# Team 9

**Vote Counting System**

# Software Design Document

Names: Abdikarem Fareh, Isaac Xiong, Luke Chen, Rohan Abraham

Team09

Date: (02/28/2021)

**TABLE OF CONTENTS**

1. [**INTRODUCTION 2**](#_heading=h.gjdgxs)
   1. [Purpose 2](#_heading=h.30j0zll)
   2. [Scope 2](#_heading=h.1fob9te)
   3. [Overview 2](#_heading=h.3znysh7)
   4. [Reference Material 2](#_heading=h.2et92p0)
   5. [Definitions and Acronyms 2](#_heading=h.tyjcwt)
2. [**SYSTEM OVERVIEW 2**](#_heading=h.3dy6vkm)
3. [**SYSTEM ARCHITECTURE 2**](#_heading=h.1t3h5sf)
   1. [Architectural Design 2](#_heading=h.4d34og8)
   2. [Decomposition Description 3](#_heading=h.2s8eyo1)
   3. [Design Rationale 3](#_heading=h.17dp8vu)
4. [**DATA DESIGN 3**](#_heading=h.3rdcrjn)
   1. [Data Description 3](#_heading=h.26in1rg)
   2. [Data Dictionary 3](#_heading=h.lnxbz9)
5. [**COMPONENT DESIGN 3**](#_heading=h.35nkun2)
6. [**HUMAN INTERFACE DESIGN 4**](#_heading=h.1ksv4uv)
   1. [Overview of User Interface 4](#_heading=h.44sinio)
   2. [Screen Images 4](#_heading=h.2jxsxqh)
   3. [Screen Objects and Actions 4](#_heading=h.z337ya)
7. [**REQUIREMENTS MATRIX 4**](#_heading=h.3j2qqm3)
8. [**APPENDICES 4**](#_heading=h.1y810tw)

### INTRODUCTION

## Purpose

This software design document describes the architecture and system design details of the Vote Count System. This document is intended for election officials who will run an election as an Instant Run Off (IR) or Open Party Listing (OPL) type and will return the results of the chosen election.

## Scope

This product will create a Vote Counting System application. This will be a system where a CSV election file of the votes and election type will be entered and a file listing the winner and methodology will be given. This gives a way to document the voting process and results in an efficient and convenient manner. When dealing with large amounts of ballots, an automated system to count the votes is essential to reduce errors and increase speed to determine the winner of the election.

## Overview

The remaining sections and their components are listed below.

Section 2 concerns the System Overview of the functionality, context, and design of this product.

Section 3 is the Architecture Design that uses a UML Class Diagram. This diagram shows the different classes and structures, along with their class variables and methods, that are used in this Vote Count System. It shows the details of each design entity and how the different objects are related to each other to perform this products’ service.

Section 4 concerns the Data Design of how the ballots, candidates, and audit are implemented in this product.

Section 5 contains the Component Design which goes into detail about each component and its functionality in the Vote Count System.

Section 6 shows the Human Interface Design. This section shows screenshots of what the user will see when the Vote Count System is called and run.

Section 7 is a Requirements Matrix that traces the components and data structures to the requirements stated in the SRS document.

## Reference Material

List any documents, if any, which were used as sources of information for the test plan.

Software Design Document (SDD) Template:

<https://canvas.umn.edu/courses/217849/files/19377850/download?download_frd=>1

Instant Runoff Ballot:

<https://www.fairvote.org/rcv#where_is_ranked_choice_voting_used>

Proportional voting type:

<https://www.fairvote.org/how_proportional_representation_elections_work>

## Definitions and Acronyms

CSV - Comma separated values file

Instant Runoff Voting (IR) - Linked in section 1.4

Open Party Listing (OPL) - Linked in section 1.4

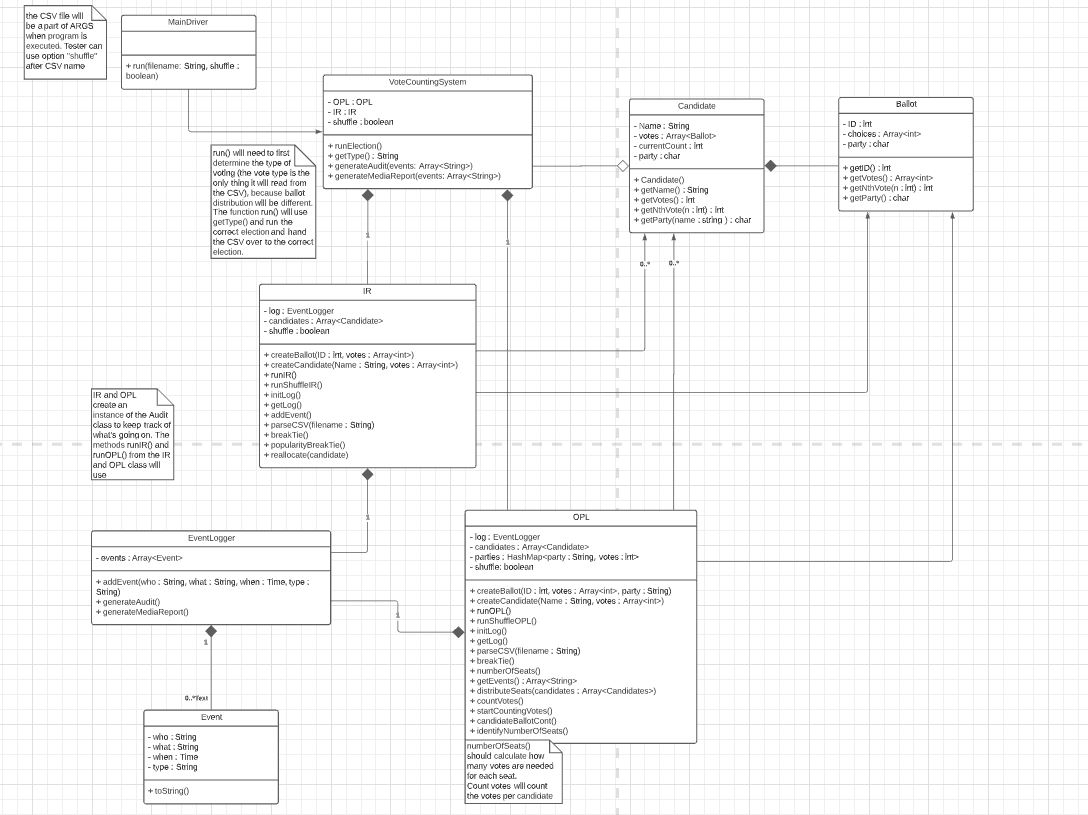
### SYSTEM OVERVIEW

Initially the user will run the program with the filename as a command line argument. After the system identifies the type of voting system (either instant runoff voting or open party listing), it will execute the following procedure: the ballot election info will be collected from the csv file. Ballots will be made and associated with candidates. Then winners will be declared and then there will be an option to generate an audit and or a media report.

