Software Design Document

for

Voting System

Version 0.1

Prepared by
Kai Wang, wang8739
Jingfan Guo, guo00109
Ruoyan Kong, kong0135
Yuan Yao, yaoxx340

November 4, 2019

TABLE OF CONTENTS

TABLE OF CONTENTS **2**

1. INTRODUCTION **3**

1.1 Purpose 3

1.2 Scope 3

1.3 Overview 3

1.4 Reference Material 3

1.5 Definitions and Acronyms 4

2. SYSTEM OVERVIEW **4**

3. SYSTEM ARCHITECTURE **5**

3.1 Architectural Design 5

3.2 Decomposition Description 3 (functional descriptions of each class) 6

3.3 Design Rationale 7

4. DATA DESIGN **8**

4.1 Data Description 8

4.2 Data Dictionary 8

5. COMPONENT DESIGN **9**

6. HUMAN INTERFACE DESIGN **14**

6.1 Overview of User Interface 14

6.2 Screen Images 14

6.3 Screen Objects and Actions 15

7. REQUIREMENTS MATRIX **15**

8. APPENDICES **16**

1. INTRODUCTION

1.1 Purpose

This Software Design Document provides the design details of Voting System (VS). The expected audience are programmers, testers, and election officials of VS. It will also serve as a reference for voters.

1.2 Scope

This document contains a complete description of the design of VS. The basic architecture is a server from a command line window. The basic pages will be in command line mode. The designated programmers, testers, and election officials of VS will have full access to make changes, as he/she deems necessary. The changes could include, but not limited to, changing the menu window, data collected on each sub-window, and the calculation algorithms.

1.3 Overview

This document is intended for developers, project managers, marketing staff, users, testers, and documentation writers of VS.
Chapter 2: System overview, gives a general description of the functionality, context and design of the product.
Chapter 3: System architecture, describes organizing of the system, the major services provided by the product.
Chapter 4: Data design, describes how the information domain of your system is transformed into data structures, the system entities or major data along with their types and descriptions.
Chapter 5: Component design, describes what each component does systematically.
Chapter 6: Human interface design, describes the functionality of the system from the user's perspective.
Chapter 7: Requirements matrix, describes a cross reference that traces components and data structures to the requirements in the SRS document.
We recommend the readers to read the chapter 2 first, chapter 3 & 4 is mainly for developers and testers, users should also read chapter 5 - 7.

1.4 Reference Material

- Sommerville, I. (2016). *Software engineering*. Boston: Pearson.
- Software Requirements Specification Document for Voting System

- Project Waterfall: Writing the Software Design Document (SDD)
- Rubric for Grading of Writing the Software Design Document
- Software Design Document Template

1.5 Definitions and Acronyms

Term	Definition
SRS	Software Requirements Specification
VS	Voting System
SDD	Software Design Document

2. SYSTEM OVERVIEW

The Voting System (VS) is designed to facilitate party list voting, including both open party list voting and closed party list voting. It will provide the following capabilities:

- Reading and parsing a comma delimited text file containing ballots.
- Performing the election.
- Producing an audit file with the election information.
- Displaying to the screen the winners and information about the election.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

