enter their own choice by clicking the "other" option. The "other" option will take them to another page where they can type in their choice and click submit. It will direct the user back to the original form. At the end of the form, there will be a "done" button. Before submitting the form, they will be asked to confirm their choices. At the end, they will be given the option to go back or confirm. If they confirm, they will be taken to a page saying "Thank you for voting". If they are not happy with the choices, they can go back and make changes. The flow of this user experience can be seen in figure 2 below.

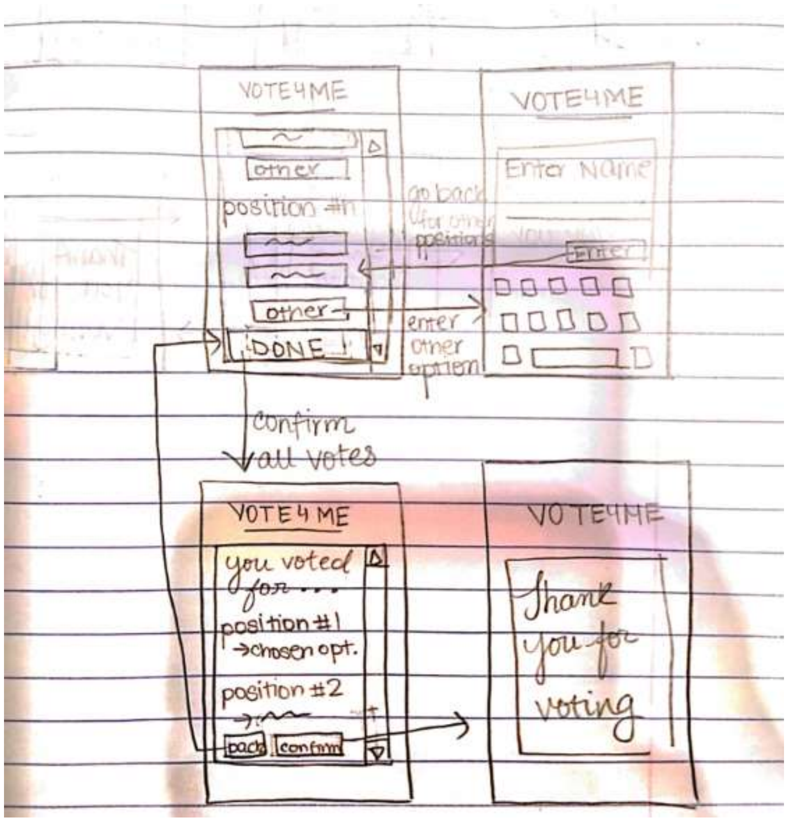
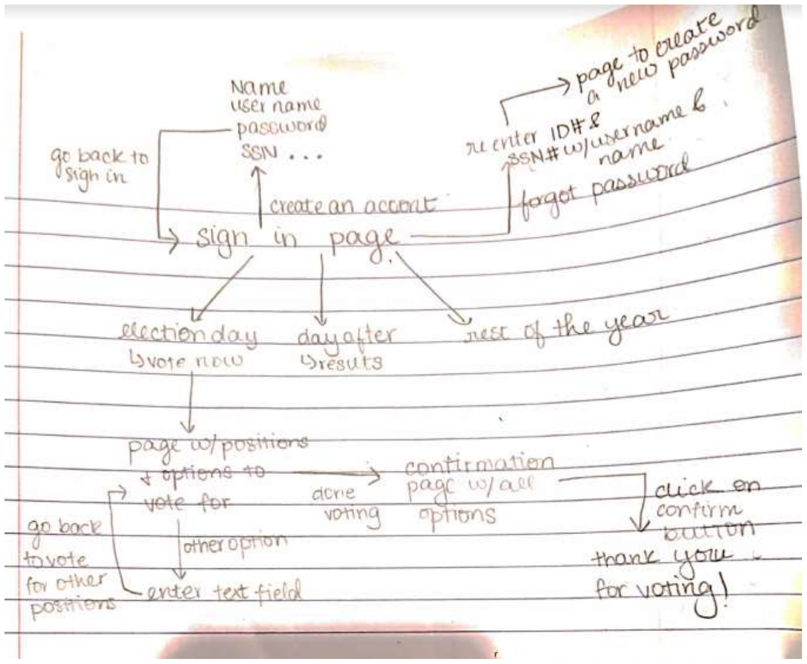


Figure 2

The full flow of the application can be seen in figures 3a and 3b on the next page



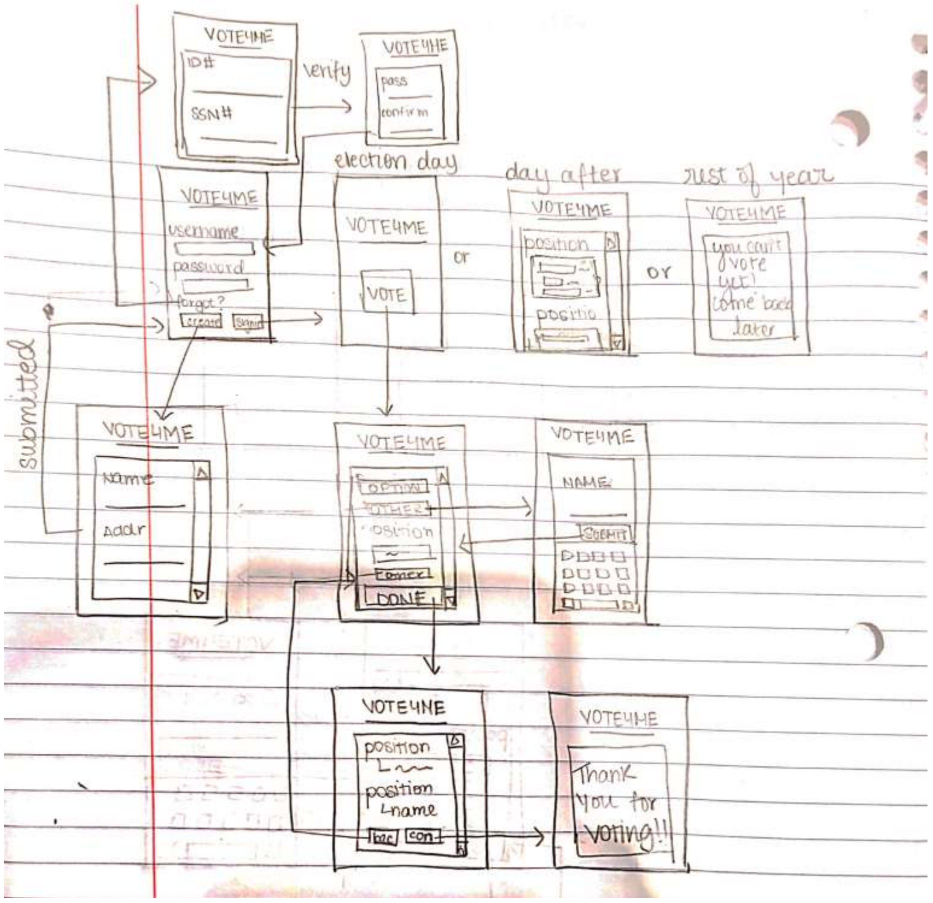


Figure 3a(top) and 3b(bottom)

Functional Requirements Specification

Stakeholders

The main stakeholders in this application are nearly all parties traditionally involved in the voting process. With the exception of any forces involved in the physical/in-person handling of current election tallying, all those parties would hold a stake in ensuring the application is used as intended and is an effective solution to the problem. Below, we have highlighted four major stakeholders that would utilize this process more than any other party.

Voter:

The first major stakeholder would be the average citizen. The average voter is the most important stakeholder since this category will make up a large majority of stakeholders, and the system only succeeds if the average user uses the system, meaning that their needs must be kept in mind. These needs mainly consisting of:

- Ease of access
- Understandable user controls/directions
- Registering for Vote4Me
- Resetting a password for Vote4Me

Federal Election Commission:

Another major stakeholder in the Vote4Me application is the Federal Election Commission (FEC). Unlike the average citizen, the Federal Election Commision would not necessarily hold much interest in the aspect of how easy the system is to use. Instead, the FEC would hold more interest in being able to easily view the tally of the votes when a given election is over. As another interested party, the FEC would mainly be concerned with:

- Ensuring voter security and security in the vote tally
- Easy view of final vote tally

Major Political Parties:

Other stakeholders include the major political parties. Similar to the FEC, the major political parties would be interested in the integrity of the system, along with ensuring that the display of the candidates is done in a fair, easy, and intuitive fashion to allow for fair representation of their candidate. In short, they would be interested in:

- Ensuring voter tally security
- Ensuring equal and fair representation in UI

News Organizations:

The final major stakeholder is news organizations. As they are often the distributors of the final election results to the masses, they would want easy access to the results when they are released. Moreover, they would want to ensure that these results are conveyed to them at exactly the same time as their competitors and they would want these results to be accurate to ensure that they are able to release the news in the most efficient manner. This means that the interests of the news organizations are:

- Ensuring voter tally authenticity
- Easy view of final vote tally

Actors and Goals

The actors involved with the Vote 4 Me application are similarly related to the stakeholders. This is in large part due to the fact that most stakeholders will have some form of interaction with the application. Below listed are the actors and how they interact with the system as a means to achieve some goal.

The User

The main initiating actor in the system would be the user. The main goals that the user would have consist of:

1.) Voting for a candidate that the system already recognized as a valid candidate. In this case, the system should clearly guide the user through their login and allow the user to see all the elections that they can vote in along with the candidates in those elections. Then, the system should allow the user to select different candidates they wish to vote for based on the given election. Once the user has voted, a confirmation should be received.

2.) Voting for a write-in candidate. This goal would be achieved similarly to the first goal, except once the write-in option is selected under the given election, the user would be prompted to write in their own candidate.

