

Spring 2017

## MeetingScheduler - Group 4

Alexander Martin, [alexm118@vt.edu](mailto:alexm118@vt.edu)

Erin Kocis, [erink13@vt.edu](mailto:erink13@vt.edu)

Jeffrey Shih, [jshih11@vt.edu](mailto:jshih11@vt.edu)

Jisu You, [silven@vt.edu](mailto:silven@vt.edu)

Patrick Gatewood, [pg8wood@vt.edu](mailto:pg8wood@vt.edu)

Department of Computer Science

Virginia Tech

Blacksburg, Va 24061

*Date:* May 2, 2017

*Team Number:* 4

*Instructor:* Dr. Osman Balci

# Executive Summary

Our team developed a cloud application that enhances coordination amongst a team of people by improving the process for scheduling and organizing meetings. The application utilizes user feedback to determine the availability of meeting participants in order to deduce the optimal potential meeting times for any group of individuals. The application allows individuals to create meetings, allocate times that work with their availability and schedules, swiftly invite participants, and finalize meeting plans based on invitees' responses. In order to include as much relevant information as possible, each meeting allows associated files to be uploaded and downloaded, specifies the list of meeting invitees, and also provides an interactive map focused on the meeting location.

Accounting for the fact that individuals may have incredibly busy schedules, the cloud application utilizes email functionality to notify meeting invitees when they've been invited to a meeting, when the meeting owner updates some of the meeting information, if the meeting owner cancels the meeting, and when the meeting owner finalizes the meeting time. The email functionality also alerts the meeting owner when a participant responds to the invitation. This helps no meeting go unnoticed. The application also introduces a restful web service, exposing our data and allowing for others to utilize it in their own applications.

A complete list of features implemented is in the list below.

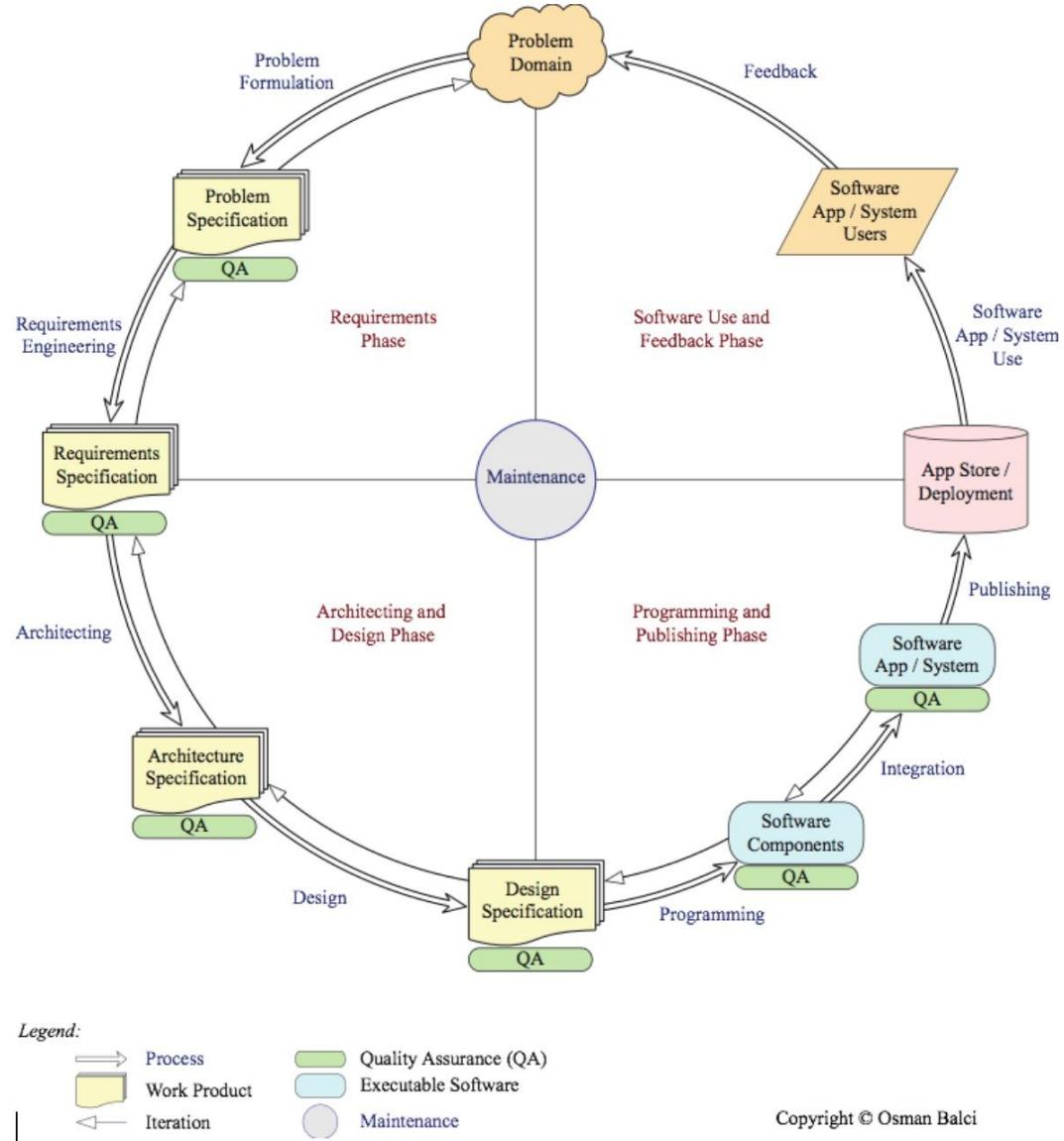
1. Activity Data and CRUD Operations
2. Email reminders for Users
3. Email reminders for Meetings
4. File Upload and Download
5. Google Maps with location pins
6. Member-Only Access
7. Photo Gallery of Users Attending a Meeting
8. Primefaces Calendar
9. Restful Webservices
10. User Account CRUD
11. MySQL Database utilizing Many to Many Relationships
12. Materialize CSS Library

# Table of Contents

<b>Executive Summary</b>	1
<b>Table of Contents</b>	2
<b>1. Software Life Cycle</b>	3
<b>2. Problem Specification</b>	4
2.1 Description of the Problem Context	4
2.2 Cloud Software Features to be Implemented	4
2.3 Database Management System	5
2.4 Complexity	5
<b>3. Requirements Specification</b>	7
3.1 Functional Requirements	7
3.2 Non Functional Requirements	8
<b>4. Architecture</b>	9
<b>5. Design Specification</b>	10
<b>6. Delivered Software Functionality</b>	18
<b>7. Step by Step Instructions for Developing New Features</b>	29
7.1 Calendar - Jeffrey Shih	29
7.2 Utilizing Materialize CSS Library Side Nav - Alex Martin	36
7.3 Utilizing Materialize CSS Parallax - Alex Martin	38
7.4 Login Filter for Members-Only Access - Patrick Gatewood	39
7.5 Collapsible “Card” Interface - Patrick Gatewood	45
7.5 Repeatable Content for Interoperability between JSF Implementations and PrimeFaces Components - Patrick Gatewood	56
<b>8. Conclusions</b>	59
<b>9. Percentages of Contribution</b>	60
<b>11. Version Control</b>	66
<b>11. Collaborative Project Management</b>	67
<b>12. Meeting Report Forms</b>	68

# 1. Software Life Cycle

A good engineer always develops software by following the life cycle shown below.



A **programmer (hacker or ad-hoc developer)** develops software by looking at the problem and directly coding in an IDE. This approach is known as the **Build-and-Fix Approach**, which must never be used!

## 2. Problem Specification

Scheduling a meeting is often more difficult than actually holding the meeting itself. The goal of our cloud application is to simplify the task of coordinating and organizing a meeting for a group of individuals with different schedules. It is often challenging to determine when everyone in your meeting is available to meet, which is why our application allows an individual, a meeting owner, to provide a meeting host which indicates personal availability timeslots. It also provides a meeting host that highlights the timeslots that participants have indicated their availability for. By utilizing a strategy of create, determine, and finalize we are able to enable users to quickly create, respond to, and finalize meeting dates and times along with other details including meeting title, description, location, and list of participants.

### 2.1 Description of the Problem Context

The goal of our cloud application is to tackle the difficulties associated with scheduling a meeting for a group of individuals. Some of the issues surround how to easily match people's schedules to account for when they're available to make finalizing a meeting time a smooth experience for the host. This application would have strong uses across companies and student organizations alike, as almost any group of people could use a tool such as this to facilitate coordinating the best time to hold a meeting. The application also allows individuals to provide relevant information and details on a meeting so that meeting planning process leaves participants prepared for the meeting when it comes time to meet.

### 2.2 Cloud Software Features to be Implemented

1. Activity Data and CRUD Operations
2. Email (Email-based notification system to notify when a user takes an action)
  - a. Email reminders for participants when meeting owner takes an action
  - b. Email reminders for meeting owners when participants take an action
3. File Upload (photo file) with CRUD operations
4. File Upload and Download
5. Google Maps with location pins
6. Member-Only Access
7. Photo Gallery of Users
8. Primefaces Calendar
9. Restful Webservices
10. User Account CRUD
11. MySQL Database utilizing Many to Many Relationships

## 12. Materialize CSS Library for additional front end features

### 2.3 Database Management System

We have chosen to use a structured MySQL database to store and retrieve information from. We have created four tables to encapsulate our data. The implemented tables include User, Meeting, MeetingFile, and MeetingUsers. We utilize a many to many relationship between users and meetings represented by the table MeetingUsers which allows us to associate all users with a meeting and all meetings with a user. We store a large amount of information in our database, including information for a user's profile, information associated with meetings such as title, description, potential times, final time, invitees, location, and host. We then use the middle table MeetingUsers, which has an entry for each meeting user association, to store a user's potential availability for a specific meeting and an indication as to whether or not the user has responded.

### 2.4 Complexity

Our project involved a lot of complexity in both the database schema and in addressing the actual problem at hand. Being tasked with a scheduling application, writing an algorithm for efficiently scheduling meetings for large groups can take on a lot of complexity. We spent a large portion of our time developing and implementing an algorithm to aid the meeting host in determining the best possible time to hold any given meeting. This algorithm pulls in user information on availability and aggregates the data across all meeting invitees to determine which dates and times are optimal for the meeting.

When looking at our database schema, we had some complexity added in structuring our data within a database in order to store all the information that our application required. One complexity was storing user's availability from their response to a meeting invitation. We needed to transform an array of java Date objects into a value to be stored in our MySQL database. We chose to use a long Varchar string, formed by transforming the Date array into a comma separated string. This design choice lead to complexity when handling the data within our java code, since we needed to perform string operations to write, update, and read the data in this field from our database.

When implementing a calendar for our users to view their meetings in a familiar, simple, chronological format, we used a primefaces component called 'schedule.' This component had a set of backing java classes called 'ScheduleModel' and 'DefaultScheduleEvent' that quickly render the calendar. DefaultScheduleEvents would be added to a ScheduleModel in an overridden loadEvents method so that when the

ScheduleModel is returned, all of the added DefaultScheduleEvents would show up on the calendar. DefaultScheduleEvents required a topic, start, and end times. In our database schema, we only included start times of meetings because we decided that end times would not have to be required as meetings rarely have a planned length. And because the start and end time parameters were Java Date objects, we had to serialize and deserialize our database String objects. This added minimal complexity by requiring us to add a default end time which was decided to be an hour after the start. Because Java Date methods are mostly deprecated and replaced by Java Calendar methods, there were some Date object manipulations required to create the end time without serializing and deserializing the start times another time.

User experience across our website was another added complexity. As we started to finalize features, we realized that what our application was lacking was an identity and to a lesser extent, a smooth user experience. We imported a third party library to help us add a more polished and refined look to our application, as well as aid in providing a smoother user experience. This library, while greatly improving the overall feel of the application, did introduce some complications. Since primefaces was our default user interface library, we needed to mold the two libraries together through cohesive CSS styling, but on multiple occasions the two libraries conflicted. This required us to spend significant time researching the problems in order to find solutions. We were forced to learn javascript and jquery in order to remedy some of the issues we were facing, but the overall improvement of the application through the use of Materialize made the increase in complexity worthwhile.

### 3. Requirements Specification

This section will discuss the Functional and Nonfunctional requirements that our cloud application will be developed.

