Assignment 4 - Open Source Software: Analysis and Design

CSC 415 Fall 2019 – Jordan Sinoway

Project Title: SOLVER (Student On-Line Voter Engagement and Registration)

GitHub Repository: https://github.com/jbennet-t/SOLVER

VM: sysadmin@csc415-server20.hpc.tcnj.edu

VM Path: SOLVER/src/test2

Website Hosted on VM: http://csc415-server20.hpc.tcnj.edu:3000/

Use Case Descriptions:

1. Use Case: View List of Upcoming Elections
    - Primary Actor: All site users including admin
    - Goal in Context: To view list of all upcoming elections
    - Preconditions: States, counties, elections have been added to database
    - Trigger: User would like to see list of elections
    - Scenario:
        1. User navigates to SOLVER website
        2. User clicks "View Elections" on menu bar
        3. User selects from a list of states
        4. User selects from list of counties
        5. List of elections for that area is presented to user
    - Exceptions:
        - User selects incorrect state/county
        - Desired state/county not present
    - Priority: High priority, to be implemented after basic functions
    - When available: 1st Increment
    - Frequency of Use: high frequency
    - Channel to Actor: Web based browser
    - Secondary Actor: None
    - Channel to Secondary Actors: N/A
    - Open Issues:
        - How many states/counties included?
        - Making sure elections that have ended are not included in results
        - What format is list presented in, and with what options for each election?

2. Use Case: Select and View Specific Election and Details
    - Primary Actor: All site users including admin
    - Goal in Context: To view specific election and details
    - Preconditions: User has selected state/county, elections have been added to database
    - Trigger: User would like to view specific election and details

- Scenario:
    1. User clicks on an election from list
    2. Details regarding election (date, candidates, where to vote) are presented to user
- Exceptions:
    - User selects incorrect election
    - Desired election not present not present
- Priority: Medium priority, to be implemented after upcoming election list
- When available: 2nd Increment
- Frequency of Use: moderate to high frequency
- Channel to Actor: Web based browser
- Secondary Actor: None
- Channel to Secondary Actors: N/A
- Open Issues:
    - How is information formatted?
    - Specifically, what information is displayed?

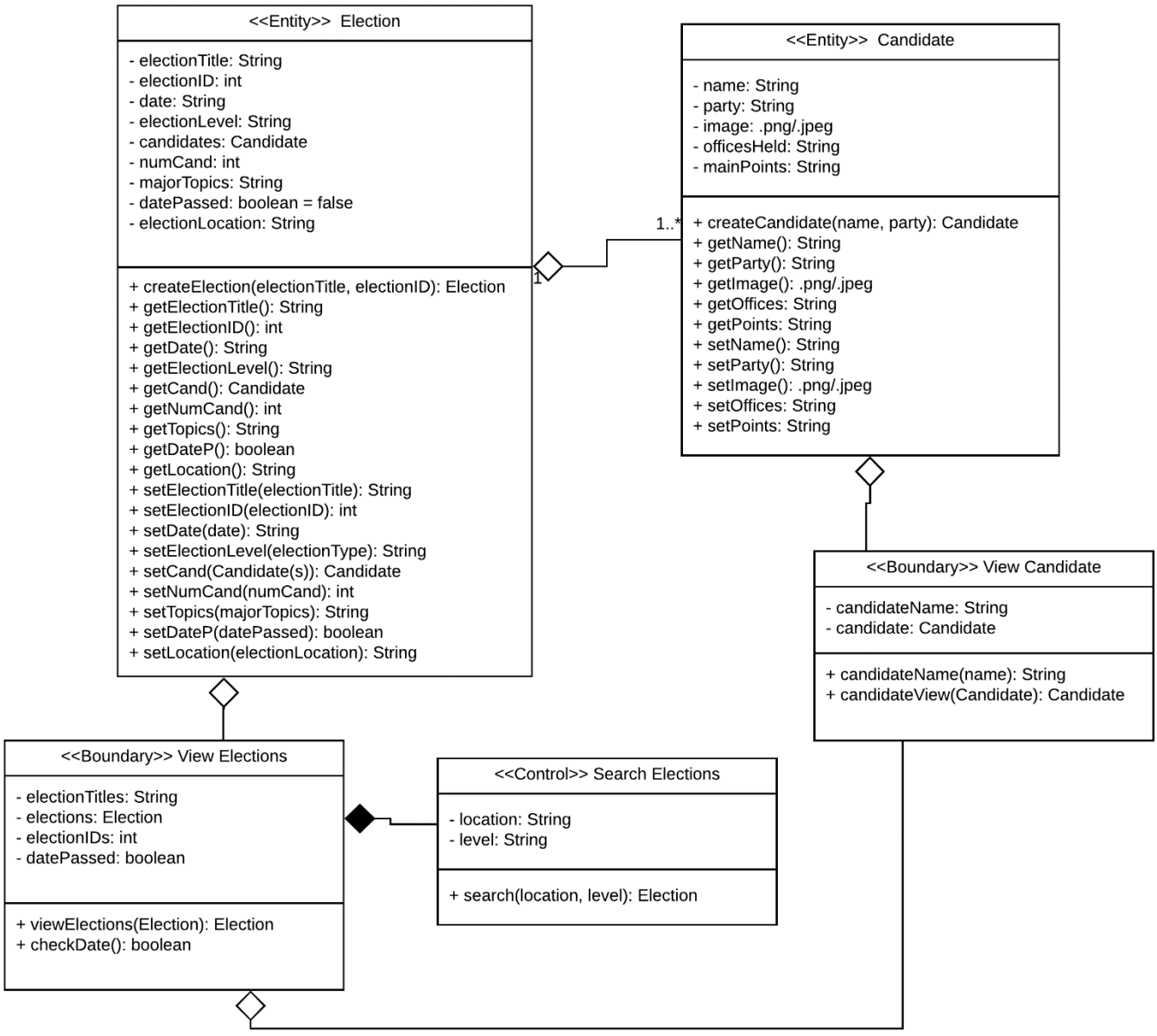
3. Use Case: Select Candidate, View Affiliation and Policies
    - Primary Actor: All site users including admin
    - Goal in Context: To view policies for individual candidates
    - Preconditions: User has selected election, candidates have been added to database
    - Trigger: User would like to view candidates' policies
    - Scenario:
        1. User selects candidates name on specific election page
        2. Details regarding candidates are presented to user
    - Exceptions:
        - User selects incorrect election
        - Desired election not present not present
    - Priority: Medium priority, to be implemented after election details
    - When available: 2nd Increment
    - Frequency of Use: low to moderate frequency
    - Channel to Actor: Web based browser
    - Secondary Actor: None
    - Channel to Secondary Actors: N/A
    - Open Issues:
        - How to ensure no bias for information?
        - What information is displayed?

