

AN ONLINE REGISTRATION SYSTEM FOR THE MATH DAY EVENT

by

Matthew A. DeHaven

A THESIS

Presented to the Faculty of

The College of Art and Sciences at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Bachelor of Science

Major: Computer Science

Under the Supervision of Professor Berthe Y. Choueiry

Lincoln, Nebraska

May, 2017

# AN ONLINE REGISTRATION SYSTEM FOR THE MATH DAY EVENT

Matthew A. DeHaven, B.S.

University of Nebraska, 2017

Adviser: Berthe Y. Choueiry

Math Day is an annual outreach event of the Department of Mathematics of the University of Nebraska-Lincoln, organized by the Center for Science, Mathematics and Computer Education (CSMCE) of UNL. Running the event requires the participation of a large number of volunteers. Currently, gathering volunteer information is a time-intensive task. We have created a system with a database and web interfaces to streamline the collection of volunteer information. This document describes the foundations of the system design and of the web interface that volunteers will use to sign up for Math Day.

## ACKNOWLEDGMENTS

I would like to thank Dr. Berthe Choueiry for offering me the opportunity to conduct this research and for her guidance and encouragement throughout.

I am grateful to Ms. Stephanie Vendetti for introducing me to the Math Day event and guiding me through the required functionalities to implement in the system.

I am grateful to Mr. Charles Daniel for initiating me to Ruby on Rails and helping me with all system-related issues.

The design of the database was contributed by Dylan Laible and Christopher Lyons. The design of the ‘contact us’ functionality was contributed by Justin Collier. The design of the survey page mentioned in Section 2.3.2 was contributed by Gerald Thornton.

*This research was supported by NSF Grant No. NSF RI-1619344.*

# Contents

<b>Contents</b>	<b>iv</b>
<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Application Requirements . . . . .	2
1.3 Contributions . . . . .	3
1.4 Thesis Organization . . . . .	4
<b>2 The Volunteers Interface</b>	<b>5</b>
2.1 Volunteers' Front Page . . . . .	5
2.1.1 The WELCOME Page . . . . .	6
2.1.2 The LOGIN Page . . . . .	7
2.1.3 The REGISTRATION Page . . . . .	9
2.1.4 The ABOUT US and CONTACT US Pages . . . . .	12
2.2 Application Process . . . . .	12
2.2.1 The Progress Bar . . . . .	14

2.2.2	The APPLICATION Form . . . . .	15
2.2.3	The YASP Form . . . . .	17
2.2.4	The TASK SIGNUP Form . . . . .	19
2.3	The DASHBOARD . . . . .	22
2.3.1	The Rationale . . . . .	23
2.3.2	The Functionalities . . . . .	25
<b>3</b>	<b>The Administrator Interface</b>	<b>27</b>
3.1	Active Admin Pages . . . . .	27
3.1.1	Authentication . . . . .	28
3.1.2	Namespace . . . . .	28
3.1.3	Dashboard . . . . .	29
3.1.4	Applications . . . . .	29
3.1.5	Users . . . . .	29
3.1.6	Faculty Recruiters . . . . .	30
3.2	‘Regular’ Admin Pages . . . . .	31
3.3	Creating a New Year . . . . .	32
<b>4</b>	<b>System Design</b>	<b>34</b>
4.1	Software Design . . . . .	34
4.1.1	Program Organization . . . . .	35
4.1.2	Runtime Configurations . . . . .	36
4.1.3	Account Authentication . . . . .	37
4.2	Environments . . . . .	39
4.3	Database Synchronization . . . . .	40
4.3.1	Data Changes . . . . .	40
4.3.2	Schema Changes . . . . .	41

<b>5 Database Documentation</b>	<b>44</b>
5.1 Activities and Tasks . . . . .	46
5.1.1 activities_seeds . . . . .	46
5.1.2 activities . . . . .	47
5.1.3 tasks_seeds . . . . .	48
5.1.4 tasks . . . . .	49
5.1.5 user_tasks . . . . .	51
5.1.6 capable_tasks_seeds . . . . .	51
5.1.7 capable_tasks . . . . .	52
5.1.8 visible_tasks_seeds . . . . .	53
5.1.9 visible_tasks . . . . .	53
5.2 Volunteers . . . . .	54
5.2.1 user_types . . . . .	55
5.2.2 users . . . . .	55
5.2.3 affiliations . . . . .	58
5.2.4 application . . . . .	59
5.2.5 t_windows . . . . .	61
5.2.6 assignment_by_users . . . . .	62
5.2.7 assignments . . . . .	63
5.2.8 class_recruiteds . . . . .	64
5.2.9 faculty_recruiters . . . . .	65
5.3 Manager . . . . .	66
5.3.1 announcements . . . . .	66
5.3.2 locations . . . . .	67
5.3.3 surveys . . . . .	67
5.3.4 email_messages . . . . .	69

**6 Conclusions**

# List of Figures

2.1	Front page for volunteers . . . . .	6
2.2	WELCOME page . . . . .	6
2.3	LOGIN page . . . . .	7
2.4	LOG IN WITH EMAIL page . . . . .	8
2.5	FORGOT PASSWORD page . . . . .	9
2.6	REGISTRATION page . . . . .	9
2.7	Registration after MY.UNL authentication . . . . .	10
2.8	Registration after authentication via email . . . . .	11
2.9	Application process . . . . .	13
2.10	Four states of the progress bar . . . . .	14
2.11	Application form . . . . .	15
2.12	Accessing the APPLICATION page . . . . .	16
2.13	The YASP form . . . . .	17
2.14	Accessing the YASP form . . . . .	18
2.15	The TASK SIGNUP form . . . . .	19
2.16	Accessing the TASK SIGNUP form . . . . .	21
2.17	The routing process . . . . .	23
2.18	The DASHBOARD page . . . . .	25



3.1	Four pages in Active Admin . . . . .	28
3.2	Applications page in Active Admin . . . . .	29
3.3	Accessing custom actions in Active Admin . . . . .	30
3.4	Accessing faculty recruiters' page in Active Admin . . . . .	30
4.1	Model-View-Controller connections . . . . .	34
4.2	Database Data Flow . . . . .	40
4.3	Database Schema Flow . . . . .	42
5.1	Database Diagram . . . . .	45

## List of Tables

5.1	Structure of table activities_seeds . . . . .	46
5.1	Structure of table activities_seeds (continued) . . . . .	47
5.2	Structure of table activities . . . . .	47
5.3	Structure of table tasks_seeds . . . . .	48
5.4	Structure of table tasks . . . . .	49
5.4	Structure of table tasks (continued) . . . . .	50
5.5	Structure of table user_tasks . . . . .	51
5.6	Structure of table capable_tasks_seeds . . . . .	52
5.7	Structure of table capable_tasks . . . . .	52
5.8	Structure of table visible_tasks_seeds . . . . .	53
5.9	Structure of table visible_tasks . . . . .	53
5.9	Structure of table visible_tasks (continued) . . . . .	54
5.10	Structure of table user_types . . . . .	55
5.11	Structure of table users . . . . .	55
5.11	Structure of table users (continued) . . . . .	56
5.12	Structure of table affiliations . . . . .	58
5.13	Structure of table applications . . . . .	59
5.13	Structure of table applications (continued) . . . . .	60

5.14	Structure of table t_windows . . . . .	61
5.15	Structure of table assignment_by_users . . . . .	62
5.16	Structure of table assignments . . . . .	63
5.17	Structure of table class_recruiteds . . . . .	64
5.18	Structure of table faculty_recruiters . . . . .	65
5.19	Structure of table announcements . . . . .	66
5.20	Structure of table locations . . . . .	67
5.21	Structure of table surveys . . . . .	67
5.21	Structure of table surveys (continued) . . . . .	68
5.22	Structure of table email_messages . . . . .	69

# Chapter 1

## Introduction

In this chapter, we describe the motivation behind designing and building an online system to support the management of volunteers for running the Math Day event. We overview the rationale behind the design decisions and the various components of the system that we implemented. Finally, we summarize the organization of this thesis.

### 1.1 Motivation

The Math Day even is annually organized by the Center for for Science, Mathematics and Computer Education (CSMCE) of the University of Nebraska-Lincoln (UNL) as an outreach activity of the Department of Mathematics of UNL.

Each year, over 1500 students from all over the state of Nebraska come to UNL to participate in the event. Math Day serves as an important recruitment and outreach activity for UNL, allowing students to learn about the opportunities of which that they may not be aware to study mathematics at UNL. Also, a number of students who win the competition are offered scholarships to study at UNL.

To run Math Day, well over 250 volunteers are needed each year. Currently, collecting volunteer information is done by hand. This process requires a significant amount of effort and careful follow up. Our goal is to streamline the collection and storage of volunteer information by creating an online system that volunteers can use to register and the event manager can examine for follow up.

## 1.2 Application Requirements

The web interface for volunteer signup has several requirements we have to meet.

First, because the web server is hosted at UNL, our site needs to match the guidelines for UNL's official web pages. This aspect is mainly a stylistic requirement of look-and-feel features. For this reason, we decided to use UNL's Web Developer Network (WDN) stylesheets.

Next, the site needs to allow two different methods of authentication. Many of the volunteers for the event will be associated with UNL and will have an account through UNL's Client Authentication System (CAS). As such, we want to allow volunteers to register using UNL's CAS if they so desire. However, there are also volunteers that do not have UNL CAS accounts (e.g., alumni, South East Community College, and UNMC). Such volunteers must also be able to use the site. To accommodate these volunteers, we need to provide an email-based authentication method as well.

Additionally, we need to provide a method to capture information (i.e., properties) about volunteers that is either 'permanent' (i.e., static) or may change with time (i.e., fluents). A person's name is unlikely to change. However, a person's affiliation, status, or available time windows can change from year-to-year. This requirement implies our system should allow a volunteer to enter the static information only once but update fluents every year, as they volunteer to help in the event.

When collecting fluents from volunteers, we need to have a dynamic system that allows the volunteer to enter information that depends on selections they have made earlier in the process. For example, the set of tasks visible to a volunteer depends on the volunteer's status and role. Also, undergraduate students, but no other volunteers, are allowed to specify a Math faculty and Math class that will grant them extra credit for their participation.

Last, we need to provide a method for the event administrators to be able to manage volunteers and tasks. A separate interface needs to be created that is not accessible to anyone other than the administrators. This interface must provide functionalities to view and edit applications and send individual and bulk email-messages to volunteers and recruiting faculty.

## 1.3 Contributions

The contributions of this thesis are the following:

1. Elucidate the application requirements
2. Refine the design of an initial MySQL database and document it
3. Design, implement, and test the volunteer authentication process
4. Design, implement, and test various interactive webpages for collecting volunteer information
5. Design, implement, and test a control mechanism for linearly guiding a volunteer through the application process
6. Design, implement, and test various mechanisms for error detection, error reporting, and email communications with the users

7. Produce an initial prototype for the administrator interface

## **1.4 Thesis Organization**

This thesis is organized as follows. In Chapter 2, we describe the interface used by the volunteers. In Chapter 3, we describe the interface used by the event administrators. In Chapter 4, we describe the design of the system. Finally, in Chapter 5, we describe the database and its structure.

## Chapter 2

# The Volunteers Interface

In this chapter, we describe the various pages, forms, and processes that form the interface accessible by a volunteer. The description is organized in three main sections: The front page, the application process, and the dashboard.

### 2.1 Volunteers' Front Page

Volunteers access the system through the URL:

`http://mathdayvolunteers.unl.edu`

The first page that they see is the WELCOME page, shown in Figures 2.1 and 2.2, which gives them access gives a volunteer access to the following pages:

1. WELCOME (Section 2.1.1)
2. LOGIN (Section 2.1.2)
3. REGISTRATION (Section 2.1.3)
4. ABOUT US and CONTACT US (Section 2.1.4)

Below, we discuss in detail each of these pages.



