**CSCI-6908 - Individual Research**

**CERTUS Voting**

**Design Document**

**Supervisor:** ProfessorMichael Clarkson

**Team:**

|  |  |  |
| --- | --- | --- |
| Ahmad Kouraiem | [ahmadko@gmail.com](mailto:ahmadko@gmail.com) | 703-477-8333 |
| Dmitriy Karmazin | [dkarmazi@gwu.edu](mailto:dkarmazi@gwu.edu) | 202-758-6826 |
| Hirosh Wickramasuriya | [hirosh@gwmail.gwu.edu](mailto:hirosh@gwmail.gwu.edu) | 202-594-5448 |
| Steven Frink | [sfrink1@gmail.com](mailto:sfrink1@gmail.com) | 518-478-5880 |

Contents

[**1. Application Architecture** 4](#_Toc386699652)

[1.1. High Level Abstract 4](#_Toc386699653)

[1.2. Low Level Abstract 5](#_Toc386699654)

[1.3. RMI over SSL 7](#_Toc386699655)

[**2. Design Decisions** 8](#_Toc386699656)

[2.1. HTML Framework 8](#_Toc386699657)

[2.2. Structure of Projects 8](#_Toc386699658)

[2.3. Application Configurations 8](#_Toc386699659)

[2.3.1. Server Configuration Properties 8](#_Toc386699660)

[2.3.2. Web Configuration Properties 9](#_Toc386699661)

[2.4. RMI initialization at runtime 10](#_Toc386699662)

[2.5. DB reconnection at runtime 10](#_Toc386699663)

[**3. Core Functionality and Business Logic** 11](#_Toc386699664)

[3.1. User Management 11](#_Toc386699665)

[3.1.1. User types 11](#_Toc386699666)

[3.1.2 New user registration 11](#_Toc386699667)

[3.1.3. Edit Profile 11](#_Toc386699668)

[3.1.4. User Key Management 11](#_Toc386699669)

[3.1.5 Forgot password 12](#_Toc386699670)

[3.1.6. Blocking Users 12](#_Toc386699671)

[3.2. Election management 13](#_Toc386699672)

[3.2.1. Adding new election 13](#_Toc386699673)

[3.2.2 Edit elections 13](#_Toc386699674)

[3.2.3. Public Elections 13](#_Toc386699675)

[3.2.4. Private Elections 14](#_Toc386699676)

[3.2.5. States of Elections 14](#_Toc386699677)

[3.2.6. Election Actions 15](#_Toc386699678)

[3.3. Voting and double envelope protocol 16](#_Toc386699679)

[3.4. Tally and publish election 16](#_Toc386699680)

[**4. Reference Monitor** 17](#_Toc386699681)

[4.1. User Roles: 17](#_Toc386699682)

[4.2. Reference Monitor Architect: 18](#_Toc386699683)

[4.2.1. Rights Group 0: 18](#_Toc386699684)

[4.2.2. Rights Group 1: (operations on the requester’s data) 18](#_Toc386699685)

[4.2.3. Rights Group 2: (operations for election authorities or admins) 18](#_Toc386699686)

[4.2.4. Vote Right: 18](#_Toc386699687)

[4.2.5. View Results Right: 18](#_Toc386699688)

[4.3. Remote Methods Permissions: 19](#_Toc386699689)

[**5. Database Model** 21](#_Toc386699690)