3.) The user being able to reset their password. This option should allow a user to use a previously verified source as a means to reset their login credentials. This would involve providing their SSN and legal state ID.

4.) The user being able to create an account on the Vote4Me app. The user should have the option to do so as soon as the app is opened and should be an option that is displayed to the user along with an intuitive process on how to proceed with account creation.

The Federal Election Commision

Another initiating actor would be the Federal Election Commission (FEC) with the following goal:

The goal of the FEC would be recieving the tally of all the votes that were cast on the system. In this case, it would be on the system to provide a tally to the FEC trying to access this information.

<u>Political Parties and Observing Organizations</u>

The main participating actor would be the political parties and other observing organizations (such as new outlets). This is due to the fact that these actors would mostly be concerned with the fact that all actions concerning any given election should be viewed as fair to everyone regardless of political affiliation. In this respect, political parties can be viewed as an offstage actor since they should just be given information that shows everything is being handled fairly. An example of this would be showing the layout of the selection screen beforehand to make sure that there are no objections concerning displaying information in an unfair manner. Furthermore, all these participating actors should be given the results of the elections at the exact same time to increase the actors trust in the results.

<u>Plan of Action Chart</u>

The below chart lists all the actors stated above and a list of goals they may want to achieve while using the system.

Actor	**Actor's Goal**
User	Be able to view all candidates and be able to pick a candidate from the list provided.
User	To write a candidate into the system and vote for the written in candidate.
User	The user should be able to reset their password using SSN and state ID.
User	To be able to create a Vote 4 Me account.
FEC	Access the tallied votes in an organized and timely manner.
Political Parties (and other interested parties)	Should be able to view the app's layout beforehand to see that everyone is being represented equally and fairly.

Use Cases

Casual Description

UC-x	Descriptions
UC-1	Registration - Allows the Guest to register for the blockchain voting platform and become a Voter. Extension Point: Voter may need to change their account password. Derived from requirements: REQ-1, REQ-2, REQ-5, REQ-6
UC-2	Vote - Allows the Voter to cast their vote via broadcasting it onto a blockchain network on Election Days. Extension Point: Vote needs to be verified prior to casting and after. Voter may cast a write in vote. Derived from requirement: REQ-5, REQ-7
UC-3	VerifyVote - Vote must be verified before casting, cannot be cast more than once and cannot be changed after. <<include>>UC-2 Derived from requirement: REQ-3, REQ-4, REQ-5, REQ-8
UC-4	ResetPassword - Voter might have a need to reset their password if they forget login credentials. <<extends>>UC-1 Derived from requirement: REQ-9 *FUTURE IMPLEMENTATION* because this is not necessary for our system to function properly, and is an additional feature for user's ease of use, we have decided to wait on implementing this.
UC-5	WriteIn - Voter select a write in candidate. <<extends>>UC-2 Derived from requirement: REQ-5, REQ-7

Diagram

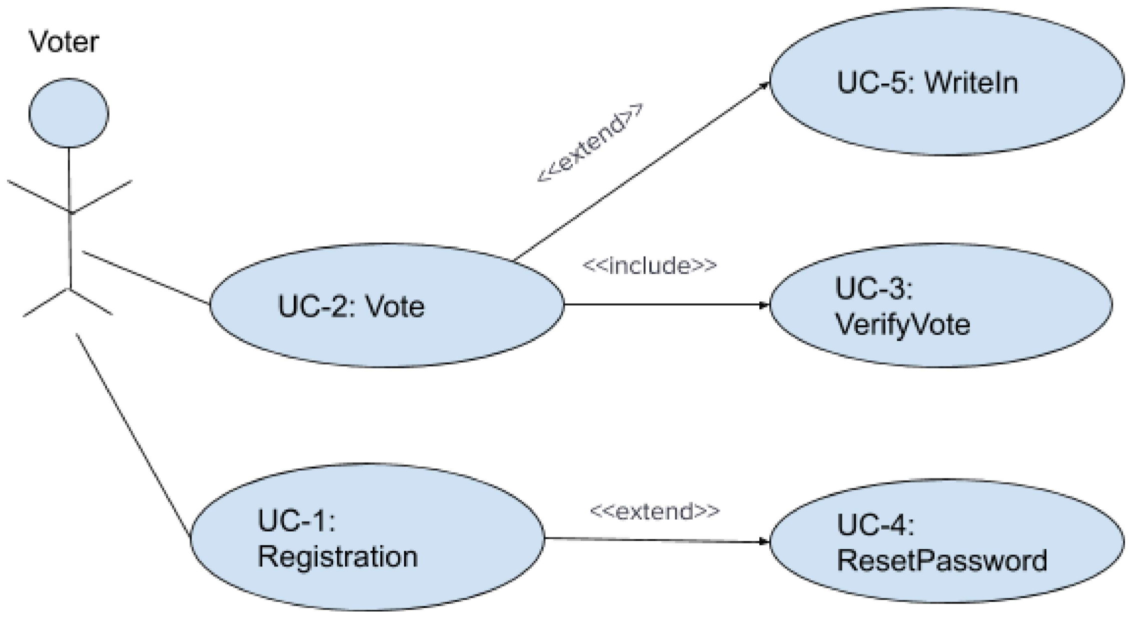


Figure 4

With this diagram is the concept that the user will be able to vote and register, as well an implementation of the ability to write in a vote, verify the vote, and reset the password.

Some things of note:

- The ability to register will consume quite a bit of time to implement
- Write vote implementation will give the voter complete functionality, but will be harder to interpret
- Voter verification is a very important aspect of the system, but could consume resources more heavily

Overall, there are trade offs to consider when implementing the above diagram, but each use case is essential to the full functionality of the software.

Full Description

Use Case UC-1: Registration		
Related Requirements:	REQ-1, REQ-2, REQ-5, REQ-6	
Initiating Actor:	Guest	
Actor's Goal:	Gain access to the voting platform	
Participating Actors:	System	
Preconditions:	● Guest has registered to vote in their state ● Guest has a valid: ○ State ID ○ Driving Permit ○ State Issued Driving License	
Success End Condition:	● Guest gains new privileges and becomes a Voter after information is provided	
Failed End Condition:	● Guests information is not valid, registration is cancelled.	
Extension Points:	*ResetPassword* (UC-4)	
Flow of Events for Main Success Scenario:		
←	1.	**SYSTEM** displays the option to login or create an account.
→	2.	**GUEST** selects to create an account.
←	3.	**SYSTEM** requests the following info from the **GUEST:** ● Legal Name (First, Last) ● DOB ● Address ● State Issued ID Number (DL, ID, etc...)
→	4.	**GUEST** provides the information requested.
←	5.	**SYSTEM** verifies the information provided and either: a) Confirms the information is correct and grants the user access, allowing the **GUEST** to become a **VOTER**. b) Denies the validity of the information and denies the user registration with the supplied

		credentials.

Use Case UC-2: Vote		
Related Requirements:	REQ-5, REQ-7	
Initiating Actor:	Voter	
Actor's Goal:	Cast their vote in a secure manner	
Participating Actors:	System	
Preconditions:	● Voter has registered to use the voting platform	
Success End Condition:	● System broadcasts the Voter's vote ○ Assumes vote has been verified	
Failed End Condition:	● Voter cancels their vote selection or denies confirmation on their end.	
Extension Points:	*VerifyVote* (UC-3)	
Flow of Events for Main Success Scenario:		
←	1.	**SYSTEM** displays the options to vote for candidates
→	2.	**VOTER** selects their candidates and clicks *Submit* to proceed. <<See UC-5 for WriteIn>>
←	3.	**SYSTEM** presents selections and requests the **VOTER** to press *Confirm* in order to cast their vote
→	4.	**GUEST** confirms vote.
←	5.	**SYSTEM** broadcasts vote onto the blockchain

System Sequence Diagrams

The 2 most important use cases for our design are use case 1 (registration) and use case 3 (Verify Vote), as evidenced by our traceability matrix. Using UML each of these use cases are laid out.

Use Case 1 (Registration):

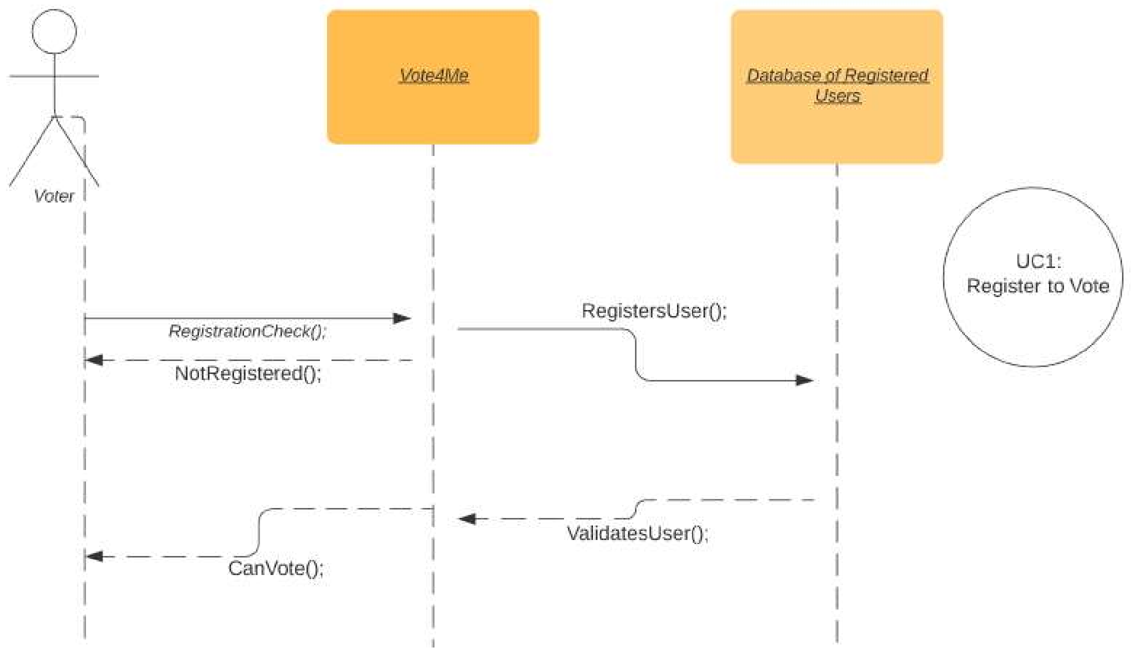


Figure 5

Use Case 3 (VerifyVote):

What makes up the CastsVote() function in this System Sequence Diagram is the object of what we are trying to accomplish, which is making sure that the votes are held securely, counted only once, and cannot be tampered with by an outside source. All of this has to do with the blockchain network we will implement.