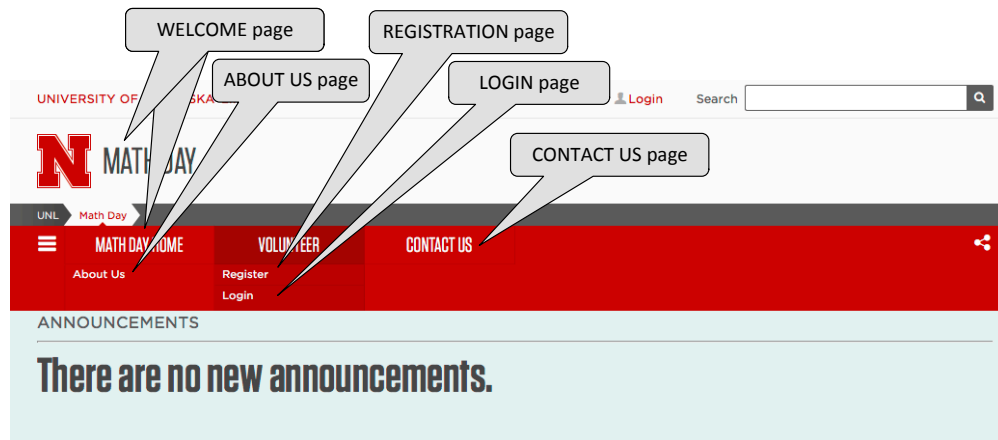


Figure 2.1: Front page for volunteers

### 2.1.1 The WELCOME Page

The first page displayed to a volunteer is the WELCOME page, shown in Figure 2.2. On this page, the volunteer sees, on top, announcements (i.e., message of the day or

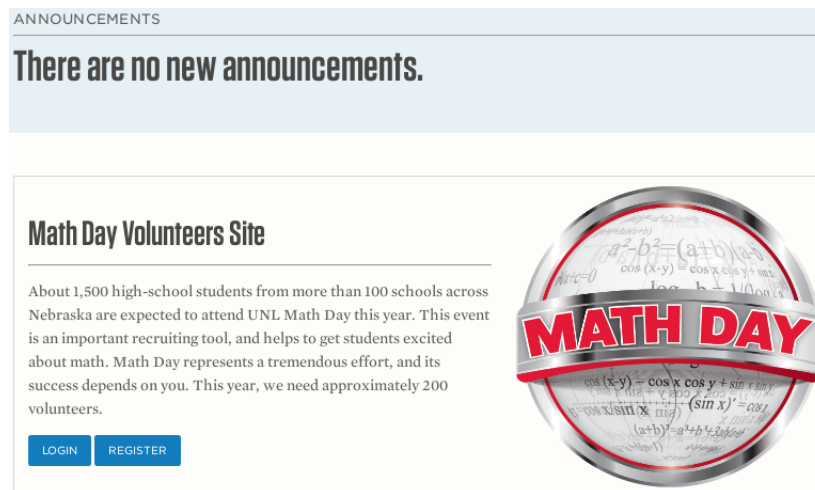


Figure 2.2: WELCOME page

MOTD) and, in the middle, general information about Math Day. At the bottom of this page, are two buttons: one to login to an existing account (i.e., the LOGIN page) and another one for registering a new account (i.e., the REGISTRATION page).

## 2.1.2 The LOGIN Page

When a volunteer selects the ‘LOGIN’ option on the WELCOME page, they are taken to the LOGIN page shown in Figure 2.3. A volunteer would choose this option

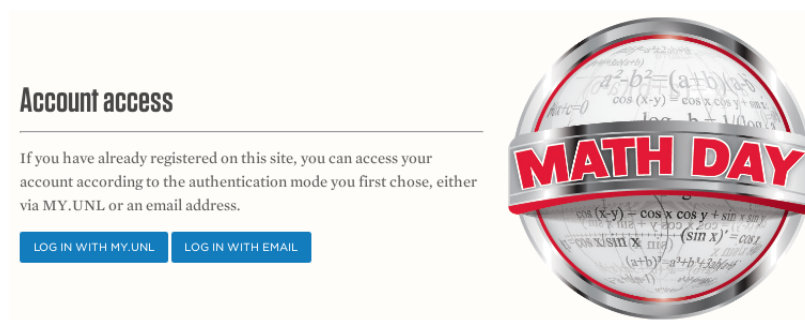


Figure 2.3: LOGIN page

if they have an existing account that they need to access. This page is similar to the WELCOME page (Figure 2.2), but has two different options at the bottom, namely, ‘LOG IN WITH MY.UNL’ and ‘LOG IN WITH EMAIL.’ Because we provide two different methods for account authentication, we have two different links that the volunteer can use depending on which method they used to create their account.

### 2.1.2.1 Login with My.UNL

If a volunteer chooses to login with My.UNL, they are taken to the My.UNL authentication page, which is not part of our system but belongs to the UNL Central Authentication Service (CAS) of the university. If they are successfully authenticated, they are immediately directed to the Math Day site. The My.UNL authentication system provides the Math Day site with the information needed to identify the user that has just signed in by providing the user’s CAS username.

Because My.UNL account authentication is beyond our control, utilities such as password resetting are provided by that system for those registering under this option.

This functionality significantly reduces the burden of remembering usernames and password for UNL users.

**2.1.2.2 Login by Email**

If a volunteer chooses to ‘LOG IN WITH EMAIL,’ they are directed to the form shown in Figure 2.4. This form simply asks for the email address and password that

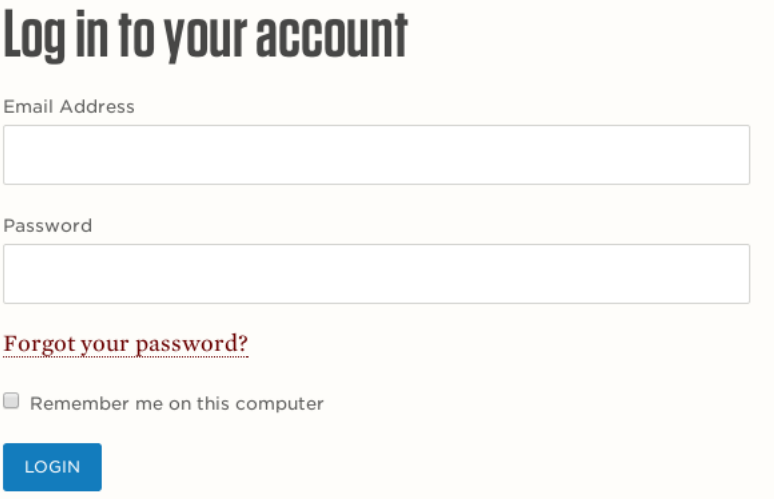


Figure 2.4: LOG IN WITH EMAIL page

the volunteer had initially used to register. If they provide the correct combination of email address and password, they are given access to their account.

Because these accounts are managed by the MathDay site, we need to provide a functionality for a volunteer to recover their account if they forgot their password. A link to recover an account is provided on the LOGIN page. Choosing this link takes the volunteer to the FORGOT PASSWORD page shown in Figure 2.5. When the form on this page is submitted, our system sends a link the email address provided. Clicking that link allows the user to change the password for that account.

Figure 2.5: FORGOT PASSWORD page

### 2.1.3 The REGISTRATION Page

The first time a volunteer accesses our system, they must create an account by registering. To this end, from the WELCOME page, they press the 'Register' button, which directs them to the REGISTRATION page, see Figure 2.6.

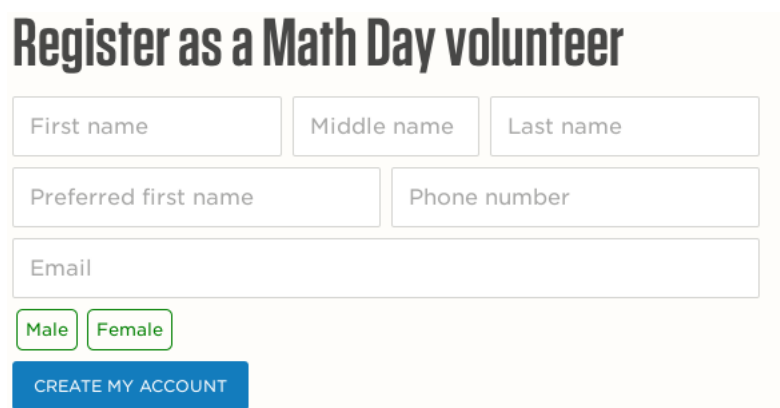
Figure 2.6: REGISTRATION page

This page is similar to the WELCOME page (Figure 2.2). It has the same layout, but two different buttons, which now are: 'Register with My.UNL' and 'Register with email,' which are the two authentication methods that we provide. We allow a volunteer to create their account based on their My.UNL login, which may be more convenient for many students because they will not have to create a new account and password. They can simply use their existing account they already use for many other UNL services. If a volunteer does not have a My.UNL account, or would rather

not use it, they can instead register with an email address.

### 2.1.3.1 My.UNL Registration

If a volunteer chooses to register with their My.UNL account by choosing ‘Register with My.UNL’ from the REGISTRATION page, they are directed to the My.UNL login page, which is not part of our system. After authentication with My.UNL, the volunteer is given a form to input their profile, as shown in Figure 2.7. This



The image shows a registration form titled "Register as a Math Day volunteer". The form is set against a light yellow background. It contains several input fields: "First name", "Middle name", and "Last name" in a row; "Preferred first name" and "Phone number" in a row; and a larger "Email" field. Below these fields are two radio buttons labeled "Male" and "Female". At the bottom of the form is a blue button with the text "CREATE MY ACCOUNT".

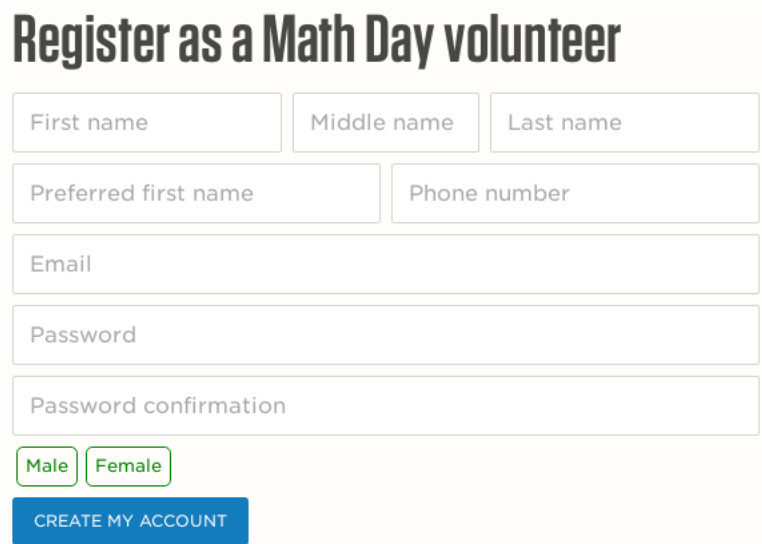
Figure 2.7: Registration after MY.UNL authentication

form asks for the volunteers name, cellphone number, email, and gender. We need this information so we know who is registering and how to contact them. When registering using this method, the volunteer does not need to enter a password because the My.UNL system manages the authentication process.

After successfully submitting the form, the volunteer is directed back to the WELCOME page and shown a message prompting them to check their provided mailbox for an activation link. Clicking the link in that email message activates the account and completes the registration.

### 2.1.3.2 Registration by Email

If a volunteer elects to register with an email address by choosing ‘Register with email’ in the REGISTRATION page, they are directed to the registration form shown in Figure 2.8.



The registration form is titled "Register as a Math Day volunteer". It contains the following fields and elements:

- Three input boxes for "First name", "Middle name", and "Last name".
- Two input boxes for "Preferred first name" and "Phone number".
- A single input box for "Email".
- A single input box for "Password".
- A single input box for "Password confirmation".
- Two radio buttons for "Male" and "Female".
- A blue button labeled "CREATE MY ACCOUNT".

Figure 2.8: Registration after authentication via email

This form is almost identical to the form a volunteer fills out when registering with My.UNL (see Figure 2.7). The only difference is that the volunteer must enter a password. Because they are registering with an email address, we have to provide the authentication and, consequently, the volunteer must provide a password.

After successfully submitting the form, the volunteer is taken back to the WELCOME page and prompted to check their provided mailbox for an activation link. Clicking the link in that email message activates the account and completes the registration.

### 2.1.4 The ABOUT US and CONTACT US Pages

There are two pages available from the REGISTRATION page that are not part of the normal registration process. These are the ABOUT US and CONTACT US pages. Both pages are accessible to any visitor of the site through the top bar (see Figure 2.1). The ABOUT US page simply contains information describing the Math Day event. The CONTACT US page allows a visitor of the site to send a message to either the Math Day administrator or the website administrators.