# **1. Application Architecture**

## 

## 1.1. High Level Abstract

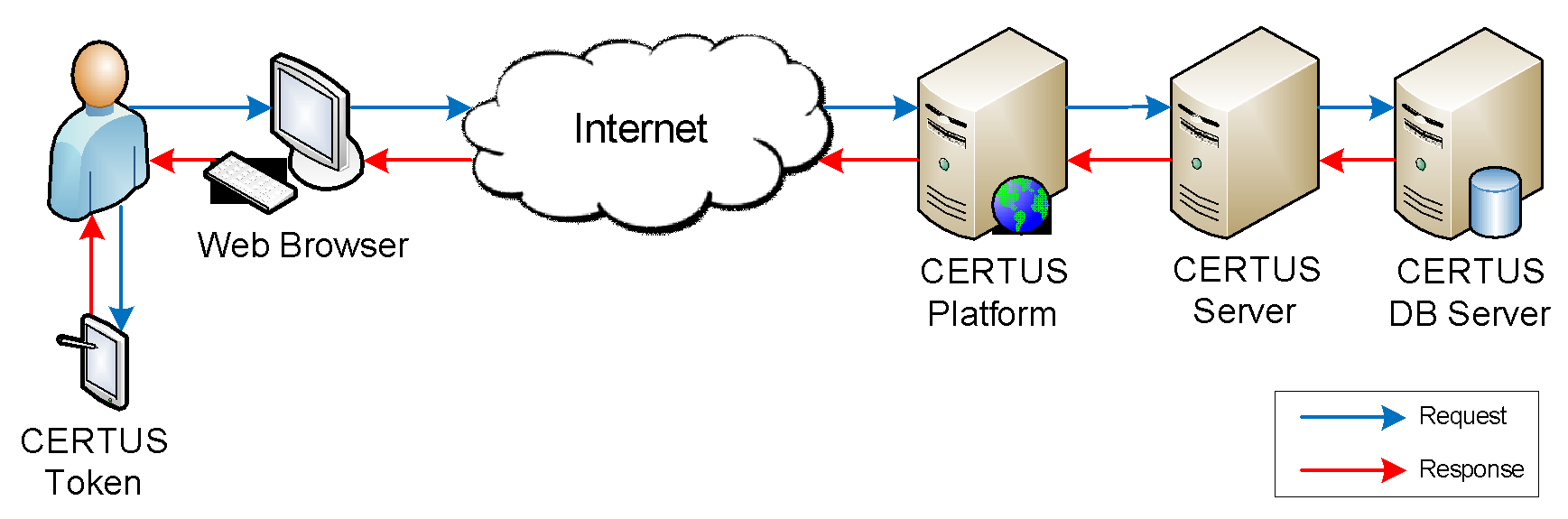


Figure 1 - CERTUS High Level Abstract

**System Main Components:**

* **CERTUS DB Server:**

On this server we host the needed database for CERTUS system. This server should not be published on the internet, and should not be accessible by any principal except the CERTUS Server.

* **CERTUS Server:**

On this server we run the CERTUS Server which receives all of the remote methods from any public platform. This server is the RMI server, which listens on port 2019 for incoming RMI connections, and provides an RMI Interface which contains all the available methods to be invoked by a public client. Before executing any method, this server checks with the Reference Monitor if the requester has the right to invoke the method. This server holds the main system business logic and acts as a proxy between the clients and the database.

* **CERTUS Platform:**

On this server we will run CERTUS Platform implementation, which is a JSP web application. Certus Platform can also be implemented with any other technology as long as it supports RMI. The CERTUS Server has an ability to support more than one active platform. All connecting platforms would be working with the same database through the server. To develop a new platform, the developing team needs a copy of the CERTUS Server RMI Interface, to know the available remote methods and the output for each method. Once the web application is launched, it tries to establish a RMI connection with the CERTUS Server over SSL and retrieve a copy of RMI Interface.java in order to get a list of all the available remote methods which can be invoked by the platform.

* **CERTUS Token:**

CERTUS Token is a hardware token used by CERTUS users to digitally sign their encrypted votes before casting. Each token should have a unique private key used by the user to be able to sign. The user should share the public key with CERTUS Server so that the server can verify the votes for each user before tallying.

For this project, we developed software to simulate the functionality of the hardware token where the users can sign their encrypted votes before casting, and attach that signature along with the encrypted vote on the CERTUS platform.

