ElectConnect

Final Design Document
March 10, 2005

Mike Black
Alexander Boutelle
Catherine Hill
Lars Johansson
Alex Kossey
Dan Silverman
Pawel Wrotek

1.0 Description

The goal of this project is to design a system which allows the UCS Election Committee to design a ballot through the use of a straight-forward graphical user interface, similar to a standard Microsoft program. The software will then generate the necessary HTML, php and web scripts to administer the election online. The software will run a web server which students can connect to and submit their votes. At the conclusion of the election the system will end voting and will prepare a summary of the votes to be reviewed and analyzed by UCS.

2.0 Requirements

2.1 Critical

- Ballot Creation
  - provide an interface similar to standard Windows applications for designing the ballot on a question by question basis (PowerPoint as a model)
  - user can save and load ballots
  - support four question types
    - choose 1
    - choose n of N (where n <= N)
    - rank
    - open response
  - output ballot file which can be uploaded to server
  - support both elections and referendums
  - allow for each candidate to have a picture and bio
- Election Web
  - election website is clean and uncluttered with a well defined flow for the user to follow
  - site has an admin section where data can be downloaded in excel format
  - admin section of the site is protected by a single login and password
  - secure login system to ensure privacy and that each user votes only once
  - ballot questions can be restricted based on the user's class year
  - the system must be able to support between 2,500 and 5,500 votes cast and a minimum of 200 simultaneous users
  - appropriate security methods ensure that the integrity of the election data is protected at all times
- Other
  - develop a plan for long-term maintenance
  - provide user documentation including quick start and detailed walk-throughs

2.2 High

- Ballot Creation
  - allow modification of text style and colors of the web ballot
  - allow each candidate to have a custom webpage linked from the ballot
- Election Web
  - admin section of the site contains a summary of the current status of the election results
  - web interface compatible with all browser types

2.3 Medium

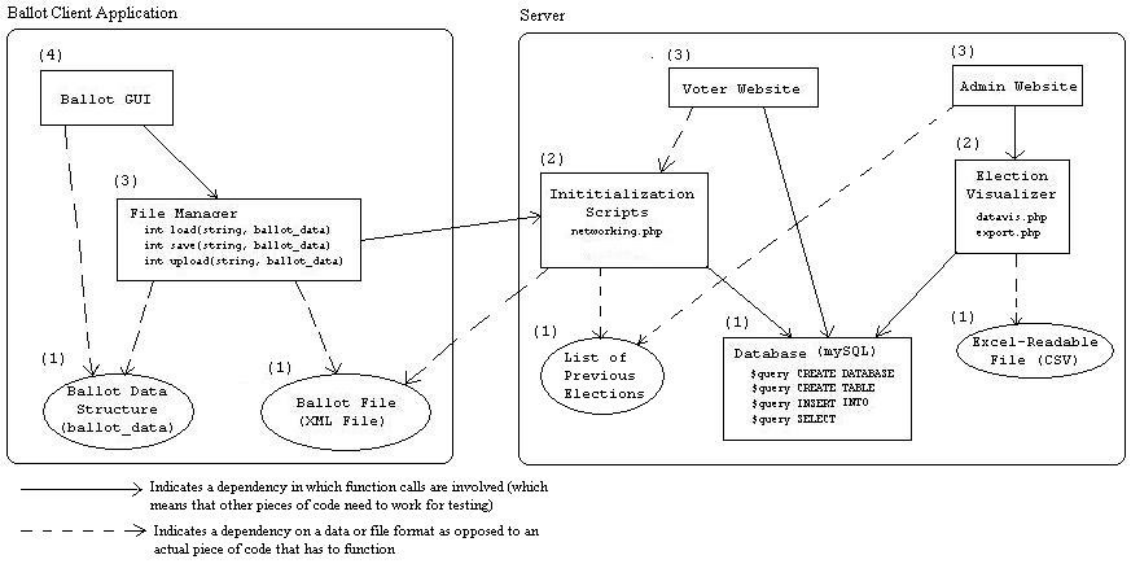
- Ballot Creation
  - C++ application connects directly to the server to upload ballot data file
- Election Web
  - fully graphical visualization of election data including colorful bar charts
  - admin section of the site allows modification of election start and end times
  - admin access can be allowed or denied on a user by user basis
  - current administrators can create and delete new admin accounts

2.4 Low

- Ballot Creation
  - support for instant run-off voting
  - provide an interface through which the group designing the election can fully the appearance of the web ballot itself including adding graphics
  - implement “wizards” to simplify the ballot creation process

3.0 Top Level Design

3.1 System Diagram



3.2 Component Description

3.2.1 Ballot Creation

The graphical user interface used to create a ballot. A user can choose to start a new ballot, load/save a ballot from/to a file, and upload the current ballot to a server. While working on a ballot, the user can add, remove, or edit questions. Each question must fall under one of four categories (as specified by UCS): yes/no, choose x of y options, rank the options, and open-ended (give the voter a text box to enter an answer). After choosing the question type, the user enters the actual question text, the list of available options/candidates, the groups that can vote on this question (freshman, sophomore, etc.), and possibly a link to a web page with more information about the question. The GUI stores the ballot information entered by the user in the ballot_data data structure, which serves as the main interface between the GUI and the file manager. The GUI calls three of the file manager's methods:

```
int load(std::string filename, ballot_data_t &data);
int save(std::string filename, ballot_data_t data);
int upload(std::string server, ballot_data_t data);
```

Each function returns 0 if it succeeds and 1 if it fails. In load, the data is passed by reference so that the GUI can read it after the file manager fills it in.

