A VOTING BALLOT WEB APPLICATION AS A COLLABORATION SUPPORT SYSTEM

A Thesis
Presented to the
Faculty of
San Diego State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Varun Jayakumar

Fall 2013
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Varun Jayakumar:

A Voting Ballot Web Application as a Collaboration Support System

Carl Eckberg, Chair
Department of Computer Science

William Root
Department of Computer Science

Robert Briggs
Department of Management Information Systems

Nov. 20, 2013
Approval Date
Copyright © 2013
by
Varun Jayakumar
All Rights Reserved
DEDICATION

I would like to dedicate this work of Thesis to my family, and friends for their support through my rough and easy times.
ABSTRACT OF THE THESIS

A Voting Ballot Web Application as a Collaboration Support System

by

Varun Jayakumar
Master of Science in Computer Science
San Diego State University, 2013

The goal of this Thesis project is to implement the client side of a Voting Ballot Web Application taking into context Collaboration software methodologies. People from remote locations having a unanimous agenda will be able to bring in their feedback and opinions in the form of a vote to help better understand the requirement of a task, and also provide means through which a solution can be achieved. The client side interfaces with the middleware in order to communicate with the database on server to store information and retrieve information so the information is displayed at the front end. Client side scripting is done in order to minimize load on the server side and help response times be quicker. An interactive and a user friendly User Interface is provided to make software use unburdensome.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>x</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>xi</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 DESIGN AND METHODOLOGIES</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Overview</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Requirements</td>
<td>3</td>
</tr>
<tr>
<td>2.3 Technology</td>
<td>3</td>
</tr>
<tr>
<td>2.4 Development Environment</td>
<td>4</td>
</tr>
<tr>
<td>3 IMPLEMENTATION</td>
<td>5</td>
</tr>
<tr>
<td>3.1 Templates</td>
<td>5</td>
</tr>
<tr>
<td>3.1.1 Login</td>
<td>5</td>
</tr>
<tr>
<td>3.1.2 Create Ballot</td>
<td>6</td>
</tr>
<tr>
<td>3.1.3 Cast Vote</td>
<td>7</td>
</tr>
<tr>
<td>3.1.4 Results Display</td>
<td>7</td>
</tr>
<tr>
<td>3.1.5 Registration</td>
<td>16</td>
</tr>
<tr>
<td>3.2 Interfacing</td>
<td>16</td>
</tr>
<tr>
<td>3.2.1 Getter API</td>
<td>19</td>
</tr>
<tr>
<td>3.2.2 Setter API</td>
<td>20</td>
</tr>
<tr>
<td>3.3 Core Features</td>
<td>23</td>
</tr>
<tr>
<td>3.3.1 Chart View</td>
<td>23</td>
</tr>
<tr>
<td>3.3.2 Ballot Creation with Ballot Items Entry</td>
<td>24</td>
</tr>
<tr>
<td>3.3.3 Ballot Item List and Voting</td>
<td>25</td>
</tr>
<tr>
<td>3.3.4 Credential Page</td>
<td>27</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 2.1. Technology IDE’s.................................................................4
LIST OF FIGURES

PAGE

Figure 2.1. Technology stack....................................................................................................3
Figure 3.1. Login template........................................................................................................6
Figure 3.2. Create Ballot template ............................................................................................8
Figure 3.3. Cast Vote template ...............................................................................................11
Figure 3.4. Result template with static values .......................................................................14
Figure 3.5. Result template with dynamic push .......................................................................17
Figure 3.6. Interface for retrieving data from server ..............................................................20
Figure 3.7. Interface for sending data over to server ...............................................................21
Figure 3.8. Result list ..............................................................................................................24
Figure 3.9. Result graph ..........................................................................................................25
Figure 3.10. Ballot entry .........................................................................................................26
Figure 3.11. Ballot list not voted on .......................................................................................26
Figure 3.12. Rank order polling on ballot items .....................................................................27
Figure 3.13. Sign-in page ........................................................................................................28
Figure 3.14. Registration page ...............................................................................................29
**LIST OF ABBREVIATIONS**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIS</td>
<td>Management Information Systems</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheet</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ID</td>
<td>Identity</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>ESClient</td>
<td>ElasticSearch Client</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>3-D</td>
<td>Three Dimensional</td>
</tr>
<tr>
<td>SDSU</td>
<td>San Diego State University</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript and Extensible Markup Language</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I convey my gratitude to Dr. Robert Briggs for his guidance through this project. I thank Dr. Carl Eckberg for being the Chair of the Thesis committee and for his advice throughout my time on this Thesis and also would like to convey my regards to Dr. William Root for serving in the Thesis committee.
CHAPTER 1