## 1.2. Low Level Abstract

## 

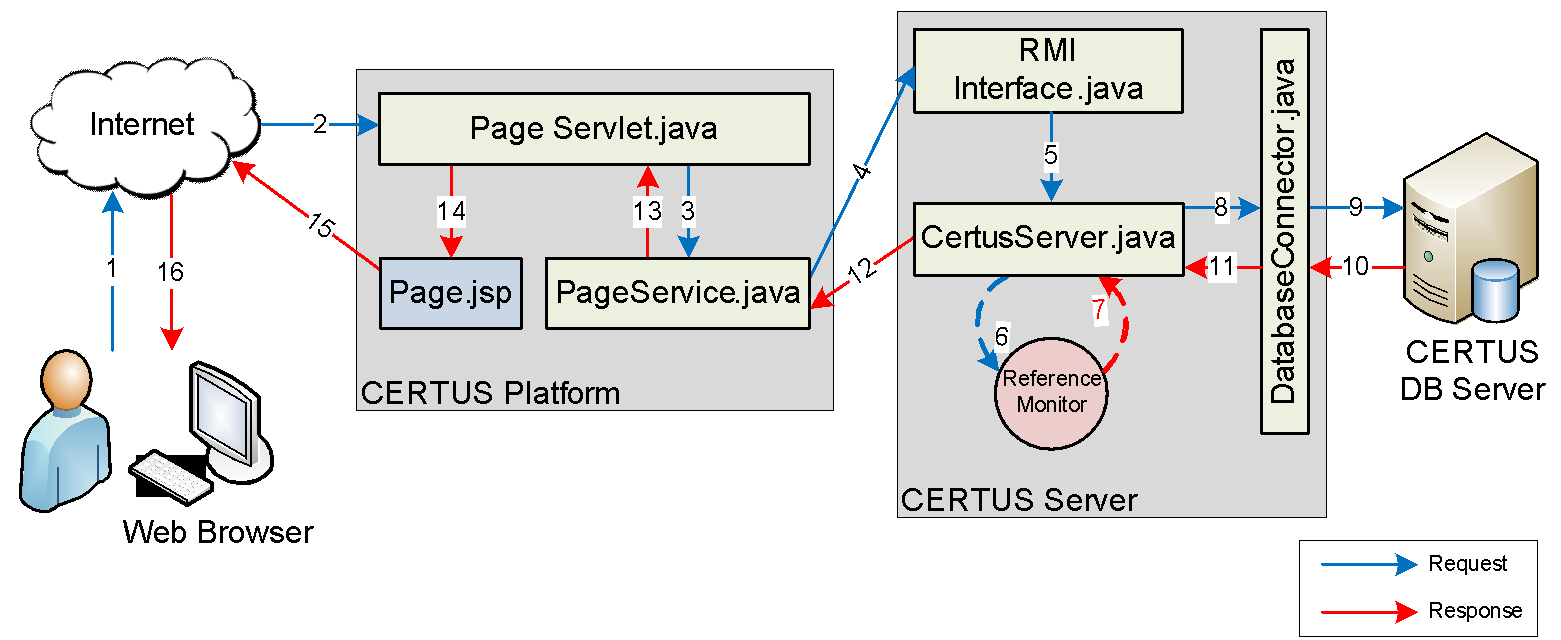


Figure 2 - Low Level Abstract

The best way to explain the low level abstract of the system is to go through the necessary steps to invoke a remote method that accesses the DB by a system user using CERTUS Platform (Figure 2):

Step 1: Suppose a user would like to login. He/she opens CERTUS login page, provides the required credentials, and hits the login button. These credentials go to the CERTUS Server, which then accesses the DB server to check the provided credentials against the stored hash values. All information submitted by the user will be transmitted to the CERTUS Platform using HTTPS.

Step 2: Each HTTP request (get or post) will be handled by a dedicated servlet. Each servlet can be accessed by unique URL suffix. Servlets typically have 2 standard methods: doPost and doGet, which process post and get requests respectively. Servlets are responsible for handling logical tasks and dynamic generation of HTML output which is then passed to to be displayed on corresponding JSP page. Servlets also make calls to methods to services, which have the same naming as servlets in services package, in order to invoke RMI functionality. In MVC paradigm, servlets represent controllers, jsp files represent views, and objects with suffix “Dto” represent data transfer objects, known as models.

Step 3: The servlet page invokes a method from the page service based on the client request and passes all the necessary parameters to the server.

Step 4: Once the service method is invoked, it will try to invoke the required remote method from the available remote methods provided by the RMI interface at the CERTUS Server.

Steps 5, 6, 7: The implementation of each remote method is written in CertusServer.java, and the implementation in this class invokes methods in DatabaseConnector.java where the actual business logic and communication with database takes place. Before invoking the DatabaseConnector.java methods, CertusServer.java checks with the Reference Monitor to make sure that the requester is allowed to perform the requested action. If the requester is not allowed to invoke the function, it will receive a “permission denied” message, and otherwise the requester is permitted to invoke the method, and the request will be passed to the DatabaseConnector.java.

Step 8: DatabaseConnector.java is the largest class we have. It contains the entire backend business logic for this project, and is the only class which queries the CERTUS DB Server.

Steps 9, 10: DatabaseConnector.java accesses the database and runs requested queries.

Step 11: DatabaseConnector.java method prepares the results of the query and sends them back to CertusServer.java.

Step 12: CertusServer.java forwards the results directly back to PageService.java, which invoked the remote method in the first place.

Step 13: PageService.java forwards the results back to the PageServlet.java.

Step 14: PageServlet.java receives the results and prepares dynamic HTML code to display the results and forwards the HTML code to the related JSP page.

Step 15: The JSP page is rendered at the user’s web browser to display the results.

All the data is transferred between application layers using Validator objects which represent the following set of fields:

NOTE on VALIDATORS:

1. boolean isVerified - true/false value of the requested action, will be set to false if action could not be completed for any possible reason (failed call to database, failed call to RMI server, denied permissions, and etc.).

2. String status - contains the actual error message which explains what exactly could not be processed in case of an error.

3. Object object - a universal object that can pass any additional variables needed for requests (including models and sets of models).

For example, a successful request to select all users from database would return the following Validator: true, “Users selected”, Arraylist<Users>. Failed request would return Validator: false, “Invalid query...”, null.

## 1.3. RMI over SSL

Running CERTUS-Server follows the following sequence on actions:

1. Reserve a port and listen on any RMI connection established on that port. For this implementation of the project we are using Port 2019.
2. Read the policy file for the server, and load all the security entries in the file. For this implementation of the project, the policy file can be found under “CERTUS-Server → resources → rmi-files → policy”. This file tells the server who is permitted to establish RMI connections with the server, and who is not.
3. Create a security manager, which is responsible for enforcing the security policy.
4. Create the RMI registry and bind an object instance to it so all permitted clients can get a copy of the RMI Method interfaces. Another important thing to be bound to the RMI registry is the server’s public certificate, which will be used later by the client to establish a secure RMI connection over SSL. This certificate, along with the server private key, is stored in a keystore called CertusKeyStore which can be found under “CERTUS-Server → resources → rmi-files → CertusKeyStore”. The password to access that keystore can be found in config.properties file (CETUS-Server → src → config.properties).
5. Listen for RMI connections.

