System Requirements Report and Architecture

Introduction

Online election systems have replaced the traditional paper ballot in many democracies. Our own institute, IIT Guwahati, is no exception to this trend. They afford convenience, both to the voters and to the counters, are cheaper to conduct, produce results faster and are considered to be more reliable and less susceptible to corruption. A country like India faces many challenges due to large scale illiteracy and incomplete penetration of technology. Fortunately, this is not the case with our campus where the majority of the electorate is literate and access to internet/intranet enabled computers is easy, enabling us to make an elaborate online system. Further an internet based system utilizing the humble personal computer as the endpoint is cheaper as PCs are ubiquitous and require minimal planning and expenditure

A good online election system must satisfy the following requirements:

1. The identities of the candidates who are contesting the elections should be freely available to all. This is especially true for us as candidates repeatedly apply for and withdraw their applications.
2. The source code should be open for all to see and verify.
3. The voting process should be convenient and efforts need to be taken to ensure that people can vote from their personal PCs. Extensive encryption and security features need to be included to enable this.
4. The system should be secure and difficult to hack.
5. All interested parties must be able to verify the authenticity of every vote. For this we propose a public key encryption based certification system which will allow all interested parties to verify that the declared votes are cast as claimed by the system. Hence we ensure that unless it is easy to factor large numbers, no hacker can modify the results of the election. Not even the administrator will have this ability.
6. The votes cast by the electorate must be anonymous. We propose to meet this criterion even though we have user accounts and certification. This is achieved by randomly permuting all votes.

The following features, although not absolutely critical, will greatly increase the utility of the system:

1. Results can be generated rapidly once the election is over.
2. Detailed analysis of electorate behavior can be made available publicly. This should be done while ensuring full confidentiality of votes (in our case, even the system does not know who casted which vote).
3. To reduce administrative burden, the system should be largely automated.
4. The interfaces should be intuitive and self explanatory, for the administration, electorate and the candidates.

Naturally we plan to make our system as good as possible. To this end we

propose to a system satisfying all the above mentioned criteria. Any software, no matter how well designed, needs to be useful. Hence the requirements of the final end-users must be always kept in mind. Hence we do a System Requirements Analysis by talking to various people who have and will use the system. This is presented below followed by a detailed architectural specification.

Field Work

From the perspective of

Mr.Brijesh Kumbhani - Chief Election Officer Gymkhana Elections-2014

Election Procedure in IITG:

Initiation:

1.DOSA appoints the CEO for the elections.

2.CEO is given the rules for elections by DOSA.

3.CEO selects his teams.Appointed team are of three kinds:

(i)Election organising team

(ii)Technical team

(iii)Squad

4.Based on the election rules and policies of the elections of that year,the CEO instructs the technical team to build a system to conduct the elections.

5.The technical team starts modifying the previous version of the election system as per the requirements.

Nomination:

6.CEO sends the mail calling for nominations to various posts in The Gymkhana Council.

7.Candidates are asked to submit a Nomination forms along with the required documents to the election organising team on a given date.

8.CEO sends the details of the candidates to the student affairs to confirm their candidature.

9.A final list of the nominated candidates is released along with the agendas and credentials of the candidates by the election commission on the election webpage.

Complaints:

11.Squad sees to it that the prescribed rules are being followed and no illegal campaigning happens.

12.CEO handles any complaints against the candidates.

Conduction:

13.On the day of elections,DOSA and CEO verify the election system by voting themselves and checking the results.
14.The organising team gives each of the voter a unique key.
15.The voters then login from a set of selected computers in CC using their unique key and webmail id to vote.
16.Organising team makes sure no malicious activity happens during the voting procedure.

Declaration:

17.Results are compiled and released on the election page.

Suggestion:

In the present system, nomination happens offline which is a painsome process.It could be made online.

From the perspective of
Mr.Soni - Head of the Technical team Gymkhana Elections 2014

What technical team does:

1.Receives the list of additional features to be incorporated into the existing election system.
2.Makes the necessary edits to or adds code to the existing program to achieve the demanded functionality.
3.Monitors the server traffic during the election to ensure no malicious activity happens.
4.Make periodic backup of the database manually.Should anything go wrong,the database is restored to the last database is restored to last safe point thus avoiding the question of re-elections.
5.Compile the results of the election to a PDF and send to the CEO.

Some technical flaws in the system:

- System was quite vulnerable.
- No feasibility check on the server capacity.
- CC portal wasn't requested for allowing login from webmail.
- No record of who connects to the server.
- System was not open source.
- Keys generated were not confidential.
- Webmail IDs for some people were not registered.
- A voter's vote is not anonymous.Each vote is mapped to a webmail ID.

Suggestions for improvements:

- Automatic backup of data could be done instead of manual back up as practised.
- Online nomination and verification of the candidates.(CEO's take on this: this could make the nomination process less serious as a candidate can be nominated by a friend with the knowledge of his password)
- Barcode behind the ID card could be used to mark the students at the time of issuing the key instead of the searching in a list of students and marking them manually.This could decrease the manpower required drastically.
- A page for each candidate where his/her details,videos,agenda and credentials are available.
- Rejection of candidate was done publicly in the last elections.This could be made private.
- PDFs were generated for results.Something fancy could be done like displaying the stats etc.

