

# **Cornell Peer Group Finder**

## Requirements Analysis and Preliminary Design

---

### **Design Team:**

Abrams, Gabriel  
Huang, Andy  
Ruengsakulrach, Natchaphon  
Wei, Xing  
Zhang, Jing  
Zhang, Xiangyu  
Zhong, Peimin  
Zhou, Tao

**October 2015**

# Overview

## Purpose and Scope

The design team will outline and create a Peer Learning Group Finder application for Mann Library visitors and staff members. The purpose of this responsive web application is to encourage group learning and team collaboration among students in Mann library. It will also provide librarians with useful room usage and meeting statistics for future library service improvements.

The system will consist of four primary functional components:

1. Admin Interface (Statistics View) allows admins to view useful room usage and meeting statistics through an admin interface.
2. Meeting Creation allows a user to create a meeting by providing meeting's title, location, time, type, and contact.
3. Meeting Editing allows user(s) to cancel the meeting or edit all basic and advanced information of the meeting.
4. Meeting Search allows users to search for meetings. The team will provide API and system documentations, which include system requirement, database ER diagram, a list of available functionalities, and RESTful API specification sheet. The client can extend the software system in the future using the API and documentations.

## Objectives and Goals

The main objective of this project is to help individuals in the Mann library to advertise their meetings or to help them to discover other meetings happening currently or in the future. The second objective is to provide the library staff with an administration interface to generate reports and manage meetings.

The project will be a success if the team is able to create an intuitive responsive web application that efficiently, securely, and intuitively implements all four of the primary functional components listed above.

# Project Requirements

To our understanding, the project requirements may be broken down into the following categories: functional requirements, usability requirements, and non-functional requirements.

## Functional Requirements

This section of the requirements outline will account for functional requirements including data management, user interface, and key functionality requirements.

As previously mentioned, the system will be built upon the understanding that there will be four key functional components: Admin Interface, Meeting Creation, Meeting Edit/Cancel, Meeting View/Search. These functions must be implemented in an intuitive, responsive, and bug free manner. To test the functionality of these components, we will run all of these functions through three rounds of usability testing, which we outline further in a later section of this document.

It should be carefully noted that usability testing forms the backbone of our development process. And therefore, we will take full advantage of the benefits of the iterative refinement method while implementing these primary functional components.

Additionally, we are carefully designing a RESTful API and documentation that meets the requirements listed below. In the detailed summary of requirements below, we have listed the requirements that we are presenting at the usability testing session. If a feature is not listed below, it will be implemented in a later iteration cycle. These features below will be tested and put up for review and modification.

### 1. Meeting Creation:

*\* denotes a required field*

**Title\*** A text entry field containing the title of the meeting

**Location\*** A popup location selection tool. We will use a hierarchical definition system as follows: (floor → room)

**Time\*** A date and time selection pane for selecting a range of time for the meeting

**Meetingtype\*** A dropdown selection field to select a meeting type (project meeting, etc.)

**Contact\*** A text entry field containing the private contact email of the creator

**Public:** User chooses whether the meeting is open to the public

**Description/Details:** A text area for giving additional meeting details. Default is none.

Note that, after request from the client, the following features have been removed:

**TeamMembers** A text field for entering team member emails. Default is none. (This field is tentative)

**EditLock** A checkbox restricting edit to the creator of the meeting. If selected, a password will be assigned to the meeting. Default is unlocked.

Functional Features that have been revised since the previous milestone:

**Reserve-only** If a meeting creator attempts to schedule a meeting in a room and time slot that is already taken in one of the library's previously created and populated meeting scheduling systems, the creator will be prompted with information about the conflicting meeting.

### 2. Meeting Edit:

*Credential Checking System:*

**Authentication** All meetings will require a password for editing, unless the edit request is sent by an admin user.

**PasswordReset** The system will send a password reset email with a randomly generated password to the creator email.

*Editing Options:*

**AvailableFields** All meeting fields will be editable except the meeting creator contact field. The meeting creator contact field will not be shown due to the fact that it is private information.

**MeetingCancellation** Remove the meeting from the calendar

### 3. Meeting Search:

*Search Fields:*

**AutomaticallySearched:** The search field will automatically search meeting title and description.

**MeetingType** Meeting type filter will show only meetings of a certain type

**Time** Allows specification of date and time range to search

**Location** Allows hierarchical searching of locations (allows searching an entire floor or a single room)

**4. Meeting List:** function to view a list of meetings in summary form. This will be used on the front page and as a way to show results of a search query.

*Visible Data:*

**Title** Meeting title

**Location** Shows floor and room (if selected)

**Time** Time range of meeting

*Related API Functionality:*

**currentMeetings** Returns meetings shown on the meeting list page

**searchMeetings** Returns meetings based on given search options (all search options specified in the calendar search page will be available in the API)

**getMeetingInfo** Returns all data from given meeting.

**changeMeeting** Allows editing of meeting information others Other API functionality will be available for statistics and data management

## Usability Requirements

We understand that usability is more than the program interface. Therefore, we focus on the entire process as the user experience. With this scope and perspective, we have identified many important usability requirements that we will be testing throughout our iterations of usability testing.

### 1. Intuitiveness

All implemented features should be available to all different types of users. Every user must easily and intuitively be able to perform all of the principal program features. For instance, everyone should be able to easily create a meeting without worrying about ambiguities and confusing information.