4. Use Case: Navigate to Registration Page
    - Primary Actor: All site users including admin
    - Goal in Context: To view page which describes how to register to vote
    - Preconditions: None
    - Trigger: User would like to view details of how to register to vote
    - Scenario:
        1. User navigates to SOLVER website
        2. User clicks "How to Register" on menu bar
        3. Details of how to register are presented to user
    - Exceptions:

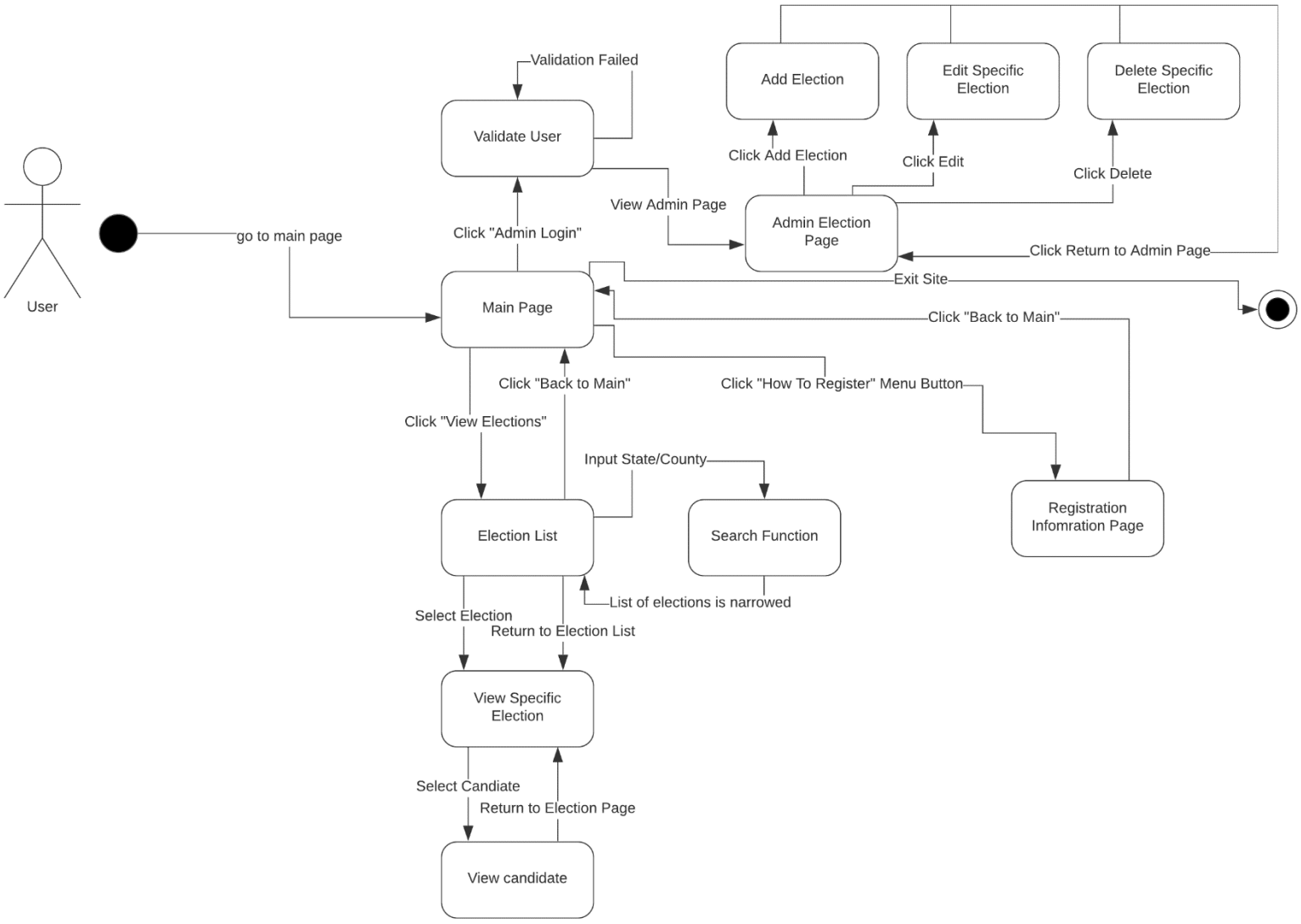
        - User selects wrong menu button
    - Priority: High priority, to be implemented after basic functions
    - When available: 1st Increment
    - Frequency of Use: high frequency
    - Channel to Actor: Web based browser
    - Secondary Actor: None
    - Channel to Secondary Actors: N/A
    - Open Issues:
        - How specific should registration info be?
        - Add link to external registration sources?