## 2.2 Application Process

Once a volunteer has created an account, they need to complete the application process, which is a sequence of forms that they need to fill out to indicate how they can help on Math Day. The forms are the following:

1. APPLICATION (Section 2.2.2)
2. YASP (Section 2.2.3)
3. TASK SIGNUP (Section 2.2.4)

Because there are multiple parts to this process, we guide volunteers through it in a step-by-step manner.

Figure 2.9 describes the application process. This diagram shows that there are three parts to the process, namely, APPLICATION, YASP, and TASK SIGNUP. Each step is a different form that must be completed. The APPLICATION and the YASP forms must be completed by *all* volunteers. The last form, TASK SIGNUP, is available only to volunteers who are 'activity workers.' A volunteer indicates if they are an activity worker in the APPLICATION form. This fact means that it is

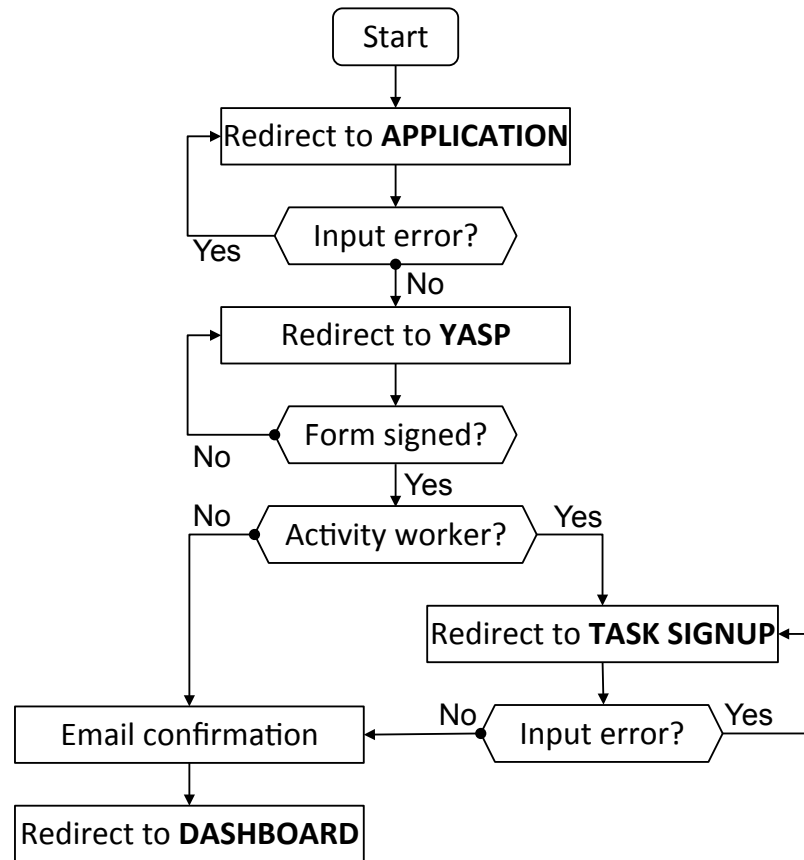


Figure 2.9: Application process

important that the steps be completed in sequence because later steps rely on input provided in earlier steps.

To ensure that each step is completed in order while keeping the process simple, we guide a volunteer through the process in a step-by-step manner. We first show them the APPLICATION form. If they submit the form and there are errors in the input, we keep them on the application form until it is submitted without errors. Once it has been submitted without errors, we then show the volunteer the YASP form. Again, if there are any errors on the form, the volunteer is kept on the YASP form until it is error free. Once the YASP form is submitted without errors, the process continues in one of two ways. If the volunteer indicated on the APPLICATION form to be an



activity worker, the TASK SIGNUP form is displayed. Otherwise, the application process is completed.

When the volunteer is directed to the TASK SIGNUP form, they are prevented from moving on in the process until the form is submitted without errors. Once the TASK SIGNUP form is completed, the volunteer has completed the application process.

Regardless of the path that the volunteer followed to complete the application process, after the process is completed, the volunteer is directed to the DASHBOARD page (see Section 2.3) and receive a confirmation message by email listing all the details of the application. After completing the process, a volunteer can still make changes to any of the forms previously filled out. Further, at each update, a confirmation message with the updated information is sent to the volunteer for traceability.

### 2.2.1 The Progress Bar

The volunteer's progress in the application process follows Figure 2.9 and its state is reflected by a progress bar shown in each of the three forms. The progress bar can be in any of the four states shown in Figure 2.10 indicating whether or not a given form was completed. Notice that the TASK SIGNUP option is shown only if a volunteer has indicated to be an activity worker on the APPLICATION form.

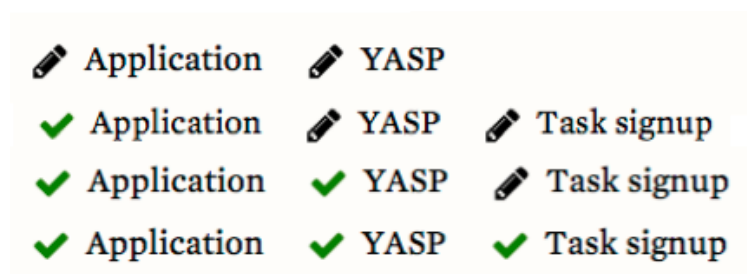


Figure 2.10: Four states of the progress bar

If a volunteer clicks on a link to try to access a part of the process that they should not yet access, they will be redirected to the earliest step of the process that they have not yet completed. For instance, if a person has not yet completed the APPLICATION form, it is possible for that person to click the YASP link on the progress bar. The volunteer is not given access to the YASP form until the APPLICATION form has been completed. Even without the links in the progress bar, a volunteer could enter in the browser the URL of a later step. Therefore, any time a step in the process is requested, our system ensures that the volunteer has access to the requested step. Note that any completed step can be accessed from the progress bar at any time.

## 2.2.2 The APPLICATION Form

The first form that the volunteer has to fill is the APPLICATION form. This form is shown in Figure 2.11. In the application form, the volunteer enters information

**Update application** ✓ Application ✓ YASP ✓ Task signup

Select all that apply

- Speaker ⓘ
- Department Display ⓘ
- Activity Worker ⓘ

Status: Undergraduate

Affiliation: MATH

Faculty recruiter: [Empty]

Class recruited: [Empty]

Comments: [Empty]

**UPDATE APPLICATION**

Figure 2.11: Application form

about the roles they would like to perform, their current status, affiliation, and for undergraduate math students, the instructor who has recruited them for the event

and the name of the class in which they were recruited so that they receive proper credit for their participation.

The logic that controls the process is shown in Figure 2.12. Given that the AP-

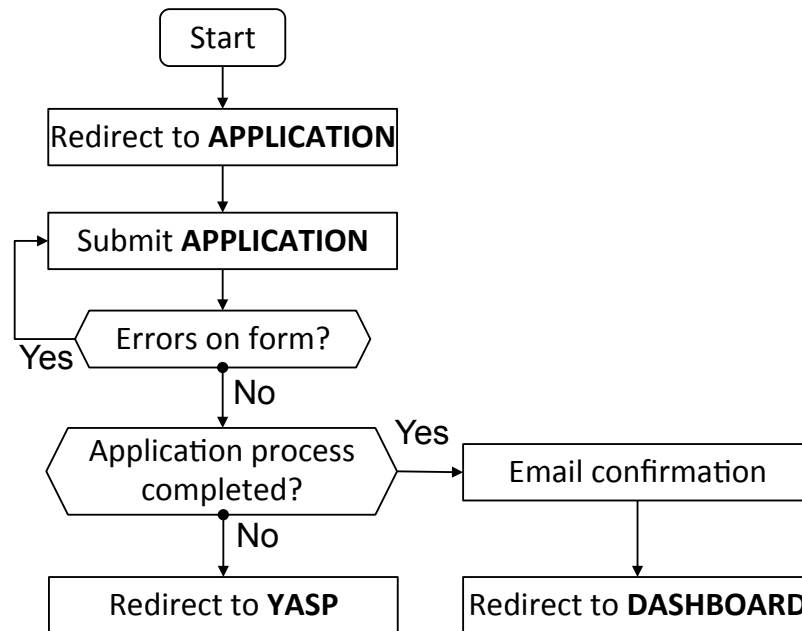


Figure 2.12: Accessing the APPLICATION page

PLICATION form is the first step in the application process, no no conditions are checked to determine whether or not a volunteer can be shown this form.

If there is an error on the form when it is submitted, the volunteer is prompted with the appropriate message about the error and shown the application form again.

Once the volunteer has submitted the form, one of two actions can occur. If the volunteer has already completed all of the other forms in the process, which can occur only in the case that the volunteer is updating the APPLICATION form, the volunteer is redirected to the DASHBOARD (see Section 2.3) and receives an email summarizing the content of the updated application. Otherwise, the volunteer is shown the earliest form in the process that is yet to be completed, which is, either the YASP form, if it has not been completed, or the TASK SIGNUP form.

### 2.2.3 The YASP Form

The second form in the application process is the YASP form. This form is shown in Figure 2.13. This form simply displays an official university document and prompts

✓ APPLICATION   ✎ YASP   ✎ TASK SIGNUP

**N** UNIVERSITY OF NEBRASKA-LINCOLN

## Youth Activities Safety Policy

### Activity Worker Guidelines

*(Activity Directors: Provide one copy of this document to Activity Worker)*

The following guidelines were developed to avoid questionable contact between Activity Workers and youth participants. The preferred method is to avoid private one-on-one interactions and always have another adult observer present during all interactions with youth. This will provide greater protection to the youth and the Activity Workers.

#### Definitions

**Youth** – Any person under the age of 19, excluding full- and part-time UNL students

**Activity Worker** – includes directors, coaches, assistant coaches, trainers, student assistants, staff, faculty, officials, referees or contracted teachers or any other adult or student acting as a supervisor/mentor/worker in a paid, unpaid or volunteer status.

**Activity Director** – a person who plans, directs and supervises all youth activity programs and staff

**Activity Support Staff** – any person who provides support services such as food service, custodial, maintenance, etc. for the activity.

#### Activity Worker Indorsement

I have read and understand these policies and agree to abide by them.

\* Full Legal Name   \* Phone Number

[SIGN FORM](#)

Figure 2.13: The YASP form

the volunteer to sign it indicating that they agree with the guidelines specified in that document.

The logic controlling the process of filling out the YASP form is shown in Figure 2.14. The YASP form is only available if the APPLICATION form has been completed. If a volunteer requests the YASP form before first filling the APPLICATION form, the volunteer is redirected to the APPLICATION form.

The YASP form is unique in comparison to the other forms in the process in that it

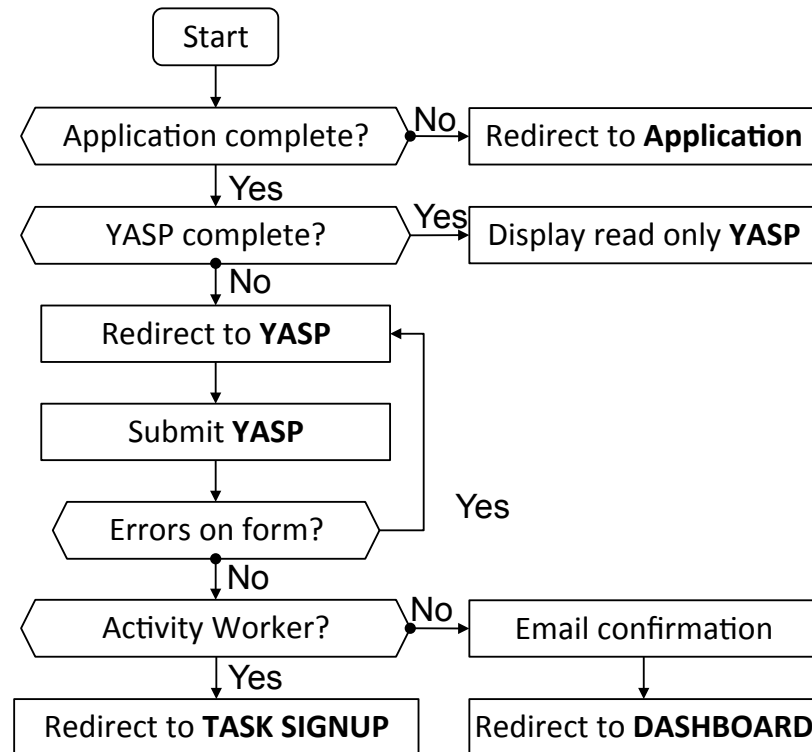


Figure 2.14: Accessing the YASP form

cannot be edited once it has been submitted. This is because it is just a confirmation that the volunteer agrees to the terms specified by the document displayed in the form. Therefore, if a volunteer has already filled out the YASP form, and requests it again, they will receive a read-only version of the form duly indicating the date on which the form was signed.

