**ITMD 511**

**Application Development Methodologies**

**Deborah Barndt**

**Erik Escobar-Ocon**

**Spring 2018**

**Mobile Voting Application**

**Abstraction**

In most recent years, the art of voting has welcomed the use of technology versus the ancient art of paper ballots. When one is voting in the present day, one can choose to either vote with the legacy way, that is using a paper ballot, or using a “digitalized” way to cast their ballots. These two ways of voting can have an impact on users not wanting to vote, as they have to physically visit a polling place to cast their votes. Since 90% of humans are attached to their phones, why not give them the opportunity to cast their votes using the device they are most comfortable with, their cell phones.

**Introduction**

For our final project, we wanted to focus on an application that isn’t usually seen today. We thought, what is one thing that could make people's’ lives easier? What is something that will get people to do more of if they had the choice to use a medium they use on a daily basis? Well, a voting app! Most people use mobile devices to shop, to check in on their homes, and even to bank. Why not have humans do more with the touch of their fingers! We understand that this project is a long shot, given the ethical reasoning behind it, but we will give it a shot. Ideally, this will make it easier for voting admins to not have to spend much setup at the polling sites, or waste paper on paper ballots.

**Purpose**

The purpose of this document is to provide the necessary foundation to start applying the idea of creating a mobile application for the American citizens to use when voting in the elections. This first documentation is to define the specifications for the initial minimal viable product.

**Project Scope**

The initial scope of this project is to do the very minimal basics to be able to register a user, login that user, and cast their votes. Their votes will be hashed and will not be visible to the end user unless they have the necessary tools to decode the hashes. There will be no social media connectivity, but users will be allowed to collect badges in the effort that it will provide some fun in voting.

**Requirements Specification**

The following are the requirements for the mobile voting application. The requirements are broken down into 5 sections needed to complete the MVP, the minimal viable product.

1. Registration/Login Requirements:
   1. Users will register by entering their personal info (street address in format XXXX, 10 digit phone number in xxx-xxx-xxxx format, dob in the format mm/dd/yyyy), voter id number, 6 digit registration code, password, and district number.
   2. Users will submit 2 factor authorization to login with the first phase using the username and password, and the second phase by entering a code provided via sms or email with a time limit of 3 minutes to enter the given code.
   3. For security purposes, user session will end after 5 minutes of app inactivity.
   4. User will receive notification message that the session will end in 2 minutes, 1 minutes, and 30 seconds due to inactivity.
2. Voting Requirements:
   1. Users can only vote while on the wireless network that is connected to the polling place, and within X feet from the premise (GPS location based).
   2. Users can select to vote for a party with one-click selection by clicking on the red colored elephant labeled republican or the blue colored donkey labeled democrat.
   3. Users can save vote selections for the current ballot, but can not submit multiple times and the information will be dropped after the polling window closes if it is not submitted.
   4. Ballot selections will be multiple choice with oval-shaped icons, and write-in selections will be text-based in Arial font with font size 20.
   5. User will receive notifications at 2 hours, 1 hour, and 15 minutes until closing of the poll window.
   6. User will receive a window to submit their poll selections (look up polling times for real elections).
3. Submission Requirements:
   1. If users leave the premise (network or outside of GPS range), the submission will be forfeited and they will not be allowed to start another session until they get back on the network.
   2. Users will receive confirmation/verification and upon successful vote submission, and a digital badge will be added to their account.
   3. Users will confirm vote selections before submitting via confirmation message with “Yes” or “No” options.
   4. Users will only be allowed to submit one ballot.
4. Error Handling/Security Requirements:
   1. Upon frozen app or crashed app, new session will be restarted and error report will be sent.
   2. Encrypts all the stored data to ensure data integrity and security.
   3. Application will not allow users to share on social media profiles.
5. UI Requirements:
   1. Each screen will have a header rectangle box at the top of the screen in a blue with hex code #3B7FBE, and header text in Franklin Gothic Heavy and font size 40 in white with red border with hex code #C73338.
   2. The application will have a red colored elephant for the Republican Party and a blue colored donkey for the Democratic Party allowing for one touch voting for candidates in chosen party.