5. Use Case: Adding, Updating, and Deleting Elections and Candidates
    - Primary Actor: Admin Only
    - Goal in Context: To add new elections and candidates
    - Preconditions: None
    - Trigger: Admin wants to add/update/delete elections and
    - Scenario:
        1. Admin navigates to SOLVER main page
        2. Clicks "Admin Login"
        3. Logs in with admin credentials
        4. Page with list of elections and buttons to add, update, and delete elections is presented to admin
        5. Admin selects option, adds/updates/deletes election/candidate as desired
        6. Admin logs out
    - Exceptions:
        - Admin login credentials are invalid
        - Admin adds candidate/election that already exists
    - Priority: High priority, to be implemented after basic functions
    - When available: 1st Increment
    - Frequency of Use: high frequency
    - Channel to Actor: Web based browser
    - Secondary Actor: None
    - Channel to Secondary Actors: N/A
    - Open Issues:
        - Separate pages for candidate and election creation/editing?
        - Having better admin login method than button on front page
        - Max # of elections?

Detailed Design Class Diagram:

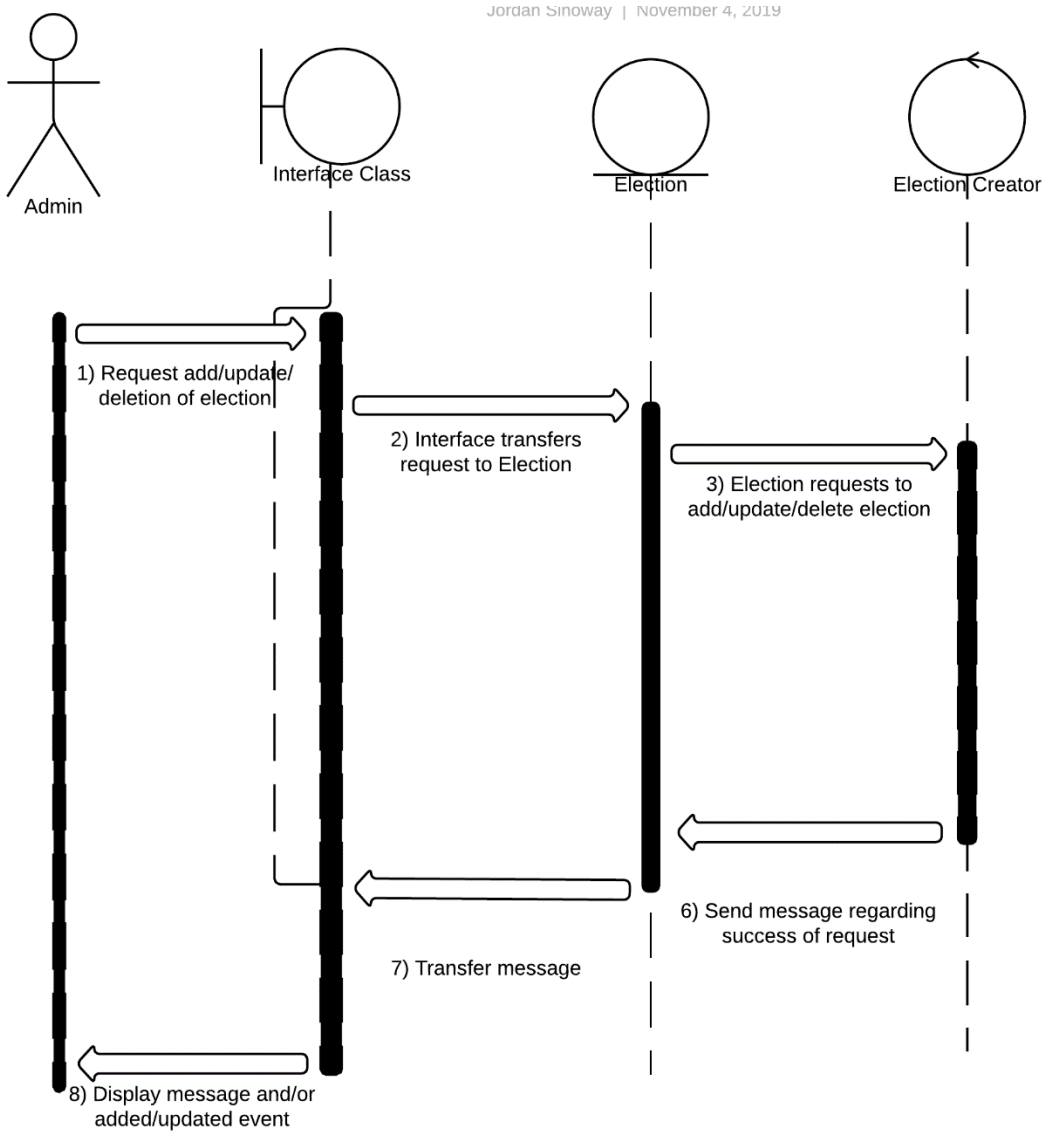


State Chart:

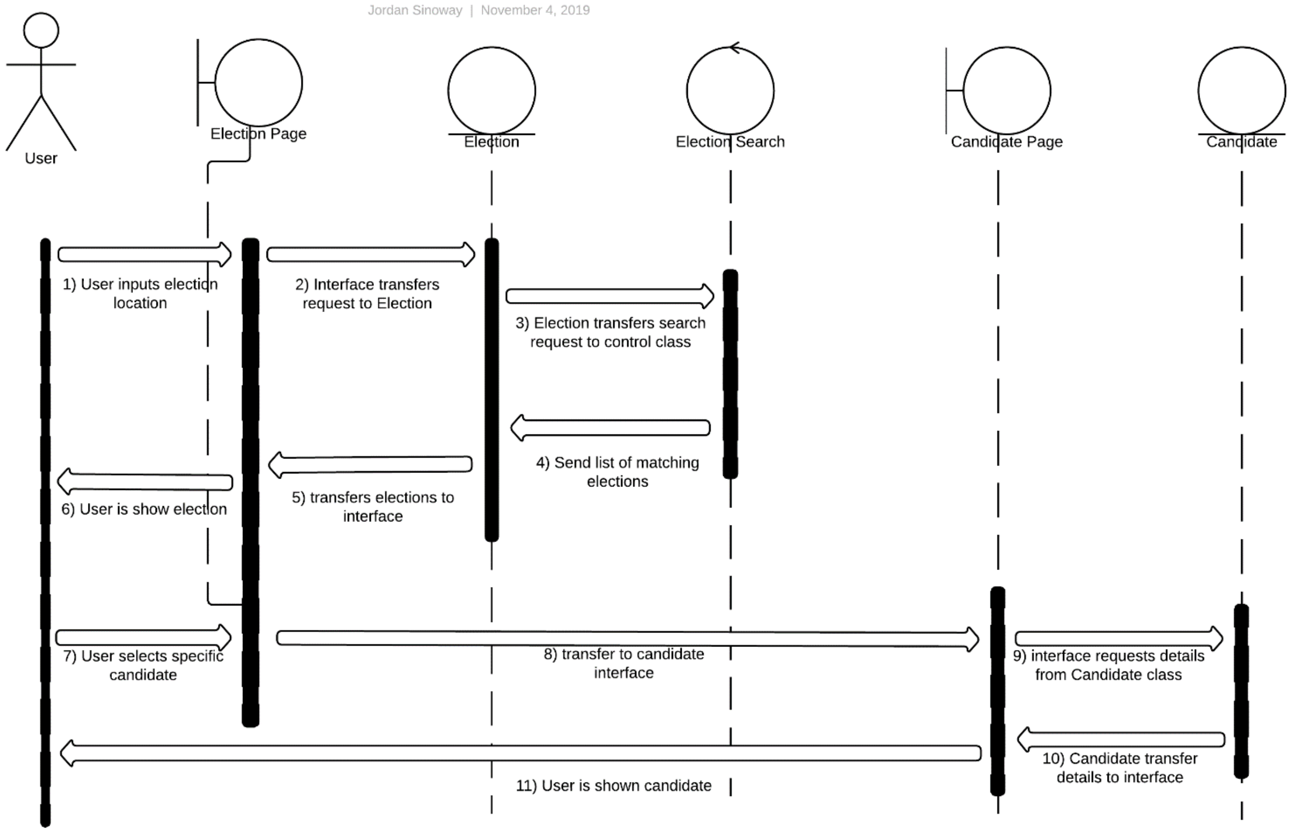

User
go to main page
Validate User
Validation Failed
Click "Admin Login"
Main Page
Add Election
Edit Specific Election
Delete Specific Election
Click Add Election
Click Edit
Click Delete
View Admin Page
Admin Election Page
Click Return to Admin Page
Exit Site
Click "Back to Main"
Click "Back to Main"
Click "How To Register" Menu Button
Click "View Elections"
Input State/County
Election List
Search Function
Registration Infomration Page
List of elections is narrowed
Select Election
Return to Election List
View Specific Election
Select Candiate
Return to Election Page
View candidate