From the perspective of the junta:

Some random students were interviewed as follows

1.Did you vote in the last Gymkhana elections?If not why?

80 percent of the students replied yes.

2.Did you face any problem with in the UI while voting?

80 percent of the students who voted did not face any problem.

3.Rate the entire voting experience.

When asked to rate the voting experience on a scale of 5,the average rating turned out to be 3.6

4.Where would you like the debating on the election agendas of the candidates happen?

The suggested options being facebook,election page, 50 percent students prefered facebook and remaining the election page.

5.How important is it to you that your vote remains anonymous?

Every student wanted that his/her vote should remain anonymous.

6.Do you feel the admin or any volunteer involved could be manipulating the results for satisfying their interests?If yes,do you have any suggestions to make sure this doesn't happen?

90 percent of the students didn’t doubt any such malicious activity during elections.

7.Would you like ensure that every vote cast is delivered to the intended candidate?

Most of the students were interested to check the validity of each vote cast in the election.

8.Did you find any technical loophole in the present election system?If yes please tell us.

some students mentioned some loopholes in the process as follows:
-

9.Should the code used in the election conduction be open source?why?

We aspire to make an election system that meets the expectations of each resident of the campus.

Do you have any suggestions,improvements or new features that could be incorporated?

Interesting Comments

Why did you not vote in the Gymkhana elections?

> As a final year UG student, I do not consider it my place to vote in an election the results of which are not likely to affect me.

> All the main positions were uncontested. However the feedback below is for the election before that

Did you Face any technical difficulties last time? If so what?

> The database could be easily manipulated by a mole in the team conducting the election. Although I believe this was not the case, it would be reassuring if such loopholes are fixed.

When I logged in to vote, it said that "your vote has already been cast", when in fact, I hadn't voted. So either there was some glitch in the back end management, or somebody else had voted through my account, which is an even more serious security issue.

bad interface !!

1. The ability to vote for 1-7 candidates(senators) should be done away with. Everyone should be able to vote for a fixed number of candidates, either 1 or 2 or, say, 7. Some people vote for 2 senators whereas some vote for 7. That automatically puts different weights to the votes of different voters.
2. I would like the administration to go through Wikipedia pages on Gerrymandering and other loopholes that crop up during elections. The first-past-the-post method should be done away with. The primaries method should be adopted. In other words, the plurality voting method should be done away with and the majority voting method adopted.

this time i had to get my password 3 times in order to vote as every time there occurs some technical error

the people there know the password and webmail id, so they can manipulate the results.

Do you think it should be made open source? Why?

It should be made by a group of professors whose identities should not be revealed to anyone but the Director and the Chief Election Commissioner.

for further scope of improvement and detect any kind of loophole.

So that other developers can suggest improvements.

So we can suggest any improvements

The code shouldn't be open source as it makes the website vulnerable to threats. Here are some features we thought for online election system.
https://docs.google.com/document/d/1x8tAMUsDx4hLkQn5lnHMlZWjEVZtNjgy9sc4uB8JRzo/edit?usp=sharing

Any other suggestions?

The user interface could be better.
Instead of OTPs if the system could be implemented through webmail password itself without compromising the security.

Authentication of users can be done through the bar codes on ID cards.

1. There should be year-wise constituencies. That is, a 2nd year voter should be able to vote for 2nd year candidate for a seat reserved for a 2nd yearite, a 3rd year student for a 3rd year candidate for a seat reserved for 3rd yearite. As most students of different years do not know the capability of people from different batches, a lot of random voting takes place. However, for some posts such as those of the VP or Executive members, all should be eligible to vote. Further, 4th year students should serve as advisors in the Gymkhana Council and should be roped in to supervise elections as they are not likely to be influenced by fear or favour.
2. Further, a disabled student may be appointed to serve on the Gymkhana Council. Also during elections, special facilities for disabled voters should be provided.
3. The election commissioner should be provided with security cover during the period of elections so that she/he may be able to conduct elections fairly.

1.) even there should be a portal where we can ask them on agenda anonymously
2.) there should be a open page accessible to all where % of agenda completed by outgoing senate should be mentioned in details like what they had in their agenda & what they accomplished.

GUI can be improved making the system user friendly. The database can be made more secure so that no one can alter the results.

Desired Salient Features

Features

From our fieldwork, we have come up with a list of things that we would desire from the election management system. The architecture is carefully designed to absorb these and many other features that one may desire of the system in the future.

Pre-Election:-

- Once the posts for the election are decided the website will be updated with those.
- The nominations for the various positions will be displayed.