INTRODUCTION

Multiple solutions can be provided to a single problem as most of us have come to know through our experiences. A solution could be good enough for the problem to not relapse. That alone is not sufficient for an organization facing an issue. In a competitive world such as the world we live in, efficient, effective, and feasible solutions are sought after and given paramount importance by organizations. Such solutions can be made possible when multiple minds are involved in deriving one. Ideas are nurtured by different perspectives. Different perspectives are attained by Collaboration. Collaboration has been at the forefront of paving ways into innovation and enhancing efficiency. It is therefore a Collaboration Software that is the topic of discussion here.

A faculty member from the MIS department at SDSU has his research interests in the field of collaboration. Often, it has been a case where multiple opinions from remote locations have mattered in order for him to arrive at a solution for problems that needed his expertise. Before a solution is arrived at, priorities and concerns of the Requirement Owner needs to be taken into consideration. In order to attain such dynamic data, dynamic collaboration between the Solution Provider and the Requirement Owner is imperative. The need was to have a Voting Ballot Web Application as a Collaboration Support System, which is the topic of my Thesis.

There are similar paid and free versions of voting tools available in the online market currently. A couple of examples are:

- SourceForge EasyVote, where online voting for a relatively small number of voters is made available [1]. It is mostly written in PHP [1].
- Kwik Surveys, which is a free and a paid tool where mixing and matching of questions is made possible as part of a survey and certain restriction features are available too within the tool [2].

A Voting tool that is dedicated to the Research interests of the Management Information Systems faculty is presented here to fulfill our Thesis requirement. The Web Application here is also a rank order voting tool; meaning, the votes are casted by users in the order of
their priority. This type of voting is not too common except for in the elections in select countries.

This Thesis Report is categorized into five chapters:

- Chapter 1: Introduces to the topic of Collaboration and Voting Ballot systems
- Chapter 2: Covers overview, requirements of the software and the technology used
- Chapter 3: Mentions some of the means through which implementation was done
- Chapter 4: A summary of the conclusion is provided along with some of the major hindrances faced
- Chapter 5: Work that can be done in the future is touched upon
CHAPTER 2
DESIGN AND METHODOLOGIES

2.1 OVERVIEW
The Voting Ballot Web Application enables users to create Ballots and use Ballot items to vote with the help of a GUI. Communication with the database on server is established to transmit data to the database and from the database. Features to notify users, view results in multiple ways, commenting on a particular Ballot and several more features are part of the client side. Primary focus of this document is on the client side which includes the front end and part of the middleware of the application.

2.2 REQUIREMENTS
The web application requires the following to run:
- 3 cluster system for ElasticSearch
- 1 system for running Python API server
- Hostname

2.3 TECHNOLOGY
Figure 2.1 shows the technology stack of this web application.

![Technology Stack Diagram](image)

Figure 2.1. Technology stack.
• HTML
  o A Markup language that is used for structuring content, and displaying web pages on the web browser
  o Helps us communicate with the others on the World Wide Web
• CSS
  o Used to define the visual styling of the web pages
  o Content is separated from design to enable versatility
  o Regardless of the Markup language used, appearance semantics of a page is described
• JavaScript
  o To add interactivity to web pages
  o Runs on client side, thereby making processing time faster
  o Embedded to HTML code
• Python is used as an interface between the client side and server side.