If the YASP form has not yet been submitted but the APPLICATION form has, the user is directed to the YASP form. If the YASP form is submitted with errors, the volunteer will be shown the form again. Otherwise, one of two actions will occur. In case the volunteer indicated on the APPLICATION form to be an activity worker, they are directed to the TASK SIGNUP form. Otherwise, the volunteer is directed to the DASHBOARD (Section 2.3) and receives a confirmation message by email with a summary of the application.

## 2.2.4 The TASK SIGNUP Form

The third and last form in the application process is the TASK SIGNUP form. This form is shown in Figure 2.15.

Task Name	Status	Time Window	Preference
Opening Ceremony II	📍	7:00 AM - 9:00 AM	Acceptable ▾
Setup II	📍	7:00 AM - 9:00 AM	Acceptable ▾
PROBE I Proctor	📍	8:15 AM - 10:00 AM	Acceptable ▾
Command Center I	📍	10:00 AM - 12:15 PM	Acceptable ▾
Timer	📍	10:00 AM - 12:15 PM	Acceptable ▾
PROBE II Proctor	📍	12:00 PM - 1:20 PM	Acceptable ▾
Command Center II	📍	1:00 PM - 3:45 PM	Acceptable ▾
Timer	📍	1:00 PM - 3:30 PM	Acceptable ▾
Championship Rounds	📍	3:00 PM - 5:00 PM	Acceptable ▾
Timer	📍	3:00 PM - 4:30 PM	Acceptable ▾
Command Center	📍	8:30 AM - 4:30 PM	Acceptable ▾
Puzzles	📍	12:00 PM - 3:30 PM	Acceptable ▾

Figure 2.15: The TASK SIGNUP form

In this form, a volunteer indicates the time intervals in which they can make themselves available during the Math Day event by increments of 15 minutes.

The list of *all* the tasks to which a volunteer can be assigned is displayed even if the tasks that do not fit in the time windows selected by the volunteer. The tasks that a volunteer is shown depends on the information that they entered in the APPLICATION form, notably their status. Different volunteer statuses can be entrusted on different tasks.

The tasks that are contained within the time windows selected by the volunteer are highlighted in green, and the volunteer is given the option of indicating one of four preferences for each task (i.e., Preferred, Acceptable, Avoid, and Can't Do) of the green-colored tasks.

We intentionally choose to show a volunteer all of the tasks their status entitles them to do, even if a given task is not contained in any of the time windows they indicated. The reason behind this decision is that, sometimes, volunteers may adjust their availability depending on the tasks that they prefer to do. If they know when all the tasks are occurring, then they can be sure to edit their availability to match whatever tasks they prefer.

Additionally, each task has a tool-tip that gives a description of the task. The name of the task alone may not be particularly meaningful to a volunteer. If a volunteer would like to know more about the task, they simply hover over the question mark symbol displayed with each task and a more information about that task will be shown.

The logic controlling the process is shown in Figure 2.16. If a volunteer requests access to the TASK SIGNUP form before finishing either the APPLICATION form or the YASP form, the volunteer is shown the earliest form in the process that has not yet been completed. If the volunteer has completed both of those two forms, but did not indicate on the APPLICATION form to be an activity worker, then they are not offered the option of filling out the TASK SIGNUP form, and are instead redirected to the DASHBOARD (see Section 2.3).

If all earlier forms have been completed and the volunteer indicated to be an activity worker, the volunteer is shown the TASK SIGNUP form. When the volunteer submits the form with errors, they are shown the form again with a message indicating the errors to be fixed. Otherwise, the volunteer is redirected to the DASHBOARD and receives a confirmation message by email summarizing their selections.

As part of the TASK SIGNUP form, a volunteer indicates the window times at which they are available. It is possible for a volunteer to enter time windows that overlap. If two time windows overlap at all, they are updated to form the

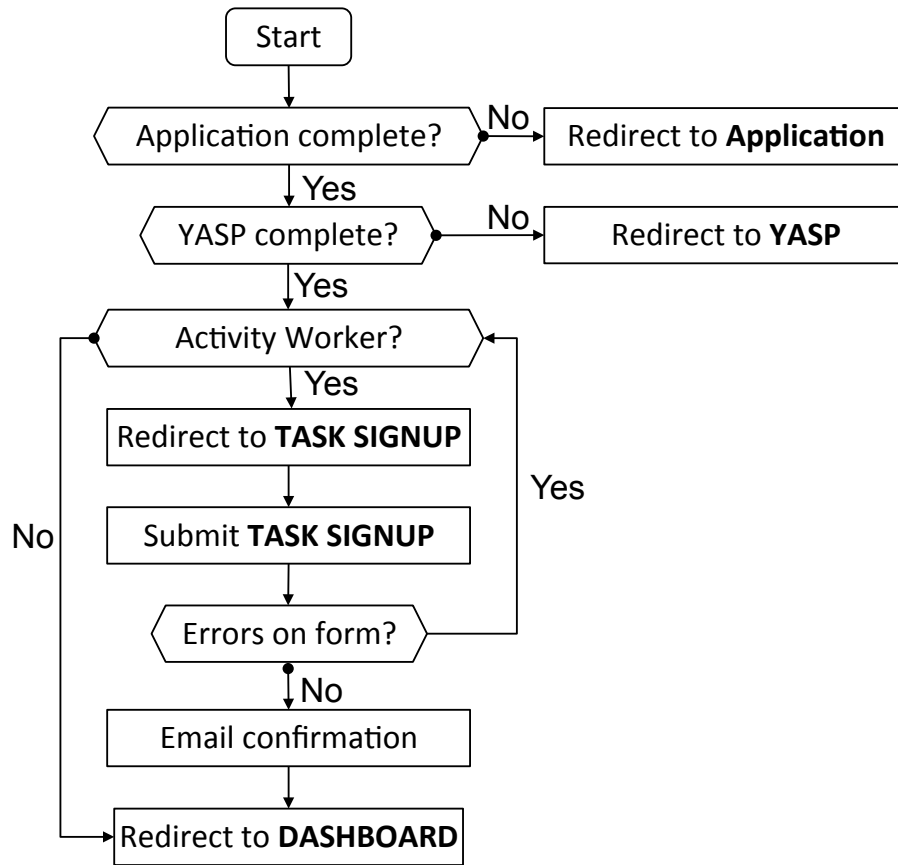


Figure 2.16: Accessing the TASK SIGNUP form

smallest encompassing time window. Therefore, every time a volunteer enters a new time window, we run the algorithm `MERGINGNEWINTERVAL` (Algorithm 1) to test whether or not the new entered window overlaps with any previously entered time window and, when so, do the merger. The volunteer has the option of removing any of the entered or computed windows.

The algorithm operates by iterating through the existing time windows. Any time window that overlaps with the new time window entered is merged with the new window and the result replaces the older window. Time windows that do not overlap with the new window are kept. The new merged window is added to the list of time windows after all existing overlapping time windows are merged with it.



---

**Algorithm 1:** MERGINGNEWINTERVAL(*IntervalList*, *NewInterval*)

---

**Input:**  $I[\cdot]$ : a zero-based index vector of  $n$  intervals where no two intervals overlap in time  
 $T$ : a new time interval  
**Output:**  $R[\cdot]$ : a zero-based index vector of  $m$  intervals where  $m \leq n + 1$  and no two intervals overlap in time

```

1  $i \leftarrow 0$ 
2  $j \leftarrow 0$ 
3 repeat
4   if  $overlap(T, I[i])$  then
5      $begin(T) \leftarrow \min(begin(T), begin(I[i]))$ 
6      $end(T) \leftarrow \max(end(T), end(I[i]))$ 
7   else
8      $R[j] \leftarrow I[i]$ 
9      $j \leftarrow j + 1$ 
10   $i \leftarrow i + 1$ 
11 until  $I[i] = nil$ 
12  $R[j] \leftarrow T$ 
13 return  $R[\cdot]$ 

```

---

The algorithm assumes that, at any point in time, all the stored time windows entered do not overlap. This means this algorithm has to be run every time a new time window is entered, or it may not be correct. We ensure that this algorithm is executed every time a new time window is entered by attaching it to the pre-save event on the database. Before the database saves a record in the time windows table, it executes this code and ensures that the new entry has no overlaps with any other already recorded time windows.

## 2.3 The DASHBOARD

DASHBOARD gives access to all the forms filled by a volunteer in the application process so that they can view them in one place and make changes and updates as they see fit.

### 2.3.1 The Rationale

We do not want a volunteer to have access to all of these forms before the application process is completed because it can be confusing and overwhelming to see all the forms at once. Therefore, we do not allow a volunteer to access the DASHBOARD until the application process is completed. Figure 2.17 shows the process for determining where to direct a volunteer when they try to access the DASHBOARD.

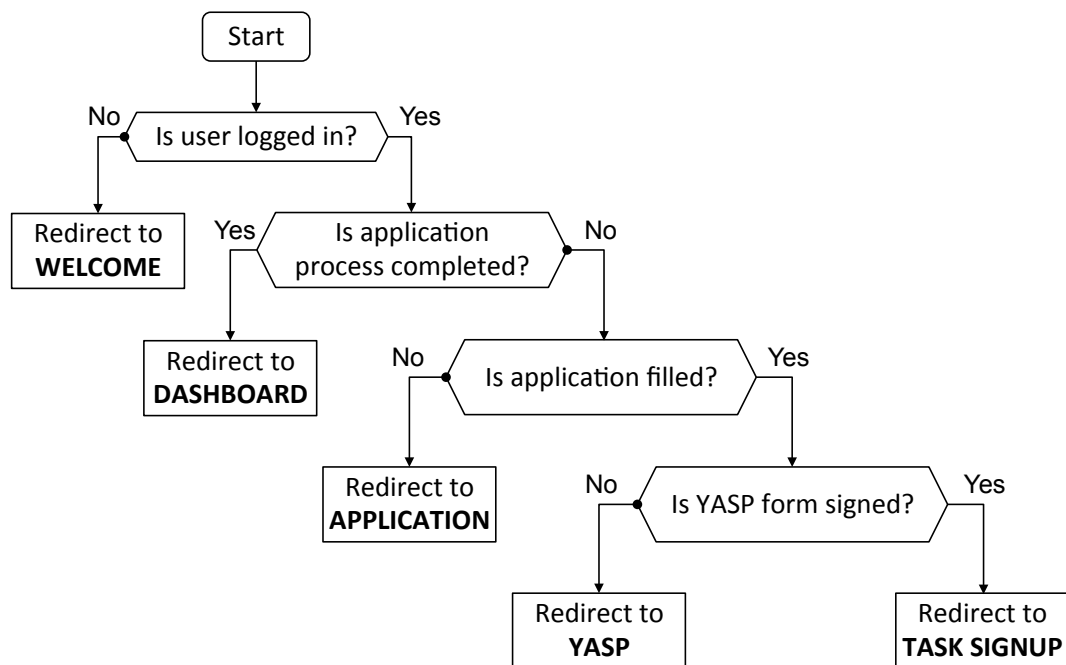


Figure 2.17: The routing process

If a volunteer is not signed in and tries to access the DASHBOARD, they are redirected to the WELCOME page with a message indicating that they must first sign in. If a volunteer tries to access the DASHBOARD while signed in but before completing the application process, the volunteer is redirected to the earliest step in the application process that has not yet been completed with a message indicating the entire application process must be completed before the DASHBOARD becomes accessible. Only once the volunteer is signed in and the entire application process is

completed does the DASHBOARD become accessible.