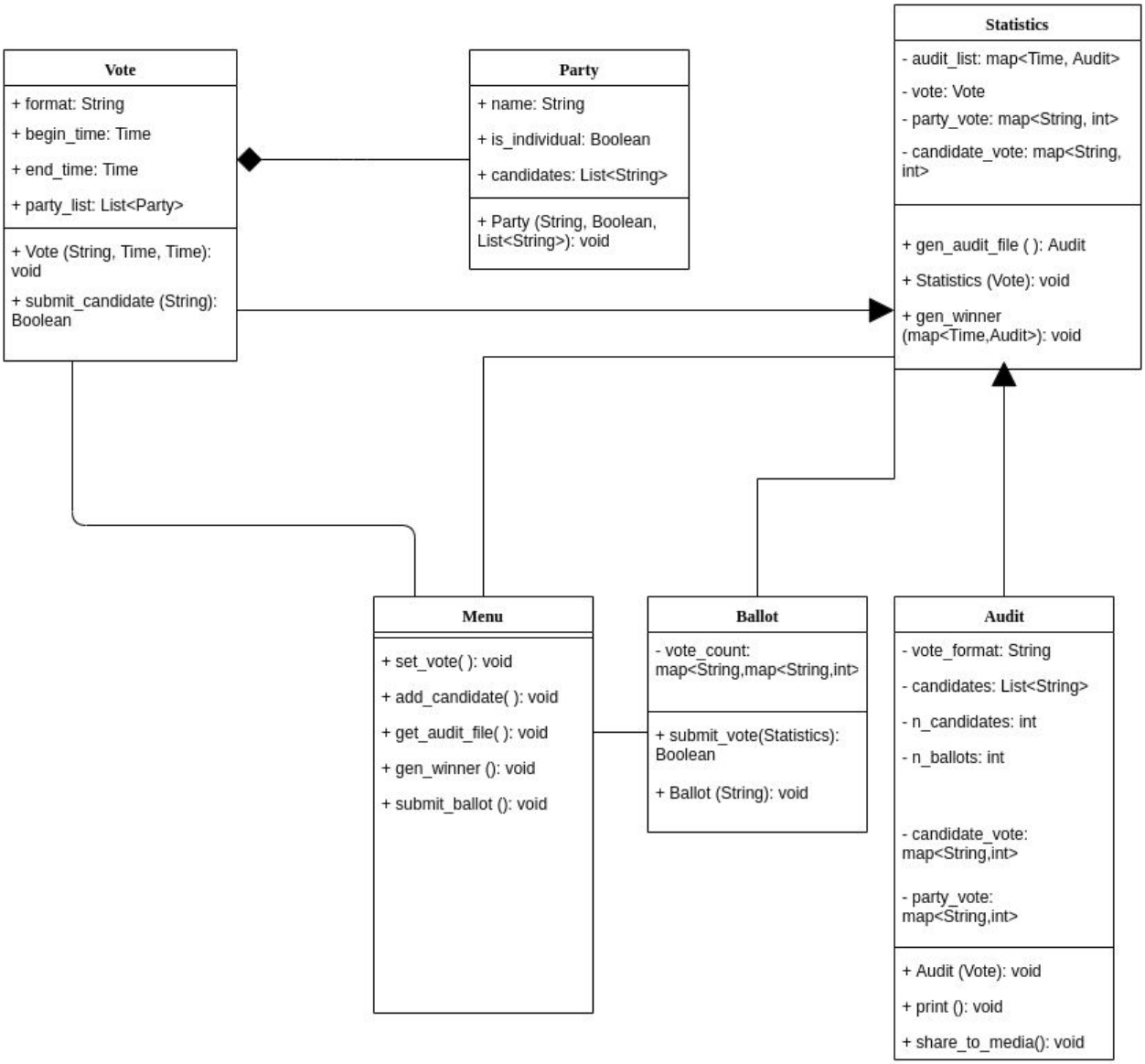


Figure 1. UML class diagram for the entire system

3.2 Decomposition Description 3 (functional descriptions of each class)

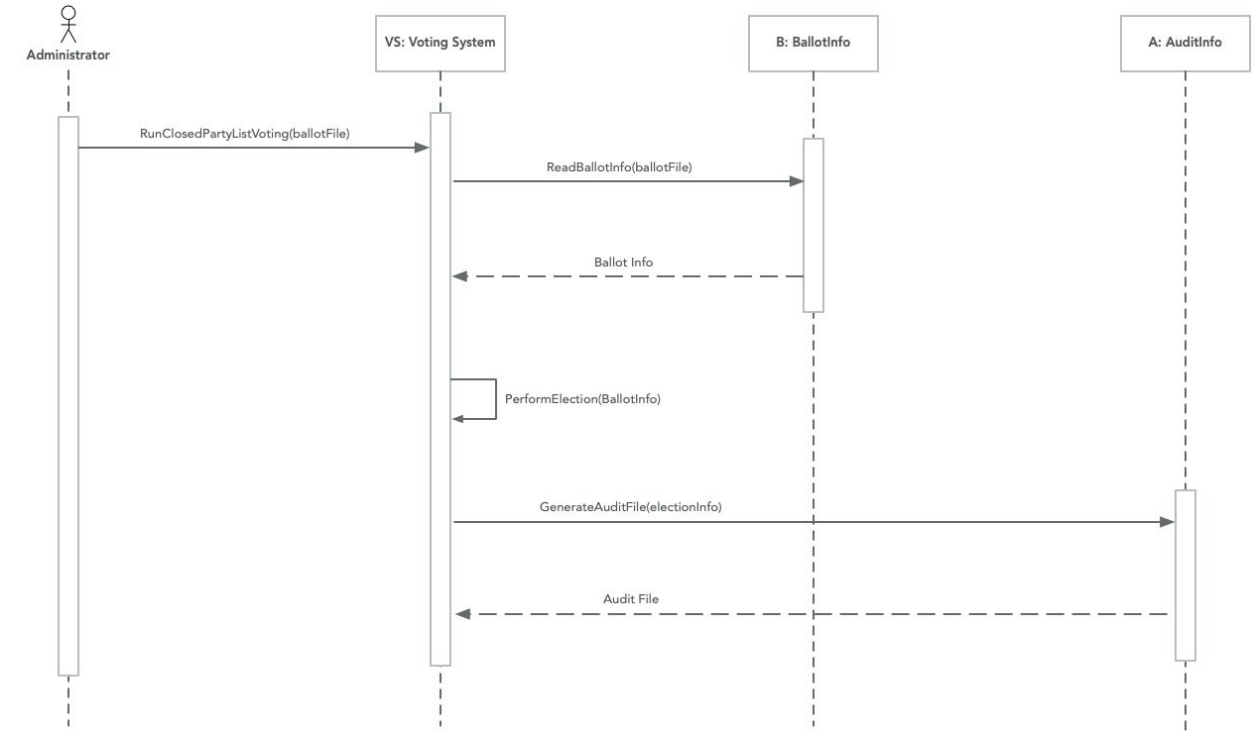


Figure 2. Sequence diagram for running the closed party list voting

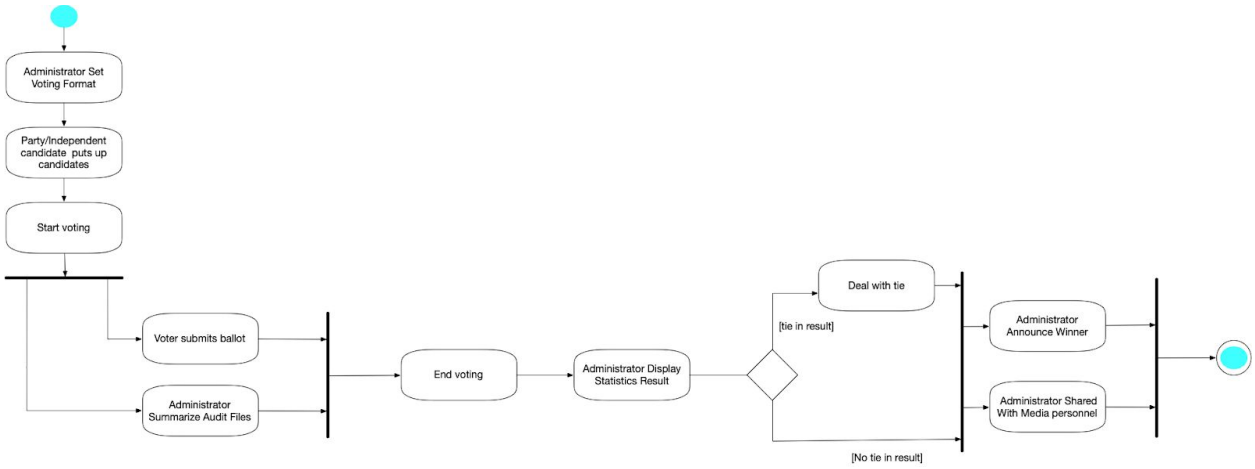


Figure 3. UML activity diagram (process model) for open party voting

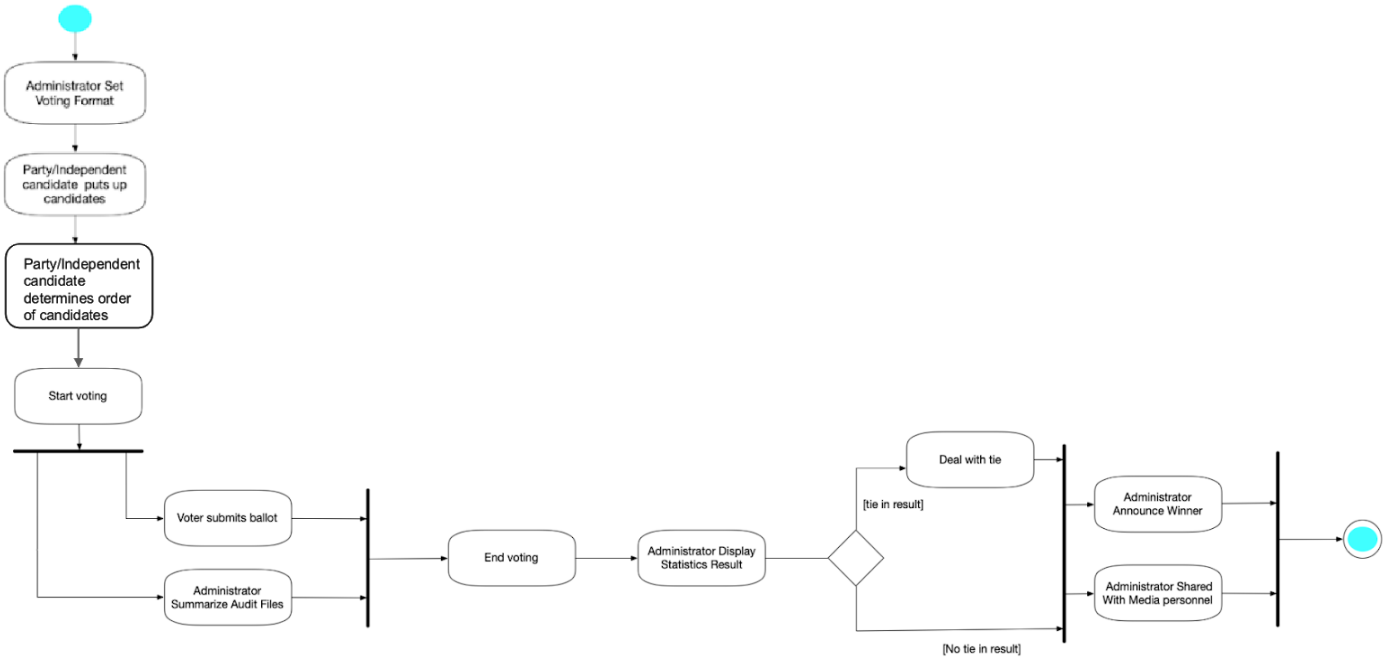


Figure 4. UML activity diagram (process model) for close party voting

3.3 Design Rationale

According to our design, the system is split into 6 entities -- Menu, Vote, Party, Statistics, Ballot, and Audit in the consideration of the balance of availability, performance and security.
Menu is in charge of all the interaction with users of VS. Another design -- letting each class in charge of their own UI was considered, but was discarded because it is not secure to expose the system architecture to all possible users.

Vote is in charge of controlling the process of the system, like storing the candidates information, so it composes Party class. Statistics is a separate entity to do all the statistics-related activities, like storing data, generated report, generated winner. We do consider combine these 2 entities -- let Vote also in charge of data-related duties. But we finally give up this design because the entity who controls the Vote process and the entity who do statistics should be 2 split parts, otherwise it becomes easy to change the vote results.

Ballot is the class in charge of transferring the ballots to Statistics. We split this part to make the parallel process of dealing with a lot of ballots at the same time possible.

Audit is the audit file generated for report. We split it because its main function is to show all the users of VS, and also wider users (like from social media) the result.

4. DATA DESIGN

4.1 Data Description

Audit Class store the current status of ballot audit, including vote format, candidate list, the number of candidates, the number of ballots, candidates voting map, and party voting map.

Ballot Class read the ballot file from the user input, extract the information include the name of the party, the name of the candidates, and their vote assigned. Then call the statistics class and update the vote count into the database.

Party Class stores the input information including party name, whether the party is an individual, and the list of candidates in the party.

Vote Class is constructed of the vote format (open or closed), the begin and end vote time, and the list of parties which participate in the election. It also updates whether the candidates are submitted.

Statistics Class is responsible for all the statistics-related activities. It stores an instance of Vote, a list of audits that it generated, the vote for each party and the vote for each candidate.

4.2 Data Dictionary

Attribute Name	Attribute Type	Attribute Size
ballot_file_name	String	50
begin_time	Time	8
candidate_count	int	8
candidate_name	String	30
end_time	Time	8
party_count	int	8
party_name	String	30
vote_format	String	30

5. COMPONENT DESIGN

Audit

Name: Audit

Type: Class

Description:

This class has 3 functions — audit a voting, print current audit results, and once finish auditing all the ballots, share the final result with media personnel. The user should input the vote name to Audit. Once an Audit instance gets the vote name, it starts auditing the current voting result, including vote format, candidate list, the number of candidates, the number of ballots, candidates voting map, and party voting map. When call the print method of the audit instance, it will print the statistic data stored in it. And it can send the data to media personnel when call the share_to_media method.