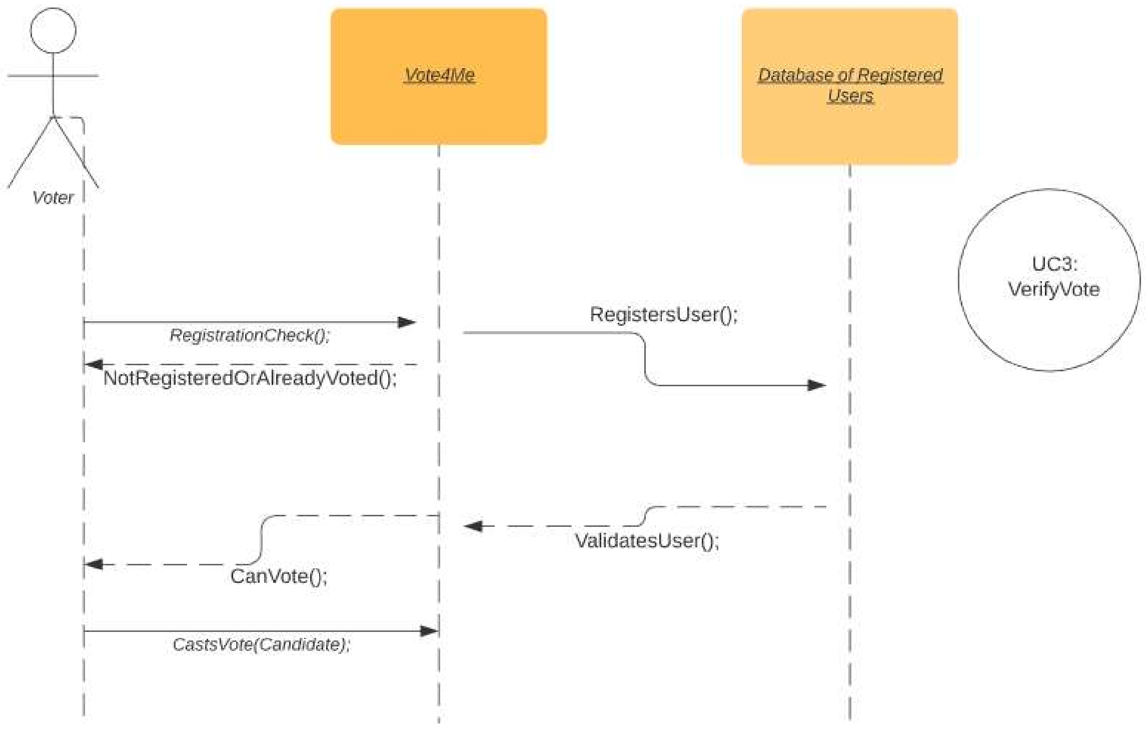


Figure 5

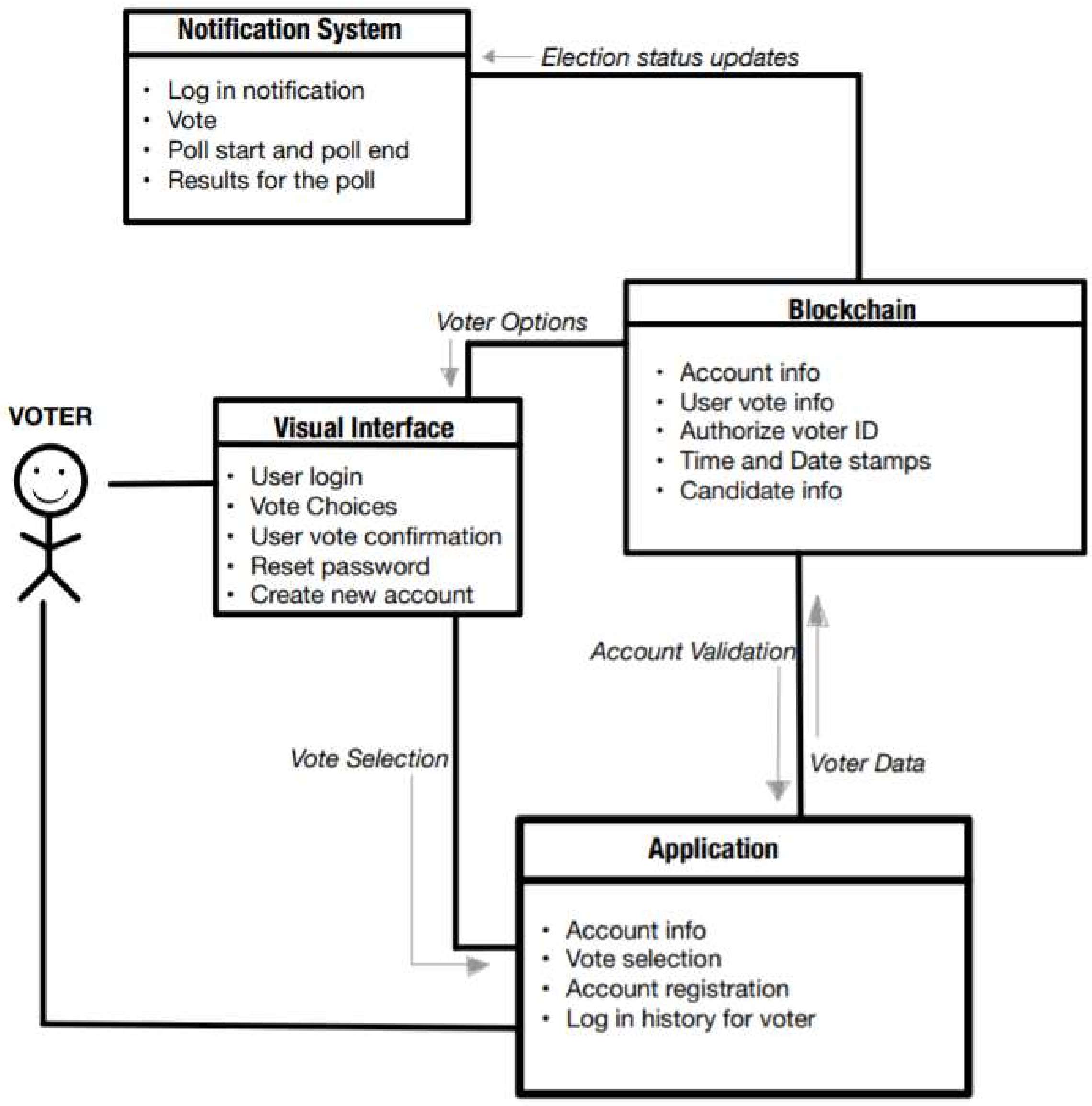
Effort Estimation Points for Use Cases

UUCP = 5 + 1 + 3(2) = 12

Use Case	Description	Effort Estimation Points
UC-1: Vote	The user voting use case takes 2-5 clicks. One to select the candidate that is being voted for (possibly a couple more clicks if there are multiple positions being voted for), and one click to submit the vote	5- The user has to log-in and select multiple options (for potential multiple positions) and submit their vote. This comparatively takes more steps and effort than other steps so it is rated higher.
UC-2: Verify Vote	The user will receive a verification after they vote confirming that they have voted.	1- The user must click one button after voting to show that they have verified their vote. This is a small amount of effort on the user.
UC-3: Reset Password	A user that needs to reset their password, will be shown the option to select in one click, and will be prompted to enter a new password to change the password.	3- The user will be prompted with the option to reset their password and will have to type in a new password to change their current one.
UC-4: Write-in	A User can select to do a write-in ballot from the select candidate screen. This will prompt them to write in a candidate.	3 - When voting, the user will have to write in a new candidate. This takes moderate effort from the user since they are manually entering their candidate.

Domain Analysis

Domain Model



By looking at the use cases and requirements of the application, we were able to create this domain model. One of the most important Use Cases is vote (UC-1). The key boundary concept of this system is visual interface and notification system, which serves as the main way people will interact with the application on the day of the election and while voting with their account.

Internal concepts includes the blockchain, and the application. When the customer first opens the app and are registered, they will be able to login into the app. If they have not registered, they must register before using the application. Once the user is logged in, they are allowed to vote. They must enter the vote twice (UC 3) to verify the vote before they cast it. This information is then sent to the blockchain. The user has direct access to the application and can open it anytime. They will however, only have access to the voting form on the day of the election. After election day, the visual interface uses the information from the blockchain to show the user the results of the election. The notification system will alert the user when they the polls start and end. It will also notify them when the results of the elections are available. Through the notification system, the user will not miss election day or its results.