Whenever we run CERTUS-Platform, it will be performing the following:

1. In order to establish a secure RMI connection with the server, the client will get the server certificate to get the public key and establish an SSL connection. The client should trust this certificate, and in order to do that, the client has a keystore contains all the trusted certificates by the client, and whenever the clients receive any certificate from the RMI server, it checks if this certificate is trusted, by looking in this protected key store. For the implementation of this project, Certus Platform is using a keystore called ClientTrustedCerts, which can be found under “CERTUS-Web → resources → rmi-files → ClientTrustedCerts”. This keystore is protected by a password, which can be found in config.properties file (CERTUS-web → Java Resources → src → config.properties).
2. Access the RMI Registry on the server side using the server IP address and listening port (the server IP address and port can be found in the config.properties file).
3. After successfully locating the RMI registry, the RMI client gets a copy of the Server RMI interface which contains a list of all the remote method interfaces that can be called remotely by the client.

# **2. Design Decisions**

## 

## 2.1. HTML Framework

All of the web interfaces available on platform were using the Foundation 5 framework by Zurb Foundation (<http://foundation.zurb.com>). This product is an open source solution that comes with a useful and convenient functionality built using HTML, JQuery, and CSS and guarantees easy cross-browser compatibility. Particularly, we utilize the following plugins provided by the framework:

* Alerting users of certain events (alert module).
* Management of modal windows (reveal module).
* Client side user input validation (abide module).
* Management of dropdown input types (dropdown module).
* Displaying hints to users (tooltip module).
* Dynamic equalization of panels (equalizer module).

The entire source code of Foundation 5 is located in <platformBasePath>/WebContent/resources.

## 2.2. Structure of Projects

The primary IDE used by all team members during development of this project is *Eclipse Java EE IDE for Web Developers*. The recommended version is *Kepler*.

The main software components of this project are; the application server, web server (platform), and the hardware token simulator. The source code for these modules are separated into individual Java projects.

1. **CERTUS-Server**: contains the Java source code for the application server implementation.
2. **CERTUS-Token**: contains the Java source code for the hardware token simulator implementation.
3. **CERTUS-web**: contains the Java source code for the web server implementation. Furthermore, the shared java classes between the CERTUS-Server project and CERTUS-web project are included in the CERTUS-Common project.
4. **CERTUS-Common**: contains the Java source code for the shared classes between application server and web server.

## 2.3. Application Configurations

Application specific configuration details are stored in config.properties files. There are separate .properties files for the Server and Web modules. Each file is described below.

The main advantage of having .properties files to store application specific configuration details is to provide flexibility to change this parameter at the deployment stage.

### 2.3.1. Server Configuration Properties

Server configuration properties are included in this file. These properties are picked up by the server at run time.

File Location:

<serverBasePath>/CERTUS-Server/src/config.properties

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| db\_host | IP address of the database server |
| db\_password | password to access the database |
| db\_port | port number to access the database server |
| db\_schema | name of the database schema |
| db\_user | user name of the database |
| log\_fileName | name of the file that keeps logs of application |
| rmi\_basepath | the base path which all the RMI files are located in the file system of the server |
| rmi\_file\_policy | name of the policy file for RMI |
| rmi\_file\_keystore | file name of the key store for RMI |
| rmi\_file\_keystore\_password | password to access the RMI keystore |
| rmi\_port | port number that the RMI server is listening for connections |
| rmi\_registry | name of the RMI registry |
| email\_address | The email address used to send certus emails to the users. |
| email\_password | The password to access email\_address |
| email\_smtp\_server | The SMTP domain name used to send emails. |
| email\_smtp\_port | The SMTP port used to access SMTP service. |
| email\_password\_reset\_url | the url which should be include to the email message when user attempt a password reset |
| email\_system\_url | system url which should be included to the email messages, when directing users to access the system. |
| log\_filename | path and name of the RMI log file |

Table 1- Server configuration parameters

### 2.3.2. Web Configuration Properties

Web module specific configuration properties are included in this file. Relevant values are read from the .properties file during the runtime.

File Location:

<platformBasePath>/CERTUS-Web/src/config.properties

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| rmi\_basepath | the base path which all the RMI files are located in the file system of the web project |
| rmi\_file\_certificate | name of the client certificate file |
| rmi\_file\_certificate\_password | password to access the rmi certificate file |
| rmi\_host | IP address of the server which RMI service is running. |
| rmi\_port | port number that the RMI server is listening for connections |
| rmi\_registry | name of the RMI registry |

Table 2 - Web server configuration parameters

## 2.4. RMI initialization at runtime

In the production environment, the RMI server is expected to be constantly running and listening on specified port for incoming connections. Connection to the RMI server can be established in 2 possible ways:

1. On deployment and startup of the platform (Application on Tomcat). This behaviour is controlled by setting up the Listener static class in /projectRoot/rmi/Listener.java and having this class be invoked by default when Tomcat deploys the application, which is set in web.xml <platformBasePath>/WebContent/WEB-INF/web.xml. Once invoked, this class will establish a connection to the RMI service and load configuration properties.
2. In case the RMI connection to the server is lost and the user tries to invoke any of the RMI methods, the platform will throw an exception and immediately try to reconnect to the server. In case reconnection is been successful, the next call to RMI will be successful as well.