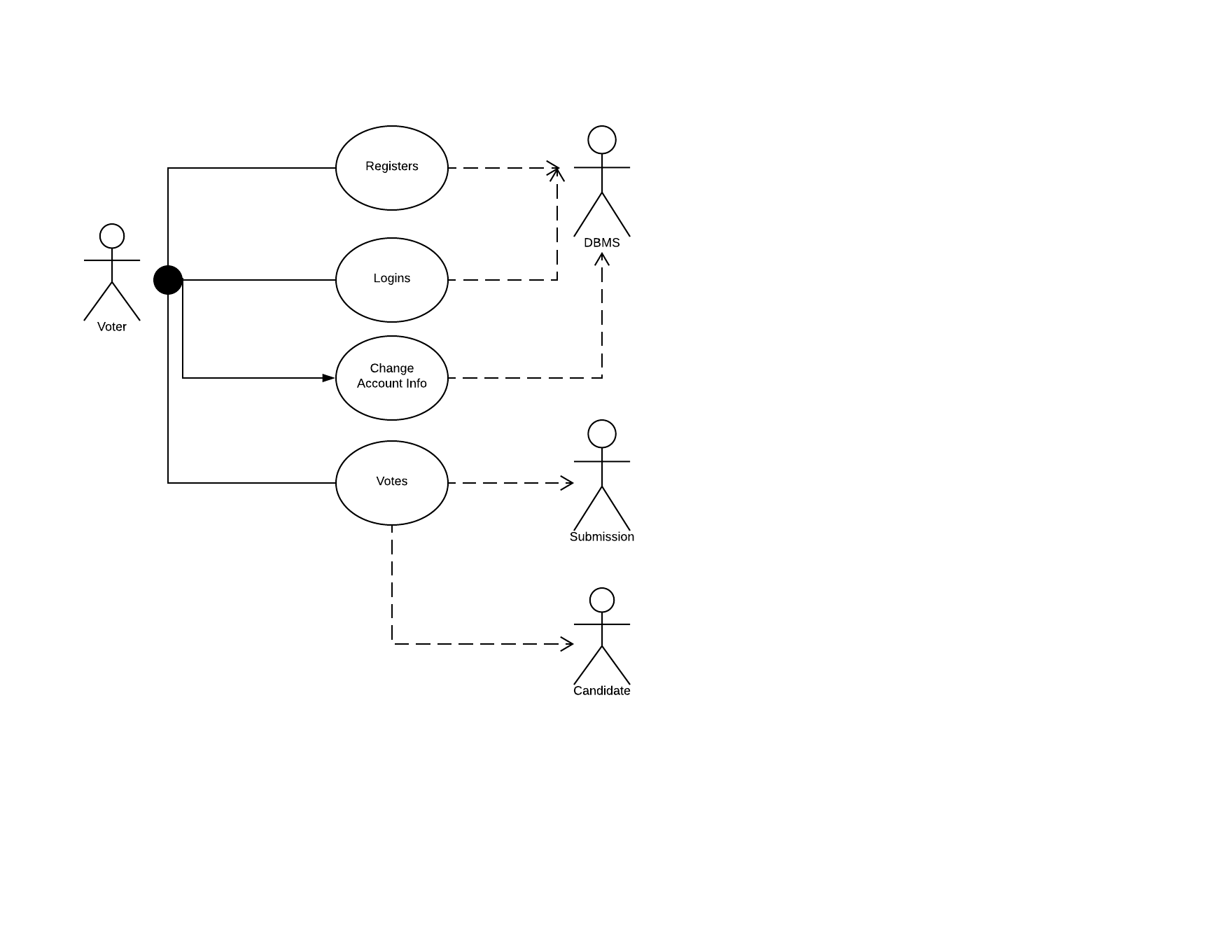
**Use-Case Modeling**

In this section, we will discuss the use case models and user scenarios for the mobile voting application.

Let’s start with the following user scenario and then move into the use case for the application. The persona here will be a John Doe who is voting from his cell phone.

*“John Doe, a 35 year old resident from Chicago, Illinois, wants to vote in the upcoming election. John is very active on his phone, using it to check in on his home, mobile banking, and sending emails. John is not a huge fan of old-school voting ballots because he is in with the latest and greatest technology. He feels that it is very archaic. John downloads the new voting application before voting. He registers with application, and confirms his login credentials alongside a 2 factor authorization code sent to his phone. While on the voting premise and on the voting network, John submit his ballot and receives a confirmation email stating his vote was successfully received. John spends 2 minutes voting through the app.”*

Now, let us get into the use case for this application. The following is the use case diagram, which will be followed by the use cases itself.



