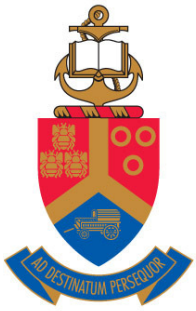

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS 301

Main Project

Functional Requirements Specification

Team Codex

Andreas du Preez	12207871
Azhar Mohungoo	12239799
Gift Sefako	12231097

Contents

List of Figures

1 Introduction

This document aims to specify the functional for an electronic voting system specified by CodeX and the client Mr Roelof Naude.

2 Vision

What is intended for this project is to create a web and mobile platform, which can be used to cast votes in the provincial and national elections of South Africa. The following are benefits of the system:

1. The use of the Blockchain will allow for a transparent election process while still maintaining anonymity of the voters.
2. Votes that get lost or disregarded be it on purpose or not.
3. Incorrect counting.
4. Manual vote counting will no longer be required
5. Remote casting of votes from a users preferred web browser or mobile smartphone.
6. Secure voting
7. Prevent invalid votes from being cast through robust validation with each request.

3 Background

Elections are always a time where emotions are at a high and passion for a leader has never been greater. Sometimes this emotion and passion for a leader can lead to unlawful activities to get their chosen leader to to win the elections. By moving the system to an electronic environment it removes almost, if not all, these possibilities for unlawful activities to take place by using computers instead of humans. Although with the use of the Blockchain implementation, these activities can be suppressed.

Additionally to a secure and safe voting environment that allows ease of access from the voter's mobile or web browser, it will also make the voting process more efficient, with shorter queues to cast a vote and the progress of elections and the final result will be presented more rapidly, with periodic updates as the system analyses these votes. This system can be used in multiple scenarios not only in election.

A more generic version of the electronic voting system can be implemented to allow users to partake in surveys, where instead of an election poll, users can create their own custom polls in which they can control participation. A further instance would be using such a system for national wide statistics, gathered by completing a poll of some kind, where anonymity is vital.

4 Functional Requirements

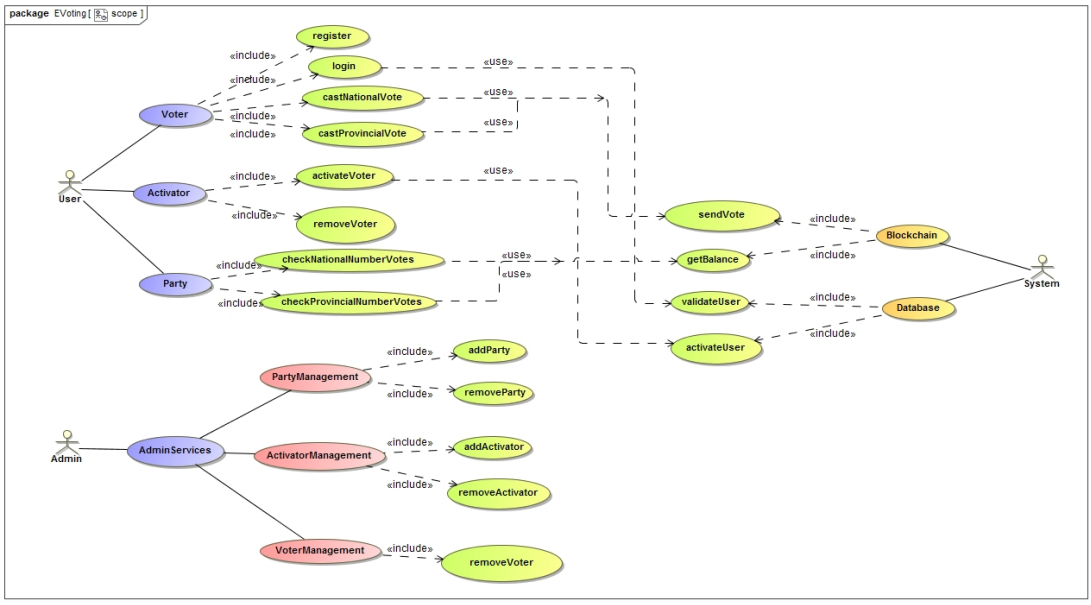

Figure 1: Scope Diagram

4.1 Blockchain

1. **Send Vote**

   (a) **Service Contract**

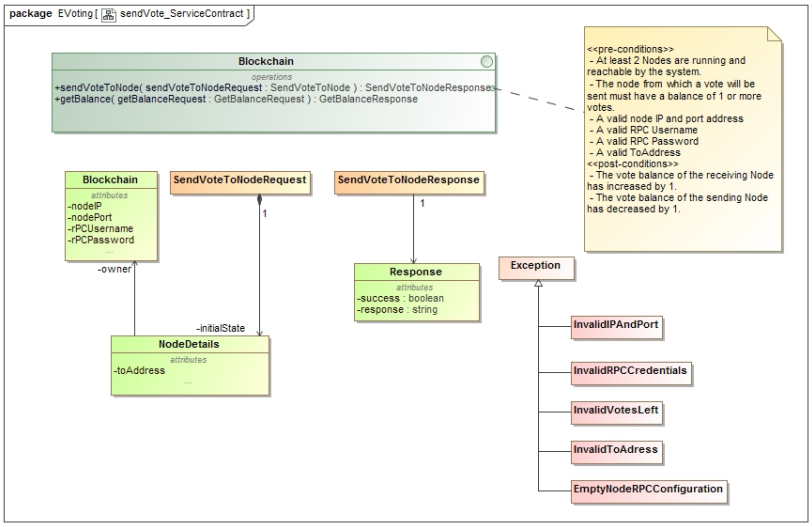

Figure 2: Send Vote Service Contract

SendVote requires a ToAddress string which is the receiving node's address. The BlockchainModule is required to be instantiated with a NodeIP string, NodePort string, RPC username string and a RPC password string from where the vote will be sent.

i. Pre-conditions
   - At least 2 Nodes are running and reachable by the system.
   - The node from which a vote will be sent must have a balance of at least 1.
   - A valid node IP and port address.
   - A valid RPC Username.
   - A valid RPC Password.
   - A valid toAddress.

ii. Exceptions
   - If the configuration strings(node IP address, port address, RPC Username, RPC Password and toAddress) are empty, the EmptyNodeRPCConfiguration exception will be thrown.
   - If the system cannot reach the node specified by the IP and port address, the InvalidIPAndPort exception will be thrown.
   - If the specified RPC credentials are incorrect, the InvalidRPCCredentials exception will be thrown.

- If toAddress does not exist in the blockchain, the InvalidToAddress exception will be thrown.
- If the sending node does not have a balance of at least 1, the Invalid-VotesLeft exception will be thrown.

iii. Post-conditions

- The vote balance of the receiving node has increased by 1.
- The vote balance of the sending node has decreased by 1.

(b) **Functional Requirements**

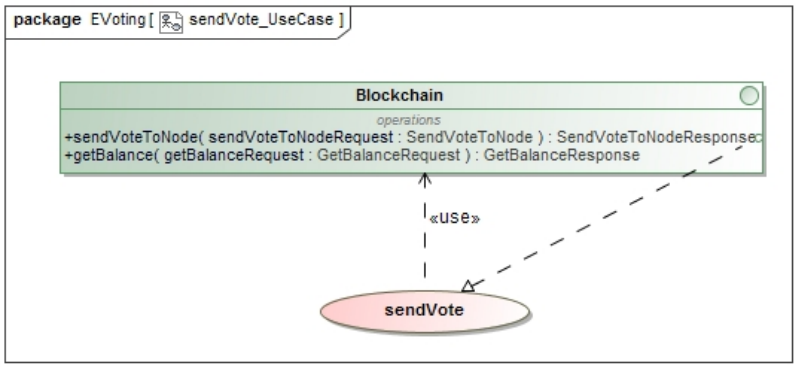

Figure 3: Send Vote Use Case

The sendVote process does not rely on any other use case, instead the send-Vote process will be used by various other use cases.