System Operation Contracts

Contract CO1:

Operation:	RegistrationCheck()
Cross References:	Use Cases 1 & 3: Vote & VerifyVote
Preconditions:	A user is trying to vote for a candidate in Vote4me
Postconditions:	- A user has either been sent to either: - the registration page to make sure that they are eligible voters - the vote page so that they can cast their vote

Contract CO2:

Operation:	RegistersUser()
Cross References:	Use Cases 1 & 3: Vote & VerifyVote
Preconditions:	A user is trying to vote for a candidate in Vote4me
Postconditions:	- A user was able to be registered to vote

Contract CO3:

Operation:	NotRegisteredOrAlreadyVoted();
Cross References:	Use Case 3: VerifyVote

Preconditions:	A user is trying to vote for a candidate in Vote4me but their vote is not counted
Postconditions:	- A user has seen WHY their vote wasn't counted (if they have yet to register, or if they have already voted)

Contract CO4:

Operation:	CastsVote()
Cross References:	Use Case3: VerifyVote
Preconditions:	A user has voted for a candidate in Vote4me app
Postconditions:	- A user was able to view their vote, and can see a confirmation and thank you message for voting

Contract CO5:

Operation:	ValidatesUser()
Cross References:	Use Cases 1 & 3: Vote & VerifyVote
Preconditions:	A user is trying to vote for a candidate in Vote4me, but need to register first
Postconditions:	- A user was registered in the database for being able to vote, and they can now cast their vote

Traceability Matrix

Use Case	PW	Notification System	Visual Interface	Database	Application
UC1	4	X	X	X	X
UC2	4	X	X	X	X
UC3	1			X	

UC4	2		X	X	
UC5	3	X	X	X	

Interaction Diagrams

UC1 - Registration

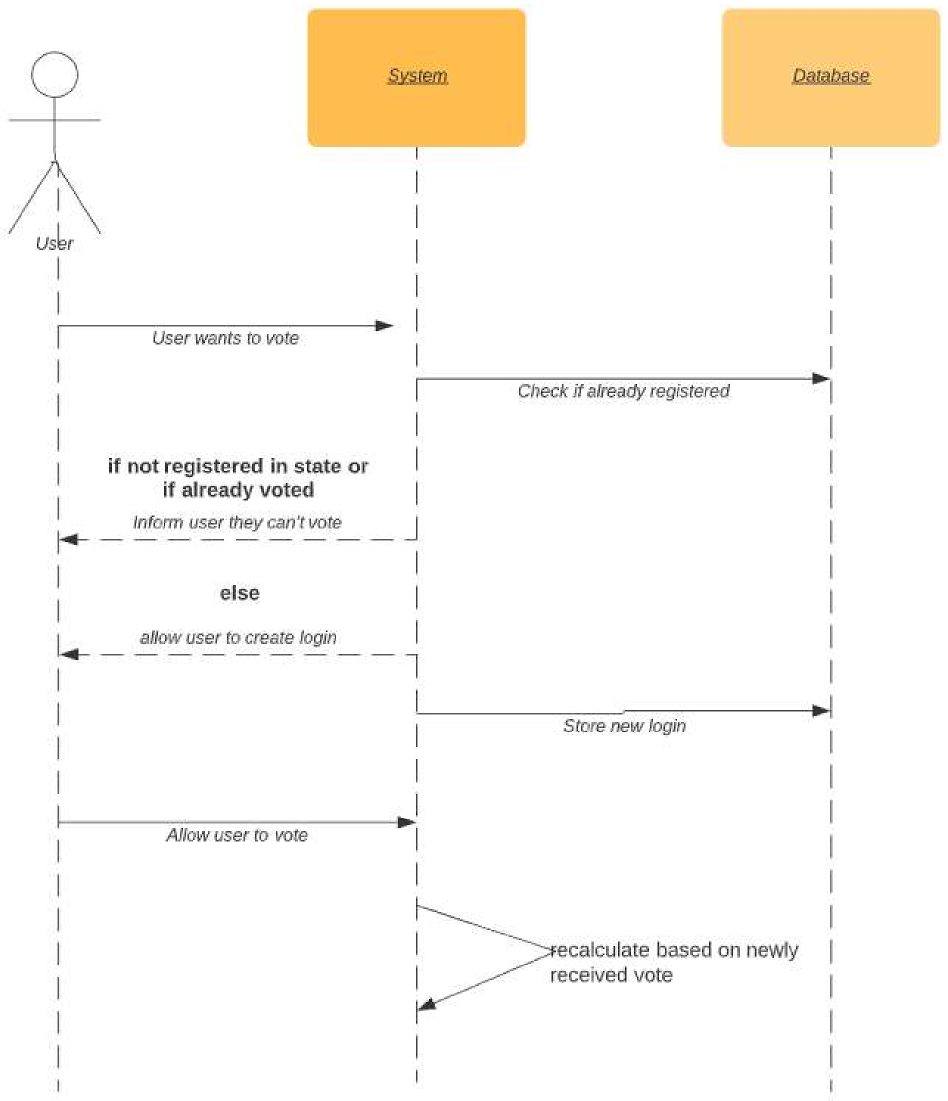

System
Database
User
User wants to vote
Check if already registered
if not registered in state or if already voted
Inform user they can't vote
else
allow user to create login
Store new login
Allow user to vote
recalculate based on newly received vote

Figure 1

In Figure 1, we are following a user through setting up their account for Vote4me system. When a user first logs in to the app, it will verify the user's identity, and check if they have already pre-registered to vote. If they have registered to vote, they will be forced to create a login and password, so that their vote is secure and can not be accessed by someone on the outside. If the user has not registered to vote, they will be notified that they must do so before they are able to vote. Once the user has created their login credentials, our database stores this login, and also stores the fact that they have voted, once their vote has been cast. Our system recalculates who is in the lead and by how much on every vote.

UC2 - Vote

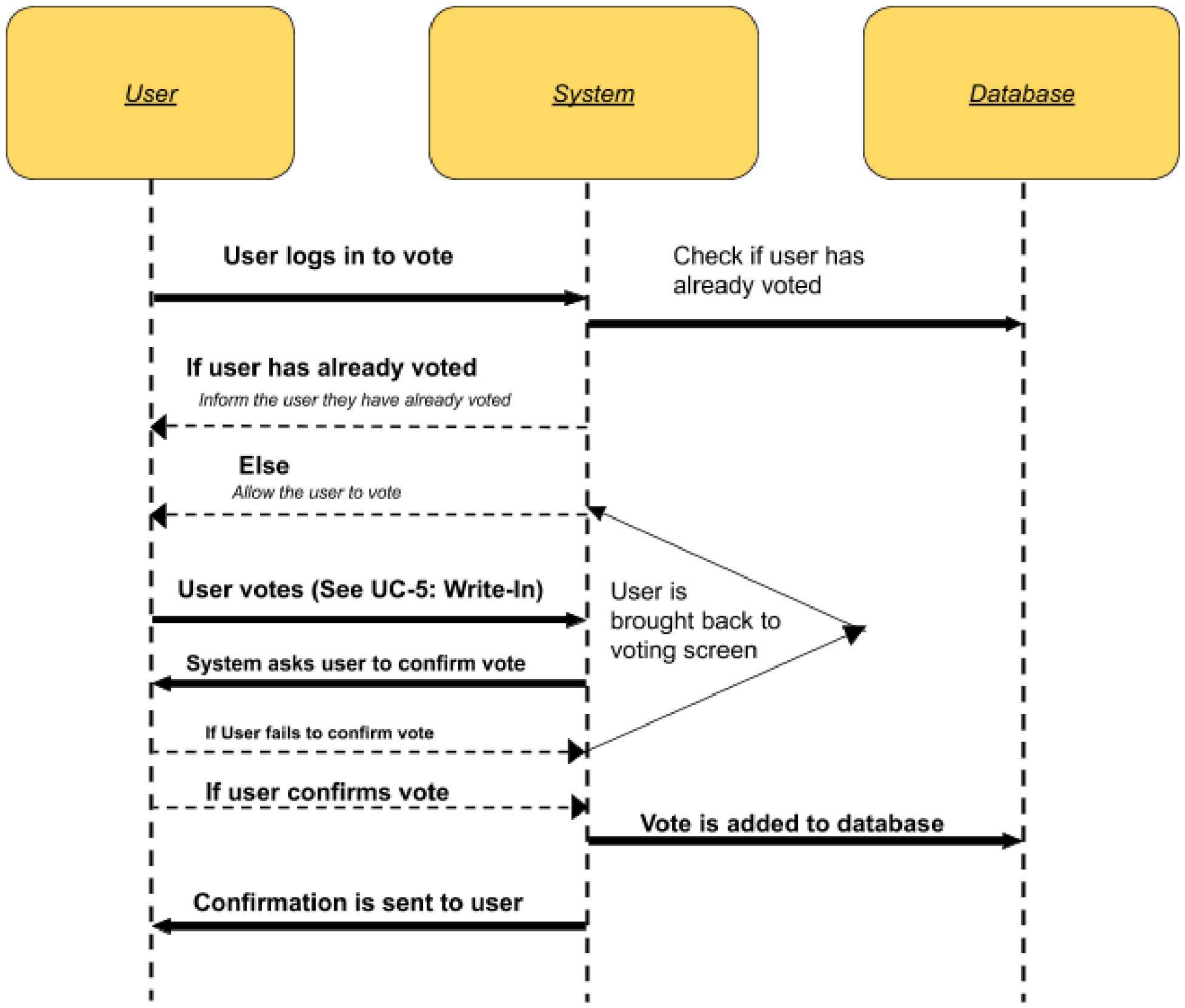

User
System
Database
User logs in to vote
Check if user has already voted
If user has already voted
Inform the user they have already voted
Else
Allow the user to vote
User votes (See UC-5: Write-In)
User is brought back to voting screen
System asks user to confirm vote
If User fails to confirm vote
If user confirms vote
Vote is added to database
Confirmation is sent to user

Figure 2

In Figure 2, we walk through a user attempting to vote on the application (without Write-In ballots). When the user enters the application, they are prompted to log in. Once they enter their credentials, the system checks the database to ensure that the credentials are correct and that the user has not already voted. If both cases occur, then the user is given the ability to vote, otherwise, the user will not be allowed to vote on the system. See Use Case 5 for Write-In option. Once the user has completed their selections, the system will ask the user to confirm their choices. If the user fails to confirm the vote, then the user is brought back to the voting screen. If the user confirms the vote, the user’s vote is added to the database and a confirmation message is sent to the user.

