(Group 2)

**(Elect and Verify (EnV): A system for online verifiable elections)**

Software Design Document (SDD)

**Prepared By:**

Ahmed Abd El Fattah Ewais

Hussam Ashraf El-Araby

Islam Faisal Ibrahim

Mohamed Amr Gadalla

**Supervised by:**

Dr. Sherif El-Kassas

Mr. Hossam Medhat

**Date: (12/06/2015)**

1. Introduction 3

1.1.Document Purpose 3

1.2.Document Scope 3

1.3. Document Overview 3

1.4. Reference Material 3

1.5. Definitions, acronyms and abbreviations 4

2. System Overview 6

2.1. Product Perspective 6

2.2. System Context 6

2.3. Process Model 7

2.4. Product Functions 8

2.4. User Classes and Characteristics 8

3. System Architecture 10

3.1 Architectural Design 10

4. DATA DESIGN 14

4.1 Data Description 14

4.2 Data Dictionary 18

5. Dynamic Model 39

5.1. Sequence Diagrams 39

6. Human Interface Design 43

6.1 Overview of User Interface 43

6.2 Screen Images 43

6.3 Screen Objects and Actions 53

# Introduction

## Document Purpose

The purpose of this Software Design Document (SDD) is to give an overview of the system architecture, the communication and interfaces, as well as the design of each component. The SDD includes various graphical diagrams that explain the design in more details such as architecture, context, class and sequence diagrams. The intended audience of this document are the client engineers as well as the development, testing and deployment team. Client engineers should review the SDD to make sure the design meets their business needs. The development team shall use this as the main reference during the development of the high level details of the system. Testing team shall use the document to generate load tests, unit tests and integration tests. Deployment team shall take into consideration the backup, interface and communication notes outline in the document.

## Document Scope

This document is a software design document for the online digital democracy solution ***Elect and Verify***. This document is a follow-up for the software requirements specification (SRS) published earlier in November, 2015. *EnV*’s main goal is to make the whole campaigning, voting and result publishing online and at the same time maintaining the privacy of ballots and integrity of the elections. The elections laws and rules are flexible and can be determined via configuration files. *EnV* allows the users to verify the elections and tallying process.

## 1.3. Document Overview

This document starts with an introduction explaining the motivation of the software, scope of the project and purpose of the document. The document then highlights and explains in details

## 1.4. Reference Material

[1] B. Schneier, *Applied cryptography*. New York: Wiley, 1996.

[2] A. Ewais, H. El-Araby, I. Faisal and M. Gadalla, *Software Requirements Specification For Elect and Verify (EnV): A system for online verifiable elections*, November 2015.

## 1.5. Definitions, acronyms and abbreviations

Table 1: Important Definitions

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Administrators, (System) | System users who are in charge of technical configurations, installations and maintenance. |
| Ballot | A secured verifiable cast vote. |
| Candidate | A person who is running for elections. |
| Elect and Verify (EnV) | The whole system (see system) and all its documentation, and accompanying source code and design documents. |
| Election | The process of asking a group of people for choices in a private aggregated way. |
| Officials, Elections | Users of the system who are authorized to configure all the election configurations and laws. |
| Subsystem | Any partial component or communication interface of the system. |
| System, (The) | The whole electoral system including the registration, pre-elections, voting, counting, verification, campainging and debating subsystems with all its offline and online components. |
| Trustees, Elections | Users of the system who are capable of collabroatively issue tallying requests. |
| Type, Election | The type of elections including the type of candidates and voting type. |
| Voters, (Eligible) | All eligible voters who can participate in an election. |

Table 2: Important acronyms

|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| API | Application Programming Interface |
| CAPTCHA | Completely Automated Public Turing test to tell Computers and Humans Apart |
| CRUD | Create, Read, Update and Delete |
| CSRF | Cross-Site Request Forgery |
| DBMS | Database Management System |
| DoS | Denial-of-Service |
| EnV | Elect and Verify (The system) |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol over Transport Layer Security |
| IP | Internet Protocol |
| JSON | JavaScript Object Notation |
| MVC | Model–view–controller |
| OOP | Object-Oriented Programming |
| REST | Representational State Transfer |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| UI | User Interface |
| XSS | Cross-site scripting |

# System Overview

This section gives a high-level picture of the system, its boundaries, and interaction with external systems, business process and main functionalities.

## 2.1. Product Perspective

EnV is intended to replace traditional paper-based voting systems as well as online voting systems whose results cannot be publicly audited. EnV is an online verifiable open-audit voting system. EnV also includes solutions for electoral campaigns and debates. EnV is not designed however to solve the problem of coercion.

## 2.2. System Context

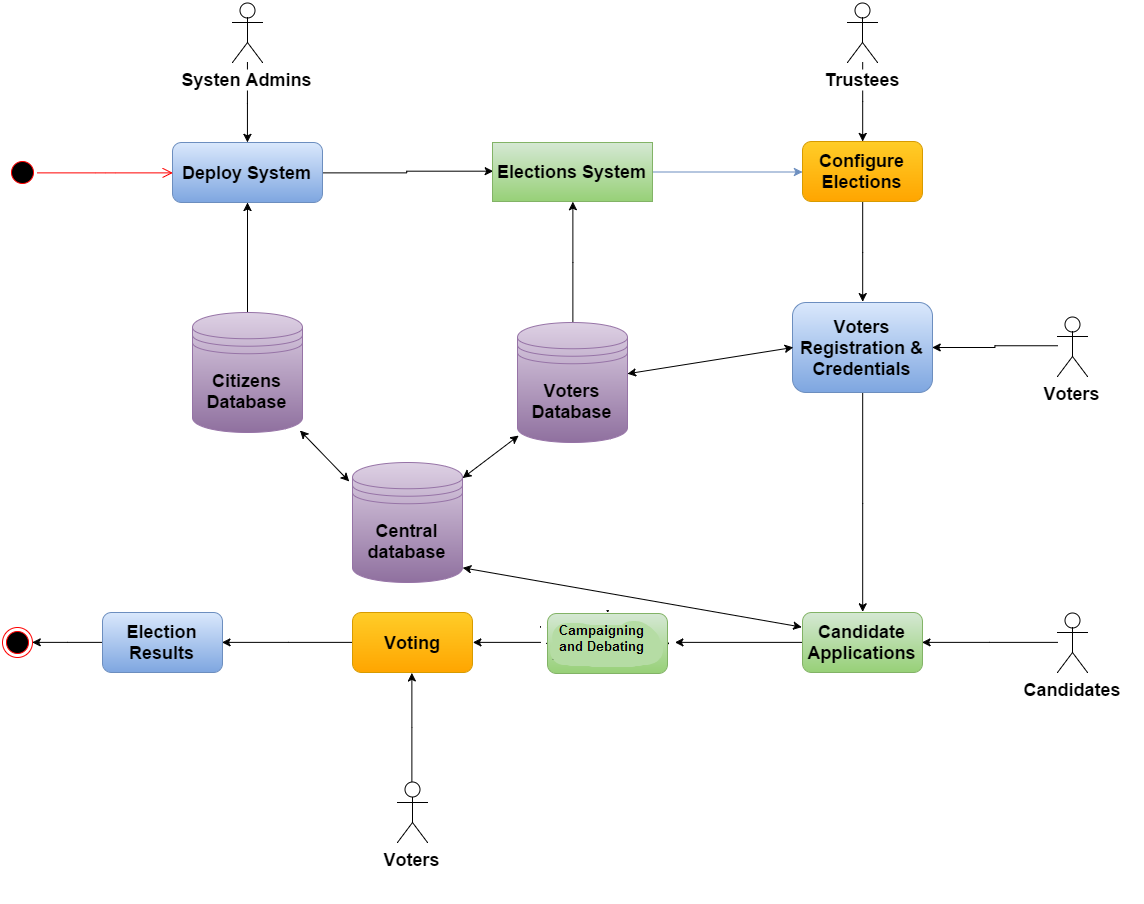
The system interacts with different external systems and a database to ease the work. Figure 2.1. is a context diagram that represents these relations and interactions in an abstract way with respect to EnV, our system.



**Figure 2.1: Context diagram for the system and its boundaries**

## 2.3. Process Model

Figure 2.2 demonstrates the main business process of the system. More details are provided in sequence diagrams for different important parts of the system.



**Figure 2.2: The process model of the system as a whole**

## 2.4. Product Functions

EnV will provide functions for elections officials to organize elections, configure the voting system and candidates’ eligibility, receive nominations, invite voters and organize debates between candidates. Details of these functionalities are listed and discussed in section 3 as well as the appendices.

