Ballot Casting Design Document

Section 1 - Project Description

1.1 Project
Ballot Casting

1.2 Description
Our goal is to create a system in which ballots are securely and privately cast, verifying that they are eligible to be counted, ready to be sent off to tabulation.

1.3 Revision History

Date	Comment	Author
11/9/2023	Initial Start of design document	Gabriel Sharbel
11/14/2023	Modification to Testing Plan and adding requirements	Gabriel Sharbel
11/16/2023	End of Sprint 2, finalization of work	Gabriel Sharbel

Contents

Section 2 - Overview

2.1 Purpose

This software system is designed to facilitate the electronic voting process. It encompasses various functionalities to ensure secure, accurate, and efficient casting of votes. This includes local, state, and federal elections. Some of our audience are as follows:

2.1.1 Election Officials

- The main audience would be those in charge of planning and supervising the electoral process. They would have to be familiar with the operating features, security protocols, and functionality of the system.

2.1.2 Government Regulators and Authorities

- Ensuring that the system satisfies legal and security benchmarks would be of importance to officials tasked with establishing electoral legislation, standards, and compliance criteria.

2.1.3 IT Professionals and Developers

- Those in charge of the software system's development, upkeep, and security would need in-depth technical knowledge of the system's code and architecture.

2.1.4 Political Representatives and Candidates:

- To maintain confidence in the election process, they may want to know how transparent and equitable the system is.

2.1.5 Voters

- Although they were not the system's main target audience when it was being developed, they are still crucial users. To assure their trust in the voting process, they may want to know about the system's usability, accessibility, and security features.

It is imperative to customize the material for every audience segment in order to effectively address their particular concerns and guarantee openness, usability, and security throughout the voting process.

2.2 Scope

2.2.1 Ballot Casting

- **Options for Voting:** Offer various methods for casting votes (e.g., electronic, paper). Confirmation: Allow voters to review and confirm their choices before finalizing the vote.
- **Transmission and Storage:** Security during Transmission: Ensure secure transmission of votes to a central database.
- **Data Integrity:** Implement safeguards to prevent data corruption or loss

2.2.2 Voting Process

- **Security:** Implement measures to prevent tampering, fraud, or unauthorized access. Privacy: Guarantee the confidentiality of each voter's choices.
- **Auditability:** Provide a system for verifying and auditing the results.

2.3 Project Roles

- Manager - Gabriel Sharbel
- Lead Programmer - Evan Hudson
- Secondary Programmer - Hunter Mosley
- Document Editor - Jacob Nguyen
- Tester - Paul (Trey) McKinney

Section 3 - Sprint 1

3.1 Development

3.1.1 Planning (Sprint 1)

- Propose a solution architecture
- Gather requirements
- Create GitHub code repository
- Select a language & toolchain

3.1.1.1 Deliverables

- Use Case Diagram
- Jira
- Team Slack

3.1.2 Implementation (Sprint 2)

- Create a design document
- Adhere to design principles
- Establish a test plan
- Begin MVP coding

3.1.2.1 Deliverables

- Design documentation (includes class
- & sequence diagrams)
- Testing acceptance plan
- Versioned, well-commented code

3.1.4 Testing (Sprint 3)

- Complete MVP coding
- Execute test plan
- Maintain the design document

3.1.4.1 Deliverables

- Design documentation (includes class
- & sequence diagrams)
- Testing acceptance plan
- Versioned, well-commented code

3.1.5 Deployment/Presentation

- Design Presentation
- Demonstration of code

3.1.5.1 Deliverables

- Design presentation slide deck (5 – 8 slides)

3.2 Solution Architecture

Develop a Python program that seamlessly facilitates the voting process for eligible users while ensuring the integrity and security of the ballots submitted. This sophisticated system not only presents a user-friendly interface for casting votes but also incorporates robust verification mechanisms to guarantee the accuracy and validity of each ballot. The meticulously designed program is engineered to streamline the voting experience, promoting efficiency and transparency. Once verified, the ballots are transmitted securely to the tabulation team, contributing to a reliable and seamless election process.

