Software Design Specification

IU Committee

November 13, 2012

Team Banana

Justin Ashdown

Eric Grounds

Joel Haubold

Jared Short

Dung Truong

Table of Contents

List of Figures

1.0 Introduction

1.1 Goals and Objectives

There are many committees across the Indiana University System. This project proposes to create a web-based committee management system that will allow committees to function in a more uniform manner, increase their productivity, and allow for greater transparency to the committee's work. This program will include features for scheduling meetings, managing committee documents and members, and allow the public to easily find committee meeting members, and minutes. Since this application will be web-based, there is no special software committee members will need to use the application. Also, since the users are already a part of the IU system, they will not need a different logon and will use the same method to login on other IU systems (such as the Oncourse system).

2.0 Database design

2.1 Relational Schema

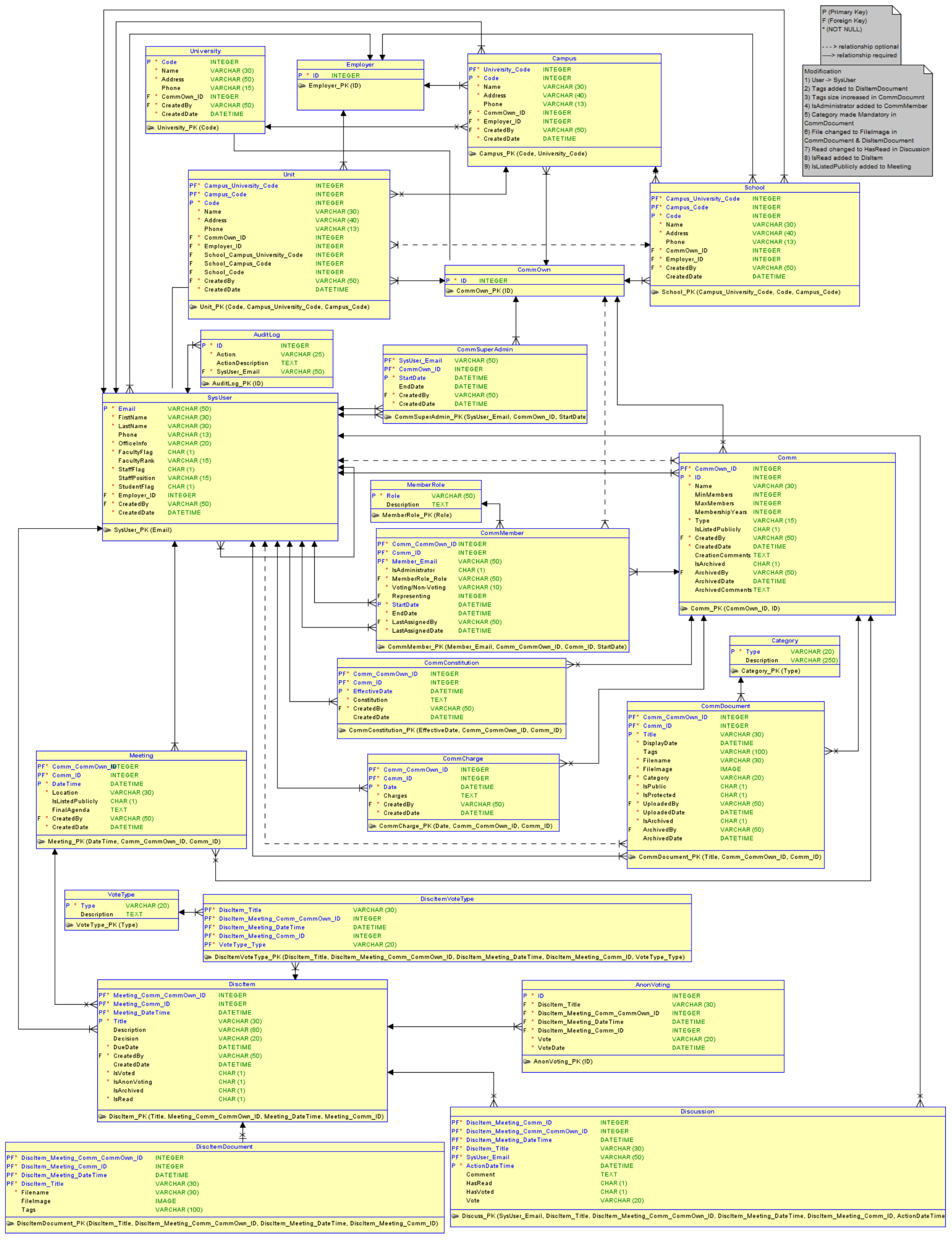
P (Primary Key)
F (Foreign Key)
* (NOT NULL)
- - - > relationship optional
-----> relationship required
Modification
1) User -> SysUser
2) Tags added to DisItemDocument
3) Tags size increased in CommDocumnt
4) IsAdministrator added to CommMember
5) Category made Mandatory in CommDocument
6) File changed to FileImage in CommDocument & DisItemDocument
7) Read changed to HasRead in Discussion
8) IsRead added to DisItem
9) IsListedPublicly added to Meeting
University
P * Code INTEGER
* Name VARCHAR (30)
* Address VARCHAR (50)
Phone VARCHAR (15)
F * CommOwn_ID INTEGER
F * CreatedBy VARCHAR (50)
* CreatedDate DATETIME
University_PK (Code)
Employer
P * ID INTEGER
Employer_PK (ID)
Campus
PF* University_Code INTEGER
P * Code INTEGER
* Name VARCHAR (30)
* Address VARCHAR (40)
Phone VARCHAR (13)
F * CommOwn_ID INTEGER
F * Employer_ID INTEGER
F * CreatedBy VARCHAR (50)
* CreatedDate DATETIME
Campus_PK (Code, University_Code)
Unit
PF* Campus_University_Code INTEGER
PF* Campus_Code INTEGER
P * Code INTEGER
* Name VARCHAR (30)
* Address VARCHAR (40)
Phone VARCHAR (13)
F * CommOwn_ID INTEGER
F * Employer_ID INTEGER
F School_Campus_University_Code INTEGER
F School_Campus_Code INTEGER
F School_Code INTEGER
F * CreatedBy VARCHAR (50)
* CreatedDate DATETIME
Unit_PK (Code, Campus_University_Code, Campus_Code)
School
PF* Campus_University_Code INTEGER
PF* Campus_Code INTEGER
P * Code INTEGER
* Name VARCHAR (30)
* Address VARCHAR (40)
Phone VARCHAR (13)
F * CommOwn_ID INTEGER
F * Employer_ID INTEGER
F * CreatedBy VARCHAR (50)
CreatedDate DATETIME
School_PK (Campus_University_Code, Code, Campus_Code)
CommOwn
P * ID INTEGER
CommOwn_PK (ID)
AuditLog
P * ID INTEGER
* Action VARCHAR (25)
ActionDescription TEXT
F * SysUser_Email VARCHAR (50)
AuditLog_PK (ID)
CommSuperAdmin
PF* SysUser_Email VARCHAR (50)
PF* CommOwn_ID INTEGER
P * StartDate DATETIME
EndDate DATETIME
F * CreatedBy VARCHAR (50)
* CreatedDate DATETIME
CommSuperAdmin_PK (SysUser_Email, CommOwn_ID, StartDate)
SysUser
P * Email VARCHAR (50)
* FirstName VARCHAR (30)
* LastName VARCHAR (30)
Phone VARCHAR (13)
* OfficeInfo VARCHAR (20)
* FacultyFlag CHAR (1)
FacultyRank VARCHAR (15)
* StaffFlag CHAR (1)
StaffPosition VARCHAR (15)
* StudentFlag CHAR (1)
F * Employer_ID INTEGER
F * CreatedBy VARCHAR (50)
* CreatedDate DATETIME
SysUser_PK (Email)
MemberRole
P * Role VARCHAR (50)
Description TEXT
MemberRole_PK (Role)
CommMember
PF* Comm_CommOwn_ID INTEGER
PF* Comm_ID INTEGER
PF* Member_Email VARCHAR (50)
* IsAdministrator CHAR (1)