### 2.3.1. Pre-Elections and Voters Registration

The aim of pre voting is to give candidates or parties, who wish to participate in the elections, the opportunity to register for the vacant places and compete in the elections. The registration process includes registration and eligibility check to make sure that the candidates are fit to fulfill the position’s mission and goals.

### 2.3.2. Debating and campaigning

This platform’s main goal is to allow candidates to publish content of their campaign to let voters know about them. This platform will also facilitate debates between different candidates that can be viewed by the voters.

### 2.3.3. Voting

This is the core functionality of the system. This module should provide a secure verifiable way of casting ballots. Each voter should be able to cast their ballots for the candidate they want. After the conclusion of the elections, the voters should be able to verify their vote. No votes can be duplicated without being detected.

## 2.4. User Classes and Characteristics

EnV is an online system that makes it possible to handle the campaigning, debating and actual elections online, and so the targeted actors are: public viewers, voters, candidates, system administrators, trustees and debate moderators.

All the users are expected to know how to read either Arabic or English as the system interface will offer both, and also they are expected to know how to deal with web browsers and electronic devices such as mobiles or personal computers since the system is an online system. Trustees, however require further training on how to keep confidential data secure. The following subsection provides more details about each user role.

### 2.4.1. Elections Trustees

Elections officials are responsible for setting the elections type, inserting the elections laws and protocols in the system and updating the eligible voters’ database before the elections. Trustees will feed eligibility information of both candidates and voters to the system and will deal with all legal lists and databases. In case of building a Helios [2] similar system, elections trustees are responsible for initiating the elections by providing the public keys and share the keys that decrypt the tallies.

### 2.4.2. System Administrators

A system administrator is a logged in user that creates, sets up, and updates the system platform. Every subsystem will have a different group of admins. System administrators are responsible for technical details of the system, backups, logs, maintenance and providing help to other user roles. The authority must be distributed between all system administrators. Sensitive functionalities require multi-user authentication.

### 2.4.3. Voters

A voter must be an eligible voter and his name must be in the list of eligible citizens to vote that will be provided by the government, otherwise he will not be able to sign up for the online voting.

### 2.4.5. Candidates

Candidates are the political parties or persons that will be running for the elections depending on whether the elections is list or individuals based. Candidates must be eligible to run for this elections, he must be able to handle his personal space and publish content online. He is expected to behave in ethical manners.

### 2.4.6. Public Viewers

A public viewer is a person or search engine that has internet access and is able to navigate to the system’s web address. He may not be logged in. Public viewers including search engines are viewers to the voting portal, campaigning and debating system who are not authenticated or do not have credentials.

### 2.4.7. Debate Moderators

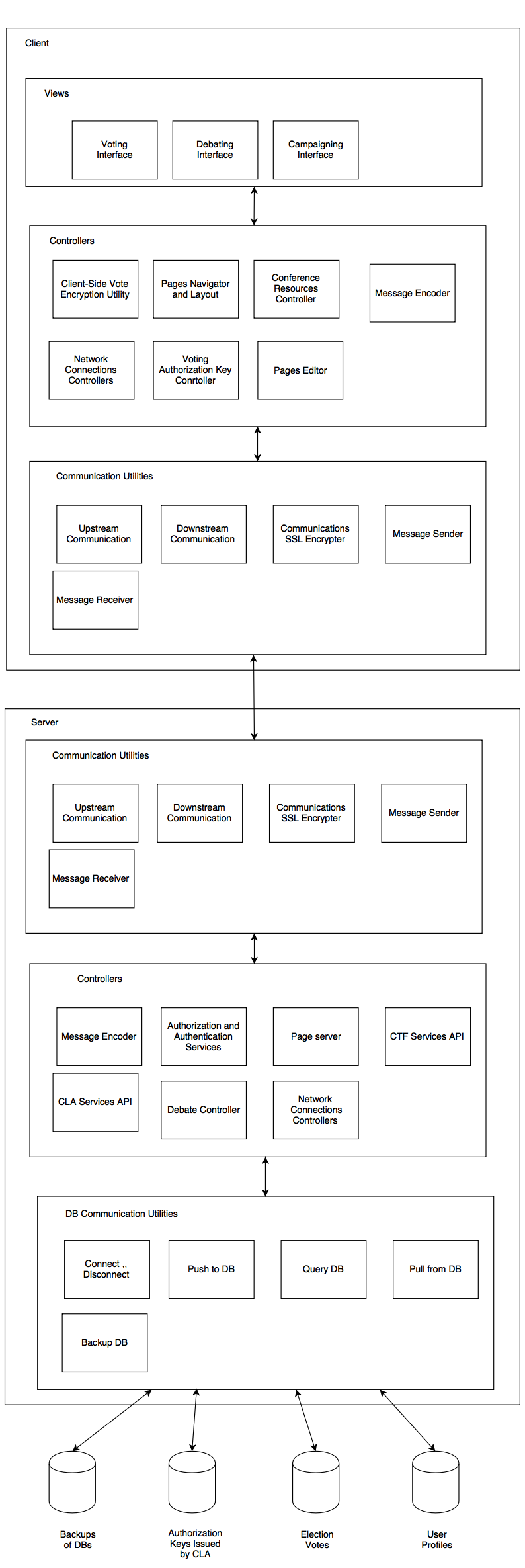
A debate moderator is a logged in user that represents a well-known public figure that is assigned by the system administrator prior to each debate. He is responsible for debating with the candidate(s) moderating the debate.

# System Architecture

## 3.1 Architectural Design

The diagram below describes main system components, and the way they are organized.

(Clearer diagram is in the folder provided with submission).

**3.2 Decomposition Description**

The software architecture is a hybrid between *client-server architecture, layered* and *MVC* patterns.

The client (voter or candidate) communicates with the server (our system), and then the server application communicates with the databases in the system.

The client side consists of:

1. (Topmost Layer) Views
   1. Voting Interface: The interface, buttons, and forms used in the voting and voting registration process.
   2. Debating Interface: The interface, buttons, and forms used in the debate rooms.
   3. Campaigning Interface: The interface, buttons, and forms used for:
      1. Candidates to post new content on their websites
      2. Users of the system to view candidate websites.
2. (Middle Layer) Controllers
   1. Client-Side Vote Encryption Utility: A cryptography component that is involved in encryption regarding votes.
   2. Pages Navigator and Layout Controller: A component that controls the layout of objects in the forms and websites viewed. In addition, it controls navigation requests from user interfaces.
   3. Conference Resources Controller: A component that controls the resources available for user to use in a debate. For example, mute/unmute microphone, etc.
   4. Message Encoder:
      1. Encodes all actions triggered by any other middle layer component to be sent to the server into appropriate messages to be sent.
      2. Receives all messages from the server into actions that can be forwarded to any other middle layer component.
   5. Network Connections Controller: A component that receives/sends connection requests.
   6. Voting Authorization Key Controller: A component that carefully handles logic in regards to the authorization key to be received from the CLA.
   7. Pages editor: Allows candidates to edit their pages on the website.
3. (Bottommost Layer) Communication Utilities
   1. Message Sender: Takes encoded messages from Message Encoder (above) and uses network technologies to send them to the server.
   2. Message Receiver: Receives messages from server through network and forwards them to the Message Encoder.
   3. Communications SSL Encryption Utility: A cryptography component that is involved in encryption regarding sending/receiving messages on network securely.
   4. Downstream communication: Utility that helps downstream content and media.
   5. Upstream communication Utility that helps upstream content and media.

The server side consists of:

1. (Topmost Layer) Communication Utilities: The same as client side.
2. (Middle Layer) Controllers
   1. Message Encoder : Serves same purpose as client message encoder
   2. Page server: Controls the serving of layouts, forms, pages, and their content.
   3. CTF Services API: Contains the tools, functions, and classes that controls the CTF
   4. CLA Services API: Contains the tools, functions, and classes that controls the CLA
   5. Network Connections Controller: Serves same purpose as client connections controller.
   6. Debate controller: Overseas the control of debate rooms
   7. Authentication and Authorization Controller
3. (Bottommost Layer) Database Communication Utilities: Database APIs that allow the server controller to communicate with its databases.
   1. Connect/Disconnect from DB
   2. Push to DB: push new update to DB
   3. Pull from DB: copies specified content from DB
   4. Query DB
   5. Backup DB

The databases needed by our system to function:

1. User profiles DB: This database contains records of all the system’s users and their roles and authorizations. It also holds (encrypted) information needed to authenticate the users.
2. Election Votes DB: This database contains records of all ballots that were casted in the elections
3. CLA Authorization Keys DB: This database contains records of all the keys issued by CLA during voting.
4. Backups of DB: Database of database backups.