The process of checking whether or not the volunteer is allowed to access their DASHBOARD is executed every time a request to accessing the DASHBOARD page is made. This check is necessary because it is possible for a volunteer who can access their DASHBOARD to make a change in one of the forms that then enables options in the process that were not completed. For instance, suppose a volunteer did not indicate on the APPLICATION form to be an activity worker, which means that volunteer is not be shown the TASK SIGNUP form. After completing the application process, that volunteer can then make changes to the information entered on the APPLICATION form. If that volunteer then changes the APPLICATION form to indicate that they are an activity worker, the volunteer now needs to fill out the TASK SIGNUP form. So, although this volunteer could at one point access their DASHBOARD, after updating the APPLICATION form, their application is no longer complete. In this case, the volunteer is redirected to complete the TASK SIGNUP form before being allowed access to the DASHBOARD.

Another issue can arise when having all of the forms in the process accessible on one page. Some parts of the forms are dependent on information entered in earlier steps of the application process. For instance, on the TASK SIGNUP form, the volunteer is shown different tasks depending on the status they indicated in the APPLICATION form. On the DASHBOARD, they have access to both the APPLICATION and the TASK SIGNUP forms. The volunteer could possibly change information in either form in a way that is not consistent. They should be prevented from doing so.

If a volunteer changes their status on the APPLICATION form, the options shown in the TASK SIGNUP form may no longer be correct. If the volunteer also changes entries in the TASK SIGNUP form, both the APPLICATION and the TASK SIGNUP forms may end up totally inconsistent, which is a major problem. To solve this

problem, each form has its own submit button. This means that, though all the forms are visible at once, only one form can be changed at any given time. Anytime any form is entered or modified, the DASHBOARD page is refreshed and the changes are reflected in the options available in the other forms.

This method could potentially be confusing to a person editing forms on the DASHBOARD. A person may make changes in two different places, press a submit button, and see the changes they made on one of the forms undone. However, the tradeoff is that it is much simpler to do it the way we designed and implemented. Allowing all of the forms to be changed in parallel, the corresponding pages would have to be dynamic, updating every time a value in a field is changed, which may occur overhead and delays.

### 2.3.2 The Functionalities

After a volunteer has registered their account (Section 2.1) and completed the application process (Section 2.2), they are able to access the DASHBOARD. The DASHBOARD is shown in Figure 2.18. On this page, an accordion gives the volunteer to

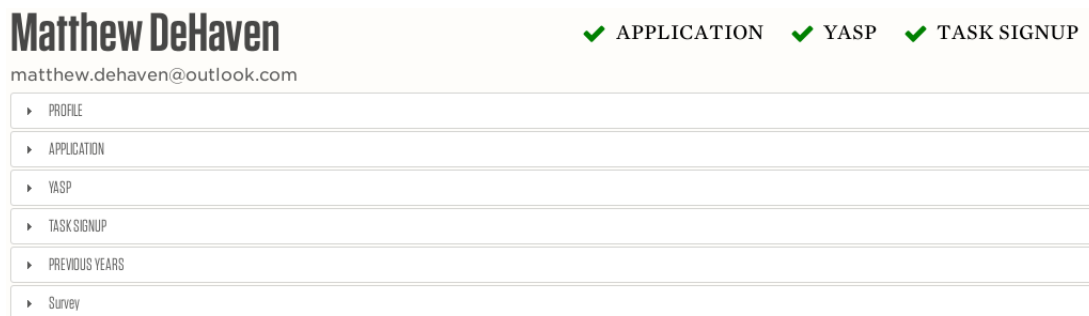


Figure 2.18: The DASHBOARD page

the various forms filled in the application process and also other functionalities as listed below:

**PROFILE** A form that allows a volunteer to modify their profile information in one of two possible forms, either as shown in Figure 2.7 or as shown in Figure 2.8, depending on the authentication system with which they registered their account.

**APPLICATION** This option allows the volunteer to update the information previously entered in the APPLICATION form shown in Figure 2.11.

**YASP** This option allows the volunteer to update the information entered in the YASP form shown in Figure 2.13.

**TASK SIGNUP** This option allows the volunteer to update the information entered in the TASK SIGNUP form shown in Figure 2.15.

**PREVIOUS YEARS** This option displays a table showing all the tasks that the volunteer had performed on in the previous Math Day events stored in the database.

**SURVEY** Finally, this option allows the volunteer to fill out an optional survey to provide feedback about the site. The information entered is stored in the database without any identifying information and a copy is sent by email to the administrator of our system. A volunteer can enter as many surveys as they wish. The values entered at one point are not shown in future attempts. The survey page was contributed by Gerald Thornton.

## Summary

In this chapter, we reviewed all the pages available to a volunteer for registering on the system, applying for a Math Day event, and viewing updating their application.

## Chapter 3

# The Administrator Interface

In this chapter, we describe the various pages, forms, and processes that make up the interface accessible by an administrator. The description is organized in two main sections:

1. The Active Admin pages, which is a temporary interface for the administrator (Section 3.1). It provides access, to the Math Day administrator, to the data in the database. However, the functionalities lack ‘flexibility.’ It constitutes a quick and temporary solution.
2. The ‘regular’ administrator pages, which is a proof of concept of the system to be built (Section 3.2).

### 3.1 Active Admin Pages

As a temporary measure to add critical administrator functionalities, we are using a Rails gem called ‘Active Admin.’ This gem provides a framework for quickly and easily creating admin pages. The downside of this solution is that there is little room for customizing the look of the pages created by this framework.

Currently, we have four pages available on the active-admin pages, shown in Figure 3.1: the dashboard, applications, users, and faculty recruiters.

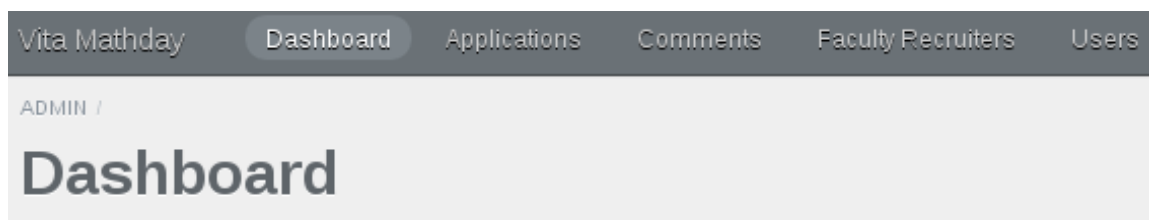


Figure 3.1: Four pages in Active Admin

The Active Admin dashboard can be found at `[url]/admin_temporary`.

### 3.1.1 Authentication

Because the Active Admin pages provide access to most of the data in the database, only the event administrator should have access to these pages. Active Admin has a built-in method for authentication. We have placed this authentication method in the file `config/initializers/active_admin.rb`. This functionality allows only users with the `admin` field set to `true` to access these pages.

### 3.1.2 Namespace

The path of Active Admin is by default set to `[url]/admin`. Because we already use this path in our pages for the admin, we had to use a different path. We set the value of the variable `config.default_namespace` to `admin_temporary` in file `config/initializers/active_admin.rb`. This initialization changes the path of the Active Admin pages from `[url]/admin` to `[url]/admin_temporary`.

### 3.1.3 Dashboard

This page is currently empty. It does not serve any purpose. It could be used to display summary information that could be useful to an administrator, such as how many people have signed up, who has recently signed up, or explanations about the various pages provided.

### 3.1.4 Applications

This page shows *all* of the applications in the database (see Figure 3.2). It includes actions to email the faculty recruiters, de-activate/re-activate applications, email individual/group volunteers, and end a date for the SOR attribute.

ADMIN / Applications

Batch Actions -

<input checked="" type="checkbox"/>	User	Year	Active	Updated At	User Type	Affiliation	Class Recruited	Faculty Recruiter	Speaker	Activity Worker	Department Display	Youth Form Date	Sor Check	Comments
<input checked="" type="checkbox"/>	<a href="#">DEHAVEN, Matthea</a>	2016	YES	March 27, 2017 19:38	Undergraduate	CVIE	MATH 106B		YES	NO	NO			
<input checked="" type="checkbox"/>	<a href="#">DEHAVEN, Matthea</a>	2015	YES	March 27, 2017 19:38	Undergraduate	CHEM	MATH 203		NO	YES	YES	November 01, 2015 00:00	November 01, 2015 00:00	
<input checked="" type="checkbox"/>	<a href="#">DEHAVEN, Matthea</a>	2017	YES	April 22, 2017 13:13	Graduate	CCFL	MATH 198H		NO	YES	NO	March 07, 2017 19:37	April 18, 2017 00:00	asdfsjjjjjjjj

Download: CSV XML JSON

Displaying 3 Applications

Figure 3.2: Applications page in Active Admin

The custom actions that can be performed on this page are available by selecting the check-boxes of the relevant applications, then clicking the batch actions check-box as seen in Figure 3.3.

### 3.1.5 Users

This page shows all the users in the database. There are currently no custom actions on this page. Nevertheless, this page is useful for viewing, editing, and deleting applicants.



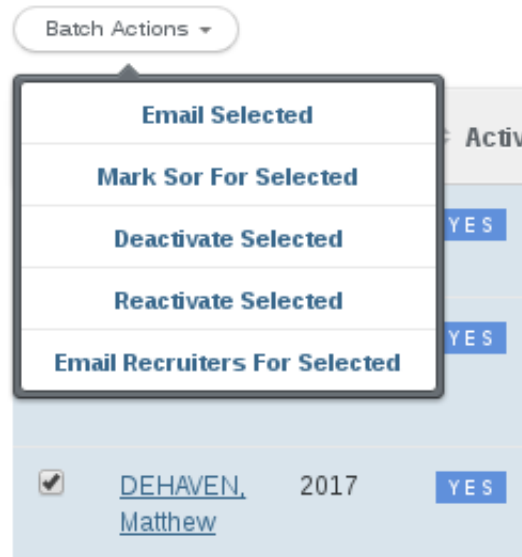


Figure 3.3: Accessing custom actions in Active Admin

### 3.1.6 Faculty Recruiters

This page currently has only one custom action, which allows the administrator to email the faculty recruiters (see Figure 3.4). This functionality can be used, for example, when the administrator needs to contact recruiters to alert them that more volunteers are needed and to solicit the help of students in their classes.

Faculty Recruiters									
Batch Actions ▾									
<input type="checkbox"/>	Id	Created At	Updated At	Priority	Approved	First Name	Last Name	Email	
<input type="checkbox"/>	39				YES				View Edit Delete
<input type="checkbox"/>	15				YES				View Edit Delete
<input type="checkbox"/>	22				YES				View Edit Delete
<input type="checkbox"/>	12				YES				View Edit Delete
<input type="checkbox"/>	23				YES				View Edit Delete
<input type="checkbox"/>	31				YES				View Edit Delete
<input type="checkbox"/>	45	November 29, 2016 19:45	November 29, 2016 19:45		YES				View Edit Delete
<input type="checkbox"/>	14				YES				View Edit Delete
<input type="checkbox"/>	21				YES				View Edit Delete
<input type="checkbox"/>	22				YES				View Edit Delete
<input type="checkbox"/>	51	November 29, 2016 21:15	November 29, 2016 21:15		YES				View Edit Delete
<input type="checkbox"/>	10				YES				View Edit Delete
<input type="checkbox"/>	1				YES				View Edit Delete
<input type="checkbox"/>	40	October 20, 2016 17:01	October 20, 2016 17:01		YES				View Edit Delete
<input type="checkbox"/>	48	November 29, 2016 20:41	November 29, 2016 20:41		YES				View Edit Delete
<input type="checkbox"/>	29				YES				View Edit Delete

Figure 3.4: Accessing faculty recruiters' page in Active Admin

## 3.2 ‘Regular’ Admin Pages

The long-term solution to the administration needs to run the event are located in the directory `[url]/admin`. The files in this directory will eventually provide all the functionalities to be used by the event administrator with a look-and-feel matching the volunteer pages.

Currently, the pages located in this namespace are temporary solutions used to enter previous years data. There are two main pages, namely, users and activities, which allow an administrator to edit users and activities, respectively. To access these two pages, a user has to pass the authentication process described below.

As with ‘Active Admin,’ the ‘regular’ admin pages should provide access to most of the data in the database and offer all the functionalities that an administrator may need. We need to restrict access to these pages to only admin users.