## 2.5. DB reconnection at runtime

Design of the system assumes that only the server will be establishing connection to the database. Database connection is established in 2 possible ways:

1. On startup, the server establishes a connection to the database. This connection persists throughout the entire uptime period.
2. Should the link to the database fail while the server is still running, any function which tries to invoke a query on database will throw an exception and fail. Along with that, it will also try to reestablish the connection to the database and in case this reconnection is successful, the next function which tries to connect to the database will succeed.

# **3. Core Functionality and Business Logic**

## 

## 3.1. User Management

### 

### 3.1.1. User types

Certus voting system internally identifies 4 possible user roles:

**1. Public users** - users who have not registered with the system or did not authenticate. This group of users has the lowest level of privileges in the system and can only view public pages and register.

**2. Invited users** - users who have been invited to sign up to the system and have not yet claimed their account. A typical invitation is an email with login and temporary password information which the user can use in order to sign in to the system for the first time and complete registration. Once the invitation email is sent by the server, a new user account, identified by this email address, will be created in the system. Upon authentication, the invited user will be forced to complete the registration process before accessing any functionality of the system.

**3. Registered users** - users who have successfully completed registration process, have login credentials and public key uploaded to the system and passed authentication.

**4. Administrators** - Super-users of the system with limited rights to edit records of other users, lock/unlock users, and archive elections.

### 3.1.2 New user registration

When a user tries to register as a new user, he/she is presented with two options: basic registration and advanced registration:

- Basic Registration: The user has to provide the following information: first name, last name, email address, user password. All those fields are required and are subject of validation. The user email address will be used as the username. The user password will be used to encrypt the private key before sending it to the user’s provided email. Later if the user changes the account password, this change will not affect the password which was used to protect the key.

- Advanced Registration: At first the user has to provide information identical to basic registration, however, with the click on advanced options users are presented with 3 possible options to generate public keys. These options will be further explored in User Key Management section.

### 3.1.3. Edit Profile

Once the user logs in, he/she has an option to change basic profile information: first name and last name. The user email cannot be changed since it is used as the login username. Users can also change the login password, however, this change will not affect password, which were used to protect their corresponding private key.

### 3.1.4. User Key Management

It is important that we store users’ keys in a secure way. When a user first registers with the system, they can set up their signing keys in one of three ways:

* + - 1. The user can let the server generate a key pair using their login password. With this option, the user’s public key is stored in the database, and the user’s private key is encrypted using their login password and sent to the user via email. The private key is not stored on the server in any form.
      2. The user can let the server generate their key pair using a password other than the login password. This option proceeds as the previous option, except that the private is encrypted under a different password specified by the user.
      3. The user can upload its own public key. This option is for more advanced users who either have a key pair already or know how to generate their own key pair. It is expected that their private key will be protected by some password, which they will need to provide to the token in order to generate signatures. Uploaded public key is subject of validation and its size cannot exceed 10kb.

We also need to store election keys. These are the keys used to encrypt and decrypt votes for a particular election. When a user creates an election, they must provide a password which will be used to protect the election’s private key. Once the private key has been encrypted, the public and private keys are stored in the database. When the user wants to close and publish this election, they must provide the password in order to decrypt the private key so that the server can decrypt votes and tally them.

### 3.1.5 Forgot password

When a user has forgotten their password, we have included functionality to allow the user to reset it. The user goes to the CERTUS login screen and clicks on the “Forgot your password?” link. This will bring them to an interface in which they are asked to provide their email address. The user will do so and press the “Reset Password” button. This will bring the user to a third screen, which notifies them that an email has been sent to the provided address with instructions on how to proceed. Behind the scenes, however, one of three things has occurred.

First, if the provided email matches the email address of a registered CERTUS user, an email is sent to their address with a temporary password and a link to the next step in the password reset process. Upon following this link, the user will be prompted to provide their email address, the temporary password, and a new password. If the email and temporary password are correct, the user’s password will be updated and the user will be logged in.

The second possibility is that the provided email address matches the email address of a user who has been invited to register with CERTUS, but has not completed registration yet. In this case, we re-send the invitation to this user which includes instructions on how to complete registration.

The third possibility is that the provided email address does not match any email addresses in the database. In this case, no emails are sent. Note, however, that a user who enters an invalid email address will see the same thing as a user who enters a valid email address (aside from the actual email). This presents malicious users from determining which email addresses are and are not associated with users of the system.

### 3.1.6. Blocking Users

The system administrator can change the user status to *Locked*, by accessing the User Management page. Once the user is in the *Locked* state, they cannot login to the system.

To unlock a user, the system administrator can change the status to *Activated*.

## 3.2. Election management

### 3.2.1. Adding new election

Any user of the system can create an election. The owner of an election is commonly referred to as the *Election Authority* across all documentation. One Election Authority can own multiple elections and each election can have exactly one Election Authority. In order to create an election, the user has to enter the following fields about the new election:

1. Election name

2. Election description

3. Tentative election end date - in free format and optional. This field will be shown to users for reference only and is never enforced.