(c) **Process Design**

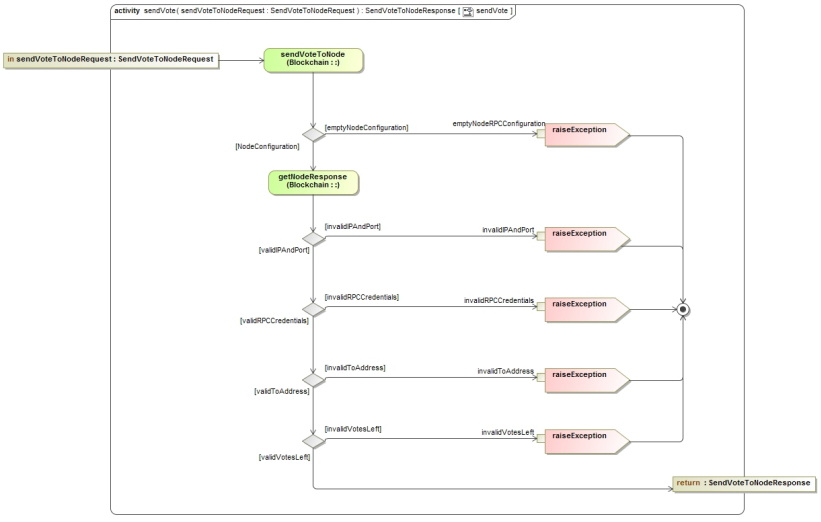

Figure 4: Send Vote Activity

Once the Send Vote process is called, it first checks if it has all the necesary configuration strings in order to send a vote. If any of these configuration strings are empty, an exception will be thrown. Sencondly the system will check if the node is reachable by sending it a 'ping' request and await a response. If the node does not respond, an exception will be thrown. The system will then try to make a transaction in the blockchain from the sending node to the receiving node. If the RPC credentials are incorrect, the InvalidRPCCredentials exception will be thrown. If the ToAddress does not exist in the blockchain, the InvalidToAddress exception will be thrown. If the sending node does not have atleast a balance of 1, the InvalidVotesLeft exception will be thrown. If no exceptions are thrown, and the node returns a string containing the transaction hash, it means the vote has been successfully sent.

2. **Get Balance**

   (a) **Service Contract**

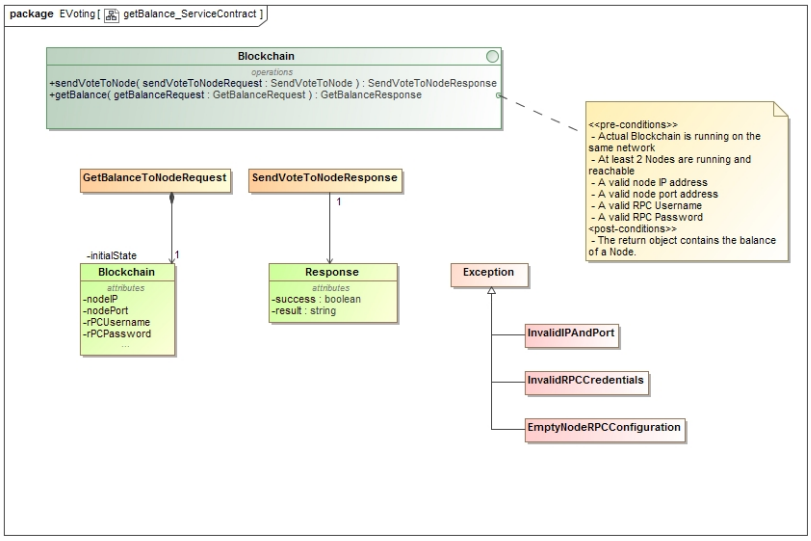

Figure 5: Get Balance Service Contract

In order for a political party to check how many votes they have, the system can return the balance of a specified node. The getBalance process returns the balance of the node specified by the configuration strings (nodeIP and nodePort).

i. Pre-conditions
   - At least 1 Node are running and reachable by the system.
   - A valid node IP and port address.
   - A valid RPC Username.
   - A valid RPC Password.

ii. Exceptions
    - If the configuration strings(node IP address, port address, RPC Username, RPC Password) are empty, the EmptyNodeRPCConfiguration exception will be thrown.
    - If the system cannot reach the node specified by the IP and port address, the InvalidIPAndPort exception will be thrown.
    - If the specified RPC credentials are incorrect, the InvalidRPCCredentials exception will be thrown.

iii. Post-conditions
    - The return object contains the balance of a Node in the 'reponse' field.

(b) **Functional Requirements**

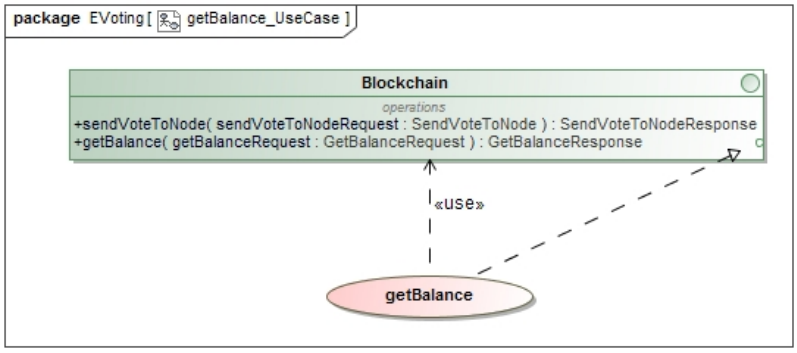

Figure 6: Get Balance Use Case

The getBalance process does not rely on any other use case, instead the Get Balance process will be used by various other use cases.

(c) **Process Design**

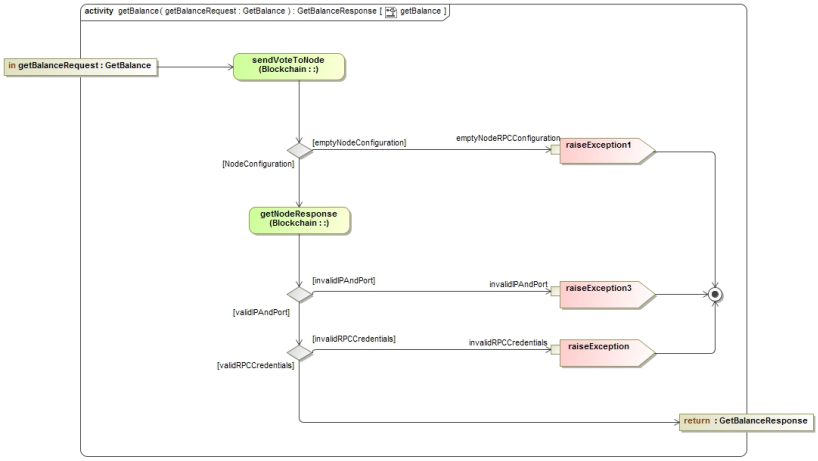

Figure 7: Get Balance Activity

When the getBalance process is called, it first checks if it has all the necesary configuration strings in order to connect to the node. If any of these configuration strings are empty, an exception will be thrown. Sencondly the system will check if the node is reachable by sending it a 'ping' request and await a response. If the node does not respond, an exception will be thrown. The system will then try to get the balance of the node. If the RPC credentials are incorrect, the InvalidRPCCredentials exception will be thrown. If no exceptions are thrown, the return object will contain the balance of that node.

