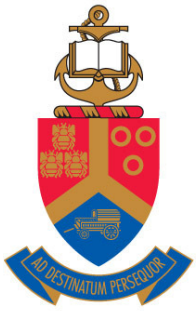

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS 301

Main Project

Functional Requirements Specification

Team Codex

Andreas du Preez	12207871
Frederick Ehlers	11061112
Azhar Mohungoo	12239799
Gift Sefako	12231097

Contents

List of Figures

1 Introduction

This document aims to specify the functional and non-functional requirements as well as the architectural requirements for an electronic voting system specified by CodeX and the client EpiUse.

It will serve as a means of communication between the client and developers as well as providing an elaboration and a clear description of its implementation specifications.

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. This project will help the elections in a number of ways by removing all possible fraudlent activities that come around elections in the world. Along with these benefits there are other benefits that will listed as well:

1. Things such as fake votes added by groups to assist a party with votes.
2. Votes that get lost or disgarded be it on purpose or not.
3. Incorrect counting.
4. Manual vote counting will no longer be required
5. Ease of acess to cast a vote
6. Secure voting
7. Prevent invalid votes to be casted making that vote not count (even if the invalid vote was intended).

3 Background

Elections are always a time were emotions are at a high and passion for a leader has never been more. Sometimes these emotions 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 possiblilities for unlawful activities to take place by using computers instead of humans. Computers are not influenced by emotions but only by the instructions given by their programming.

Addtionally to a secure and safe voting environment that allows ease of access from the voter's phone, computer or voting station it will also make the the voting process go a lot faster, have shorter queues to cast a vote and the progress of election and final result will be presented a lot faster with periodic updates as the system analyses the votes. This electronic voting system can be used in multiple scenarios not only in election. Any kind of fitting situation this system can be implemented for it. For instance it can

be used for national wide statistics gathering by completing a poll of some kind, where anonymity is vital or not.