4. Election password - password and password confirmation fields specified by Election Authority. This password will be used to protect the private key, which is automatically generated by the server for this election. Election Authority will be asked to enter this password before invoking *publish election* action in order to decrypt the private key of this election and use it in *tallying* process.

5. List of candidates - the list of candidates is entered as a set of names separated by newline. This list will be saved in the database in the same format, until the election is opened. Candidates will appear in same order as specified in this field across all user interfaces.

6. Election availability - users specify whether this election is private or public. More on public and private elections is in sections below.

Every field is subjected to input validation and no information will be processed unless all input fields pass validation. Input validation occurs both on the client and server sides.

### 3.2.2 Edit elections

While election is in the *open* state, the Election Authority can come back and edit the following parameters in the election:

1. Election name

2. Election description

3. Tentative election end date

4. List of candidates

5. Election availability

The election password can no longer be changed as the private key for this election has already been generated, encrypted by the password, and saved to the database.

Every field is subjected to input validation and no information will be processed unless all input fields pass validation. Similar to adding an election, input validation occurs both on the client and server side.

### 3.2.3. Public Elections

Public elections in Certus system grant all valid users registered in the system the right to vote and access election results. Once an election has been defined as public and opened, public election cannot be switched to private.

### 3.2.4. Private Elections

Private elections limit election availability only to set of users which were explicitly granted voting rights by an Election Authority. In order to invite users to vote, the Election Authority must enter a list of emails, separated by newlines, of all the users she wants to be able to vote. Once submitted, the system will check whether every single email entry can be parsed and associated with a registered user of Certus. Users who pass validation, will be successfully marked as the ones who will get the right to vote. Users who do not pass validation will be returned to the Election Authority in a form of a list of checkboxes. Election Authority has a chance to double check the correctness of entered emails and if necessary, mark emails to invite people to register in the system.

Invited users will receive automated email from Certus with login and temporary password information, as well as a link to access the system. After that, invited users will be able to authenticate to Certus and complete their registration.

When the Election Authority decides to open an election, the list of voters will be entered into the *participate* table, which will grant them the right to vote. In addition to that, Election Authority can further invite more people to vote. This process is similar to adding users to a new election except that voters are added to participate table immediately.

### 3.2.5. States of Elections

Election lifecycle with corresponding states in each stage is shown in the Figure 3: Election states.

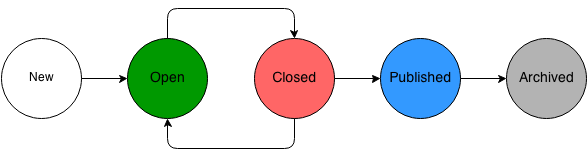


Figure 3 - Election States

Any election in Certus can have 5 possible states:

1. **New** - election just has been added to the system. All election parameters can be edited, Election Authority can switch election availability from public to private and vice versa, add/delete voters and invite new users. New elections are not visible to voters and can only be seen from elections management interface by the Election Authority.

Available actions: open election.

2. **Open** - an election that has already been started or restarted by an Election Authority. This election is visible to all eligible voters and they can vote. In private elections, Election Authority can add or invite additional users who are eligible to vote in this election.

Available actions: close election.

3. **Closed** - an election which has been closed by an Election Authority. Voters are no longer able to vote.

Available actions: reopen election, publish results

4. **Published** - election results have been computed by the tallying procedure and published. The results can be accessed by Election Authority and all eligible users.

Available actions: none.

5. **Archived** - election permanently hidden from all users.

Available actions: none.

### 3.2.6. Election Actions

|  |  |  |  |
| --- | --- | --- | --- |
| **Action name** | **Who can invoke** | **Eligible election states** | **What happens** |
| Add election | Election Authority | - | Election entry appears in database. A key pair is generated for this election. Private key is protected by a password, specified by Election Authority. All election fields are also saved to database. List of candidates saved as string. List of users who can vote is also saved as string. |
| Open | Election Authority | New | Change of state from New to Open. String of candidates is parsed into real candidates which are added to Candidates table in database. String of users eligible to vote is parsed into corresponding relationships in table participate. |
| Close | Election Authority | Open | Change of state from Open to Closed |
| Reopen | Election Authority | Closed | Change of state from Closed to Open |
| Publish | Election Authority | Closed | Change of state from Closed to Published. Tallying procedures verify all votes casted for this election, and upon successful verification, will be counted towards the election results. Election results are saved to database and are available to be reviewed by users. |
| Archive | Administrator | Any | Change of state to Archived. Election is no longer visible to any users. |

Table 3- Election Actions

One of the design features specifically thought for this project was to allow maximum flexibility for election authorities during creation of elections in terms of specifying candidates list, list of eligible voters, and yet provide simple and user friendly interfaces. As a result, we do not parse candidates list and list of voters before the election is actually started.