2.4 DEVELOPMENT ENVIRONMENT

There have been multiple technologies used in the development. The tool used for coding using each of these technologies is mentioned in Table 2.1

<table>
<thead>
<tr>
<th>Programming language</th>
<th>IDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML, CSS, JavaScript Python</td>
<td>Dreamweaver</td>
</tr>
<tr>
<td></td>
<td>XCODE</td>
</tr>
</tbody>
</table>
CHAPTER 3

IMPLEMENTATION

3.1 TEMPLATES

In any Web Application, consistency is important. A feel of going to a different website with every click of a button within the application is not what someone intends to create as this would not be a user friendly approach. Stock templates are available, however, this does not satisfy a special need of a developer, nor does it enhance the capability of a developer. It does indeed spur new ideas within a developer and also does save time along with money in certain development scenarios. By using a custom design to web pages, unique designs can be achieved. It also serves to be more adaptable to a customer’s needs [3]. Many software development tools offer templates too. Considering the benefits of both custom designs and stock templates, a mixture of both has been used in the development of the front end of the web application. The Dreamweaver software used for coding purpose in this Thesis implementation does offer certain templates. They however are very basic and therefore, have been worked upon it to a massive extent in this project.

3.1.1 Login

Authenticated users alone are allowed to access the web application. This is a basic need of any tool or application in order to keep trespassers away [4]. Once credentials have been entered, and the login button has been clicked, HTML calls the Flask API that lies in the middleware. It also retrieves information to check if the user is valid or not. If the user id or the password is incorrect, then the page is refreshed and set to blank fields for the user id field along with the password field. The credentials would have to be entered correctly, only this time, with the correct credentials. Three style sheets are used in this page. Figure 3.1 shows the template of the login page.
3.1.2 Create Ballot

The appearance of a page has a lot to do with how often a user would want to use a tool. It is for this reason that seven Cascading Style Sheets are being used in the Create Ballot page. These style sheets can be linked in multiple ways such as, external linking, inline linking, and internal linking. The style sheets here are linked externally to the HTML code with the help of ‘link’ attribute. With external linking of CSS to HTML, there is much cleaner web page code which helps in maintenance, and organizes styling in dedicated files.

To create a ballot, the poll name, and options to select from need to be specified. This page is found after clicking on the Create Ballot tab on the left side of the window. Textfields are used to achieve each of these cases. Four options are asked to be specified. There is an
option to reset the textfields back to empty fields. After all textfields are worked on, the submit button needs to be clicked which will then initiate the Post method in the middleware. Figure 3.2 contains the create ballot template.

3.1.3 Cast Vote

The major aspect of this web application is the voting. The implementation is such that the user gets to choose multiple options for a ballot depending on priorities. This approach is ideal when two or more options are to be chosen from. Clicking on the Ballot Item List tab that is located on the left sidebar will take the user to the window where voting is done. The Ballot Item when clicked on displays the list of ballots that have been created and not been voted on yet.

Clicking on the Ballot List will show the user a list of Ballots that are yet to be voted on. The user can select any of the ballots that are on display and poll on it using the rank order voting system. Each option has a dropdown menu that indicates numbers from 1 to 4, with 1 being the highest ranked option and 4 being the lowest. No two options can have the same rank number. Once the vote is complete and the data is entered into the database, this ballot is moved from the Ballot Item tab to the Results tab. Right after voting, the result of the vote is displayed. This feature has been implemented in a fashion such that segregation is clear and no ambiguity exists. All non-voted ballots being placed in the Ballot Item tab and all other ballots in the Results tab.

After the rank values have been placed for each of the options, the rank for the corresponding option needs to be stored in some buffer so it can be retrieved by the Getter API from the database. For this, an array is used. Each option has a unique array index whose value is read with the help of the get(options).keys method. The ranks are stored as part of a unique ballot id and not against the name. Figure 3.3 shows the Cast Vote template.

3.1.4 Results Display

The votes that have been cast is seen here as a chart. The Google API loader is being made use of to display the results we extract in the kind of fashion we want it to be displayed. JavaScript code is written to draw the chart and display the results interactively. The Google.load function takes in three parameters which are API name, API version, and optional settings. Each API has a version and a revision number of the form
Figure 3.2. Create Ballot template.
<form name="submit_box" action="/createBallot/postBallot" method="POST">
  <fieldset>
    <div class="control-group">
      <label class="control-label" for="focusedInput">Ballot Name</label>
      <div class="controls">
        <input name="ballotName" class="input-xlarge focused" id="focusedInput" type="text" />
      </div>
    </div>
    <div class="control-group">
      <label class="control-label" for="focusedInput">Option Name 1</label>
      <div class="controls">
        <input name="option1" class="input-xlarge focused" id="focusedInput" type="text" />
      </div>
    </div>
    <div class="control-group">
      <label class="control-label" for="focusedInput">Option Name 2</label>
      <div class="controls">
        <input name="option2" class="input-xlarge focused" id="focusedInput" type="text" />
      </div>
    </div>
  </fieldset>
</form>
<input name="option2" class="input-xlarge focused" id="focusedInput" type="text" />