4.2 Database

1. **Validate User**

   (a) **Service Contract**

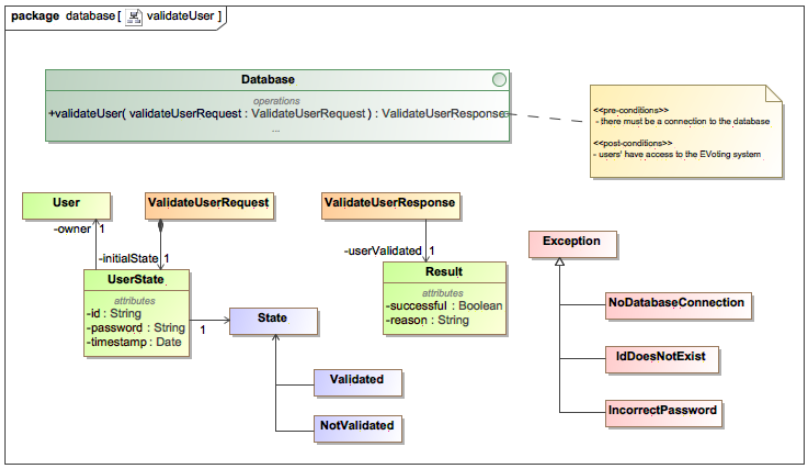

Figure 8: Validate User Service Contract

i. Pre-conditions
   - There must be a connection to the database

ii. Exceptions
   - If there is no connection to the database, the NoDatabaseConnection exception will be thrown
   - If a user's ID is not valid, the IdDoesNotExist exception will be thrown
   - If the user's password is entered incorrectly, the IncorrectPassword exception will be thrown

iii. Post-conditions
   - Users have access to the EVoting system

(b) **Functional Requirements**

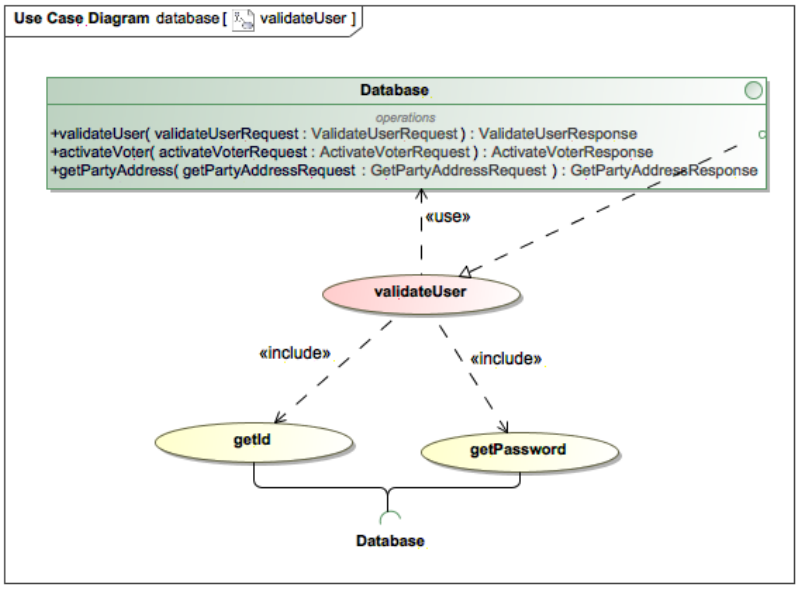

Figure 9: Validate User Use Case

(c) **Process Design**

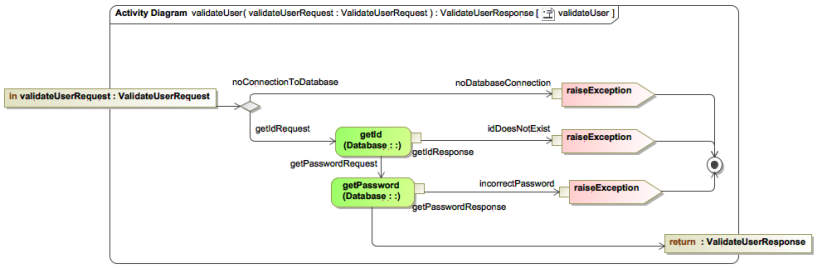

Figure 10: Validate User Activity

2. **Activate User**

(a) **Service Contract**

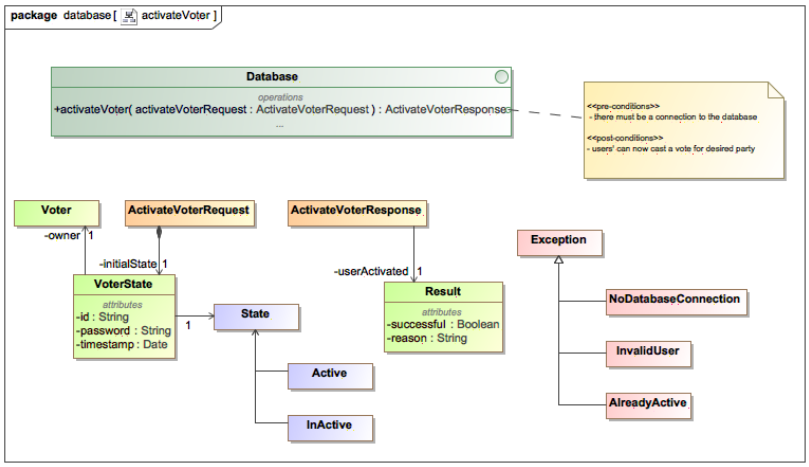

Figure 11: Activate Voter Service Contract

i. Pre-conditions
   - There must be a connection to the database

ii. Exceptions
   - If there is no connection to the database, the NoDatabaseConnection exception will be thrown
   - If a user could not be validated, the InvalidUser exception will be thrown
   - If the user has already been activated, the AlreadyActive exception will be thrown

iii. Post-conditions
   - Users can now cast a vote for their desired party

(b) **Functional Requirements**

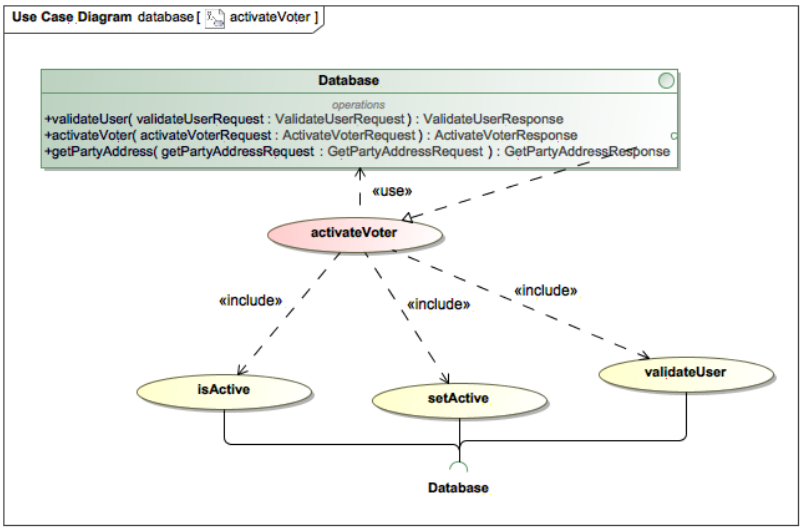

Figure 12: Activate Voter Use Case

(c) **Process Design**

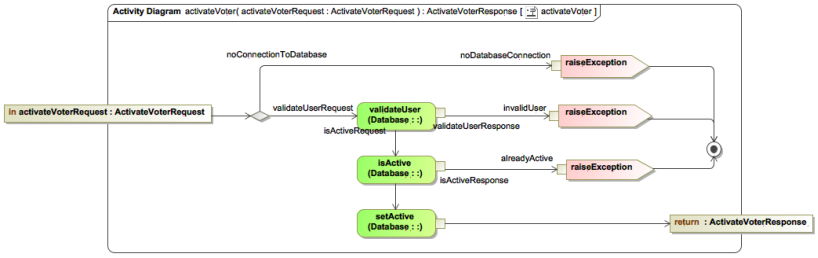

Figure 13: Activate Voter Activity

4.3 Admin

4.4 Voter

1. **Register**

   Registering a Voter collects all the data required for a user to be valid and adds that information to the Voter database. Once the Voter has been added to the database they will have access to a minimum system. The minimum system only allows them to log in and view where they can go to get activated (to allow access to the entire system) to allow them to participate in the current election.