To limit access to only admin users, we created a base admin controller

```
app/controllers/admin_controller.rb
```

This controller provides the authentication method that checks to ensure the current user is an admin user. This controller sets that method as a ‘before action,’ which means that the authentication method will run before any requests are handled. If a user, who is not an admin user, tries to access an admin page they are redirected to WELCOME page.

To enforce this behavior on all of the admin pages, it is critical that each controller in the admin namespace (`app/controllers/admin`) inherit the `admin_controller`. If any controller in this namespace does not inherit the `admin_controller`, the pages it serves will be accessible to any user. All existing controllers in this namespace currently inherit the `admin_controller`. In the future, any newly created controllers in this namespace need to manually be set to inherit the `admin_controller` as new

controllers default to inheriting the `applications_controller`.

Currently, the method that authenticates users accessing the admin pages is commented out. We adopted this temporary solution because these pages are currently being used to enter previous years data by student workers who do not have admin accounts. However, before the site goes live, this method has to be uncommented. Again, this method is located in `app/controllers/admin_controller.rb`.

### 3.3 Creating a New Year

An important functionality for the administrator interface that is not yet implemented is creating the activities and tasks for a new year. This operation is achieved by copying the seed tables, but it is (yet) not as simple as it should be. When copying the seed tables, the associations between activities and tasks for the new year must be built. For example, when copying a task seed, the task seed must link to an activity seed. However, for the new year's tables of activities and tasks, we need to link each new task (not task seed) to the appropriate new activity (not the activity seed).

The method for copying the seeds for a new year is implemented and can be found in `app/models/new_year.rb`. This method will copy the seed tables and add all the associations between the new records. A link to call this new-year method will have to be added to the final/regular administrator interface. Until then, this method can be called from the rails console as needed. We have already used this method to create the 2017 tasks and activities, so it should not be needed again until 2018. Note that the tables of tasks and activities for 2014, 2015, and 2016 do not have the appropriate associations (because they predated the seed creation).

## Summary

In this chapter, we described two interfaces for the event administrator. The first one, a quick but temporary solution, is built with the gem ‘Active Admin.’ The second one is a prototype of the interface to be deployed. Finally, we described a method that generates new tables for activities and tasks from seeds.

# Chapter 4

## System Design

In this chapter, we review the system design.

### 4.1 Software Design

We implemented the website using Ruby on Rails. Rails uses a Model-View-Controller design pattern, which is thus the design pattern we use. The html webpages are the ‘view’ component in the design. The connections between models, views, and controllers is illustrated in Figure 4.1.

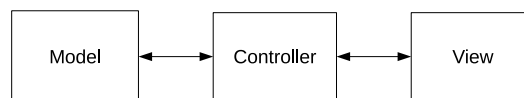


Figure 4.1: Model-View-Controller connections

When a volunteer requests information, the request is routed to a controller for that page of the browser. The controller determines which view to send to the browser. Then, the controller loads the data from the model, which connects to the database, to insert the data into the view.

When a volunteer submits information, the volunteer enters the data into a view. The data is then routed to a controller. The controller then sends the data to the model to store in the database. The controller then determines what view to send back to the page of the browser.

### 4.1.1 Program Organization

Ruby on Rails has a default structure for organizing the files of a project. Our project conforms to this default style, which makes it easy to figure where a file can be found. The directories are the following:

**app/views/** This directory contains all of the views (HTML templates) and email templates.

**app/controllers/** This directory contains all controller files, which are Ruby files.

**app/models/** This directory contains all model files, which are Ruby files.

**app/assets/stylesheets/** This directory contains all css stylesheets used in the views.

**app/assets/javascripts/** This directory contains all javascript files that are used in the views.

**app/assets/images/** This directory contains all images used in the views.

**config/** This directory contains all of the configuration files for the project.

Rails creates other directories that it uses, but the above listed directories are those containing almost all of our work.

### 4.1.2 Runtime Configurations

There are some configurations that we make accessible to the Math Day managers that they can change while the system is running. These configurations are needed because there are certain options that the organizers will be changing over time and we want them to be able to control these options easily without having to dig into files and restart the server. We use global variables to represent these configuration options. Any parts of the system that depend on these settings refer to these global variables. However, the only place these global variables are set are at startup and in the administrator's interface where the organizers will be given forms in order to change the values of those global variables.

The most obvious and important global variable is certainly the 'current' year. The organizers of the event have indicated that they would like to have control over which year the system is to run for. For instance, it may be 2017, but the organizers may want the system to run as though it is 2016. This means that everything displayed to a visitor of the site should appear as though the current year is 2016. If a volunteer goes to their dashboard, they should see the application they filled out for 2016, not 2017. This functionality is accomplished with the global variable indicating the current year. This variable is used throughout the code to determine which information from the database should be loaded.

There are two different versions of current year variable. The normal version of current year changes the appearance of the site to volunteers. The other version is the 'browse current year' variable. When this variable is set, it should change the appearance of the administrator interfaces only. This way, an administrator can look at different years without affecting what volunteers are seeing. The browse current year variable has been created, but has not been added throughout the code.

Currently, the normal version of 'current year' is being used throughout the site. This will need to be changed in the future. All of the admin pages should refer to the 'browse current year' and all the volunteer pages should refer to the normal 'current year'.

One downside of this method of storing runtime configurations is that they do not persist. If an organizer changes one of these options and the server reset overnight, when they view the site again the next day, it would not reflect the change they made. An alternative implementation that would solve this problem is a database table that stores these runtime configurations. At the moment, the options for which we are using global variables are not needed to persist, so this method works fine for now.

### **4.1.3 Account Authentication**

One interesting aspect of this project is that we provide two methods of account authentication. A volunteer can either use their My.UNL login or an email address to register. However, because of the entailed security issues, we defer to libraries to handle this work for us.

#### **4.1.3.1 My.UNL Authentication**

The authentication method used by My.UNL is the Central Authentication Service (CAS) protocol. A library called RubyCAS handles the interfacing with My.UNL's CAS system making it simple for us. This library redirects a volunteer to the My.UNL login page. After the volunteer successfully logs in, they are redirected back to the Math Day website and their the CAS library stores their My.UNL username in a session cookie.

Because we still have My.UNL authenticated volunteers fill out profile information



that is stored in the Math Day database, we have a user record in the database for all volunteers, regardless of which authentication method is used. One of the fields in the user table is the My.UNL username. When a My.UNL authenticated volunteer submits their profile form during registration, we store their My.UNL username in their user record, which allows our system to associate a user record with a My.UNL account. This is also how we differentiate between volunteers who ‘authenticate’ with My.UNL and volunteers who ‘authenticate’ with email addresses. A volunteer who ‘authenticates’ with an email address has no value in the My.UNL username field of their user record.

#### **4.1.3.2 Email Authentication**

Email authentication requires that we have a way to validate a volunteer with an email/password combination. Because it is common for people to use the same password across multiple sites, we do not want to store a volunteer’s actual password in the database. To avoid using this solution, we use a library called BCrypt. This library creates a hash value of a password and stores instead the hash value in the database. Because we never store any actual passwords, even if an untrusted source gained access to our database, the source would not have access to the volunteers’ passwords and therefore will not be able to access their accounts on other sites where they reuse the same password.

When a user enters login information, the BCrypt library hashes the entered password and compares it to the hashed value stored in the database. If the two hashed values do not match, the volunteer is not logged in. If the volunteer is logged in, their user ID is stored in the same session cookie that is used to store a volunteer’s My.UNL ID.

## 4.2 Environments

We have different (development) environments in different locations. Each different environment requires a separate database. Below are the different environments and where they reside.

**Production** This environment is the one intended for actual use with live data. At any time, all data in this environment should be real. No testing data should be introduced in this environment. The database for this environment is on the CSE server and is called `mathday`. The code for this environment is on the CSCE server.

**Evaluation** This environment is intended for use by anyone and for demo and testing purposes. Fake information is allowed here because it is only for testing the site. The database for this environment is also on the CSE server, but is a separate instance than the Production database. It is called `mathdayeval`. However, the code is on the CSCE server and is the *same* code as the Production namespace. This means the Production and Evaluation namespace are always running the *same version of the code*.

**Development** This environment is actually not a single environment. Any developer working on the project has their own development environment. Therefore, there may be multiple development environments existing at the same time. The database and code for any development environment is located on the developer's own computer.

## 4.3 Database Synchronization

When changes are made in the database of one environment, it is sometimes necessary for those changes to be moved to another environment. There are mainly two different types of changes that are made to our databases, namely, changes to the data and changes to the schema. Propagating such changes to other environments should be done differently depending on change's type.

### 4.3.1 Data Changes

The Production environment contains all of the live data, which is considered the correct data. Therefore, when copying data from one environment to another, the source should *always* be the Production environment. The trigger for copying data is different depending on the target environment. Figure 4.2 illustrates the way data should be copied between environments.

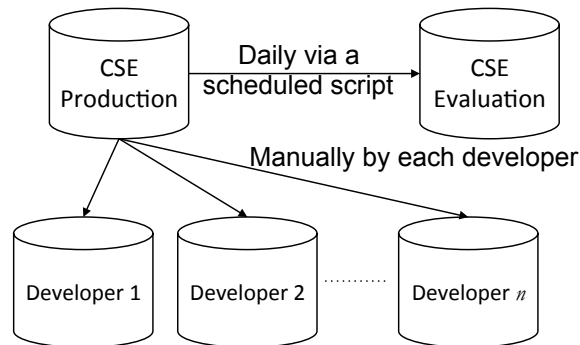


Figure 4.2: Database Data Flow

The Data in the Production environment is copied into the Evaluation environment daily via a scheduled script. This is because we often enter fake data in the Evaluation environment and we want the changes to be regularly cleared.

When copying data to a development environment, the developer executes manually the copy. This rationale is the following. As a developer is making changes and

exploring alternatives, their schemas may differ from the schemas in the development development environment. Therefore, a developer is given manual control over when to copy the Production data into the database in their environment.

Regardless of the target and source environments, the method for copying data is always the same. All databases in the three environments are mysql databases. Therefore, we always use the mysql utilities to dump and import databases when copying data.

```
# Exports production database into a sql file
# On CSE:
mysqldump -u mathday -p mathday > mathday_production.sql

# Imports production database into development database
# On developer machine with mathday_production.sql
# in current directory:
mysql -u [user-name] -p mathdaydev < mathday_production.sql
```

### 4.3.2 Schema Changes

New development always occurs in the development environments, therefore, all new schema changes come from the development environment and must be propagated to the Production environment. The process for moving a given change of a database schema from one environment to another one is shown in Figure 4.3.

Rails stores all database changes in code files and can apply the changes to the database it is connected to using a command. A schema change originates from a change in the code of one of the development environments. To make that change accessible to the other environments, developers push their changes to the code reposi-

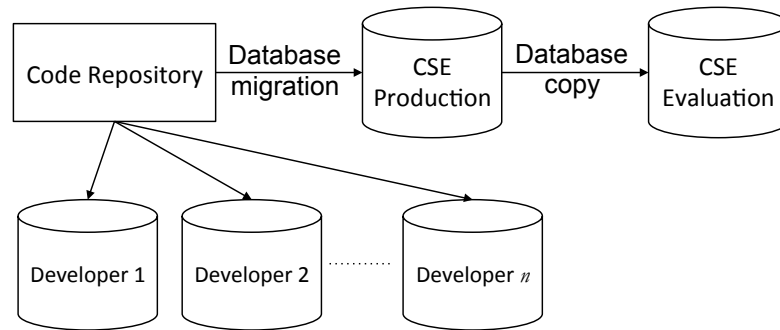


Figure 4.3: Database Schema Flow

tory. From there, all other developers can pull this code change to their local machines and run Rails' database migration command to update their database.

```

# Pulls new code and runs new migrations
# In project directory on developer machine:
git pull
rake db:migrate
  
```

Because the Production and Evaluation environments share a code base and because the Production database is copied over the Evaluation database every day, schema changes need to be applied to both the Production and Evaluation environments at the same time. First, the new code is pulled to the Production and Evaluation environments. Next, the Rails command to execute the database migration on the Production database is executed. Finally, the Production database is copied over the Evaluation database, bringing the schema changes with it.

```

# Pulls new code to production/evaluation
# and runs migration on production DB.
# On CSCE in project directory:
./update_VITA_with_master_branch.sh
rake db:migrate RAILS_ENV=production
  
```

```
# Copies production database back to evaluation
# On CSE in mathday user's home directory:
./database_dump_to_test_db.sh
```

## Summary

In this chapter, we described the software organization, the various environments used, and the synchronization between the various environment databases in the system we developed. This chapter is a must read for anyone wanting to add to the system functionalities.