Class Diagram and Interface Specification

Class Diagram

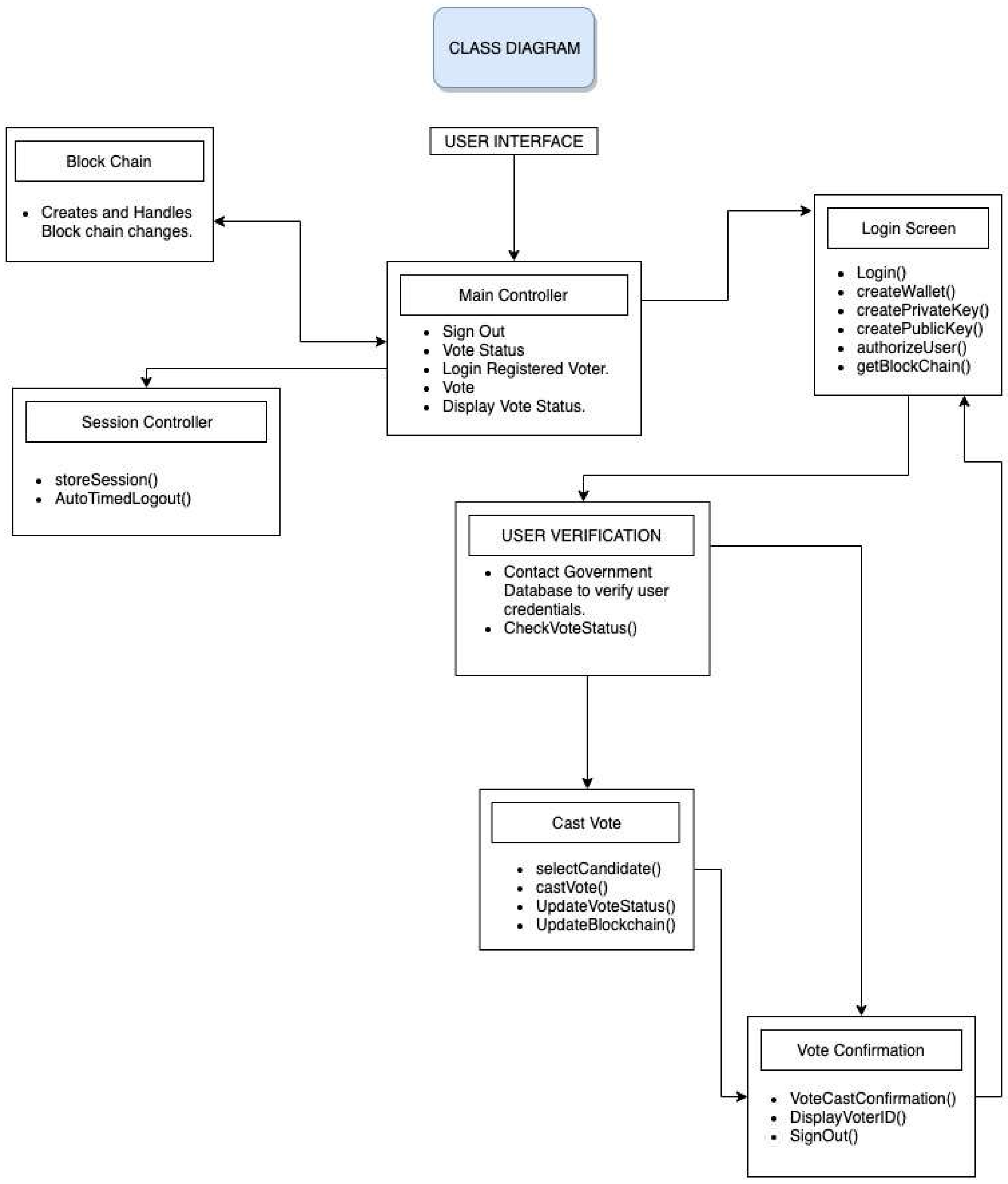


Figure 3

Data Types and Operation Signatures

1. Login Screen:
    a. Attributes:
        - String SSN
        - String License Number
        - String alternate ID
    b. Operation:
        - authorizeUser(): given user credentials, verify the legitimacy of the voter. If this fails, the user is brought back to the same login screen.
        - getBlockChain(): get the most recent/common copy of the block chain and save it.
        - createWallet(), createPrivateKey(), and createPublicKey(): are all used for identification, management and organization of the blocks inside the block chain.
2. User Verification:
    a. Attributes:
        - String SSN
        - String License Number
        - String alternate ID
    b. Operation:
        - CheckVoteStatus() checks whether or not the user has already voted after verifying their identity. If yes, then the app continues to the Vote Confirmation page that tells the user that their vote has been cast already, otherwise they move forward to Cast Vote.
3. Cast Vote
    a. Attributes:
        - String Candidate Select
    b. Operations:
        - selectCandidate(): Updates the selected candidates value when the user selects the candidate. Only candidates name will appear on screen.
        - castVote(): submits the vote to the blockchain.
        - UpdateVoteStatus(): saves the fact that the user already voted to prevent duplicate or multiple votes from the same user.
        - UpdateBlockchain(): called on by cast vote, updates the block chain with the new vote.
4. Vote Confirmation
    a. Attributes:

- NONE

b. Operations:
   - VoteConfirmation(): gives a confirmation that the user had voted and the vote us saved in the block chain.
   - DisplayVoterID(): shows the voters identification number.
   - SignOut(): saves the session and exits.

Traceability Matrix

		CLASSES			
Domain Concepts	**PW**	**Sign Out**	**Vote Status**	**Login/Register**	**Vote Cast**
Cast a Vote	4	X	X	X	X
Try to Re-Vote/change Vote	3	X	X	X	
Login With Wrong Information	2			X	
Confirm Voting	3	X	X	X	

System Architecture and System Design

Architectural Styles

In short, the architectural styles of a project displays how the code is organized and set up in order to achieve the end result. The architectural styles are the highest level of organization of the project, as it touches on the high level modules of project, and how those modules interact. There are multiple examples of architectural styles, many of which are used in our Vote4me application. In our particular application, in order to best organize things, we plan on incorporating java documents on each of the modules to ensure everything is explained properly. The types of architectural styles used, and how they are used in this project is outlined below:

1. Layered style - we have layered our application into multiple levels (the UI, backend made strictly of blockchain, and peer to peer network to prevent tampering with votes, and other attempts at malicious activity)
2. Peer to Peer style - we have our peer to peer network, which checks the blockchain of each individual user, and validates it with each other, and whichever results make up the majority, and match each other, makes up the “correct” chain. Having each of these chains to communicate requires peer to peer communication
3. Client Server style - we have our users functioning as the client, obviously as they are sending their votes out, however, they are also functioning as a server, since they are taking information (the other blockchains) and checking for congruency.
4. Model View Controller style - we are splitting up our application into 3 parts, the model (which holds the core functionality and data), the view (which is our UI), and the controller (which is made up of UI and other method calls to cast votes).