   (a) **Service Contract**

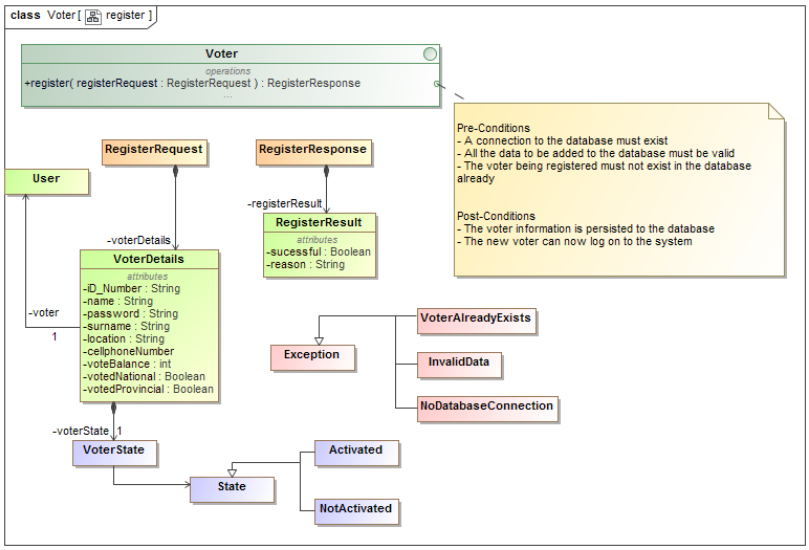

Figure 14: Register Voter Service Contract

Register creates an initial state for a Voter and captures all the provided information in the database. Once information has been added to the database, the user can edit it themselves by accessing their account once they have logged in. Only valid data will be accepted.

i. Pre-conditions
   - There must be a connection to the database
   - The information provided by the user must be valid.
   - The user must not exist in the current database.

ii. Exceptions
   - If there is no connection to the database, the NoDatabaseConnection exception will be thrown.
   - If the data is invalid then the service is refused and the InvalidData exception is thrown.
   - If the user already exists in the database then the service is refused and the UserAlreadyExists exception is thrown.

iii. Post-conditions

- The new Voter can log in and access the Electronic Voting system.
- The Voters information is persisted to the database.

(b) **Functional Requirements**

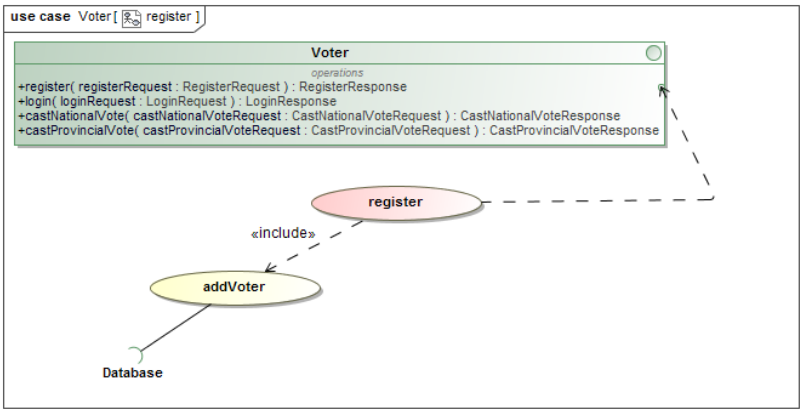

Figure 15: Register Voter Use-Case

i. The registerRequest object encapsulates all the necessary information required to add a new Voter to the system.

ii. All the information collected by the register function is persisted to the database through the Database module's addVoter function.

(c) **Process Design**

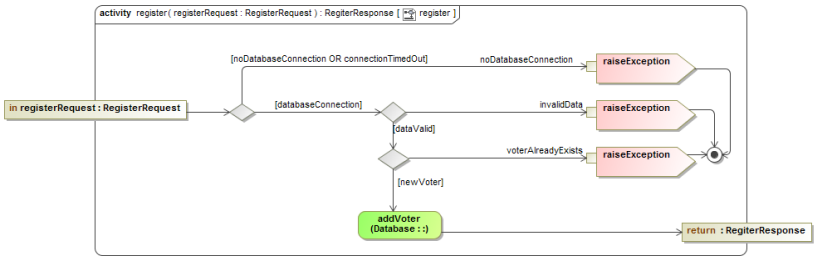

Figure 16: Register Voter Activity

i. The Register function first checks to see if there is a connection to the database, if there is not or the connection times out, the NoDatabaseConnection exception is thrown.

ii. If there is a database connection, it checks whether the Voter's data is valid(according to the data types specified in the database).

iii. If the data is not valid, the InvalidData exception is thrown.

iv. Then the function checks whether the Voter's information already exists in the database and throws the VoterAlreadyExists exception if the Voter already exists.

v. Otherwise no exception is thrown and the Voter's information is persisted to the database using the Database module's addVoter function.

2. **Login**

Login functionality gives the user access to system and allows them to view their account information, and only once they have been activated, are they allowed to cast votes and view other relevant information.

(a) **Service Contract**

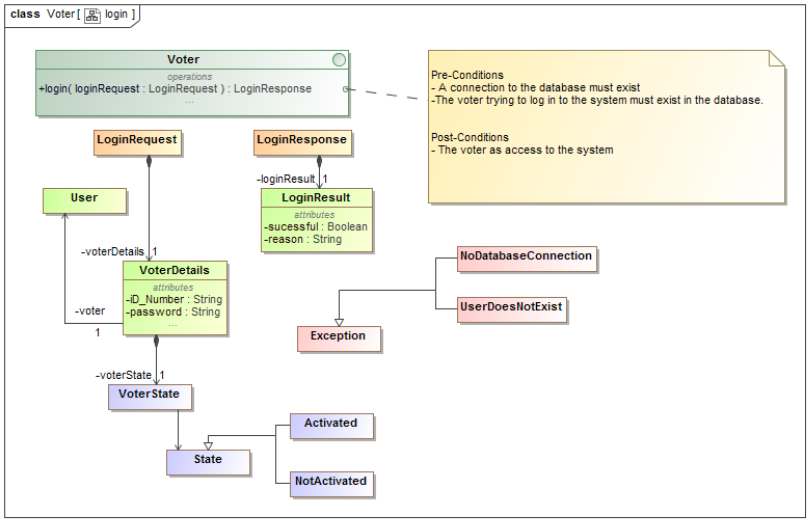

Figure 17: Login Service Contract

Login validates a user and gives them rights to interact with the core system.

i. Pre-conditions
   - There must be a connection to the database
   - The user must have already registered.

ii. Exceptions
   - If there is no connection to the database, the NoDatabaseConnection exception will be thrown
   - If the user has not registered then the UserDoesNotExist exception is thrown and the service is denied.

iii. Post-conditions
   - The Voter has access to the Electronic Voting system.

(b) **Functional Requirements**

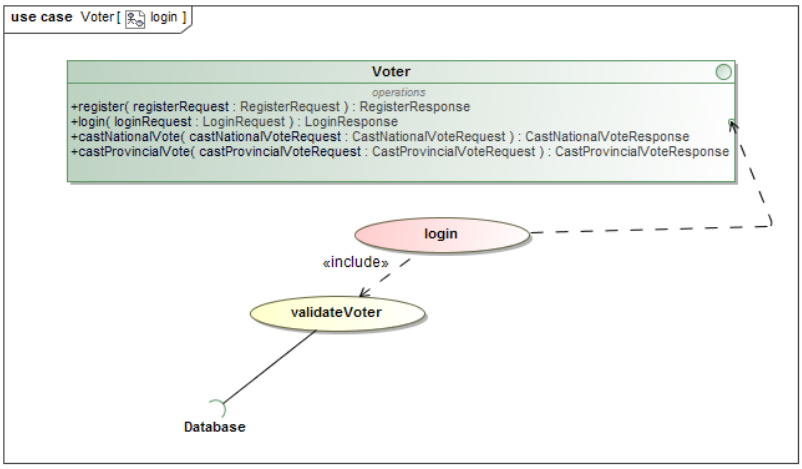

Figure 18: Login Voter Use-Case

i. The login use-case collects Voter information which will be used to validate a Voter.

ii. A Voter is invalid if their credentials do not checkout.

iii. The Voter module's login passes information to the Database modules validateVoter function to check whether the Voter attempting to log in is valid or not.

(c) **Process Design**

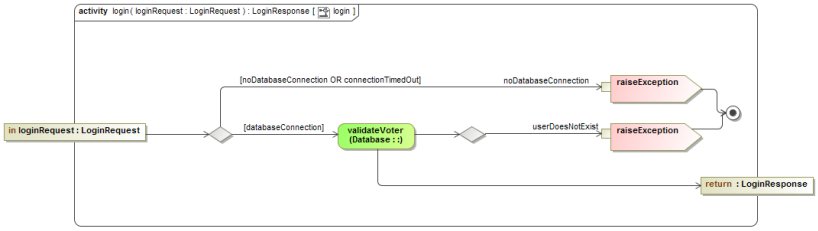

Figure 19: Login Voter Activity

i. The Login function first checks to see if there is a connection to the database, if there is not or the connection times out, the NoDatabaseConnection exception is thrown.

ii. Then it calls the Database module's validateUser function which will return an object that contains a boolean value of whether the voter is valid (and can thus access the system) as well as a reason or false (thus the Voter is denied access) as well as reason as to why the service has been denied.

iii. The UserDoesNotExist exception is thrown if validateUser returns false.