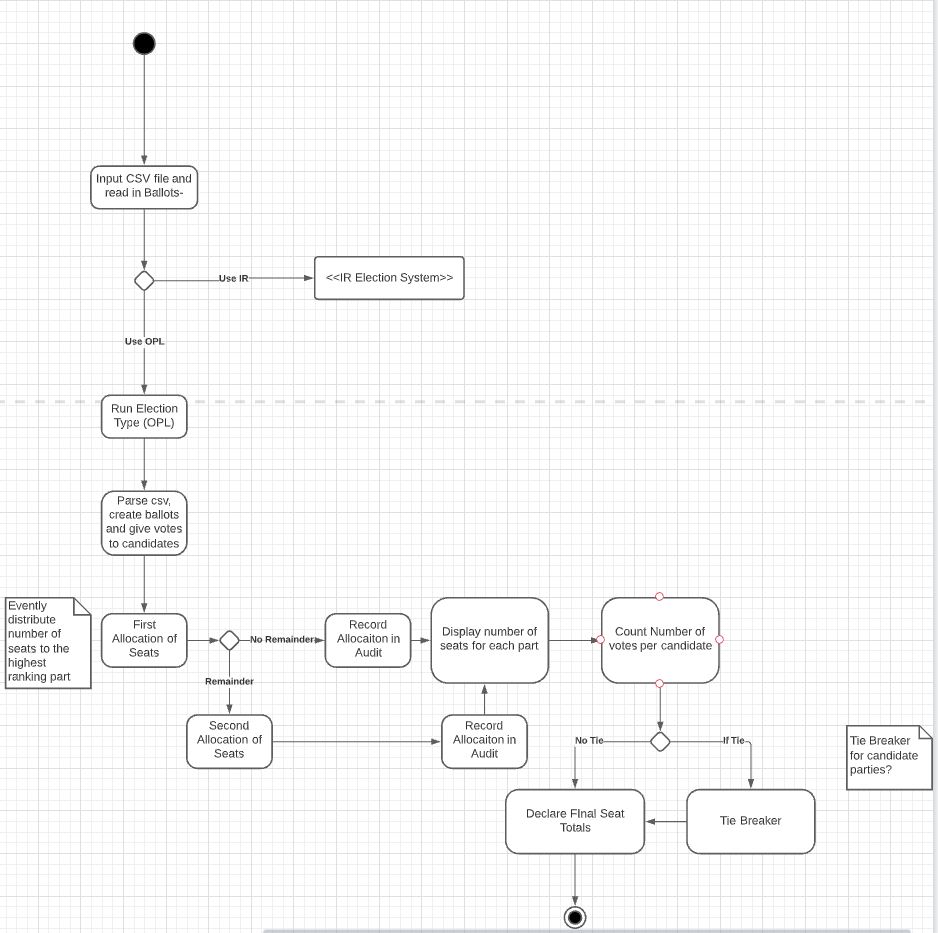
### SYSTEM ARCHITECTURE

## Architectural Design

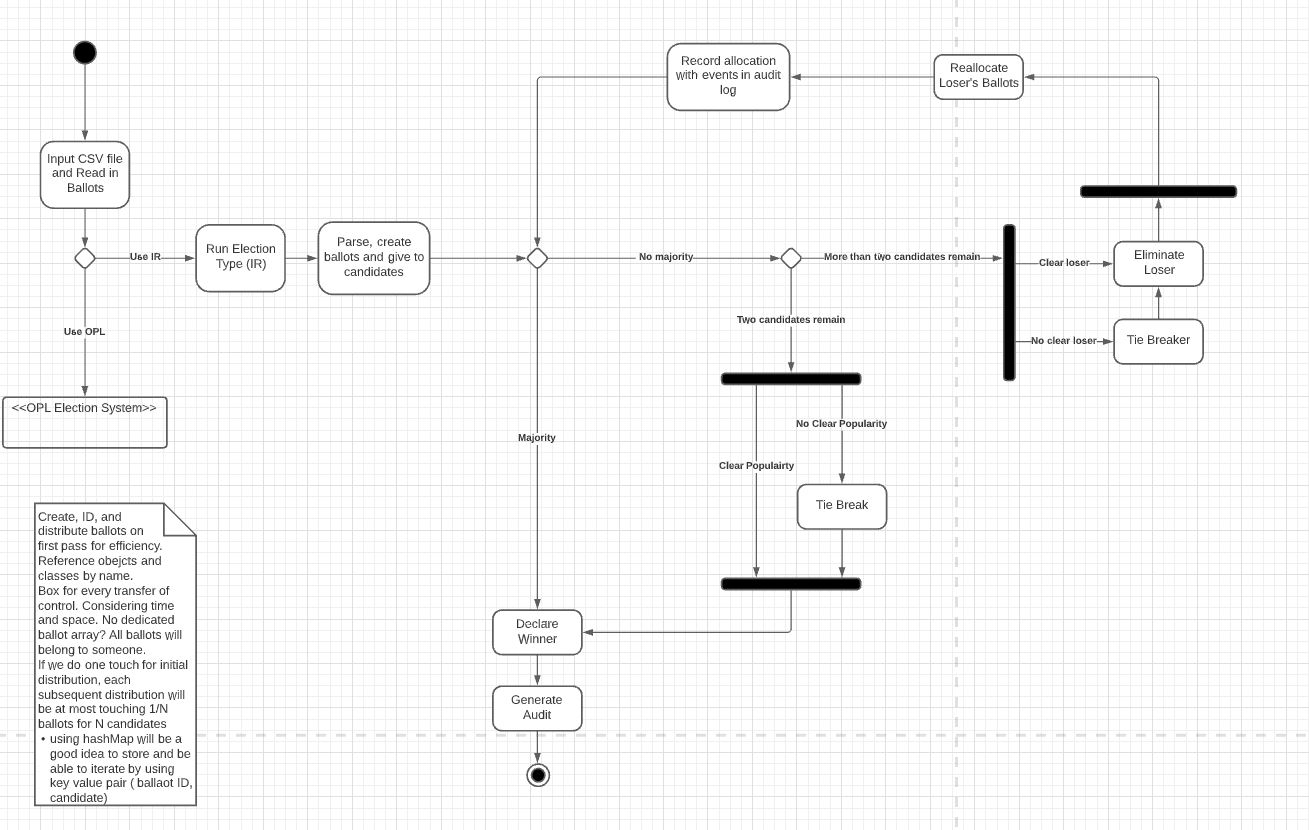
If any of the diagrams are hard to read, they are in other parts of the SDD folder.