#### 3.1 Functional Requirements

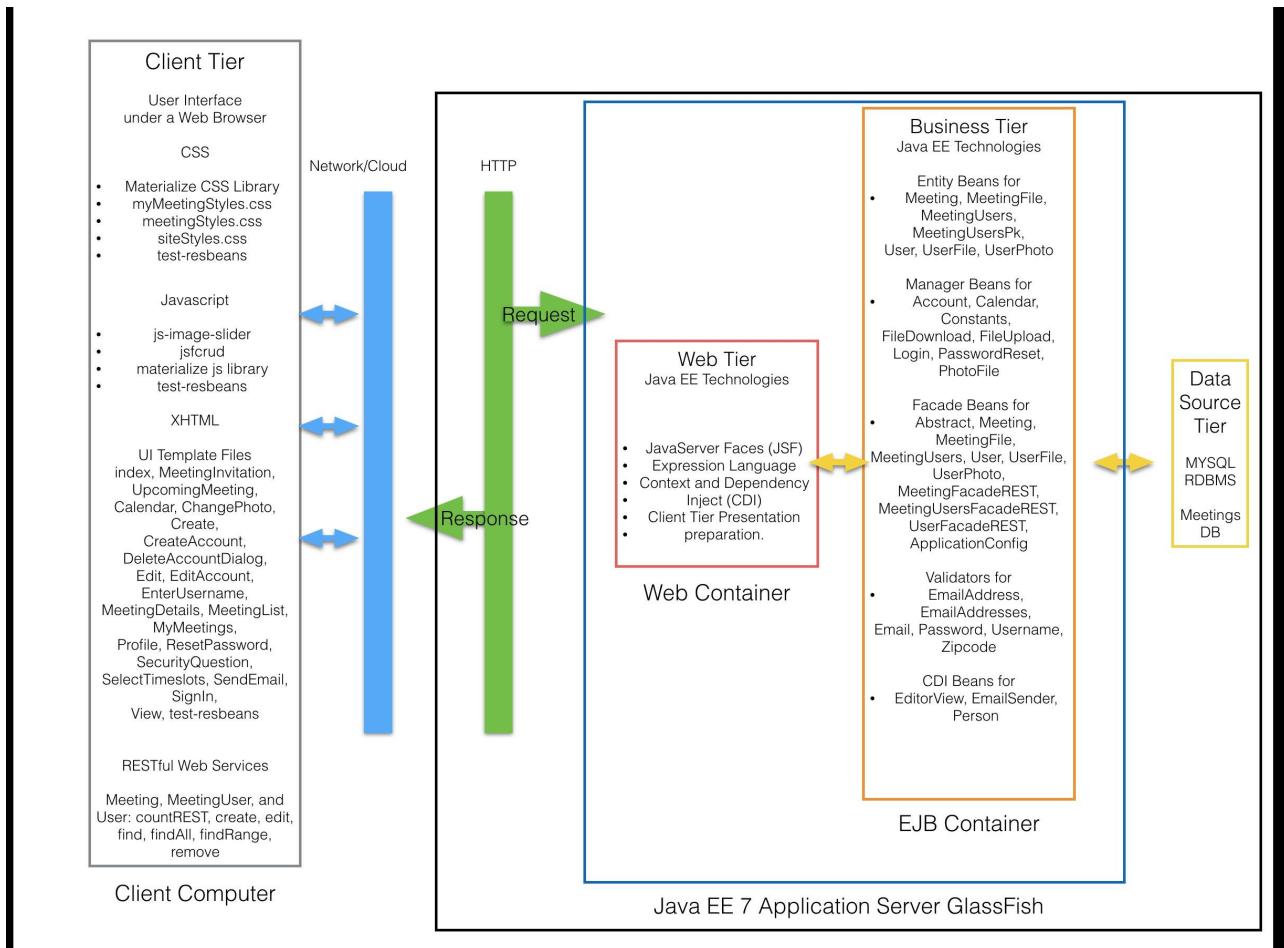
1. Our application shall be able to create a meeting, given a location, a topic, a description, a list of people to be invited, and a list of possible times in which the meeting creator is available.
2. Our application shall be able to display a list of potential times a meeting can be held to an invited user and receive a list of times a user can attend the meeting from that list of potential times.
3. Our application shall allow users to register an account for the website, given a name, address, security question, security answer, username, and password.
4. Our application shall allow users to set a profile picture to be displayed on their profile page and on their side navigation bar.
5. Our application shall allow users to upload a files associated with a meeting.
6. Our application shall allow users to view and download files associated with a meeting.
7. Our application shall allow users that created a meeting (meeting owners) to edit the meeting once it is created.
8. Our application shall allow users that created a meeting (meeting owners) to delete or cancel the meeting.
9. Our application shall email invited users to a meeting (invitees) when a meeting is created notifying them of their invitation.
10. Our application shall email invited users to a meeting (invitees) when a meeting is updated.
11. Our application shall email invited users to a meeting (invitees) when a meeting is cancelled.
12. Our application shall email invited users to a meeting (invitees) once a meeting time is finalized by the meeting owner.
13. Our application shall email the owner of a meeting when an invitee responds to the invitation, indicating their available times.
14. Our application shall give only the meeting owner the ability to check the number of responses/non-responses, to view the responses themselves, and to select a meeting time to finalize the meeting.

15. Our application shall display all finalized meetings on their respective days on a calendar.
16. Our application shall support a restful endpoint to allow for people to utilize our data for third party applications.
17. Our application shall support a meeting to be created with multiple potential time slots, by indicating a day and time during the meeting creation process. (is this a repeat of number 1?)

## 3.2 Non Functional Requirements

1. Our application shall enable users to find their next meeting in less than 3 clicks.
2. Our application shall enable users to respond to a meeting without needing to navigate through multiple pages.
3. Our application shall allow users to see meeting information for their next meeting on any given page.

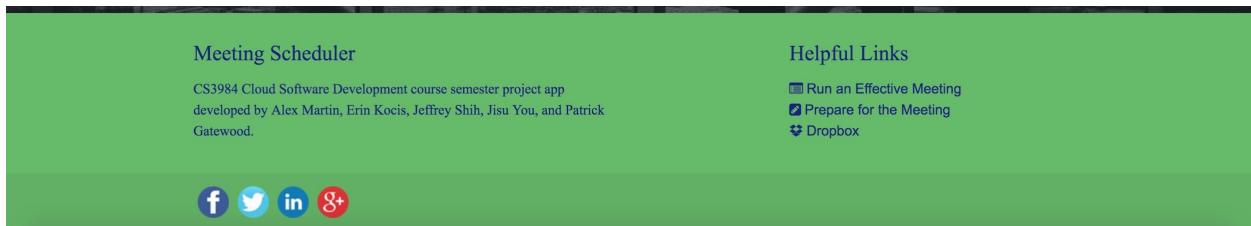
## 4. Architecture



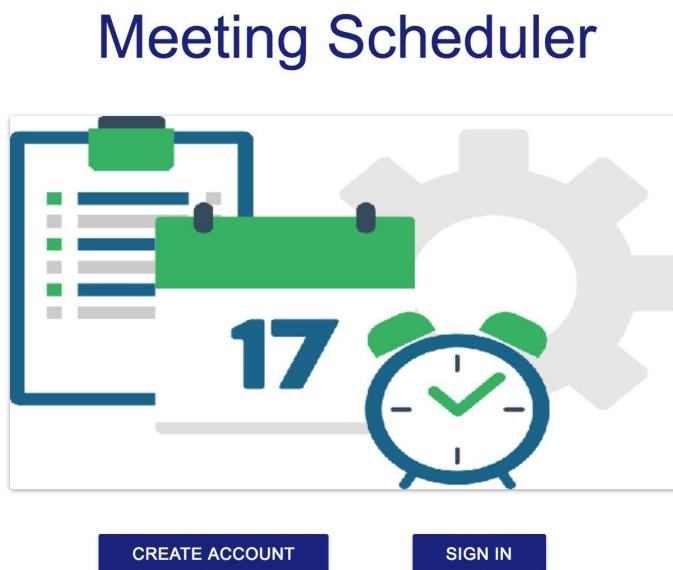
## 5. Design Specification



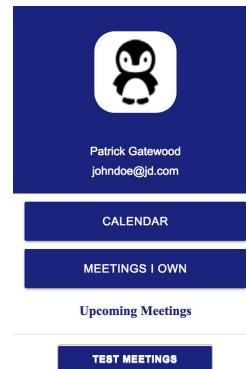
1. Header and Footer, common to all web pages.



2. Homepage



3. Side Navigation Menu



#### 4. Sign In Page

## Sign In

Username:

Password:

[SIGN IN](#) [CREATE AN ACCOUNT](#) [FORGOT PASSWORD](#) [CANCEL](#)

## 5. Create Account Page

**Create an Account**

FirstName: \*  MiddleName:  LastName: \*

Address1: \*  City: \*

Address2:  State: \*  Zipcode: \*   
VA ▾

SecurityQuestion: \*  SecurityAnswer: \*   
What street did you grow up on? ▾

Email: \*  Username: \*

Password: \*  Confirm Password: \*

**SUBMIT** **CANCEL**

## 6. Password Reset Screens

**Password Reset**

Username

**SUBMIT** **CANCEL**

## 7. Security Question

**Please Answer Your Security Question**

What is your mother's maiden name?

**SUBMIT** **CANCEL**

## 8. Password Change

### Please Enter Your New Password

Password: \*

---

Confirm Password: \*

---

**SUBMIT**   **CANCEL**

## 9. Profile Page

-  Home
-  Edit Profile
-  Change Photo
-  Change Password
-  Delete Account
-  Sign Out



### Patrick Gatewood

Location: 100 Main Street Blacksburg, VA 24060  
Email: johndoe@jd.com  
Username: p

## 10. Meeting Creation

The screenshot shows a 'Create Meeting' form and a 'Add Timeslots' modal.

**Create Meeting Form Fields:**

- Address Line 1: Osmanbalci
- Address Line 2: 100 Drillfield Dr
- City: Blacksburg
- State: VA
- Zip Code: 24060
- Topic: Finish Semester Project Paper
- Description: We will finish composing the semester project paper and prepare our project for delivery to D

**Add Timeslots Modal Fields:**

- Date: 5/5/17
- Time: 1:00
- Current Timeslots:
  - Fri May 05 12:30:00 EDT 2017
  - Fri May 05 01:00:00 EDT 2017
- Buttons:
  - ADD SELECTED TIME
  - CLEAR TIME SLOTS
  - SAVE
  - CANCEL

## 11. My Meetings Page Initial Markup Design

**Create New Meeting**

TODO Style This

- Home
- My Profile
- View Calendar
- Sign Out

◀	APRIL 2017	▶				
S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Sidebars stay in place, center shall be vertically scrollable

## Invitations

**Meeting Title**

Beginning of meeting description. Lorem ipsum blah blah trails off...

**Respond**

**Meeting Title**

Beginning of meeting description. Lorem ipsum blah blah trails off....

When can you make it?

April 8
 April 9
 April 10
 April 10

Horizontally scrollable (X stays in place at all times)

Can't go

option 1 (selected)

option 2

option 3 (indeterminate)

option 4 (disabled)

Use on/off buttons instead of radio buttons

Clicking the title expands the card to show times

Horizontally scrollable (X stays in place at all times)

Can't go

## Upcoming Meetings

**Meeting Title**

July 17 at 7:00 AM

Burrus Hall

Beginning of meeting description. Lorem ipsum blah blah trails off...

**See More**

**Meeting Title**

July 17 at 7:00 AM

Burrus Hall

Clicking location scrolls down to the map

**Participants**

Albus Dumbledore

Gandalf

Osman Balci

**Attachments**

Prereading.pdf

Image.jpg

**Location**

100 Example Ave, Suite 208

15

## 12. Meeting Response

Finish Semester Project Paper

We will finish composing the semester project paper and prepare our project for delivery to Dr. Balci's office.

When can you make it?

May 5 Sat

Times for Sat, May 5

Double click to select a time.

0:30 PM  
1:00 PM

**SUBMIT ALL SELECTED TIMES**

Can't go

## 13. Meeting View

# Semester Project Presentation!!

Owner Actions

You set the final meeting time for:  
Fri May 04 02:00:00 EDT 3917

Torgerson Hall  
Blacksburg, VA 24060

We will be giving our semester project presentation to  
Dr. Balci, Dr. Laurian Vega, and the rest of the Cloud  
Software Development class!

Fri May 04 02:00:00 EDT 3917

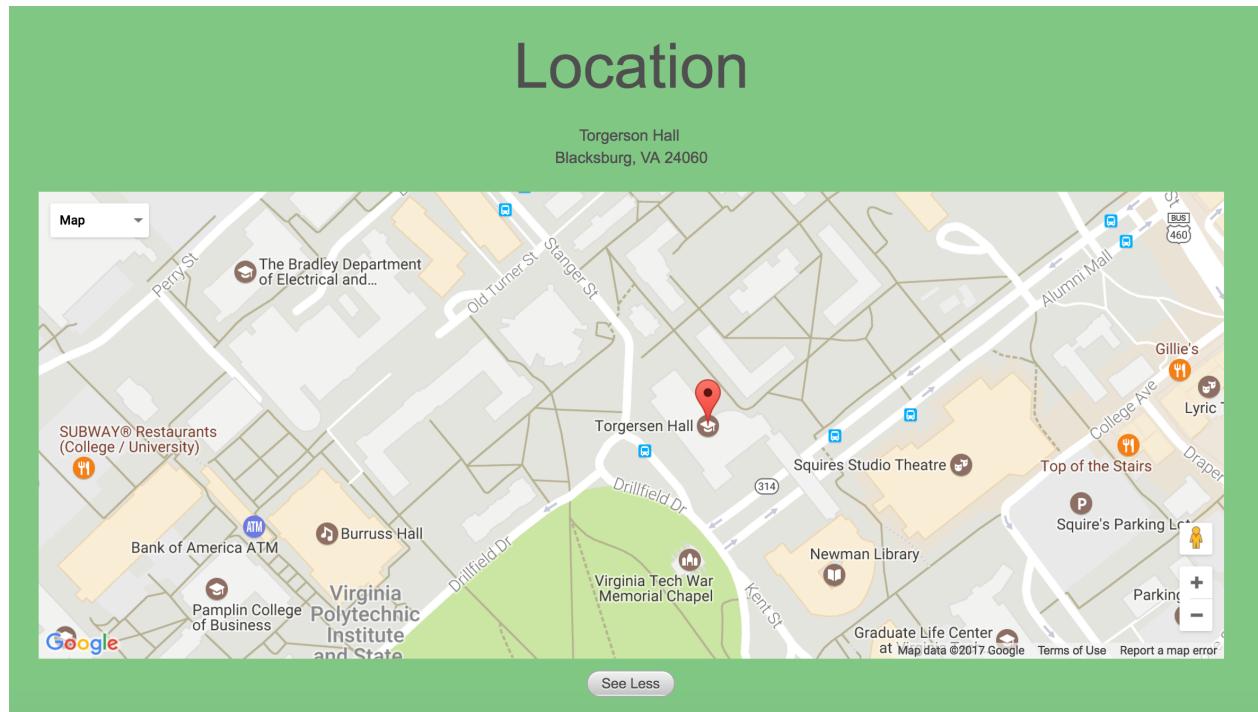
## Participants

Alex Martin  Patrick Gatewood  Osman Balci  Erin Kocis 

## 14. File Upload

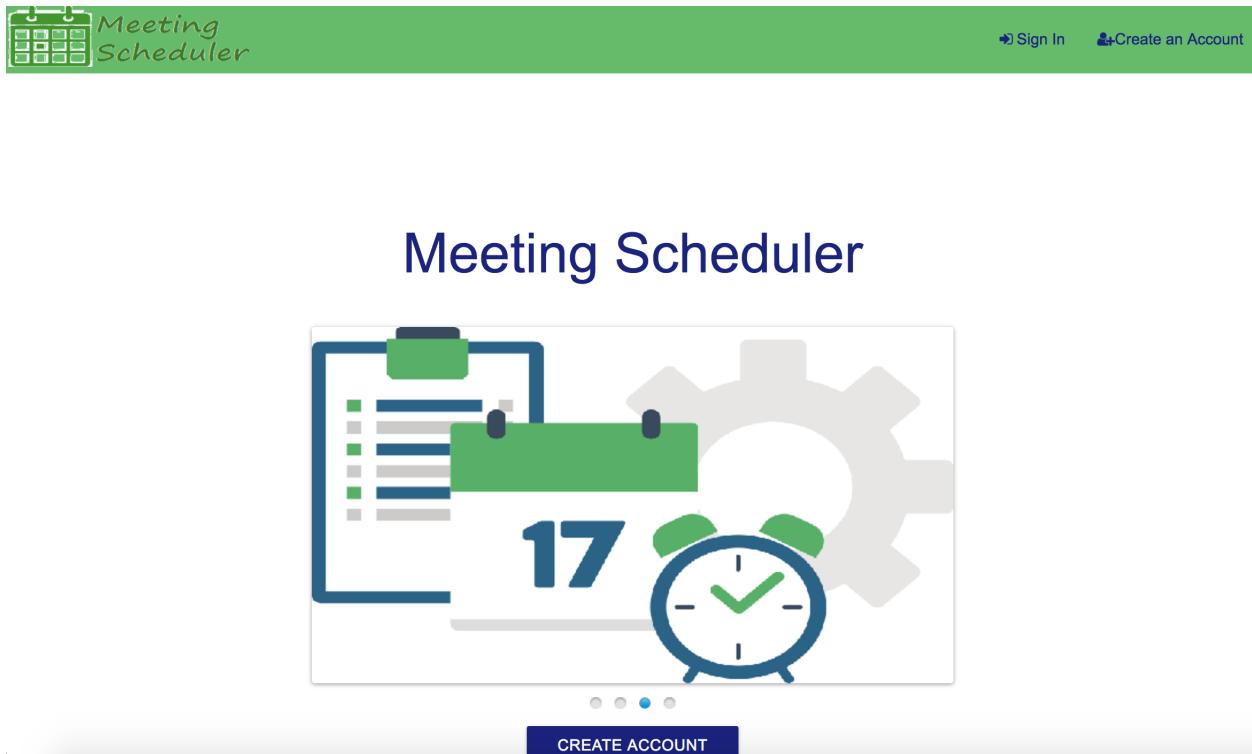


## 15. View Meeting Location - Google Maps



## 6. Delivered Software Functionality

MeetingScheduler can be accessed at <http://venus.cs.vt.edu/group>. First, the user must make an account by clicking the “Create Account” button either in the header or at the



bottom of the screen.