**3.3 Design Rationale**

This hybrid architecture was drawn to accumulate all advantages to three unique architectural patterns:

1- Client-Server architecture:

* By default, the architecture is a client-server.
* Clients are the voters and candidates. Server(s) are the voting system.
* This architecture enables the functions to be distributed across the network (Internet)
* In addition, some functions, such as vote encryption must happen with no interaction with our system and therefore that needs a specialized function on the client side.

2-MVC architecture:

* The views lie within the client portion of the application.
* Then there are layer of controllers spread across both client and server.
* The model (databases described above) lie in the servers of our system.
* This architecture enables easier maintenance and development could be split easier.
* In addition, views require emphasis on ease of use. While internal components require more emphasis on security and data integrity. The separation then allows focus on different non-functional requirements.

3- Layered Architecture:

* The system is developed and deployed as layers
* This facilitates the communications in the system where the interfaces can be fixed and each layer only deals with the one below it, not caring about changes in other layers.

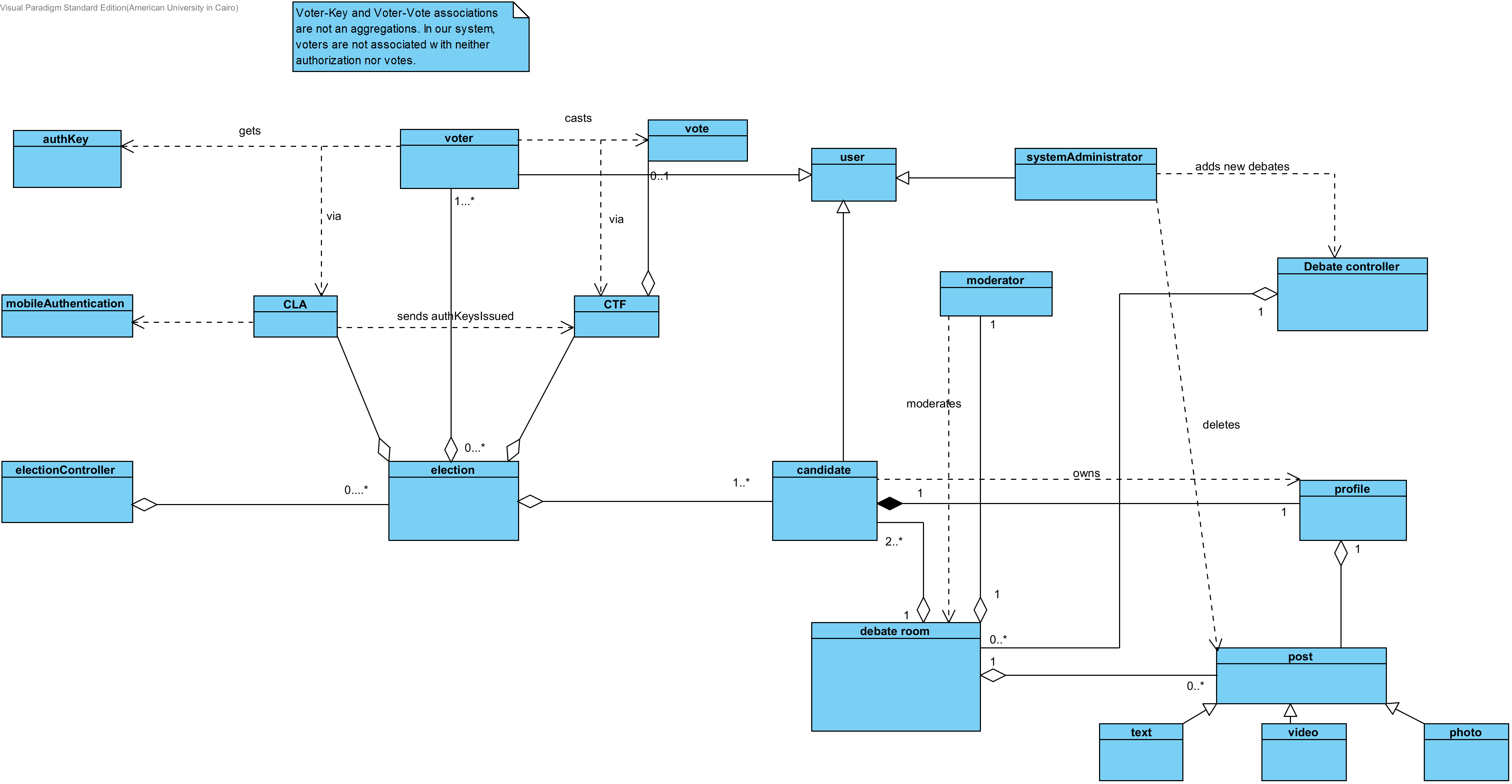
# 4. DATA DESIGN

## 4.1 Data Description

In the data design, we follow an Object-Oriented (OO) approach. In this subsection, class diagrams for different subsystems are provided. In section 4.2., tables are presented to explain each entity. At the end of this section, the whole system is wrapped in one class diagram to explain the big picture.

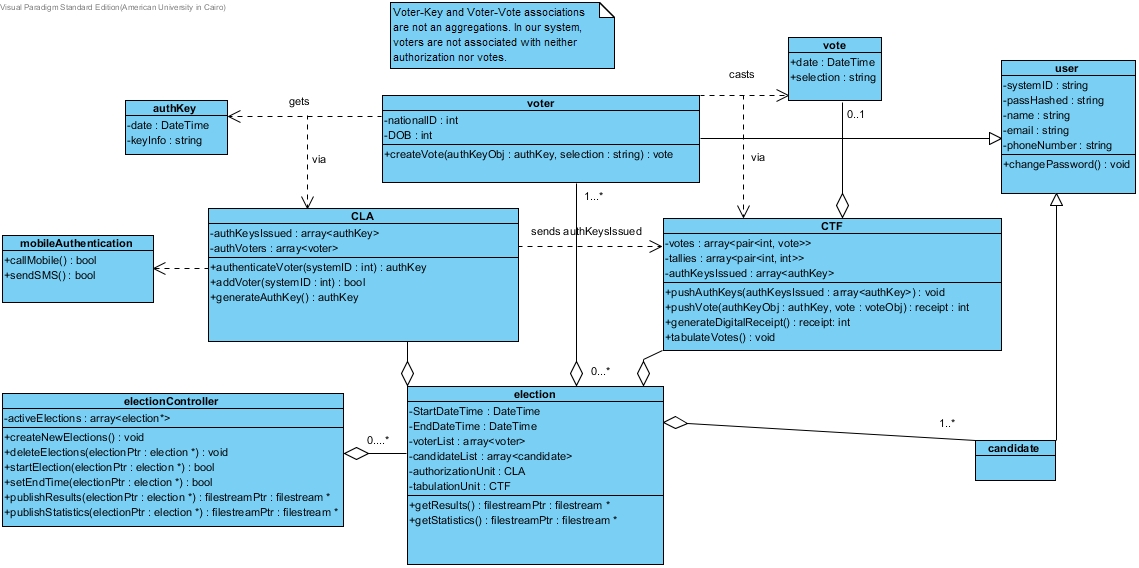
(Clearer diagrams are in folder provided with submission).

### 4.1.0 EnV System (The Voting and Campaigning/Debating Subsystems)

****

**Figure 4.1.0: EnV class diagram**

### 4.1.1 Voting Subsystem

****

**Figure 4.1.1: The voting system’s class diagram**

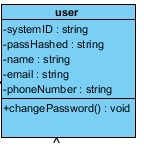
### 4.1.2Debating and Campaigning

**

***Figure 4.1.2: The debating system’s class diagram***

## 4.2 Data Dictionary

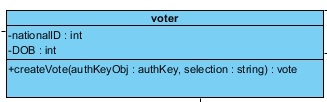
In this subsection, we list the structure including attributes and methods for each of the classes mentioned in section 4.1.