Operations:

Name: Audit

Arguments: vote(Vote)

Returns: None

Pre-condition: The voting has begun

Post-condition: Statistical data is assigned to the corresponding fields

Exceptions: None

Flow:

```
Audit(Vote vote)
    this.vote_format = vote.format
    this.party_list = vote.part_list
    for party in vote.part_list:
        this.candidate_list.append(party.candidate_list)
    this.n_candidates = this.candidate_list.len()
    this.n_ballot = current_count
    this.candidate_vote = Statistics.candidate_vote
    this.party_vote = Statistics.party_vote
```

Name: print

Arguments: None

Return: None

Pre-condition: Audit(vote) has run

Post-condition: Data output to screen

Exceptions: None

Flow:

```
print()
    print(this.vote_format)
    print(this.candidates)
    print(this.n_candidates)
```

print(this.n_ballots)
print(this.candidate_vote)
print(this.party_vote)

Name: share_to_media
Arguments: None
Return: None
Pre-condition: Audit(vote) has run
Post-condition: None
Exceptions: Cannot connect to media server

Flow:
try:
connect to media server
catch NoConnectionExceptiom:
print(“Fail to connect to media”)
return
save(audit)

Ballot

Name: Ballot
Type: class
Description:
This class has 2 functions -- construct a Ballot class from a file (for example, ballot.csv), and submit the count to Statistics class.
The user should input the ballot file name to Ballot. Once the Ballot get the file content from the file name, it will extract the vote count information of each party and candidate.
If the information pass the data integrity check, Ballot will call submit_vote to submit the information to Statistics. If the submit is failed, it will ask the user to try it again,
Operations:
Name: Ballot
Arguments: filename(String)
Return: None
Pre-conditions: The voting has begun and has not finished.
Post-conditions: The vote count for each candidate and party has been registered to the class variables.
Exceptions:
The file does not exist/ The format of the file is incorrect -- ask the user to check the file and input again.
Flow:
Ballot(String filename):
If not vote_begin_time <= current_time <= vote_end_time:
Print (“Can’t vote now!”)
Return
Try:

File_context = readfile(filename)
Catch NotExistException:
Print (“File Not Exist”)
Return to user input page
Try:
vote_count = extract_info(File_context)
Catch FormatMismatchException:
Print (“Wrong Format”)
Return to user input page
Call submit_vote

Name: submit_vote
Arguments: stat(Statistics)
Pre-conditions: vote_count has been registered.
Post-conditions: The vote count for each candidate and party has been registered to Statistics.
Return: None
Exceptions:
The party/candidate does not exist -- ask the user to check and input again.
Flow:
submit_Vote(Statistics stat):
If vote_count is_null:
Print (“No vote count information!”)
Return to user input page
try:
update(Statistics.party_vote, Statistics.candidate_vote)
Catch candidateNotExistException:
Print(“Candidate does not exist.”)
Return to user input page

Party

Name: Party
Type: class
Description:
This class has one function -- construct a Party class from party name (String), whether it’s individual (Boolean), and list of candidates. These are the essential information for a vote.
Operations:
Name: Party
Arguments: name(String), is_individual(Boolean), candidate(List<String>).
Return: None
Pre-conditions: The Party has not created.
Post-conditions: The Party entity is created.
Exceptions:

The candidate list is empty/ The Boolean is_individual is true, but the candidate list has more than one element -- ask the user to check the information and input again.
Flow:
Party(String name, is_individual Boolean, List<String> candidate):
If is_individual == true and len(candidate)>1:
Print (“Inconsistent input!”)
Return to user input page
If len(candidate) == 0:
Print (“Empty candidate list!”)
Return to user input page

Vote
Name: Vote
Type: class
Description:
This class has 2 functions -- construct a Vote class from vote type (String), begin time (Time), end time (Time), and list of parties, and submit the candidates.
The user should input the type of the vote, open or close, begin and end time of the vote, and the list of candidates’ names. These are the essential information for a vote.
If the information pass the data integrity check, Vote will call submit_candidate to submit the information about the candidates from all the parties. If the submit is failed, it will ask the user to try it again,
Operations:
Name: Vote
Arguments: format(String), begin_time(Time), end_time(Time), party_list(List<Party>).
Return: None
Pre-conditions: The voting has not begun.
Post-conditions: The vote event is created.
Exceptions:
The input format or party doesn’t exist/ The format of the time is incorrect -- ask the user to check the vote information and input again.
Flow:
Vote(String format, Time begin_time, Time end_time, List<Party> party_list):
If not begin_time < end_time:
Print (“Invalid time!”)
Return to user input page
If format != ‘open’ or ‘close’:
Print (“Invalid format!”)
Return to user input page
If len(party_list) == 0:
Print (“Empty party list!”)