Account creation is identical to that found in CloudDrive, but the form has been stylized.

# Create an Account

FirstName: \*

Patrick

MiddleName:

Eugene

LastName: \*

Gatewood

Address1: \*

310 Webb St

City: \*

Blacksburg

Address2:

State: \*

Zipcode: \*

VA

24060

SecurityQuestion: \*

What is your father's middle name?

Dale

SecurityAnswer: \*

Email: \*

pg8wood@vt.edu

Username: \*

Gatewood

Password: \*

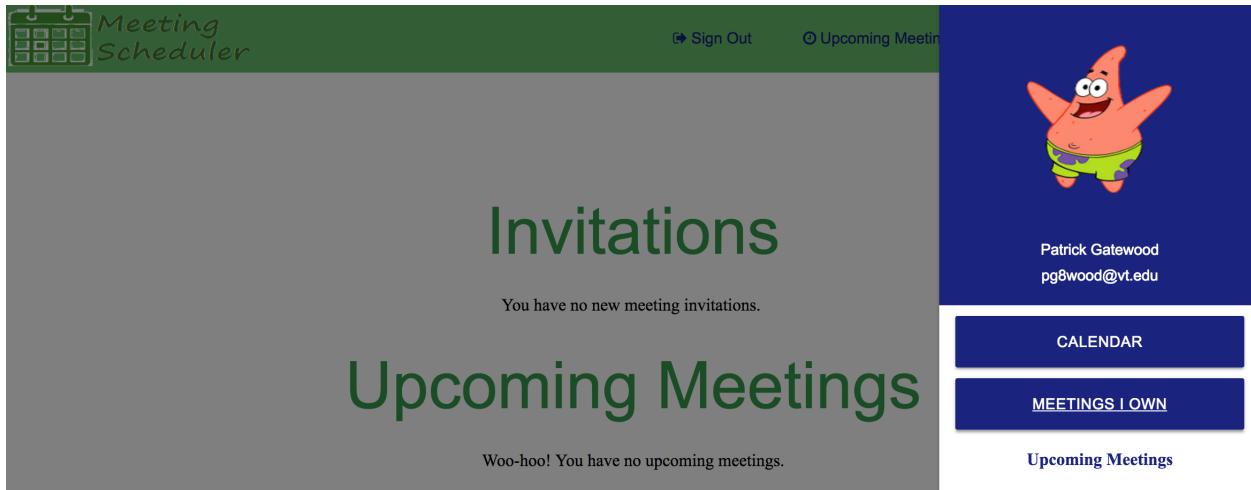
.....

Confirm Password: \*

.....

Upon first logging in, the user will be greeted with the landing page. Here, meeting invitations and upcoming meetings will be displayed to the user. Initially, the user will

have no meetings. To create a meeting, the user shall click “Upcoming Meetings” in the header, and “Meetings I Own” in the exposed sidebar.



The screenshot shows the Meeting Scheduler application interface. At the top, there is a green header bar with the "Meeting Scheduler" logo on the left, and "Sign Out" and "Upcoming Meetings" buttons on the right. The main content area is divided into two sections: "Invitations" on the left and "Upcoming Meetings" on the right. The "Invitations" section displays the message "You have no new meeting invitations." The "Upcoming Meetings" section displays the message "Woo-hoo! You have no upcoming meetings." To the right of these sections is a sidebar with a cartoon starfish icon and the user's profile information: "Patrick Gatewood" and "pg8wood@vt.edu". Below the sidebar are three buttons: "CALENDAR", "MEETINGS I OWN", and "Upcoming Meetings".

Meeting Scheduler

Sign Out Upcoming Meetings

Invitations

You have no new meeting invitations.

Upcoming Meetings

Woo-hoo! You have no upcoming meetings.

Patrick Gatewood  
pg8wood@vt.edu

CALENDAR

MEETINGS I OWN

Upcoming Meetings

This link navigates to the Meetings I Own page. Here, the user can perform basic CRUD operations on meetings. To create a meeting, the user shall click the “Create” button. This shows the modal window that allows the user to input information about the meeting. If required input is malformed, the application will display a helpful message. Note that the user must invite users via username.

**Create Meeting**

alexmartin, Osmanbalci, fakedude, fakerdude

Address Line 1 \*

225 Stanger St

Address Line 2

City \* Blackburg

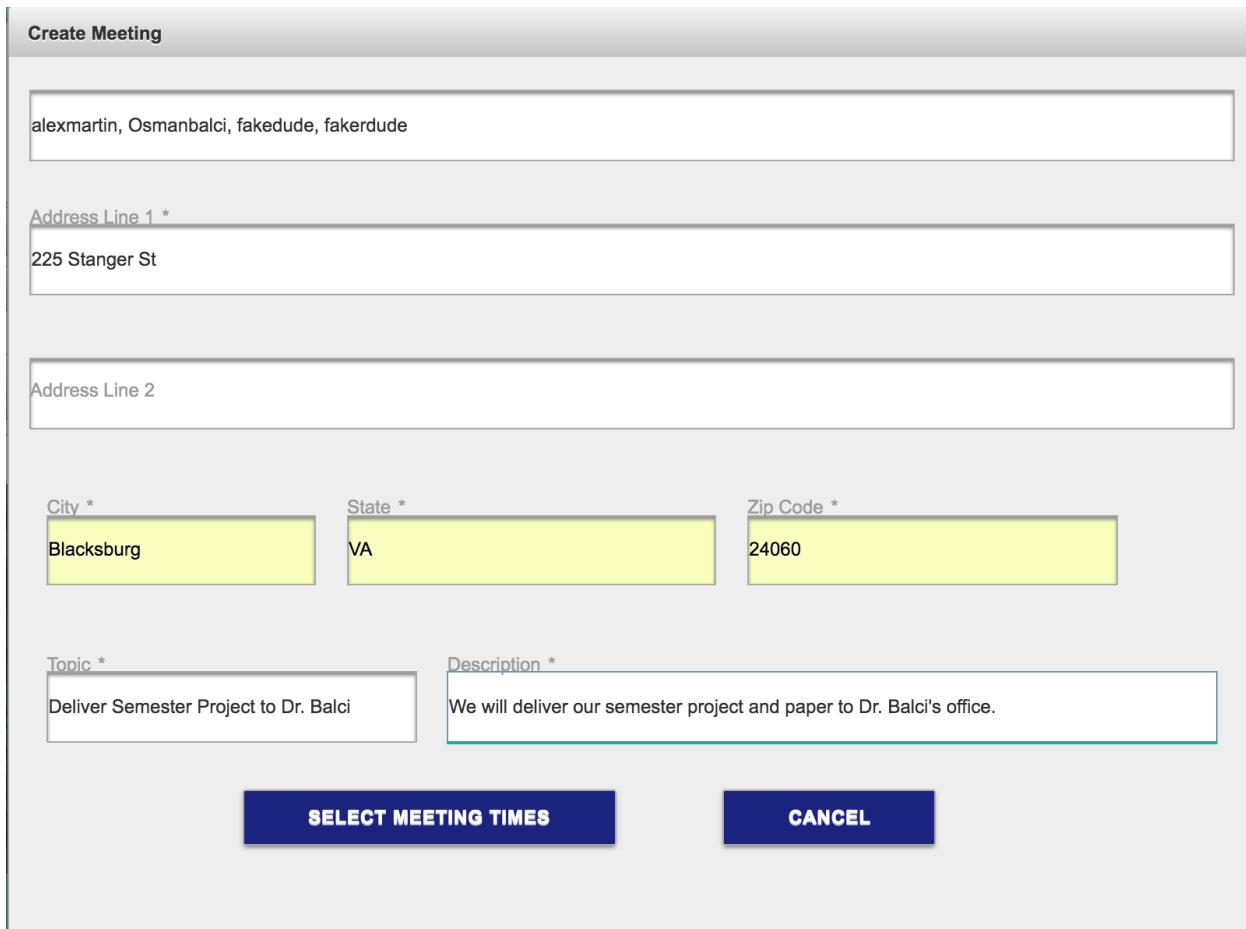
State \* VA

Zip Code \* 24060

Topic \* Deliver Semester Project to Dr. Balci

Description \* We will deliver our semester project and paper to Dr. Balci's office.

**SELECT MEETING TIMES**      **CANCEL**



After clicking “Select Meeting Times”, the user shall input the timeslots that they wish to open for the meeting. Again, if the input is malformed, a helpful growl message will display.

**Add Timeslots**

**Date**

**Time**

**ADD SELECTED TIME**

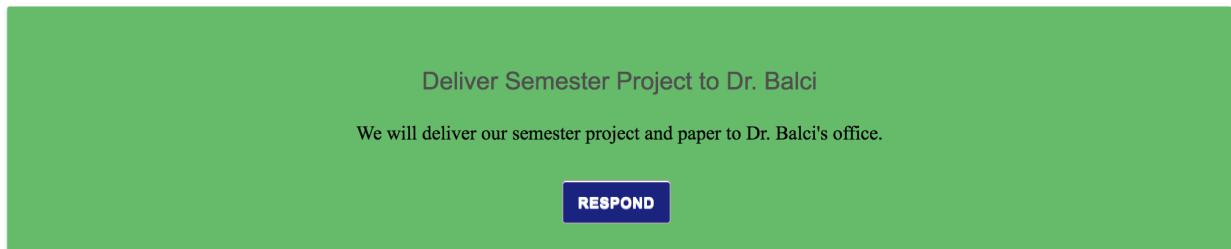
**CLEAR TIME SLOTS**

Current Timeslots
Mon May 08 11:00:00 EDT 2017
Mon May 08 11:30:00 EDT 2017

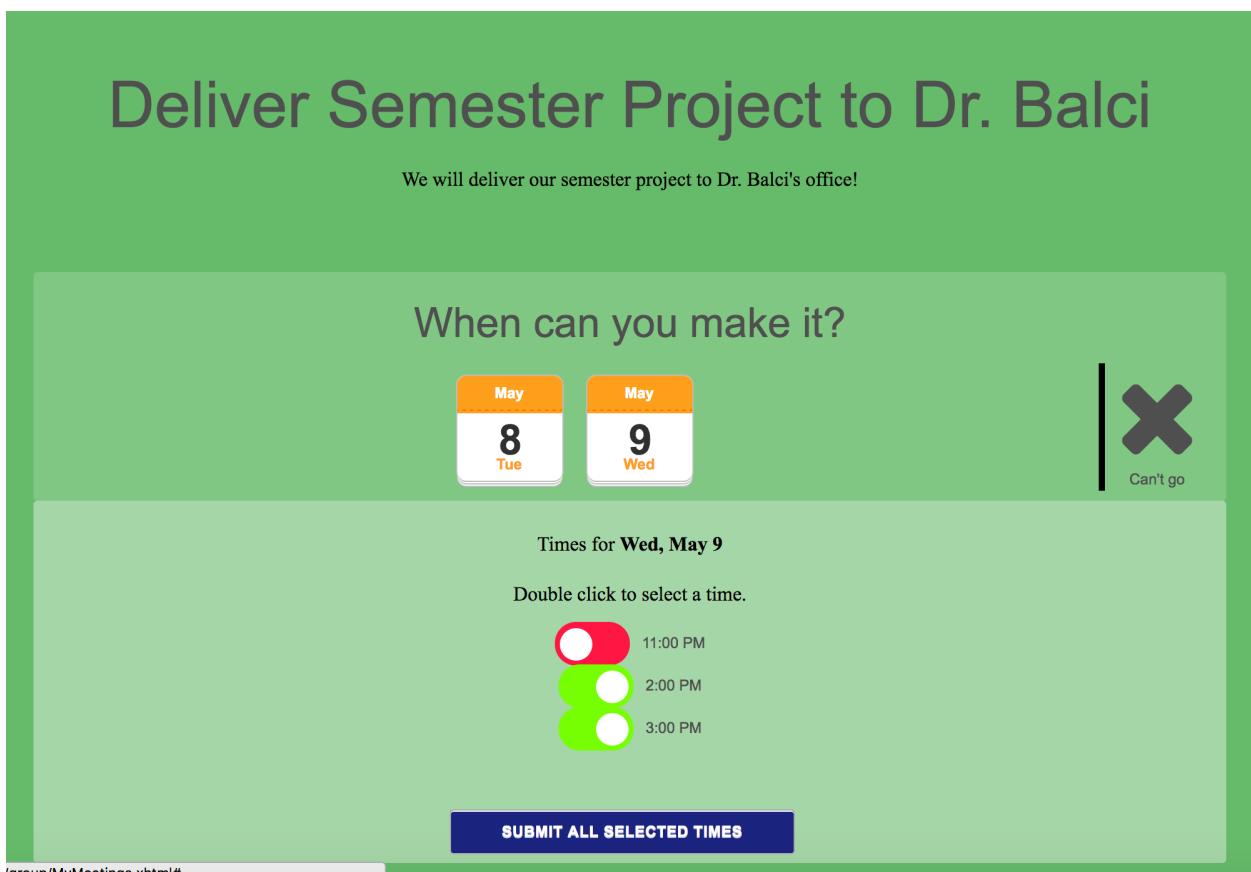
Clicking the “Save” button will display the meeting in the datatable. This will send an email to the invitees via an email sender that runs on a background thread. This sends the email without hanging the GUI thread, improving the user experience.