3.2.2 Ballot Data

Ballot Data is a critical component in this implementation because it provides an interface between the Ballot Creation component and the File Manager. It also drives, in large part, what is in the ballot data file.

The ballot_data data structure stores the following information:

- Election ID
- Election name
- Election start time
- Election end time
- Welcoming text
- Questions
  - Question type
  - Candidates
    - Candidate Name
    - Candidate Info
  - Question text
  - More question info
  - Voters qualified to view this question

3.2.3 File Manager

The component labeled file manager will do three things: it will create XML files out of the data inputted into the ballot creation, load previously saved XML files into the ballot creation component, and package the ballot XML file along with all other files the server will need (images and personal pages submitted by candidates) in a zip file that contains an appropriate directory structure, and upload this file to the voting server. File Manager's interface provides methods for loading and saving files to disk as well as uploading a file to the web server. These functions are called by the ballot design component and interfaces are given in the description of the ballot creation module (3.2.1).

The parsing portion of the file manager will depend on nothing except an external XML parsing library. The uploading functionality will have to send a ballot zip file to the server, which will require calling a script on the server (networking.php).

Independent of its exact format, the election data file will contain all the data held in the ballot_data data structure previously described.

3.2.4 Web Database

The database we are using resides on the CIS server and is a mySQL database. Each election will have it's own database, with a unique name and each database will have 2 tables.

Table 1 will be the table containing the vote data for the given election. A given row will have all of the question numbers followed by the answer provided by the user, with an appropriate value being used for no answer.

The second table will be a table consisting of all of the users who have already voted in the election. This will be used to make sure users who have already voted do not go through the process of voting again, and that users who have already submitted their vote but not logged out do not hit the "Back" button and attempt to submit their data again.

- Interfaces with:
  - Nothing: This is a bottom level component
- Things that access it (see their descriptions for what the interfaces are):
  - Web pages (will use $query "INSERT INTO" to add data to database and $query "SELECT" to get data on who's already voted)
  - Server-initialization (will use $query "CREATE DATABASE" and $query "CREATE TABLE")
  - Data visualizer (will use $query "SELECT" to get data)

3.2.5 Web Initialization

This component is the component containing the scripts which initialize the database, unzip the file sent from the ballot creation unit, and update the file containing the list of all elections.

When the ballot creation unit sends the .zip file to the server, a php script is used which archives the data from the last election file sent, unzips the new election file, updates the all-elections file, and initializes a new database for the current election file on the mySQL database running on the CIS server.

The mySQL database which is created will have a unique name relating to the time the election file was created and sent. This information will be recorded in the all-elections file, as well as the title and date of the election. This file is used by the data visualization unit to determine which database to access when displaying results.

What this interfaces with and how:

- Ballot Creation
  - Interfaces through the .zip file sent from the creation unit
- Database
  - Interfaces through the queries it sends to create the database and table
    - $query = "CREATE DATABASE <unique election db name>"
    - $query = "CREATE TABLE votedata ..."
- Web Page Scripts
  - Web page scripts use the XML file and the other source files (i.e. image files) that are uploaded by the ballot creation unit.

3.2.6 Web Election Administration

The web page scripts are the php scripts that are responsible for generating the web pages a user sees when they go to vote. These scripts are also responsible for storing the vote information in the database.

There are several pages that the user sees. These include:

- welcome page - here the user is asked to log in so that they can vote. If there is no election or the user has already voted, then they are directed to pages which inform them of these facts. If they have not voted in the currently running election, they are sent to the election pages.
- already voted page - this page is seen by the user when they try to log in after they have voted.
- no election page - this page is seen by the user when they try to access the website and there is no election running.
- election pages - these pages contain the questions that the user votes on. The information for the layout of the pages, as well as the current question, the type of question and the choices are determined by parsing the XML file sent to the initialization component. Once the user has voted on all questions, their information will be sent to the database. The user will be sent to the vote-success page.
- vote-success page - This page tells the user whether or not their information was successfully entered into the database. The page will also contain a logout option.