- **Contestants' Pages:** Once the electoral debate is held , the admin will have the option to embed it's video in the web page(considering the limited server space provided embedding the video will be preferred over uploading it.)
- All the contestants will have their own web pages in which they will display their agendas. Also there will be an option for the student body to chat with the contestant through a chat server.
- **Online Debate:** There would be a dedicated system for online debate in which the IIT Guwahati junta can post their questions and doubts concerning the contestant or in general. *Like* and *dislike* buttons will also be provided for the answers so that the people can show their support to the contestants they feel are capable.
- **News Feed:**There will be an option of newsfeed where the the voters can be informed about the heated discussions going on between contestants and the people.
- There will be a report abuse option if any person feels that something wrong is being done by a person.
- **Endorsements:** An option of endorsement of a particular skill put up by a contestant on his/her portfolio will also be provided.This is to show the support to a particular contestant .However to maintain transparency the people will only be able to endorse after logging in and the list of people endorsed in a particular contestant will be available publically.
- **Question Answer Interface:** There will be a separate question and answer interface for each contestant separately where arguments will be exchanged on the agenda webpage itself so that it is convenient to the user to just *highlight* a topic in the agenda and ask a question relating to that.

<u>During Elections</u>:-

- **One vote per person:** The voting status of each voter will be recorded while voting so that each student can only vote once. Furthermore duplicates in votes can be easily detected in the final list of certificates, making it impossible, even for a malicious administrator to forge votes.
- **Votes are everything:** Although we trust the administrator to take the proper action in many instances, we try to trust nobody with the votes. Nobody can forge votes.
- **Anonymity:**The anonymity of the voter is maintained during the process.
- **Security:**Major focus will be kept on encryption and security of data.

<u>Post-Elections</u>:-

- **Analysis and Statistics:** Once the elections are over a detailed analysis of the voter turnout will be provided.
- **Result:**The number of votes received to each of the candidate will be displayed.
- **A record of the promises fulfilled:** A log will be maintained all throughout the year to keep a tab on the the promises made by the contestants and how many of them have been fulfilled. We are unsure of how to implement this though. This will be helpful in rating the the elected members according to the work they are doing.

System Architecture

Purpose

This article describes, in detail, the planned architecture of our online election system. It serves three purposes. First, it formalises the organisation of the system and enables us to see, in advance, any problems relating to code organisation that can arise. Further it also serves as a medium to communicate to the entire team what is to be done so that we do not waste time holding long meetings. Integration is also easier if everybody works towards a common, well defined, goal. Moreover, it also serves as documentation as we describe almost every function and class in detail. In fact, for the database manager, we end up writing the inline documentation before we write even a single line of code! This is partly in line with Donald Knuth's ideas of 'literate programming'.

This also formalises exactly what the system will do and what requirements will be met. We consider this important as merely stating the requirements leaves out many details which may be crucial to some of the functionality of the system

Overall Layout

We broadly divide the software into three major components. One handles the individual web pages (ie. is present in views.py). It includes gathering of data and transferring it between the web pages and the database. The second component consists of the HTML/JS/CSS components which decides how data will look like and provides client-side dynamism. The third component handles the database querying and provides cryptographic services, encapsulating all data organization details into one module. It is important to note that the database will never be directly accessed by any other part of the program. They will only perform function calls to the database manager and will be unaware of cryptographic services (except for the existence of passwords, of course) and data organization. Further the views.py will have exclusive access to the web-pages and will be responsible for issuing commands and performing queries. Naturally the template will be solely responsible for the physical layout of the system. We plan to use the excellent Bootstrap libraries for this purpose.

This strict and deep division serves two purposes. First, it enables us to divide the work amongst our team members. Second, it is good software practice to isolate various parts of a program so that they can be developed and modified independently of each other.

Further major divisions are made in the view manager (views.py). The divisions are according to the various pages in the website. Again, note that these components do not communicate to each other directly. Instead all communications are through the database. This is both good software practice and is in keeping with the stateless nature of most web servers. The various web-pages are divided as elucidated in the UML component diagram and in the documentation below.

Here we follow two different format for two different kinds of modules. One of the primary advantages of having a detailed technical documentation prior to actually writing any code is that the interface between modules is clearly specified. However it has the disadvantage of creating inflexibility in the system. Therefore we follow the policy of having detailed descriptions of all the interface functions and merely a functional description where no interfacing is present. Note that even though we do not give a function by function description, we do fix their behaviour so that all modules can behave accordingly, in full awareness of what the rest of the system will do.

The Platform

We plan to use the Python-Django platform for the server and Bootstrap templates for the client side front end. Below we justify our choice and elaborate our concerns with it.
We chose to use the Django platform primarily for its security features. With this, we greatly reduce (and almost eliminate) the possibility of the most common and dangerous attacks like SQL-injection attack, cross site forgery attack, identity theft attack etc. Furthermore it comes equipped with the pyCrypto which is essential for our extensive cryptographic services. The Django framework is also designed to be scalable which is important for server side applications, especially if we decide to take this to a setting larger than the Gymkhana elections. Our primary concern with using Django is that many web hosts do not support it or support it at an additional cost. Since, inside the campus, we have our own servers this should not be a worry.
We choose bootstrap because it gives a very convenient way to generate good looking web-pages that also scale well across different screen sizes. Since we have quite an extensive back-end, we do not want to waste resources writing our own CSS/JS when a wonderful, readymade option is available.