Identifying Subsystems

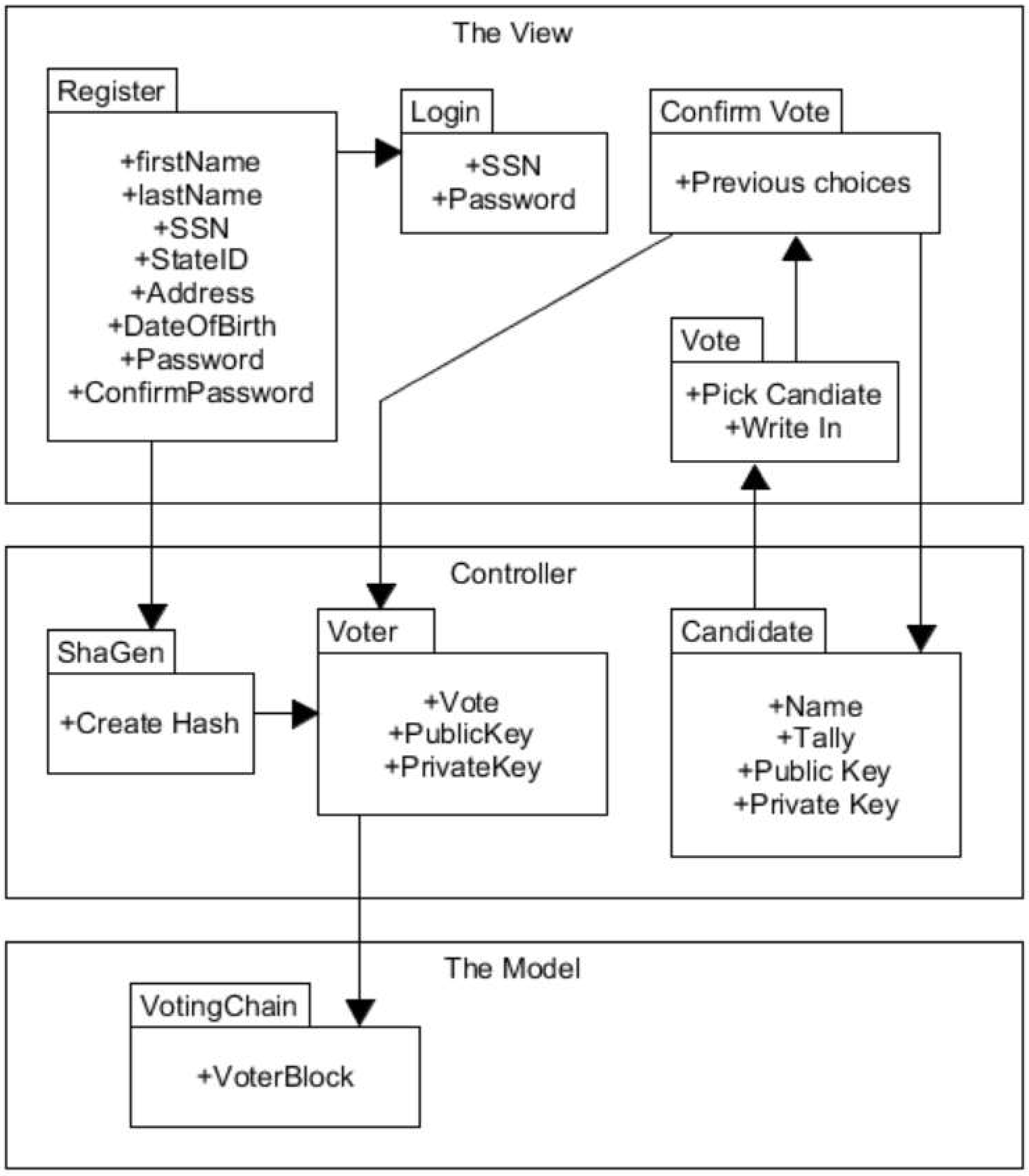
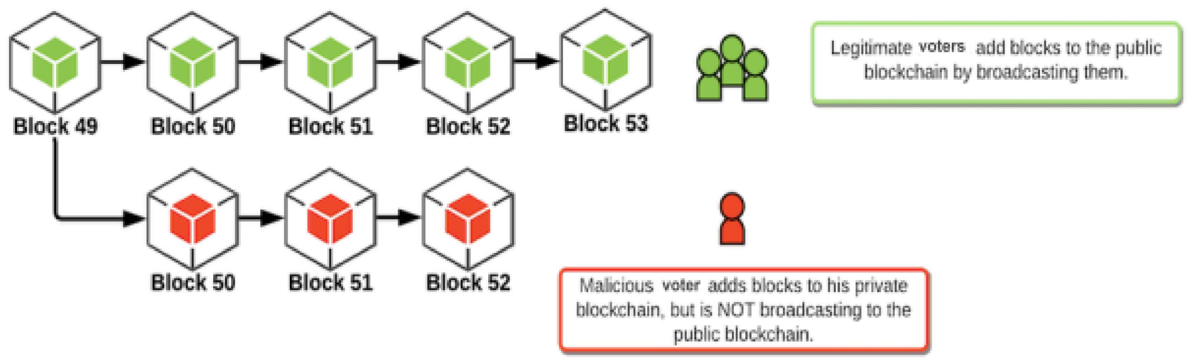
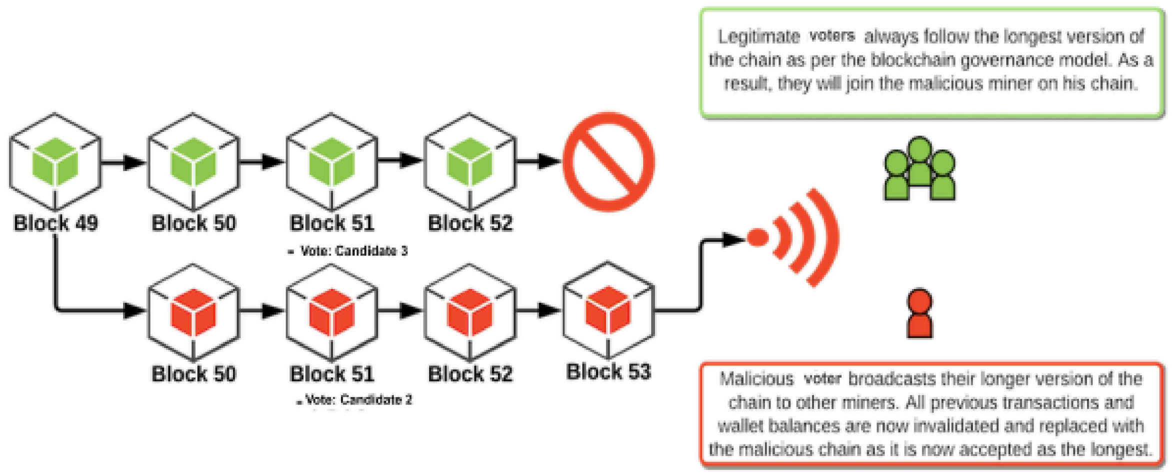


Figure 4

Mapping Subsystems to Hardware

Our system utilizes a peer-to-peer network to communicate to multiple devices. This is necessary to persist data storage across devices but is also crucial to our security structure. Our application requires each user be broadcasting onto the same chain from the beginning to verify that votes have not been manipulated. This decision of which is the valid chain is made by a consensus of peers, or nodes, residing on the network. Each node is asked to present its current chain and whichever chain is the majority, becomes the valid chain. Our blockchain network utilizes this network to mitigate 51% attacks on the voting system. In which a rogue agent may attempt to overtake the current blockchain by broadcasting a chain with invalid or manipulated information. This form of attack is described below:


Block 49
Block 50
Block 51
Block 52
Block 53
Legitimate voters add blocks to the public blockchain by broadcasting them.
Block 50
Block 51
Block 52
Malicious voter adds blocks to his private blockchain, but is NOT broadcasting to the public blockchain.


Legitimate voters always follow the longest version of the chain as per the blockchain governance model. As a result, they will join the malicious miner on his chain.
Block 49
Block 50
Block 51
- Vote: Candidate 3
Block 52
Block 50
Block 51
- Vote: Candidate 2
Block 52
Block 53
Malicious voter broadcasts their longer version of the chain to other miners. All previous transactions and wallet balances are now invalidated and replaced with the malicious chain as it is now accepted as the longest.

The issue with this form of attack is that it also requires the malicious actor own at least 51% of the nodes on the network to be able to repeat the malicious chain. Otherwise it will be overwritten by the valid chain.

Our peer-to-peer network requires that each device or instance of the application be connected to the network act as both a server and a client. This is that data can be sent back and forth between peers and they all have the same permissions on the network. If the data were to be stored in a centralized server, that could be subject to manipulation from inside attacks and does not allow for full transparent access to where the votes are going and confirmed they have not been changed in any sense. To have each device achieve the functionality of server and client, we will be utilizing Java Sockets to create instances of both to be able to transport data appropriately.

While the majority of data that will be sent across this network will be the blockchain structure itself. We also need to maintain persistent information for all the users that have connected to be able to issue them ballots at the start of an election period. We also need to maintain a list of all the active users so that the application can know which blockchains are the most maintained at a given moment and to broadcast transaction to those active users when a valid vote is placed. This information will be sent as well with the blockchain. The application will always be listening for data being sent up until a vote is broadcast to keep the chain as up-to-date as possible.

Persistent Data Storage

For our system, there needs to be a means to store the blockchain, and certain user data. We would do this to ensure that the blockchain maintains its persistency across user sessions. Moreover, since the blockchain is crucial for verifying votes, it is essential that it persists to allow the function of the peer to peer network. Additionally, certain user data must also be stored, for example whether the user has voted or not. We will maintain a profile of certain essential data of the user such as their password and username. All of this will be implemented via serialization of the data. The blockchain is stored via an arraylist, and can be written to a text file. This can be read in through the program to call the data, whenever it is needed. Similarly, the data of each user will be stored in a hash table, stored in 2D arraylist. That way the username is used as the key and any values of a user that need to be stored, such as whether or not a user has voted, or their password can be accessed. The implementation of each storage medium can perform a read or write quickly and efficiently. As we expect to handle a large number of users, quick access to each account will be crucial and a hashtable will give us suitable performance for such operations. In regards to the security of the user information, it will be stored in such a format that it is not readable by people. This is do to the serialization process within Java, which writes object to the file in such a format that is not readable. Thus the security of the system is ensured. The system does not use a database, and instead uses the blockchain for the blocks as well as ensuring the userid and password is a match.

Network Protocol

For the application, we will need to use a peer-to-peer network using Java sockets as a means of transferring data. This is to allow the updating of the valid blockchain on each device to ensure that every device connected is operating correctly. Using this implementation, we can prevent adversaries from adding in extra ballots or changing casted votes by ensuring that transactions on the network consist of only one block that is added to the blockchain with a 51% majority on all devices. We have opted to use Java sockets for our network since our application is not entirely written in Java and does not have a database, which many of the alternatives require. This being said, the data that will be sent over the socket will be the information that is needed to create a new block on the list of transactions. Specifically, we will need to send:

- A ballot number: Signifies who the user is voting for.
- A hash: Contains a hash of the user’s public key. This is done so that the user could potentially confirm that their vote is what they intended it to be.
- A timestamp: Shows what time the user updated the blockchain and also is used to help confirm what the latest version of the blockchain is (this is important for security).
- A prevHash: The hash of the previous block is needed to make sure all data added to the blockchain is correct and consistent.
- User and candidate strings: Allows clarity in terms of who cast the vote and who the vote was cast for.

On the peer to peer network each user, after casting their vote, will have their data processed by the system (explained in more detail in the concurrency section), and the data that is read will be added to the blockchain and to the “wallets” of the candidates that are in the election.

Another primary use for Java sockets in this application will be for keeping track of votes that a candidate receives in their candidate wallet. This is done by taking the information from a block that will be added to the blockchain (this contains the user’s vote), finding the corresponding “candidate wallet” (which contains a “tokenID” for the candidate) and updating that candidate’s tally by incrementing it by one. This will be done for a candidate every time a block is added that shows that a user has voted for them.

Global Control Flow

Execution Order: The Vote4Me application is, for the most part, procedure driven, with certain functions being event driven. This means that most, if not all, users will have the same end goal and experience when using the software, but some small differences may be seen depending on the user. One example of this is registration, as most users will log into the system with their intention being to vote. However, if a user has not registered previously, they will be required to do so prior to voting. Moreover, every voter follows the same set of steps in order to submit their vote, but the process differs when the user is submitting a write-in vote. A user’s experience will be slightly different in this case than if they were voting for a pre-registered (i.e: party nominee) candidate. That being said, the majority of the actions the user will take will follow a linear progression that only deviates slightly given certain use cases.