Primary scripts used:

- welcome script - this script generates the welcome page where users are asked to log in.
- validation script - this script makes sure that there is an election running and that the user has not already voted. It then directs them to the already voted page, no election page or actual election pages.
- election script - this script generates the election questions by parsing pieces of the election data XML file. It sends the results of the questions to the submit script.
- submit script - this script first enters all of the user's data into the database (making sure they haven't already entered data, if they have they are directed to the page indicating that they have already voted - this is for users who try and submit multiple times using the back button). If the data is successfully entered into the database, the script then generates a page telling the user their votes have been entered. If there is an error entering the data, the user is informed that there has been a problem and is asked to try to submit their data again.

An important note on security: We are using WebAuth, the CIS authentication system, and we will use their systems to make sure the user is logged in from the same computer throughout their voting session to ensure that the site is as secure as possible.

We will also be using Grouper, another piece of CIS software that allows us to make sure users are in the correct group in order to see certain questions. For example, only sophomores would see questions pertaining to their class's representatives.

Interfaces With:

- Database
  - $query = "INSERT INTO votedata ..."
    - Adds vote data to the database
  - $query = "INSERT INTO voted ..."
    - Adds a user who has finished voting to the database
  - $query = "SELECT user FROM voted WHERE login=<username>"
    - Checks to see if a specific user has voted or is logged in.
- Server-Initialization component
  - Uses the XML file and the other files in the .zip file sent over by the ballot creation unit

3.2.7 Web Data Visualization

Admin Logon

The admin logon page is simply the .html document where the administrator starts. Here is where they have to initially logon. If logging on is successful, then they will be redirected to the Admin Main page.

We will use our own security system for the admin logon instead of using the CIS AuthID/NetID system. This will make things much easier from both a technical and user standpoint. The UCS users can now simply have one logon and password that they can use instead of having to worry about deleting and adding a set of allowed users to the system.

Admin Main

Here the administrator will be able to perform 3 basic tasks:

1. View ongoing election statistics
2. Select an old election and view those statistics
3. Change the admin password

The only dependency this component has is on the Election Visualizer.

Election Visualizer

This component retries election information from the database and displays it for the user. If the election is currently ongoing, then additional information will be displayed, such as number of users logged on, etc. At the bottom of each Election Visualization page, there will be an option to download the election results as an Excel spreadsheet.

This will all be done with a combination of mySQL and PHP.

Election Saver

The Election Visualizer will call on the Election Saver to export the election results to an Excel spreadsheet. The Election Saver will be in charge of taking the information from the database and formatting it into Excel nicely.

This will again be done with mySQL, PHP, and any other scripting languages necessary.

4.0 Schedule

4.1 Milestones

3/11

- Final Top Level Design Due

3/16

- Interface Proposals
- GUI Designs Complete
- ballot_data structure complete
- File Manager interface set
- Ballot data file format set

3/18

- php and mySQL testing and experimentation complete
- security conversation with CIS

3/25

- Final Interfaces
- GUI shell set up (empty window opens)
- Login system functional

4/8

- Detailed Designs
- GUI visually complete (not full functionality)
- Parser interface functions complete
- Visualizer can display basic election results
- Exporting to excel complete
- File Manager save and load functions complete

- Server initialization unpacking script complete

4/11

- Generating web pages from ballot data file functional

4/16

- Initial Integration
- Core Ballot Creation functionality complete (creating a new ballot, adding new questions)
- File Manager functionality complete
- Visualizer can display graphical data
- Web Initialization functionality complete
- Votes can be submitted to database

4/25

- Secondary ballot creation functionality complete (moving questions, editing questions)
- Begin comprehensive full system testing
- Begin screenshots and user documentation