</div>

</div>

<div class="control-group">
  <label class="control-label" for="focusedInput">Option Name 3:</label>
  <div class="controls">
    <input name="option3" class="input-xlarge focused" id="focusedInput" type="text" />
  </div>
</div>

<div class="control-group">
  <label class="control-label" for="focusedInput">Option Name 4:</label>
  <div class="controls">
    <input name="option4" class="input-xlarge focused" id="focusedInput" type="text" />
  </div>
</div>

<!--div class="form-actions" -->
<!--button type="submit" class="btn btn-primary">Save changes</button-->-->
<input type="submit" value="Submit"/>
<!--button type="reset" class="btn">Reset</button-->-->
<!--</div>-->
</fieldset>
</form>
</div>
Figure 3.3. Cast Vote template.
<div class="control-group">
  <label class="control-label" for="focusedInput">{{{ ballot.get('options').keys()[0] }}}</label>
  <div class="controls">
    <input name="value1" class="input-xlarge focused" id="focusedInput" type="text">
  </div>
</div>

<div class="control-group">
  <label class="control-label" for="focusedInput">{{{ ballot.get('options').keys()[1] }}}</label>
  <div class="controls">
    <input name="value2" class="input-xlarge focused" id="focusedInput" type="text">
  </div>
</div>

<div class="control-group">
  <label class="control-label" for="focusedInput">{{{ ballot.get('options').keys()[2] }}}</label>
  <div class="controls">
    <input name="value3" class="input-xlarge focused" id="focusedInput" type="text">
  </div>
</div>

<div class="control-group">
  <label class="control-label" for="focusedInput">{{{ ballot.get('options').keys()[3] }}}</label>
  <div class="controls">
    <input name="value4" class="input-xlarge focused" id="focusedInput" type="text">
  </div>
</div>
version.revision. Mentioning just the version number would mean that the latest revision from that particular version is being used [5]. The optional settings parameter can having one of several parameter options. In the context of our code, packages are what are being used. Since the results need to be interactive and make use of charts, it is the packages parameter that specifies that a pie chart or a bar chart for instance, could be loaded.

Dynamic loading is possible using google.load API’s. This will enable API’s to not load during loading of a web page [5]. Instead, it is dynamically loaded when requested for it. Dynamic load is not being implemented here however since scanning of results of multiple ballots are made easy when dynamic loading is not being made use.

Some examples of the module names are Earth module, Orkut module, Visualization module and many more exist [6]. Visualization module is what is used in the code since google charts are part of the visualization API. When the visualization API is loaded, then a callback is invoked that initializes the chart, in this case a horizontal bar chart, and goes on to draw the chart after the data table is populated with the values that we intend to display in the chart form [6]. The google.visualization.arrayToDataTable function does this by generating a data table. Based on the input given, the data types of the columns are automatically identified.

Once the data table is populated, then the kind of chart that needs to be displayed is mentioned using google.visualization.BarChart function. There are several options to choose from depending on which one serves best to the customers’ eyes, and here, we go with the horizontal bar chart. A bar chart for every data table is created. A data table for every ballot is created. Therefore implying, a chart for every ballot is created. Some of the more easier retrieval techniques that are used is with the help of the RESTful approach by making use of the http request methods, such as get method unlike in the SOAP API. The poll.get method can be used to obtain the ballot name, id, and the ballot owner name.

By Clicking on the Results tab, the list of all ballots created are displayed. The user can select the ballot for which they need to see the result.