Task	Requirement
<b>Meeting Creation</b>	User should be able to navigate creation form without tutorials, questions, or difficulties.
<b>Meeting Search</b>	User should easily understand the use of filters and text to perform simple queries.
<b>Meeting View</b>	User should be able to open the view feature within 5s on a first attempt. User should also be able to identify all key meeting information on view page.
<b>Meeting Edit</b>	User should be able to edit a specific information field in a meeting on a first attempt. Additionally, the user should understand how to commit changes.
<b>Authentication Interface</b>	User should be able to enter a password and request a password via email.

## 2. Speed and Responsiveness

Our responsive web application is designed to be intuitive and lightweight. Thus, users must be able to perform program functions quickly without noticeable lag. For instance, a simple meeting search should return results quickly, and after creating a meeting, instantaneous confirmation should be displayed to the user.

Task	Requirement	Time
<b>Meeting Creation</b>	Creation confirmation popup	1s
<b>Meeting Creation</b>	Confirmation email	15m
<b>Meeting Search</b>	Return search results	1s
<b>Meeting View</b>	Load meeting from database	0.1s
<b>Meeting Edit</b>	Load meeting edit data	0.1s
<b>Meeting Edit</b>	Creation confirmation popup	1s
<b>Meeting Edit</b>	Confirmation email	15m
<b>Authentication Interface</b>	Authenticate	1s

## 3. Security

To ensure that the user's private information is preserved and to protect the internal systems (web application and database) from illegal or malicious inputs, the following security requirements will have to be fulfilled:

Security Feature	Description
<b>Limit size of user's text input</b>	Use text input should be limited in length to prevent the database from crashing
<b>Store non-plain-text password</b>	MD5 hash of password will be stored in the database
<b>Validate user's input</b>	Inputs such as meeting's time range and email address must be validated prior to being stored in the database. For example, meeting's end time should not be before meeting's start time and email should be in a recognizable format <u>XXX.XXX@XXX.XXX</u>
<b>Filter spam email</b>	The system shall prevent Denial of Service spam attack
<b>Keep creator email secure and private</b>	Do not show on meeting view or edit page
<b>Standard web security</b>	The system should mandate HTTPS

## Non-functional Requirements

### Compatibility

The front-end application must render the user interface properly when accessed via desktop browsers, laptop browsers and mobile device browsers. The backend program must function properly when moved into Mann Library's production servers.

### Maintainability and Extensibility

The backend program must support RESTful APIs in order to enable connection with other Mann Library applications and to allow the entire system to be enhanced after the initial release

## Current Systems

There is currently no Peer Learning Group Finder system available at Mann Library.

## Use Case Diagrams

We will have three different user roles. Each of these roles have different functions that they may take advantage of and therefore different use case diagrams.

Note that detailed use case specifications are listed in the following section: Use Case Specifications. A brief outline of the three user roles is found below:

### **Admin:**

Admin accounts are allotted by the client. These users will have the following capabilities:

- Edit any event without a password
- Change locked data fields on meetings (ex: will be able to modify a meeting's creator email)
- Remove any event from the system
- Access more detailed data from the database (ex: statistics)

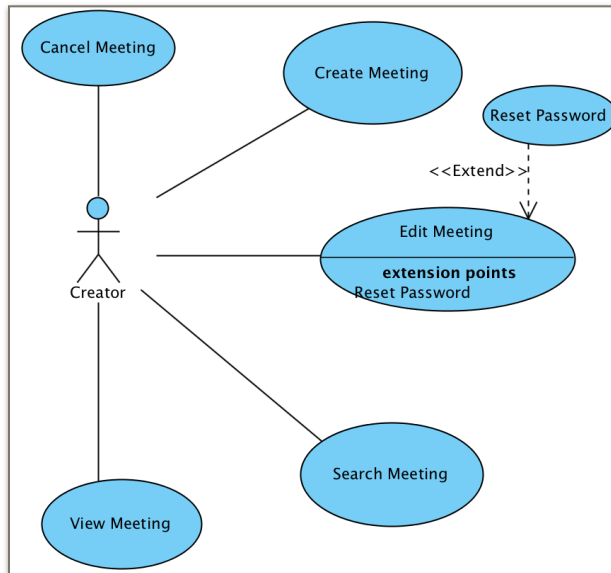
Additionally, this role will have all of the capabilities of the other two roles: Creator and Visitor.

### **Creator:**

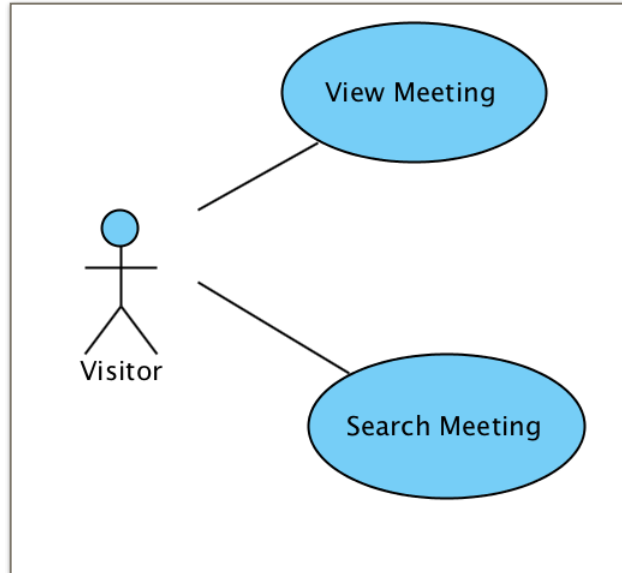
A user is promoted to Creator for the meeting(s) that they create. Thus, a visitor is promoted to Creator when they create a meeting.

- Reset the password of their meeting(s) by requesting a password reset link. Because the password will be sent to the creator email (which is unchangeable without Admin access), only creators or admins may reset the password of a meeting.