5/1

- Full System Implementation

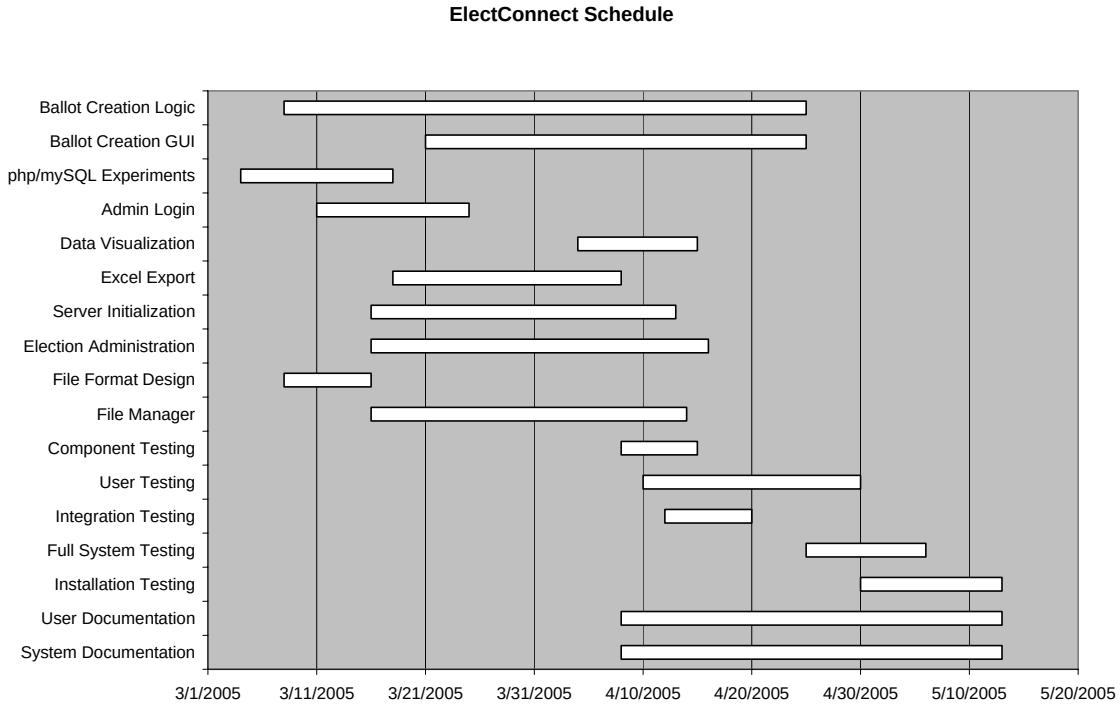
5/6

- Public Demo
- Core user documentation complete

5/13

- Completed System

4.2 Progress Chart



5.0 Testing Plan

Our testing strategy is necessarily advanced and robust. Our project will be held to a higher standard of durability than BikeQuest. Our user base is much better defined and demanding – more important, we have a product that will be used intensely for sporadic periods, with zero margin of acceptable error. The main sections of the testing plan will fall into the following categories: component comment and testing; integration testing, user testing, and installation testing (including stress testing). Each of these will be aligned to certain dates and releases. Let's review them in order.

5.1 Component Testing

4/8/05 – 4/15/05

Each major component (Ballot Creation, Visualization, Election Administration, and Server Database) will need to have a separate testing plan. These must be tested and passed before the beginning of integration.

5.1.1 Ballot Creation

Ballot Creation will be somewhat difficult, because of its integration with the GUI. I would recommend keeping the two separate initially. The GUI is deceptively simple,

and will need to be pristine before we connect it up to other semi-tested code. The GUI sector will output the function calls (from button clicks and field submits) to a testing log. This log will eventually resemble the script used to test the Logic, which will essentially be a set of fake function calls to the logical backend, simulating a User operating the GUI. The output of this will be a completed Created Ballot Data File, according to the specs laid down in this design. Then the Ballot Creation sector will be ready for integration. UCS members will have been consulted repeatedly on GUI design and layout, but they should again be brought in to see the functionality and approve its functionality freeze and general HCI.

5.1.2 Web Election Administration

This testing plan will be simple from Day One. With the Created Ballot Data File nailed down, we can hard-wire a number of potential ballots. The first will be very basic, and will probably not resemble the ballot at all – rather, it will have very basic HTML commands. Over iterations, all of which are independent to the Generation Scripts coder, the Generator will create more and more complex pages from fake CBD Files. Quality Assurance will work with Generation Scriptor to design appropriate files that test all ElectConnect functionality. Once we can generate all necessary CBD File instances, we are ready to integrate.

5.1.3 Database

The major methods called on this item will be the input and output of information. Input will be faked with bots scripts that originally affect the DB itself. After some integration the online ballots will be operated by bots that simulate actual users (see User Testing below). If input scripts create a database with the correct information, we are ready to move on. Output methods will be called by the visualizer, and either returned as summary information to the DataViz or outputted raw to an Excel file. A test database with fake files can be queried for either of these functions, and once the correct results are tracked we are ready to move onto the next stage. The database should also be tested for Candidates with the same name or other information that could throw it off.

5.1.4 Visualization

This testing will rely eventually on live information in the Server Database to display its results. Until then, we can create a test database that will be used to simulate an actual election. The Visualization will be ready when a variety of text databases (see above) will correctly display their contents. Of course, elaborate DataViz is our lowest priority, as long as an Excel File can be created.

5.1.5 All Components

All components (all thus all coders) need to adhere to a comprehensive commenting convention. This is largely due to the halflife of our program; we want it to survive in active use after we've left Brown, and maintenance coders will need to understand our

setup. Moreover, we will have several coders (as well as other team members) working with this code: no individually-understood code is acceptable. To this end, we should have detailed function descriptions in the comment section of each method, line-by-line of particularly complex algorithms, and NO non-intuitive variable/methods names. **Also,** we need to have an across-the-board decision to allow zero memory leaks or compiler warnings in the code before we go to integration stage – these will propagate down to much larger and more complex problems down the line. Our code should be bulletproof, then be integrated.