This is the UML Class diagram of the Vote Count System.

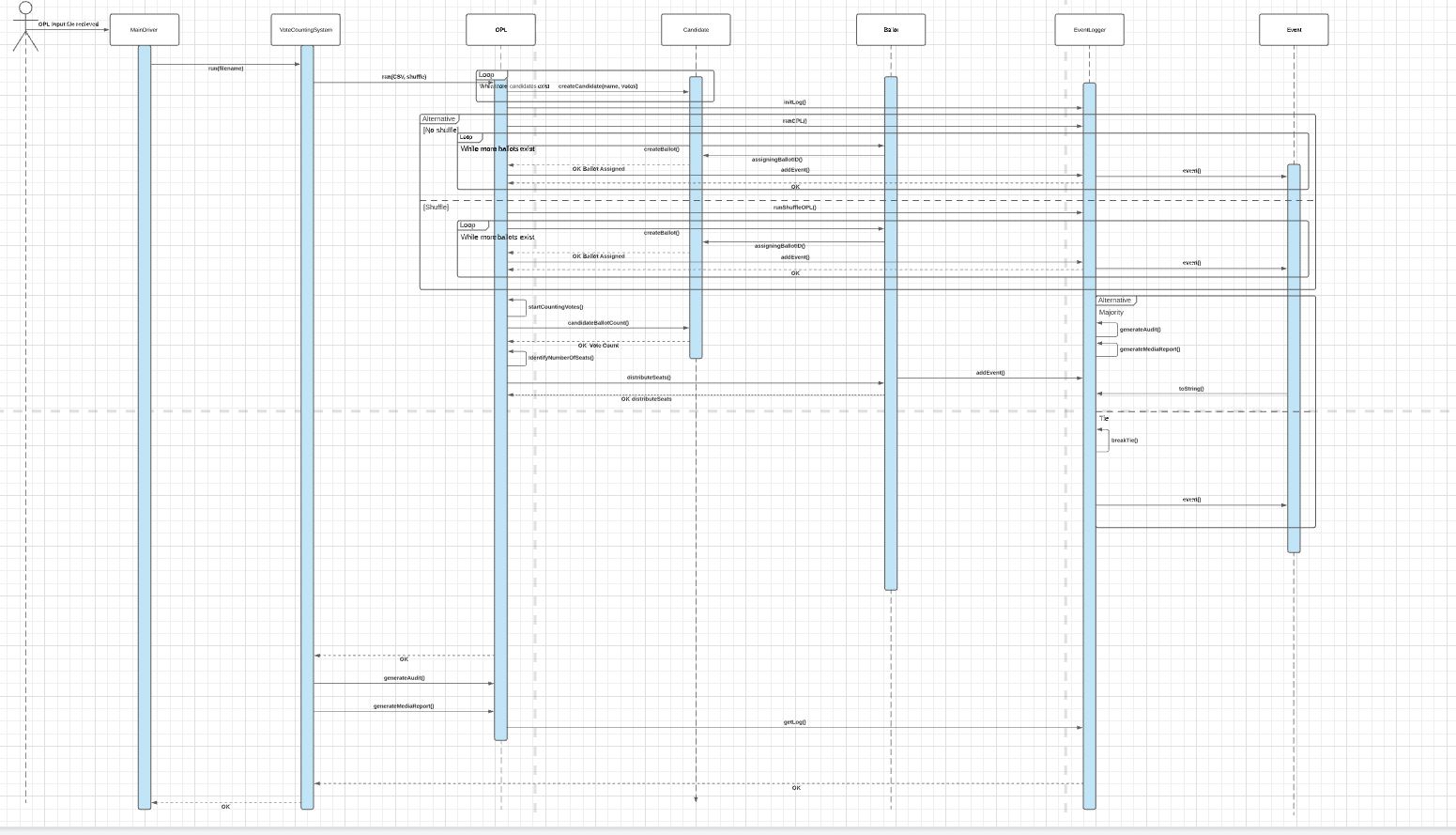


This is the activity diagram for when the election is run in OPL.