When an invitee logs in, the new meeting invitation will be shown under their invitations.

## Invitations

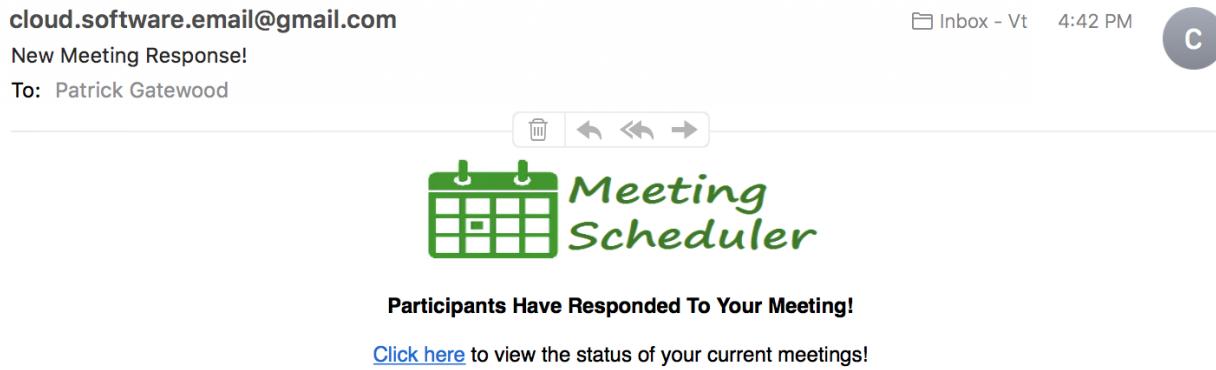


Clicking “Respond” shall expand the card and allow the user to select which times they are available.



Clicking “Submit all selected times” shall submit the response to the owner, and the owner shall receive an email that an invitee has responded to their meeting. Clicking

“Can’t go” will display a confirmation, and upon confirmation will indicate to the meeting owner that the invitee is unavailable for every time slot.



Still on the invitee's account, the invitee can now view more information about the upcoming meeting that was added to their Upcoming Meetings.

If the user is the creator (and therefore owner) of the meeting, an “Owner Actions” section shall display in the meeting preview.

The image shows a meeting preview with a green header 'Finish Semester Project Paper'. Below the header is a 'Owner Actions' bar with a '+' icon. The main content area shows the meeting details: '100 Drillfield Dr Blacksburg, VA 24060', 'We will finish composing the semester project paper and prepare our project for delivery to Dr. Balci's office.', and 'Final Time Not Yet Set'. A 'SEE MORE' button is visible. The entire preview is set against a green background.

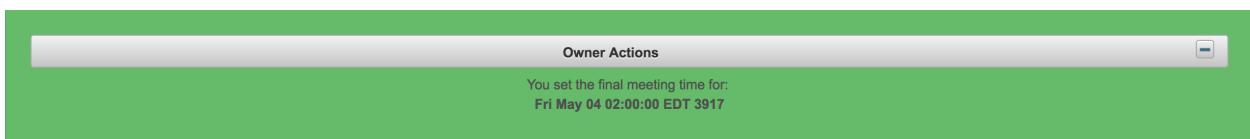
Here the user can view the times that invitees have indicated they are available.

The image shows a meeting preview with a green header 'Finish Semester Project Paper'. Below the header is a 'Owner Actions' bar with a '+' icon. The main content area shows the meeting details: '100 Drillfield Dr Blacksburg, VA 24060', 'We will finish composing the semester project paper and prepare our project for delivery to Dr. Balci's office.', and 'Final Time Not Yet Set'. A 'SEE MORE' button is visible. The entire preview is set against a green background.

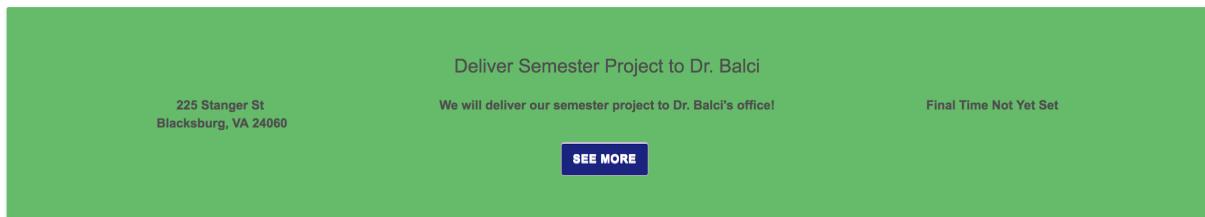
At *any time*, the meeting owner may choose to finalize the meeting time by clicking one of the timeslots.



After the meeting has been finalized, the Owner Actions section shall display the final meeting time, which cannot be changed.



## Upcoming Meetings



Similar to meeting invitations, the user shall click “See More” to reveal more information about the selected meeting. This will reveal a variety of features, which will be discussed individually below.

The user can view who has responded to the meeting invitation in the Participants section.

# Semester Project Delivery

225 Stanger St  
Blacksburg, VA 24060

We will deploy our semester project on Dr. Balci's iMac in order  
to be graded on our design.

Final Time Not Yet Set

## Participants



Patrick Gatewood    Mr. Krabs    Osman Balci

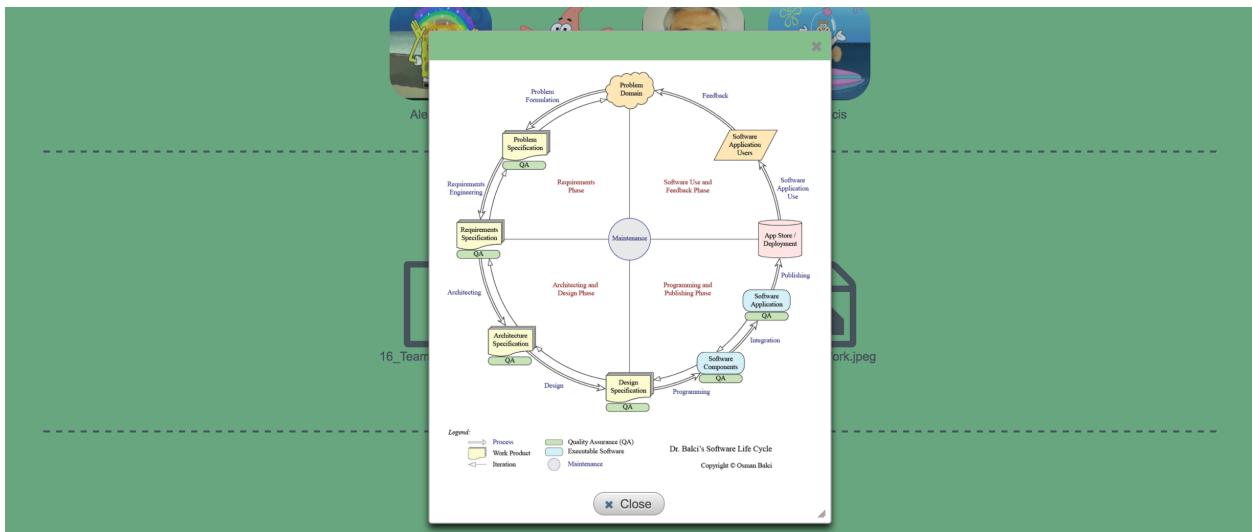
Scrolling down, the user can view and/or upload files in the Attachments section.

## Attachments

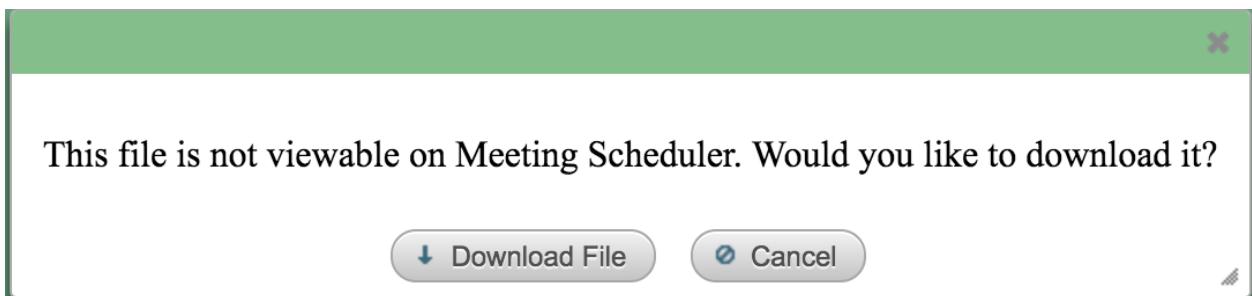


16\_Team 4 Semeste... 16\_SoftwareLifeCyc... 16\_CS3984 Java EE... 16\_HardAtWork.jpeg

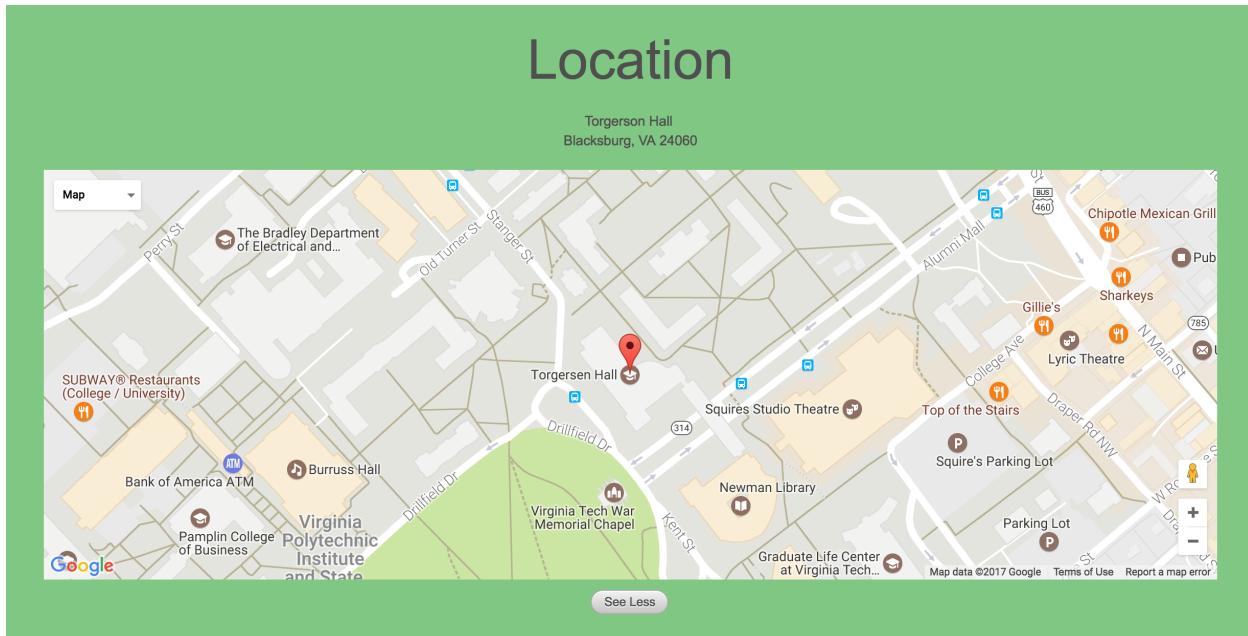
The file viewer is more robust than the CloudDrive file viewer: it will determine the file type and display it accordingly. For instance, unlike in CloudDrive, simply clicking a video file will play the video file: the user does not have to click and explicit “play MP4” button.



If the file is a file type that cannot be “viewed” in the traditional sense of the word (i.e. a .tar file), the application will convey this to the user and allow them to download the file instead.



Below the Attachments section, the user can view the location of the meeting in an embedded Google Map. Clicking the map will allow the user to interact with it.



Finally, clicking “See Less” shall collapse the card so that it takes up less space on the page. This is useful should the user wish to view another meeting and does not wish to scroll through the lengthy meeting card.

Two collapsed meeting cards.

## 7. Step by Step Instructions for Developing New Features

### 7.1 Calendar - Jeffrey Shih

The calendar feature we implemented is the 'Schedule' component provided by primefaces. The component itself renders a month, week, and day view with week and day views also providing hourly dividers. Arrows to scroll forward and back between the respective unit of time is provided as well as a button to take the user to the current date and a changing header to fit the current view. The component requires a backing Java manager in order to load any events within the calendar and to handle any clicks within the component.

Instructions:

1. Right click the **template** folder and select **New → XHTML...** from the pop-up menu.
2. Name the file **Calendar** and click Finish to create the **Calendar.xhtml** file.
3. Copy the code below into the **Calendar.xhtml** file (the value tag within the component acquires the ScheduleModel that will be used to grab all events and place them into the component when rendered).