F * MemberRole_Role VARCHAR (50)
* Voting/Non-Voting VARCHAR (10)
F Representing INTEGER
P * StartDate DATETIME
* EndDate DATETIME
F * LastAssignedBy VARCHAR (50)
* LastAssignedDate DATETIME
CommMember_PK (Member_Email, Comm_CommOwn_ID, Comm_ID, StartDate)
Comm
PF* CommOwn_ID INTEGER
P * ID INTEGER
* Name VARCHAR (30)
MinMembers INTEGER
MaxMembers INTEGER
MembershipYears INTEGER
* Type VARCHAR (15)
IsListedPublicly CHAR (1)
F * CreatedBy VARCHAR (50)
* CreatedDate DATETIME
CreationComments TEXT
IsArchived CHAR (1)
F ArchivedBy VARCHAR (50)
ArchivedDate DATETIME
ArchivedComments TEXT
Comm_PK (CommOwn_ID, ID)
Category
P * Type VARCHAR (20)
Description VARCHAR (250)
Category_PK (Type)
CommConstitution
PF* Comm_CommOwn_ID INTEGER
PF* Comm_ID INTEGER
P * EffectiveDate DATETIME
* Constitution TEXT
F * CreatedBy VARCHAR (50)
CreatedDate DATETIME
CommConstitution_PK (EffectiveDate, Comm_CommOwn_ID, Comm_ID)
CommDocument
PF* Comm_CommOwn_ID INTEGER
PF* Comm_ID INTEGER
P * Title VARCHAR (30)
* DisplayDate DATETIME
Tags VARCHAR (100)
* Filename VARCHAR (30)
* FileImage IMAGE
F * Category VARCHAR (20)
* IsPublic CHAR (1)
* IsProtected CHAR (1)
F * UploadedBy VARCHAR (50)
* UploadedDate DATETIME
* IsArchived CHAR (1)
F ArchivedBy VARCHAR (50)
ArchivedDate DATETIME
CommDocument_PK (Title, Comm_CommOwn_ID, Comm_ID)
Meeting
PF* Comm_CommOwn_ID INTEGER
PF* Comm_ID INTEGER
P * DateTime DATETIME
* Location VARCHAR (30)
IsListedPublicly CHAR (1)
FinalAgenda TEXT
F * CreatedBy VARCHAR (50)
* CreatedDate DATETIME
Meeting_PK (DateTime, Comm_CommOwn_ID, Comm_ID)
CommCharge
PF* Comm_CommOwn_ID INTEGER
PF* Comm_ID INTEGER
P * Date DATETIME
* Charges TEXT
F * CreatedBy VARCHAR (50)
* CreatedDate DATETIME
CommCharge_PK (Date, Comm_CommOwn_ID, Comm_ID)
VoteType
P * Type VARCHAR (20)
Description TEXT
VoteType_PK (Type)
DiscItemVoteType
PF* DiscItem_Title VARCHAR (30)
PF* DiscItem_Meeting_Comm_CommOwn_ID INTEGER
PF* DiscItem_Meeting_DateTime DATETIME
PF* DiscItem_Meeting_Comm_ID INTEGER
PF* VoteType_Type VARCHAR (20)
DiscItemVoteType_PK (DiscItem_Title, DiscItem_Meeting_Comm_CommOwn_ID, DiscItem_Meeting_DateTime, DiscItem_Meeting_Comm_ID, VoteType_Type)
DiscItem
PF* Meeting_Comm_CommOwn_ID INTEGER
PF* Meeting_Comm_ID INTEGER
PF* Meeting_DateTime DATETIME
P * Title VARCHAR (30)
Description VARCHAR (80)
Decision VARCHAR (20)
* DueDate DATETIME
F * CreatedBy VARCHAR (50)
CreatedDate DATETIME
* IsVoted CHAR (1)
* IsAnonVoting CHAR (1)
IsArchived CHAR (1)
* IsRead CHAR (1)
DiscItem_PK (Title, Meeting_Comm_CommOwn_ID, Meeting_DateTime, Meeting_Comm_ID)
AnonVoting
P * ID INTEGER
F * DiscItem_Title VARCHAR (30)
F * DiscItem_Meeting_Comm_CommOwn_ID INTEGER
F * DiscItem_Meeting_DateTime DATETIME
F * DiscItem_Meeting_Comm_ID INTEGER
* Vote VARCHAR (20)
* VoteDate DATETIME
AnonVoting_PK (ID)
DiscItemDocument
PF* DiscItem_Meeting_Comm_CommOwn_ID INTEGER
PF* DiscItem_Meeting_Comm_ID INTEGER
PF* DiscItem_Meeting_DateTime DATETIME
PF* DiscItem_Title VARCHAR (30)
* Filename VARCHAR (30)
FileImage IMAGE
Tags VARCHAR (100)
DiscItemDocument_PK (DiscItem_Title, DiscItem_Meeting_Comm_CommOwn_ID, DiscItem_Meeting_DateTime, DiscItem_Meeting_Comm_ID)
Discussion
PF* DiscItem_Meeting_Comm_ID INTEGER
PF* DiscItem_Meeting_Comm_CommOwn_ID INTEGER
PF* DiscItem_Meeting_DateTime DATETIME
PF* DiscItem_Title VARCHAR (30)
PF* SysUser_Email VARCHAR (50)
P * ActionDateTime DATETIME
Comment TEXT
HasRead CHAR (1)
HasVoted CHAR (1)
Vote VARCHAR (20)
Discuss_PK (SysUser_Email, DiscItem_Title, DiscItem_Meeting_Comm_CommOwn_ID, DiscItem_Meeting_DateTime, DiscItem_Meeting_Comm_ID, ActionDateTime