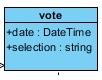
|  |  |
| --- | --- |
| **Class Name:** | User |
| **Brief Description:** | That is the basic class for any user in the system. This class is the one that any other type of user inherits from; the admin, the candidate and the voter. |
| **Attributes: (Fields):** | **Attribute Description:** |
| systemID : string | The ID of the user logging in to the system |
| passHashed : string | Hash of user password |
| email : string | The email of the user to which all details will be sent. |
| phoneNumber : string | The phone numbr of the user to which mobile authentication would take place using. |
| **Methods:** | **Methods description** |
| changePassword() : void | Function that starts password changing process. |
| AccessSystem() : bool | Decides whether the user has the right to log in through an account to the system or not, and decides the type of access if he can log in. |
| viewPrevious (Debate room, profile) | This function allows the user to browse and view the old posts on profiles, or see old debates stored on the system. |

G:\Dodjo Personal\Google Drive\Assignments\Fall 15\Software\Elections Project\SDD\Hussam\Voting Classes Pics\candidate.jpg

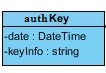
|  |  |
| --- | --- |
| **Class Name:** | Candidate |
| **Brief Description:** | This class represents the candidate in our system. This is not a voter class at all. In addition, it may represent individuals as well as parties. It inherits from “user”. |
| **Attributes: (Fields):** | **Attribute Description:** |
| profile : profile | The profile that the candidates creates for himself on the website. |
| Debate\_in\_action : Debate room | That is the debate room that the candidate is already using to debate with other candidate at the same time while users can see their debate live. |
| **Methods:** | **Method Description:** |
| Post\_on\_profile(posts) | The function that the candidate uses to post any content he desires on his profile on the website. |
| DeleteContent(posts) | The function that the candidate uses to delete any content he desires on his profile on the website. |
| getSlogan() : String | The function to get the slogan from the candidate. |
| setSlogan(slogan : String) : void | This function sets the slogan that the candidate has decided before on the profile. |
| getNumber() : integer | The function to get the number from the candidate. |
| setNumber(number : integer) : void | This function sets the number that the candidate has decided before on the profile. |
| getName() : String | The function to get the name from the candidate. |
| setName(name : String) : void | This function sets the name that the candidate has decided before on the profile. |
| getDebate\_in\_action() : Debate room | This function is used to create a debate room for the candidate to participate in. |
| setDebate\_in\_action(Debate\_in\_action : Debate room) : void | This function sets the debate room called debate\_in\_action for the candidate. |
| post\_in\_Debate() : bool | The function that allows the candidate to participate in debate, and returns whether it is possible or not. |
| payForAds\_onSocialMedia(Posts, string : slogan) | This function allows the candidate to post his posts on social media too and then charges him for that campaigning. |
| complain\_aboutOtherCandidate(Posts, System Administrator) | This function allows the candidate to complain about other candidates posts to the system administrator. |
| campaign\_allowed(System Administrator) : bool | This function returns whether campaigning is allowed by the system administrator or not at that time. |



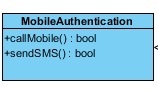
|  |  |
| --- | --- |
| **Class Name:** | voter |
| **Brief Description:** | Class for voter representation inside system. Inherits from “user” class. |
| **Attributes: (Fields):** | **Attribute Description:** |
| nationalID : int | National ID of voter |
| DOB : int | Date of Birth of voter |
| Favorite candidate : string | This attribute saves the voter’s favorite candidate in order to have his news all the time. |
| **Methods:** | **Methods description** |
| createVote (authKeyObj : authKey , selection : string) : vote | Essential function so that the voter can create a vote ready to be pushed to the CTF |
| shareOnSocialMedia( posts, candidate) | This method allows the voter to share anything related to a candidate from his profile to social media. |



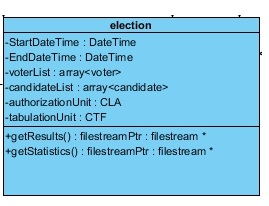
|  |  |
| --- | --- |
| **Class Name:** | vote |
| **Brief Description:** | Class for voter representation inside system. Inherits from “user” class. |
| **Attributes: (Fields):** | **Attribute Description:** |
| date : DateTIme | Date and time of vote creation |
| selection : string | Vote information about choice in election |



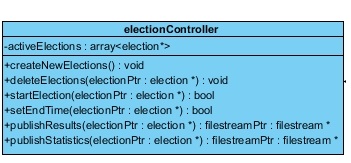
|  |  |
| --- | --- |
| **Class Name:** | authKey |
| **Brief Description:** | Class for authorization key representation inside system. |
| **Attributes: (Fields):** | **Attribute Description:** |
| date : DateTime | Date and time of key issuing |
| keyInfo : string | Information of the key |



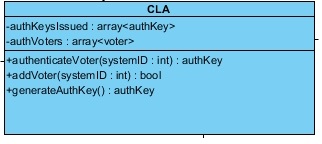
|  |  |
| --- | --- |
| **Class Name:** | mobileAuthentication |
| **Brief Description:** | Utility class for mobile authentication purposes |
| **Methods:** | **Methods description** |
| callMobile() : bool | Verification call to voter’s phoneNumber |
| sendSMS() : bool | Send verification SMS to voter’s phoneNumber |



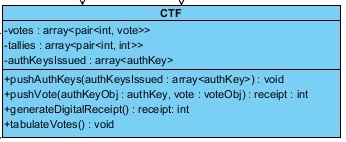
|  |  |
| --- | --- |
| **Class Name:** | election |
| **Brief Description:** | Class to represent an election inside the system |
| **Attributes: (Fields):** | **Attribute Description:** |
| StartDateTime : DateTime | The date and time of elections start |
| EndDateTime : DateTime | The date and time of elections end |
| voterList : array < voter ? | Array of eligible voters in an election |
| phoneNumber : string | The phone numbr of the user to which mobile authentication would take place using. |
| **Methods:** | **Methods description** |
| getResults () : filestreamPtr : filestream\* | Communciates with CTF to get results, returns a file output |
| getStatistics () : filestreamPtr : filestream\* | Creates statistics about elections, returns a file output |



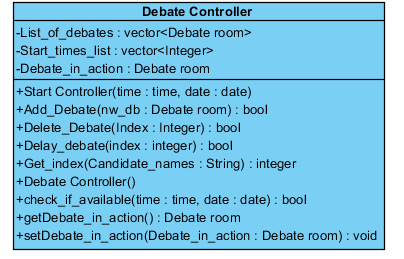
|  |  |
| --- | --- |
| **Class Name:** | electionController |
| **Brief Description:** | Class to represent elections controller. May control more than one concurrent election. |
| **Attributes: (Fields):** | **Attribute Description:** |
| activeElections : array <election\*> | Array of all active elections |
| **Methods:** | **Methods description** |
| creatNewElections() : void | Creates new elections |
| deleteElections ( electionPtr : election\*) : void | Delete a certain election |
| startElection ( electionPtr : election\*) : bool | Start a certain election |
| setEndTime ( electionPtr : election\*) : bool | Set the end time of a certain election |
| publishResults ( electionPtr : election\*) : filestreamPtr : filestream\* | Publish results of a certain election |
| publishStatistics ( electionPtr : election\*) : filestreamPtr : filestream\* | Publish statistics of a certain election |



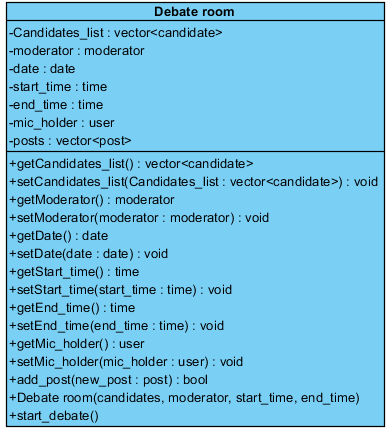
|  |  |
| --- | --- |
| **Class Name:** | CLA |
| **Brief Description:** | Central Legitimacy Agency (description as described in the document) |
| **Attributes: (Fields):** | **Attribute Description:** |
| authKeysIssued : array <authKey> | Array of all keys issues. |
| authVoters : array <voter > | Array of all voters who took keys. |
| **Methods:** | **Methods description** |
| authenticateVoter (systemId : string) : authKey | Essential function that authenticates a voter before voting and send him an authKey (if eligible to vote) |
| addVoter (systemID : int) : bool | Function that adds a certain voter to authVoters |
| generateAuthKey() : authKey | Utility function that generates a unique random authKey each invocation |