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:f="http://xmlns.jcp.org/jsf/core"
  xmlns:p="http://primefaces.org/ui"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

  <!-- Do not enter tags before the composition line since they are ignored by JSF -->

  <!-- This page is constructed based on the siteTemplate -->
<ui:composition template="/template/siteTemplate.xhtml">

  <!-- Create the content for this page below -->
  <ui:define name="editableContent">
    <h:form>
      <p:growl id="messages" showDetail="true" />
      <div class="row">
        <div class="col s8 push-s1 indigo-text text-darken-4">
          <h:panelGrid columnClasses="value">
            <p:schedule value="#{calendarManager.getLazyEventModel(accountManager.selected)}"
              widgetVar="myschedule" timeZone="GMT-4" style="width:1000px">
              <p:ajax event="eventSelect" listener="#{calendarManager.onEventSelect}"
                update=":MeetingDetailsForm" oncomplete="PF('MeetingDetailsDialog').show();"/>
            </p:schedule>
          </h:panelGrid>
        </div>
      </div>
    </h:form>
    <ui:include src="MeetingDetails.xhtml"/>
  </ui:define>

</ui:composition>

</html>

```

We only process an event select for simplicity and felt that it was the only useful event that we could process in regards to the calendar.

1. Right click the **Source Packages** folder and select **New → Java Package...** from the pop-up menu.
2. Name the package **com.mycompany.managers** and click Finish to create the **com.mycompany.managers** package.
3. Right click the **com.mycompany.managers** package and select **New → Java Class...** from the pop-up menu.
4. Name the file **CalendarManager** and click Finish to create the **CalendarManager.java** file.
5. Copy the code below into **CalendarManager.java**

Copy and paste the documented code given below. **Carefully study the code, understand what it is doing, and learn from it!**

```
/*
```

```

* Created by Jeffrey Shih on 2017.04.05 *
* Copyright © 2017 Jeffrey Shih. All rights reserved. *
*/
package com.mycompany.managers;

/**
 *
 * @author Jeff
 */
import com.mycompany.entityClasses.Meeting;
import com.mycompany.entityClasses.MeetingUsers;
import com.mycompany.entityClasses.User;
import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collection;
import java.util.Date;
import java.util.List;
import java.util.concurrent.TimeUnit;
import javax.ejb.EJB;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;

import org.primefaces.event.ScheduleEntryMoveEvent;
import org.primefaces.event.ScheduleEntryResizeEvent;
import org.primefaces.event.SelectEvent;
import org.primefaces.model.DefaultScheduleEvent;
import org.primefaces.model.LazyScheduleModel;
import org.primefaces.model.ScheduleEvent;
import org.primefaces.model.ScheduleModel;

@ManagedBean(name = "calendarManager")
@SessionScoped
public class CalendarManager implements Serializable {

    private ScheduleModel eventModel;

    private ScheduleModel lazyEventModel;

    private ScheduleEvent event = new DefaultScheduleEvent();

```

```

@EJB
private com.mycompany.sessionBeans.MeetingFacade meetingFacade;
@EJB
private com.mycompany.sessionBeans.MeetingUsersFacade meetingUsersFacade;

private Collection<Meeting> meetingsList;

private Meeting selectedMeeting = new Meeting();

public Meeting getSelectedMeeting() {
    return selectedMeeting;
}

public void setSelectedMeeting(Meeting selectedMeeting) {
    this.selectedMeeting = selectedMeeting;
}

public ScheduleModel getEventModel() {
    return eventModel;
}

public ScheduleModel getLazyEventModel(User user) {
    System.out.print(user);
    if (lazyEventModel == null) {
        lazyEventModel = new LazyScheduleModel() {

            @Override
            public void loadEvents(Date start, Date end) {

                if (user.getMeetingCollection().isEmpty()) {
                    System.out.print("no meetings");
                } else {

                    //List<MeetingUsers> muL = meetingUsersFacade.getMeetings(user);
                    for (int i = 0; i < muL.size(); i++) {
                        Meeting updatedM =
                            meetingFacade.getMeetingById(muL.get(i).getMeeting().getId());

                        if (updatedM.getFinaltime() != null && updatedM.getFinaltime().length() >
                            0) {
                            //System.out.print("Final Start Time: " + updatedM.getFinaltime());
                        }
                    }
                }
            }
        };
    }
}

```

```

        ArrayList<Date> dateObj =
meetingFacade.deserialize(updatedM.getFinaltime());
        //System.out.print("AFter deserialized: " + dateObj.get(0).toString());
        Date endTime = addHour(dateObj.get(0));
        addEvent(new DefaultScheduleEvent(updatedM.getTopic(),
dateObj.get(0), endTime));
        //System.out.print("Shouldve Added");
        }
    }
}

};

}

return lazyEventModel;
}

private Date yearCheck(Date date) {
Calendar cl = Calendar.getInstance();
cl.setTime(date);
int year = cl.get(Calendar.YEAR);
if (year > 3000) {
cl.add(Calendar.YEAR, -1900);
}
System.out.print("Year check: " + cl.getTime().toString());
return cl.getTime();
}

private Date addHour(Date date) {

Calendar cl = Calendar.getInstance();
cl.setTime(date);
cl.add(Calendar.HOUR, 1);
cl.add(Calendar.DAY_OF_WEEK, -1);
cl.add(Calendar.DATE, 1);
int year = cl.get(Calendar.YEAR);
if (year > 3000) {
cl.add(Calendar.YEAR, -1900);
}
System.out.print("Add hour: " + cl.getTime().toString());
return cl.getTime();
}
}

```

```

public ScheduleEvent getEvent() {
    return event;
}

public void setEvent(ScheduleEvent event) {
    this.event = event;
}

public void addEvent(ActionEvent actionEvent) {
    if (event.getId() == null) {
        eventModel.addEvent(event);
    } else {
        eventModel.updateEvent(event);
    }
}

event = new DefaultScheduleEvent();
}

public void onEventSelect(SelectEvent selectEvent) {

    event = (ScheduleEvent) selectEvent.getObject();

    for (Meeting m : meetingsList) {
        if (event.getTitle().equals(m.getTopic())) {
            selectedMeeting = m;
            break;
        }
    }
}

public ArrayList<User> getParticipants() {
    Collection<MeetingUsers> participants = selectedMeeting.getMeetingUsersList();
    ArrayList<User> results = new ArrayList<>();
    if (participants == null || participants.size() < 1) {
        return results;
    }
    for (MeetingUsers mu : participants) {
        results.add(mu.getUser());
    }
}

return results;
}

```

```

public String getSelectedParticipants() {
    String out = "";

    Collection<MeetingUsers> participants = selectedMeeting.getMeetingUsersList();
    if (participants == null || participants.size() < 1) {
        return out;
    }
    for (MeetingUsers u : participants) {
        out += u.getUser().getFirstName() + " " + u.getUser().getLastName() + ", ";
    }
    return out.substring(0, out.length() - 2);
}

public void onDateSelect(SelectEvent selectEvent) {

}

public String getTime(Date date) {
    SimpleDateFormat localDateFormat = new SimpleDateFormat("HH:mm:ss");
    String time = localDateFormat.format(date);
    return time;
}

public void onEventMove(ScheduleEntryMoveEvent event) {
    FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_INFO,
        "Event moved", "Day delta:" + event.getDayDelta() + ", Minute delta:" + event.getMinuteDelta());
    addMessage(message);
}

public void onEventResize(ScheduleEntryResizeEvent event) {
    FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_INFO,
        "Event resized", "Day delta:" + event.getDayDelta() + ", Minute delta:" +
        event.getMinuteDelta());
    addMessage(message);
}

private void addMessage(FacesMessage message) {
    FacesContext.getCurrentInstance().addMessage(null, message);
}
}

```

The most important part of the java code is the getLazyEventModel method which is called by the schedule component. The method creates a single lazyEventModel if it has not already been created and loads all the events needed. In this case, all the events that are loaded are the meetings a user is a part of that has been finalized. The loop takes each meeting and deserializes the String representations of the final times and converts them into Java Date objects. An end time is also required for DefaultScheduleEvents which are the objects that are added to lazyEventModel, so an end time is created by processing the final time and adding an hour to it and creating it as a new Date object. Once all the meetings are added, the lazyEventModel is returned and rendered in the xhtml.

## 7.2 Utilizing Materialize CSS Library Side Nav - Alex Martin

The side nav implementation that we used from the Material CSS library. It allowed us to allow for an improved user experience by providing additional functionality at just one click without cluttering up the main views. We wanted to allow the user to see some of their account information, such as name, email, and profile image as well as provide some useful functionality such as linking to the calendar, the meetings they are hosting, and the information on their upcoming meetings. Follow the instructions below to implement your own side navigation menu.

1. Install Materialize, utilize the CDN repositories they host and add them inside the siteTemplate.html file so that they can be accessed site wide.
2. Place the code snippet below into your headerTemplate.html and review the code to understand what is going on.

```

1 //This is the declaration for the side nav menu. It must have an id, and it must be triggered by a button or link that has a data-activates="slide-out"
2 <ul id="slide-out" class="side-nav">
3     <c:if test="#{accountManager.isLoggedIn() == true}">
4         <div class="center">
5             <li>
6                 // This is where we setup the profile information view on the side bar. The Materialize library defines userView, profilePhoto, name, and email
7                 <div class="userView">
8                     <div class="background indigo darken-4">
9
10                     </div>
11                     <h:graphicImage class="profilePhoto center" value="#{accountManager.selected.getRelativeThumbnailFilePath()}" />
12                     <a href="#{!name}"><span class="white-text name">#{accountManager.selected.firstName} #{accountManager.selected.lastName}</span></a>
13                     <a href="#{!email}"><span class="white-text email">#{accountManager.selected.email}</span></a>
14                 </div>
15             </li>
16         </div>
17         //Now we link to the various places using materialize classes to create buttons
18         <li>
19             <h:link class="btn indigo darken-4 white-text" outcome="/Calendar">
20                 Calendar
21             </h:link>
22
23         </li>
24         <li>
25             <h:link outcome="/MeetingList" class="btn indigo darken-4">Meetings I Own
26             </h:link>
27         </li>
28         <p class='indigo-text text-darken-4 center'><b>Upcoming Meetings</b></p>
29
30         <c:forEach items="#{meetingController.getUpcomingMeetingsAfterToday(accountManager.selected)}" var='meeting'>
31             <li>
32                 <div class='divider'>
33
34             </div>
35         </li>
36         <li>
37             <div class="row center">
38                 <p:commandButton class="btn indigo darken-4 white-text"
39                     style="background: none;" actionListener="#{meetingController.prepareView(meeting)}"
40                     oncomplete="PF('MeetingViewHeaderDialog').show()"
41                     update=".MeetingViewHeaderForm"
42                     value="#{meeting.topic}"/>
43             </div>
44
45             <!--<a class="btn indigo darken-4 center" href="#{meetingModal#{meeting.id}}">#{meeting.topic}</a>-->
46         </li>
47     </c:forEach>
48     </c:if>
49 </ul>
50
51 <dialog id="MeetingViewHeaderDialog" widgetVar="MeetingViewHeaderDialog"
52     modal="true" resizable="false"
53     appendTo="body" header="View"
54     width="500" height="500"
55     class="grey lighten-3">
56
57     <h:form id="MeetingViewHeaderForm" class="ui-datatable-data">
58
59         <h:panelGroup id="display" rendered="#{meetingController.selected != null}">
60             <c:if test="#{meetingController.selected != null}">
61
62                 <div class="row center">
63                     <div class="col s3">
64                         <h:outputText value="Invitees" class="indigo-text text-darken-4" />
65                     </div>
66                     <div class="col s9">
67                         <pre>
68                             value="#{meetingController.getMeetingFacade().getParticipantList(meetingController.selected)}"
69                             var="invitee"
70                             size="#{meetingController.getMeetingFacade().getParticipantList(meetingController.selected).size()}"
71                             offset="0"
72                             step="1"
73                             varStatus="status">
74                             <div class="chip">
75                                 
76
77                                 #{{invitee.firstName} #{invitee.lastName}}
78                             </div>
79                         </pre>
80                     </div>
81                 </div>
82             </c:if>
83
84             <div class="row center">
85                 <div class="col s4">
86                     <p>Address 1:</p>
87                 </div>
88                 <div class="col s8">
89                     <p>Address 2:</p>
90                 </div>
91             </div>
92             <c:if test="#{meetingController.selected.address2 != null}">
93                 <div class="row">
94                     <div class="col s4">
95                         <p>Address 2:</p>
96                     </div>
97                     <div class="col s8">
98                         <p>Address 2:</p>
99                     </div>
100                </div>
101            </c:if>
102        </div>
103    </h:panelGroup>
104 </h:form>
105
106 </dialog>

```

```

<div class="row">
  <div class="col s4">
    <p class="indigo-text text-darken-4">City:</p>
  </div>
  <div class="col s8">
    <p class="indigo-text text-darken-4">#{meetingController.selected.city}</p>
  </div>
</div>
<div class="row">
  <div class="col s4">
    <p class="indigo-text text-darken-4">State:</p>
  </div>
  <div class="col s8">
    <p class="indigo-text text-darken-4">#{meetingController.selected.state}</p>
  </div>
</div>
<div class="row">
  <div class="col s4">
    <p class="indigo-text text-darken-4">Zipcode:</p>
  </div>
  <div class="col s8">
    <p class="indigo-text text-darken-4">#{meetingController.selected.zipcode}</p>
  </div>
</div>
<div class="row">
  <div class="col s4">
    <p class="indigo-text text-darken-4">Topic:</p>
  </div>
  <div class="col s8">
    <p class="indigo-text text-darken-4">#{meetingController.selected.topic}</p>
  </div>
</div>
<div class="row">
  <div class="col s4">
    <p class="indigo-text text-darken-4">Description:</p>
  </div>
  <div class="col s8">
    <p class="indigo-text text-darken-4">#{meetingController.selected.description}</p>
  </div>
</div>
<div class="row center">
  <p:commandButton value="#{bundle.Close}">
    onclick="MeetingViewDialog.hide()"
    class="btn indigo darken-4 white-text"
    style="background: none;"/>
</div>
</p:dialog>
</p:panelGroup>
</h:form>

```

When implementing the side navigation it is important to have a data-activates field on a link that has the id of the side nav menu, in order for the materialize javascript to properly perform the necessary functions.

## 7.3 Utilizing Materialize CSS Parallax - Alex Martin

Parallax is a design concept that raises the main content of a view above some images to allow content to separate and gives some flow to the webpages. We utilize the standard materialize parallax, but only on the bottom section of each page. Follow the instructions below to implement your own parallax content to your webpage through materialize.

1. Import the materialize css library through the use of CDN repositories into your siteTemplate.html file.
2. Paste the following code into your siteTemplate file, in order to utilize the parallax component.