System Sequence Diagrams:

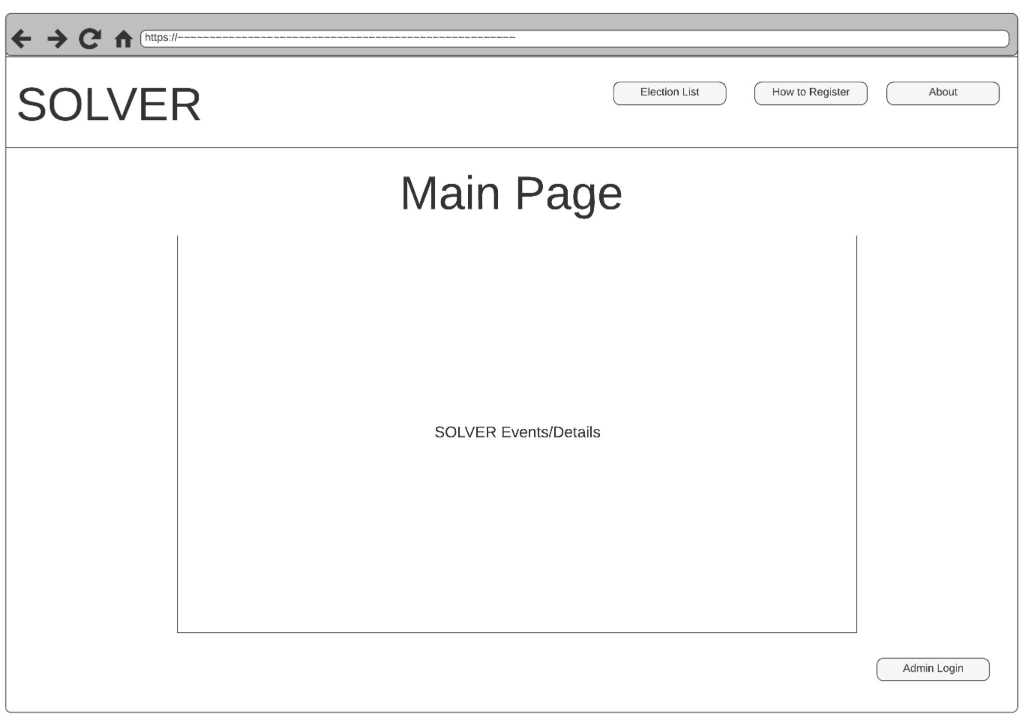
Admin Diagram:

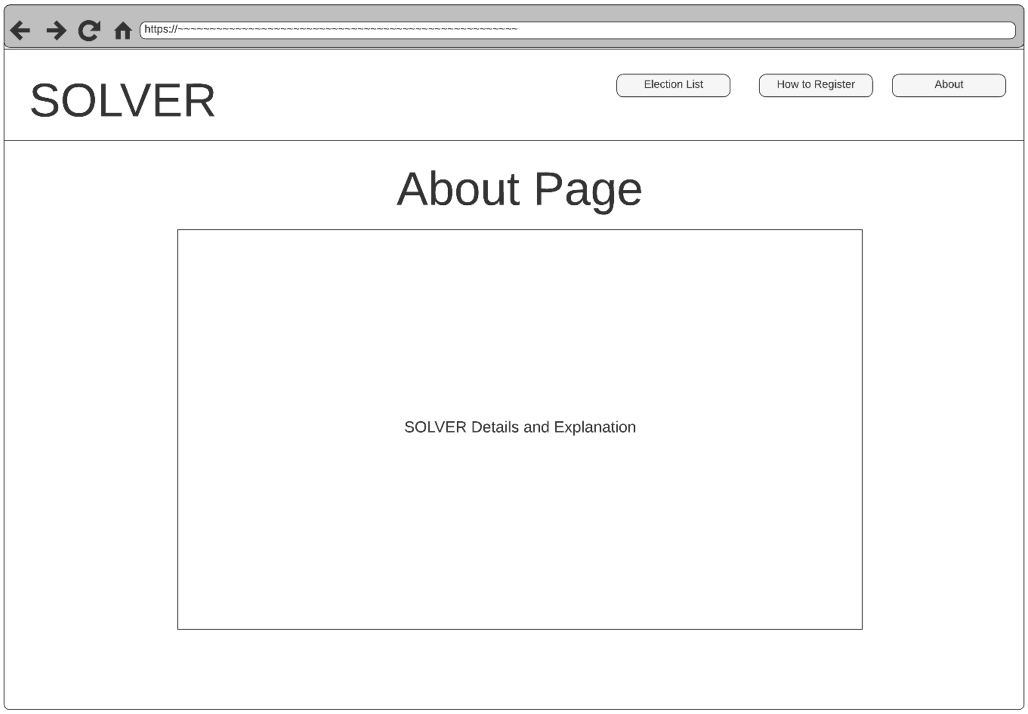


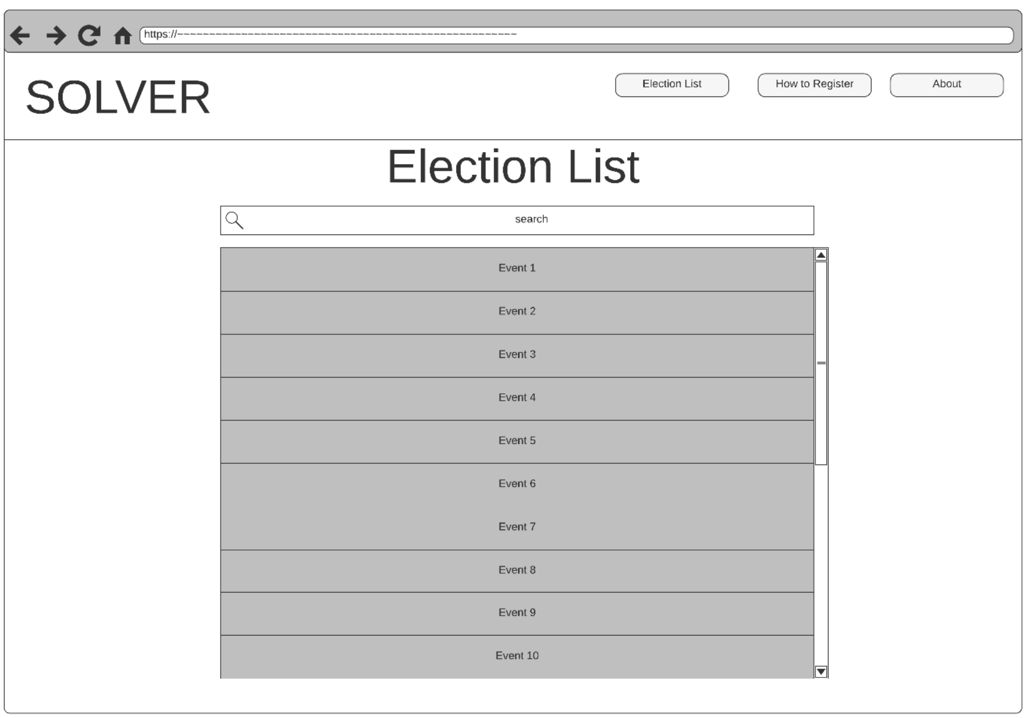
Election and User Diagrams:

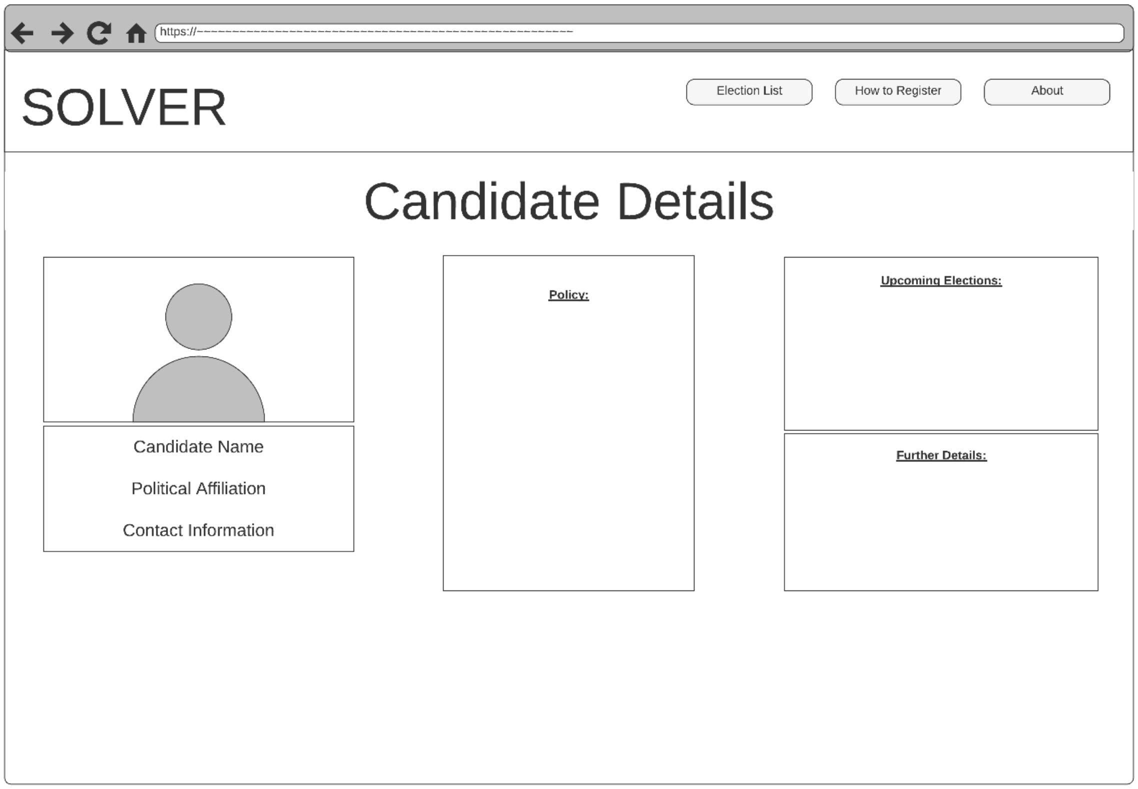


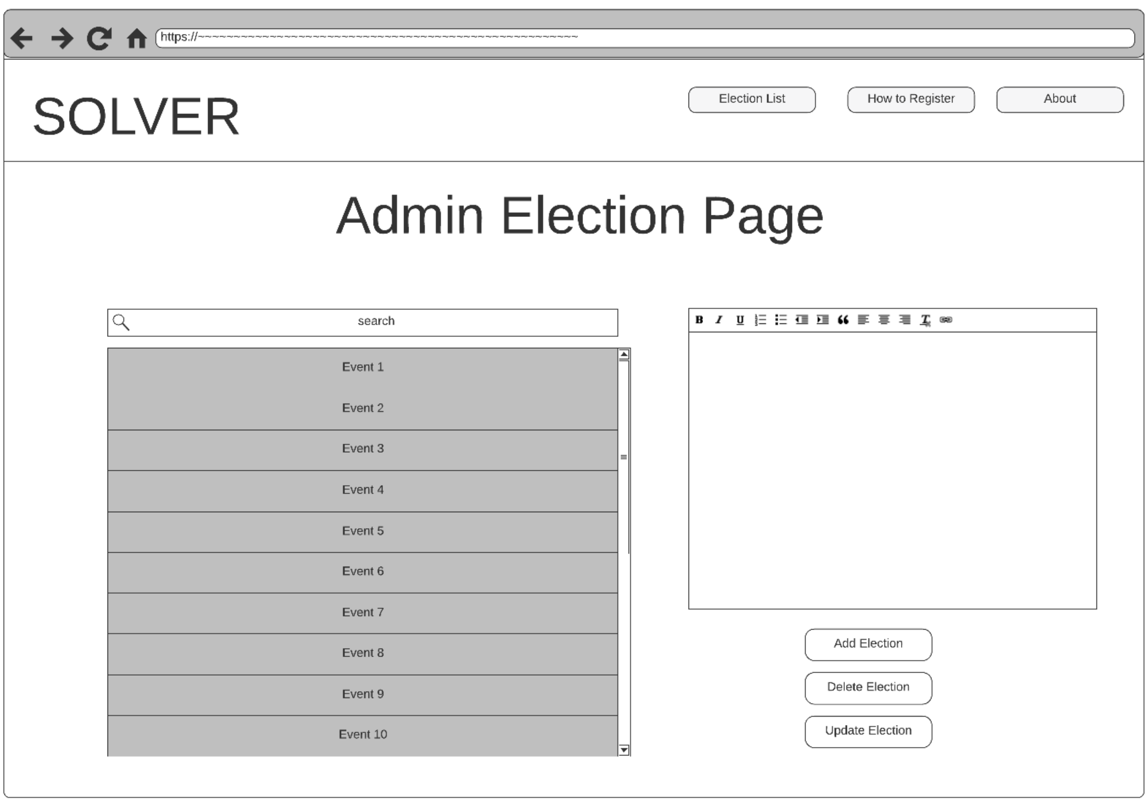
UI Mockups and Writeup:




https://~~~~~~
SOLVER
Election List
How to Register
About
About Page
SOLVER Details and Explanation


https://~~~~~~
SOLVER
Election List
How to Register
About
Election List
search
Event 1
Event 2
Event 3
Event 4
Event 5
Event 6
Event 7
Event 8
Event 9
Event 10


https://~~~~~~
SOLVER
Election List
How to Register
About
Candidate Details
Candidate Name
Political Affiliation
Contact Information
Policy:
Upcoming Elections:
Further Details:


https://~~~~~~
SOLVER
Election List
How to Register
About
Admin Election Page
search
Event 1
Event 2
Event 3
Event 4
Event 5
Event 6
Event 7
Event 8
Event 9
Event 10
Add Election
Delete Election
Update Election

In order to ensure that my UI design follows the key principles of affordance and visibility, and aligns with the "8 Golden Rules," I have created the writeup below. As my web-app is relatively simple overall, some rules apply differently to it.