This is the activity diagram of when the election is run in IR.

## Decomposition Description



This is the sequence diagram which goes through the flow of the Vote Count System when it is run.

## Design Rationale

Our initial architecture was more barebones featuring around 5 classes including a driver, candidate, ballot, IR and OPL classes. We felt like this initial design was not good enough to encompass all of the complexity found within the voting system properly. We created a class that would generate the audit and media reports as well as an event and event logger class. The idea was we wanted a clean way to represent the stepping through the system for the audit. This made class interactions more complicated to design, but will make the code more clean and understandable.

### DATA DESIGN

## Data Description

Data parsed from the input CSV will be categorized into election type, candidates (and parties), and ballots. The election type is used to identify what type of election, whether IR or OPL, the system should run for that CSV. Candidates and ballots will be parsed into Candidate and Ballot objects so that their associated information stays together. For a Candidate, the most important pieces of information to keep together are their name (String), their party (char), and their voters (Array<Ballots>). For a Ballot, the most important pieces of information are their ballot ID (int) and which candidates they voted for and in what order (Array<int>). The system will create an OPL or IR class to handle the election. The OPL or IR object will hold an array of candidates, and the candidates will hold an array of the ballots that voted for them.

Aside from the main Ballot, Candidate, OPL, and IR classes, there will be an EventLogger class to keep track of where ballots go at all times so that an audit report can be made. The EventLogger will hold a chronological array of Events. Events are a class describing what action happened, who did it, and any other parties, rules, or functionality (like tie-breaking) that was involved. These records of who, what, and when are string attributes of the Event class.

## Data Dictionary

|  |  |  |
| --- | --- | --- |
| **System Entities and Major Data** | **Types** | **Methods** |
| Ballot | - ID : int  - choices : Array<int>  - party : char | + getID() : int  + getVotes() : Array<int>  + getNthVote(n : int) : int  + getParty() : char |
| Candidate | - name : String  - votes : Array<Ballot>  - currentCount : int  - party : char | + Candidate()  + getName() : String  + getVotes() : int  + getNthVote(n : int) : int  + getParty(name : string ) : char |
| Event | - who : String  - what : String  - when : Time  - type : String | + toString() |
| EventLogger | - events : Array<Event> | + addEvent(who : String, what : String, when : Time, type : String)  + generateAudit()  + generateMediaReport() |
| IR | - log : EventLogger  - candidates : Array<Candidate>  - shuffle : boolean | + createBallot(ID : int, votes : Array<int>)  + createCandidate(Name : String, votes : Array<int>)  + runIR()  + runShuffleIR()  + initLog()  + getLog()  + addEvent()  + parseCSV(filename : String)  + breakTie()  + popularityBreakTie()  + reallocate(candidate) |
| OPL | - log : EventLogger  - candidates : Array<Candidate>  - parties : HashMap<party : String, votes : int>  - shuffle: boolean | + createBallot(ID : int, votes : Array<int>, party : String)  + createCandidate(Name : String, votes : Array<int>)  + runOPL()  + runShuffleOPL()  + initLog()  + getLog()  + parseCSV(filename : String)  + breakTie()  + numberOfSeats()  + getEvents() : Array<String>  + distributeSeats(candidates : Array<Candidates>)  + countVotes()  + startCountingVotes()  +candidateBallotCont() |

### COMPONENT DESIGN

We have two sequence diagrams for OPL and IR. For OPL sequence, as soon as we receive the CSV input file, the main driver will call run(fileName) via VoteCountingSystem where the main class for OPL will be called. Then, for each file received, we are going to record candidates with their parties while creating ballot ID. After that, we will calculate the total number of votes for each party to identify the number of seats. We will also identify the candidates with the highest number of votes. If there is a tie, then we will call the tie breaker to break the tie and announce a winner. Lastly, we log every event that took place from the initial time we received the input file till we announced a winner.