Portability

First, our system need not be very portable as, due to security considerations, we will actually be restricting the number of computers from which election will take place. The pre and post election parts are portable as they are entirely web based and will work on any browser. The fact that we are using Bootstrap further ensures that it will run in all browsers and all screen

sizes. Nevertheless the system is highly portable as the server can run in any system containing Python and the necessary libraries. In fact, we can have an installation script that will install all the dependencies. The front end client can of course run in any web browser.

Stability

Since our application is data critical (it would be disastrous to lose vote data), we need to ensure that it is extremely stable. To this end, we have dedicated a week to system testing and debugging. Furthermore, we constantly back our data up (across different places) so that recovery is always possible.

Design Considerations

When we design our system, we need to keep several things in mind:

1. The system is to be developed by many people. Therefore we need to come up with a decoupled set of modules that can be developed in parallel
2. Our system needs to be used by laypersons. Currently an entire technical team is employed to manage the elections. We want to greatly reduce the requirement for that, so that the CEO can easily manage most parts of the election without technical support.
3. Security is of high importance as people need to be able to trust the results of the election
4.

Component By Component Architecture:

Database access methods (databaseManager.py)

The database access methods form the interface to the database. No other component accesses the database directly or is aware of any internal details about how data is accessed. Often times we require the other modules to provide formatted JSON to store the data and this module never looks inside the string (except of course to prevent SQL injection attacks). This module also handles all cryptographic functions

loginUser (username :string, password :string) :bool

Although we have not yet decided how we will authenticate users, this function needs to be called so that the plaintext password can be stored in the database while the user is logged in (it will naturally be removed as soon as the user logs out). This is to enable our cryptographic services. This function will only store the password if the election is in progress as otherwise the plaintext password is not required.

loginUser (username :string, password :string) :bool

This deletes the plaintext password from the database and reports the status of success/failure

registerVote(username :string ; vote :string) :bool

The registerVote function will contain 2 input strings one containing the username of the voter and another containing the vote. The vote itself will be a JSON string that contains votes for each position.
Further it will generate a certificate and store it as validation of the vote. The password is required to be present in plaintext in the database for this purpose. The password is added when the user logs in and is removed when the user logs out.
It will then modify the database to reflect these changes - assigning the vote, changing the voted value for the voter to true etc.The function will return True if the vote is registered, else in case of an error it will return False.

getUserDetails (username :string) : dict

the function will contain username as an input string which it will use to get all the other information about the voters as given in models.voter and return it as a dictionary. This may be used by various other modules.

Format of value returned:
{'name': …, 'username": …, "voted": …, …} #this will be as specified in the UML diagram

validateAllVotes () :list

Returns a list of boolean values. Each element indicates whether or not that particular vote was valid (after confirming against the certificate string).

getElectionStats () :dict

This function will query the database for votes and generate a list of all candidates along with their obtained votes. It may also find other statistics about the election. Each statistic collected is returned as an element of a dictionary. Eg. we could have a structure as
{"vote-count": {"candidate1": …, "candidate xyz": asdf, …}, "voter-turnout": 0.67, "voter-demographics": {"btech": 0.5, "mtech": 0.4, …}, ... }
Note: this function could be developed by the team working on the front-end of the statistics part as they will know the precise requirements better.

getWinner (stats:dict) :dict

This function will take as an input the dictionary returned by getElectionStats and then find out the winning candidates for each positions and return them as a dictionary (with the position as key and result as value). Note that positions are refered to by a unique id string which is generally not displayed to users and is meant for programmatic use only.

approveCandidate(username :string, approved :bool)

This function will be used by the admin to approve or disapprove a candidate.
It assumes that the person has already indicated their desire to become a candidate and the appropriate record is present in the table.

getCandidateDetails (username :string) :dict

This function returns the details corresponding to a particular candidate. views.py formats this as a JSON string and the database manager is unaware of the details. Proposed format for storing candidate details:
{"username": "...", "post": "*post-id-str*", "picture": *image-reference-to-file*, "form-data": {"*form-field-id*": "...", "*form-field-id*": "...", …} …}

setCandidateDetails (username :string) :dict

As a complement to getCandidateDetails, it sets the JSON string that describes the details of a candidate

getElectionState (state : int)

pre election stage= 0, during election = 1 , post election = 2

setElectionState (state : int)

pre election stage= 0, during election = 1 , post election = 2

getCandidatePost (postId :string) :dict

This function will return all the candidates along with their data competing for a particular post. postId is the unique string identifying a post.

importElectorateData (src :string) :dict

This will be used to initialize the system with data about the electorate. We have not yet decided on the nature of the source. Currently we are considering using either google docs or allowing the upload of files (.xls, .csv, etc.)

addQuestion (candidateUsername :string, srcUsername :string, question :string) :void

Adds a question to be asked of the given candidate's username by the given source's username. The question is given in the 'question' string

getDiscussions (candidateUsername :string) :list

Gets discussions pertaining to the given candidate. The returned datatype has the following format. Note that 'approved' indicates whether or not a question/response has been disapproved by the administrator. All data is retained for records. No posts are deleted by the system.

```
[
        {
```