Additionally, this role will have all of the capabilities of Visitors.



**Meeting Creator Use Case Diagram**

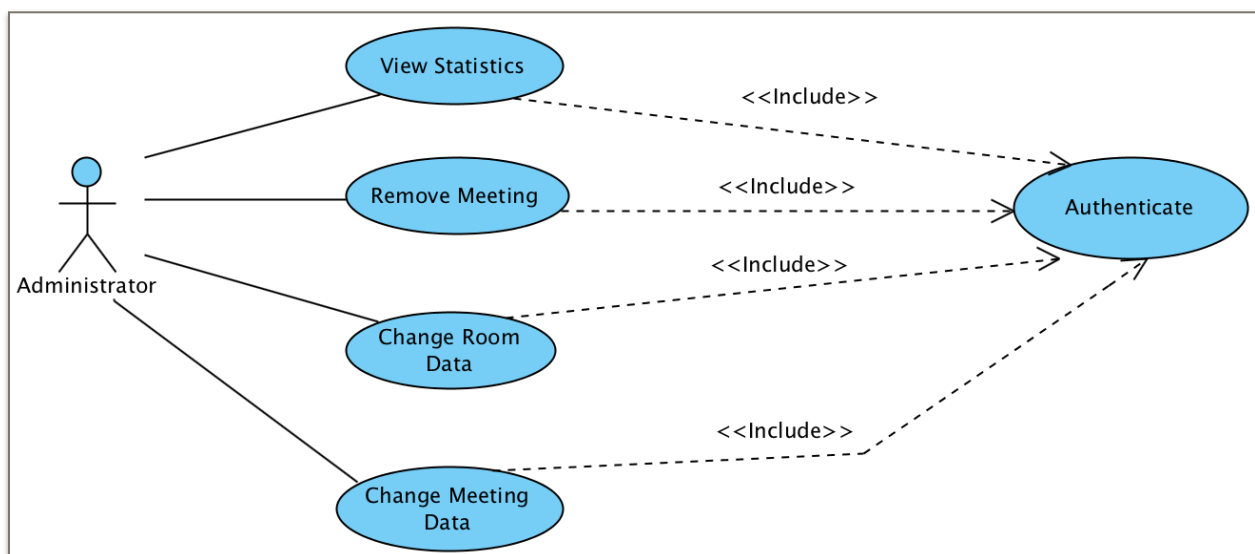


**Visitor Use Case Diagram**

### Visitor:

All other users are automatically Visitors. Also, it should be noted that Creators have Visitor privileges when viewing meetings that they did not create.

- View/search for meetings
- Create meetings (they will be promoted to Creator for that meeting)
- Edit meetings with a password
- Cancel meetings with a password



## Use Case Specifications

**Name of Use Case:** Create Meeting

**Summary:** The creator creates a meeting by filling out a meeting creation form.

**Preconditions:** The Cornell group finder user interface must be open in a browser meeting the system requirements.

**Trigger:** The creator clicks the text area right below “Cornell Meeting Finder @ Mann Library”.

**Flow of events:**

1. The creator enters all the required fields of the form and can choose to fill out the optional fields as well.
2. The creator submits the meeting creation form.
3. The server inserts the new meeting information into the database and a message indicating the successful completion of meeting creation is displayed on the browser.
4. An email confirmation is sent to the creator’s email address.

**Name of Use Case:** View Meeting

**Summary:** The creator/visitor views the list of meetings.

**Preconditions:**

1. The Cornell group finder user interface must be open in a browser meeting the system requirements.
2. There are meeting that have already been created.

**Trigger:** The creator/visitor is on the homepage.

**Flow of events:**

1. The creator/visitor tailors the meeting list by clicking the option buttons right on top of the list which allows user to decide what number of meetings to be shown within a single page during what time slots.
2. The creator/visitor clicks the highlighted link to enter the viewing section of each particular meeting.
3. The creator/visitor can choose to modify the meeting by clicking the “Modify?” button.
4. The creator/visitor can choose to share the meeting information with Facebook’s friends by clicking the Facebook share button.

**Name of Use Case:** Edit Meeting

**Summary:** The creator edits the meeting by updating the meeting creation form.

**Preconditions:**

1. The Cornell group finder user interface must be open in a browser meeting the system requirements.
2. The meeting that the creator wants to edit must be created previously.
3. For meetings that are editable, a password is required for creator authentication.

**Trigger:**

1. The creator clicks the the edit button of the navigation bar at the very top of the homepage.
2. The creator clicks the highlighted link corresponding to each particular meeting in the meeting list.

**Flow of events:**

1. If the creator uses the navigation bar, he or she will be prompted with a password to edit meeting.
2. The creator enters the password and changes any field according to the specific need.
3. The creator clicked the “Update” button to submit the new form.



4. The server updates the database with the newly-updated meeting information and a message indicating the completion of a successful update is displayed on the browser.

**Alternative path:**

1. If the creator clicks the link instead, he or she subsequently clicks the “Modify?” button and then enter the required password for editing the meeting information.
2. In the edit page, the creator changes any field according to the specific need.
3. The creator clicks the “Update” button to submit the new form.

**Name of Use Case:** Reset Password

**Summary:** The creator tries to reset the password in order to access the edit page.

**Preconditions:**

1. The Cornell group finder user interface must be open in a browser meeting the system requirements.
2. The creator must be the creator since an email regarding password reset will be sent to the creator’s email.

**Trigger:**

The creator clicks the “Forgot?” button below the password input bar.

**Flow of events:**

1. The server either randomly generates a new password and sends it directly to the creator via email or includes a link to reset password and sends it to the creator via email.
2. The creator is prompted with a message indicating that an email has been sent to his or her email address.