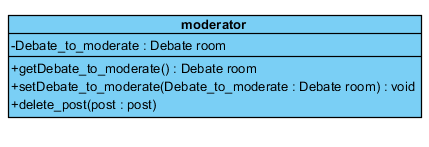
|  |  |
| --- | --- |
| **Class Name:** | CTF |
| **Brief Description:** | Central Tabulating Facility (description as described in the document) |
| **Attributes: (Fields):** | **Attribute Description:** |
| votes : array <pair <int, vote>> | Array of pairs of receipts and votes to store votes information before tallying |
| tallies : array < pair<int, int>> | Array of pairs of election choice number and number of votes for that choice |
| authKeysIssued : array <authKey> | same as CLA (actually copied from there) |
| **Methods:** | **Methods description** |
| pushAuthKey ( authKeysIssued : array <authKey> ) : void | Utility function that CLA uses to push the issued keys to the CTF |
| pushAuthKey ( authKeyObj : authKey , voteObj : vote) receipt : int | Voter uses this function to push his vote to the CF, return a digital receipt for the voter |
| generateDigitalReceipt () : receipt : int | Utility function that generates a unique random receipt each invocation |
| tabulateVotes() : void | Utility function that calculates tallies after voting is complete |



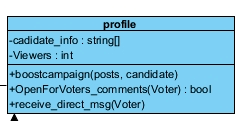
|  |  |
| --- | --- |
| **Class Name:** | Debate controller |
| **Brief Description:** | This class acts as a container for all the planned debates. It makes sure that no time clashes exist between debates and it will start and end the debates on the planned time without external intervention. Only the Admin can access this class. |
| **Attributes: (Fields):** | **Attribute Description:** |
| -List\_of\_debates : vector<Debate room> | A list of all created debates that will happen. |
| -Debate\_in\_action : Debate room | The Debate that is taking place in current time, only one debate can take place on a given time. |
| -Start\_times\_list : vector<Integer> | A sorted list of the start times of debates. |
| **Methods:** | **Method Description:** |
| +Add\_Debate(nw\_db : Debate room) : bool | The admin uses this method to add a new debate to the schedule. It takes a Debate room object as a parameter then checks if its time is free and if so, it adds it to the “List of debates” and the “start time list” then returns true, otherwise it returns false. |
| +Delete\_Debate(Index : Integer) : bool | The admin can delete a scheduled debate using this method. It takes the index of the debate to be deleted, checks if it exists, deletes it and returns true if found, returns false otherwise. |
| +Delay\_debate(index : integer) : bool | The admin can delay a scheduled debate using this method. It takes the index of the debate to be delayed, checks if it exists, checks that there is no time clash for the new time delays it and returns true if no clash exists, returns false otherwise. |
| +Get\_index(Candidate\_names : String) : integer | Returns the index of the debate given the name of candidates debating |
| +check\_if\_available(time : time, date : date) : bool | Checks if there is a debate scheduled in a given time and date, returns true if the time is free, false otherwise. |
| +setDebate\_in\_action(Debate\_in\_action : Debate room) : void | A setter for “Debate\_in\_action” called by the “start\_controller” method if the debate starts. |
| +Start Controller(time : time, date : date) | This method acts as an internal clock, it checks every minute if a debate should start now or not. If a debate should start, the method calls the “start debate” method of the Debate room object. The method has to be called once by the admin after adding all the debates and should take the current time and date as a parameter |



|  |  |
| --- | --- |
| **Class Name:** | Debate room |
| **Brief Description:** | This class models a Debate, its attributes, the methods that operates on them. |
| **Attributes: (Fields):** | **Attribute Description:** |
| -Candidates\_list : vector<candidate> | A list of the candidates debating. |
| -moderator : moderator | The moderator of this debate. |
| -date : date | The date for the debate. |
| -start\_time : time | The planned starting time of the debate. |
| -end\_time : time | The time in which the debate ends. |
| -mic\_holder : user | The user that can post on the debate in any given time. Only one user (candidate, moderator) can post at any given time. |
| -posts : vector<post> | a list of posts made by the users in this debate room. |
| **Methods:** | **Method Description:** |
| +getCandidates\_list() : vector<candidate> | Returns the candidate list in this debate room. |
| +setCandidates\_list(Candidates\_list : vector<candidate>) : void | This method is called to set the “candidate\_list” : the candidates will be debating in this room. |
| +getModerator() : moderator | Returns the moderator in this debate room. |
| +setModerator(moderator : moderator) : void | Used to set or change the moderator of the debate. |
| +getDate() : date | Returns the date of the debate. |
| +setDate(date : date) : void | Sets the date of the debate. |
| +getStart\_time() : time | Returns the starting time of this debate. |
| +setStart\_time(start\_time : time) : void | Sets the starting time of this debate. |
| +getEnd\_time() : time | Returns the end time of the debate. |
| +setEnd\_time(end\_time : time) : void | Sets the end time of the debate. |
| +getMic\_holder() : user | Returns the user that is allowed to post in that in a given time. |
| +setMic\_holder(mic\_holder : user) : void | Sets the user allowed to post in this debate room. |
| +add\_post(new\_post : post, caller :user) : bool | This method is called to post a new post (text, video or a photo) in this debate room. The method checks if this user is the mic holder, places the new post and returns true, otherwise it returns false and tells the user that he is not allowed to post at this moment. |
| + view() : vector<post> | Returns the posts that have been posted in the debate room. |
| +start\_debate() | The trigger for starting the debate, it calls “setDebate\_in\_action” for the candidates in this room and “setDebate\_to\_moderate” for the moderator to give them access to the room. |



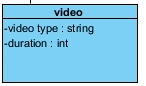
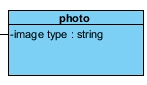
|  |  |
| --- | --- |
| **Class Name:** | moderator |
| **Brief Description:** | This class models the moderator for a debate, his attributes and methods. |
| **Attributes: (Fields):** | **Attribute Description:** |
| Debate\_to\_moderate : Debate room | The debate room in which this moderator is moderating. |
| **Methods:** | **Method Description:** |
| getDebate\_to\_moderate() : Debate room | Returns the debate room in which he is a moderator. |
| setDebate\_to\_moderate(Debate\_to\_moderate : Debate room) : void | Sets debate room in which he is a moderator. This method is called by “start\_debate” in the debate room class. |
| Delete\_post(post : post) | The moderator can delete a post in the debate room using this method. |



|  |  |
| --- | --- |
| **Class Name:** | Profile |
| **Brief Description:** | That is the class controlling the candidate’s profile |
| **Attributes: (Fields):** | **Attribute Description:** |
| Candidate\_info: string[] | This array of strings include all the information that the candidate wishes to post about himself on the website. |
| Viewers : int | That is the number of viewers and visitors who accessed the profile to know more about the candidate. |
| **Methods:** | **Methods Description** |
| getCandidate\_info(candidate) : string [] | This function is responsible for getting all the information of the candidate who owns the profile. |
| setCandidate\_info(candidate\_info : string []) : void | That is the function used to the data that the candidate has already given about himself. |
| boostcampaign(posts, candidate) | That is a function used by the user to boost all his post once at a time on the website and social media if he deserves, for increased exposure to voters. |
| OpenForVoters\_comments(Voter) : bool | This function indicates whether voters can comment or not |
| receive\_direct\_msg(Voter) | This function allows voters to send direct messages to candidates directly through their profiles. |

G:\Dodjo Personal\Google Drive\Assignments\Fall 15\Software\Elections Project\SDD\Final Revision\Diagrams\Debating Classes Pics\post.PNG

|  |  |
| --- | --- |
| **Class Name:** | Post |
| **Brief Description:** | The Posts class is responsible for the posts that will be posted by the candidate on his profile, different types of posts as text, video and images inherit from it. |
| **Attributes: (Fields):** | **Attribute Description:** |
| Topics related : string[] | This array includes all the topics that a single post would contain or all people he will mention. |
| Candidates mentioned : string[] | The unique number of the candidate used in the elections. |

G:\Dodjo Personal\Google Drive\Assignments\Fall 15\Software\Elections Project\SDD\Final Revision\Diagrams\Debating Classes Pics\text.PNG  

|  |  |
| --- | --- |
| **Class Name:** | Text |
| **Brief Description:** | The class text is a type of post that inherits from the class posts all its features. |
| **Attributes: (Fields):** | **Attribute Description:** |
| Refrences : string[] | This array includes all the references the text has included. |