3.3 Requirements

Functional Requirements:

1. **User Authentication:** Verifying user eligibility for voting
2. **User Profile Management:** Users should be able to update their profiles, including personal information
3. **Privacy Settings:** All data should be protected

Non-Functional Requirements:

1. **Performance:** The system should respond to user actions within 2 seconds to provide a responsive user experience.

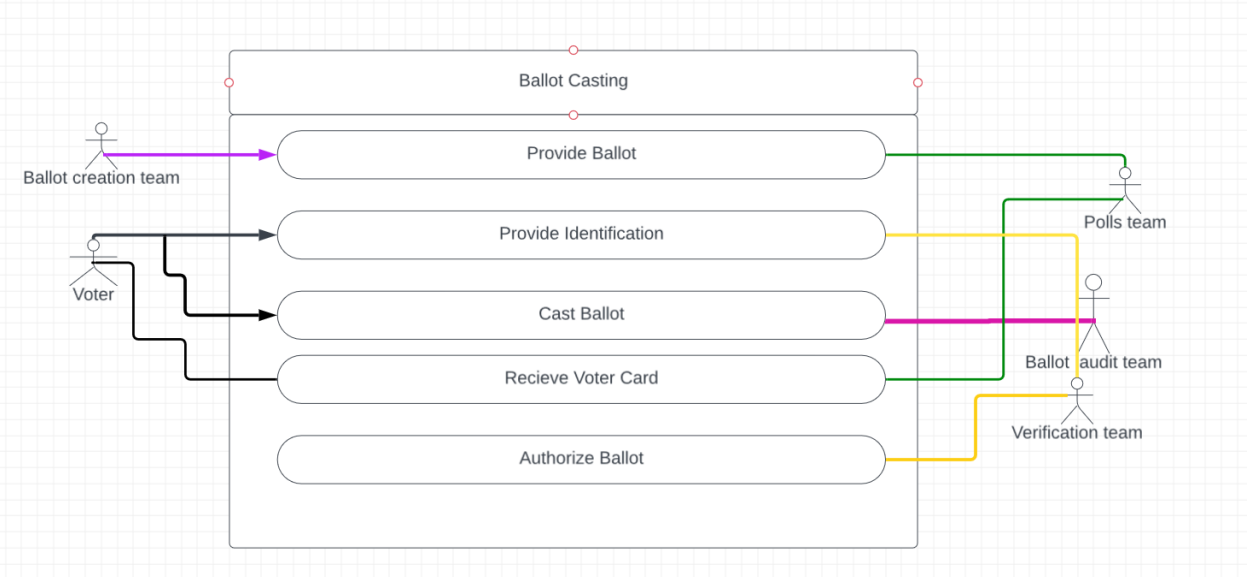
2. **Security:** User data and uploaded files should be securely stored and protected. Passwords should be hashed, and data should be encrypted during transmission.
3. **Scalability:** The system should be able to handle a growing number of users and content without a significant drop in performance.
4. **Usability:** The user interface should be intuitive and user-friendly, with clear navigation and visual design.
5. **Data Backup:** Regular backups of user data should be performed to prevent data loss.
6. **Regulatory Compliance:** The system should adhere to legal and regulatory requirements regarding user data privacy.
7. **Constraints:** Legally the voter's id can not be associated with the voter's license or identification.

3.4 Toolchain

- Python (Language)
- Jira
- Slack
- Github (https://github.com/GSharb/Scopecreepers7)
- Google Docs
- LucidCharts