Return to user input page
Call submit_candidate

Name: submit_candidate
Arguments: str(String)
Pre-conditions: Vote has been created.
Post-conditions: The candidates from all the parties is submitted.
Return: None
Exceptions:
The input candidate is empty -- ask the user to check and input again.
Flow:
submit_candidate(String str):
If str is_null:
Print (“No input candidate!”)
Return to user input page

Statistics
Name: Statistics
Type: class
Description:
This class has 3 functions -- performing statistics, generating an audit file and generating the winner.
The vote for each party and the vote for each candidate in Statistics can be updated by Ballot. When it is called with gen_audit_file(), it will perform the statistics and generate an audit and store it with current timestamp. When it is called with gen_winner(), it will generate the winner for each audit.
Operations:
Name: Statistics
Arguments: vote_in(Vote)
Return: None
Pre-conditions: The voting has begun or finished
Post-conditions: The vote is registered to Statistics class.
Exceptions: None
Flow:
Statistics(Vote vote_in):
If not current_time >= vote_begin_time:
Print (“Can’t register vote now!”)
Return
vote = vote_in

Name: gen_audit_file
Arguments: None
Return: Audit

Pre-conditions: The vote has been registered to Statistics class.
Post-conditions: An audit with current timestamp is generated and stored in a list.
Exceptions:
The vote has not been registered to Statistics class.

Flow:

```
gen_audit_file():
    If vote is_null:
        Print (“Can’t perform statistics now!”)
        Return
    audit = Audit(vote)
    audit_list[current_time] = audit
    Return audit
```

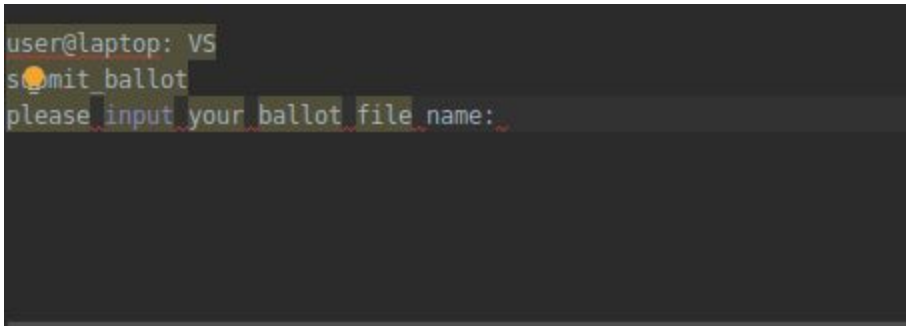
6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

VS is a command prompt application. After voting has finished, the user can input ballot file into the system. The user needs to input the path and filename to the command line. VS will find and process the ballot file. Once the file process has finished, the user is able to select what kind of output to see, for example, the election information (e.g. Type of Voting, Number of Candidates, Candidates, Number of Ballots, calculations, how many votes a candidate or party had, etc) and the winner of the election.

6.2 Screen Images

```
user@laptop: VS
help
    -- help : display actions and descriptions
    -- submit_ballot : submit your ballot
    -- set_voting : set voting format
    -- start_audit : start auditing ballot files
    -- get_audit_file : get current audit status
    -- get_winner : return the final winner reuslt
    -- share_with_media : pass final result to the media
```



6.3 Screen Objects and Actions

The user input VS to start VS server to load Menu.
There are following actions in the menu:

1) help
   This action displays all actions can be done with brief descriptions.
2) submit_ballot
   This action is for submitting a ballot. It will call class, then construct Ballot (filename).
   Then Ballot submit vote_count to Statistics.
3) set_voting
   This action set the voting format, including open/close, start/end time
4) start_audit
   This action starts auditing ballot files, when all file is done it outputs reminds of all done.
5) get_audit_file
   This action generates the audit file of current statistical information.

6) gen_winner
   This action generates the winner that will be displayed to the screen.
7) share_with_media
   This action pass final audit file to the media.

7. REQUIREMENTS MATRIX

Requirement ID	Requirement	SRS ID
001	Voter should be able to submit their votes	4.3
002	Administrator should be able to set voting format	4.4
003	Administrator should be able to announce Winner	4.5
004	Administrator should be able to display statistics result	4.6
005	Administrator should be able to summarize audit files	4.7
006	The voting will be shared with media personnel	4.8

8. APPENDICES

No appendices for now.