## 3.3. Voting and double envelope protocol

CERTUS uses the double envelope protocol to cast votes from users. The voting procedure can be represented as the following sequence of actions:

1. User picks an election he/she wants to vote in.

2. User is presented with a list of candidates, in the form of radio buttons. Once the user chooses a candidate, he/she will be presented with a candidate ID, encrypted by the public key of this particular election. The user’s choice of candidate (in plaintext) is never saved in the system.

3. The user copies the ciphertext and enters it to the token. In addition to that, the user also specifies the location of his/her private key in order to sign the encrypted choice of candidate. Once the token has the private key and ciphertext, it will generate a corresponding signature which confirms authenticity of user’s choice.

4. User copies the signature, generated by the token into the platform.

5. User submits the ciphertext of the encrypted vote and signature, which is considered as completed vote.

6. If the server can verify the user’s signature, the vote is saved to the database and awaits tallying.

## 3.4. Tally and publish election

We have implemented a simple tallying protocol, which can later be replaced with a more secure protocol (likely using mixnets or homomorphic encryption). First, the server use the password provided by the user to decrypt the private decryption key for the election. The server then gets the set of signed, encrypted votes for this election from the database. For each vote, the server checks its signature, and if it verifies the server decrypts the vote and attributes it to the corresponding candidate (here the server also checks that the decrypted value is a valid candidate). The results are then stored in the database to be published.

# **4. Reference Monitor**

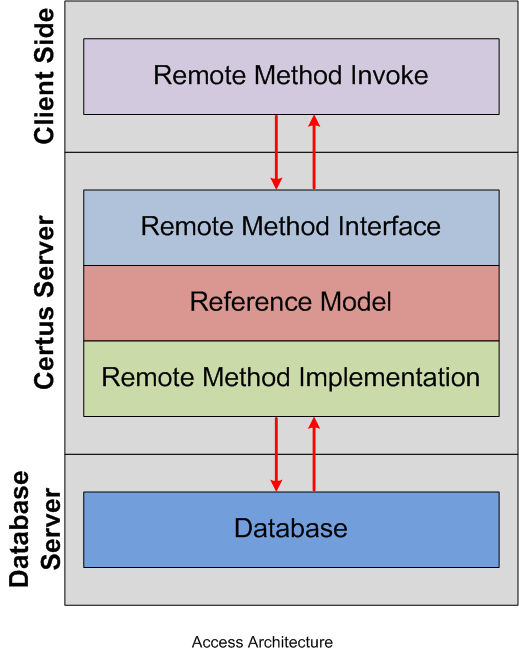


Figure 4 - Access Architecture

## 4.1. User Roles:

- System Administrators (Admin).

- Registered Users - can also be referred as regular users (can be either a voter or an election authority).

- Invited Users - can also be referred as Temp Users (Those who are not registered at the system, but they were invited to a private election).

- Public.

## 4.2. Reference Monitor Architect:

So far Certus Server provides any client with 29 remote methods can be invoked over RMI to handle all the needed functionality of the system. After analyzing those remote methods, we were able to divide it into five different right groups:

### 4.2.1. Rights Group 0:

Method: gotRightsGroup0 (*requesterID*, *action*):

Will return true only:

- if *requesterID* has the right to do *action* (basic role base access control).

### 4.2.2. Rights Group 1: (operations on the requester’s data)

Method: gotRightsGroup1(*requesterID*, *targetedUserID*, *action*):

to get the permission, the following rules should match:

- *requesterID* should have RightsGroup0 on the *action*.

- if *requesterID* is an admin: it has the right to do action regardless of the *targetedUserID*

- if *requesterID* is not admin: it has the right to do action if the *targetedUserID* is the same as the *requesterID* (Apply action on it’s own record).

### 4.2.3. Rights Group 2: (operations for election authorities or admins)

Method: gotRightsGroup2(*requesterID*, *electionID*, *action*):

to get the permission, the following rules should match:

- *requesterID* should have RightsGroup0 on the *action*.

- if *requesterID* is an admin: it can invoke *action*.

- if *requesterID* is not admin: it should be *electionID* authority.

### 4.2.4. Vote Right:

Method: isAllowedToVote(*requesterID*, *voterID*, *electionID, action*):

Will return true only if:

- *requesterID* should have RightsGroup0 on *action*.

- *requesterID* should be the same as *voterID*.

- *requesterID* should be invited to *electionID*.

### 4.2.5. View Results Right:

Method: isAllowedToViewResults(*requesterID, electionID, action*):

Will return true only if:

- *requesterID* should have RightsGroup0 on *action*.

- if *requesterID* is an admin: it has the right.

- if *requesterID* is not admin: it should be allowed to access *electionID* (either public election,

**Notes:**

- If the election is public, then any user role can access the election.

- To be allowed to access a private election, the requester should be either the election authority, or invited to that election.