Figure 1 Database relational schema

3.0 Data design

All of the data classes are represented in the following diagram. Each model also has methods provided by the MVC Model framework for accessing the database and querying data.

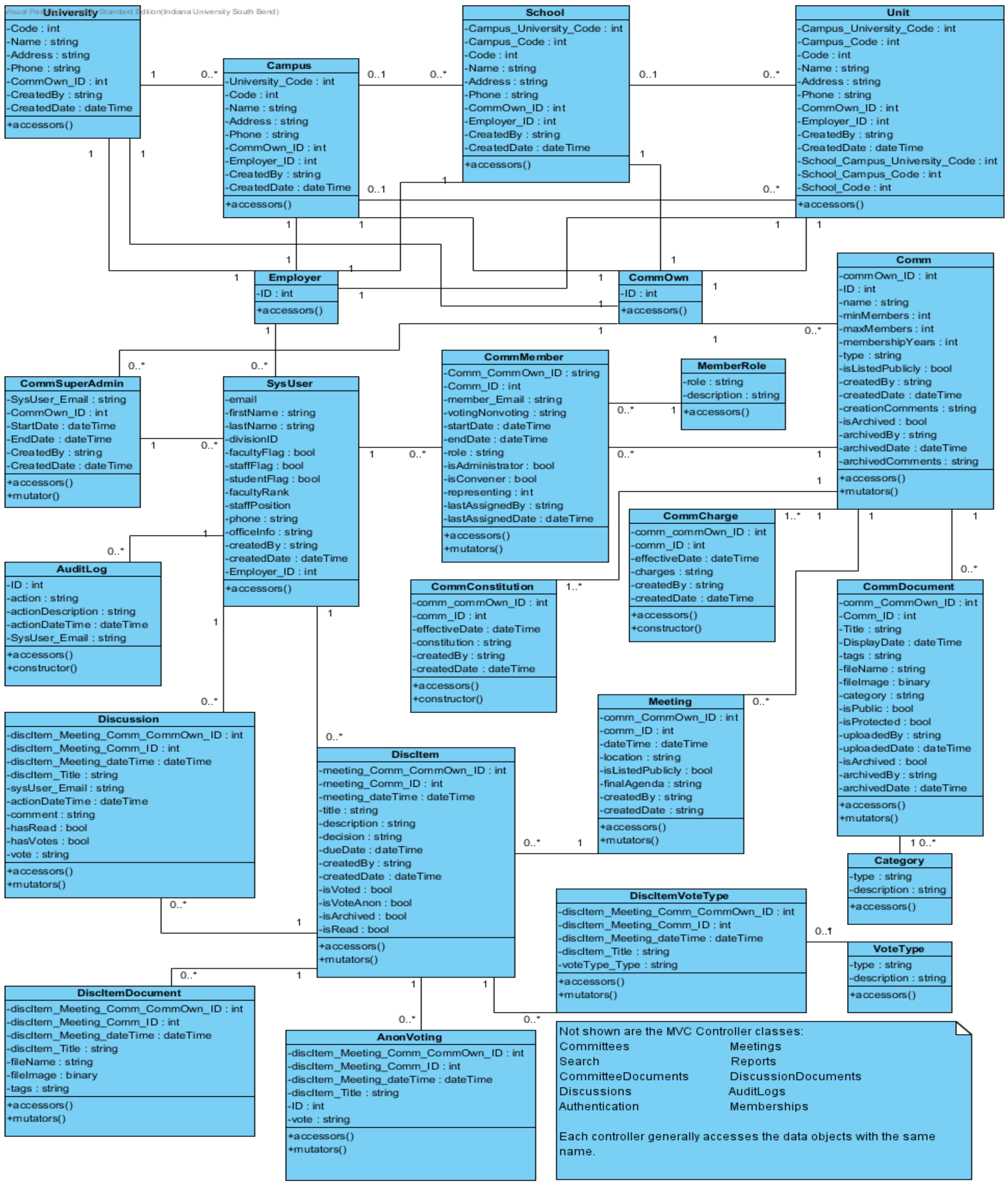

Figure 2 Data class diagram

4.0 Detail design

The software will be implemented using ASP.NET MVC 4. The pseudo code for the controllers, and a couple of helper control classes follow.

4.1 CommitteeDocuments Controller

4.1.1 Add Committee Document

```
// HTTP Get Request
// sends user to add committee document view if they have the privelege
void add()
{
    // check if user is admin of this committee
    if(User not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }

    // display add page with fields for each document attribute
    send to committee documents add view;
}

// HTTP Post Request
// adds a committee document to the database
void add(CommDocument committeeDocument)
{
    // check if user is admin of this committee
    if(user not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }

    if(postedData.isValid)
    {
        DataBase.CommDocument.Add(committeeDocument);
        DataBase.SaveChanges();
        return to committee view;
    }

    // display add view because data is invalid
    send to committee documents add view;
}
```

4.1.2 Modify Committee Document

```
// HTTP Get Request
// sends user to modify committee document view if they have the privelege
void edit(int commId, int commOwnId, string title)
{
    // check if user is admin of this committee
    if(User not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }

    // gets committee document to edit and checks if it exists
    CommDocument committeeDocument = DataBase.CommDocument.Find(commId, commOwnId, 
title);

    if(committeeDocument not exist)
    {
        return "Not Found";
    }

    // display edit page with fields for each document attribute that can be edited
    send to committee document to view;
}

// HTTP Post Request
// modifies a committee document and saves the changes to the database
void edit(CommDocument committeeDocument)
{
    // check if user is admin of this committee
    if(User not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }

    if(postedData.isValid) // checks if new document fields are valid
    {
        DataBase.CommDocument.Edit(committeeDocument);
        DataBase.SaveChanges();
        create new committee document modified audit log;
        send to committee view;
    }

    // display edit page because data is invalid
    send to committee documents edit view;
}
```

4.1.3 Delete Committee Document

```
// HTTP Get Request
// sends user to delete committee document confirmation view if they have the 
privelege
void delete(int commId, int commOwnId, string title)
{
    // check if user is admin of this committee
    if(User not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }

    // gets committee document to delete and checks if it exists and is private
    CommDocument committeeDocument = DataBase.CommDocument.Find(commId, commOwnId, 
title);

    if(committeeDocument not exist)
    {
        return "Not Found";
    }
    else if (committeeDocument is private)
    {
        send to delete confirmation view;    // only delete private documents
    }
    else
        return "Can not delete public document.";
}

// HTTP Post Request
// deletes a committee document from the database
void deleteConfirmed(CommDocument committeeDocument)
{
    // check if user is admin of this committee
    if(User not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }

    if(committeeDocument is private)
    {
        // only delete private documents
        DataBase.CommDocument.Remove(committeeDocument);
        DataBase.SaveChanges();     // saves changes to database
        create new committee document deleted audit log;
        send to committee view;
    }
    else
    {
        error "document is not private"
        redirect to Committee Documents view;    // document is not private
    }
}
```

4.1.4 Archive Committee Document

```
// HTTP Get Request
// sends user to archive committee document confirmation view if they have the
privelege
void archive(int commId, int commOwnId, string title)
{
    // check if user is admin of this committee
    if(User not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }

    // gets committee document to archive and checks if it exists
    CommDocument committeeDocument = DataBase.CommDocument.Find(commId, commOwnId,
title);

    if(committeeDocument not exist)
    {
        return "Not Found";
    }
    send to committee document archive confirmation view;
}

// HTTP Post Request
// archives a document in the database, once done can not be undone
void archiveConfirmed(CommDocument committeeDocument)
{
    // check if user is admin of this committee
    if(User not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }

    // marks committeeDocument as archived
    committeeDocument.IsArchived = 'Y';
    committeeDocument.ArchivedBy = SysUser.current.Email;
    committeeDocument.ArchivedDate = SystemDate;

    if(postedData.isValid) // checks if new document fields are valid
    {
        DataBase.CommDocument.Edit(committeeDocument);
        DataBase.SaveChanges();
        send to committee view;
    }
    else   // should only happen if data becomes invalid through network communication
         return "Network Error, retry";
}
```

4.1.5 View Committee Document

```
// gets committee documents information and ability for user to download the document
void details(int commId, int commOwnId, string title)
{
    // check if user is current member of this committee
    if(User not member)
    {
        error "Do not have permission";
        return to previous page;
    }

    // gets committee document to archive and checks if it exists
    CommDocument committeeDocument = DataBase.CommDocument.Find(commId, commOwnId,
title);

    if(committeeDocument not exist)
    {
        return "Not Found";
    }
    else
    {
        // displays information about the document and a link to download
        send to committee document details view;
    }
}
```

4.1.6 Download Document

```
// sends the committee document to the user if they have permission
void download(int commId, int commOwnId, string title)
{
    // check if user is current member of this committee
    if(User not member)
    {
        error "Do not have permission";
        return to previous page;
    }

    send file;
}
```

4.2 DiscItem Controller

4.2.1 Add Discussion Item

```
//HTTP Get Request
add(){
       //Check if user is actually this committee's admin
       if(User not CA)
       {
              error "Do not have permission";
              redirect to previous page;
       }

       return View();
}

//HTTP Post Request
add(Discussion discussion){
       //Check if user is actually this committee admin
       if(User not CA)
       {
              error "Do not have permission";
              redirect to previous page;
       }

       if(postedData.isValid)
       {
              Discussions.Add(discussion);
              SaveChanges();
              redirect to Details(discussion);
       }

       return View(discussion);
}
```