4 Functional Requirements

4.1 Web Service

4.2 Multichain

4.3 Database

1. **Validate User**

   (a) **Service Contract**

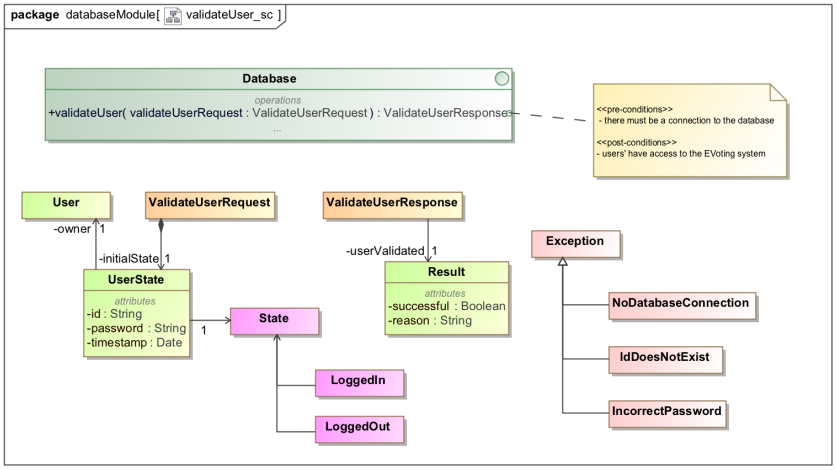

Figure 1: Validate User Service Contract

insert discription here

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**

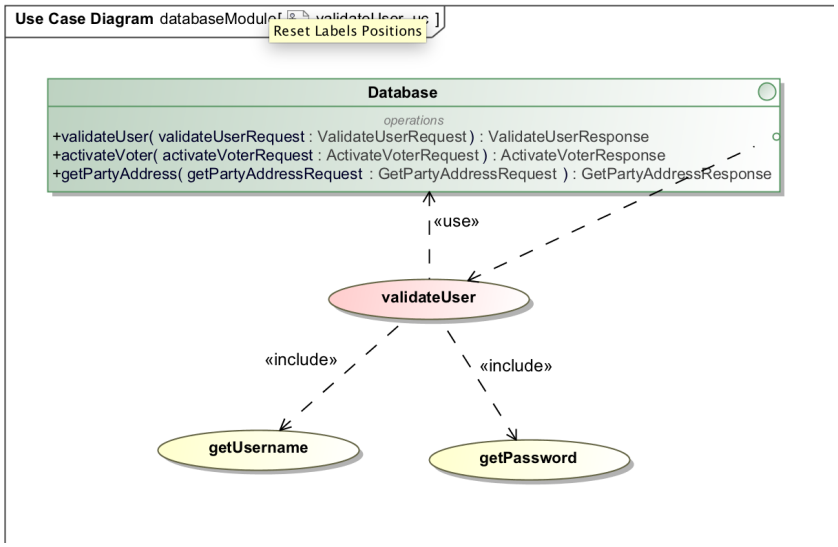

Figure 2: Validate User Use Case

insert discription here

(c) **Process Design**

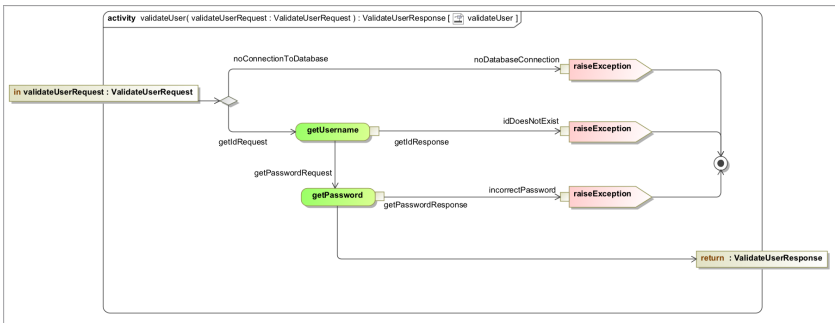

Figure 3: Validate User Activity

insert discription here

2. **Activate User**

(a) **Service Contract**

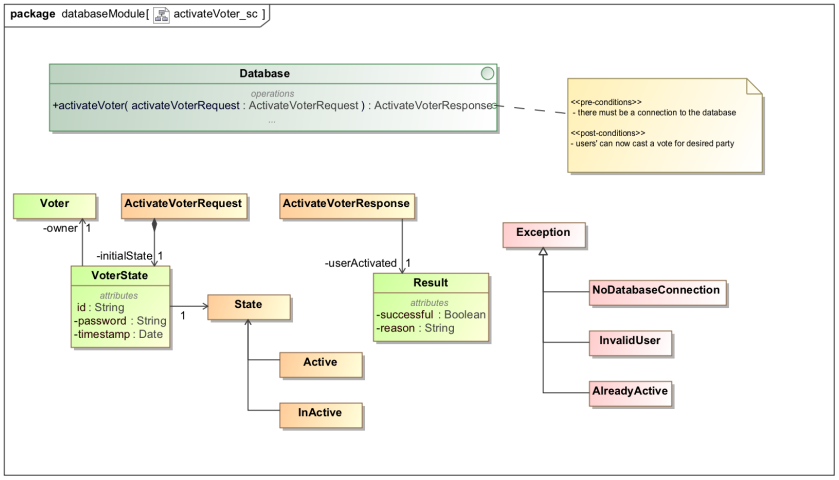