```

<script>
    $(document).ready(function(){
        $('.parallax').parallax();
    });
</script>

<div class="section white">
    <div id="editableContent">
        <ui:insert name="editableContent" >

            <ui:include src="contentTemplate.xhtml" />
        </ui:insert>
    </div>
</div>
<div class="parallax-container" style="height: 250px;">
    <div class="parallax" style="z-index: 0;">
        
</div>

```

Some key things to remember when adding parallax to your application is to ensure that you set the parallax div to have a z-index of 0, in order to make it visible on the page. Notice the script at the top of the snippet, this section is required in order to initialize the JavaScript that is going to enable the scrolling speed altering between the content and the images.

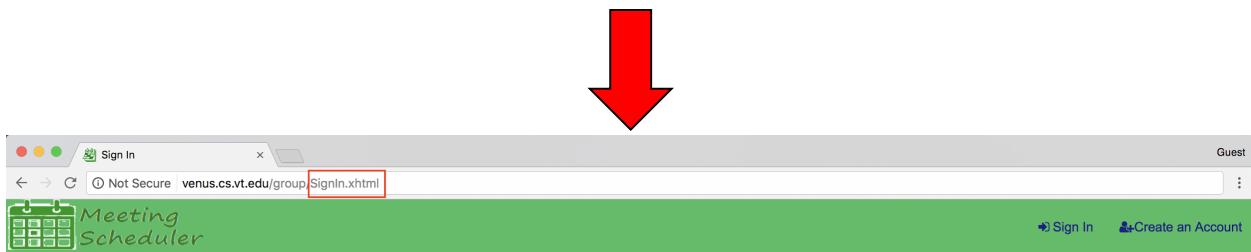
## 7.4 Login Filter for Members-Only Access - Patrick Gatewood

In order to restrict site access to logged-in members, we implemented a login filter. This filter intercepts every HTTP request and only allows resources through if the user is signed in, or is on the index, sign-in, create account, or other predetermined “public” pages. Clicking a link to a members only page or attempting to manually enter a members only URL will redirect the user to the sign-in page.

### Step 1: New Feature Functionality Specification

Attempting to manually enter a member’s only URL or clicking a members only link will redirect the user to the sign-in page.





## Sign In

Username:

Password:

**SIGN IN**

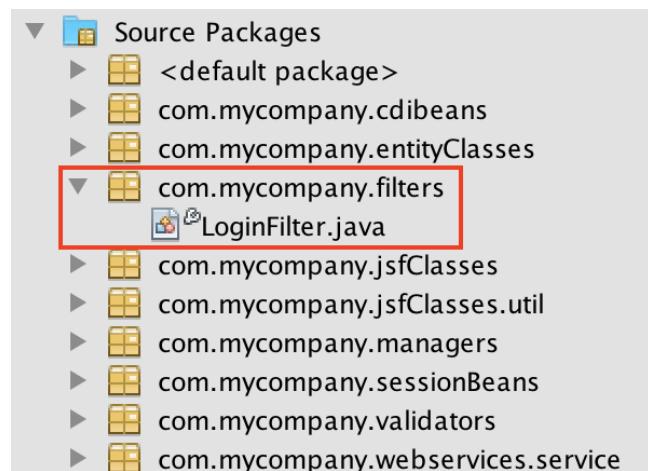
**CREATE AN ACCOUNT**

**FORGOT PASSWORD**

**CANCEL**

## Step 2: Packages and Java Class Files

First, create a “filters” package in order to keep your source folder organized. Create a new file, named LoginFilter.java. Place any additional filter classes you create in this package.



Copy and paste the documented code given below. **Carefully study the code, understand what it is doing, and learn from it!**

```

/*
 * Created by Patrick Gatewood on 2017.04.24
 * Adapted from the accepted answer at:
 * http://stackoverflow.com/questions/13274279/authentication-filter-and-servlet-for-login
 *
 * Copyright © 2017 Patrick Gatewood. All rights reserved.
 */

package com.mycompany.filters;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

/**
 * The purpose of this filter is to prevent users who are not logged in from
 * accessing confidential website areas.
 */
public class LoginFilter implements Filter {

    /**
     * Required method override. Nothing to initialize for now
     *
     * @param filterConfig the filter's configuration
     * @throws javax.servlet.ServletException
     */
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    /**
     * Checks HTTP requests and responses and restricts access to the main site
     * to logged-in users only
     */
}

```

```

/*
* @param req the HTTP request sent
* @param res the HTTP response received
* @param chain
* @throws java.io.IOException
* @throws javax.servlet.ServletException
* @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
*/
@Override
public void doFilter(ServletRequest req, ServletResponse res,
    FilterChain chain) throws IOException, ServletException {
    // Cast parameters to usable types
    HttpServletRequest request = (HttpServletRequest) req;
    HttpServletResponse response = (HttpServletResponse) res;

    // Path to the webpage requested
    String path = request.getRequestURI()
        .substring(request.getContextPath().length());

    HttpSession session = request.getSession(false);

    /* Restrict access to members-only content, but allow
     * the homepage, SignIn, and resources to be loaded */
    if (path.equals("/"))
        || path.endsWith(".css.xhtml")
        || path.endsWith(".js")
        || path.endsWith("index.xhtml")
        || path.endsWith("index.xhtml?faces-redirect=true")
        || path.endsWith("CreateAccount.xhtml")
        || path.endsWith("SignIn.xhtml")
        || path.endsWith("EnterUsername.xhtml")
        || path.endsWith("SecurityQuestion.xhtml")
        || path.endsWith("ResetPassword.xhtml")
        || path.contains("/resources/")
        || path.contains("js")) {
        chain.doFilter(request, response);
    } else if (session == null || session.getAttribute("username") == null) {
        response.sendRedirect(request.getContextPath() + "/SignIn.xhtml");
    } else {
        chain.doFilter(request, response);
    }
}

```

```

        }
    }

    /**
     * @see Filter#destroy()
     */
    @Override
    public void destroy() {
    }

}

```

This Login Filter was adapted from the accepted answer found at:  
<http://stackoverflow.com/questions/13274279/authentication-filter-and-servlet-for-login>.

### Step 3: WEB-INF Files

Open your web.xml file and add the following lines:

```

<!-- Login filter for restricted site access -->
<filter>
    <filter-name>Login Filter</filter-name>
    <filter-class>com.mycompany.filters.LoginFilter</filter-class> <!-- Use your own
package name! -->
</filter>
<filter-mapping>
    <filter-name>Login Filter</filter-name>
    <url-pattern>/*</url-pattern> <!-- Filter every request -->
</filter-mapping>

```



## 7.5 Collapsible “Card” Interface - Patrick Gatewood

Vertical screen real estate is expensive. In order to prevent extremely long web pages and to avoid using a tabbed interface, we implemented a collapsible card interface. This allows the user to see only the information they are interested in, and keeps vertical screen real estate usage to a minimum.

### Step 1: New Feature Functionality Specification

Meeting invitations and upcoming meetings alike will display a preview of the content upon navigating to the My Meetings page.

## Invitations

### Prospective Avengers Recruiting Event

If you are interested in joining the Avengers, please come to Stark Tower!

RESPOND

### Jennifer Lawrence Fan Club Meeting

Join us as we watch Jennifer's latest movies and talk all things Jennifer!

RESPOND

## Upcoming Meetings

### Deliver Semester Project to Dr. Balci

Owner Actions



225 Stanger St  
Blacksburg, VA 24060

We will deliver our semester project to Dr. Balci's  
office!

Final Time Not Yet Set

Clicking “Respond” on an invitation expands the invitation card to show the available dates and collapses all other open invitation cards to prevent the user from answering multiple invitations at the same time. The “Respond” button is also disabled and hidden.



Clicking a date will expand the card further to display the available times.



Clicking “See More” on an upcoming meeting expands the card to display more information about the meeting and disables the “See More” button.

# End of Semester Party

310 Webb St  
Blacksburg, VA 24060

The semester is over! Come out to The Edge and hang out to celebrate!

Final Time Not Yet Set

## Participants



Thor Odinson



Mr. Krabs



Erin Kocis



Patrick Gatewood

## Attachments

Clicking “Upload Files” expands the file upload section.

## Participants



Thor Odinson



Mr. Krabs



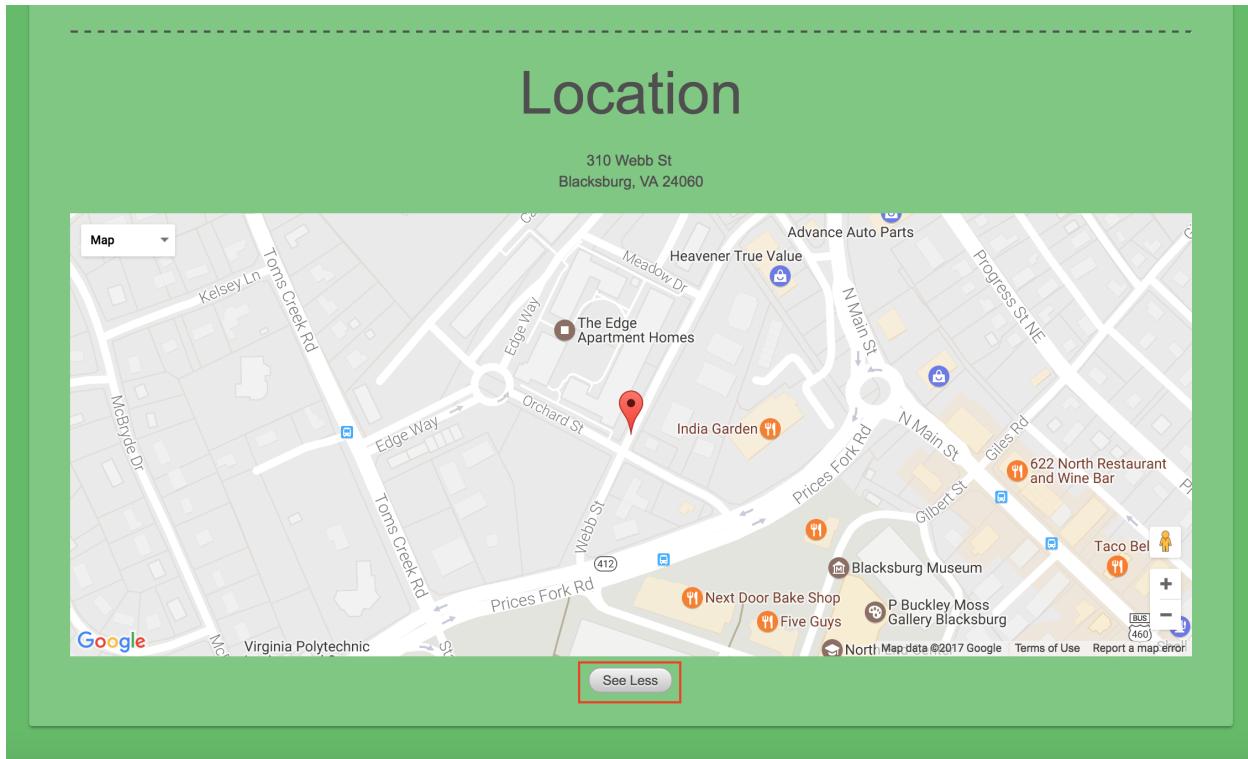
Erin Kocis



Patrick Gatewood

## Attachments

Clicking “See Less” collapses the card.



## End of Semester Party

310 Webb St  
Blacksburg, VA 24060

The semester is over! Come out to The Edge and hang out to celebrate!

Final Time Not Yet Set

[SEE MORE](#)

### Step 2: User Interface Development

Within an existing XHTML file to which you would like to add a collapsible panel, implement your collapsible panel similarly to the implementation below. **Carefully study the code, understand what it is doing, and learn from it! The XHTML tags and attributes highlighted in yellow are CRITICALLY IMPORTANT to making this functionality work!**

MeetingInvitation.xhtml

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
Created by Patrick Gatewood on 2017.04.08
Copyright © 2017 Patrick Gatewood. All rights reserved.
-->
<!DOCTYPE html [
  <!ENTITY nbsp "&#160;">
]>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:p="http://primefaces.org/ui"
      xmlns:c="http://xmlns.jcp.org/jsp/jstl/core">

<ui:composition>
  <!-- CSS for meetings -->
  <h:head>
    <h:outputStylesheet library="css" name="meetingStyles.css" />
  </h:head>

  <!-- Font Awesome CSS -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.6.2/css/font-awesome.min.css" />

  <div id="meetingWrapper#{meetingInvitation.id}" class="card-panel green
lighten-1" align="center">
    <!-- Meeting Invitation Preview with Respond Button -->
    <h2>#{meetingInvitation.topic}</h2>
    <p style="text-align: center">#{meetingInvitation.description}</p>

    <!-- Respond button clears any selected times and displays
    response choices to the user -->
    <p:commandButton action="#{userController.clearPotentialTimes()}">
      actionListener="#{meetingUsersController.setSelected(accountManager.selected,
meetingInvitation)}"
      widgetVar="respondButton#{meetingInvitation.id}"
      value="Respond"
      onclick="PF('respondButton#{meetingInvitation.id}').disable();"
    </p:commandButton>
  </div>
</ui:composition>

```

```

PF('dayPanel#{meetingInvitation.id}').expand();
collapseOtherInvitations("#{meetingInvitation.id}");
style="background: none;" class="btn indigo darken-4 white-text respond-button" />
<br />

<!-- Response section: user indicates their availability -->
<h:form enctype="multipart/form-data" class="timeform">
  <p:panel class="dayPanel green lighten-2" toggleable="true"
  closable="true" toggleSpeed="500" closeSpeed="500"
  collapsed="#{!meetingController.shouldHideTimeForMeeting(meetingInvitation)}"
  widgetVar="dayPanel#{meetingInvitation.id}" style="border: none; padding: 0px;" >
    <h4>When can you make it?</h4>

    <!-- Can't go button -->
    <p:commandLink
      actionListener="#{userController.clearPotentialTimes()}">
      action="#{meetingUsersController.finalizeMeetingAvailability(userController.potentialTimes, accountManager.selected, meetingController.selected.ownerId.email)}"
      oncomplete="collapseOtherInvitations('-1')"
      update=":meetingInvitations :growlForm:growl"
      class="declineLink">
        <i id="declineIcon#{meetingInvitation.id}" class="fa fa-times" aria-hidden="true"></i>
        <div class="declineText">Can't go</div>

        <p:confirm header="Confirmation"
          message="Are you sure you would like to decline this meeting
          invitation?">
          icon="ui-icon-alert"/>
        </p:commandLink>

      <!-- Div that floats left in order to force #horizontalScrollArea
      to fill the content width up to #declineLine -->
      <div style="float: left;"></div>

      <p:scrollPanel class="green lighten-2"
      id="horizontalScrollArea#{meetingInvitation.id}" styleClass="horizontalScrollArea"
      mode="native">