4.2.2 Edit Discussion Item

```
//HTTP Get Request
edit(int id)
{
       //Check if user is actually this committee admin
       if(User not CA)
       {
              error "Do not have permission";
              redirect to previous page;
       }

       Discussion discussion = Discussions.Find(id)
       if(discussion not exist)
       {
              return "Not Found";
       }

       return View(discussion)
}

//HTTP Post Request
edit(Discussion discussion)
{
       //Check if user is actually this committee admin
       if(User not CA)
       {
           error "Do not have permission";
           redirect to previous page;
       }

       if(postedData.isValid)
       {
           Discussions.Edit(discussion);
           SaveChanges();
           add audit log for editing a discussion
           redirect to Details(discussion);
       }
}
```

4.2.3 Archive Discussion Item

```
//HTTP Get Request
archive(int id)
{
       Discussion discussion = Discussions.Find(id);
       if(discussion not exist)
       {
           return "Not Found";
       }

       return View(discussion);
}

//HTTP Post Request
archive(int id)
{
       Discussion discussions = Discussions.Find(id);
       if(discussion not exist)
       {
           return "Not Found";
       }
```

```
        Discussions.isArchived = "Yes";
        SaveChanges()

        redirect to discussions page;
}
```

4.2.4 Delete Discussion Item

```
//HTTP Get Request
delete(int id)
{
        Discussion discussion = Discussions.Find(id);
        if(discussion not exist)
        {
                return "Not Found";
        }

        return View(discussion);
}

//HTTP Post Request
delete(int id)
{
        Discussion discussions = Discussions.Find(id);
        if(discussion not exist)
        {
                return "Not Found";
        }

        Discussions.Remove(discussion);
        SaveChanges()

        redirect to discussions page;
}
```

4.2.5 View Discussion Item

```
//Http Get Request
details(int id)
{
        //Check if user is actually a member of this committee
        if(User not Member)
        {
                error "Do not have permission";
                redirect to previous page;
        }

        Discussion discussion = Discussions.Find(id);
        if(discussion not exist)
        {
                return "Not Found";
        }

        count votes for discussion
        VoteTypes voteTypes = VotesTypes.Find(by discussion.id );
        Comment comments = Comments.Find(by discussion.id );
        Document documents = Documents.Find (by discussion.id );
        int hasRead = discussion.count(hasRead);
       Discussion.Votes

       return View(discussion,votes,hasRead,voteTypes,comments,documents);
}
```

4.3 Discussion Controller

4.3.1 View Discussion

```
//HTTP Get Request
// Sends user to discussion item view, if they have the privilege
discussGet(int id)
{
    //Check if user is a member of this committee
    if(User not member)
    {
        return View("Do not have permission");
    }

    Discussion discuss = Discussions.Find(id)

    if(discuss not exist)
    {
        return View("Discussion not found");
    }

    return View(Discuss);
}
```

4.3.2 Add Comment

```
//HTTP Post Request
// Allows user to add a comment to the discussion item
addCommentPost()
{
    //Check if user is a member of this committee
    if(User not member)
    {
        return View("Do not have permission");
    }

    //Check if the user entered a valid comment
    if(commentData.isValid)
    {
        //Add comment to discussion
        discuss.addComment(commentData);
        SaveChanges();
    }
    else
    {
        //Return the Get page with error message
        return discussGet();
    }

    // Return to discussion view with message saying comment was added
    return View(Discuss, "Comment added successfully");
}
```

4.3.3 Vote

```
//HTTP Post Request
// Allows user to add a vote to the discussion item, if they have the privilege
addVotePost()
{
    //Check if user is a member of this committee and has voting privilege
    if(user is not member or user is non-voting)
    {
        return View("Do not have permission");
    }

    //Check if the user already voted
    if(discuss.HasVoted)
    {
        //Return the discussGet page with message
        return discussGet();
    }

    //Check if the user entered a valid vote
    if(voteData.isValid)
    {
        //Add vote to discussion
        discuss.addVote(voteData);
        SaveChanges();
    }
    else
    {
        //Return the discussGet page with error message
        return discussGet();
    }

    // Return to discussion view with message saying vote was added
    return View(Discuss, "Vote added successfully");
}
```

4.3.4 Vote Anonymously

```
//HTTP Post Request
// Allows user to add an anonymous vote to the discussion item,
// if they have the privilege
addAnonymousVotePost()
{
    //Check if user is a member of this committee and has voting privilege
    if(user is not member or user is non-voting)
    {
        return View("Do not have permission");
    }

    //Check if the user already voted
    if(discuss.HasVoted)
    {
        //Return the discussGet page with message
        return discussGet();
    }

    //Check if the user entered a valid vote
    if(commentData.isValid)
    {
        //Add anonymous vote to discussion
        discuss.addAnonymousVote(voteData);
        SaveChanges();
    }
    else
    {
        //Return the discussGet page with error message
        return discussGet();
    }

    // Return to discussion view with message saying anonymous vote was added
    return View(Discuss, "Anonymous vote added successfully");
}
```

4.3.5 Mark as Read

```
//HTTP Post Request
// Allows a user to mark that the discussion item has been read
hasReadPost()
{
    //Check if user is a member of this committee
    if(User not member)
    {
        return View("Do not have permission");
    }

    //Check if the user already read the discussion
    if(User has read discussion)
    {
        //Return the discussGet page with message
        return discussGet();
    }
    else
    {
        //Mark that the user has read the discussion
        discuss.HasRead();
        SaveChanges();
    }

    // Return to discussion view with message saying discussion item has been read
    return View(Discuss, "You have now read this discussion");
}
```

4.4 DiscussionDocuments Controller

4.4.1 Add Discussion Document

```
// HTTP Get Request
// sends user to add discussion document view if they have the privilege
void add()
{
    // Check if user is actually this committee's CA
    if(User not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }
    // display add page with fields for each document attribute
    send to discussion documents add view;
}

// HTTP Post Request
// adds a discussion document to the database
void add(DiscItemDocument discussionDocument)
{
    // Check if user is actually this committee's CA
    if(user not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }

    if(postedData.isValid)
    {
        DataBase.DiscItemDocument.Add(discussionDocument);
        DataBase.SaveChanges();
        return to committee view;
    }

    // display add view because data is invalid
    send to discussion documents add view;
}
```

4.4.2 Make Discussion Document a Committee Document

```
// HTTP Get Request
// makes a discussion document also a committee document, if user has privilege
void makeCommitteeDocument(int meetingCommOwnID, int meetingCommID, string dateTime,
string title, string filename)
{
    // Check if user is actually this committee's CA
    if(User not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }

    // gets discussion document to make into a committee document

    DiscItemDocument discussionDocument =
DataBase.DiscItemDocument.Find(meetingCommOwnID, meetingCommID, dateTime, title,
filename);

    send to discussion to committee document add view;
}

// HTTP Post Request
void makeCommitteeDocument(CommDocument committeeDocument)
{
    // check if user is admin of this committee
    if(user not CA)
    {
        error "Do not have permission";
        redirect to previous page;
    }

    if(postedData.isValid)
    {
        DataBase.CommDocument.Add(committeeDocument);
        DataBase.SaveChanges();
        return to discussion documents view;
    }

    // display add view because data is invalid
    send to discussion to committee document add view;
}
```

4.5 Audit Logs Control class

```
// Adds audit information to the log table in the database.
```

4.5.1 Add Audit

```
// Inputs:
//        action - The action being recorded.
//        sysUser_Email - The user responsible for the action
//        (Optional) actionDescription: a description of the action made.

addAudit(string action, string sysUser_Email, string actionDescription = "")
{
    //Add audit information to log file
    AuditLog.Add(action, sysUser_Email, actionDescription);
    SaveChanges();
}
```

4.6 Meetings Controller

```
//Meetings Control Class

//Meetings controller provides access to meetings.
//Access is only granted to members of meetings.
//Access logic is provided by authentication routines.
```