5.2 Integration Testing

4/12/05 – 4/20/05

Once the individual components have been tested out and inputs/outputs adhere to the defined interfaces, then integration and the associated testing should become MUCH simpler (and loading the difficulty onto the beginning of the process is a smart idea). Connecting the parsing scripts to the live-created CBD File will be a simple process, and should be no technically different than the test cases we ran. The web voting page will need to update the database correctly, but we will have tested it from the database side previously. Then the DataViz file should output correctly to either Excel or its visualization, but the only difference is that the Database will be the actual DB, instead of a fake tester.

5.3 User Testing

4/10/05 – 4/30/05

We have called for the system to be able to handle 3000 records all together and several hundred at once. I'm not aware why we can't up those numbers: UCS' priority is to not crash their computers, as they famously did before. Therefore, we should be building for thousands of people accessing the ballots simultaneously. This will involve the use of scripted bots simulating users, and logging on in massive groups. The functionality of the voting face is much simpler than that of ballot creation. Essentially, all the user can do is make some selections and submit the form. This type of action should be very simple to implement and randomize with bots. The ballot creation integration testing ban be handled by group members, since this functionality will be used by roughly the same number of end users. We should again bring in UCS members to test and give us feedback here – we are good technical testers, but like Coke says, you can't beat the real thing.

5.4 Installation Testing

4/30/05 – 5/13/05

After the program has been installed on CIS and UCS computers, we will need to stress test the network connections to the server, then do some actual user testing. This could

be accomplished by setting up a special online test that required no login, and enlisting the class' help (or perhaps make it a CS2 assignment?) to submit, log, and track a bunch of test votes.

6.0 User Interfaces

6.1 Ballot Creation

The following is an example screen from the Ballot Creation GUI.

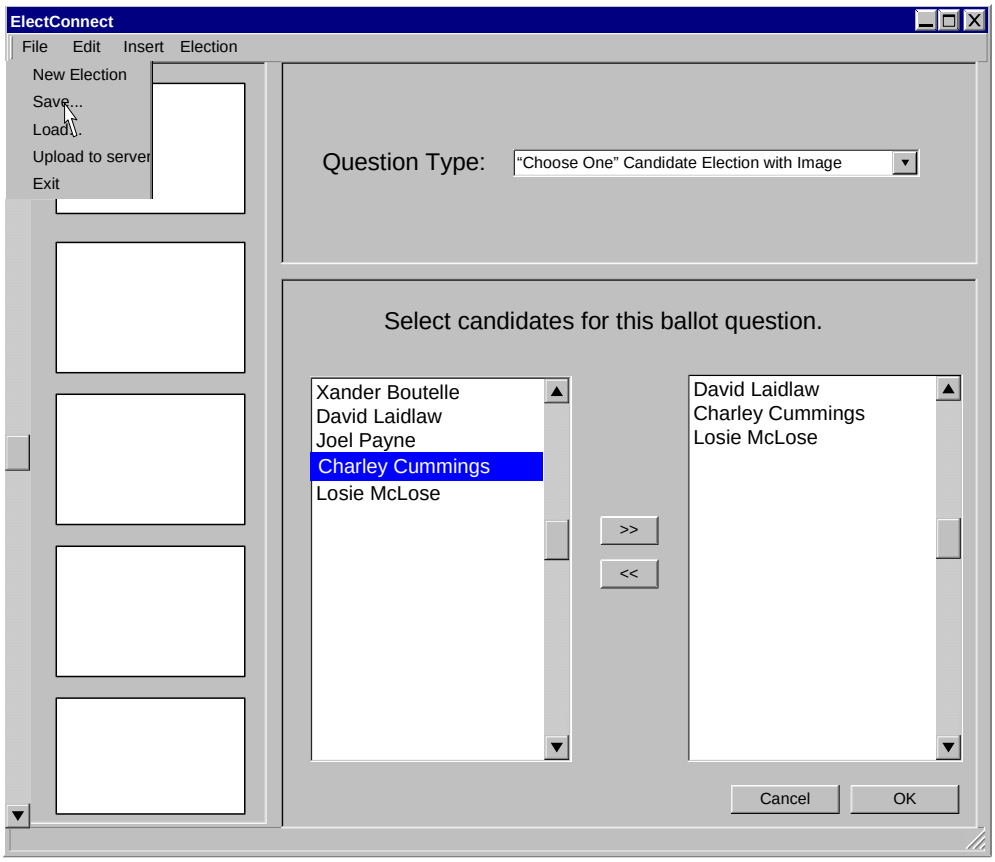


Figure 1: Ballot Creation Interface

The File menu items are displayed in Figure 1: Ballot Creation Interface, additional menu items are the following:

Edit

- Question Type
- Response
- Allowed Users
- Remove Question

Insert

- Question
- Candidate

Election
- Set Start Time
- Set End Time

6.2 Election Website

The following are screenshots of the election website. These are subject to minor variation as we iterate through designs and elicit feedback from UCS. However, the level of complexity and general design is indicative of what we expect to see in the final version.