‘question-id’: ‘...’,
‘src’: ‘...’,
‘question’: ‘...’,
‘popularity’: *int*,
‘approved’: *bool*,
‘responses’: [
{
‘response-string’: ‘...’,
‘src-username’: ‘...’,
‘response-id’: ‘...’,
‘popularity’: *int,*
‘approved’: *bool*
},
…
]
}, {...}, …
]

changeQuestionPopularity (username :string, postId :string, likeStatus :bool, response-id=None :string) :void

Used to like/dislike a post/question or a response referred to by it’s id. This id is the same as the one returned by getDiscussions.

changePostApprovalStatus (adminUsername :string, postId :string, approvedStatus :bool, response-id=None) :string

Used by the administrator to approve/disapprove text in case it is objectionable. This function retains all text for records and the system informs views.py of all posts whether approved or not and it is the page’s discussion as to whether or not to display it (that is, it wll be displayed to the admin but not to the general public)

The Cryptographic Module (cryptography.py)

Although the cryptographic protocols will be handled by the database manager, we want to abstract away details of cryptography away in a separate module. This module will provide high level functions like ‘symmetricEncrypt’, ‘asymmetricDecrypt’ etc. which will take in ascii/unicode strings and return ascii/unicode strings. To manage the precise functions, we will probably be using ‘pyCrypto’ a python library that supports many cryptographic functions including RSA, SHA and DES.

This module will decide on the format of the encrypted text. For instance, since RSA can encrypt only upto a certain length of strings, we will encrypt a symmetric key using RSA and

use that key to actually encrypt the data. All these details will be hidden from other modules which can happily assume that we are running RSA on the entire string. Further issues with the encoding of strings can be appropriately handled by this module (sqlLite does not support the storage of non-ascii characters as strings). We describe some of the functions of this module.

symmetricEncrypt(inpString :string, key :string) :string
Perform symmetric encryption with inpString as plaintext and the given key. The key may be of variable length and the string returned will be in proper ascii format, compatible with symmetricDecrypt

symmetricDecrypt(inpString :string, key :string) :string
Decrypt a string returned by symmetricEncrypt given the same key as was supplied during encryption

asymmetricPublicEncrypt(inpString :string, key :string) :string
Public key encryption using a public key. The key will be a public key (string) generated by Crypto.PublicKey.RSA module

asymmetricPrivateDecrypt(inpString :string, key :string) :string
Decrypt a string encrypted using a public key given the private key. The key will be a private key (string) generated by Crypto.PublicKey.RSA module

asymmetricPrivateEncrypt(inpString :string, key :string) :string
Public key encryption using a private key. The key will be a private key (string) generated by Crypto.PublicKey.RSA module

asymmetricPublicDecrypt(inpString :string, key :string) :string
Decrypt a string encrypted using a public key. The key will be a public key (string) generated by Crypto.PublicKey.RSA module

Web Pages (views.py)

Here we adopt a different method of documentation. Rather than precisely specifying all details of each and every component, we describe the generic functionality of all the parts. This is to give flexibility during implementation. This flexibility can be afforded as each of these components work largely in isolation of each other and communicate solely through the database manager. Hence though the interface between the database manager and these functions need to be defined very precisely (as we have done above), we can afford flexibility in the requirements of each of these pages.

Information Pages:

The information related to the election is shown to the users. Information such as how to vote, rules for voting, Information about the candidates along with their details.
candidate details can be shown using getCandidateDetails.
Information about how to vote and rules for voting can be displayed using a html page which the user will be redirected to if he wishes to see the rules. This information will come directly from the Gymkhana and the technical team conducting the elections will be responsible for generating the corresponding HTML.
Note that after the election, various election statistics will be displayed. Which ones to be displayed will be decided by the system administrator. Furthermore, the various votes and their certificates will be permuted and displayed for anyone who wants to verify the authenticity of each vote. Note that before the election all the public keys are displayed.

Voter's Page

The voter will be redirected to this page if his authentication is successful as a voter
Here the voter will see different thing depending on whether the election is going on or not.
If getElectionState == 0 display the link to the candidate registration form so that the voter can register as a candidate. Further the voter can change their password (again the method of authentication is yet to be completely finalized)
If getElectionState == 1 display the voting system where each candidate is displayed in order of the post they are contesting for ,we can use the functions getCandidatePost and getCandidateDetails. The vote is then stored using the database manager and the portal exits. Note that this page should be accessible only from the relevant PCs. Although we have not yet decided how we will control this (and hence these details are missing from the architecture), we will probably be using a cryptographic system to ensure maximum security. We may even decide to make a disjoint system that is completely disconnected from the outside world.
If getElectionState == 2 display the election results page (once it has been approved by the admin). Note that this page must not be displayed before the election is over and the administrator has approved the display of election results.

Candidate's Page

Once a person has indicated their desire to become a candidate, the system shall give them access to pages that allow them to select the post they are standing for and to fill in the form designed by the administrator for each post. Furthermore they will be given access to pages that allow them to participate in debates and question-answering sessions. Furthermore they can also view the status of the approval of their candidature (deadlines for the processing of the approval will be set by the rules and any complaints will be redressed offline).