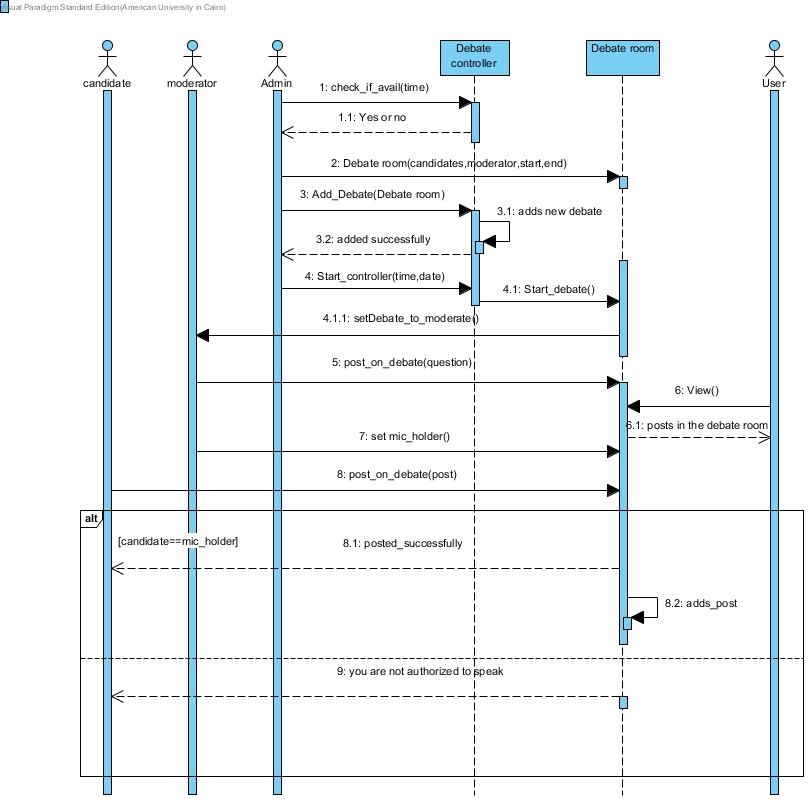
|  |  |
| --- | --- |
| **Class Name:** | Video |
| **Brief Description:** | The class video is a type of post that inherits from the class posts all its features. |
| **Attributes: (Fields):** | **Attribute Description:** |
| Video type : string | This attribute stores the type of the video. |
| Video duration : int | The attribute that stores the duration of the video. |

|  |  |
| --- | --- |
| **Class Name:** | Photo |
| **Brief Description:** | The class photos is a type of post that inherits from the class posts all its features. |
| **Attributes: (Fields):** | **Attribute Description:** |
| Image type : string | This attribute stores the type of the image. |

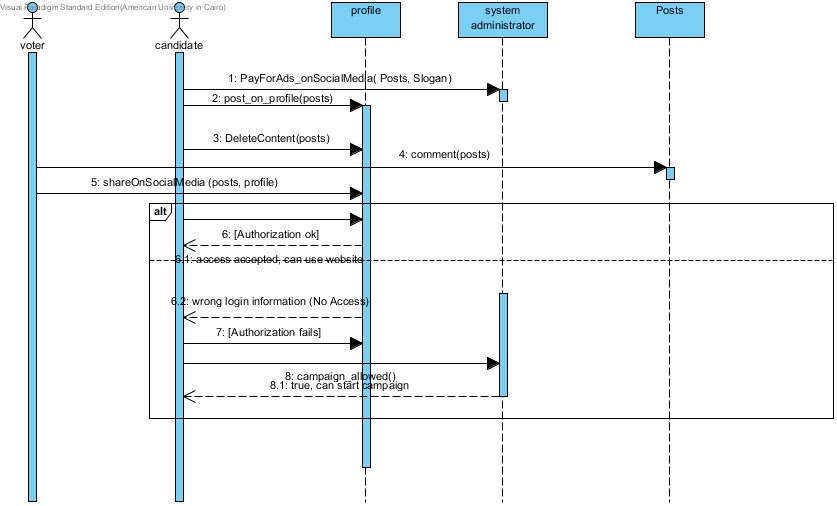
# 5. Dynamic Model

This section demonstrates the interaction between system’s components. We illustrate the dynamic interaction using the sequence diagrams below.

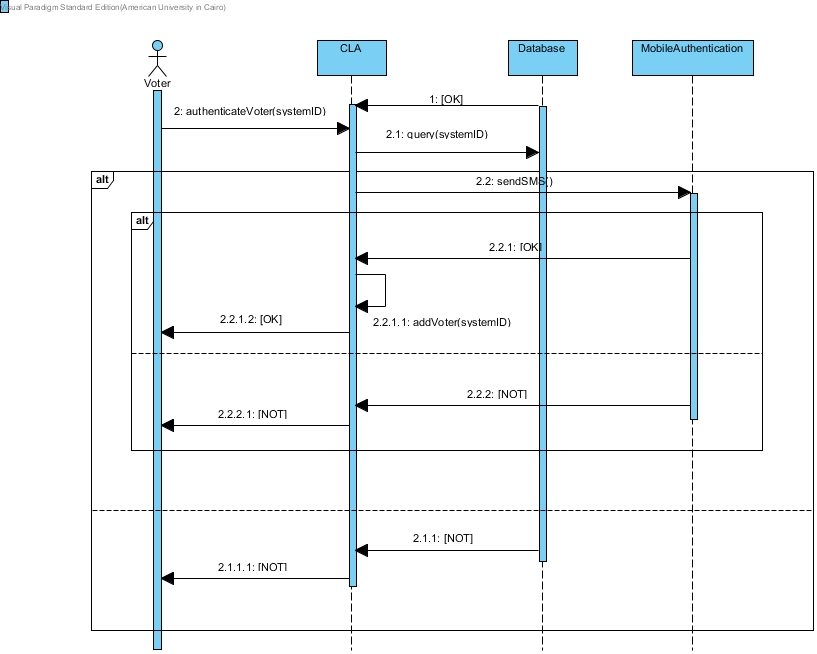
## 5.1. Sequence Diagrams



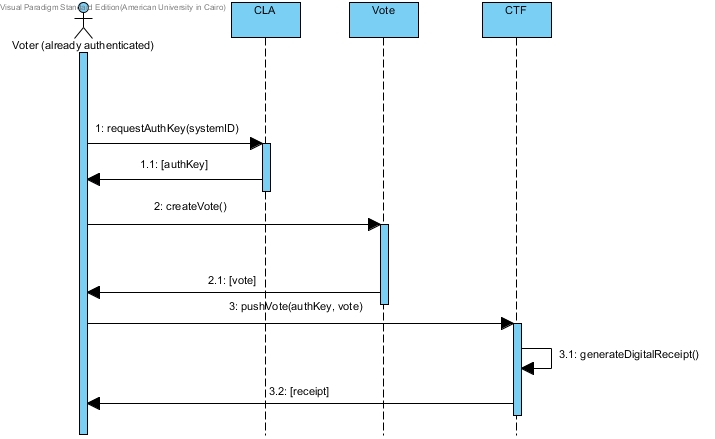
**Figure 5.1: Sequence diagram for the debating process**

****

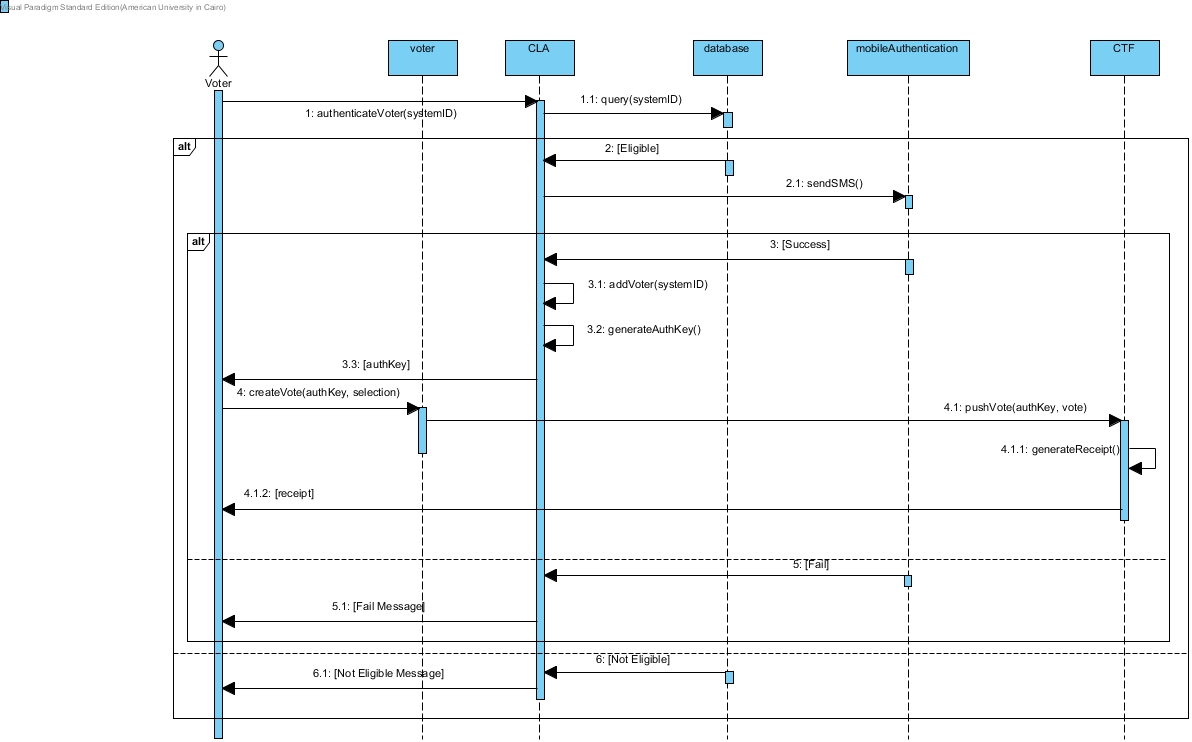
**Figure 5.2: Sequence Diagram for the campaigning process**