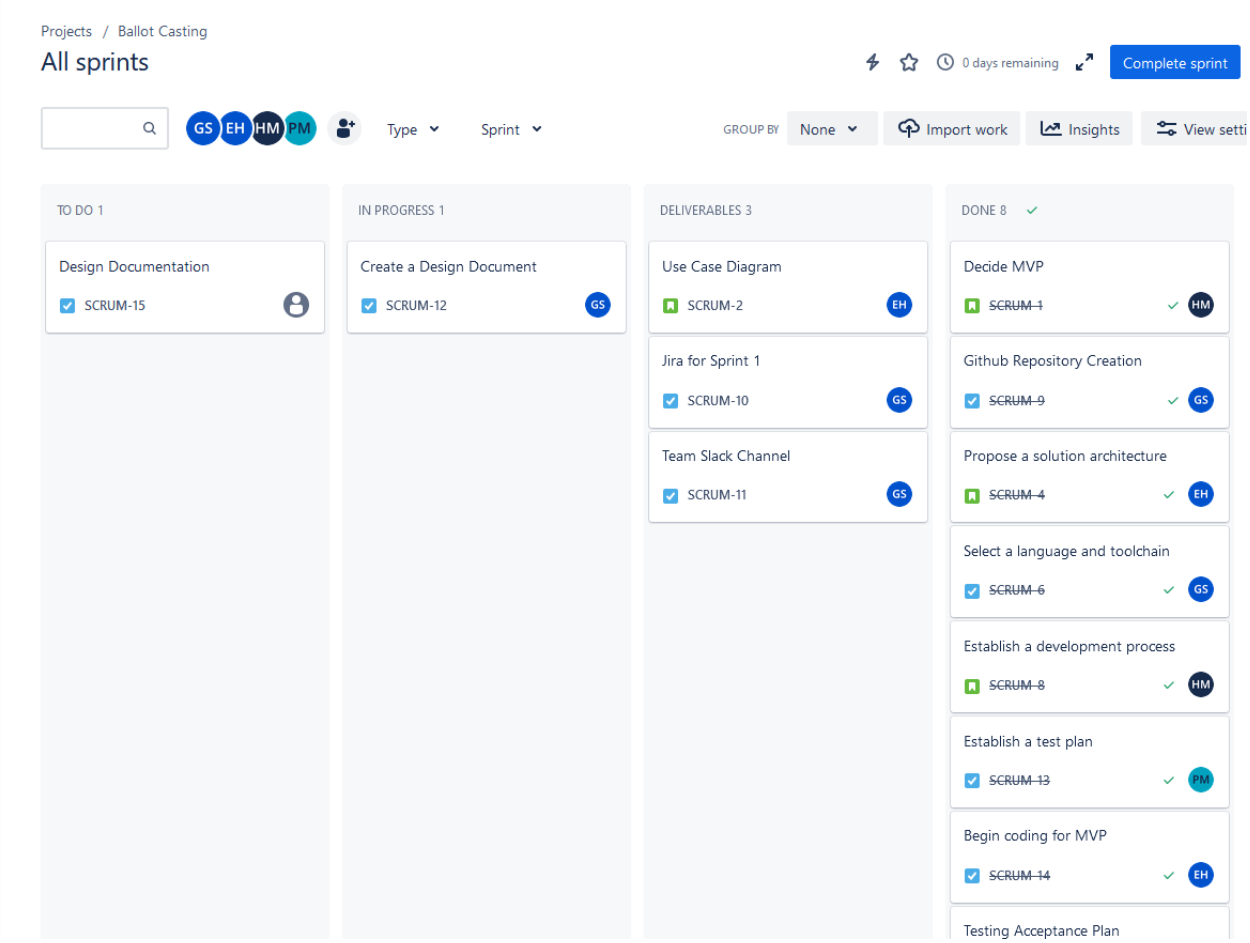
3.5 Deliverables

3.5.1 Use Case Diagram



3.5.2 Jira

- Link for those with access: https://scopecreepers7.atlassian.net/jira/software/projects/SCRUM/boards/1



3.5.3 Slack

Link: https://team7-scopecreepers.slack.com

Section 4 - Sprint 2

4.1 Implementation

- Create a design document
- Adhere to design principles
- Establish a test plan

- Begin MVP coding

4.1.1 Planned Deliverables

- Design documentation (includes class
- & sequence diagrams)
- Testing acceptance plan
- Versioned, well-commented code

4.2 Design Principles

4.2.1 Open/Closed Principle (OCP)

- The TabulateContext class demonstrates the Open/Closed Principle by being open for extension and closed for modification. When you introduce a new election type, you can extend the system by creating a new strategy class without modifying the existing TabulateContext class.
- The logic for handling different election types is encapsulated within separate strategy classes (FederalElectionStrategy, LocalElectionStrategy, StateElectionStrategy). If a new election type is introduced, you can create a new strategy class without modifying the existing classes.
- This approach adheres to the principle of allowing the system to be easily extended with new functionality (new election types) without modifying existing, stable code.
-