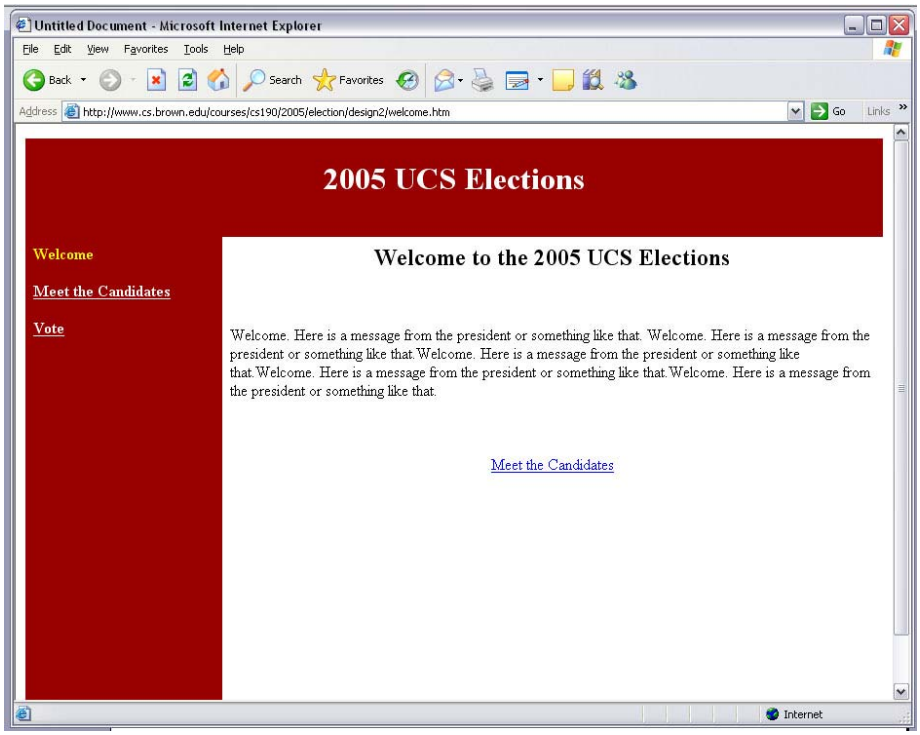


Figure 2: Welcome Page

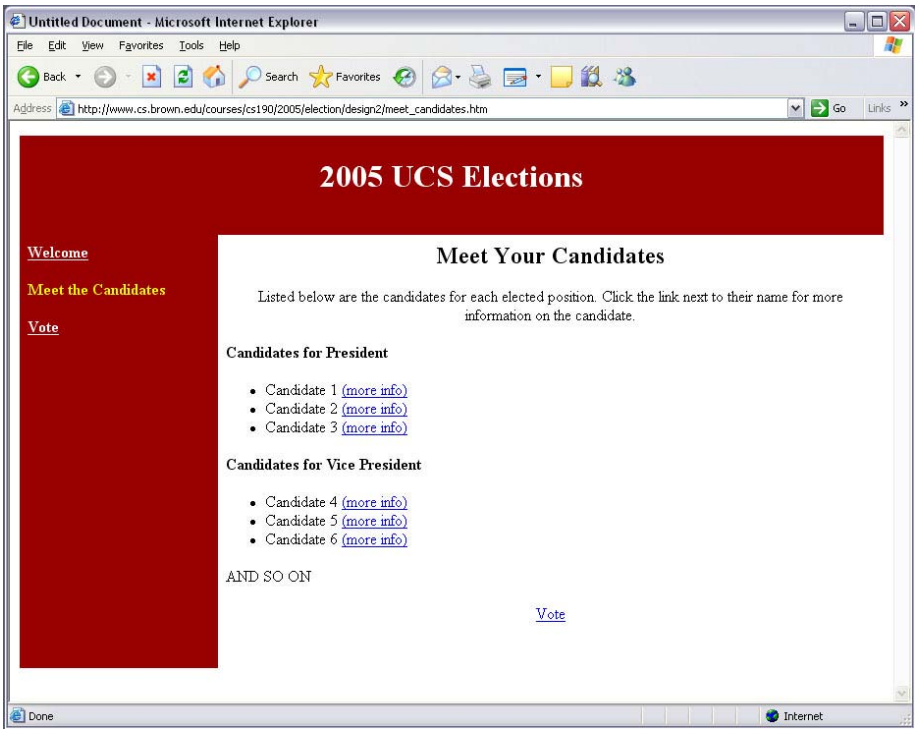


Figure 3: Meet the Candidates Page

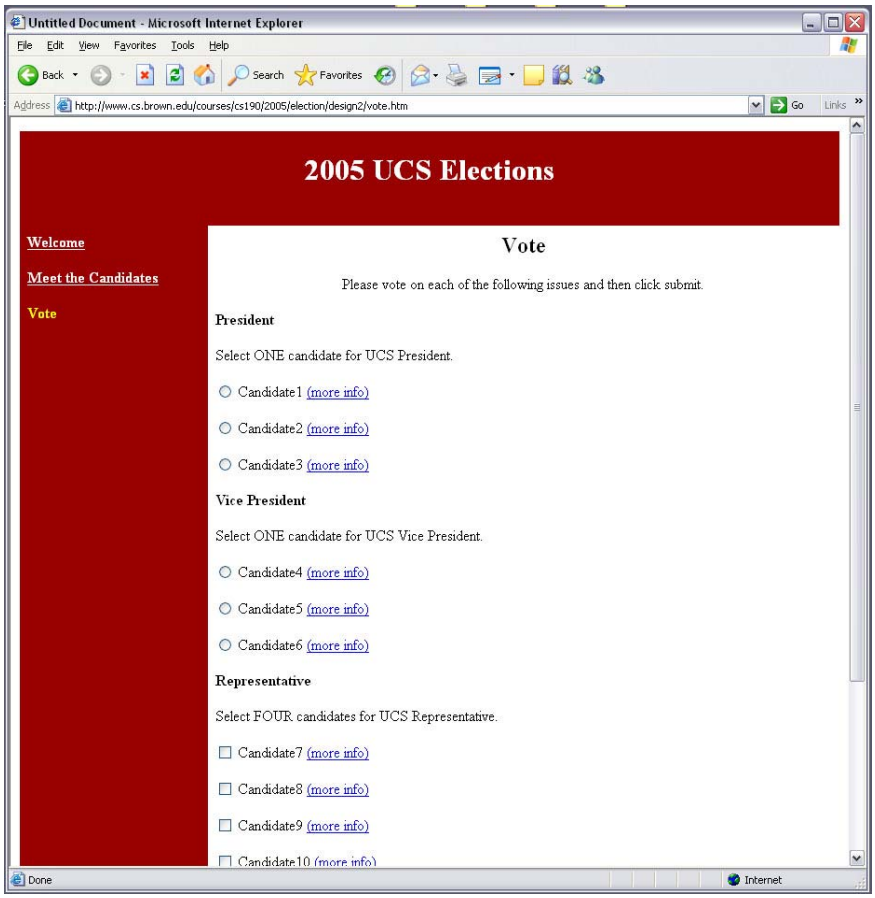


Figure 4: Ballot Page

7.0 External Dependencies and Risks

7.1 Computing and Information Services

There are two primary ways in which this project relies on CIS. First, ElectConnect will web-auth, a proprietary CIS tool, to authenticate users. Second, in order to fully satisfy UCS's concerns over the security and stability of the website, it will be hosted on a CIS server.

7.1.1 Web-Auth

We will be able to use the CIS web-auth server to require users to log in to the election website using their authID and password. Web-Auth also works in conjunction with a tool called Grouper. Grouper will allow us to screen users based on whether they are a member of a particular "group". The groups we will be concerned with are; freshman, sophomore, junior and senior as well as student and faculty. CIS is providing a software tool which will assist in the creation of Grouper files as well as documentation which gives an overview of the interfaces for both Grouper and Web-Auth.

7.1.2 Server

CIS is giving us access to their server, LAMP. LAMP runs Linux, Apache, mySQL and php. It is capable of handling (at minimum) the user load outlined in the requirements section of this document and the packages available on the server, namely php and mySQL are sufficient to be able to implement all the functionality listed there. Alex, Mike and Catherine will be granted accounts to upload files to the server. CIS will also provide a URL for the site, their suggestion being ucselection.brown.edu .

7.1.3 Contact

Our primary contact at CIS is Bill Dennen (William_Dennen@brown.edu).

7.2 Long-Term Maintenance Plan