4.6.1 Add Meeting

```
add()
{
    check if user is admin of this committee
    if(User not CA)
    {
        display error
        return to previous pages
    }

    return view(); //display add page with fields for each meeting attribute
}

//POST
//receive posted page from user, presumeably after filling in the form fields.
add()
{
    check if user is admin of the committe the new meeting belongs to
    if(User not CA)
    {
        display error
        return to previous pages
    } else {
        if !(model.isValid)          //check if new data is valid
            return view(model);      //send data back to view because it is invalid
        else
        {
            set created by field     //audit log creation not needed.
            set createdDate field
            add new meeting to db
            save db
            return to committee page
        }
    }
}
```

4.6.2 View Meeting

```
//lets the user view meeting details and a list of it's discussion items.
//GET
details(int Comm_CommOwn_ID, int Comm_ID, string DateTime)
{
    //check if user is current member of this meeting's committee
    if(User not member)
    {
        display error
        return to previous page
    }

    find meeting in database         //make sure meeting exists before finding it's
                                     //discussion items.
    if(meeting == null)
    {
        display error: "Meeting not found"
        return to previous page
    } else {
        list<discussions> discussions = new list<discussions>();
        discussions = all discussions belonging to this meeting
        return meeting and discussions
    }
}
```

4.6.3 Delete Meeting

```
//delete meeting, displays the meeting to the user and give the option to delete it.
//GET
delete(int Comm_CommOwn_ID, int Comm_ID, string DateTime)
{
    determine if userID admin of this committee
    if (user != CA)
    {
        display error: "You don't have permissions to delete this meeting"
        return to previous page
    }

    return view(meeting); //send meeting to view for delete confirmation.
}
```

```
//user had confirmed delete.
//POST
deleteConfirmed(int Comm_CommOwn_ID, int Comm_ID, string DateTime)
{
    determine if userID admin of this committee
    if (user is not CA)
    {
        display error: "You don't have permissions to delete this meeting"
        return to previous page
    }
    find meeting in db
    delete meeting
    save db
    add entry to audit log
    return to committee
}
```

4.6.4 Modify Meeting

```
//display meeting information to user, does not include discussion items.
//GET
edit(int Comm_CommOwn_ID, int Comm_ID, string DateTime)
{
    determine if user is admin of this committee
    if (user is not CA)
    {
        display error: "You don't have permission to edit this meeting"
        return to previous page
    }
    find meeting in db
    return view(meeting);
}

//receive updated meeting from user, validate it, save if ok,
//return it to user if not
//POST
edit(Meeting meeting)
{
    determine if user is admin of this committee
    if (user is not CA)
    {
        display error: "You can't touch this"
        return to previous page ??
    }
    if (ModelState.IsValid) //check if meeting is in a valid state
    {
        mark meeting as modified in database
        save database
        add entry to audit log
        return to meeting view
    }
    return meeting to view (because it is not valid)
}
```

4.7 Search Controller

4.7.1 Search

```
//Search Controller
//search view has textbox for to enter search text in and a radio buttons group for
//all committees or user's committes.
//Searching all committee's only returns results from public documents.

//display search page to user, if search string is empty show empty page.
//if search string exists, execute search and return list to user.
//GET
search(string searchString, int whichCommittees )
{
    if searchString == "" return to view;
    if (whichCommittees == allCommittee)
    {
        var documents = query for all accessable documents in all committees with
                        searchString in the tags
        var discussiondocuments = query for all accessable (really just public)
                                  discussion item docs with searchString in the tags.
        //these two committees maybe able to be joined with a union.
    }
    else
    {
        get current userId
        var documents = query for documents in committee of current user
        var discussiondocuments = query for discussion items in committees
        //these two committees maybe able to be joined with a union.
    }
    return documents and discussiondocuments to view
}
```

4.8 Reports Controller

```
Reports Controller

//if no report is selected return a list of reports and report names based on
//user roles.
//reports return results without respect to an individual committee or division.
//If a user is a CA of two committees, the List of discussion items that have no
//votes will return discussion items from both committees.
```

4.8.1 Display Available Reports

```
//GET
index()
{
    list<string> reports = new list<string>();
    if (user is CSA)
    {
        reports.add("List of underfilled committees");  //add report description
        reports.add("underfilled");                     //add report name
        reports.add("List of previous chairs of all committees");
        reports.add("previouschairs");
    }
    if (user is CA)
    {
        //add report description and name
        reports.add("List of discussion items that have no votes");
        reports.add("novotes");
    }
    if (user is member)
    {
        //add report description and name
        reports.add("List of discussion items that were approved");
        reports.add("approved");
    }
    return list of report names & descriptions to view
}
```

4.8.2 Run Report

```
//run selected report, first verifying permission.
//GET
report(string reportName)
{
    switch case (reportName)        //determine which report to run
    {
    case "underfilled":             //list of committees with not enough members
        check if user is an CSA
        if (user not CSA)
        {
            display error "You don't have permission to view this report"
            return to previous page
        }
        determine list of committees that
            belong to a divisions that the CSA is an admin of and
            have less members than required.
        return list to view
        break;
    case "previouschairs":          //list of previous chairs
    check if user is an CSA
        if (user not CSA)
        {
            display error "You don't have permission to view this report"
            return to previous page
        }
        determine list of users who
            were chairs or co-chairs of committees that
                are not archived and
                belong one of the CSA's division
        return list to view
        break;
    case "novotes":                 //list of discussion items without votes
        check if user is a CA
        if (user not CA)
        {
            display error "You don't have permission to view this report"
            return to previous page
        }
        determine list of discussion items that
            have no votes and
            are votable and
            were created during the CA's term as CA and
            belong to a committee the CA sits on.
        return list to view
        break;
    case "approved":                //list of approved discussion items.
        check if user is a member
        if (user not member)
        {
            display error "You don't have permission to view this report"
            return to previous page
        }
        determine list of discussion items that
            belong to a committee the member sits on and
            were created during the members term and
            are approved
        return list to view
        break;
    case default:
        display error "Report does not exists"
        return to previous page
}
```

4.9 Committee Controller

4.9.1 Add Committee

```
//HTTP Post request
// this function assumes to add a new committee
addCommittees(Required all parameters respect to all attributes in Comm table)
{
    //a new committee's ID will be assigned by the database
    create a new model of CommConstitution
    create a new model of CommCharge
    if (the current user is not CSA)
    {
        error "Do not have permission"
        redirect to previous page
    }
    else if (all required fields are not null)
    {
        Send to add confirmation
        CommitteesController inserts a tuple for new committee into database
        CommitteesController inserts a new tuple into CommConstitution table in the
database
        CommitteesController inserts a new tuple into CommCharge table in the database
        Return view
    }
    else
    {
        error "The required fields are empty"
        return to previous page
    }
}
```

4.9.2 View a Committee

```
//HTTP Get request
// this function assumes to get the information of a committee
getCommittees(int idComm)
{
    if (idComm exist in the database)
    {
        CommitteeController gets the entire Committee object respect to idComm
        return Committee object to view
    }
    else
    {
        error "The committee is not existing"
        return to previous page
    }
}
```

4.9.3 Archive Committee

```
//HTTP Post request
archiveCommittees(int idComm, bool isArchive, string ArchiveBy, date ArchiveDate,
string ArchiveComments)
{
    if (idComm is exist and current user is CSA)
    {
        Send to archive confirmation
        CommitteesController changes all attributes contents in the database
        Return to view
    }
    else
    {
        error "The committee is not exist or do not have permission"
        return to previous page
    }
}
```