**Name of Use Case:** Search Meeting

**Summary:** The visitor/creator searches the information of a particular type of meetings by typing in keywords and using filters.

**Preconditions:**

1. The Cornell group finder user interface must be open in a browser meeting the system requirements.
2. The meetings that the visitor/creator wants to search for must be existent.

**Trigger:**

The visitor/creator clicks the “Search Section” button of the navigation bar at the very top of the homepage.

**Flow of events:**

1. The visitor/creator types in search keywords and uses filters to customize search experience.
2. The visitor/creator selects the number of pages to be shown within a single page.

**Name of Use Case:** Cancel Meeting

**Summary:** The creator cancels the meeting and it deletes all the related meeting information.

**Preconditions:**

1. The Cornell group finder user interface must be open in a browser meeting the system requirements.
2. The meetings that the creator wants to search for must be existent.

**Trigger:**

The creator clicks the “Contact Staff” button at the footer of the homepage.

**Flow of events:**

1. The creator enters the required password in order to pass creator authentication.
2. The creator chooses the meeting that needs to be canceled and sends a request.

3. The administrator receives the request for cancelling the meeting and removes the meeting from the list of events.
4. The message indicating the successful cancelling of the meeting is displayed on the browser.

**Name of Use Case:** Authenticate

**Summary:** The administrator passes authentication by providing the correct username and password.

**Preconditions:**

1. The Cornell group finder admin interface must be open in a browser meeting the system requirements.
2. The person who access the admin interface must be an administrator of the system.

**Trigger:**

The administrator has typed in the login information needed to access the system.

**Flow of events:**

1. The server validates the administrator's input and send a message back to the browser indicating the success or failure of the login.

**Name of Use Case:** View Statistics

**Summary:** The administrator accesses detailed data from the system's database.

**Preconditions:**

1. The Cornell group finder admin interface must be open in a browser meeting the system requirements.
2. The user must pass through the administrator authentication.

**Trigger:**

The administrator inputs the search keywords and selects different filters.

**Flow of events:**

1. The administrator chooses different options to tailor the search result in terms of different aspects of data that is of particular interest.

**Name of Use Case:** Remove Meeting

**Summary:** The administrator removes any meeting from the system.

**Preconditions:**

1. The Cornell group finder admin interface must be open in a browser meeting the system requirements.
2. The user must pass through the administrator authentication.

**Trigger:**

The administrator selects the delete option provided by each meeting that has been created.

**Flow of events:**

1. On the serve side, the entire record associated with the meeting is deleted from the database.
2. The server sends back a message to the browser indicating the success removing of the meeting.

**Name of Use Case:** Edit Meeting Data

**Summary:** The administrator is authorized to edit any meeting event even if it is locked and password protected.

**Preconditions:**

1. The Cornell group finder admin interface must be open in a browser meeting the system requirements.
2. The user must pass through the administrator authentication.

**Trigger:**

The administrator selects the highlighted link of the meeting that he or she wants to edit.

**Flow of events:**

1. The administrator enters the edit page, and is authorized to change any data fields.
2. The administrator confirms the modification by clicking the complete button.
3. The server updates the database and sends a message back to the browser indicating the success update of meeting information.

**Name of Use Case:** Change Room Data

**Summary:** The administrator is authorized to change the room data.

**Preconditions:**

The Cornell group finder admin interface must be open in a browser meeting the system requirements.

The user must pass through the administrator authentication.

**Trigger:**

The administrator selects the option of changing room data.

**Flow of events:**

1. In the room data edit page, the administrator can alter room data fields.
2. The administrator clicks the update button.
3. The server updates the database and sends a message back to the browser indicating the success update of room information.

## Scenario 1 – Meeting Creation

**Purpose:** This scenario describes the process of entering information and posting meetings

**Individual:** A meeting creator (i.e. teaching assistants, senior students, group study leader)

**Equipment:** A computing device with internet browser and network connections

**Scenario:**

1. Meeting creator turns on computing device and opens browser
2. Meeting creator types the URL to access the Cornell Group Finder homepage
3. The backend server fetches the homepage layout sends it to the browser for display
4. Meeting creator clicks on the textbox that says "Click to Create a Meeting!"
5. The textbox will expand into a form with labels and fields
6. Meeting creator enters the following information into the form: meeting title, meeting contact email, meeting location, meeting time, meeting type, meeting description and meeting privacy preference
7. After completing the form, the meeting creator clicks the "Done" button
8. The front-end application invokes the backend program to add the meeting. Upon serving the request, the backend will send the status of the transaction (i.e. success or failure) to the front-end application
9. The browser displays confirmation response or error messages to the meeting creator
10. The meeting creator sees the response and closes the Cornell Group Finder webpage or repeats steps 4 to 9 again if there is any error

## Scenario 2 - View Current Meetings

**Purpose:** This scenario describes the process of viewing a list of current meetings and the details of any particular meeting on that list

**Individual:** Any person interested in finding current meetings (i.e. students, group study members, library administrator)

**Equipment:** A computing device with internet browser and network connections

**Scenario:** (We will use "student" to represent the individuals in this scenario)

1. Student turns on the computing device and opens browser
2. Student types the URL to access the Cornell Group Finder homepage
3. The backend server fetches the homepage layout sends it to the browser for display
4. From the homepage, student selects the number of meetings to show per page and selects a time window (i.e. day, week or customized window)
5. The front-end application sends a request to the backend, receives a response with a list of meetings that fits the student's time window of interest and displays the list of meetings
6. Student clicks on a cell in the list view to select a particular meeting
7. The front-end application switches to a new page that shows all the detail information associated with that particular meeting