Figure 4: Activate Voter Service Contract

insert discription here

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**

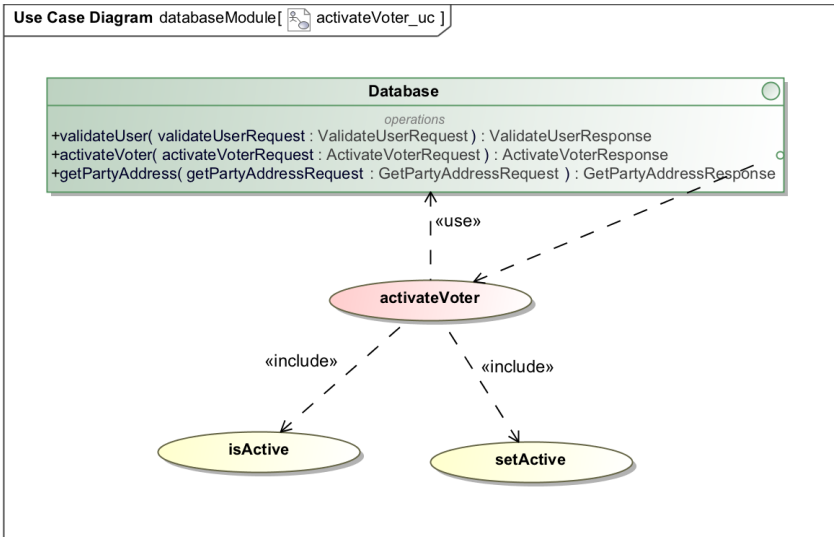

Figure 5: Activate Voter Use Case

insert discription here

(c) **Process Design**

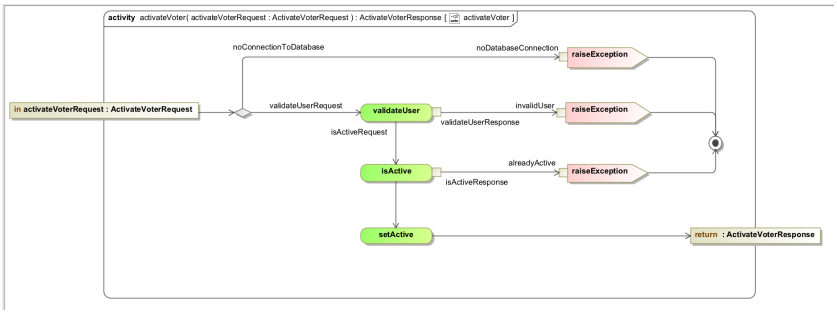

Figure 6: Activate Voter Activity

insert discription here

4.4 Security

4.5 Admin

4.6 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**

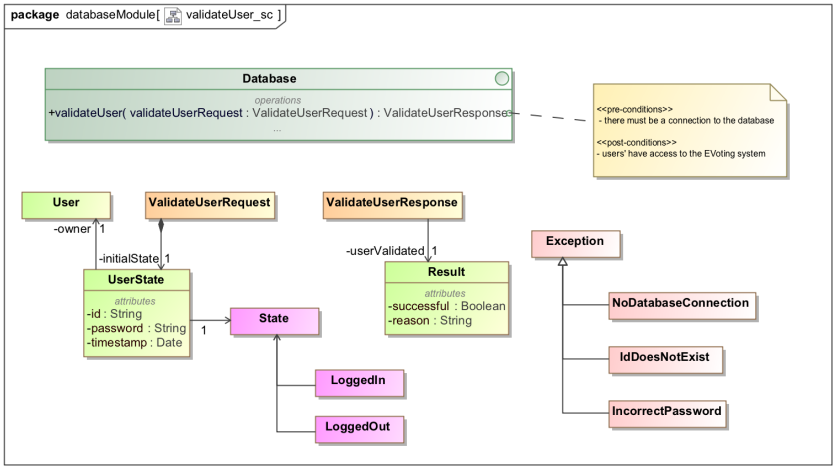

Figure 7: Validate User Service Contract

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**

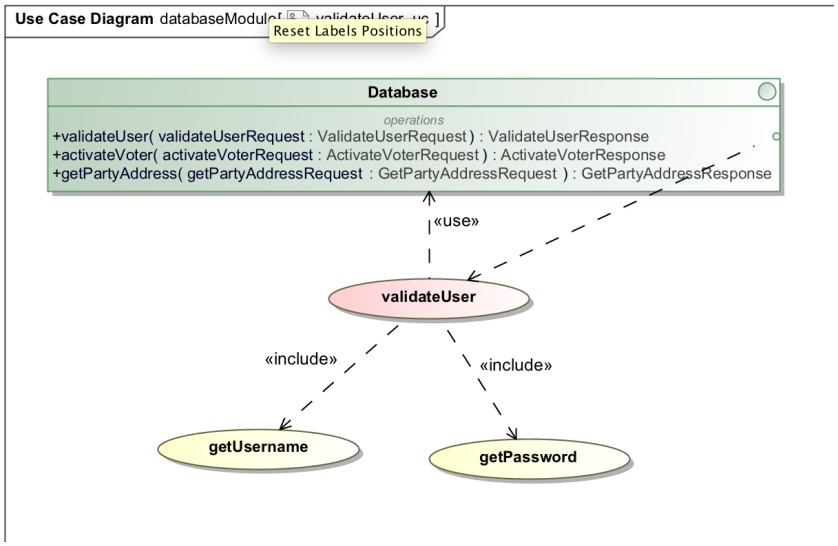