4.2.2 Strategy Pattern

- Strategy Interface (ElectionType): Defines the verify method, acting as a contract for different verification strategies.
- Concrete Strategy Classes (FederalElectionStrategy, LocalElectionStrategy, StateElectionStrategy): Implement the ElectionType interface, providing specific verification logic for federal, local, and state elections, respectively. These classes compose the abstract class verifyMechanism.
- Context Class (TabulateContext): Manages the selection of a verification strategy based on the election type provided in the vote object. It delegates the verification process to the selected strategy.
- Dynamic Strategy Selection: The context class dynamically selects the appropriate strategy at runtime based on the election type specified in the vote object.

4.2.3 Open for Extension, Closed for Modification:

- The Strategy Pattern allows easy extension for new election types. You can introduce new strategy classes without modifying the existing code in the context class.

4.2.4 Versioning Control

- Stubv1.0 - functions and classes with bridge pattern
- Jsonv1.0- functions and classes with bridge pattern implementation

- Stubv2.0- functions and classes with strategy pattern redone because of testing acceptance
- Jsonv2.0- functions and classes with strategy pattern implementation of verification logic and general tests for correct vote objects, wrong/missing keys in vote object/

4.3 Class Modules

4.3.1 Vote

Client class with the attributes for a vote object with the right keys.

4.3.2 TabulateContext

Manages the selection of a verification strategy based on the election type provided in the vote object. Delegates the verification process to the selected strategy.

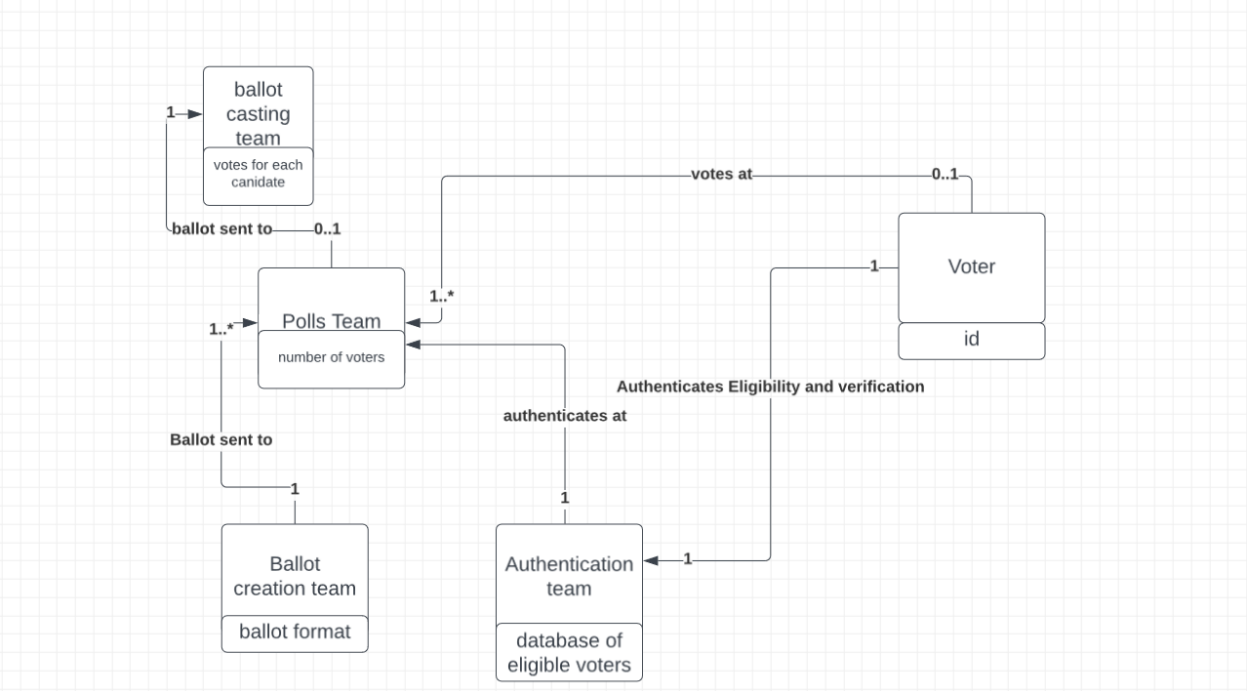
- **Strategy Interfaces:**
  - ElectionType: Interface defining the verify method for different verification strategies.
  - VerifyLocal: Interface for local election verification.
  - VerifyState: Interface for state election verification.
  - VerifyFederal: Interface for federal election verification.