```

```

        <!-- Get list of times for the selected meetingInvitation -->
        <ui:param name="times"
value="#{meetingController.getMeetingFacade().getTimeslotsForMeeting(meetingInvitation)}" />

        <p:repeat
value="#{meetingController.getUniqueDateListForMeeting(meetingInvitation)}"
offset="0" step="1"
size="#{meetingController.getUniqueDateListForMeeting(meetingInvitation).size()}"
var="timeslot" varStatus="status">
            <!-- Clicking this link updates the collapsed time panel and displays
it -->
            <p:commandLink class="calendarLink"
action="#{meetingController.setSelectedDate(timeslot,
component.namingContainer.parent.namingContainer.clientId, meetingInvitation)}"
onclick="PF('timePanel#{meetingInvitation.id}').expand()"
ajax="true"
update="#{component.namingContainer.parent.namingContainer.clientId}">
                <time datetime="" class="icon">
<em>#{meetingController.getDayOfWeek(timeslot.getDay())}</em>

                <strong>#{meetingController.getMonthName(timeslot.getMonth())}</strong>
                <span>#{timeslot.getDate()}</span>
                </time>
            </p:commandLink>
        </p:repeat>
    </p:scrollPanel>
</p:panel> <!-- .dayPanel -->

        <!-- Available time options shown after the user selects a date -->
        <p:panel rendered="true" toggleable="true" closable="true"
toggleSpeed="500" closeSpeed="500"
collapsed="#{!meetingController.shouldHideTimeForMeeting(meetingInvitation)}"
widgetVar="timePanel#{meetingInvitation.id}" style="border:
none; padding: 0px;" class="green lighten-3">
            <p style="text-align: center;">Times for
            <b>#{meetingController.getSelectedDateAsString()}</b></p>

```

```

<p style="text-align: center;">Double click to select a time.</p>

<!-- Time selection: user indicates their availability -->
<p:panelGrid id="timePanel#{meetingInvitation.id}" columns="4"
columnClasses="columnOne, columnTwo, columnOne, columnTwo" style="padding: 0;">
<ui:fragment>
<h:panelGroup id="timeGroup#{meetingInvitation.id}">
<ui:param name="timesForDay" value="#{meetingController.getTimesForDay(times, meetingController.selectedDate)}" />
<p:repeat value="#{timesForDay}" offset="0" step="1" size="#{timesForDay.size()}" var="timeslot" varStatus="status">
<!-- On/Off switch -->
<p:commandLink id="switch#{meetingInvitation.id}" action="#{userController.updatePotentialAvailability(timeslot)}" ajax="true" update="@form" style='text-decoration: none; padding-bottom: 10px;'>
<div class="green lighten-3 center">
<label class="switch">
<ui:fragment rendered="#{userController.checkboxShouldBeChecked(timeslot)}">
<input type="checkbox" checked="true"/>
</ui:fragment>
<ui:fragment rendered="#{!userController.checkboxShouldBeChecked(timeslot)}">
<input type="checkbox"/>
</ui:fragment>
<div class="slider round" style="height: 3em;"></div>
</label>
<br/>
<h:outputText styleClass="onOffLabel" value="#{meetingController.getMeetingFacade().getMeetingString(timeslot)}" />
</p:commandLink>
</p:repeat>
</h:panelGroup>

```

```

</p:panelGrid>

<br /><br/>

<!-- Submit button -->
<p:commandButton
action="#{meetingUsersController.finalizeMeetingAvailability(userController.potentialTimes, accountManager.selected, meetingController.selected.ownerId.email)}"
    ajax='true'
    value="Submit all selected times"
    oncomplete="collapseOtherInvitations(-1)"
    update="upcomingMeetings, meetingInvitations, @form,
:growlForm:growl"
    style="background: none;"
    class="btn indigo darken-4 white-text respond-button" >

    <p:confirm header="Confirmation"
        message="Are you sure you would like to submit all selected
dates and times?"
        icon="ui-icon-alert" />
    </p:commandButton>
    &nbsp;&nbsp;
</p:panel> <!-- .timePanel -->
</h:form>

</div> <!-- #meetingWrapper -->

<!-- Collapses other panels when an invitation is opened.
This prevents the user from accepting multiple invitations
at once and keeps the interface clean.

NOTE: This script must be included inline because p:repeat tags
do not put elements in the DOM in time, and this whole page is
contained in a p:repeat. Therefore this JavaScript needs to be
included right here. -->
<script type="text/javascript">
    function collapseOtherInvitations(keepOpenId) {
        // Special case to prevent collapsing the panel just expanded
        if (keepOpenId === -1) {
            return;
        }
    }
</script>

```

```

        }
        // Variable declaration
        var widgets = PrimeFaces.widgets;
        var dayPanelKeepOpen = "dayPanel" + keepOpenId;
        var timePanelKeepOpen = "timePanel" + keepOpenId;
        var property;

        // Find all expanded panels
        for (property in widgets) {
            // Don't collapse the panel currently being opened
            if (property.includes(dayPanelKeepOpen)
                || property.includes(timePanelKeepOpen)) {
                continue;
            }

            // Collapse day panels
            if (property.includes("dayPanel")) {
                PF(property).collapse();
            }

            // Collapse time panels
            if (property.includes("timePanel")) {
                PF(property).collapse();
            }

            // Re-enable all other respond buttons
            if (property.includes("respondButton")) {
                if (!(!property.includes(keepOpenId))) {
                    PF(property).enable();
                }
            }
        }
    }

    </script>

</ui:composition>
</html>

```



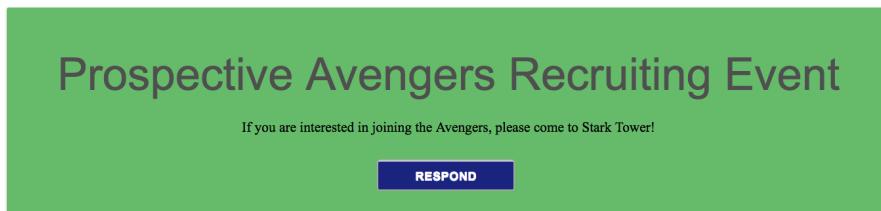
## 7.5 Repeatable Content for Interoperability between JSF Implementations and PrimeFaces Components - Patrick Gatewood

In order to create a custom interface template that does not depend on existing PrimeFaces components and allow that template to be reused for data extracted from our SQL database, we implemented our interface template and wrapped it in the PrimeFaces Repeat extension. This gave us *complete* control over our template rather than limiting us to styling an existing PrimeFaces component such as the PrimeFaces Datatable.

### Step 1: New Feature Functionality Specification

For each meeting extracted from the database, a meeting card will be created and displayed using our template.

#### Invitations



#### Upcoming Meetings



### Step 2: Template Development

To create your own template, follow the implementation below. Include the template file you wish to repeat by placing your template within the

**<ui:include src="Your-XHTML-File-To-Repeat"/>** tag wrapped in a **<p:repeat>**.

**DO NOT** use the **<ui:insert>** component, as the **<ui:insert>** tag handler runs at build time, while the **<p:repeat>** component is run during render time! This will result in none of your XHTML code being inserted!

**Carefully study the code, understand what it is doing, and learn from it! The XHTML tags and attributes highlighted in yellow are CRITICALLY IMPORTANT to making this functionality work!**

MyMeetings.xhtml

---

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Created by Patrick Gatewood on 2017.03.26
Copyright © 2017 Patrick Gatewood. All rights reserved.
-->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:c="http://xmlns.jcp.org/jsp/jstl/core">

<!-- Do not enter tags before the composition line since they are ignored by JSF -->

<!-- This page is constructed based on the siteTemplate -->
<ui:composition template="template/siteTemplate.xhtml">

  <ui:define name="title">
    <!-- Set the page title -->
    <h:outputText value="My Meetings"></h:outputText>
  </ui:define>

  <ui:define name="editableContent">
    <h:outputStylesheet library="css" name="myMeetingsStyles.css" />
```

```

<h:form id="growlForm">
    <!-- Growl that shows messages to the user -->
    <p:growl id="growl" life="5000" />
</h:form>

<div class="main" align="center">

    <!-- Meeting Invitations Section -->
    <h1>Invitations</h1>
    <h:panelGroup id="meetingInvitations" >
        <ui:param name="meetingInvitations"
value="#{meetingController.getMeetingInvitations(accountManager.selected)}" />

        <!-- Display a message when there are no meetings to show -->
        <c:if test="#{meetingInvitations.size() == 0}" >
            You have no new meeting invitations.
        </c:if>

        <c:if test="#{meetingInvitations.size() != 0}" >
            <p:repeat value="#{meetingInvitations}" offset="0" step="1"
size="#{meetingInvitations.size()}" var="meetingInvitation" varStatus="status">
                <ui:include src="meetings/MeetingInvitation.xhtml"/>
                <br />
            </p:repeat>
        </c:if>
    </h:panelGroup>
    <br />

    <!-- Upcoming Meetings Section -->
    <h1>Upcoming Meetings</h1>
    <h:panelGroup id="upcomingMeetings" >
        <ui:param name="upcomingMeetings"
value="#{meetingController.getUpcomingMeetings(accountManager.selected)}" />

        <!-- User has no upcoming meetings -->
        <c:if test="#{upcomingMeetings.size() == 0}" >
            Woo-hoo! You have no upcoming meetings.
        </c:if>

```

```

<!-- Display the user's upcoming meetings, if they exist -->
<c:if test="#{upcomingMeetings.size() != 0}" >
    <p:repeat value="#{upcomingMeetings}" offset="0" step="1"
    size="#{upcomingMeetings.size()}" var="upcomingMeeting" varStatus="status">
        <ui:include src="meetings/UpcomingMeeting.xhtml"/>
        <br />
    </p:repeat>
</c:if>
</h:panelGroup>

<!-- Global confirmDialog to be used by all child pages -->
<p:confirmDialog id="confirmDialogBox" widgetVar="confirmDialog"
global="true">
    <div align="center">
        <!-- The value of the first button listed below will be shown as
        highlighted (default action) when displayed -->
        <p:commandButton value="No" type="button"
            styleClass="ui-confirmedialog-no" icon="ui-icon-close" />
        &nbsp;
        <p:commandButton value="Yes" type="button"
            styleClass="ui-confirmedialog-yes" icon="ui-icon-check"
            />
    </div>
</p:confirmDialog>