For the purpose of better understanding, static data values have been placed in the following code (see Figure 3.4 [7]).
function drawChart() {
    var data = google.visualization.arrayToDataTable(["Model', 'Votes'],
                                                  ['Toyota', 11],
                                                  ['BMW', 2],
                                                  ['Mercedes', 2],
                                                  ['Honda', 2],
                                                  ['Mitsubishi', 7]);

    var options = {
        title: 'Cars to buy',
        pieHole: 0.4,
    };

    var data2 = google.visualization.arrayToDataTable(["Location', 'Votes'],
                                                     ['Lake Tahoe', 21],
                                                     ['San Francisco', 19],
                                                     ['San Diego', 13],
                                                     ['New York', 10],
                                                     ['Las Vegas', 31],
                                                     ['Grand Canyon', 28]);

    var options2 = {
        title: 'Vacation Spots',
        pieHole: 0.4,
    };
In the Figure 3.4 [7], two ballots, one for car models, and another for vacation locations have been created and its values are fed as part of its respective data table. The code shown in Figure 3.5 [7] is the dynamic version of Figure 3.4 [7] where a two dimensional array is fed as a data table. This will enable creating multiple ballots and its respective data tables will be generated automatically, and the chart draw for every ballot voted on.

3.1.5 Registration

Users need to register before they can work with the web application. The process is simple. In the log-in screen, if the intended user does not already have a valid user id and password for the tool, then the Register link can be clicked which takes the user to the register page. Here, four textfields for first name, last name, userid, and password is asked to be entered.

3.2 Interfacing

Interfacing, in other words, a middleware API is required to post data into the database as well as request for results from the database. For this purpose, the Getter API and the Setter API is developed.

```javascript
var chart = new google.visualization.BarChart(document.getElementById('donutchart'));
chart.draw(data, options);

var chart2 = new google.visualization.BarChart(document.getElementById('donutchart2'));
chart2.draw(data2, options2);
```
{% extends "base.html" %}
{% set title = "Results" %}
{% block content %}

<script type="text/javascript">
  google.load("visualization", "1", {packages:['corechart']});
google.setOnLoadCallback(drawChart);

function drawChart() {
    '{% for poll in pollData %}'
    var array = new Array();
    var arrayInception = new Array();
    arrayInception.push("Candidate");
    arrayInception.push("Votes");
    array.push(arrayInception);

    '{% for key in poll.get("options").keys(): %}'
    arrayInception = new Array();
    arrayInception.push('{{ key }}');
    arrayInception.push(Number('{{ poll.get("options").get(key) }}'));
    array.push(arrayInception);
    '{% endfor %}'

    var data = google.visualization.arrayToDataTable(array);
    var options = {
        pieHole: 0.4,
    };

    var chart = new
    google.visualization.BarChart(document.getElementById('{{ poll.get("_id") }}'));
    chart.draw(data, options);

    '{% endfor %}'

}
</script>


3.2.1 Getter API

The data values for every ballot are stored in the database. The client side needs to find a way to interact with the database to get hold of these values for result displaying purpose. Since there is no querying as such done in the server technology being used here, the agenda is to communicate with the middleware and yield the results. This is achieved with the help of a Getter API which is shown in Figure 3.6. The Getter API retrieves information about every ballot so information on every ballot that has been voted on is displayed on the Results page. Every ballot has a unique id associated with it. It is possible to have more than one ballot with the same name, but the id still remains distinct.

The code loops till every ballot that has been voted on has been worked upon. All the voted upon ballots will be part of poll Data. The poll.get method yields us any available information on any one particular ballot at a given time. The name of the ballot is retrieved first followed by the owner name of the ballot. This feature of obtaining the owner name is especially crucial when there are two ballots of the same name, but created by different users. When fetching the id, the data values associated with every ballot item is pushed through from ElasticSearch through Flask, and to the client side. This is how we end up having the results information. No sql queries are required.
Figure 3.6. Interface for retrieving data from server.

3.2.2 Setter API

An interface had to be created between the values that the user enters and the data table which stores those values in the database. It is for this purpose that the Setter API, which is shown in Figure 3.7, is developed. The middleware of the web application takes the input values and places it in the database in JSON format which it takes care of automatically due to the structure of the database unlike in the XML format. Therefore the job in hand is for the middleware to get the input from the client.