****

**Figure 5.3: The authentication process before the voting**

****

**Figure 5.4: Overview of the voting process**

****

**Figure 5.5: Detailed explanation of the voting process**

# 6. Human Interface Design

## 6.1 Overview of User Interface

The user interface in this system is very important because it will allow the voters, candidates and trustees to interact with the system to do the most important functionalities. In this section, we divide the UI into five sections; voting, debating, campaigning, accounting and configuration. In this subsection, we explain the interface benefits for each of the system users.  
  
**Note:  
The interface prototype has been provided more functions and is better viewed through these two links.**

[**https://moqups.com/islamfm@aucegypt.edu/ZNnbnCcQ/**](https://moqups.com/islamfm@aucegypt.edu/ZNnbnCcQ/)

[**https://moqups.com/islamfaisal@i2msoft.com/3CXSPvNG/**](https://moqups.com/islamfaisal@i2msoft.com/3CXSPvNG/)

**Remark 6.1.1.** We omit a user interface for the system admins here. System admins will use a combination of the operating system’s environment and an already existing DBMS.

### 6.1.1. Voter

The voter should be able to use the interface to register, vote and verify the elections results. The voter should also be able to view the contents of debates and campaigns.

### 6.1.2. Candidate

The candidate should be able to use the interface to publish content related to their campaign, join and participate in debates.

### 6.1.3. Trustee

The trustee’s UI should enable them to setup elections and configure their type, starting and ending time.

### 6.1.4. Debate Moderator

The debate moderator’s UI should be enable them to join and manage properly a debate session by granting resources such as mic, camera, receive questions from the public and block users.

## 6.2 Screen Images

This section contains an outline of the expected UI for each module of the system.

### 6.2.1. Login

Figures 6.1 and 6.2 demonstrate the 2-factor authentication process.

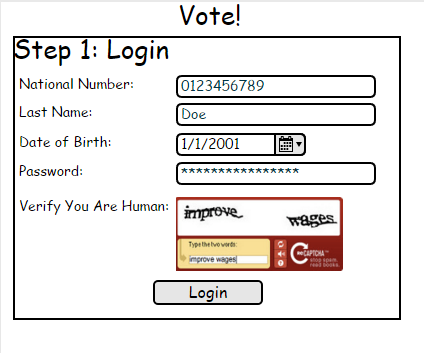


Figure 6.1: The authentication screen

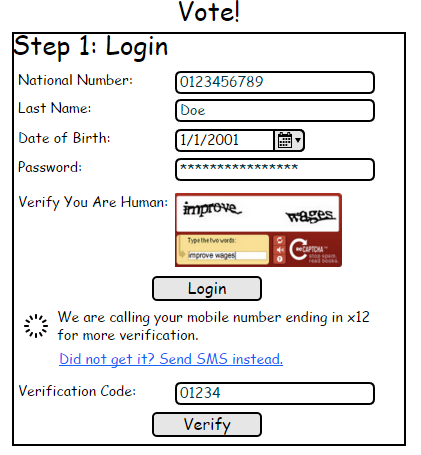


Figure 6.2: The prompt for mobile-authentication screen

### 6.2.2. Voting

This is the most essential and vital part of the system. This is the part where the voter (our main user) will interact with the system to cast their vote. Figures 6.3 through 6.5 provide a glimpse of how the voting process will look like.

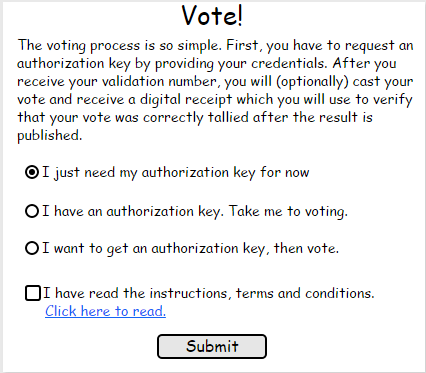


Figure 6.3: The first step in the voting process

To request an authorization key, the user needs to use the login interface in figure 6.2.

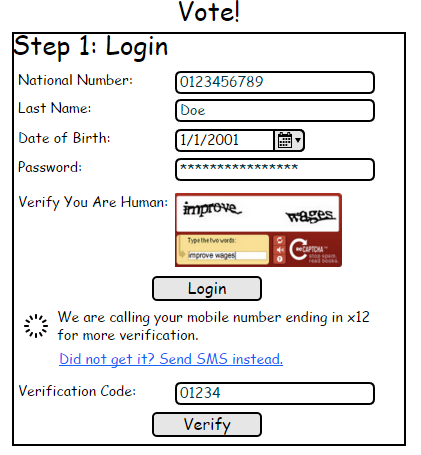
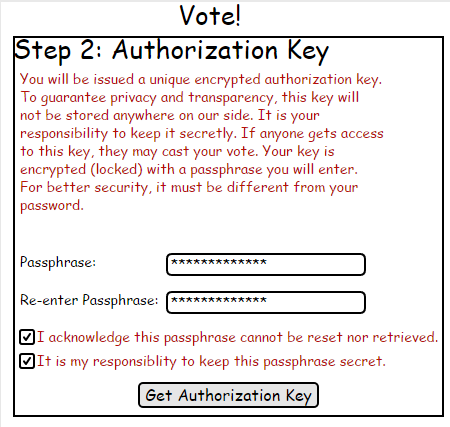


Figure 6.2: The prompt for mobile-authentication screen

After that the user enters a passphrase to encrypt his authorization key.

**Figure 6.4: The prompt for encryption passphrase**

After that depending on the given authorization information, the user is redirected to a a page to tell him whether he made a successful authorization.

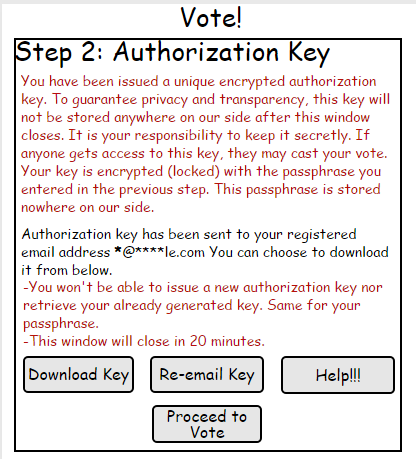
Figure 6.5: A successful authorization notification



Figure 6.6: A failed authorization attempt

Figure 6.7: The voting form where the user casts their vote

Finally, the user gets a digital receipt that he can use to check and verify that his vote was tallied correctly.

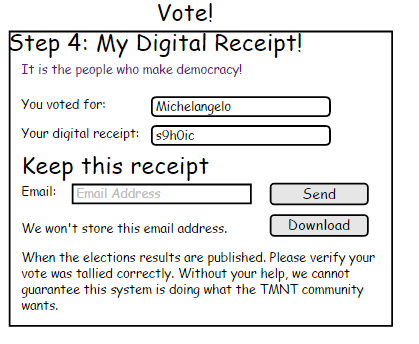


Figure 6.8: The digital receipt that the voter receives after successful voting

Finally, the user after the election ends can view the election results and verify his vote was tallied.