3. **Cast National Vote**

Casting a vote requires voter to be logged in.

(a) **Service Contract**

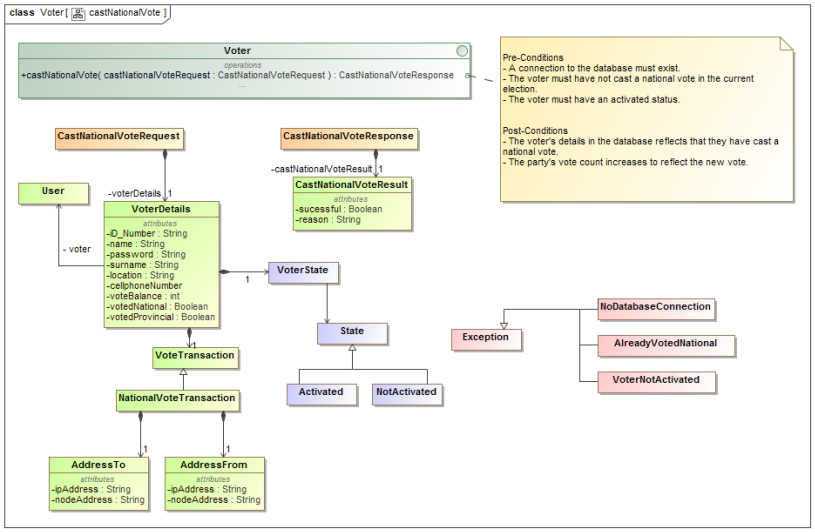

Figure 20: Cast National Vote Service Contract

Once the Voter is logged in we can retrieve their account information to view their location and their current vote balance(calculated by whether they have voted national, provincial or neither).
The voting nodes address for the current voter(based on the users location) as well as the party nodes address are retrieved, this information will be sent to the Blockchain Module to concretely cast the voters vote as Blockchain transaction.

i. Pre-conditions
   - There must be a connection to the database
   - The Voter must not have already cast a National vote in the current election.
   - The Voter must have an Activated status.

ii. Exceptions
   - If there is no connection to the database, the NoDatabaseConnection exception will be thrown.
   - If the Voter has already cast a National vote, the AlreadyVotedNational exception is thrown and the service is denied.
   - If the Voter has not been activated by an Activator, the VoterNotActivated exception is thrown and they are disallowed from casting a vote.

iii. Post-conditions

- Voter details in the database reflect that they have cast a National Vote.
- The Party which the Voter has voted for shows an increment by one in their node balance.

(b) **Functional Requirements**

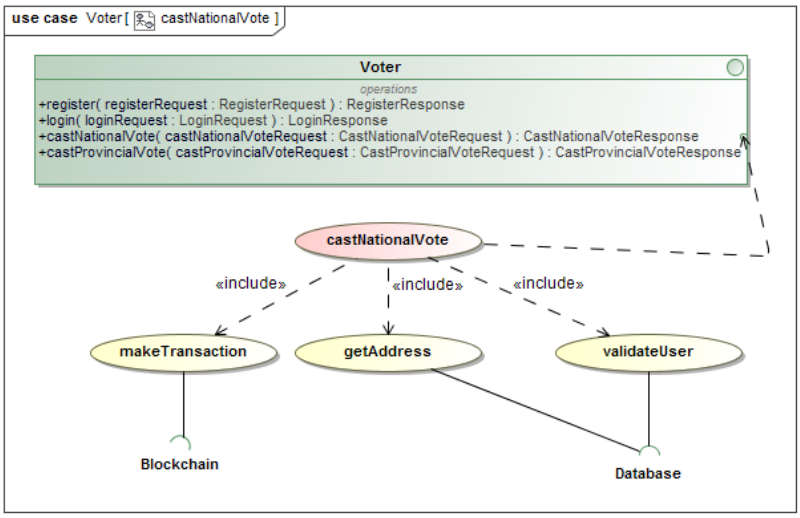

Figure 21: Cast National Vote Use Case

i. The Cast National Vote use case starts uses the database to retrieve the addresses of the Party which is being voted for as well as the Voting region node's addresses.

ii. Once it has all the addresses, it uses the Blockchain's makeTransaction functionality to cast the vote into the Blockchain. (Sending one coin from the Voting Region Node to the coin accepting Party Node of the Voter's choice.)

(c) **Process Design**

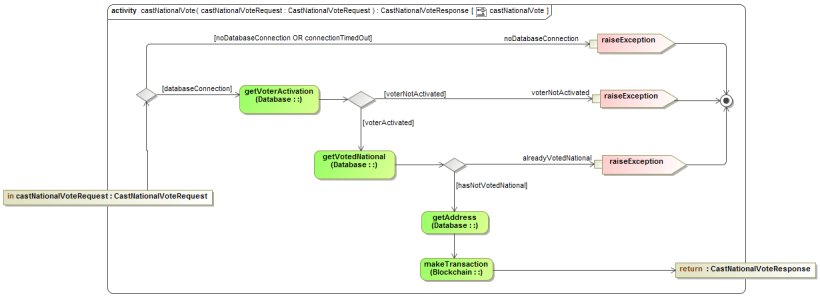

Figure 22: Cast National Vote Activity

i. The function first checks to see if there is a connection to the database,

if there is not or the connection times out, the NoDatabaseConnection exception is thrown.

ii. If a connection exists, it proceeds to check whether the Voter has been activated or not.

iii. If the Voter has not been activated, the VoterNotActivated exception is thrown.

iv. Then the function checks whether the Voter has already cast a National vote. If they have the AlreadyVotedNational exception is thrown.

v. If none of the exceptions are thrown, the function then queries the database to find all of the necessary addresses then it calls the Blockchain's makeTransaction function to cast the actual vote.

4. **Cast Provincial Vote**

Casting a vote requires voter to be logged in.

(a) **Service Contract**

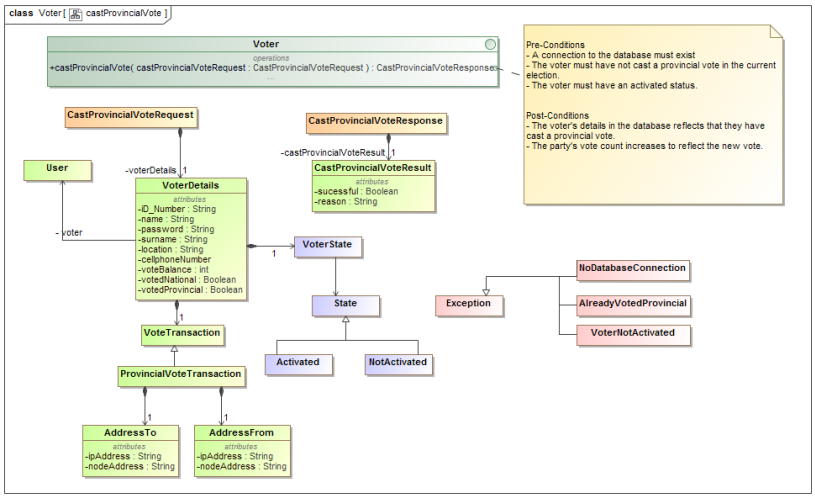

Figure 23: Cast Provincial Vote Service Contract

Once the Voter is logged in we can retrieve their account information to view their location and their current vote balance(calculated by whether they have voted national, provincial or neither).
The voting nodes address for the current voter(based on the users location) as well as the party nodes address are retrieved, this information will be sent to the Blockchain Module to concretely cast the voters vote as Blockchain transaction.

i. Pre-conditions

- There must be a connection to the database
- The Voter must not have already cast a Provincial vote in the current election.