## 4.3. Remote Methods Permissions:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Method**  **ID** | **RMI Method** | **Permitted User Role** | **Rights**  **Group** |
| 1 | 10 | addUser | Public | N/A |
| 2 | 47 | addUserWithPP | Public | N/A |
| 3 | 48 | addUserWithKey | Public | N/A |
| 4 | 9 | checkIfUsernamePasswordMatch | Public | N/A |
| 5 | 50 | checkIfUsernameTempPasswordMatch | Public | N/A |
| 6 | 56 | resetPassword | Public | N/A |
| 7 | 45 | deleteElection | Admin | Group 0 |
| 8 | 14 | editUserStatus | Admin | Group 0 |
| 9 | 12 | selectAllUsers | Admin | Group 0 |
| 10 | 44 | selectElectionsForAdmin | Admin | Group 0 |
| 11 | 19 | addElection | User | Group 0 |
| 12 | 32 | generateNewKeys | User | Group 1 |
| 13 | 36 | uploadPubKey | User | Group 1 |
| 14 | 42 | selectElectionsForOwner | User | Group 1 |
| 15 | 43 | selectElectionsForResults | User | Group 1 |
| 16 | 46 | selectElectionsForVoter | User | Group 1 |
| 17 | 20 | editElection | User | Group 2 |
| 18 | 21 | editElectionStatus | User | Group 2 |
| 19 | 22 | openElectionAndPopulateCandidates | User | Group 2 |
| 20 | 24 | publishResults | User | Group 2 |
| 21 | 37 | selectElectionFullDetail | User | Group 2 |
| 22 | 38 | addAdditionalUsersToElection | User | Group 2 |
| 23 | 40 | selectElectionForOwner | User | Group 2 |
| 24 | 30 | voteProgressStatusForElection | User | View Results Right |
| 25 | 31 | selectResults | User | View Results Right |
| 26 | 28 | vote | User | Vote Right |
| 27 | 29 | getTallierPublicKey | User | Vote Right |
| 28 | 41 | selectElectionForVoter | User | Vote Right |
| 29 | 51 | updateTempUser | TempUser | Group0 |
| 30 | 52 | UpdateTempUserWithPP | TempUser | Group0 |
| 31 | 53 | UpdateTempUserWithKey | TempUser | Group0 |
| 32 | 33 | logOut | Admin, User | Group 0 |
| 33 | 13 | editUser | Admin, User | Group 1 |
| 34 | 11 | selectUser | Admin, User | Group 1 |
| 35 | 34 | updateUser | Admin, User | Group 1 |
| 36 | 35 | updateUserPassword | Admin, User | Group 1 |

Table 4 - Remote Methods Permissions

# **5. Database Model**

This section describes the database model of the CERTUS application.

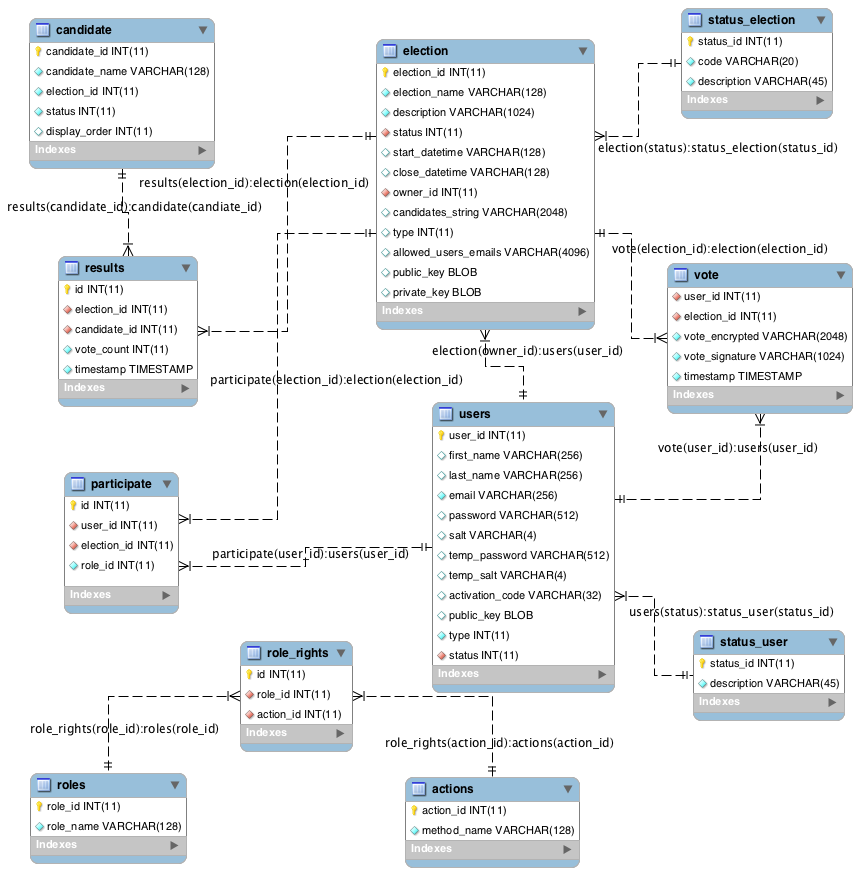
The Entity Relationship Diagram of this system is given in the Figure 5. Relationships in the diagram is given in the notation <tableName>(<foreignKeyName>): <parentTableName>(<primaryKey>)

Figure 5 - Certus Database ER diagram