*User installs voting application.*

*User registers with the application.*

*User logins.*

*Voting application sends 2 factor authorization to user.*

*User is able to vote or change their information.*

*User chooses to vote.*

*Candidates are shown for positions.*

*User chooses candidates for positions up for vote.*

*User confirms and submits ballot.*

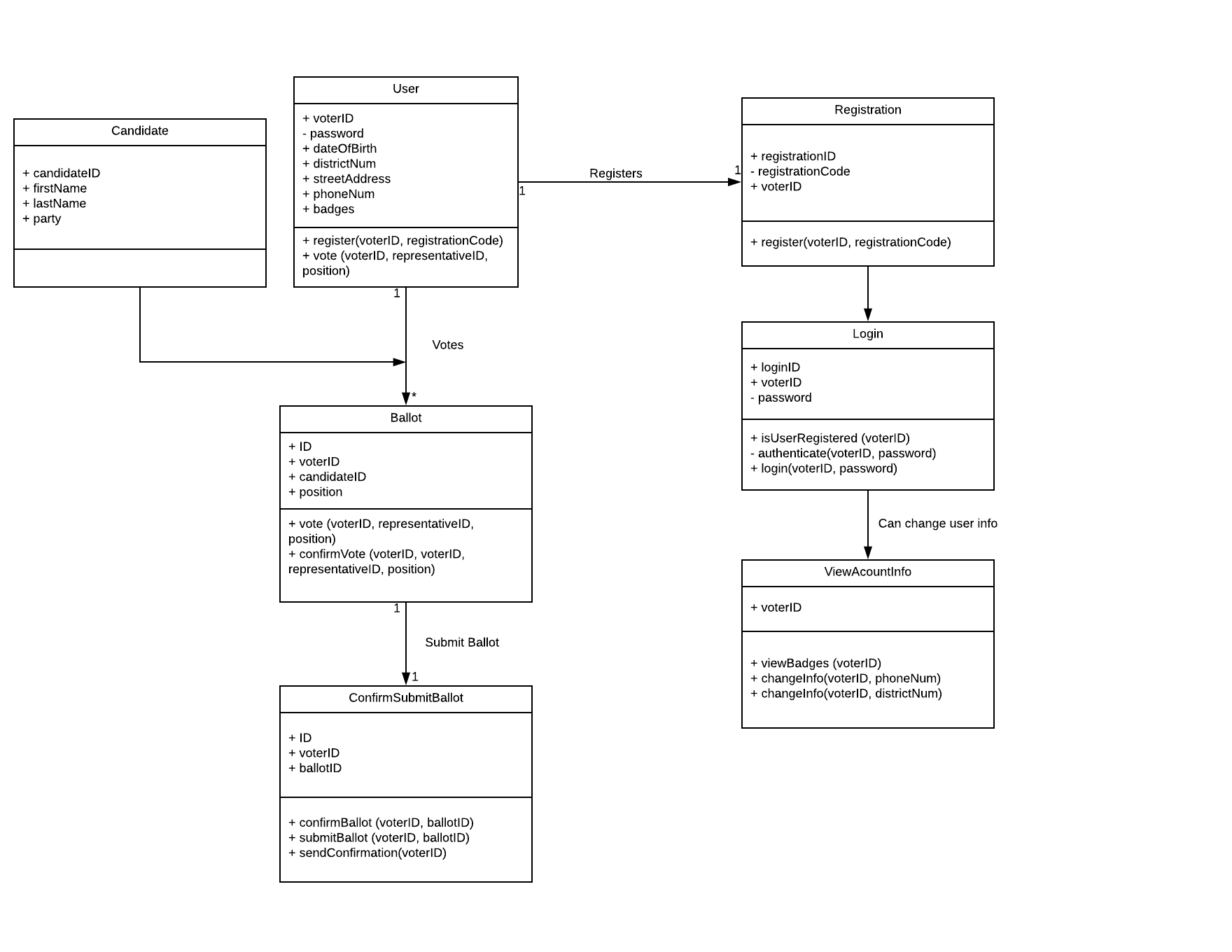
*Application send user an email confirmation.*

*User has voted successfully.*

The above use case details how the user will register, login, and vote for their representative. For this initial release, we aren’t targeting more than these functionality along with the user being able to change their account information. As you can see from the use case, the user will need to register before being able to log in to vote for their candidates. Once logged in, the user can choose to change their account or continue voting. Once the use is satisfied with their ballot, they will need to confirm before submitting the vote. An email sent to the user will signify a successful voting attempt. The user will not be able to vote again as you can only vote one time per election.