- The Voter must have an Activated status.

ii. Exceptions
- If there is no connection to the database, the NoDatabaseConnection exception will be thrown.
- If the Voter has already cast a National vote, the AlreadyVoted-Provincial exception is thrown and the service is denied.
- If the Voter has not been activated by an Activator, the VoterNo-tActivated exception is thrown and they are disallowed from casting a vote.

iii. Post-conditions
- Voter details in the database reflect that they have cast a Provincial Vote.
- The Party which the Voter has voted for shows an increment by one in their node balance.

(b) **Functional Requirements**

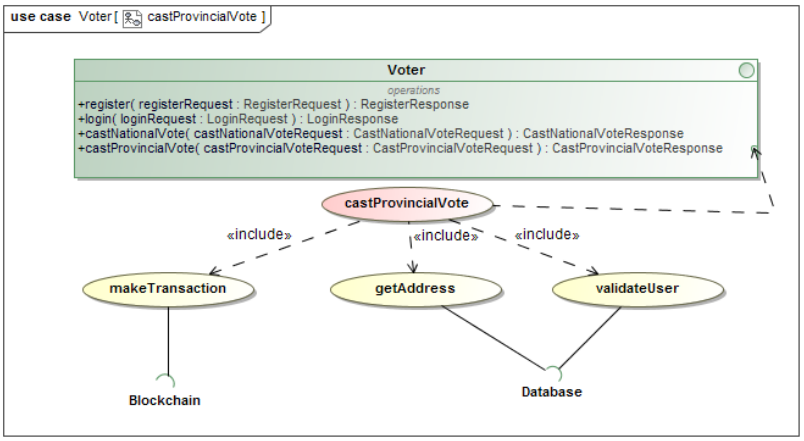

Figure 24: Cast Provincial Vote Use Case

i. The Cast Provincial Vote use case starts uses the database to retrieve the addresses of the Party which is being voted for as well as the Voting region node's addresses.

ii. Once it has all the addresses, it uses the Blockchain's makeTransaction functionality to cast the vote into the Blockchain. (Sending one coin from the Voting Region Node to the coin accepting Party Node of the Voter's choice.)

(c) **Process Design**

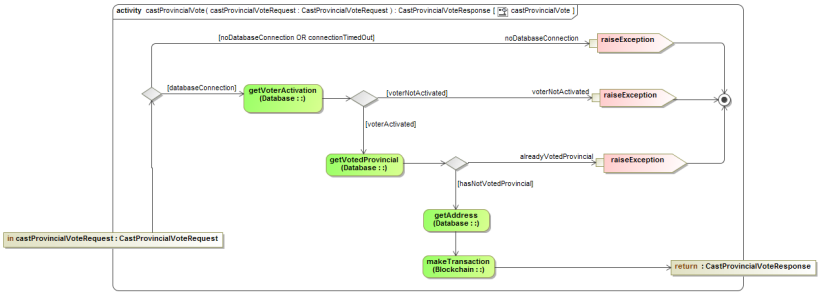

Figure 25: Cast Provincial Vote Activity

i. The function first checks to see if there is a connection to the database, if there is not or the connection times out, the NoDatabaseConnection exception is thrown.

ii. If a connection exists, it proceeds to check whether the Voter has been activated or not.

iii. If the Voter has not been activated, the VoterNotActivated exception is thrown.

iv. Then the function checks whether the Voter has already cast a Provincial vote. If they have the AlreadyVotedProvincial exception is thrown.

v. If none of the exceptions are thrown, the function then queries the database to find all of the necessary addresses then it calls the Blockchain's makeTransaction function to cast the actual vote.

4.5 Activator

1. **Activate Voter**

   (a) **Service Contract**

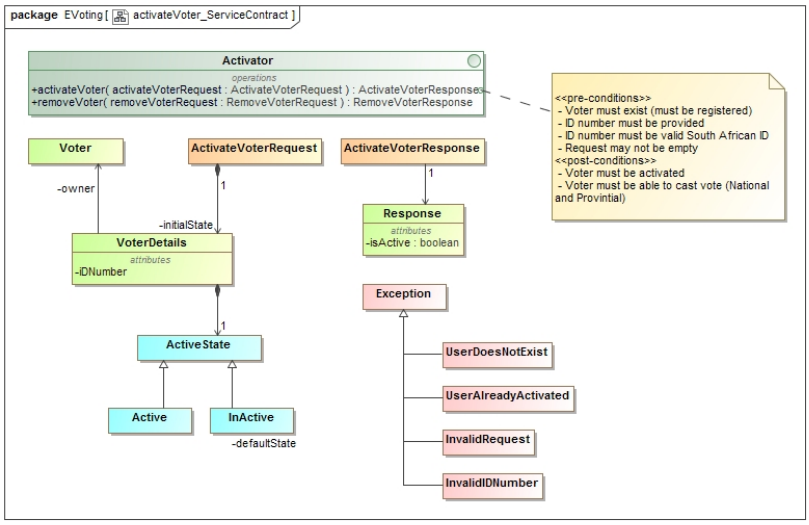

Figure 26: Activate Voter Service Contract

Activate Voter requires an ID number in the request which will be used to change the ActiveState state to active if all pre-conditions are met.

i. Pre-conditions
   - An ID number must be present in the request.
   - The ID number must be a valid South African ID number.
   - Voter must exist (must be a registered voter).
   - The voter must not already be registered.

ii. Exceptions
   - If the ID number is not a valid South African ID, the invalidIDNumber exception will be thrown.
   - If the user does not exist in the database, the userDoesNotExist exception will be thrown.
   - If the user is already activated, the userAlreadyActivated exception will be thrown.

iii. Post-conditions
   - The Voter's ActivateState must be Active.

(b) **Functional Requirements**

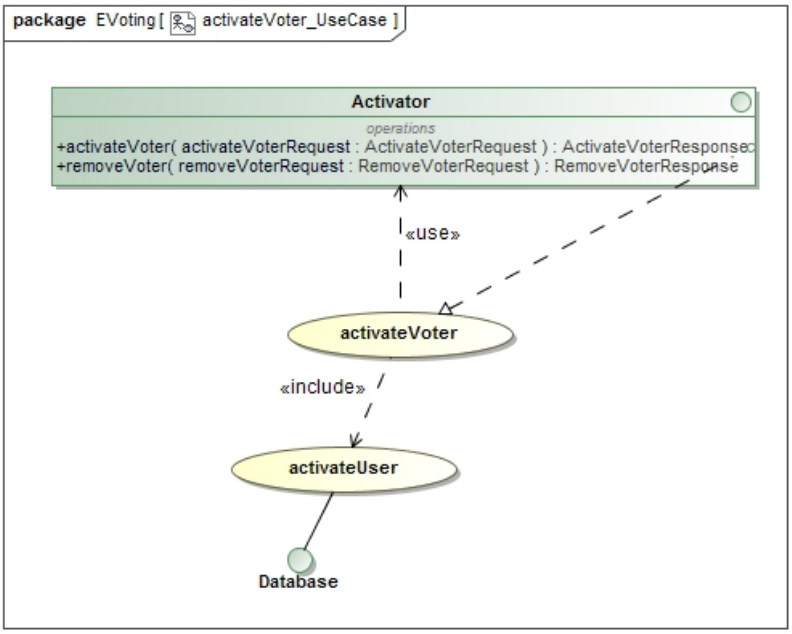

Figure 27: Activate User Use Case

The Activate Voter process will call the ActivateUser use case from the database module.

(c) **Process Design**

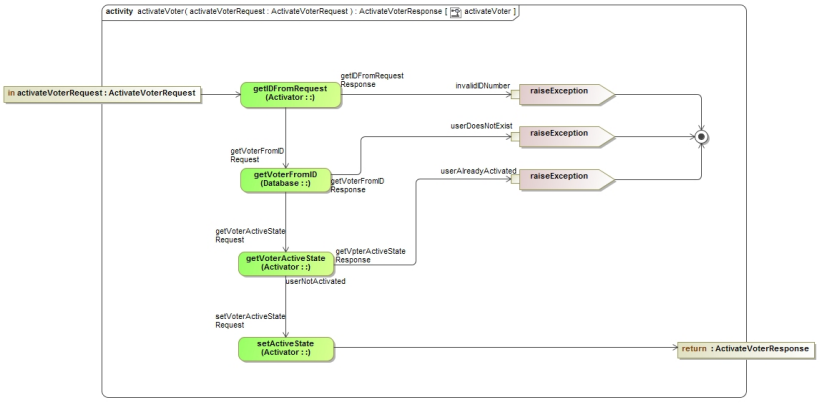

Figure 28: Activate User Activity

The Activate Voter process will first retrieve the ID number from the request and validate if is a valid ID number, after which it will get all the necessary Voter details from the database wich corresponds to that ID number. It then checks in what state the voter is to see if it has already been activated. If all

cases are valid, then the voter's ActiveState will change from Invalid to Valid.