Time Dependency: The Vote4Me application is a system that is very contingent on timing. In order to be able to use this system for any government election, strict time constraints must be in place to ensure a legal election. For instance, for a national election, voters using the Vote4Me software will only be able to vote during the designated election day and only during valid times of the day. Voting must open exactly when announced and must close exactly when announced for legality and fairness. Moreover, a future implementation of the application could also have a time-out feature that logs a user out when no actions have been taken after a certain period of time. This would be done as an added security measure.

Concurrency: In the Vote4Me application, every user acts as a thread when broadcasting a vote. This is needed for organization and accuracy when tallying votes, since multiple users voting concurrently will cause issues with critical sections of data. Specifically, this is important when calculating the vote tally or updating the blockchain, since multiple users accessing that data concurrently would lead to votes being lost or the blockchain being invalidated. Due to this, when users vote, they will have to wait in a queue until the critical sections of data are free to use. This will be accomplished through the use of semaphores, which are used with threads to "lock" parts of code that utilize critical data so that only one thread can manipulate that critical data at any given time. In this situation, we would not have issues with deadlock, since the procedure for voting is almost identical for each user and would not rely on any users who have voted after the user currently using the critical data.

Hardware Requirements

Vote4Me is an application that can be downloaded on any android device that runs on IceCreamSandwich (API 15) or later. This is because this API supports 100% of all Android systems without degrading our performance. Moreover, the device's GPS technology and network connection must be on and in working condition for the application to operate. Due to the standardization of devices due to them running on an Android OS, no specific screen resolution requirements are truly necessary. Finally, in order to store the entire blockchain and the application, it is best that we allocate at least 2 gigabytes of storage on each phone for the duration of the election, which can be deleted after the election results have been confirmed.

Data Structures and Algorithms

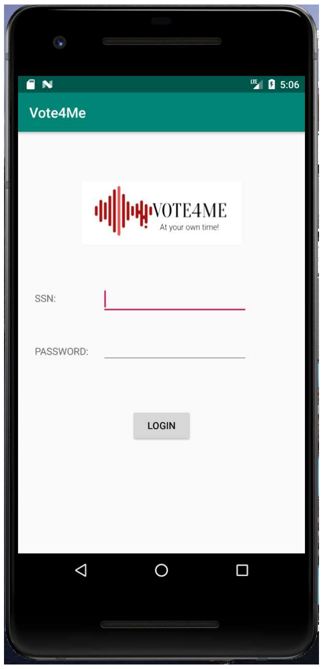
Algorithms

Our system utilizes the java.security package for 2 distinct algorithms to ensure that data is being stored privately and securely. We utilize the cryptographic hash function SHA256 and the RSA encryption algorithm in various ways throughout our program. In order to generate our private and public key pair per user, we call on the security package to generate an RSA keypair with a seed. This seed is based on the user's information and a chosen password. The keypair object is constructed of a private key and public key which we can convert into a string of length 64 for easy manipulation. The private key is used to generate the wallet while the public key is used to write transactions from. The benefit of using this java library is that it allows for easy access to signature functions to sign the blocks broadcast per individual private key. We then primarily utilize SHA256 to generate hashes for the blocks in our blockchain. Each block requires 2 hashes to be held within it: its own reference id hash and the id hash of the block prior to it in the chain. We make these simply by taking all the data held in the block and putting it through a function we made to generate the SHA256 hash. This was chosen because it allows for discrete information to be held publicly. By the nature of the hashing function, the only way to retrieve the data is by knowing what you're looking for, therefore any voter can quickly look up their vote on the chain if they have the proper information that will collide with a hash on the chain. These algorithms allow for us to securely store data on our blockchain and within our application.

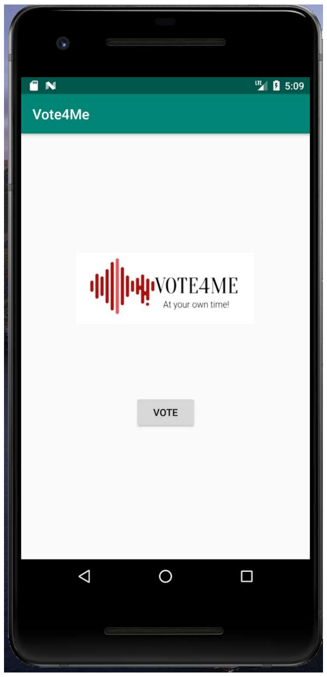
Data Structures

The data structures we used in this system are unique. First, we create an object for a block, which contains values like the private key and public key, etc. We also have an object that constitutes as our wallets, which each of the voters have, as well as the candidates, having admin wallets. These objects are very useful in voting, since they determine who voted, and are very small in terms of space efficiency. For the blockchain, which records the collection of blocks, we are using an arraylist. This is a very useful structure for this situation, since we don't have to create a static sized structure like an array, but we can also access any element in O(1) time if we know which one of the values we are searching for. This space efficient structure is useful since it has roughly only the amount of necessary blocks in the chain, and is runtime efficient since we can access a given element whenever we need to do so. We originally planned on using a linked list, where each block was a node, but transitioned into using an arraylist to improve our runtime efficiency. Adding an element is also an O(1) operation in an arraylist, since in the event that the arraylist has been capped, and the arraylist size needs to be doubled, it will be done in O(1) fashion, so clearly an arraylist is the best option to implement the blocks.

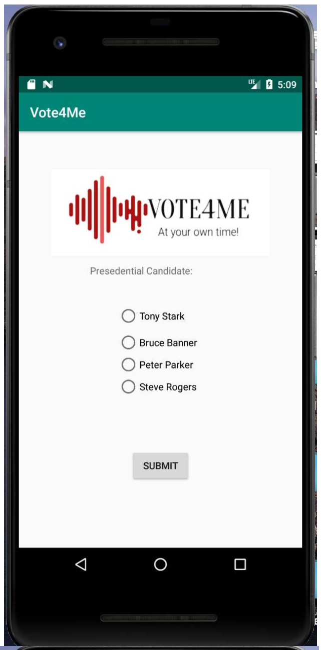
User Interface Design and Implementation



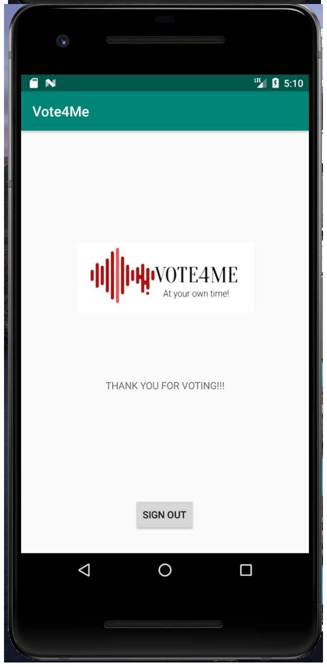
LOGIN: For the login screen, the user interface has a simple, easy to understand login page where user can enter their credentials. The final implementation of the application will take specific identification information that can ideally be validated using a government database. This is different from our initial set up because instead of saving user information on our servers, by making users create an account, they now fill out a form that takes in user data and verifies it against government database to confirm that the voter is allowed to vote. The user information that is being input in the application that is paired with all information on the government database, is the user's custom ID and password.



Vote: This is the screen seen by the voter after they login. The page simply contains a "vote" button that the voter clicks once they are ready to vote. Once the user clicks the "Vote", they are taken to the next screen, depending on weather or not voting time has commenced. This is also easy to use because the UI is designed keeping simplicity in mind and only includes one button that the user can use.



Candidate Selection: The vote ballot is where the voter selects their desired candidate. There haven't been any changes made to our design since Demo 1.



Vote Confirmation/Sign Out Page: This page also maintains the initial design, where the user is shown their voting confirmation along with the option to sign out. They can not vote again. If the user logs in again after voting, they will be automatically be shown this screen, showing they voted already (this also shows the confirmation number).

Overall, our UI has only changed slightly since the initial design. We no longer will be creating an account for each user, instead use their Identification to validate them. This means we also no longer have the Forgot/Reset Password page. Lowering the overall complexity of the UI, hence increasing the ease of use.

Design of Tests

For this demo, we wanted to test out two main use cases: Voting and Login. We believe that these are the two most important functions of our application and are at the core of our project. Therefore, we wanted to be able to complete this portion of the application before working on other aspects of the application. For integration, we designed tests to check if they were added to the blockchain once the user was done voting. In addition, we designed tests to test the functionality, usability, reliability, and performance of the application. We also tested the implementation of the SHA generation algorithm.

Unit Tests

UC: Vote - Users should be able to cast a vote on election day

1. To Vote: To test this use case, you have to be logged into the application. If the user hasn't voted yet, they should be provided with the option to vote. The user must be able to pick the candidate from the options provided and cast their vote.
2. Count multiple votes: Validate that new votes are being counted correctly. We tested this by casting votes from multiple user accounts and checking if they were casted by logging into the admin account to view the blockchain.

UC: VerifyVote- The user should be able to login if they haven't voted yet

1. User Accounts before voting: To test this, we logged into the user accounts using the credentials we had previously assigned to them.
2. User Accounts after voting: Once the user had voted, we tested this again to make sure that the user cannot log back in if they had already voted. This was done by entering the credentials again after the user had voted.
3. Admin account: To test this, we logged into the admin account with the preset username and password. After logging, the admin should be able to view all the votes that have been cast.

UC: Login - Users are able to login on election day to perform various tasks

1. User accounts can ONLY login when an election has been started (specified by the admin account) and will be unable to login to cast votes after the election has ended.
2. User accounts can only login if the combination of their username and password are valid and that is checked in our backend (stored in a blockchain to ensure security, instead of a less secure database that could be maliciously adjusted.