The postBallot method takes care of this. The user name, ballot name, and ballot items is retrieved from the front end. An id is built dynamically along with the time at which it is created. Each ballot item is automatically initialized to 1 upon creating it. Initializing this
Figure 3.7. Interface for sending data over to serve.
def postBallot():
    ballotId = "build_"+str(int(time.time()))
    username = request.cookies.get('username')
    ballotName = request.form.get("ballotName")
    option1 = request.form.get("option1")
    option2 = request.form.get("option2")
    option3 = request.form.get("option3")
    option4 = request.form.get("option4")

    esClient = ESClient("localhost:9200")
    ballotParams = {
        "name" : ballotName,
        "createdBy" : username,
        "creationDate" : int(time.time()),
        "options" : {
            option1 : "0",
            option2 : "0",
            option3 : "0",
            option4 : "0"
        }
    }
    esClient.index_update_create('manhattan','pollData',body=ballotParams,docid=ballotId)

    print "Elastic return value:"+ esClient.last_response.text

    listParams = {
        "script" : "ctx._source.ids += id",
        "params" : {
            "id" : ballotId
        }
    }
value to 1 although seems trivial, it is important to do so. Often, a non-initialized variable ends up assigning garbage or a random value to it. A buggy program could consider this nonsensical value to be the rank of that option which is not what should happen. Therefore, initializing the options to any one of the rank options available is needed. The rank value changes when the ballot item has been voted upon.

The ESClient api retrieves the ip address of the server hosting the database. A new index is then created for every ballot that holds the project name, id, ballot items, and pollData where all ballots go into. The json.dumps method will store the id and its attributes in the Json format in the database. The following is its implementation.

```python
_issuedHttpRequest(
    "POST", "/manhattan/pollData/{}/_update".format( 1 ),
    body=json.dumps(listParams),
    validResponses=[httplib.NO_CONTENT] )
eturn redirect (/)
```

3.3 CORE FEATURES

A complete picture of the tool is never given unless some video tutorial or a demo or certain slides about them are given. It is for that reason that screenshots of the core features of the tool are given in the following sections to give readers a sense of what the web application looks like. The feel of it is described the best it can by means of explanation. Below are the screenshots of all the sections of the tool.

3.3.1 Chart View

The results are displayed as a chart view. The web application makes use of the google api loader. The kind of chart used is horizontal bar chart. To view them, the Results tab from the left of the screen is selected. This will let the user see a list of ballots that have been created regardless of whether they have been voted on or not. To view the results of a particular ballot, that particular ballot name needs to be clicked on. The ballots are arranged
in alphabetical order in the results page. Once the ballot has been clicked on, the result is displayed in the form of an interactive graph. The lengthiest bar on the x-axis is the option that has the highest rank. Likewise, the least lengthy bar is the least preferred option. We call these bars the priority bar. The name of the ballot creator appears on the right top of the ballot result screen to differentiate between two ballots of the same name created by different users. The view of the Results list is seen in Figure 3.8.

![Figure 3.8. Result list.](image)

The view of the horizontal bar chart is seen in Figure 3.9.

### 3.3.2 Ballot Creation with Ballot Items Entry

This is the first tab on the left bar of the screen. When clicked on, it shows a list of textfields where the name of the Ballot along with four options need to be entered. The submit button is clicked once the information has been entered. It’s a fairly simply looking web page with no user complexity involved. Figure 3.10 shows the GUI for ballot creation.
3.3.3 Ballot Item List and Voting

The list of ballots that have not been voted on yet can be viewed by clicking on the Ballot List tab on the left side of the window, which happens to be the second option and the default page after user has logged in.

The user can select the ballot that he/she needs to vote on and it takes them to the voting page. The voting page is seen in the second screenshot. The instructions for voting are given at the top of the voting space. An option for each of the four options need to be selected from the dropdown which has 1 to 4 as the possible value without any two options have the same value. Therefore, every ballot option needs to be voted on. After having selected unique values for each option, the submit button is clicked so the information polled gets stored in the database. Figure 3.11 shows the Results list.

The reason why the results bar do not appear on the side and the list of ballot on the left side of the screen below the main tabs is because this would introduce a little more complexity in the web application coding and more importantly, introduce a new technology
Figure 3.10. Ballot entry.

Figure 3.11. Ballot list not voted on.
such as AJAX or jQuery or both. The trade-off for not using those technologies are not heavy. In fact, it is negligible. Performance or speed is not discounted. Page loading may even be in fact quicker due to lesser technologies having to interact with each other to pull up a screen. Understandability while working on future enhancements will greatly improve in this fashion. Figure 3.12 shows how the ranks can be assigned for the ballot items.

![Figure 3.12. Rank order polling on ballot items.](image)

### 3.3.4 Credential Page

The view of the login and logout pages is the same. The GUI view is shown in the following sections.