## Scenario 3 - Search Meetings

**Purpose:** This scenario describes the process of searching a list of meetings that matches certain conditions and finding the details of any particular meeting from the resulting subset

**Individual:** Any person interested in finding meetings (i.e. students, group study members, library administrator)

**Equipment:** A computing device with internet browser and network connections

**Scenario:** (We will use "student" to represent the individuals in this scenario)

1. Student turns on the computing device and opens browser
2. Student types the URL to access the Cornell Group Finder homepage
3. The backend server fetches the homepage layout sends it to the browser for display
4. From the homepage, student selects the search meeting tab
5. The front-end application displays the search page user interface
6. Student enters some search criteria (i.e. time, location, meeting type)
7. The front-end application sends a request to the backend, receives a response with a list of meetings that fits the search conditions and displays the list of meetings
8. Student clicks on a cell in the list view to select a particular meeting
9. The front-end application switches to a new page that shows all the detail information associated with that particular meeting

## Scenario 4 - Gathering Meeting Statistics

**Purpose:** This scenario describes the process of gathering some statistics regarding all the meeting records in the database

**Individual:** Mann Library Administrator

**Equipment:** A computing device with internet browser and network connections

**Scenario:**

1. Administrator turns on the computing device and opens browser

2. Administrator types the URL to access the Cornell Group Finder administrator interface
3. Administrator enters login credentials to authenticate with the system
4. Upon successful login, the administrator interface homepage will be displayed
5. Administrator inputs the conditions that matches the statistics he/she is looking for
6. Front-end application makes a request to the backend which triggers a file export transaction that contains all historic meeting data that fits the pre-select conditions

## Preliminary Design

### User Interface

We have distilled the interface into four sections associated with different key functions: meeting create, view/search, edit/cancel, and statistics or admin view.

### Meeting Create

This important functionality will allow a meeting creator to input relevant information and create a meeting. Our preliminary user interface is shown below:

Enter a meeting title			
We're excited to post your meeting!			
*indicates a required field			
<b>Contact Information:</b>			
*Email:	<input type="text" value="example@123.com"/>		
<b>Location:</b>			
*Choose:	<input type="button" value="Choose Location"/>		
Directions:	<input type="text" value="Go to big table near the window"/>		
<b>Details:</b>			
*When:	<input type="text" value="click to select date"/>	<input type="text" value="click to select time"/>	to <input type="text" value="click to select date"/> <input type="text" value="click to select time"/>
*Type:	<input type="text" value="— Select a Meeting Type —"/>		
*Public:	<input type="radio"/> Yes <input type="radio"/> No		
<b>Meeting Password:</b>			
You can reset this password using the email you provided above			
Password:	<input type="text" value="Something you'll remember"/>		
<b>Meeting Description:</b>			
<input type="text" value="Underwater basket weaving club meeting! We would love to have you!"/>			
Remember, the info you submit will be public.			
<input type="button" value="Done"/>			

Each of the fields above is summarized briefly below:

**Title:** A text entry field containing the title of the meeting

**Location:** A button that brings up a location chooser. We will use a hierarchical definition system as follows: (floor → room)

**Time:** A date and time selection pane for selecting a range of time for the meeting

**Type:** A dropdown selection field to select a meeting type (project meeting, etc.)

**Contact:** A text entry field containing the public contact email of the creator

**Location Details:** A text area for giving additional location details. Default is none.

**Public:** User chooses whether the meeting is open to the public

**Description/Details:** A text area for giving additional meeting details. Default is none.

There are additional functions that we have outlined with the client:

**Reserve:** only check If a creator selects a room marked "reserve-only" by the system, the user will be notified if the room has already been reserved (according to the library's scheduling API). Information about pre-scheduled meetings will be shown, however the meeting creator will still be able to create the meeting.

**Visual Interface:** Upon selecting a room in the location section, we will show an image and description of the room, provided that media is available.

## Meeting View and Search

Because the search and view functionalities of the responsive web application are very closely related (search brings the user to the view function), we will be speaking about them simultaneously.

The search function is the most important way to explore the meetings all user have created. You can type anything into the search field, and the view function will list the results we find that match your search. We will support several filters during the search, like location, time range, meeting type, and so on.

The search and view functions are combined in the following screen capture:

group

Search

Search Results:

5 to show: 5

[Machine Learning Study Group](#)-8:-31:-23 GMT-4:56:02--8:-31:-23 GMT-4:56:02@Floor 2-Individual Study Room 272  
This is the best ML study group you can find, we will help each other and enjoy the fun and joy of machine learning!! Come and join us!!!

[Software Engineering Presentation](#)-8:-31:-23 GMT-4:56:02--8:-31:-23 GMT-4:56:02@Floor 1-Current Periodicals Area

[Database Homework Study Group](#)-8:-31:-23 GMT-4:56:02--8:-31:-23 GMT-4:56:02@Floor 3-Large Group Room 370  
The due is on next Monday!!!!

After discussion with the client, we will be starting with the following features as the base of our search function:

**AutomaticallySearched:** The search field will automatically search meeting title and description

**MeetingType:** Meeting type filter will show only meetings of a certain type

**Time:** Allows specification of date and time range to search

**Location:** Allows hierarchical searching of locations (allows searching an entire floor or a single room)

**PublicMeetings:** Checkbox allowing searching for only open meetings


The list of meetings shown in the view section can be resized (a user may select to see fewer or more results) and each of the meetings will be linked to the meeting view page.


