AN ELECTRONIC VOTING SYSTEM
Detailed Analysis

System Overview

A system to enable web-based election. When a voter visits our site, a login form is displayed. It prompts for the voter's SIN and PIN. If the login is accepted (see below) a voting form is displayed. It shows the names of all candidates and the party of each. When the user votes for a candidate the process ends. The design of the system, explored herein, adopts the Model II MVC design pattern.

Data Sources

We assume that two files are available: Voters, which contains the SIN/PIN pairs of all voters, and Candidates, which contains the name/party pairs of all candidates. Assume that these are CSV, Tab-Del, or XML files (up to you). These files should stored in WEB-INF so they are invisible to Coyote. Make sure you always use relative paths in your servlets so that your webapp is portable across servers.

The Model

The model is represented by the following components:

- The `Candidate` bean, which has three attributes: the candidate's name, party, and vote count. This bean is a member of the `election` package.
- The `Voter` bean, which has three attributes: the voter's SIN, PIN, and boolean voting status (voted or not). This bean is a member of the `election` package.
- A POJO named `VotingModel` which is also a member of the `election` package. It has two attributes: `Map<String,Candidate>`, which store the candidate names as keys and their beans as values, and `Map<String,Voter>`, which store the voter names as keys and beans as values. The constructor of this POJO must initialize these maps using the text files and appropriate default values. This class has two more `public static final` attributes that indicate the number of digits in SIN and PIN. These can be hard-coded or read from the Voters file.

The Controller

One servlet controls the entire webapp. All output from this servlet is handled by a dispatching the request to a JSPX file; i.e. it never writes anything to its response parameter. Here are the controls:

- When a voter first visits the servlet, a login form should be displayed.
- Upon submitting the login form, the controller consults the model in order to validate the format and values of the received SIN / PIN pair.
- If login failed, the form is re-sent with an error message.
- If login continue to fail after three attempts, a terminal message is sent instructing the user to contact the election office.
- If login succeeds (a critical event) the servlet then consults the model again to see if this voter has already voted. This is done via the model's `hasVoted(String sin)` boolean method.
- If the voter has already voted, a terminal message is sent. Otherwise, the voting form is served.
- When the voting form is submitted, the model is informed of its content via a method in it named `elect(String,String)`, where the first parameter is the SIN of the voter and the second is the name of a candidate. If the voting ballot is spoiled, the second parameter is null. Note that the model will consider the voter to have voted once this method is invoked, even if the ballot is spoiled.
- Once voting is completed, a terminal thank-you message is served.
- Invoking the servlet with an `admin GET` parameter makes it return the Result of the vote.

The View

It is clear from the above that we need the following JSPX files:

- Login.jspx (which takes an optional error message).
- Vote.jspx
- Message.jspx (for terminal messages)
- Result.jspx

The Result file must show, in a tabular fashion, the candidates, their parties, and the votes that each obtained as a raw number and as a percentage of the total. In addition, the report must indicate the turn-out and the time the report is generated. Finally, the row with the highest number of votes must be shown in red. If there is a tie, then the rows of all the winning candidates must be shown in green.