4.9.4 View Committee by User/Division

```
//HTTP Get request
// this function assumes to get the list of committees respect to current user.
// the current user email is assumed to be passed into MemberEmail parameter.
getCommitteesByUser(string MemberEmail)
{
    if (user is existing)
    {
        CommitteeController gets entire Committee object respect to the MemberEmail
        return Committees objects to view
    }
    else
    {
        error "Bad request"
        return to previous page
    }
}

//HTTP Get request
// this function assumes to get the list of committees (return only id of committees
and name).
getCommitteesByDivision(int idDivision)
{
    if (if idDivision is existing)
    {
        CommitteeController gets all Committee Object respect to idDivision
        return Committees object to view
    }
    else
        error "Divsion is not exist"
}
```

4.9.5 Get Members of a Committee

```
//HTTP Get request
// this function assumes to get the list member of a committee respect to idComm
parameter.
getMembers(int idComm)
{
    if (idComm is existing)
    {
        CommitteesController gets all MemberEmail respect to idComm
        return CommMember object to view
    }
    else
    {
        error "The committee is not existing"
        return to previous page
    }
}
```

4.9.6 Edit Committee

```
//HTTP Post request
// this function assumes to modify information of a committee
modifyCommittees(int idComm and all parameters respect to all attributes in Comm
table)
{
    assign current user email to currentEmailMember
    create a new model of CommConstitution
    create a new model of CommCharge
    If (idComm is existing and current user is CSA)
    {
        send to modify comfirmation
        CommitteesController updates the record of idComm in the database
        CommitteesController inserts a new tuple into CommConstitution table in the
database
        CommitteesController inserts a new tuple into CommCharge table in the database
        addAudit(Modify, currentEmailMember, "Current committee is modified");
        return view
    {
    else
    {
        error "The Committee is not existing"
        redirect to previous page
    }
}
```

4.9.7 Delete Committee

```
//HTTP Post request
// this function assumes to delete a committee
deleteCommittees(int idComm)
{
    assign current user email to currentEmailMember
    if (idComm is existing and number of member is 0 and current user is CSA)
    {
        CommitteesController deletes the record of idComm in the database
        addAudit(Delete, currentEmailMember, "Committee is deleted");
        return view
    }
    else
    {
        error "Unable to delete, check all rules before deleting"
        return to previous page
    }
}
```

4.10 CommitteeSuperAdmin Controller

```
/*
*   Super Committee Admin Controller
*   Allows anyone who is an IT Admin to add a CommitteeSuperAdmin.
*   The CSA is able to create committee only under their particular division.
*
*/
```

4.10.1 View all CSAs

```
//Displays all the current CSAs
index(){

    if(User not ITAdmin)
    {
        return View("Do not have permission");
    }

    CommSuperAdmin commSuperAdmins = CommSuperAdmins.Find(all current by date);

    return View(commSuperAdmins);
}
```

4.10.2 View Divisions of a CSA

```
//Display all the divisions a user is a CSA of
details(string userId)
{
    if(User not ITAdmin)
    {
        return View("Do not have permission");
    }

    CommSuperAdmin commSuperAdmin = CommSuperAdmins.Find(by userId);

    return View(commSuperAdmin);
}
```

4.10.3 Add Committee Super Admin

```
//HTTP Get Request
//Allows ITAdmin to add a new CSA
addSuperAdmin()
{
    //Check if user is a member of this committee
    if(User not ITAdmin)
    {
        return View("Do not have permission");
    }

    CommOwn commOwns = CommOwn.Find(all);

    return View();
}

//HTTP Post Request
//Add the super admin to CommSuperAdmin
addSuperAdmin(postedData)
{
    //Check if user is a member of this committee
    if(User not ItAdmin)
    {
        return View("Do not have permission");
    }

    //Check if the posted data is valid
    if(postedData.isValid)
    {
        //Add user and committee super admin
        CommSuperAdmin.add(postedData);
        SaveChanges();
    }
    else
    {
        //Return the get page with error message
    error = "Could not add Committee Super Admin";
       return addSuperAdmin();
    }

    // Return to details view of the new CSA
    return detailsView();
}
```

4.10.4 Edit Committee Super Admin

```
//HTTP Get Request
//Allows ITAdmin to view details and edit the divisions
//a user is a CommitteeSuperAdmin Of
editSuperAdmin(string userId)
{
    //Check if user is a member of this committee
    if(User not ITAdmin)
    {
        return View("Do not have permission");
    }

    CommOwn superCommitteeAdmin = SuperCommitteeAdmins.Find(by userId);

    return View(superCommitteeAdmin);
}

//HTTP Post Request
//Allows user to edit the CSA
editSuperAdmin(postedData)
```

```
{
    //Check if user is a member of this committee
    if(User not ItAdmin)
    {
        return View("Do not have permission");
    }

    //Check if the postedData is valid
    if(postedData.isValid)
    {
        //Add user and committee super admin
        CommSuperAdmin.update(postedData);
        SaveChanges();

    //Add to audit logs
    AuditLogs.addAudit("Update Super Admin",
               currentUser.Email,
               postedData.User_email + " was added as admin of "
                   + postedData.CommOwn_ID + " committee.");
    }
    else
    {
        //Return the get page with error message
     error = "Could not edit Committee Super Admin";
        return addSuperAdmin();
    }

    // Return to view of the details of the recently edited CSA
    return detailsView();
}
```

4.11 Membership Controller

4.11.1 View Committee Membership

```
//HTTP Get request
// this function assumes to get Membership respect to emailMember parameter.
getMembership(string emailMember)
{
    if (emailMember is existing)
    {
        MembershipController get the entire CommMember Object
        return Membership object view
    }
    else
    {
        error "The member is not exist"
        redirect to previous page
    }
}
```

4.11.2 Edit Membership

```
//convener can only set role of members and isAdministrator flag.
//HTTP Post request
// this function assumes to modify Membership respect by emailMember parameter.
modifyMembership(string emailMember, string Role, string Voting, datetime StartDate,
datetime EndDate, string LastAssignedBy, datetime LastAssignedDate.)
```

```
{
    if (emailMember is existing and current user is Administrator)
    {
        send to modify confirmation
        MembershipController update the information of the object
        return view
    }
    else
    {
        error "The member is not exist"
        return to previous page
    }
}
```

4.11.3 Add Committee Member

```
//HTTP Post request
// this function assumes to add Membership respect to emailMember parameter.
addMembership(all attributes are passed as parameters respect to all attrbutes in
CommMember table)
{
    if (emailMember is existing and current user is CSA of committee’s division)
    {
        send to add confirmation
        MembershipController add new member
        MembershipController set role and date
        return view
    }
    else
        error "Do not have permission of member is not exist"
}
```

4.12 Authentication Controller

```
/*
*  Authentication Controller
*  Used system wide for easily authenticating users and checking
*  all of the permission tables.
*/
```