1. Strive for Consistency
    a. All of my pages will maintain a consistent menu layout and overall theme and font style, so that the user knows how to easily access functions on any given page.
2. Enables Frequent Users to Use Shortcuts
    a. This does not particularly apply to my web-app, since the overall number of pages and functions is small. However, menu buttons at the top of every page will allow for quick traversal of all pages.
3. Offer Informative Feedback
    a. For every action the user takes, feedback will be obvious in most cases, since most actions involve bringing the user to a new page or subpage.
4. Design Dialogs to Yield Closure
    a. The sequence of information will have a consistent flow, with the user navigating to the website, searching for an election, and then selecting a candidate from a particular election as the final action. They may then repeat that same process for a different election or learn about registration on a separate page.
5. Offer Simple Error Handling
    a. The majority of error handling will occur in the search functionality and login functionality for admins. If an incorrect state or county is entered, the user will be told so directly by a popup. For admin logins, a popup will indicate if the admin entered the wrong login credentials.
6. Permit Easy Reversal of Actions
    a. There are not many actions that would need to be undone, but "back" buttons will be present on every page so the user can return to the previous page easily.
7. Support Internal Locus of Control
    a. The interface will be consistent, and will not change over time, to keep the users feeling in control of the interface. There will be a low amount of data entry to keep the experience from being tedious.
8. Reduce Short-Term Memory Load
    a. The total amount of action paths for this web app will be low, and the user will be able to easily find their way to and from pages. The only things they need to remember will be the county/state the election takes place in so that they may search for it.

How Requirements for Project Are Met:

As part of the design for the project, I will work to ensure that it is modular and uses encapsulation to hide information as appropriate. As shown in the Detailed Design Class Diagrams, I have separated my design into five distinct classes, with the majority of the information being private to its respective class. This class diagram may change over time, but I expect the overall layout to stay mostly the same. Each module has getters and setters to share and get information as necessary, but the data is overall still private to each class.

The central algorithm for my web-app is the search function for the election page. The main inputs for the search function will be counties and states, and these will be referenced against the location variable for each election object.

Election and Candidate objects will hold the majority of the information necessary for the program, and these objects and additional information will be stored in either a hashmap or 2D array in an SQlite3 database. Originally, the design was intended to use PostgreSQL, but I decided to migrate to SQlite3 due to numerous errors on both my virtual machine and local machine that I was unable to resolve.

Test Case Design:

Unit Testing: For unit testing, I will ensure that each of my pages and subcomponents work individually and produce the expected outputs. Separately, I will make sure that elections can be added, viewed, and selected, that candidates' information and pages are viewable, and that each webpage displays the intended information. I will follow through all the different sections of unit testing, including data flow, selective, and boundary testing.

Integration Testing: Next, I will combine each of my subcomponents on their required pages, and make sure that there are no conflicts with layout or data. I will test groups of pages and elections/candidates together to make sure they work. I will use top-down integration to slowly work through my program structure and add functionality as I go.

System Testing: The overall web-app will be tested as a whole, and I will ensure that the page transitions are working correctly, and that data is correctly passed between all subcomponents of the system. I will also ensure that there are no major bugs, and that the layout is consistent and easy to use.

Tools: To ensure proper version control and maintenance of SOLVER, I intend to use Git and GitHub. GitHub provides a task and milestone system to keep track of a projects progress and allows for simple viewing of project flow and completion. Git allows for easy version control and would let me return to a previous version if I encounter a very difficult bug.

For programming, I am using the IDE RubyMine, which provides a convenient development environment with syntax checking and integrating debugging. I am also using the RuboCop plugin, which offers more advanced debugging.

Test Cases:

Functionality Tested	**Inputs**	**Expected Output**	**Actual Output**
View List of Upcoming Elections	1. User clicking "Election List" menu button 2. County/State input for narrowing list	1. Navigating to Election List page 2. Narrowed list of elections	
Select & View Specific Election	1. User clicks specific election on election list	1. Election details page is displayed	
Select & View Candidate	1. User clicks specific candidate on election page	1. Candidate details page displayed	
Navigate to Registration Page	1. User clicks "How to Register" menu button	1. Registration page displayed	
Adding/Updating/Deleting Elections & Candidates	1. User clicks "Admin Login" 2. User enters credentials 3. User inputs details, clicks "Add Election/Candidate" 4. User click election on list, changes details, clicks "Update Election/Candidate" 5. User clicks election on list, clicks "Delete" 6. User clicks "Logout"	1. Admin login page displayed 2. User is brought to admin page if correct, popup saying "incorrect login" if not 3. Election/Candidate added to list 4. Election/Candidate updated 5. Election/Candidate deleted 6. User is logged out	

Open Source Maintenance and Communication:

The code and documentation for SOLVER will be maintained on GitHub. GitHub has the functionality to manage issues, milestones, and project pages. Communication will be maintained via the issues, which can be used to designate tasks and bring attention to problems. Additionally, text files detailing updates, changes, questions, security issues, and priority will be added to the docs subfolder to facilitate further documentation and communication. Branches of the project will be created for new contributions.

The primary stakeholder for SOLVER is CELR, since they will most likely be using it / maintain it after my work on the project ends. It will be up to a CELR staff member in charge of the project to decide what they are looking for, and what code should be accepted. Whoever is maintaining the project will decide which code is merged into master.

Steps for Member to Add Functionality:

1. Follow steps on GitHub detailing how to clone and run project
2. View issues and text files to decide on functionality to add
3. Create Git branch and develop new functionality
4. Pushed changes to branch
5. Contact CELR staff / admin to propose adding functionality
6. Admin adds new functionality to master branch
7. Issues are resolved in Git
8. Documentation for functionality added to docs subfolder