<!-- Script for Google Maps -->
<script async="defer"
src="https://maps.googleapis.com/maps/api/js?key=AlzaSyCX_25HBwVQYvr31
Dcz3MJUryv5Aak5okQ&callback=initMap"></script>
</div>
</ui:define>
</ui:composition>
</html>

```

## 8. Conclusions

The main goal of the project was to allow users to coordinate by indicating the availability of their schedules in order to select the most effective time for a meeting. In

addition, we implemented many features that makes this application one that not only schedules meetings, but also provides users with a platform to keep track of, organize, and plan for their meetings.

To schedule meetings, the best way to gather everyone's availability information was to allow meeting participants to view a selection of times provided by the meeting owner and select the ones that work for their schedule. We alert meeting participants, via email, that they have been invited to help schedule a meeting as soon as the meeting owner creates it. Because the application also organizes all meetings a user has been invited to, the user can look at his or her calendar to check for time conflicts with any other meetings that have been finalized. The meeting owner can then check his or her own dashboard to see the results of the invitees' responses. If the owner decides that enough people have responded, he or she can go on to select the final meeting time. Having the email functionality reduces the hassle of communicating to others as an automated email is sent as soon as a relevant action is taken by the meeting owner or the meeting invitees. The ability to share documents and files to all users included in a meeting allows everyone to be well prepared before a meeting occurs and even serves as a fallback in meeting participants forget or can't access the files from another location at the time of the meeting.

While on the surface the application looks very sleek and simple, we found this project to be equally challenging and rewarding. Through our combined experience of working in teams, whether it be for school or in the professional world, all members of our team had encountered the difficult task of attempting to schedule meetings while working with the schedules of several busy individuals. These experiences gave us drive to create, in our minds, the most efficient and effective way to schedule meetings. In addition to completing the given task of scheduling a when and where for a meeting, we challenged ourselves by adding the above functionality to make our application more robust all aspects of planning, organizing, and holding meetings.

## 9. Percentages of Contribution

We hereby certify that the list of contributions and the corresponding percentages of contribution specified below truly reflect the actual contributions of the team members.  
(Write your name as your signature)

<i>Student Name &amp; Contributions</i>	<i>% Contributed</i>	<i>Signature</i>

<p>Alexander Martin</p> <p>1. SQL Script to Create DB  2. SQL Script to populate test data  3. Named Queries for MeetingUsers  4. Implementation of Materialize  5. Styling of Homepage, CRUD views, Profile, Sign In, Password Recovery, Header, Footer, and Side Navigation  6. General bug fixes and code cleanup</p> <p>Percentage of final report: 40%</p>	22%	AJM
<p>Erin Kocis</p> <p>1. Took notes during each group meeting for the Meeting Minutes.  2. My Meetings Landing Page Markup Design. (using Balsamiq with Patrick)  3. Implemented Upcoming Meetings XHTML.  4. Google Maps</p>	24%	EMK

<p>functionality in Upcoming Meetings.</p> <p>5. Populate Upcoming Meetings portion of My Meetings landing page with real data.</p> <p>6. User profile photo functionality (different implementation than tutorials).</p> <p>7. Designed email template and implemented automatic email functionality.</p> <p>8. Merge several pull requests and test teammates' code</p> <p>9. Conducted SQL database dump and prepared ZIP file for Delivery to Dr. Balci with Patrick</p> <p>Percentage of final report: 20%</p>		
<p>Jeffrey Shih</p> <p>1. Calendar implementation</p> <p>2. Initial page setup</p> <p>3. Footer links and images that scroll</p>	18%	Jeffrey Shih

<p>4. Logo designer</p> <p>5. Meeting finalization</p> <p>Percentage of final report: 20%</p>		
<p>Jisu You</p> <p>1. Meeting CRUD</p> <p>2. Meeting CRUD JSF page</p> <p>3. MeetingController development</p> <p>Percentage of final report: 0%</p>	6%	Jisu You
<p>Patrick Gatewood</p> <p>1. Create preliminary login services</p> <p>2. Login Filter for members-only access</p> <p>3. Developed and implemented SQL logic to store user and meeting availability timeslots</p> <p>4. Meeting Invitations XHTML and backing bean methods</p> <p>5. My Meetings Landing Page Markup Design.</p>	30%	Patrick Gatewood

<p>(using Balsamiq with Erin)</p> <ol style="list-style-type: none"> <li>6. Improve email sending to send on a background thread</li> <li>7. Implement Meeting File Upload</li> <li>8. Upgrade CloudDrive file viewer to more robust version</li> <li>9. Create MyMeetings template</li> <li>10. Implement collapsable card-base design</li> <li>11. Register for Google Maps API key and implement Google Map</li> <li>12. Redo Meeting CRUD operations</li> <li>13. Add Autocomplete for usernames when creating meetings</li> <li>14. Meeting Timeslot selection</li> <li>15. CSS Styling for My Meetings</li> <li>16. Merge several pull requests and test teammates' code</li> <li>17. Deployed application to the</li> </ol>		
--	--	--

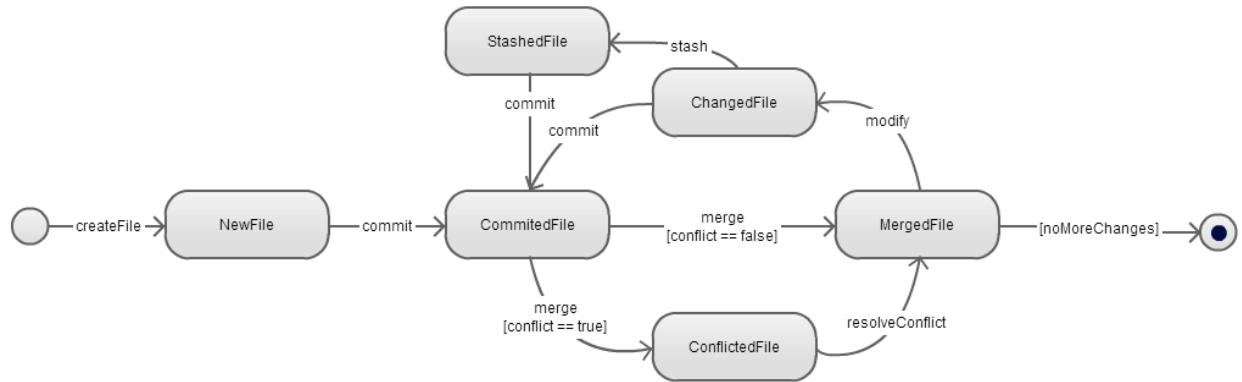
<p>server computer 18. Conducted SQL database dump and prepared ZIP file for Delivery to Dr. Balci with Erin  Percentage of final report: 20%</p>		
---	--	--

The Virginia Tech Honor Code is fully in effect for the above declaration.

# 11. Version Control

In order to help our team manage our source code over time, we placed our code inside a git repository. This allowed our team to track all changes made to files, easily revert file revisions, create development branches, and maintain a clean master branch that was ready to present at any time.

The following state machine describes how the decentralized version control system handles file changes. The diagram was obtained under fair use from: <http://nbdiff-docs.readthedocs.io/en/latest/SRS.html#information-on-version-control-systems>.



Our git repository can be found on GitHub: <https://github.com/pg8wood/cloud-semester-project>

## 11. Collaborative Project Management

After completing our Requirements Specification, our team decided to practice scrum development. We organized our development into one-week sprints, with a scrum meeting once a week.

Each week, we discussed the modules we had developed and presented our progress to the rest of the team. Whenever a developer finished a module, we assigned them a new ticket from our backlog of TODO items and added a new item to the backlog.

Our issue management system of choice was Trello. You can find our project board here: <https://trello.com/b/Shhgeo2r/web-app>

## 12. Meeting Report Forms

---

**•Meeting of Project Team 4**

**•Date:** 3.2.2017

**•Location:** TORG 1010

**•Agenda:** First meeting - Discuss project overview (features we want to build, source control, plan for team organization)

**•Names of Team Members Who Attended the Meeting:**

- Erin Kocis, Patrick Gatewood, Alex Martin, Jeffrey Shih, Jisu You

**•Names of Team Members Who were Absent:** N/A

**•Discussions:**

- Went over list of recommended features, determined which ones we want to focus on
- Team organization - made a team drive
- Project meeting times for the remainder of the semester

**•Announcements (if any):** NA

**•Decisions Made:**

- Weekly meeting (Wednesday - 9:00pm - Software Engineering Lab)
- Use GitHub, Trello, and Google Drive
- Features we are interested in : 1, 2, 3, 4, 5, 6, 8, 10, 11, 12, 14, 15, 16, 19, 21, 22, 24, 25, 28, 29

**•Items for Follow Up:**

- Git repo creation - Patrick
- Google Drive setup - Erin
- Trello setup - Alex
- Brainstorm site layout before next meeting

**•Name of the Person Taking the Minutes:**

- Erin Kocis
- 

**•Meeting of Project Team 4**

**•Date:** 3.15.2017

**•Location:** Google Hangouts – virtual meeting

**•Agenda:** Discuss physical design components & divide up work for sprints

**•Names of Team Members Who Attended the Meeting:**

- Erin Kocis, Patrick Gatewood, Alex Martin, Jeffrey Shih, Jisu You

**•Names of Team Members Who were Absent:** N/A

**•Discussions:**

- Discussed physical design components
- Discussed options for coding languages to use
- Discussed next steps and which items are the most important and fundamental to begin the project

**•Announcements (if any):** NA

**•Decisions Made:**

- Use tickets on Trello to divide up work (and assigned tickets for upcoming week)
- Work in week long sprints
- Use SQL and Java on the backend

**•Items for Follow Up:**

- Create user class – Patrick & Erin
- Create SQL script for user and Meeting Entities – Alex & Jason
- Basic XHTML Layout / Graphics - Jeffrey

**•Name of the Person Taking the Minutes:**

- Erin Kocis
- 

**•Meeting of Project Team 4**

**•Date:** 3.22.2017

**•Location:** Google Hangouts – virtual meeting

**•Agenda:**

- Go over progress/problems from this week
- Discuss next steps
- Assign tickets to team members for next week

**•Names of Team Members Who Attended the Meeting:**

- Erin Kocis, Patrick Gatewood, Alex Martin, Jeffrey Shih, Jisu You

**•Names of Team Members Who were Absent:** N/A

**•Discussions:**

- Next steps with splash page after logging in
- User profile
- Create a meetings template
- Web service for user information
- How to create users and meetings
- Various ways to display meetings a user is a part of

**•Announcements (if any):** NA

**•Decisions Made:**

- For this week's sprint, focus on working on meeting object
- Use one database with multiple tables
- Want to display meetings for each user chronologically (by date)
- Cards done :
  - Create SQL Script for User and Meeting Entities
  - Basic XHTML Layout / Graphics

**•Items for Follow Up:**

- NA

**•Name of the Person Taking the Minutes:**

- Erin Kocis
-

**•Meeting of Project Team 4**

**•Date:** 3.29.2017

**•Location:** Google Hangouts – virtual meeting

**•Agenda:**

- Go over progress/problems from this week
- Discuss next steps
- Assign tickets to team members for next week

**•Names of Team Members Who Attended the Meeting:**

- Erin Kocis, Patrick Gatewood, Alex Martin, Jeffrey Shih, Jisu You

**•Names of Team Members Who were Absent:** N/A

**•Discussions:**

- Problems and progress everyone made this past week
- Primefaces Schedule component - include it on the landing page
  - has calendar, month, day, time
  - each person has a set of meeting times
  - make sure it shows up on the calendar correctly
- Discuss login page

**•Announcements (if any):** NA

**•Decisions Made:**

- New meeting time will be Tuesdays at 1:00pm
- After cards have been completed, add them to “Verifying” before merging with the master branch

-Cards go from: Backlog -> Do this week -> In Progress -> Verifying -> Done

- Added to cards for this week:

- JSF for Meetings - Jason
- Create landing page - Erin & Patrick
- View Meetings by User - Alex
- Schedule Demo - Jeffery

- Added to cards done :

- Mock Meetings/Users for Testing Purposes - Alex
- Add sign up, login, and user functionality - Patrick
- Create User Class - Erin & Patrick

**•Items for Follow Up:**

- NA

**•Name of the Person Taking the Minutes:**

- Erin Kocis

---

**•Meeting of Project Team 4**

**•Date:** 4.5.2017

**•Location:** Google Hangouts – virtual meeting

**•Agenda:**

- Review items from last week's meeting
- Prep for this week - talk about next steps and assign tickets

- Prepare for the PowerPoint presentation for this week

- **Names of Team Members Who Attended the Meeting:**

- Erin Kocis, Patrick Gatewood, Alex Martin, Jeffrey Shih, Jisu You (slightly late)

- **Names of Team Members Who were Absent:** N/A

- **Discussions:**

- PowerPoint presentation structure
  - Objective
  - Work completed
  - Work in progress
  - Next steps

- Went over list of functionality options we intend to complete
  - Discussed next week's work items for each group member

- **Announcements (if any):** NA

- **Decisions Made:**

- Work on Email & Google Maps functionality next - Erin
  - Continue working on current in progress items listed on Trello

- **Items for Follow Up:**

- NA

- **Name of the Person Taking the Minutes:**

- Erin Kocis

---

- **Meeting of Project Team 4**

- **Date:** 4.11.2017

- **Location:** Google Hangouts – virtual meeting

- **Agenda:**

- Review items from last week's meeting
  - Prep for this week - talk about next steps and assign tickets
  - Prepare for the PowerPoint presentation for this week

- **Names of Team Members Who Attended the Meeting:**

- Erin Kocis, Patrick Gatewood, Alex Martin, Jeffrey Shih, Jisu You (slightly late)

- **Names of Team Members Who were Absent:** N/A

- **Discussions:**

- PowerPoint presentation structure - sticking to same structure as last week
  - Objective
  - Work completed
  - Work in progress
  - Next steps
- Each person presented on this week's work
  - Jason is working on CRUD related functionality for meetings
  - Patrick and Erin used Balsamiq to create a markup of the splash page and worked on some basic front end design for the splash page
    - Jeffery is working with the calendar, showing all of a user's upcoming meetings
    - Alex is working on the query to get all meetings by a specific user

- Discussed whether or not we want to keep edit functionality for meetings - after a user has accepted do we want to allow people to change meeting details

- Discussed next week's work items for each group member

- **Announcements (if any):** NA

- **Decisions Made:**

- Continue work on Email & Google Maps functionality - Erin

- For subsequent meetings, have everyone add their individual slides to the PowerPoint presentation before we meet in order to save time

- Continue working on current in progress items listed on Trello

- Edit meeting creation (Jason)

- Create email functionality (Erin)

- Populate landing page with real data (Patrick & Erin)

- Get all users for particular meeting & get all user responses for meeting (Alex)

- Schedule demo & finish calendar functionality (Jeffery)

- **Items for Follow Up:**

- NA

- **Name of the Person Taking the Minutes:**

- Erin Kocis

---

- **Meeting of Project Team 4 (surprise meeting during class)**

- **Date:** 4.13.2017

- **Location:** Torg 1010

- **Agenda:**

- Touch base with what everyone has been working on

- Everyone work independently (but have the option to ask others in the group for help)

- **Names of Team Members Who Attended the Meeting:**

- Erin Kocis, Patrick Gatewood, Alex Martin, Jeffrey Shih, Jisu You (slightly late)

- **Names of Team Members Who were Absent:** N/A

- **Discussions:**

- Talked about what everyone has been working on, progress made, and problems we have encountered

- Touched base regarding upcoming week's work items for each group member

- **Announcements (if any):** NA

- **Decisions Made:**

- NA

- **Items for Follow Up:**

- NA

- **Name of the Person Taking the Minutes:**

- Erin Kocis

---

- **Meeting of Project Team 4**

- **Date:** 4.18.2017

• **Location:** Google Hangouts

• **Agenda:**

- Touch base with what everyone has been working on / progress that has been made
- Compile and go over PowerPoint presentation for this week

• **Names of Team Members Who Attended the Meeting:**

- Erin Kocis, Patrick Gatewood, Alex Martin, Jeffrey Shih, Jisu You (slightly late)

• **Names of Team Members Who were Absent:** N/A

• **Discussions:**

- Talked about what everyone has been working on, progress made, and problems we have encountered
- Talked about adding work done to the PowerPoint and assigned who would talk about which slides
- Discussed remaining components for the base functionality
  - Picking final meeting time should wrap most of it up
- Discussed some of the additional functionality components we want to start incorporating
  - Discussed upcoming week's work items for each group member
    - Schedule Demo, Calendar Functionality & Choose/Display Best Time (JS)
    - User Profile Functionality & Email Functionality (EK)
    - Meeting File Upload (PG)
    - Meeting Response Handler & REST Endpoints (AM)
    - Edit Meeting Creation / finish Meetings CRUD (JY)

• **Announcements (if any):** NA

• **Decisions Made:**

- Meeting\_Users stores user's and meeting's primary key
- All meeting times will be stored as a single string and then deserialized

• **Items for Follow Up:**

- Majority of the group is blocked
  - Waiting on Jason to complete meeting CRUD operations in order to figure out how to select the best date

• **Name of the Person Taking the Minutes:**

- Erin Kocis
- 

• **Meeting of Project Team 4**

• **Date:** 4.25.2017

• **Location:** Google Hangouts

• **Agenda:**

- Touch base with what everyone has been working on / progress that has been made
- Compile and go over PowerPoint presentation for this week
- Evaluate project requirements to make sure we are addressing everything

• **Names of Team Members Who Attended the Meeting:**

- Erin Kocis, Patrick Gatewood, Alex Martin, Jeffrey Shih, Jisu You

- **Names of Team Members Who were Absent:** N/A

- **Discussions:**

- Talked about what everyone has been working on, progress made, and problems we have encountered

- Designed and discussed this week's PowerPoint presentation and assigned who would talk about which slides

- Discussed some of the additional functionality components we want to start incorporating

- Discussed some "new" items we have incorporated (login filter, calendar component)

- CSS to dress up the site - Alex will begin working on this

- Discussed upcoming week's work items for each group member

- Jason still working on CRUD operations

- **Announcements (if any):** NA

- **Decisions Made:**

- Instead of multiple select, add interested times as a ArrayList

- Alex will start working on CSS

- Jason will push CRUD by tonight

- Someone needs to verify Jeffery's PR

- Patrick and Erin go to office hours to address questions about functionality requirements

- **Items for Follow Up:**

- Group members blocked by unfinished meeting CRUD

- Jason said CRUD will be done by this evening

- **Name of the Person Taking the Minutes:**

- Erin Kocis

## REFERENCES

Balci, O. (2017), “CS3984 Cloud Software Development Course Website,” <http://manta.cs.vt.edu/cs3984>

GlassFish (2017), “GlassFish Application Server,” <https://glassfish.java.net/>

MySQL (2017), “MySQL Open Source Relational Database Management System,” <http://www.mysql.com/>

NetBeans (2017), “NetBeans IDE,” <https://netbeans.org/>

Software Requirements Specification — NBDiff 1 documentation. (2017). Nbdiff-docs.readthedocs.io. Retrieved 9 May 2017, from <http://nbdiff-docs.readthedocs.io/en/latest/SRS.html#information-on-version-control-systems>