## Chapter 5

# Database Documentation

This chapter provides a list and the description of all tables in the database and columns in those tables. Below, Figure 5.1 shows a diagram of all tables and their relations.

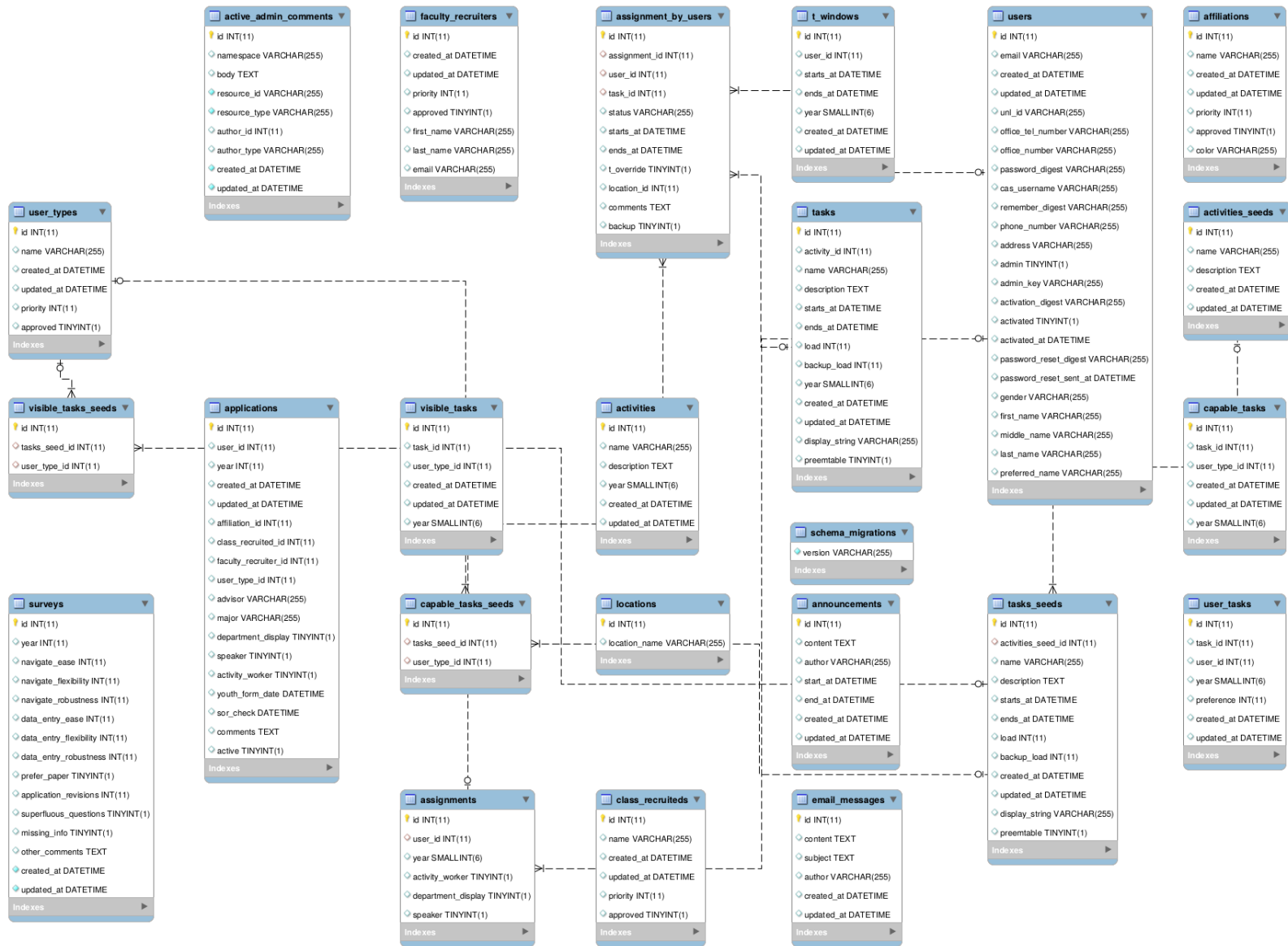


Figure 5.1: Database Diagram



## 5.1 Activities and Tasks

Below, we list all the tables pertaining to activities and tasks. Those tables are:

1. `activities_seeds`
2. `activities`
3. `tasks_seeds`
4. `tasks`
5. `user_tasks`
6. `capable_tasks_seeds`
7. `capable_tasks`
8. `visible_tasks_seeds`
9. `visible_tasks`

### 5.1.1 `activities_seeds`

The table `activities_seeds` (Table 5.1) is used as a template to create a new year of MathDay activities.

Table 5.1: Structure of table `activities_seeds`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
name	varchar(255)	Yes	NULL
description	text	Yes	NULL
created_at	datetime	Yes	NULL

Table 5.1: Structure of table `activities_seeds` (continued)

Column	Type	Null	Default
<code>updated_at</code>	<code>datetime</code>	Yes	NULL

It is mostly static but can be modified by the manager as the organization of MathDay evolves over the years. It has the following attributes:

- **name:** the activity's name.
- **description:** the activity's description

### 5.1.2 activities

The table `activities` (Table 5.2) lists all activities, their descriptions, and the year the activity took place.

Table 5.2: Structure of table `activities`

Column	Type	Null	Default
<i><b>id</b></i>	<code>int(11)</code>	No	
<code>name</code>	<code>varchar(255)</code>	Yes	NULL
<code>description</code>	<code>text</code>	Yes	NULL
<code>year</code>	<code>smallint(6)</code>	Yes	NULL
<code>created_at</code>	<code>datetime</code>	Yes	NULL
<code>updated_at</code>	<code>datetime</code>	Yes	NULL

Each activity is a set of tasks. The table `activities` has the following attributes:

- **name:** the activity's name. For example, TIMER, GRADER, SET-UP, MOD-

ERATOR.

- **description**: the activity’s description. For example, the description of SET-UP is “Sets out the PROBE I exam, bubble sheets, pencils and scratch paper before participants arrive. When students and teachers arrive, makes sure they are in their assigned rooms.”
- **year** (integer): the year of the activity. For example, 2016.

### 5.1.3 tasks\_seeds

The table `tasks_seeds` (Table 5.3) is a static table used as a template to create a new year of MathDay tasks.

Table 5.3: Structure of table `tasks_seeds`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
activities_seed_id	int(11)	Yes	NULL
name	varchar(255)	Yes	NULL
description	text	Yes	NULL
starts_at	datetime	Yes	NULL
ends_at	datetime	Yes	NULL
load	int(11)	Yes	NULL
backup_load	int(11)	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL
display_string	varchar(255)	Yes	NULL
preemtable	tinyint(1)	Yes	0

- `activities_seed_id`: The activity to which the task belongs.
- `name`: The name of the task.
- `description`: The task's description.
- `starts_at`: The time at which the task begins.
- `ends_at`: The time at which the task ends.
- `load`: The number of users which must be assigned to the task.
- `backup_load`: The number of users which must be assigned as backups to the task.
- `display_string`: The name of the task that will be displayed to the users.
- `preemptable`: Indicates whether the task can be preempted.

#### 5.1.4 tasks

The table `tasks` (Table 5.4) contains atomic tasks to which users are assigned.

Table 5.4: Structure of table `tasks`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
activity_id	int(11)	Yes	NULL
name	varchar(255)	Yes	NULL
description	text	Yes	NULL
starts_at	datetime	Yes	NULL
ends_at	datetime	Yes	NULL

Table 5.4: Structure of table tasks (continued)

Column	Type	Null	Default
load	int(11)	Yes	NULL
backup_load	int(11)	Yes	NULL
year	smallint(6)	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL
display_string	varchar(255)	Yes	NULL
preemptable	tinyint(1)	Yes	0

- **activity\_id**: The activity to which the task belongs.
- **name**: The name of the task.
- **description**: The task's description.
- **starts\_at**: The time at which the task begins.
- **ends\_at**: The time at which the task ends.
- **load**: The number of users which must be assigned to the task.
- **backup\_load**: The number of users which must be assigned as backups to the task.
- **year**: The year the task occurred.
- **display\_string**: The name of the task that will be displayed to the users.
- **preemptable**: Indicates whether the task can be preempted.

### 5.1.5 user\_tasks

The table `user_tasks` (Table 5.5) contains all the tasks that can be assigned to each user for all occurrences of the MathDay event.

Table 5.5: Structure of table `user_tasks`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
task_id	int(11)	Yes	NULL
user_id	int(11)	Yes	NULL
year	smallint(6)	Yes	NULL
preference	int(11)	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL

- `tasks_id`: The task that corresponds to this user task.
- `users_id`: The user that corresponds to this user task.
- `year`: The year the task occurred.
- `preference`: The user's preference for the task on a scale of 1 to 4, where 1 indicates a strong preference for the task and 4 indicates a strong preference against the task.

### 5.1.6 capable\_tasks\_seeds

The table `capable_tasks_seeds` (Table 5.6) is a static table used as a template to create a new year of MathDay capable tasks.

Table 5.6: Structure of table `capable_tasks_seeds`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
tasks_seed_id	int(11)	Yes	NULL
user_type_id	int(11)	Yes	NULL

- `tasks_seed_id`: The user's type.
- `user_type_id`: The task the user type can be assigned.

### 5.1.7 `capable_tasks`

The table `capable_tasks` (Table 5.7) indicates which users can be assigned to each task based on the user's type.

Table 5.7: Structure of table `capable_tasks`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
<b>task_id</b>	int(11)	Yes	NULL
<b>user_type_id</b>	int(11)	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL
year	smallint(6)	Yes	NULL

- `user_types_id`: The user's type.
- `tasks_id`: The task to which the user can be assigned.

- `year`: The year the task occurred.

### 5.1.8 `visible_tasks_seeds`

The table `visible_tasks_seeds` (Table 5.8) is a static table used as a template to create a new year of MathDay visible tasks.

Table 5.8: Structure of table `visible_tasks_seeds`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
<code>tasks_seed_id</code>	int(11)	Yes	NULL
<code>user_type_id</code>	int(11)	Yes	NULL

- `user_types_id`: The user's type.
- `tasks_id`: The task visible to the user type.

### 5.1.9 `visible_tasks`

The table `visible_tasks` (Table 5.9) indicates which tasks are visible to users based on the user's type.

Table 5.9: Structure of table `visible_tasks`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
<code>task_id</code>	int(11)	Yes	NULL
<code>user_type_id</code>	int(11)	Yes	NULL
<code>created_at</code>	datetime	Yes	NULL



Table 5.9: Structure of table `visible_tasks` (continued)

Column	Type	Null	Default
<code>updated_at</code>	<code>datetime</code>	Yes	NULL
<code>year</code>	<code>smallint(6)</code>	Yes	NULL

- `user_types_id`: The user's type.
- `tasks_id`: The task visible to the user type.
- `year`: The year the task occurred.

## 5.2 Volunteers

Below, we list all the tables pertaining to volunteers. Those tables are:

1. `user_types`
2. `users`
3. `affiliations`
4. `applications`
5. `t_windows`
6. `assignment_by_users`
7. `assignments`
8. `class_recruiteds`
9. `faculty_recruiters`

### 5.2.1 user\_types

The table `user_types` (Table 5.10) groups users based on the users' affiliation at UNL.

Table 5.10: Structure of table `user_types`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
name	varchar(255)	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL
priority	int(11)	Yes	NULL
approved	tinyint(1)	Yes	0

- `name`: The name of the user type (e.g., Undergraduate, Graduate, Staff, etc.)
- `priority`: Determines the ordering of the user types.

### 5.2.2 users

The table `users` (Table 5.11) contains all users that use the MathDay website.