4.12.1 Login/Logout

```
login()
{
  //Follow CAS Code Modules
}

logout()
{
  //Follow CAS Code Modules
}
```

4.12.2 Authentication Level Checks

```
//Determines if either the current user or passed in userId is an ITAdmin
isItAdmin(&optional string userId)
{
 //If the user variable was not when passed in, default to current user
 if(userId == null)
 {
  SysUser user = SysUsers.Find(currentUser)
 }
 else //Otherwise use passed in userId
 {
  SysUser user = SysUsers.find(by userId)
 }

 //Return if user is an IT admin or not
 if(user.itAdmin is true)
  return true;
 else
  return false;
}

//Determines if either the current user or passed in userId is a super admin
//of the passed in committee
isCommitteeSuperAdmin(int committeeId, &optional string userId)
{
  //If the user variable was not when passed in, default to current user
  if(userId == null)
  {
    SysUser user = SysUsers.Find(currentUser)
  }
 else //Otherwise use passed in userId
 {
  SysUser user = SysUsers.Find(by userId)
 }

 //retrieve all Super admins of committee
 Comms.Find(by committeeId);

 //Return if user is a CSA or not
 if(CommSuperAdmins.Find(by user.id
      and committee.CommOwn_id)))
  return true;
 else
  return false;
}

//Determines if either the current user or passed in userId is an admin
//of the passed in committee
isCommitteeAdmin(int committeeId, &optional string userId)
{
 //If the user variable was not when passed in, default to current user
 if(userId == null)
 {
  SysUser user = SysUsers.Find(currentUser)
 }
 else //Otherwise use passed in userId
 {
  SysUser user = SysUsers.find(by userId)
 }

 //Return if user is a CA or not
 if(CommMembers.Find(user.id
      and committeeId
      and isAdministrator
      and (StartDate <= currentDate <= EndDate)))
  return true;
 else
  return false;
}

//Determines if either the current user or passed in userId is a committee convener
//of the passed in committee
isCommitteeConvener(int committeeId, &optional string userId)
{
 //If the user variable was not when passed in, default to current user
 if(userId == null)
 {
  SysUser user = SysUsers.Find(currentUser)
 }
 else //Otherwise use passed in userId
 {
  SysUser user = SysUsers.find(by userId)
 }

 //Return if user is a Convener or not
 if(CommMembers.Find(user.id
       and committeeId
       and isConvener
       and (StartDate <= currentDate <= EndDate)))
  return true;
 else
  return false;
}

//Determines if either the current user or passed in userId is a member
//of the passed in committee
isCommitteeMember(int committeeId, &optional string userId)
{
 //If the user variable was not when passed in, default to current user
 if(userId == null)
 {
  SysUser user = SysUsers.Find(currentUser)
 }
 else //Otherwise use passed in userId
 {
  SysUser user = SysUsers.find(by userId)
 }

 //Return if user is a member or not
 if(CommMembers.Find(user.id
       and committeeId
       and (StartDate <= currentDate <= EndDate)))
  return true;
 else
  return false;

}

//Determines if either the current user or passed in userId is a voting member
```

```
//of the passed in committee
isCommitteeVotingMember(int committeeId, &optional string userId)
{
 //If the user variable was not when passed in, default to current user
 if(userId == null)
 {
  SysUser user = SysUsers.Find(currentUser)
 }
 else //Otherwise use passed in userId
 {
  SysUser user = SysUsers.find(by userId)
 }

 //Return if user is a voting member or not
 if(CommMembers.Find(user.id
       and committeeId
       and voting/non-voting
       and (StartDate <= currentDate <= EndDate)))
  return true;
 else
  return false;

}
```

5.0 Software interface design

5.1 External Machine Interfaces

The application will be built to run in any modern web browser, allowing for use across desktop, mobile and tablet interfaces.

5.2 External Systems Interfaces

The IU Committee Software will be utilizing the IU Central Authentication Service for integrating IU user accounts into the system. This provides an intuitive and known login mechanism for the users of the system, preventing need to create and manage a yet another set of user login information.

5.3 Human Interfaces

The human interface of the IU Committee system will be designed to be as user friendly as possible. The user will be provided with a menu to navigate through the system. Each user’s menu will be based on the access level the user has. The public user role menu will have very minimal access to the system, while the super committee administrator will have a menu that has a great amount of control over the IU Committee system.

Following are several mockups of the user interface:

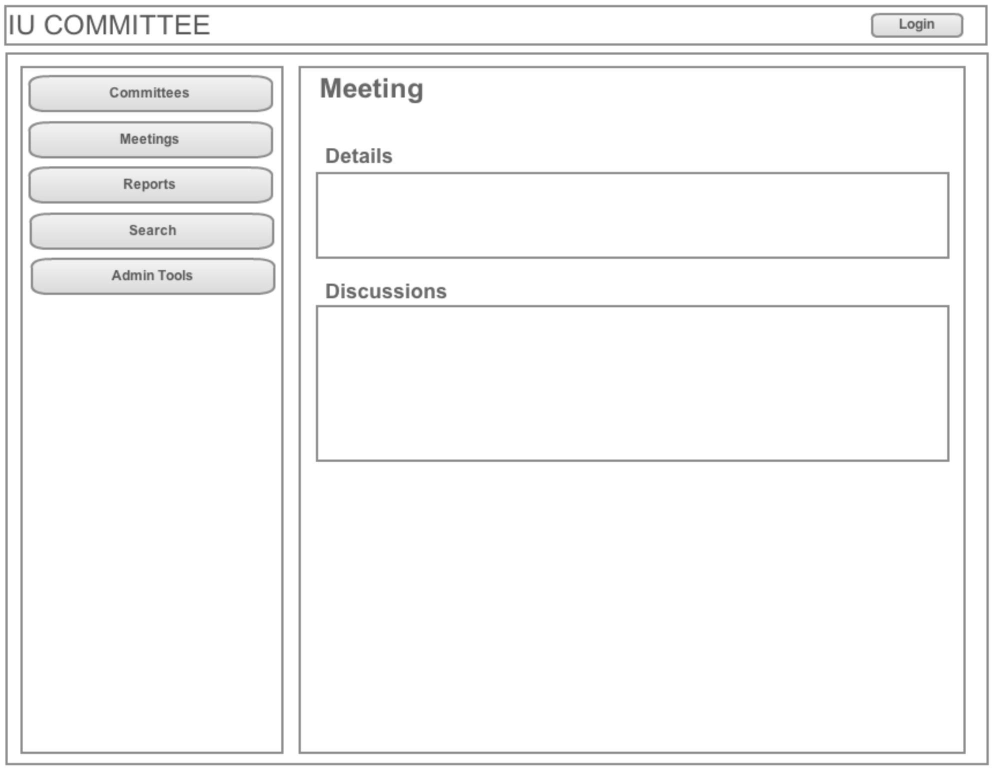

Figure 3 Meeting UI

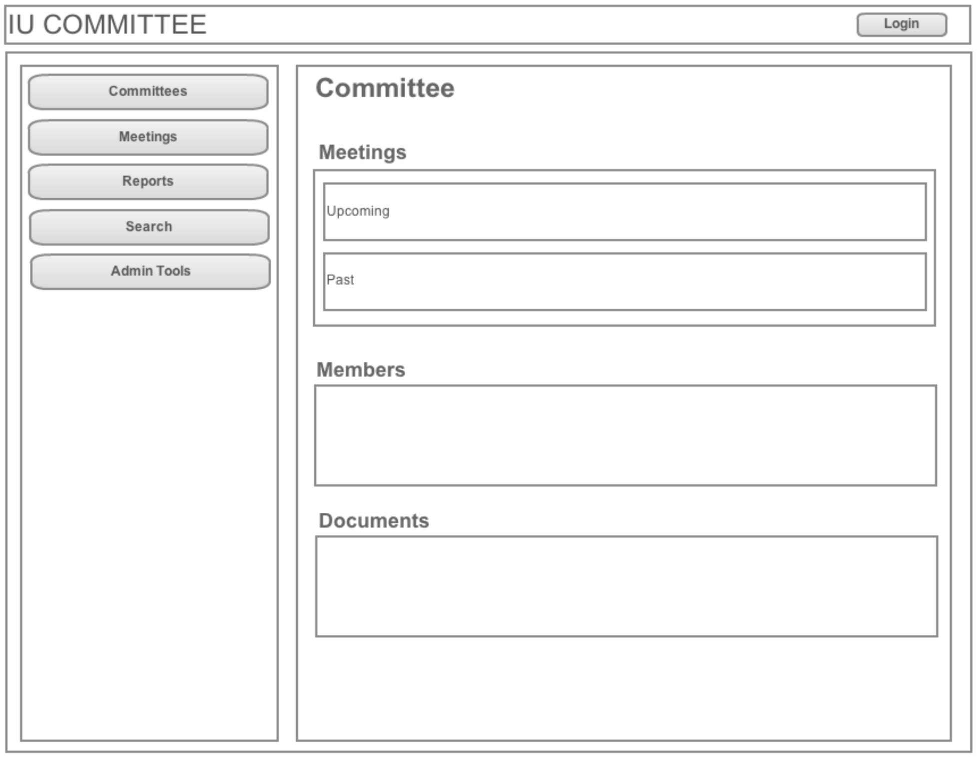

Figure 4 Committee UI

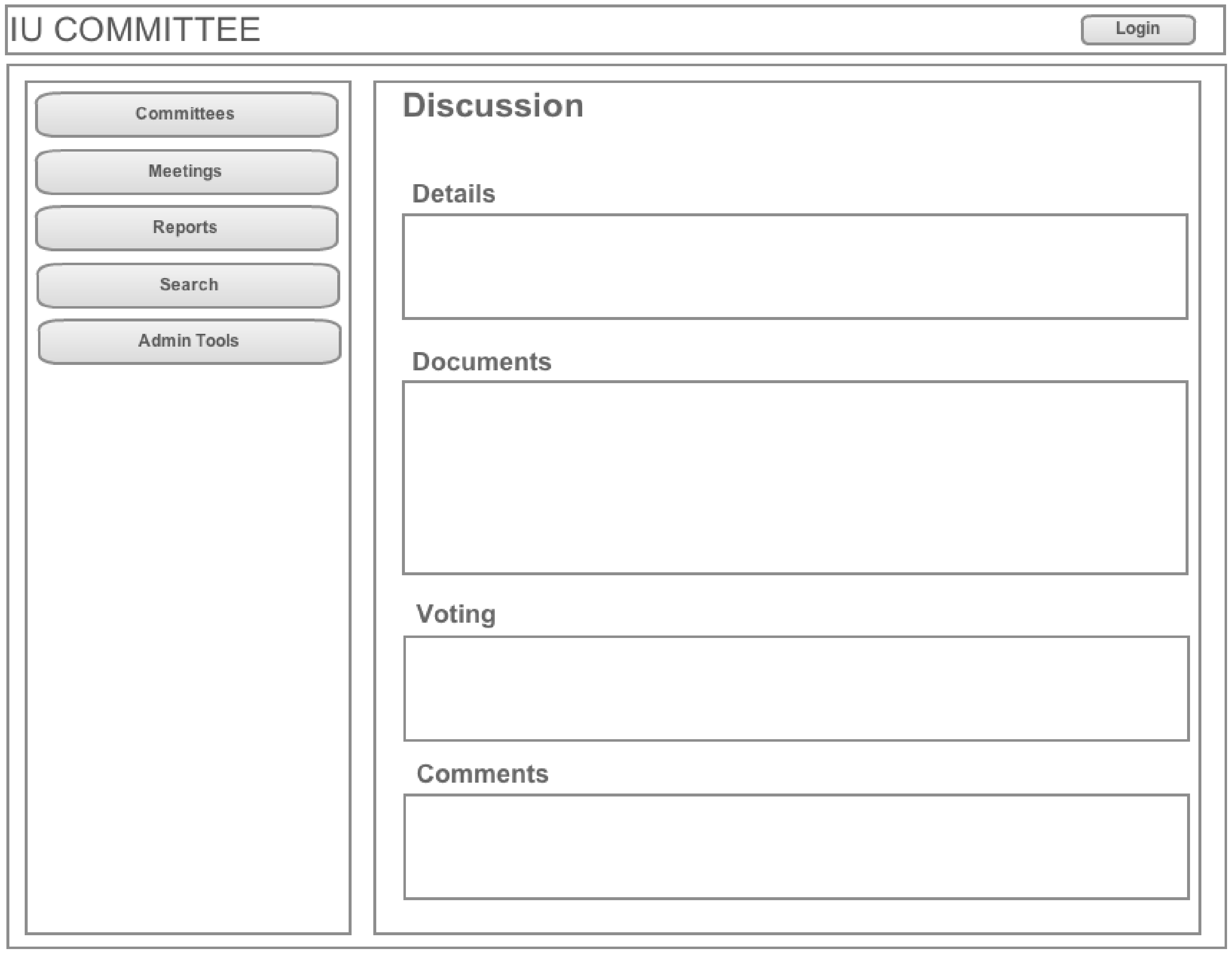

Figure 5 Discussion UI

6.0 Architecture design

6.1 Architecture description

IU Committee will be implemented using four tiers architecture. The four tiers are the web browser, the web server, the web application, and the database server. Requests are initiated by the user's web browser and are processed by the web server. The web server communicates with the web application sending it the HTML postbacks and get requests. The web application processes the html requests and accesses the database.

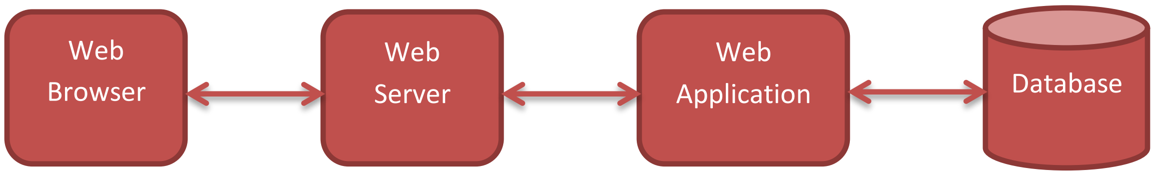

Figure 6 Architecture diagram

The Application will be written using ASP.NET MVC 4. The Model View Controller framework seperates the application into three different subsystems. The models represent the database relations and provide access to the database's data. The views contain logic to present the data in models to the web browsers and return data input by the user to the controller classes. The controller class is responsible for receiving processing data and transferring it from the user (via the views) to the database (via the models).

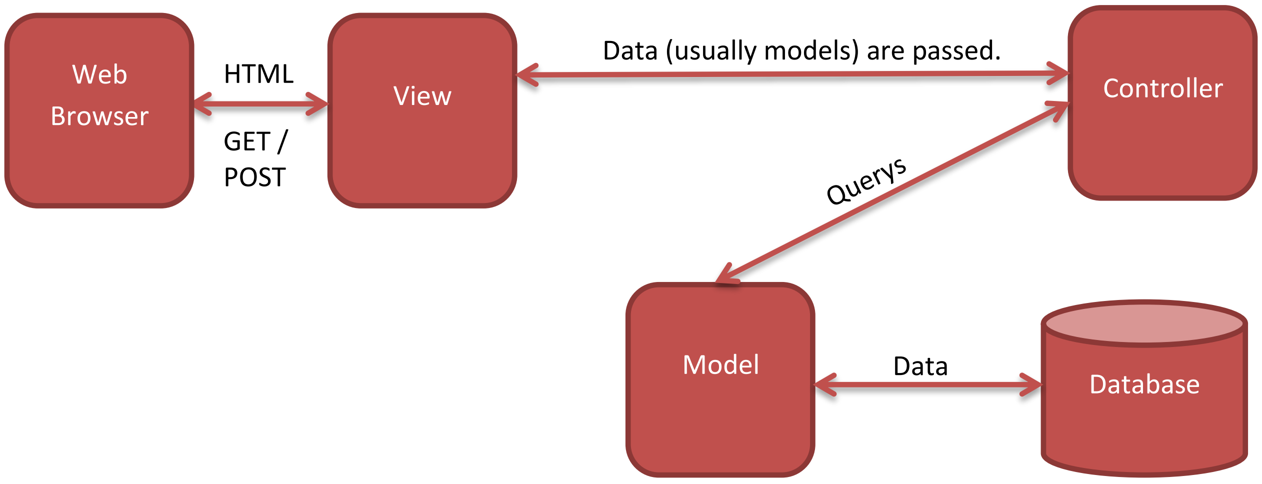

Figure 7 MVC diagram

7.0 Appendices

Supplementary information