Figure 8: Validate User Use Case

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**

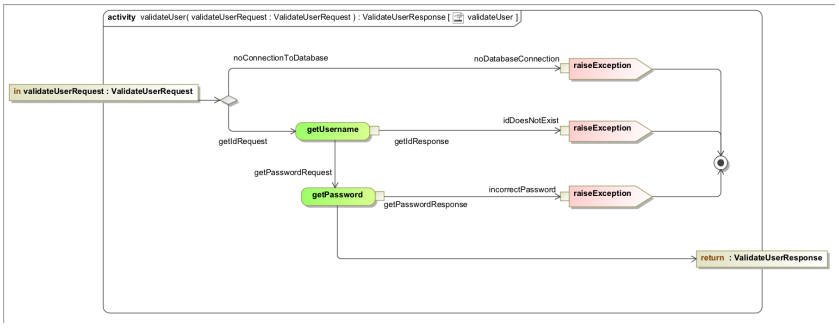

Figure 9: Validate User Activity

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**

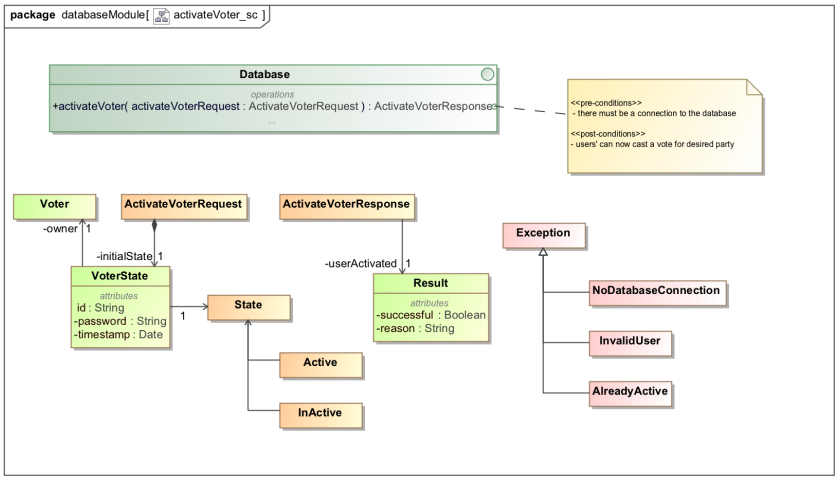

Figure 10: Activate Voter Service Contract

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**

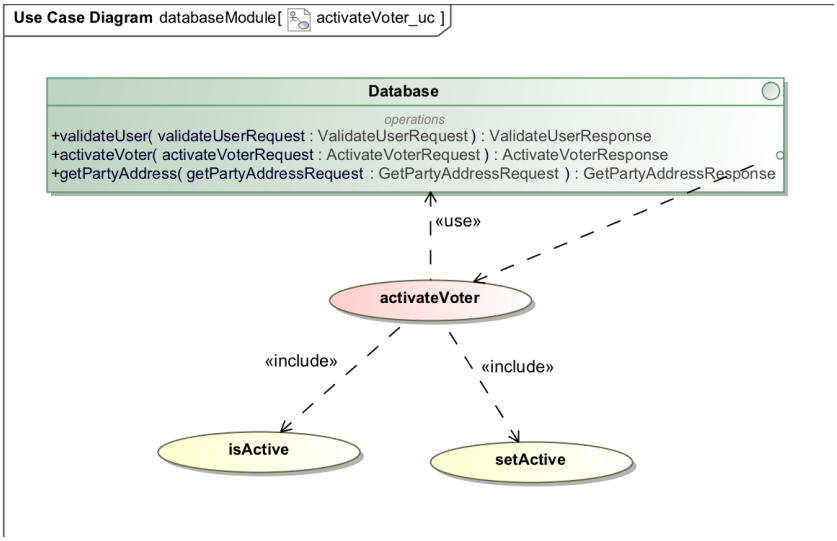

Figure 11: Activate Voter Use Case

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**

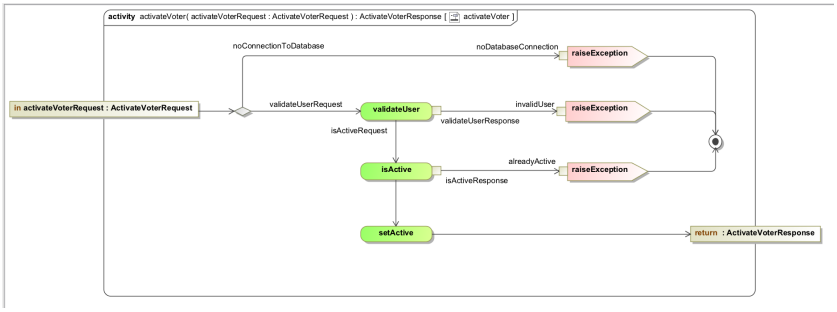

Figure 12: Activate Voter Activity

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.