2. **Remove Voter**

   (a) **Service Contract**

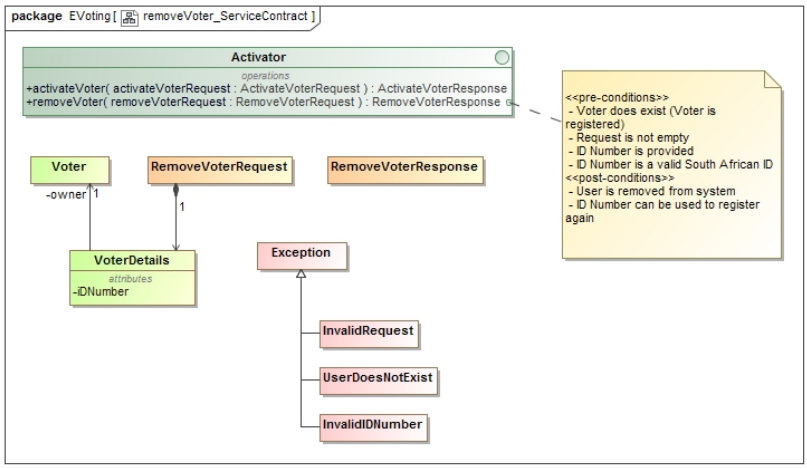

Figure 29: Remove Voter Service Contract

There are cases where the Activator will need to be able to remove a user. In the removeActivatorRequest, an ID number must be provided so that the voter associated with that ID number be removed from the system.

i. Pre-conditions
   - An ID number must be provided.
   - The ID number must be a valid South African ID.
   - The user must exist in the system.

ii. Exceptions
   - If the ID number in the request is not a valid ID number, the InvalidIDNumber exception will be thrown.
   - If the user does not exist in the database, the userDoesNotExist exception will be thrown.

iii. Post-conditions
   - The user is removed from the system.

(b) **Functional Requirements**

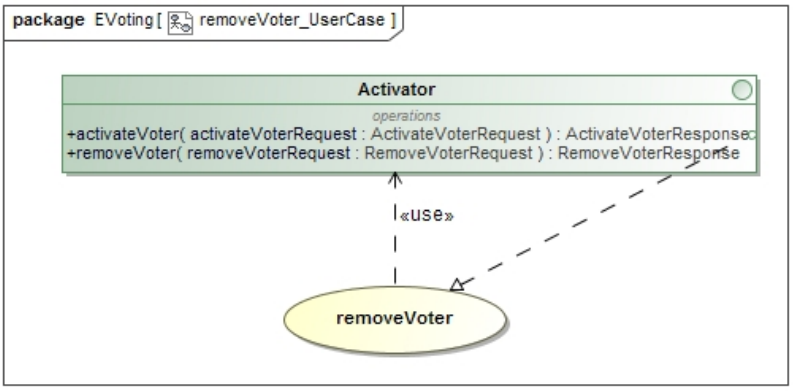

Figure 30: Remove Voter Use Case

The removeVoter will call a method of the database to remove the voter with that ID number. The database module is not fully documented as of yet.

(c) **Process Design**

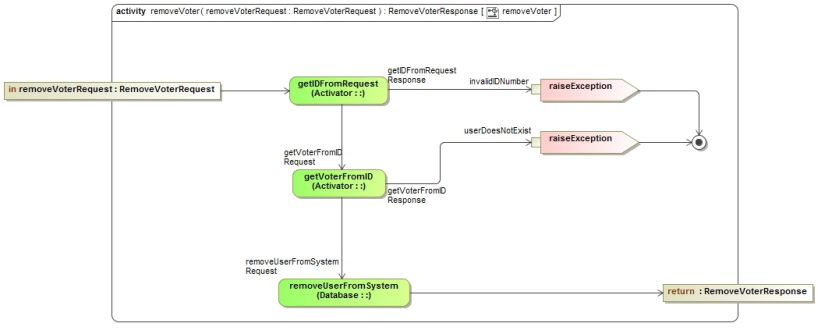

Figure 31: Remove Voter Activity

The Remove Voter process will first retrieve the ID number from the request and validate if is a valid ID number. If a user associated with that ID number is found, the process will call a function from the database module to remove the user from the system.

4.6 Party

1. Check National Number Votes

(a) Service Contract

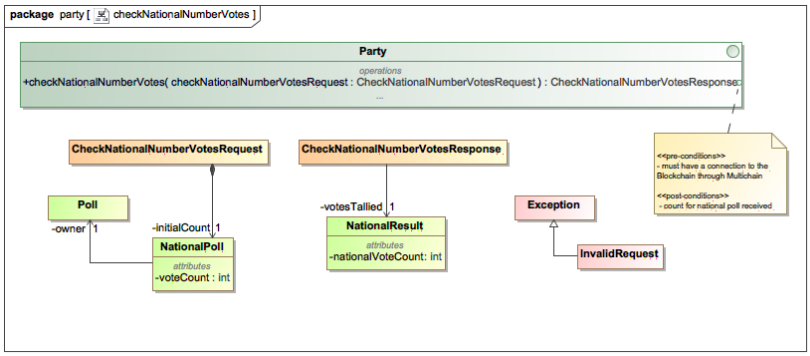

Figure 32: Check National Number Votes Service Contract

i. Pre-conditions
   - There must be a connection to the Blockchain through the Multichain

ii. Exceptions
   - If there is no connection to the Blockchain, the InvalidRequest exception will be thrown

iii. Post-conditions
   - The count for the National Poll is received

(b) Functional Requirements

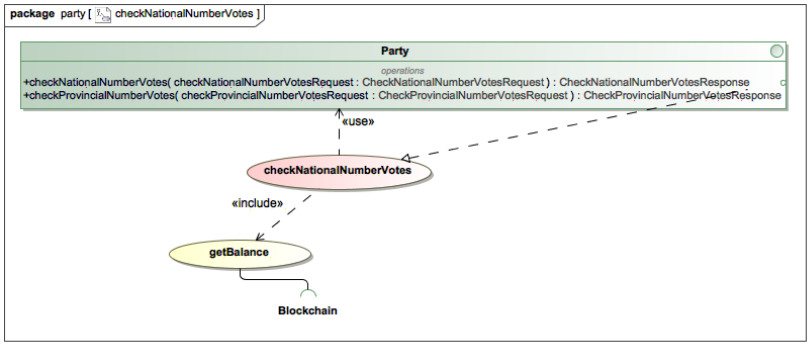

Figure 33: Check National Number Votes Use Case

(c) **Process Design**

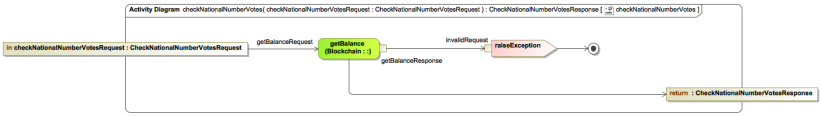

Figure 34: Check National Number Votes Activity

2. **Check National Number Votes**

(a) **Service Contract**

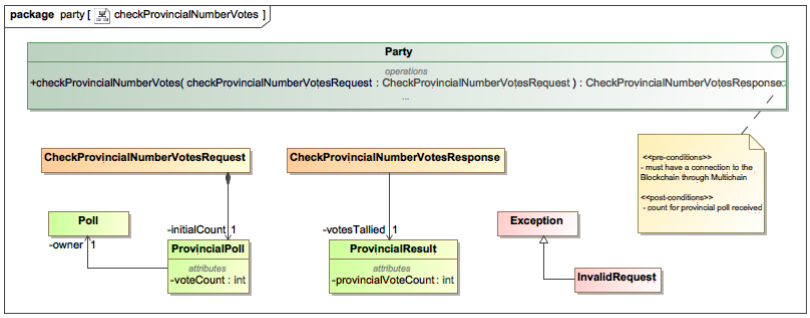

Figure 35: Check National Number Votes Service Contract

i. Pre-conditions
   - There must be a connection to the Blockchain through the Multichain