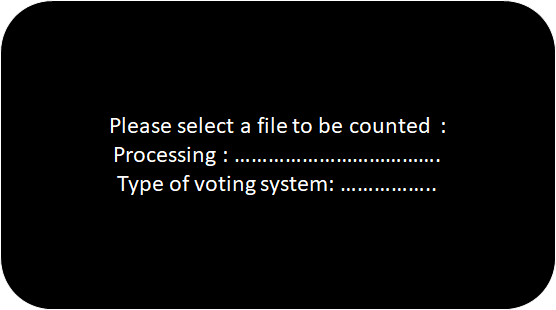
For IR, as soon as the CSV file received, the main driver will be called to run(fileName) via VoteCountingSystem where the main driver calls IR class. Then, for each file received, we are going to record candidates with their parties while creating ballot ID. Then, the IR class will calculate if there is a majority, and if there is no majority an elimination process will start. And if there is a tie, a tie breaker will be called. And finally we will announce the winner while every event is logged into the event logger.

### HUMAN INTERFACE DESIGN

## Overview of User Interface

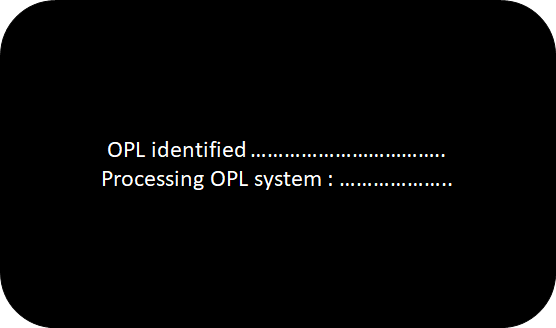
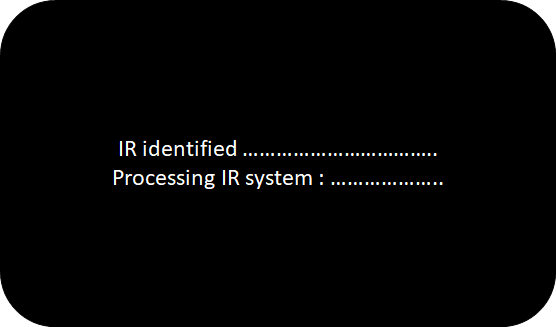
In this following section, images of how the system will run to the user is shown. The user will call the Vote Count System from the command prompt as well as the CSV election file. From the election file, IR or OPL voting will take place to determine the results of the election.

## Screen Images



This image is what the user will see first after calling the program. The system will read in the file and process the type of voting for the election.

Second prompt, depending on the type of voting: IR or OP, then process votes.



## Screen Objects and Actions

Once the election is run and the votes have been processed, the system will show the results on the screen.



This is an example of how the system will show the results of an election run in IR.

1st line: Announce that the election is over

2nd line: IR if instant runoff

3rd line: Number of Candidates

4th line : The candidates separated by commas, (\*) denotes the winner of the election

5th line: Number of ballots in the file

Rest of lines: List out the names of Candidates first, then each following line is the total amount of votes each respective candidate received. Lines after the first total are the new totals for the Candidates after the loser with the least amount of votes gets eliminated and votes are reallocated.

This is an example of how the system will show the results of an election run in OPL.

1st line: Announce that the election is over

2nd line: OPLif open party listing

3rd line: Number of Candidates

4th line : The candidates and their respective party separated by commas

Rest of the lines: Prints each party and their total amount of votes, then prints out the number of allocated seats for that party. This gets repeated for each allocation round and finally prints the percentages of votes / percentages of seats.

### REQUIREMENTS MATRIX

This is a table that cross references the use cases from the SRS with each respective system component.

|  |  |
| --- | --- |
| Use Cases from SRS | System Components |
| Input CSV File | MainDriver, VoteCountingSystem |
| Run Instant Runoff | MainDriver, VoteCountingSystem, IR, Candidate, Ballot, votes: Array<Ballot>, choices: Array<int>, candidates: Array<Candidates> |
| Run OPL | MainDriver, VoteCountingSystem, OPL, Candidate, Ballot, votes: Array<Ballot>, choices: Array<int>, Candidates: Array<Candidates>, parties: HashMap<party : String, votes : int> |
| Create Audit | Event, log: EventLogger, events: Array<Events>, VoteCountingSystem |
| View Audit | Event, log: EventLogger, events: Array<Events>, VoteCountingSystem |
| Create Media Report | Event, log: EventLogger, events: Array<Events>, VoteCountingSystem |
| View Media Report | Event, log: EventLogger, events: Array<Events>, VoteCountingSystem |
| Return Results to Display | Event, EventLogger, events: Array<Events> |
| Shuffle Votes | IR, OPL, VoteCountingSystem |
| Tie Breaker | IR, OPL, VoteCountingSystem |