1. **Cast National Vote**

   Casting a vote requires voter to be logged in.

   (a) **Service Contract**

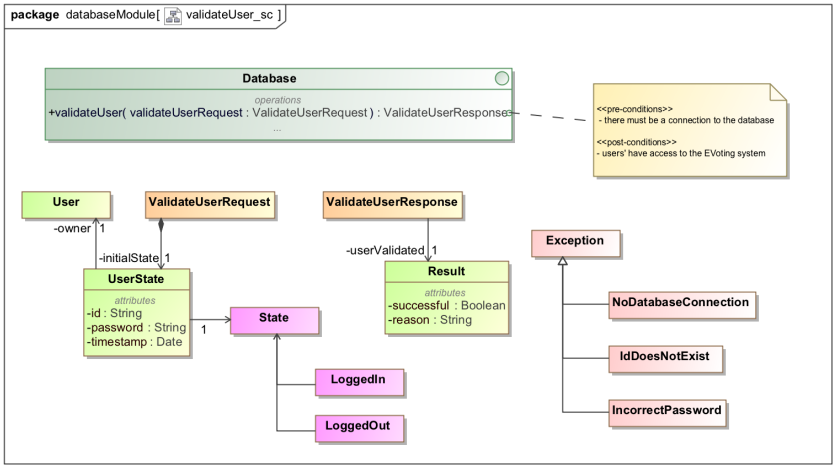

Figure 13: Validate User Service Contract

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**

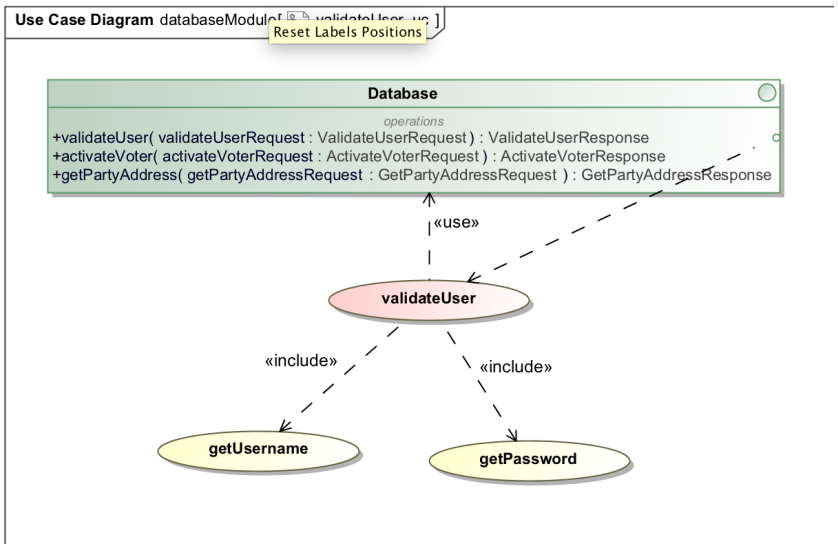

Figure 14: Validate User Use Case

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**

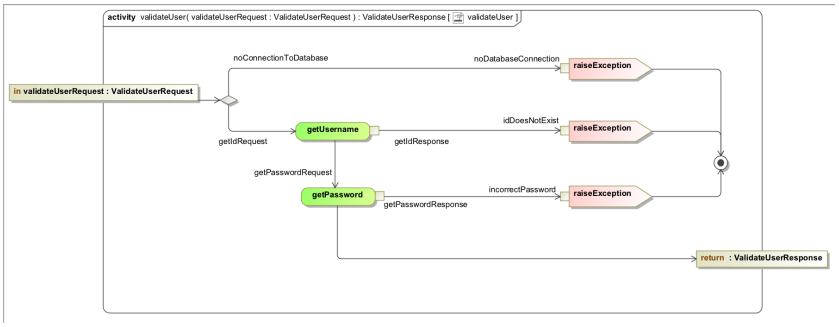

Figure 15: Validate User Activity

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.

4.7 Activator

4.8 Party

4.9 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.