4.3.3 VerifyMechanism - Abstract

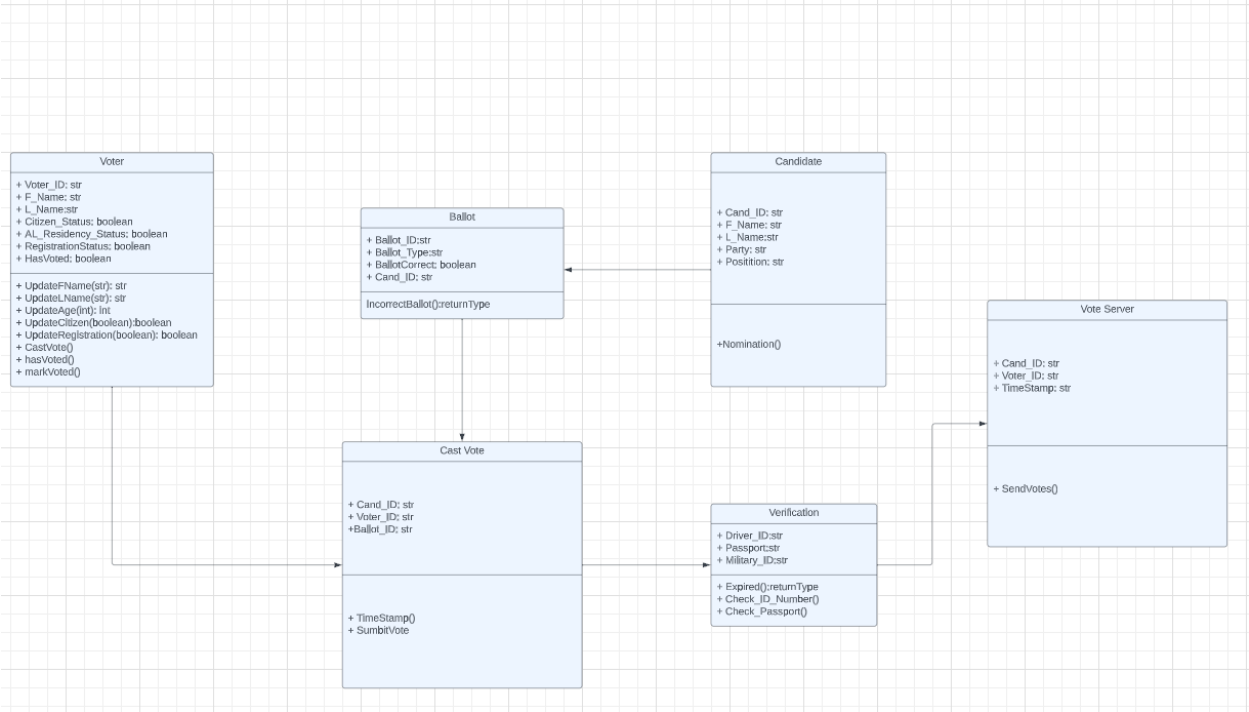
- FederalElectionStrategy- Implements ElectionType and VerifyFederal interfaces. Handles verification logic for federal elections.
- LocalElectionStrategy- Implements ElectionType and VerifyLocal interfaces. Handles verification logic for local elections.
- StateElectionStrategy- Implements ElectionType and VerifyState interfaces. Handles verification logic for local elections.