The candidates can view the results of the election along with the rest of the community through the information pages.

Discussion Forum

We aim to provide a complete solution for conducting elections. An integral part of this is the open discussion of problems between the candidates and the electorate. In order to support this, we aim to create a discussion forum where the electorate can ask questions and discuss issues with the candidates based on their agenda and the candidates will have the opportunity to prove themselves to the electorate. Note that there is a separate forum for each candidate where people can go over each and every agenda and clarify how the candidate aims to implement them.

Further, the names of the electorate will be accessible only to the administrator to promote open discussion on issues. The administrator needs to know the identities in order to curb offensive/inappropriate content (as defined by Gymkhana rules). The ground rules for the forum will be established beforehand and will be displayed quite prominently in our portal.

The discussion will be in the form of questions and answers. In order to prevent spamming and to enable people to see the most relevant content, we plan to sort all questions by their popularity (as quantified by ‘likes’).

Administrator’s Page

Setting the election up:

The administrator (in our case the election officer) will be presented with a dashboard to manage the election. First they need to set up the election by giving what posts need to be contested for. For each post, they design a form that any candidate wishing to stand for needs to fill along with other details like the number of students to be elected, length of the tenure etc. The administrator also needs to give the electoral roll containing common information like name, webmail id, roll no (if applicable), department, etc.

Pre-Election

Once the election is set up, requests should start coming in when people indicate that they wish to stand for a post (here we make the very reasonable assumption that all those who are eligible to contest elections are eligible to vote). The administrator then decides whether or not to accept the candidature (strictly according to the applicable Gymkhana rules as the admin is answerable to them). If required, the administrator may also choose to revoke candidature (something that happens regularly in IIT-G).

Social activities like debates and questions will go on regularly, but at present we do not see any active role the administrator has to play in them. If required we may give the admin the rights to remove any comments/questions containing objectionable content.

During Election

At this point the administrator has rights to pause/stop/resume the election. We may also provide them access to live statistics (we plan to hide live statistics to other people until the election is over). We have not yet have a policy on whether or not these statistics should be available to the admin. Although the system will be creating regular backups of all votes (a feature whose necessity was expressed by the technical person in last year's elections), we may also choose to give the administrator the capability to create their own backups. There are several schemes possible for creating backups. One method is to create local files, but this suffers from the drawback that if the local system is disturbed all data will be lost. Therefore we plan to send periodic emails to a designated email id containing all the votes.

In case there are multiple administrators, currently we plan to give them all equal rights. In case there are disputes, the only practical way of resolving them is offline. Regarding concurrency of decisions, our system will obey the most recent instructions and will not discriminate among the administrators.

Post Election

Now the administrator (and everybody else) can view statistics, verify the vote certificates etc. Further, we plan to generate printable reports that can be used for official purposes. This is especially important in a government institution like ours where everything must be documented in a proper format. We may also give the administrator the ability to control which statistics are displayed publicly and which need to be included in the report. This is to maintain anonymity and restrict reports to contain only relevant data.