Integration Testing

We will use a bottoms up approach when conducting integration testing. Each use case will be combined with another use case, to test the compatibility and functionality of the two. Once we have tested the compatibility thoroughly, we determine that the tests have passed. We will test other groups, until each group of use cases has also passed. At this phase, we will move to the next subset of groups when larger groups are tested for compatibility. Ultimately, this will eventually result in the entire system being integrated and tested, until we reach the system testing phase. The convenience of this method of testing is that it allows us to validate each subgroup, and ensure that there is a careful and methodical process of testing. Additionally, it will be far easier to identify problems in subgroups of software, which will speed the process of software deployment. With that said, the only difficulty that will be encountered, is that each group will require a driver to run tests. So as groups get more and more complex, it will require larger and larger drivers. This is a calculated payoff, as the gain from ensuring that each subgroup is well tested is far more important than having to create additional software to test the code. As for how the drivers will be created, each driver will have various edge cases that will test the compatibility of the software. Running through various scenarios that will stress test each group. Furthermore, as larger groups are formulated, more complex interactions will be simulated.

Testing of other Requirements

Functionality: The strategy to test this requirement will be as follows once the software implementations are in place. The SSN login will be tested to ensure that the software is correctly storing the user data under a variety of conditions. Additionally, the software will also be tested to ensure that the user voting input is correctly submitted especially when a write-in vote is entered.

Usability: The strategy to test this requirement will be as follows: Each phase of the login and voter entry process will be ensured that the correct confirmation, and user feedback appears. Additionally, a variety of scenarios mimicking users will be generated to ensure that all members of our target audience are able to use the software.

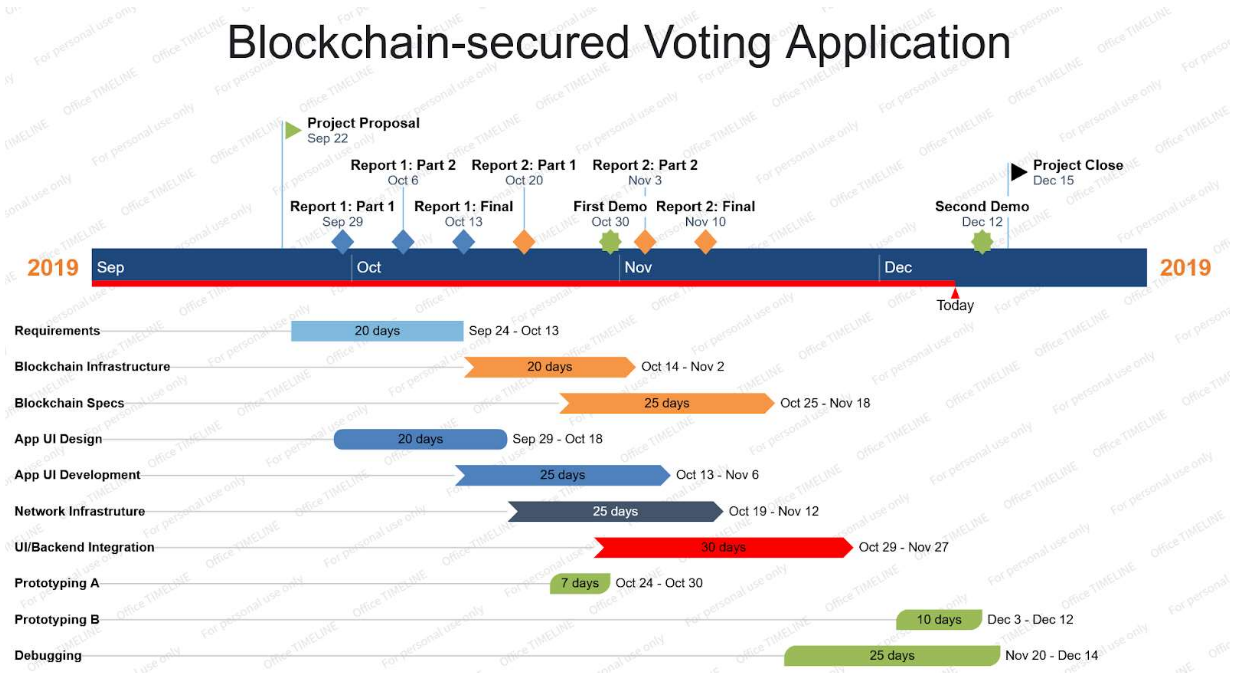
Reliability: The strategy to test this requirement will be as follows, as our goal is to ensure that the user is able to use and receive confirmation of the vote, we will stress test the software under a variety of conditions of network usage to assess and improve the capability of the network, so that the user will have a satisfying experience on election day.

Performance: The strategy to test this requirement will be as follows, due to the user-friendly nature of our software, we do not anticipate difficult hardware loads, but nonetheless the software will be tested for such cases. Namely, the cases where a large number of users may interact with a single user via the peer-to-peer network.

SHA Generation Algorithm: The strategy to test this algorithm will be as follows, as the generation algorithm is crucial to the performance of the blockchain and the overall system, this was one of the first things to be tested. Our strategy was to enter in a variety of different seeds, across different times, to ensure that SHA was generating correctly.

User Interface: Our user interface will be tested in the following manner: we want to ensure that the interactions between the user and the blockchain are functioning properly and are performing the proper tasks specified by each button/click in the backend for our blockchain.

History of Work



Legend:

▭ = **Requirements Phase**

❯ = **Development Phase**

▢ = **Design Phase**

▭ = **Testing Phase**

As per the gantt chart, we were able to stick to the deadline shown above. Currently, we are working debugging several aspects of the code and prototyping our application for the second demo. Mostly solving bugs in the integration code to get it ready for the presentation in a few days. The plans and deadlines set in Report 1 are consistent with the ones shown above. From then we have successfully designed and developed a blockchain infrastructure, the peer to peer network, and a user interface. Going into Report 2, we worked mostly on the UI and Backend integration and testing.

The division of work originally fell into three categories, based on the structure of the application. These three groups were the design and implementation of the Blockchain structure, the development of the Peer to Peer Network, and finally the User Interface design and Front End implementation. This structure changed throughout the implementation as rather than subdividing our group into 3 and tackling each section, we decided to all work together in implementing the blockchain back-end all together, followed by integration, and then the peer-to-peer network. Intermittently we had to debug and prototype a working app for our demos, thus we gave ourselves plenty of time to do so leading up into the second demo.

Key Accomplishments

- Blockchain
    - Design
    - Implementation
    - Data Persistence
- UI
    - Design
    - Integration
- Peer-to-peer Network
    - Design
    - Communication
- Android
    - Launch APK

Future Implementations

- Due to the time constraint and lack of man-power, we've included a list of features we wish we could implement but were not able to:

* Candidate Information
* iOS integration
* Website
* Ability to hold custom elections (school, commercial, private, etc...)

References

[1] “Register to Vote.” Welcome to the State of New York, 7 Aug. 2019, https://www.ny.gov/services/register-vote.

[2] “Official Site of The State of New Jersey.” NJ DOS - Division of Elections - Register to Vote!, https://www.state.nj.us/state/elections/voter-registration.shtml.

[3] “What Is Blockchain Technology? A Step-by-step Guide For Beginners” Ameer Rosic-Blockgeeks-Marko https://blockgeeks.com/guides/what-is-blockchain-technology/#comments

[4] Lucidchart. (2019). *Online Diagram Software & Visual Solution | Lucidchart*. [online] Available at: http://lucidcharts.com [Accessed 6 Oct. 2019].

[5] Android Studio Tutorials (2018). *Android Studio Essential Training*

[6] 10 Common Software Architectural Patterns in a Nutshell. (2018). *Software Architectural Styles*. Vijni Mallawaarachi.

Furps Requirements websites:
https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/
http://www.cs.sjsu.edu/faculty/pearce/modules/lectures/ooa/requirements/IdentifyingURPS.htm

The above websites were used to gain a better understanding of FURPS and this understanding was used when writing the FURPS non-functional requirements. This helped present a clear high level view of what is to be expected of “Vote 4 Me” that allows those not working on the software to understand what this software intends to accomplish.

Voting requirement information (1 & 2):

https://www.ny.gov/services/register-vote

https://www.state.nj.us/state/elections/voter-registration.shtml

The above websites were used as a means to gain a deeper understanding of the different requirements that must be met when a person wants to register to vote in the United States. Although some rules may vary depending on the state, these sources give a clear outline of the requirements that must be met to vote in these states. These requirements were always taken into consideration when considering features of our software since making sure a voter is permitted to vote is an important aspect of our software. The different state rules were often analyzed by comparing two different states. In this case, New Jersey and New York could be compared from the links given above (although several more sources could be checked for different locations).

Block Chain (3):

https://blockgeeks.com/guides/what-is-blockchain-technology/#comments

This source gave a comprehensive guide explaining what a blockchain is and how it can be utilized to create a secure system to hold a user's data. This information is the foundation of the software that we are creating since voter security is a current issue that many people find themselves concerned with, and we are creating a software with not only the goal of easy accessibility but also with the goal of achieving voter security.

ER Diagrams (4):

https://www.lucidchart.com/documents#docs?folder_id=home&browser=icon&sort=saved-desc

This source was used to create diagrams that display how the system reacts to new changes, and how the system compares information when certain functions are activated. In the case of "Vote 4 Me" one relationship that an ER diagram will display that will constantly be used, would be one that updates the user's account to contain who they voted for. Similar to the FURPS analysis, this is done so that someone can easily understand what is to be accomplished.

Android Studio (5):
https://www.lynda.com/Android-Studio-tutorials/Android-Studio-Essential-Training/677172-2.html

This source was useful in helping us understand the Android Studio and Intellij IDE. We used Android Studio in order to implement our system, and incorporated java in order to get the code to work and function properly. Our backend blockchain was implemented with java, and the frontend was implemented with the support of XML and java.

Software Architectural Styles (6):

towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

This source was very imperative in coming up with the architectural styles that are in our system. There were many examples in our system, that are used in our system, and this source was instrumental in helping understand the different types that are *possible* to be implemented in any given system.