**Class-Based Modeling**

In this section, we will be explaining the classes for the mobile voting application.

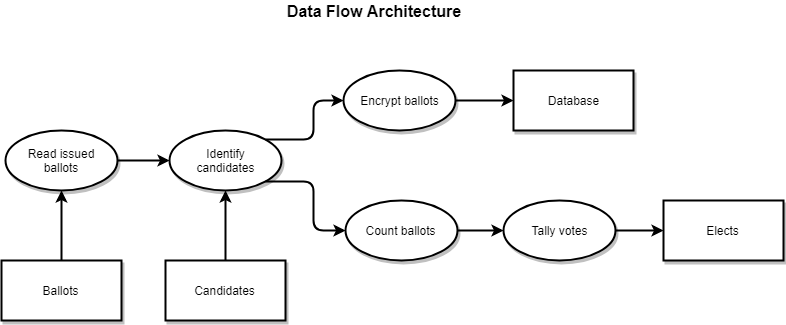


For starters, six classes will likely be implemented to complete the voting application. This does not include any UI classes, but more about functionality and entities. The first class is obviously the candidate which just creates a candidate object with their first and last name, and position they are running for. The second is the user class which will handle and store the functionality of the user registered to the application this will account for all account relation information as well as methods to register and vote. The registration class will register the user and store them into the DBMS. The login class will authenticate the user for usage, as well as perform checks to see if the user is registered. The ViewAccountInfo class will handle the changes of account information as well as storing/showing the badges the user will accumulate.

There are two classes left which will do the bulk of the work. The first of the two is the ballot entity class which will handle the choosing of candidates based per position available. This class will confirm the votes. The last class is actual submission of the ballot. This class will confirm the ballot with the user, submit the ballot, and send a confirmation email to the user stating they have successfully voted.

**Architecture Specification**

For the architecture style, we chose to tackle it using data flow model as votes will be flowing through the system. The votes/ballot is essentially the data that will be transported. The diagram below shows the flow/transportation of this data.

****

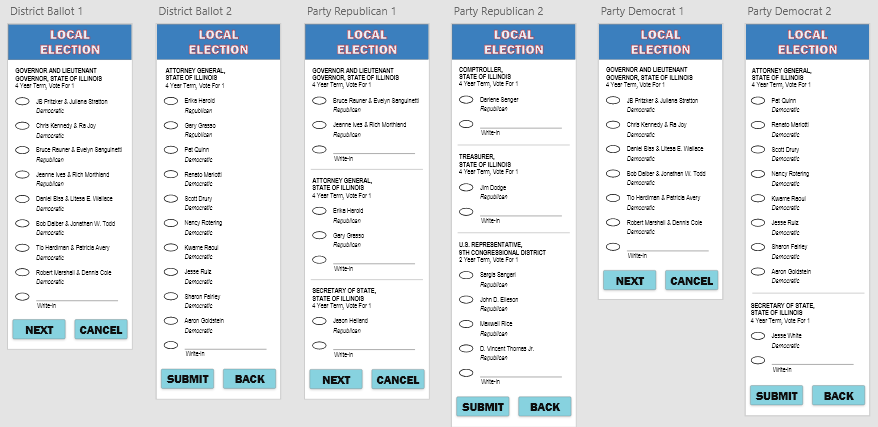
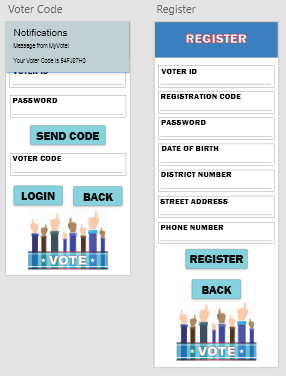
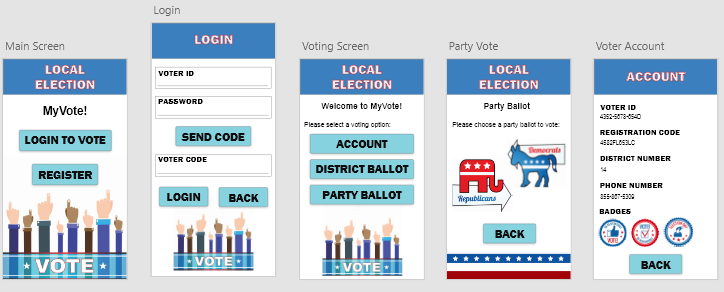
As you can see, the data(votes) will flow through different filters, or components, that will each perform a different action on the data. This data flow model here shows abstraction by hiding away parts of the data. This also deals with separation of concerns, as different filters perform different actions at different parts of the flow. This also provides reusability as the filters are separate methods that can be called whenever. This architecture is also very simple, which is perfect for this application as the proof of concept is just that, simple. Another plus is that it is very maintainable.