Both CIS and UCS are very interested in the plan for long term maintenance of the system. UCS wants assurance that there will be an individual who understands the technical aspects of the system and will be available to perform routine maintenance, upgrades as they are needed and real time technical support the night of the elections. CIS, while they are enthusiastic about granting us server space and giving initial direction with respect to Web-Auth and Grouper, does not want to be left with long-term responsibility for the system. Thus we will implement the following plan for ElectConnect's long-term administration.

(1) Fully commented source code.

(2) Robust technical system documentation
(3) Technical Chair position on UCS Election Board

It will be the Librarian's responsibility to ensure that the programmers are adequately commenting their code as they progress. He will read programmer's source code and comments to verify that they are carefully describing each function. Additionally, if the Librarian is familiar with a particular section of code he will request that the Program Manager read it to evaluate the usefulness of the comments.

The Librarian will also be in charge of creating technical documents. These documents will include a high level description of the elements of the program and their interaction, system diagrams, and detailed descriptions of the individual parts of the program.

We will also recommend that UCS create a Technical Chair on their Elections Board. The Elections Board is formed annually from a pool of applicants who need not currently be members of UCS. It will be the responsibility of the Technical Chair to learn and understand ElectConnect's system architecture. The Technical Chair should be proficient in C++, php and mySQL, but need not be a Computer Science concentrator. This year and next year the Technical Chair should be selected from the current ElectConnect team.