Table 5.11: Structure of table `users`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
<b>email</b>	varchar(255)	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL

Table 5.11: Structure of table users (continued)

Column	Type	Null	Default
unl_id	varchar(255)	Yes	NULL
office_tel_number	varchar(255)	Yes	NULL
office_number	varchar(255)	Yes	NULL
password_digest	varchar(255)	Yes	NULL
<b>cas_username</b>	varchar(255)	Yes	NULL
remember_digest	varchar(255)	Yes	NULL
phone_number	varchar(255)	Yes	NULL
address	varchar(255)	Yes	NULL
admin	tinyint(1)	Yes	0
admin_key	varchar(255)	Yes	NULL
activation_digest	varchar(255)	Yes	NULL
activated	tinyint(1)	Yes	0
activated_at	datetime	Yes	NULL
password_reset_digest	varchar(255)	Yes	NULL
password_reset_sent_at	datetime	Yes	NULL
gender	varchar(255)	Yes	NULL
first_name	varchar(255)	Yes	NULL
middle_name	varchar(255)	Yes	NULL
last_name	varchar(255)	Yes	NULL
preferred_name	varchar(255)	Yes	NULL

- `first_name`: The user's first name.
- `middle_name`: The user's middle name.

- `last_name`: The user's last name.
- `preferred_name`: The user's preferred first name.
- `gender`: The user's gender.
- `email`: The user's email address.
- `unl_id`: The user's UNL ID number.
- `office_tel_number`: The user's office telephone number.
- `office_number`: The user's office number.
- `password_digest`: The hash of the user's password.
- `cas_username`: The user's CAS username. Populated only if the user used their my.UNL account to register.
- `remember_digest`: The hash code used to match a remember me cookie to a user. This allows a user to return to the site and still be logged in.
- `phone_number`: The user's cell phone number.
- `address`: The user's home address.
- `admin`: Indicates if the user is an administrator.
- `admin_key`: Not sure what this does.
- `activation_digest`: The hash code used to match a activation link to a user. This allows a user to authenticate their account via an email link.
- `activated`: Indicates if the user has activated his or her account.
- `activated_at`: Indicates when the user activated his or her account.

- `password_reset_digest`: The hash code used to match a password reset link to a user. This allows a user to reset their password from an email link.
- `password_reset_sent_at`: The time at which the user requested a password reset.

### 5.2.3 affiliations

The table `affiliations` (Table 5.12) gives the institutional affiliation of each volunteer.

Table 5.12: Structure of table `affiliations`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
name	varchar(255)	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL
priority	int(11)	Yes	NULL
approved	tinyint(1)	Yes	0
color	varchar(255)	Yes	NULL

The table `affiliations` has the following attributes:

- `name`: the affiliation's name (e.g., Math Department, Computer Science Department, Nebraska Wesleyan University, and Mutual of Omaha).
- `priority`: groups together similar affiliations in order to determine their ordering when displayed on the interface.

- **approved:** indicates if the administrator has approved an affiliation entry. If approved, the affiliation will appear in the list of affiliations displayed to the user when he/she applies for an instance of MathDay.
- **color:** The background color for this option in a select box dropdown list.

### 5.2.4 application

The table `applications` (Table 5.13) stores all the applications submitted to the system for all users and for all years.

Table 5.13: Structure of table `applications`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
user_id	int(11)	Yes	NULL
year	int(11)	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL
affiliation_id	int(11)	Yes	NULL
class_recruited_id	int(11)	Yes	NULL
faculty_recruiter_id	int(11)	Yes	NULL
user_type_id	int(11)	Yes	NULL
advisor	varchar(255)	Yes	NULL
major	varchar(255)	Yes	NULL
department_display	tinyint(1)	Yes	0
speaker	tinyint(1)	Yes	0
activity_worker	tinyint(1)	Yes	1

Table 5.13: Structure of table applications (continued)

Column	Type	Null	Default
youth_form_date	datetime	Yes	NULL
sor_check	datetime	Yes	NULL
comments	text	Yes	NULL
active	tinyint(1)	Yes	1

It stores an applicant's information that can vary from year to year such as the role (e.g., activity worker and/or speaker), the major of study, and the dates at which the YASP form was signed. The table has the following attributes:

- **users\_id**: the user who submitted the application.
- **user\_type\_id**: the user's type (e.g., Undergraduate, Graduate, Staff, etc.)
- **year**: the year for which the user is applying
- **affiliation\_id**: the department, university, business, or any other affiliation to which the user belongs
- **class\_recruited\_id**: the class from which the user was recruited
- **faculty\_recruiter\_id**: the faculty member who recruited the user
- **advisor**: the name of the user's advisor, which is a Math instructor.
- **major**: the user's major, which can change from year to year.
- **activity\_worker**: indicates if the user is an activity worker for the MathDay event.
- **speaker**: indicates if the user is a speaker for the MathDay event.

- `department_display` (boolean, default: false): indicates if the user will present a department display at the MathDay event.
- `youth_form_date`: the date at which the user signed the Youth Activity Safety Policy Activity Worker Guidelines form.
- `sor_check`: the date at which the user's status has been checked on the Sex Offender Registry.
- `comments`: any comments the user may enter.

### 5.2.5 `t_windows`

The table `t_windows` (Table 5.14) contains time windows that indicate when a user is available to be assigned to tasks.

Table 5.14: Structure of table `t_windows`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
user_id	int(11)	Yes	NULL
starts_at	datetime	Yes	NULL
ends_at	datetime	Yes	NULL
year	smallint(6)	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL

- `users_id`: The user to whom the time window belongs.
- `starts_at`: The start time of the time window.



- `ends_at`: The end time of the time window.
- `year`: The year to which the time window belongs.

### 5.2.6 `assignment_by_users`

The table `assignment_by_users` (Table 5.15) contains assignments made for each user.

Table 5.15: Structure of table `assignment_by_users`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
assignment_id	int(11)	Yes	NULL
user_id	int(11)	Yes	NULL
task_id	int(11)	Yes	NULL
status	varchar(255)	Yes	executed
starts_at	datetime	Yes	NULL
ends_at	datetime	Yes	NULL
t_override	tinyint(1)	Yes	0
location_id	int(11)	Yes	NULL
comments	text	Yes	NULL
backup	tinyint(1)	Yes	0

- `assignment_id`: The assignment for the user.
- `user_id`: The user the assignment is for.
- `task_id`: The task the user is assigned.

- **status**: The status of the assignment (Was is completed or not?).
- **starts\_at**: When the assignment starts.
- **ends\_at**: When the assignment ends.
- **t\_override**: Not sure what this is for.
- **location\_id**: Where the assignment is located.
- **comments**: Any additional comments.
- **backup**: Is this assignment a backup?

### 5.2.7 assignments

The table `assignments` (Table 5.16) is used to group all tasks assigned to users for each MathDay event.

Table 5.16: Structure of table assignments

Column	Type	Null	Default
<i>id</i>	int(11)	No	
user_id	int(11)	Yes	NULL
year	smallint(6)	Yes	NULL
activity_worker	tinyint(1)	Yes	1
department_display	tinyint(1)	Yes	0
speaker	tinyint(1)	Yes	0

- **user\_id**: The user who was assigned (a) task(s).
- **year**: The year the assignment occurred.

- **activity\_worker**: Indicates if the user is an activity worker for the MathDay event
- **speaker**: Indicates if the user is a speaker for the MathDay event.
- **department\_display**: Indicates if the user will present a department display at the MathDay event.

### 5.2.8 class\_recruiteds

The table `class_recruiteds` (Table 5.17) contains the classes from which users are recruited.

Table 5.17: Structure of table `class_recruiteds`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
name	varchar(255)	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL
priority	int(11)	Yes	NULL
approved	tinyint(1)	Yes	0

- **name**: The name of the class.
- **priority**: Determines the ordering of the class recruiters.
- **approved**: Indicates if the administrator has approved a class recruiter entry. If approved, the class recruiter will appear in the list of class recruiters displayed to the user when he or she applies for an instance of MathDay.

### 5.2.9 faculty\_recruiters

The table `faculty_recruiters` (Table 5.18) is a table of faculty members who recruit users for MathDay.

Table 5.18: Structure of table `faculty_recruiters`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL
priority	int(11)	Yes	NULL
approved	tinyint(1)	Yes	0
first_name	varchar(255)	Yes	NULL
last_name	varchar(255)	Yes	NULL
email	varchar(255)	Yes	NULL

- `first_name`: The first name of the faculty member.
- `last_name`: The last name of the faculty member.
- `email`: The faculty member's email address.
- `priority`: Determines the ordering of the faculty members.
- `approved`: Indicates if the administrator has approved a faculty member entry. If approved, the faculty member will appear in the list of faculty recruiters displayed to the user when he or she applies for an instance of MathDay.

## 5.3 Manager

Below, we list all the tables pertaining to the manager’s functionalities. Those tables are:

1. `announcements`
2. `locations`
3. `surveys`
4. `email_messages`

### 5.3.1 announcements

The table `announcements` (Table 5.19 ) stores the messages from the manager that are displayed on the website’s login.

Table 5.19: Structure of table `announcements`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
content	text	Yes	NULL
author	varchar(255)	Yes	NULL
start_at	datetime	Yes	NULL
end_at	datetime	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL

- **content**: the announcement’s text content.

- **author**: the announcement's author
- **start\_at**: the time at which the announcement will begin being displayed on the website's homepage
- **end\_at**: the time at which the announcement will stop being displayed on the website's homepage

### 5.3.2 locations

The table `locations` (Table 5.20) contains the locations of tasks.

Table 5.20: Structure of table `locations`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
location_name	varchar(255)	Yes	NULL

- **location\_name**: The name of the location.

### 5.3.3 surveys

The table `surveys` (Table 5.21) contains the responses of surveys filled out by users of the system.

Table 5.21: Structure of table `surveys`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
year	int(11)	Yes	NULL
navigate_ease	int(11)	Yes	NULL

Table 5.21: Structure of table surveys (continued)

Column	Type	Null	Default
navigate_flexibility	int(11)	Yes	NULL
navigate_robustness	int(11)	Yes	NULL
data_entry_ease	int(11)	Yes	NULL
data_entry_flexibility	int(11)	Yes	NULL
data_entry_robustness	int(11)	Yes	NULL
prefer_paper	tinyint(1)	Yes	NULL
application_revisions	int(11)	Yes	NULL
superfluous_questions	tinyint(1)	Yes	NULL
missing_info	tinyint(1)	Yes	NULL
other_comments	text	Yes	NULL
created_at	datetime	No	
updated_at	datetime	No	

- year: The year the survey was filled out.
- navigate\_ease: How easy it is to navigate the site on a scale of 1-4.
- navigate\_flexibility: How flexible the site navigation is on a scale of 1-4.
- navigate\_robustness: How robust the site navigation is on a scale of 1-4.
- data\_entry\_ease: How easy it is to enter data on a scale of 1-4.
- data\_entry\_flexibility: How much flexibility there is in data entry on a scale of 1-4.
- data\_entry\_robustness: How robust the data entry is on a scale of 1-4.

- `prefer_paper`: Would you prefer to fill forms out on paper instead?
- `application_revisions`: How many times the user revised their application.
- `superflous_questions`: Where there any superfluous questions?
- `missing_info`: Was there any missing information?
- `other_comments`: Any additional comments.

### 5.3.4 email\_messages

The table `email_messages` (Table 5.22) contains emails sent from MathDay administrators.

Table 5.22: Structure of table `email_messages`

Column	Type	Null	Default
<i>id</i>	int(11)	No	
content	text	Yes	NULL
subject	text	Yes	NULL
author	varchar(255)	Yes	NULL
created_at	datetime	Yes	NULL
updated_at	datetime	Yes	NULL

- `content`: The content of the email.
- `subject`: The subject of the email.
- `author`: The name of the email's author.



# Chapter 6

## Conclusions

In this thesis, we described the various components of the system that we designed and built for the management of volunteers of the Math Day event. The system is ready for deployment to receive applications for the Math Day event of November 2017. We have implemented and tested all the functionalities to receive users' applications. Additionally, has been included a temporary administrator interface that provides some basic functionalities until the long-term solution is refined and can be deployed.

Naturally, there is still work that needs to be done. The long-term administrator interface has to be created to replace the temporary one. Additionally, an automated solver and an interactive one to assign volunteers to tasks through an interactive, graphical interface need to be implemented. However, all functionalities needed for volunteers to register are available for 2017 Math Day.