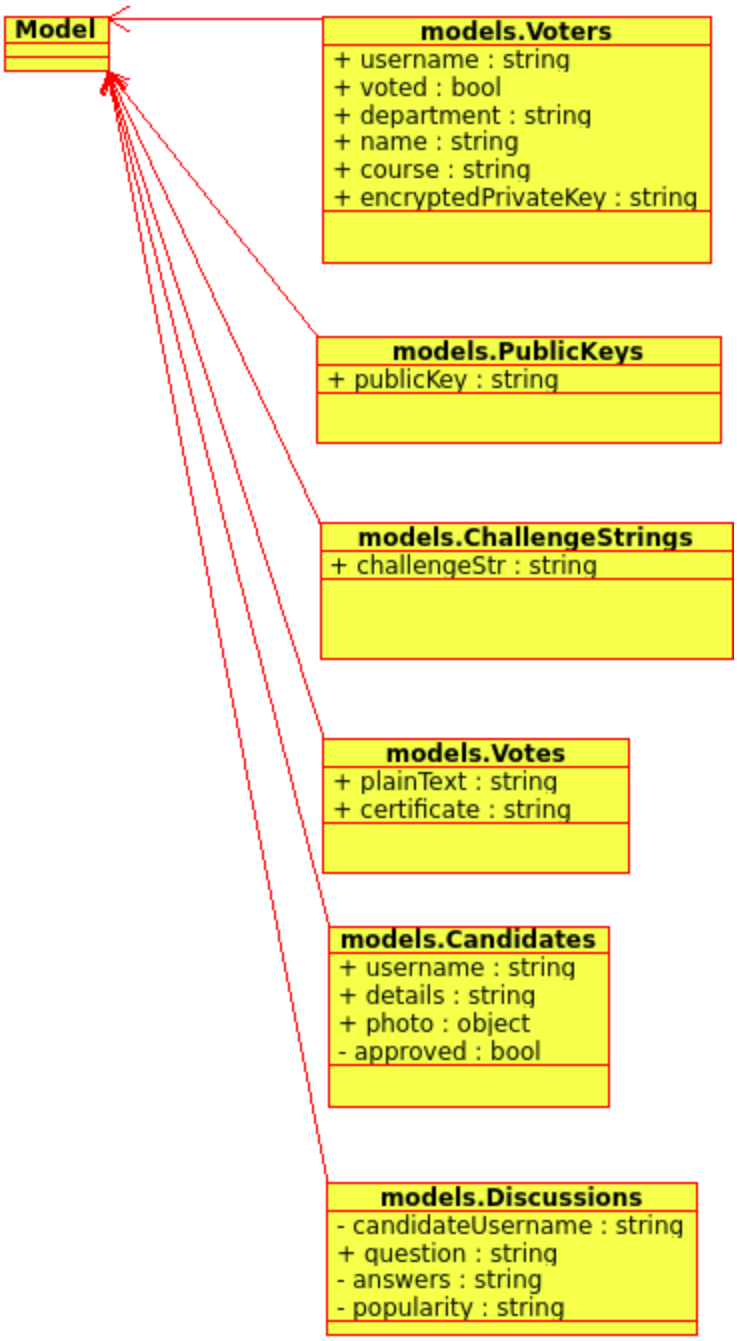
Appendix A - Naming Conventions

To maintain uniformity in naming throughout the project we follow the following conventions.

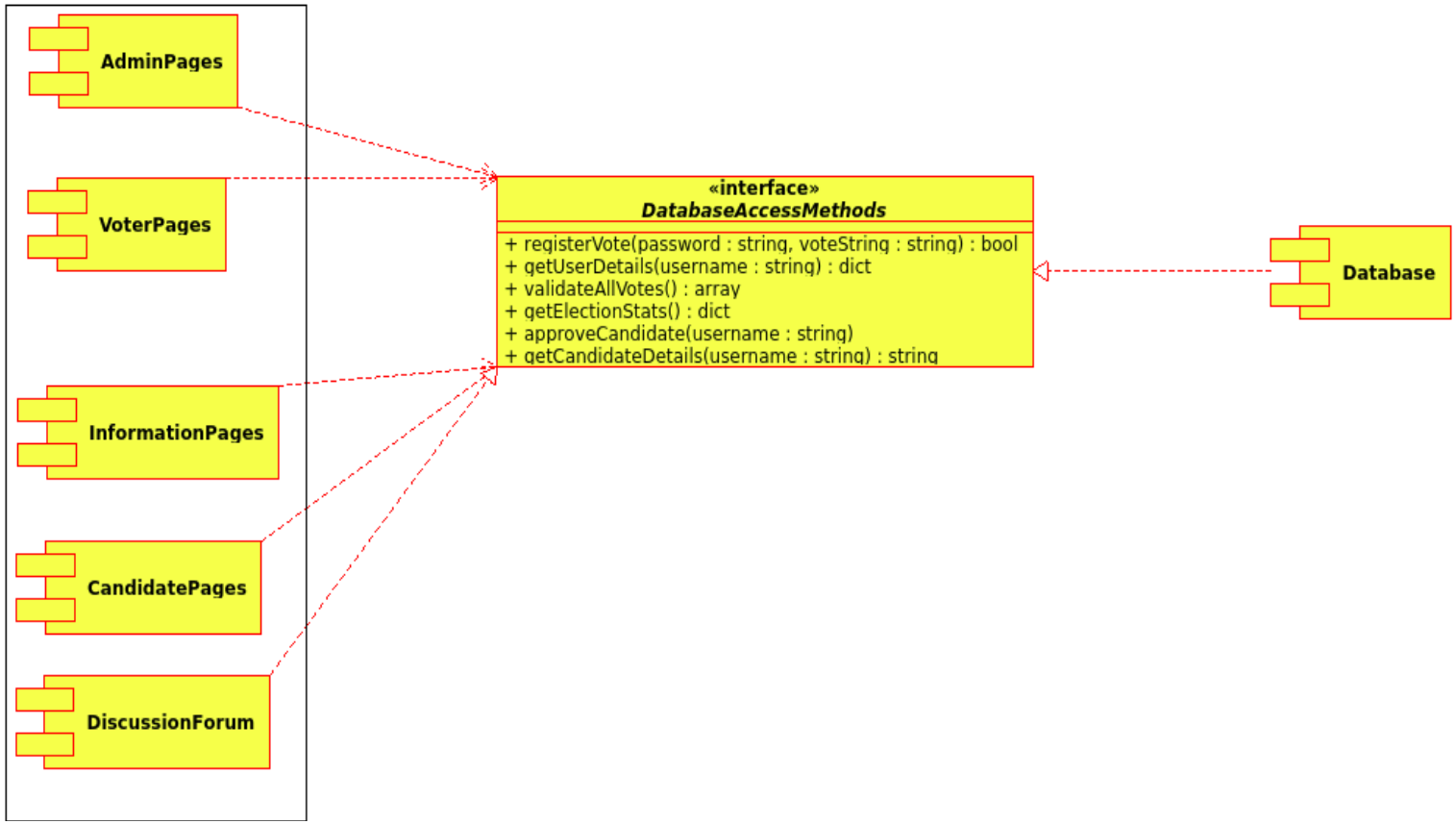
1. Class names will be in 'CamelCase' format
2. Module names will be in 'CamelCase' format
3. Function names that are accessible from other modules/classes will be in 'camelCase' format
4. There is freedom in choosing names for variables for local scope
5. Variables of public scope will have names in 'camelCase' format
6. Variables and functions that are not visible to other modules/classes will have names in ***name-goes-here*** format as recommended by python standards.