4.3 Diagrams

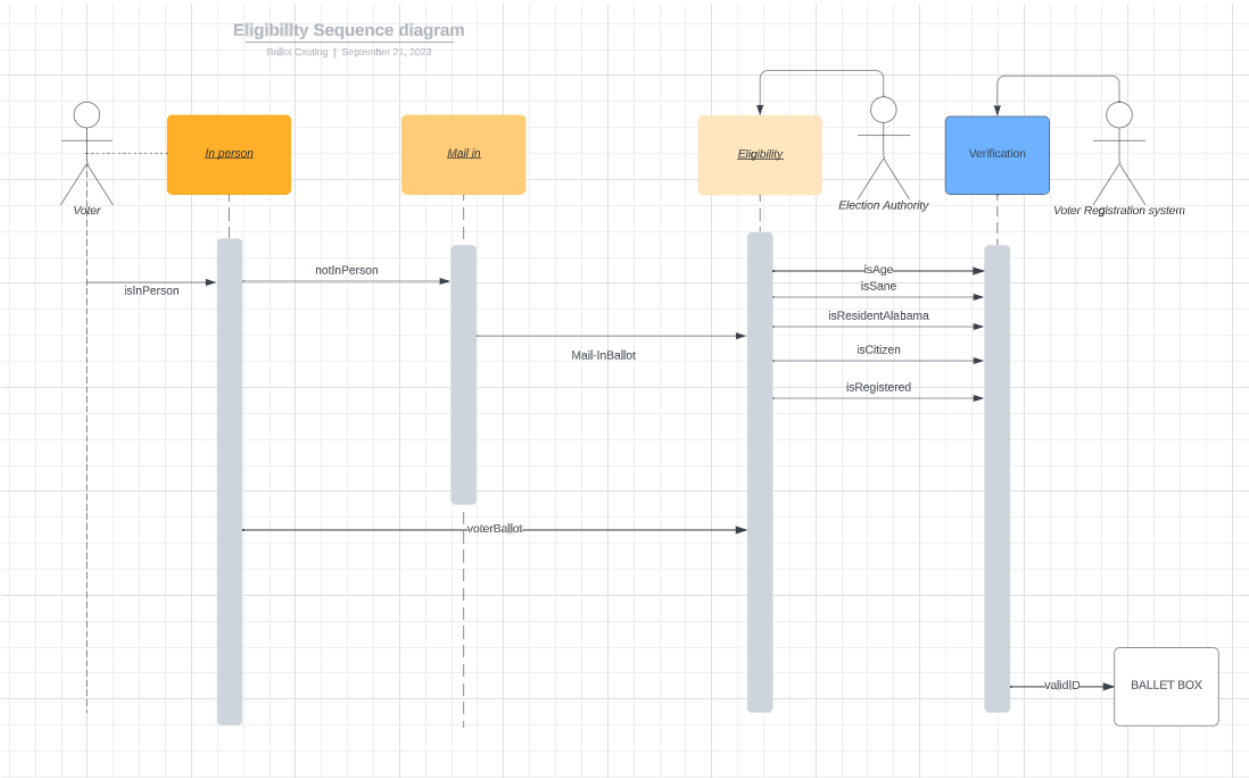
4.3.1 Domain Diagram



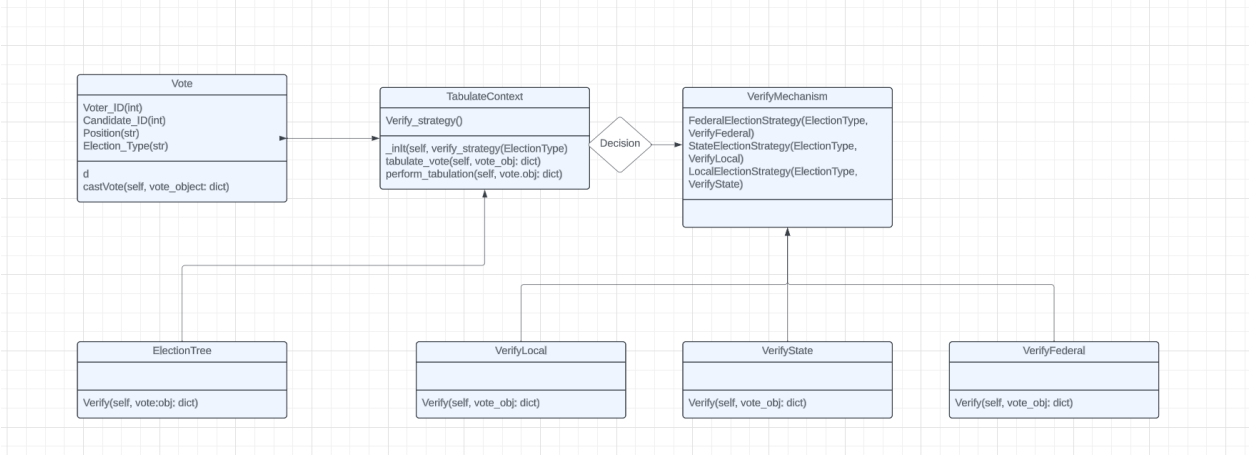
4.3.2 Class Diagram



4.3.3 Sequence Diagram



4.3.4 Strategy Diagram

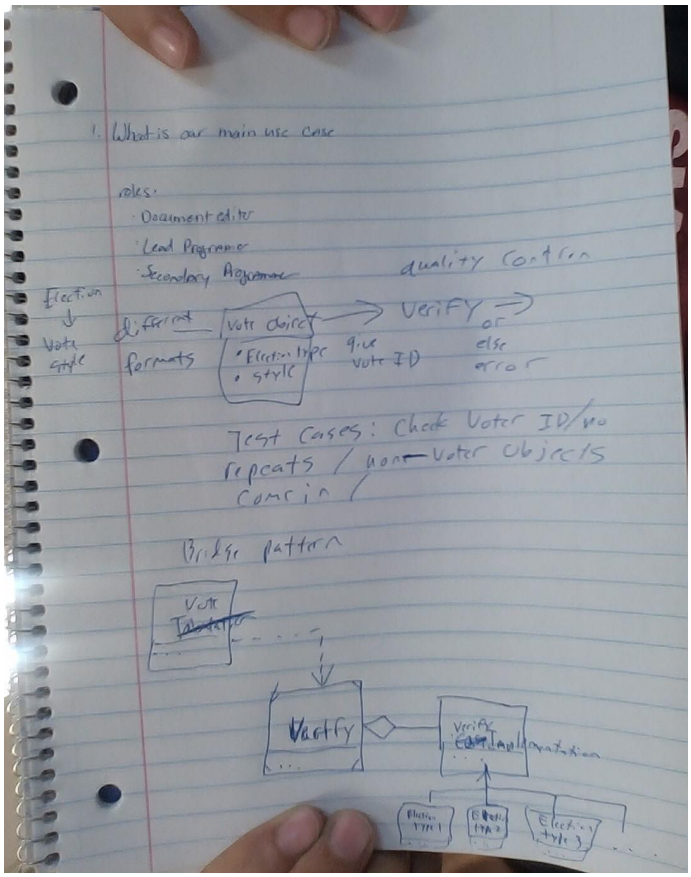


4.4 Testing Acceptance Plan

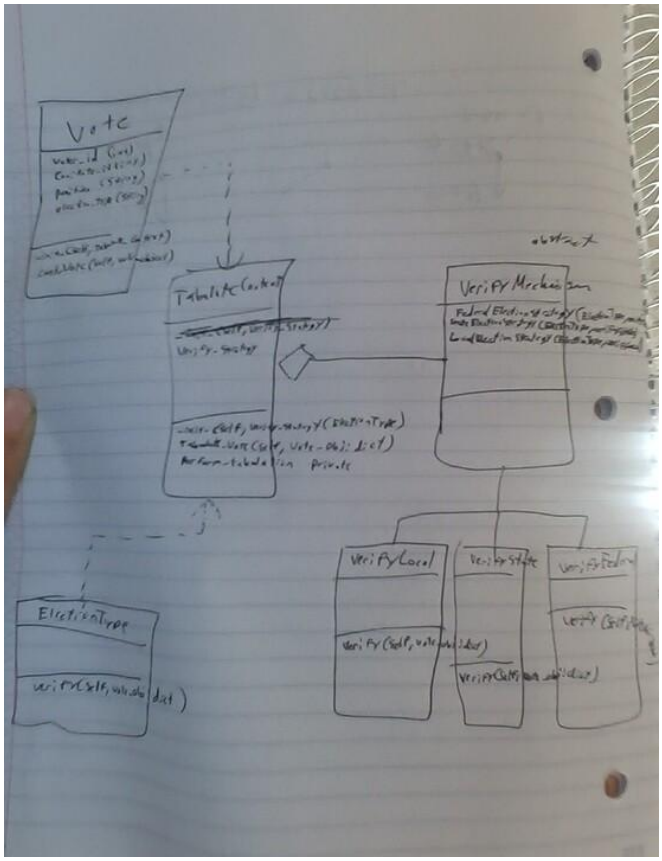
By using our testing plan found below each part of our program was systematically tested to ensure useability, reliability, and accuracy of the program. Tests consisted of using mock input values and comparing the outputs to the expected outputs. Tests were executed along with code completion and were presented to the client weekly.