#### 3.3.4.1 LOG-IN/LOG-OUT VIEW

The log in screen, seen in Figure 3.13, is a typical log in screen which requests the user to enter credentials such as the user id and password. The option of remembering the password if provided. The user can wish to let the browser save the password or the web application. Logging out can be achieved by clicking on the name of user on the top right of
the screen. This will provide a dropdown and the option of logging out exists. Clicking on it will take the user back to the log in page.

3.3.4.2 REGISTER VIEW

The user needs to register before using the web application. If no user id or password exist for the user, the Register link from the log in page needs to be clicked on and this will open the Register window. Registration has been made as simple as possible. The first name, last name, user id and password of the user has to be entered. The same user id needs to be used for logging in. Registration page is seen in Figure 3.14.
Figure 3.14. Registration page.
CHAPTER 4

SUMMARY

4.1 CONCLUSION

The Voting Ballot web application uses collaboration techniques to achieve a specific goal of attaining solutions to a pertaining problem after feedback from multiple users of the web application. Baring in mind scalability and many other factors that play a role in maintaining the web application without trouble in the future and provide ways for easy enhancements, some of the latest technologies such as ElasticSearch server and Flask micro framework are used. Speed and code complexity have been given prime importance. With the use of the mentioned technologies, minimum overhead is obtained due to the ways through which shading has been achieved in ElasticSearch and the lightweight nature of the Flask. HTML5 has been used extensively so support for browsers in the future won’t be lagging. Client side scripting in JavaScript plays an important role in helping the server respond back quicker with lesser number of computations to deal in the database or the middleware.

4.2 OBSTACLES FACED

The technologies used in this web application are some of the best that can be found. They are new and with new come the complexity of learning. Adapting to ElasticSearch and knowing how interactions take place with ElasticSearch database was a task. Learning how its environment works was a challenge.

The Flask microframework was perhaps the biggest challenge faced. Just setting it up on a machine needed a lot of research and thought. After several futile efforts for multiple hours trying to get it up and running on a Windows machine, I was eventually able to get it to run on the Linux machine. On the Linux machine again, there were package update issues and took a long time before which I was actually able to import flask api’s.

Synchronizing with the backend developer who happened to be in a remote location was an issue. While the database was being worked on, there was still no server that I could access to see the constant progress. Therefore, we had to be part of several hours of meetings
together to achieve the final product. This way, we had to at times defer certain implementations while waiting to hear back on a particular issue from the other developer.

The deadline of completing this before the end of the term was a challenge and the hours put in on the Thesis work on a daily basis was exhausting.
CHAPTER 5

FUTURE EVOLUTION

A web application can be enhanced to no bounds. It is however important to keep in mind the overhead it might cause if it is hosted on a fairly small sized server. Considering practical number of resources is available, and considering the amount of coding work that needs to go in to implement some of the features, there are multiple features that can be added to this web application.

Going beyond rank order voting system can be explored. There are several ways in which a vote can be casted and it should not be too hard an effort to enhance the already existing choice of voting standard, which is the rank order voting system, into something that is even more substantial and versatile. Ballot creators can choose to specify the kind of vote that needs to be casted to the users of that particular ballot. Instant run-off voting system can be implemented by choosing the option with the highest number of first choice votes assigned.

As the software users increase in number, the presence of each of these users on every ballot posted will not be a requirement, in fact, it could be that we need some users to not vote on the ballot at all. Permission can be enforced by the ballot owner allowing the owner to choose from a list of users and request only them to vote on the ballot item. One can go a step further and assign read, write permissions to each user. This can be done in one of two possible ways. Either the ballot creator can do this or there could be an administrator for the entire web application, and the administrator can control who sees and can do what. The latter feature will prove to be beneficial when there are multiple users that do not wish to explore new features inside of a tool.

Everything that one does, be it in life or in computer software is for the sole purpose of attaining a result that one strives for. In the web application, the result is displayed in an interactive and an attractive manner by means of a horizontal bar chart. A further enhancement would be to allow users to choose from chart options. Users should be allowed to view the results in the way that they want to. It should have an option of viewing it as
either a 3D pie chart, or a rotating pie chart, or a sliced chart, or a bar chart. There are google API’s to make use of for easier implementation of these features.
REFERENCES