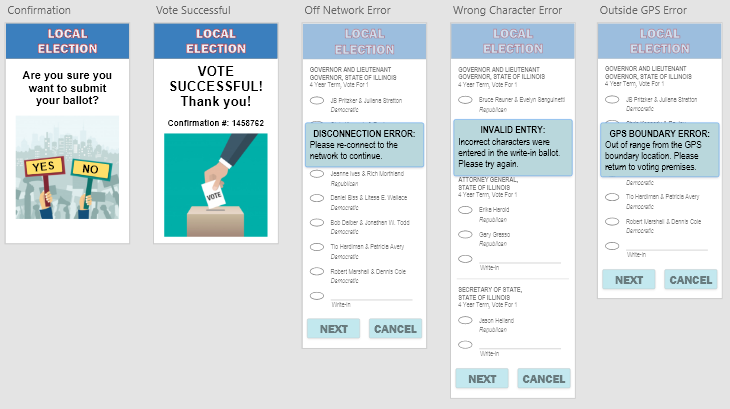
Object oriented architecture could have also been taken into consideration for this approach. Since we deal with objects, this would also make sense. This approach allows for the encapsulation as we will be hiding parts of the vote object in our application. OO design also approaches the problem of separation of concern, as it will deal with information hiding as well as modularity. This application will be broken down into many sub-methods, not just a few main methods. This provides for reusability of different modules within the application, for future use if needed due to the abstraction of the architecture. In our opinion, this will make refactoring beneficial and not difficult as it will be loosely coupled. This also makes the testing of the application very testable, which is always a good thing with regards to the quality of the product.

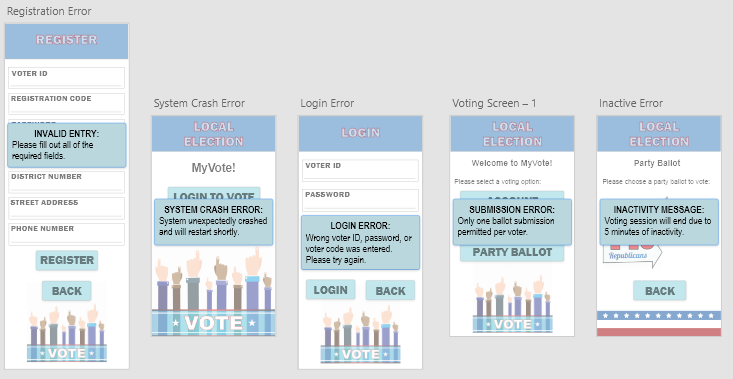
**Prototype**

Our prototype is an interactive simulation model that simulates the user using the application on their phone. Along each of the top of the screens is the header banner indicating what some of the screens are such as login, account, and register. When the simulation is run, the user can interact with various buttons on the screens and navigate between them. Our interactive prototype also features some of the error screen messages that the user would see while using the app incase an error has occurred.

Our prototype implements many of the intended view modes described in the requirements. The user starts on the main screen of the application, where they can either login to vote or register. The login page requires a two-factor identification in order to move on to the next screen, and is essential for the security of the application to verify the user. Once the user logs in to the application, they are taken to the voting screen which features three option buttons to view their account, vote on the district ballot, or vote on a select party ballot. The party ballot screen allows the user to choose to vote for either a republican or democratic ballot. The account screen displays the user’s voter ID, registration code, district number, phone number, and the digital badges that they earn when they successfully submitted their ballot. Upon ballot submission, the user is prompted to verify if they would like the submit their ballot by clicking on “Yes” or “No”. If the user selects “No”, they are taken back to the ballot screen to continue voting. If the user selects “Yes”, they are taken to the confirmation screen and given a confirmation number, and a new digital badge is added to their account.