The meeting view page is a sub-functionality of the view feature. This function will allow a user to view specific details about a given meeting. Because each meeting will have a unique meeting ID, each meeting will also have an associated meeting view link. Our current design of the meeting view page is shown below:


Cornell Meeting Finder @ Mann Library


**Title: Machine Learning Study Group**


This is the best ML study group you can find, we will help each other and enjoy the fun and joy of machine learning!! Come and join us!!!

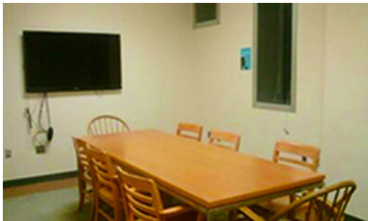
 **Location:** Mann Library Floor 2 Room 202A

 **Capacity:** 15

 **Time:** 2:20pm - 4:40pm, Oct 23rd

 **Type:** Study Group

 **Privacy:** Open to the Public



[Share](#)

[Modify?](#)

## Meeting Edit

The meeting edit feature is almost identical to the meeting create feature in the sense that all of the same input fields will be available for edit (title, description, location, time, etc). There is one key exception to note: the meeting creator email will not be editable. This is to ensure the security of the meeting edit system. Due to the fact that meeting passwords are reset via email, this field must remain secure in order to maintain the security of the meeting password.

If a meeting creator must change the email associated with a meeting, they must contact a system admin (who has the privileges to edit any information in any meeting).

The preliminary design for meeting edit is shown below:

\*Indicates a required field

Machine Learning Study Group

Contact Information:

\*Email:

Location:

Choose: Floor 2 - Individual Study Room 271

Directions:

Details:

\*When:   to

\*Type:

Advanced Options:

\*Open to the Public?

Meeting Description:

Update

## System Design

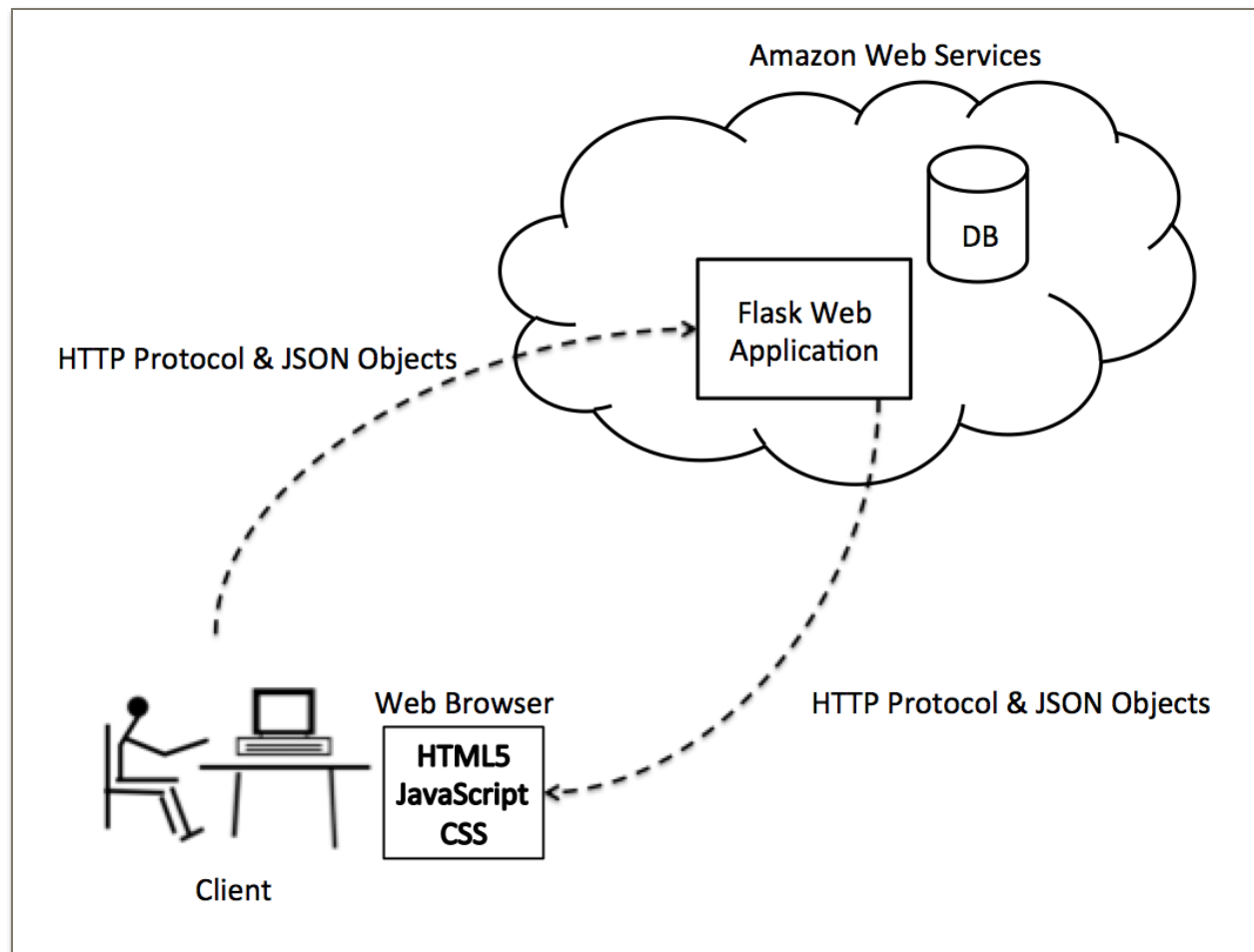
### Architectural Design

Cornell Peer Group Finder's system is based on the client/server architecture, as illustrated in the diagram on the following page.