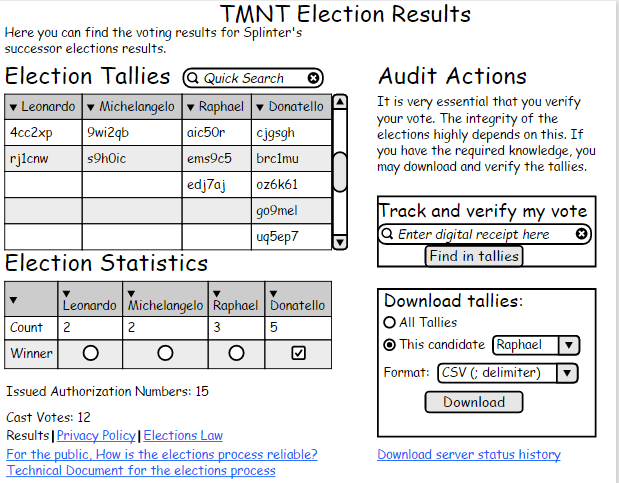
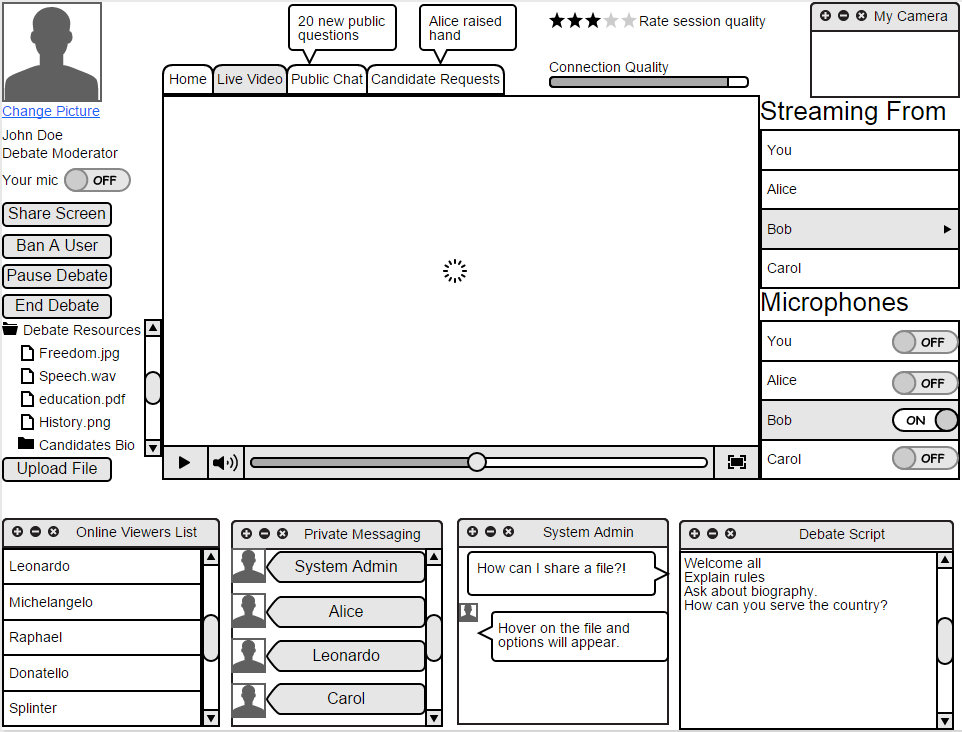


Figure 6.9: The election results viewed publicly so that the voter can verify his vote

### 6.2.3. Debating

****

**Figure 6.10: The debating screen as seen by the moderator**

## 6.3 Screen Objects and Actions

In this section, a description table is given for the important interface elements and controls in the figures outlined in section 6.2.

Table 6.1.: Description of the interface elements in Figure 6.1

|  |  |  |
| --- | --- | --- |
| Group of Element(s) | Type | Description/Action |
| National Number | Label + Textbox | The voter’s national number as in legal documents and retrieved from the citizens database |
| Last Name | Label + Textbox | The user’s last name for more verification and avoid lucky national number guesses |
| Date of Birth | Label + Textbox | The user’s date of birth |
| Password | Label + Textbox | The password the user uses to login into the system.  The password field is masked by asterix. |
| Human Verification | CAPTCHA (Label + Image + Textbix + Audio) | A CAPTCHA is used here where the user is required to enter it as seen. If the user is disabled, they can request to answer this challenge in another way such as audio. |
| Submit Button | Button | The trigger to send all the information provided for authentication |

Table 6.2.: Description of the interface elements in Figure 6.2

|  |  |  |
| --- | --- | --- |
| Group of Element(s) | Type | Description/Action |
| National Number | Label + Textbox | The voter’s national number as in legal documents and retrieved from the citizens database |
| Last Name | Label + Textbox | The user’s last name for more verification and avoid lucky national number guesses |
| Date of Birth | Label + Textbox | The user’s date of birth |
| Password | Label + Textbox | The password the user uses to login into the system.  The password field is masked by asterix. |
| Human Verification | CAPTCHA (Label + Image + Textbix + Audio) | A CAPTCHA is used here where the user is required to enter it as seen. If the user is disabled, they can request to answer this challenge in another way such as audio. |
| Submit Button | Button | The trigger to send all the information provided for authentication |
| Send SMS instead | Hyperlink (or button) | Lets the user receives a message instead of a call to complete the 2nd factor authentication |
| Verification Code | Label + Textbox | A textbox to let the user enters the verification code received on mobile |
| Verify | Button | The button used to submit the verification code for checking. |

Table 6.3.: Description of the interface elements in Figure 6.3

|  |  |  |
| --- | --- | --- |
| Group of Element(s) | Type | Description/Action |
| Navigation options | Multiple Radio buttons | Allows the voter to navigate to where he wants. The user can choose to generate an authorization key, go to voting or do both in sequence |
| Terms and Conditions | Checkbox | The user marks this to express his agreement to the terms, conditions and laws of the service. |
| Submit | Button | Takes the user to his determined choice |

Table 6.4.: Description of the interface elements in Figure 6.4

|  |  |  |
| --- | --- | --- |
| Group of Element(s) | Type | Description/Action |
| Passphrase | Label + Textbox | The passphrase used to encrypt the user’s authorization number. |
| Passphrase confirmation | Label + Textbox | The user is requested to re-enter their information for double checking. |
| Terms and conditions | 2 checkboxes |  |

Table 6.6.: Description of the interface elements in Figure 6.5

|  |  |  |
| --- | --- | --- |
| Group of Element(s) | Type | Description/Action |
| Download Key | Button | Lets the user download the encrypted authorization key to his local machine |
| Re-email key | Button | Sends again the encrypted authorization key to the email address so that the user can use it later |
| Help | Button | Request help by navigating to FAQ or a live help agent |
| Proceed to Vote | Button | Takes the user to cast their vote directly |

Table 6.6.: Description of the interface elements in Figure 6.6

|  |  |  |
| --- | --- | --- |
| Group of Element(s) | Type | Description/Action |
| Message | Paragraph | A message to the user indicating that his authorization failed |
| Help | Hyperlink | Request help by navigating to FAQ or a live help agent |
| Issue authorization key | Hyperlink | Navigates to the page where the user can issue an authorization key |

Table 6.7: Description of the interface elements in Figure 6.7

|  |  |  |
| --- | --- | --- |
| Group of Element(s) | Type | Description/Action |
| Acknowledgment | Checkbox | To make sure the user is aware that the voting process can only be done once. |
| Election Question | Label + Multiple Radio buttons | To capture the voter’s answer to the election question(s) |
| Help with authorization key | Hyperlink | Navigates to help and information about the authorization key. |
| Authorization | Textbox | The path to the authorization file of the user |
| Browse | Button | Opens a file browsing dialog to get the file |
| Passphrase | Textbox | Passphrase for encryption of the authorization key if it is encrypted |
| Decryption-By-User option | Checkbox + Radio button | Allows the user to send a decrypted authorization key |
| Human Verification | CAPTCHA (Label + Image + Textbix + Audio) | A CAPTCHA is used here where the user is required to enter it as seen. If the user is disabled, they can request to answer this challenge in another way such as audio. |
| Disability Help | Hyperlink | Help with disability issues in casting the vote |

Table 6.8.: Description of the interface elements in Figure 6.8

|  |  |  |
| --- | --- | --- |
| Group of Element(s) | Type | Description/Action |
| Voted for | Lable + Non-editable textbox | The selection of the voters that has been tabulated |
| Digital Receipt | Lable + Non-editable textbox | The unique digital receipt that the user can use after the elections to verify his vote |
| E-mail Address | Textbox | The email address to send the digital receipt to.  ***This email must not be tracked into the system*** |
| Send | Button | Sends the digital receipt to this email address |
| Download | Button | Download the digital receipt |