ii. Exceptions
   - If there is no connection to the Blockchain, the InvalidRequest exception will be thrown

iii. Post-conditions
   - The count for the Provincial Poll is received

(b) **Functional Requirements**

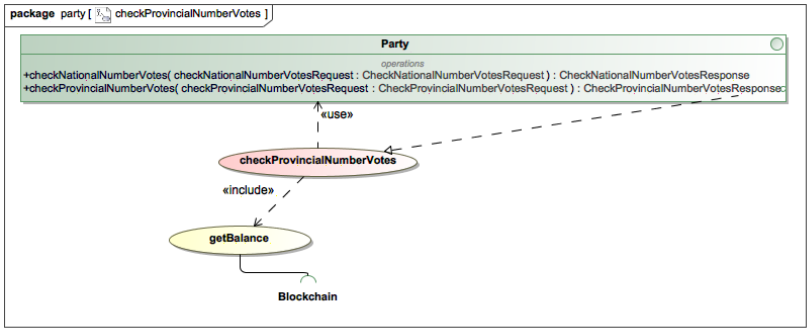

Figure 36: Check National Number Votes Use Case

(c) **Process Design**

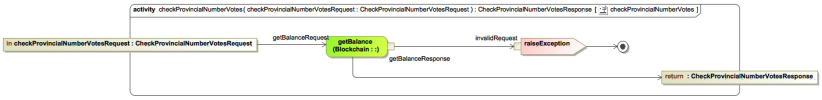

Figure 37: Check National Number Votes Activity

4.7 Domain Model

1. **Voter**

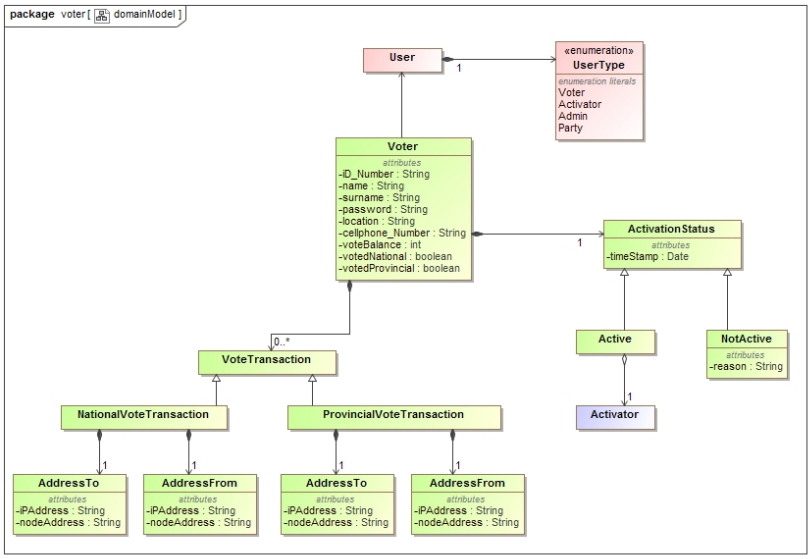

Figure 38: Voter Domain Model

2. **Party**

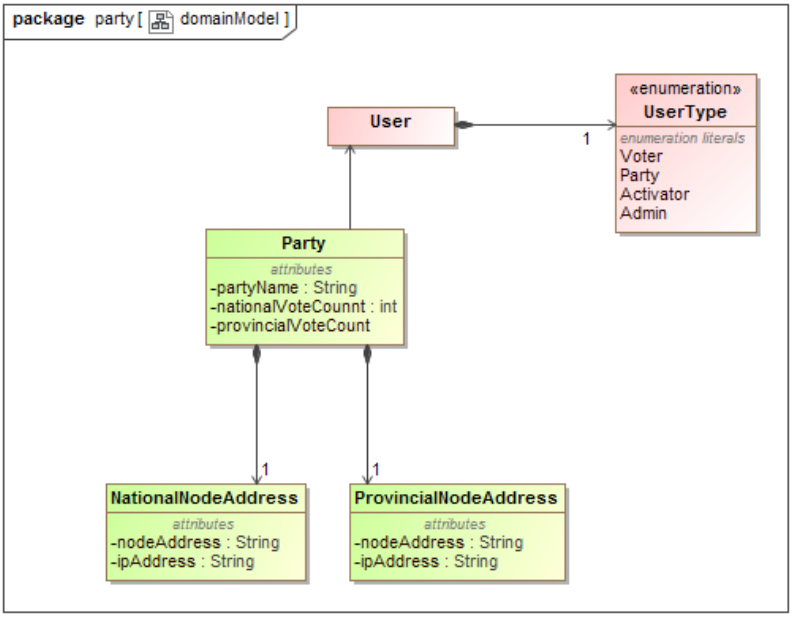

Figure 39: Party Domain Model

3. **Admin**

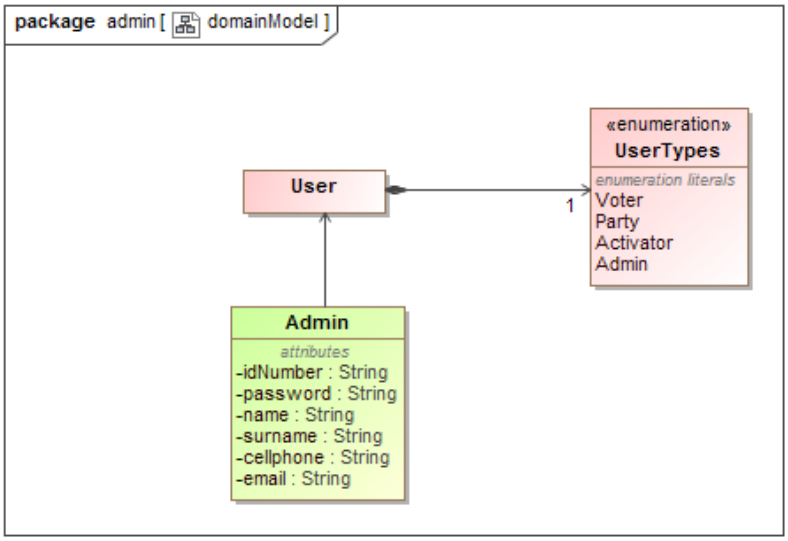

Figure 40: Admin Domain Model

4. **Activator**

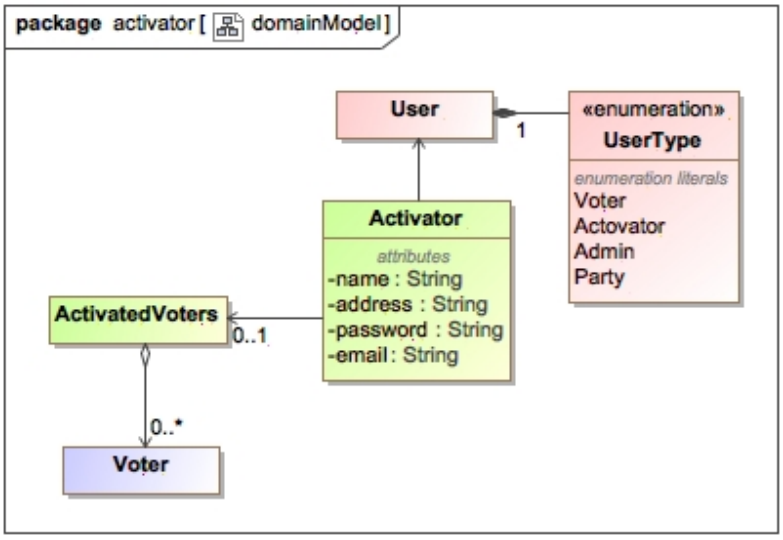

Figure 41: Activator Domain Model

5 Open Issues

5.1 GitHub Repository

For more information and/or further references, please follow this link, for access to Team CodeX's github repository.