The following patterns were approved by Dr. Shropshire

Original : Bridge Pattern for MVP



Final: Strategy Pattern for MVP



4.5 Testing Guidelines

Software testing is an important part of the software development life cycle and helps identify and correct software bugs or problems. Here are some general software testing guidelines to help ensure a complete and efficient testing process:

Consider Usability

- Evaluate the software and its usability through usability testing. Make sure the user interface is intuitive and user-friendly
- Test verification

Explore and learn from mistakes

- Review defects and test procedures after the project is completed.

Exit criteria

- Clearly define exit criteria for each test phase.
- Make sure that all the defined criteria are met before proceeding to the next step

4.5.1 Testing Chart

Test Coverage: • Have all the modules been tested? • Are different scenarios and edge cases considered in the test? • Is the code well commented with correct versions?	
Test Cases • Are there enough test cases to cover all possibilities? • Do the test cases run as expected? • Can you reproduce the test cases?	
Test data: • Are the data and inputs stated for the test cases? • Have both valid and invalid inputs been tested? • Is the data correctly set up?	
Test execution • Is the testing environment properly set up? • Have all external dependencies been mocked or stubbed? • How long does each test take?	
Assertions • Are assertions included to validate the expected results? • Do comparisons and assertions work as intended? • Do the error messages work as intended?	

Test organization ● Are all test cases organized into categories? ● Is there any specific ordering included in the test cases? ● Are the test names accurate?	
Test documentation: ● Is there proper documentation? ● Are there any issues that need to be documented? ● Is there a test plan or strategy document	
Test performance: ● Does the program run within acceptable time limits? ● Does the program hit targeted benchmarks? ● Are there any performance issues?	
Test maintenance: ● Are there any test coverage gaps? ● Does any test fail or skip? ● Are the test suites kept up to date	

4.6 Unit Tests

```
GENERAL TESTS


----------------------------------------

CORRECT FORMAT

Federal Verification Strategy
Voter Id: 12345
Candidate Id: 2
Position: president
Election Type: federal

Vote tabulated for candidate 2 for the position of president

Local Verification Strategy
Voter Id: 12345
Candidate Id: 2
Position: mayor
Election Type: local
Position: mayor
Vote tabulated for candidate 2 for the position of Mayor

State Verification Strategy
Voter Id: 12345
Candidate Id: 2
Position: governor
Election Type: state
Position: governor
Vote tabulated for candidate 2 for the position of Governor

----------------------------------------
```

```
----------------------------------------

WRONG KEYS

Federal Verification Strategy
Missing Key
Vote verification failed. Not tabulated.

Local Verification Strategy
Missing Key
Vote verification failed. Not tabulated.

State Verification Strategy
Missing Key
Vote verification failed. Not tabulated.

----------------------------------------

INCORRECT POSITIONS

Federal Verification Strategy
Voter Id: 12345
Candidate Id: 2
Position: governor
Election Type: federal
Wrong Position
Vote verification failed. Not tabulated.

Local Verification Strategy
Voter Id: 12345
Candidate Id: 2
Position: president
Election Type: local
Wrong Position
Vote verification failed. Not tabulated.

State Verification Strategy
Voter Id: 12345
Candidate Id: 2
Position: the beatles
Election Type: state
Wrong Position
Vote verification failed. Not tabulated.
```

Section 5 – Sprint 3
Will start on

Section 9 – References
TDB.

Section 10 – Glossary
TBD