Appendix B - UML DIAGRAMS

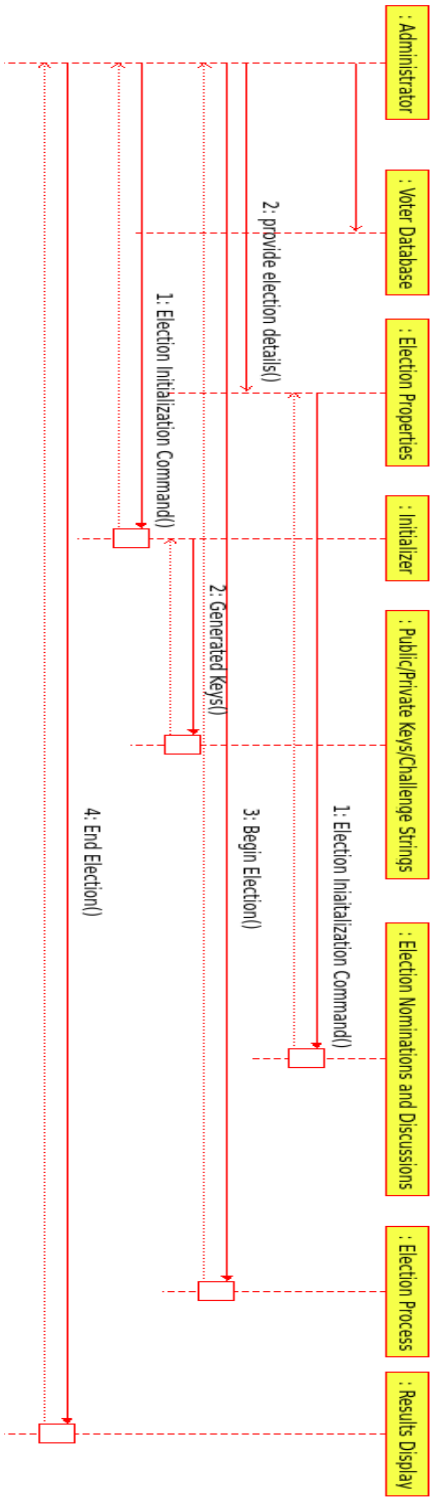
Class Diagram



Component Diagram


AdminPages
VoterPages
InformationPages
CandidatePages
DiscussionForum
«interface»
DatabaseAccessMethods
+ registerVote(password : string, voteString : string) : bool
+ getUserDetails(username : string) : dict
+ validateAllVotes() : array
+ getElectionStats() : dict
+ approveCandidate(username : string)
+ getCandidateDetails(username : string) : string
Database

Sequence Diagram



Appendix C - Cryptography and Security

Cryptography and security are crucial to any such voting system. Although we strive to make the system as secure as possible (we do have a few competent hackers in our team), we ensure that even in case unauthorized persons get access to our data or are able to modify the data, they will not be able to fake the election (unless of course, passwords are weak enough to be susceptible to dictionary attacks, a vulnerability commonly found in even the most secure of
online systems). After completion of the projects, we will test it by trying to hack the system from outside.

To maintain anonymity while having a user accounts system and simultaneously providing certification is a challenging task. We propose the following method to achieve this end.

1. The administrator needs to provide a database of the electorate including webmail ids. It would be ideal if we could get a hash of the passwords as well. In the meanwhile, we will send a mail to all voters with a randomly generated temporary password.
2. We generate a private/public key pair for each voter and store the private key after encrypting it with the voter's password. The public keys are randomly permuted (so that no one knows which key belongs to whom) and declared publicly. They will be used for certification purposes.
3. A set of random challenge strings, S are also generated and publicly declared.
4. When a voter votes, a string is randomly chosen from S and appended to a string indicating their choice. This string is encrypted using the user's private key and stored along with the original string as a vote. This serves as a certification.
5. The order of the votes should be randomized to ensure anonymity is not compromised.
6. The list of votes is publicly declared along with the results and statistics.
7. Anybody wanting to verify the voting can check the certifications provided using the public keys declared earlier. The random challenge strings are declared earlier to make it very difficult for any attacker to forge a vote.

Once the system is initialized with the asymmetric encryption keys and random challenge strings, it is quite secure. This initialization therefore needs to be done in the presence of reliable persons (ie. authorities). To ensure that the keys are not leaked prior to encryption, the source code should be made public and compiled in front of the eyes of the authorities. If one is especially paranoid, one can seed the random number generator with a combination of true random numbers (see random.org) and the system time.