On the client's side, the user will connect to Cornell Peer Group Finder via a web browser, which will execute HTML5 and CSS. Javascript will be used to create interactive pages, improve user experience, and verify user's inputs. The client communicates with and sends data in JSON object format to the server through an application protocol HTTP .

On the server's side, the RESTful Flask web application and database will be hosted on the testing environment, Amazon Web Services. The responsive web application performs input verification and connects to the MySQL 5 database, which stores information about meetings. Additionally, it will send HTTP requests to the Mann Library server to obtain information about room's availability.





## Database Design

Our preliminary database design is illustrated in the Entity-Relation diagram shown on the following page.

Here we present a relatively simple database design. The meeting table stores all meeting records with all necessary information. Each meeting record has and is identified by a unique **MEETING\_ID**, which is responsible for communication between front-end and back-end, and should not be seen by the users. The information that the users may have access to are the other attributes, such as **TITLE**, **DESCRIPTION**, **LOCATION\_ROOM**, **TIME\_FROM**, etc. Most of the attributes are self-explanatory by their names. Specially to support limited editing access, each record will be protected by a **PASSWORD**, which can either be automatically generated or set up by the creator of a meeting.

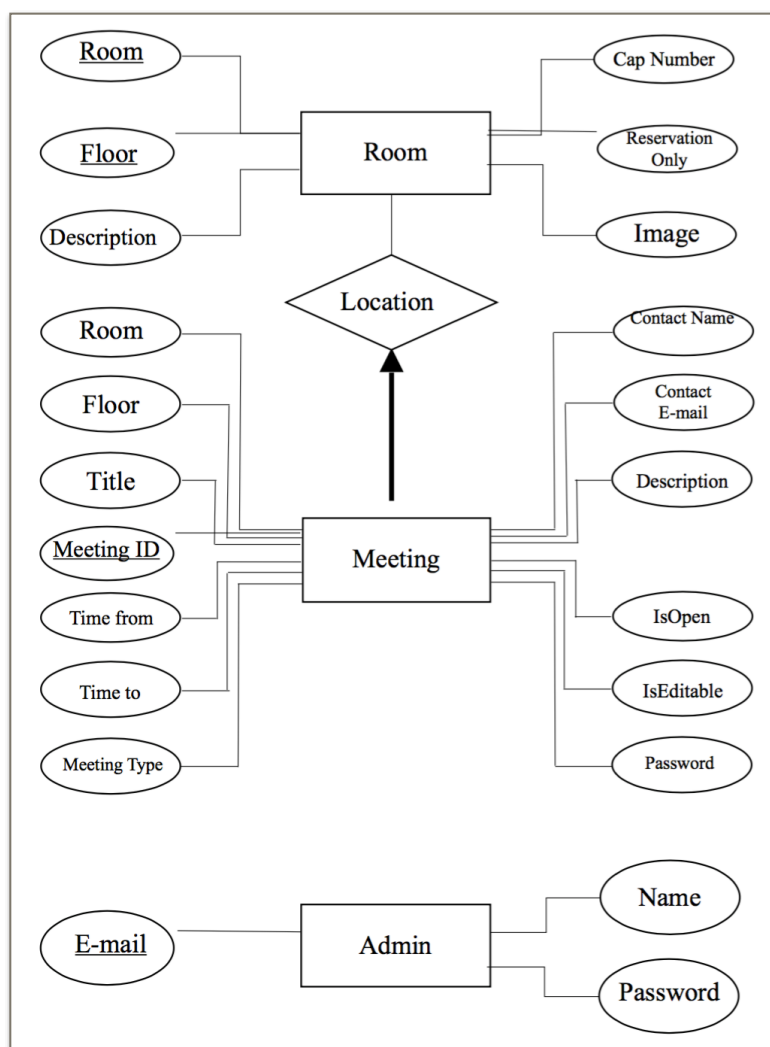
Aside from the meeting table, we have a room table to store information of rooms in the library. This table is supposedly rarely modified and inquired by the front-end to retrieve information of a specific place. All the rooms are identified by their **LOCATION\_FLOOR** and **LOCATION\_ROOM**. For each room, its **CAP\_NUM** (maximum number of people it may accommodate), **DESCRIPTION**, **IMAGE** (for the front-end to retrieve an image of the room) and whether the room is **RESERVATION\_ONLY** are stored.

We construct only one one-to-many relation between the meeting table and the room table. One meeting record must have one room, while one room may host multiple meetings provided the meeting times are different.

For administrative purposes, we have an admin table to provide authentication information for administrators. All administrators are identified by their EMAIL addresses, which they should be authenticated by along with their NAMES and PASSWORDs.

In terms of query optimization. We introduce indexing in both meeting table and room table. In the meeting table, we have clustered B+ Tree index on TIME\_FROM. This guarantees all meeting records are stored in chronological order, and allows quick queries on meeting times, which is presumably the most common query our system will receive. Additionally we have unclustered hash index on (LOCATION\_FLOOR, LOCATION\_ROOM) to support quick queries on locations. In the room table, all records are hash-indexed on (LOCATION\_FLOOR, LOCATION\_ROOM) as well for a similar reason.