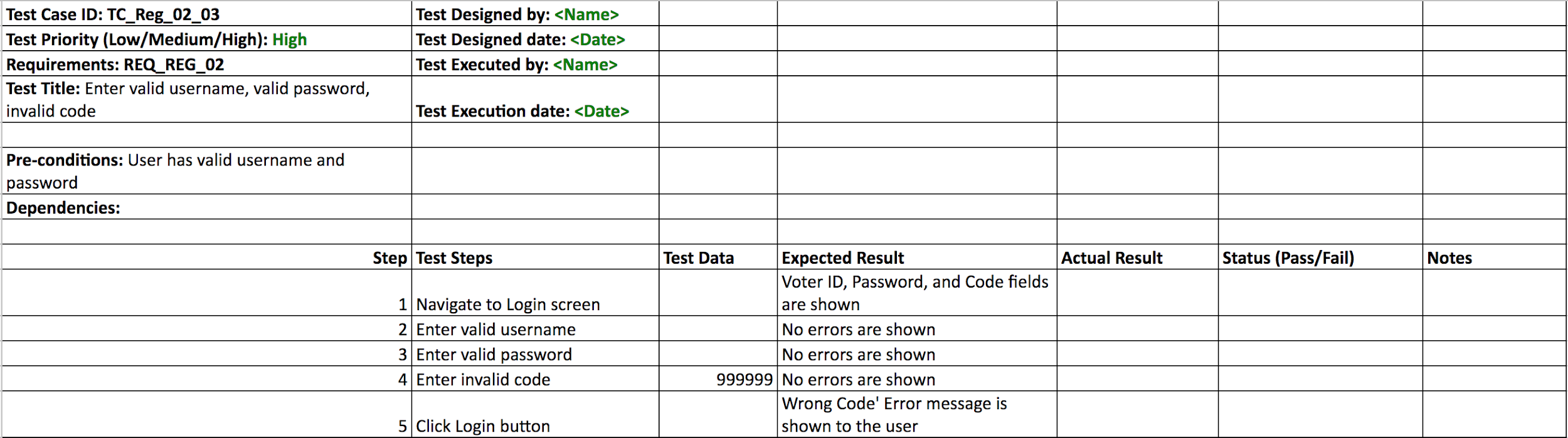


****

****

**Test Plan and Test Cases**

Normally, there would be a traceability matrix composed before the testing occurs. The purpose of this matrix is to map the test cases to the requirements. In a real world situation, there will be multiple test cases per requirement. You have your happy path, that is the correct flow and the correct usage of the app. You also have your negative and error cases where tests validate input, handle user errors, and test that what is not supposed to happen, actually doesn’t happen. As you can see, the ratio here can be one requirement to many test cases, thus accumulating to a long test plan. For this section, we have mapped about five test cases per requirement. Out of these 5 test cases, we only test out 1-2 of them, meaning we list the steps for them in the attached excel workbook. Below is an example of one of our test cases.



As you can see, we use a generic template for writing our test cases out. This includes the test case number so we can identify the test and quickly refer to it. We also include the requirement it is testing by referencing the ID of the requirement. This helps to make mapping easy. Priority is also included as we want to know how important the test is. The pre-conditions are the variables that will be known to the tester beforehand. Lastly, we have the step to produce the test. This includes a step number, the action to be performed, and any test data to be inputted as we want to be able to reproduce with the same exact steps if any bugs were to arise. Also included are the expected and actual result. Testers need to know the expected outcome, and need to validate the actual output to the expected to see if the application functions correctly. Lastly, we have status and notes. Failures need to be known at what exact point did it occur. Any important notes about the test should also be well documented.

As you can see. It can take much time to create a test plan with many cases in it. This is the reason why we scaled down and only wrote out one to two test cases per requirement. These test cases are also written in a way that anyone should be able to test the application with minimal knowledge of it. There should also be no more than 10 steps per test case as this can make the test scope increase. A proper test case review should be done beforehand, so that all cases are peer reviewed and any discrepancies are adjusted.