In the near term this plan will be formalized and delivered to both UCS and CIS for their consideration.

7.3 UCS

It is important also for our team to acknowledge the challenges we will face dealing with UCS. In particular, we must be conscious of the timing issues we may encounter. We will not always be able to elicit immediate feedback from UCS on the progress of our project. Thus, we have planned in to our schedule several UCS checkpoints. These are milestones at which we will provide UCS with an update on the status of the development progress as well as collect feedback from them. Building these in to our schedule now will provide motivation for both our group and theirs to think about the status of the project and as well as a framework for communication between the groups. Also, we should be prepared to issue at least two documents to UCS over the course of the semester. The first is the proposal for long-term maintenance already described. The second is a security plan. It will detail, in a non-technical way, the components of our system security. The goal of this document will be to erase any fears UCS has about security flaws inherent to our system.

8.0 Group Dynamics

8.1 Roles

8.1.1 Project Manager

Lars Johansson

The Project Manager is responsible for overseeing the development process. Specific duties include; maintaining an up-to-date schedule of the overall progress of the project as well as its individual pieces, facilitating communication both within the group and with other organizations, structuring and planning meetings with input from the group and to support the other group members as needed.

8.1.2 Quality Assurance Lead

Xander Boutelle

The Quality Assurance Lead is responsible for developing a testing plan and schedule. He is also in charge of writing test suites and supervising other project members involved in testing. He should establish a method of logging and tracking bugs and the steps that are being taken to correct them.

8.1.3 Architect/Librarian

Alex Kossey

The Architect and Librarian roles have been consolidated in our group because, first, we believe we can achieve some synergies from having the same person lead these two aspects of the project and, second, while both jobs will be ongoing for the duration of the project, the Architect role is probably more front loaded and Librarian tasks will be heaviest toward the end.

The Architect is responsible for making major, system level design decisions with input from the rest of the group. He has final discretion with regard to high level design. He is also a resource for other group members as they design their individual components. The Architect is responsible for the conceptual integrity of the design.

The Librarian will oversee the creation of both the technical documentation and the user documentation. For our project both of these classes of documentation are critical. The Librarian may delegate some of the authorship to the Project Manager or to coders responsible for particular modules. The Librarian will also design a plan for keeping documents up-to-date as the project progresses. We do not want to be writing all our documentation in the final three days.

8.1.4 Ballot Creation Coder

Pawel Wrotek

The Ballot Creation Coder will implement the C++ Ballot Creation portion of the project. He will be responsible for the graphical user interface as well as the logical backend. He will be in charge of component testing for this individual component with oversight from the QA Lead.

8.1.5 File Management Coder

Dan Silverman

The File Management Coder is responsible for designing the file format that acts as the interface between the Ballot Creation module and the server. He also will implement the File Manager portion of the C++ Ballot Creation software. The File Manager as described before will handle parsing ballot files as well as uploading them to the election server. The File Management Coder will component test his portion of the project and work with the Ballot Creation Coder on phase one integration testing for these two parts.

8.1.6 Web Lead

Catherine Hill

Because there are three coders working on code which will run on the server, we have designated one group member Web Lead. The Web Lead's main responsibilities are to be the point person for contact with CIS (especially with regard to technical issues) and to coordinate work and direct interaction among the three web coders.

8.1.7 Web Initialization Coder

Alex Kossey

Web Initialization Coder is in charge of implementing the web scripts which will initialize the database as well as parsing the .xml file containing the ballot data for an election. He will component to the extent possible, however much of the server testing will be done with the help of the QA Lead.

8.1.8 Web Election Administration Coder

Catherine Hill

Web Election Administration Coder will design and implement the portion of the server responsible for administering the election as it is taking place. She will component to the extent possible, however much of the server testing will be done with the help of the QA Lead.

8.1.9 Web Data Visualization Coder

Mike Black

The Web Data Visualization Coder is responsible for designing and implementing the "admin" portion of the server, including admin logon and data visualization. He will component to the extent possible, however much of the server testing will be done with the help of the QA Lead.

8.2 Cultural Mechanisms

Wow Emails

Group members will be sending out "wow" emails as they make breakthroughs or even if they discover something cool or interesting. This will keep the group informed about the progress of the other members, help create the feeling of consistent progress and potentially motivate people who aren't having frequent "wow" moments to work a little harder.

Weekend "No CS Allowed" Bonding

These will be semi-regular opportunities for the group to get together for pizza, snacks or some other entertaining diversion. They will not be required, though group members are encouraged to attend. The one binding rule will be that productive CS related conversations will be banned from these situations. The goal is for the group to realize that we are capable of interacting in ways that do not involve ElectConnect. Hopefully this will result in more comfortable, open interactions when we are actually discussing the project.