A visual representation of the databases is shown below:



## Application Programming Interface (API) Design

1. This API gets all the meeting records in the database

**API Method:** HTTP GET

**API Name:** "meeting"

**Inputs:** None

**Outputs:** An array of JSON objects and each object contains all the fields of a single meeting

2. This API gets a particular meeting detail information

**API Method:** HTTP GET

**API Name:** "getMeetingDetails"

**Inputs:**

Parameter Name	Type	Required?
id	string	yes

**Outputs:** A JSON object that contains all the fields of a single meeting with the given input id

3. This API creates a meeting record and inserts it into the MySQL database

**API Method:** HTTP POST

**API Name:** "createMeeting"

**Inputs:**

Parameter Name	Type	Required?
title	string	yes
locationFloor	string	yes
locationRoom	string	yes
timeFrom	string	yes
timeTo	string	yes
meetingType	int	yes
contactName	string	yes
contactEmail	string	yes
isOpen	boolean	yes
description	string	no

isEditable	boolean	no
password	string	no

**Outputs:** A JSON object that contains the status of the request and if success, the meeting id of the newly created meeting

4. This API finds out if a room is reserved only and the URL of the room reservation site

**API Method:** HTTP GET

**API Name:** "isReservable"

**Inputs:**

Parameter Name	Type	Required?
locationFloor	string	yes
locationRoom	string	yes

**Outputs:** A JSON object that contains a boolean field to indicate whether this room can be reserved. If true, the object will also contain a field with the URL link to the reservation system

5. This API edits existing one or more existing information in a particular meeting

**API Method:** HTTP POST

**API Name:** "editMeeting"

**Inputs:**

Parameter Name	Type	Required?
id	string	yes
title	string	no
locationFloor	string	no
locationRoom	string	no
timeFrom	string	no
timeTo	string	no
meetingType	int	no
isOpen	boolean	no
description	string	no
isEditable	boolean	no

**Outputs:** A JSON object indicating the status of the request (success/failure)

6. This API searches all current and future meetings based on a set of conditions

**API Method:** HTTP GET

**API Name:** "searchMeeting"

**Inputs:**

Parameter Name	Type	Required?
title	string	no
locationFloor	string	no
locationRoom	string	no
timeFrom	string	yes (if timeTo is set)
timeTo	string	yes (if timeFrom is set)
meetingType	int	no
isOpen	boolean	no
description	string	no
contactName	string	no

**Outputs:** An array of JSON objects that satisfies the search conditions and each object contains all the fields of a single meeting

7. This API resets the meeting edit password for the creator by sending an email with a reset link

**API Method:** HTTP GET

**API Name:** "resetMeeting"

**Inputs:**

Parameter Name	Type	Required?
id	string	yes

**Outputs:** A JSON object indicating the status of the request (success/failure)

8. This API authenticates the meeting creator in order to proceed to meeting edit

**API Method:** HTTP POST

**API Name:** "verifyCreator"

**Inputs:**

Parameter Name	Type	Required?
id	string	yes
password	string	yes

**Outputs:** A JSON object that contains a boolean field to indicate a match or no match of the passwords for this authentication transaction

9. This API authenticates the library administrators in order to proceed to the administrator interface

**API Method:** HTTP POST

**API Name:** "verifyAdmin"

**Inputs:**

Parameter Name	Type	Required?
username	string	yes
password	string	yes

**Outputs:** A JSON object that contains a boolean field to indicate a match or no match of the passwords for this authentication transaction

## Schedule

The team is currently on schedule after the first iteration. Below are tasks that were completed in the first iteration:

- Front-end
  - First version HTML pages
    - Front Page (UI for create and view meeting)
    - Search Page (UI for search and view meeting)
    - Edit Page (UI for edit meeting)
    - View Page (UI for view meeting)
  - HTML pages connected to Flask Web Application
- Back-end
  - MySQL database
  - Functional Flask Web Application
  - REST API design

As discussed and agreed with the client, the team will conduct its first usability test at Mann Library on Oct 19th and 20th. The table below describes tasks provided by the

client. A list of corresponding front-end and back-end features being tested is added by the team

Task #	Task Description	Front-end features	Back-end features
1	Set up a study group meeting just for your PSYCH 1103 section somewhere on the 2nd floor for early next week.	i. Meeting creation interface ii. Confirmation message after submit	i. Email confirmation sent to the meeting creator ii. Successful update to the database
2	Notify the rest of the students in your section about the details of the meeting you just created.	i. Share button for public link ii. Public link should direct user to current meeting	i. Identify and display meeting details from the public link
3	Find an upcoming meeting (open to the public) that you'd like to join.	i. Meeting view and search interfaces ii. Meeting list on front page	i. Simple search meeting based on time range and meeting's title
4	Edit the PSYCH 1103 meeting that you created earlier and change the start time to 7pm based on the feedback from the rest of your group.	i. Authentication interface ii. Meeting edit interface	i. Authentication Handling ii. Successful update of the database

During the usability testing, in addition to monitoring the performance of the aforementioned front-end and back-end features, the team will closely observe the user's' interaction with the system. For example, the team will note how long it takes users to locate add/search/edit/view meeting. Such information will be useful for UI/UX improvement in the second iteration.

## Future Plan

The team's design phases have been designed to be driven by usability testing. Therefore, after conducting the usability test, the team will discuss with the client the team's observations and possible improvement to the front-end and back-end features. The team will then proceed according to the second iteration plan as outlined in the feasibility report. Major tasks in the second iteration plan include:

- Front-end
  - Add and improve HTML pages according to improvements that will be discussed with the client after the usability testing
  - Connect more HTML pages with back-end REST endpoints.
- Back-end
  - Modify database schema and RESTful API to accommodate with front-end changes
  - Implement more back-end functions for REST endpoints
  - Enhance web application security by adding
    - input verification
    - email spam filtering

We will proceed with our plan as outlined above, starting with usability testing on Monday, October 19th.

## **Live Demos and Other Documents**

For live demos of our current preliminary design and other important documents relating to our project, we have assembled a project hub page at the following url:

[groupfinder.gabeabrams.com](http://groupfinder.gabeabrams.com)

There, you will find links to much of the important material that we are currently working with.