CS 316 Final Project: Study Group Scheduler

Brandon Chao

David Hemminger

Peter Yom

**The Problem**

Our motivation for creating the Study Group Scheduler was to allow students to find and study with other students that they normally would not interact with. We found that there were not any simple solutions available for students who wanted to study with classmates that they did not previously know. There are applications such as Doodle that allow for easy event scheduling. However, this is more geared towards groups that already have a list of expected participants. On the other hand, there are also applications such as Piazza based around a registered class of students. However, we felt that the main functionality of this centered around asking questions and interacting the professors and students in the same class. Our goal with the Study Group Scheduler was to combine the ease of use and scheduling capabilities of Doodle with the classroom-centered approach of Piazza.

Our application offers a simple way for students to request and find other students interested in studying the same topics. Users are able to submit a request with their preferences for studying including available times and topics they would like to study. When enough users submit a request with matching preferences to form a group, an email is sent to all members of the group with the contact information of each member.

**The System**

Our Study Group Scheduler uses PHP and a MySQL database to run. The PHP files and mySQL database are hosted on a private server. The front end UI uses Twitter’s Bootstrap JS and CSS framework.

For web development, we were deciding between PHP and Python. In the end, we decided to use PHP because of its larger and more extensive user base and supposed ease of learning. Furthermore, its prevalence on the web made support for problems very accessible and hosting support easy to find as well.

For our choice of databases, we had many options ranging from NoSQL document based databases such as Mongo, to standard relational databases like PostGreSQL and MySQL. We chose to use a relational database not only because these were what we studied most extensively in class, but also because the structured nature of our data naturally fit better in a relational database. The choice between PostGreSQL, which we had used in class, and MySQL, came down to ease of use and a few other factors. We felt that even though PostGreSQL contained many advanced features, the features MySQL contained would easily satisfy our needs for this projects and it would still be easiest to set up on our server.

In our first two milestones, our application was developed on the front-end using bare HTML generated from our PHP files. However, as we progressed on our final project, we decided to incorporate Twitter’s Bootstrap library for its CSS styling and JS functionalities.

Other technologies used is a HostGator server on which our application is run. On the control panel of the server we were able to debug SQL queries into the database as well as manage how the files are stored. This is also where we run a cronjob in order to periodically run matchRequests.php to match up users and send emails out to created groups.

**Detailed Description of the System**

When the user first accesses the website, they are directed to the main index page. From there, the user is presented with options to register as a new user or to login as a current user. Selecting ‘Register’ in the nav bar will direct the user to ‘register.php’. There, the user will input their first name, last name, email, and password. As an added measure of security, all user passwords will be salted. With the user’s inputs and the salt hash, a new tuple is added to the ‘Person’ table.

When the user returns to the main index, they may then login with their login information. Upon logging in, the user is presented with ‘hub.php’. There, the user will see side navigation tabs:

There are two components in the ‘Create’ tab. There, a user can create a new topic or request a new meeting. New topics are added by simply inserting a new tuple based on the user’s inputs of topic name and description. When creating a new request, the user selects a topic and a group size preference. Should the topic dropdown not contain the desired topic, the user can create a new topic below the request creation. When the user submits their request, a tuple is added to the ‘Request’ table. This tuple’s primary key, ‘rid’, will be used to match up the next sequence of inserts. The user is then directed to ‘addDateTime.php’. There, the options for date and time are made available. Here, the user selects what days and when they are available to meet. After submitting, the user will then have the option to return home or add more times. Adding more times means that more dates and times can be added to the same request. Should the user want to create a new and unique request, they must return home and create a new request from the ‘Create’ tab.

The ‘Topics’ tab lists all of the available study topics in a table. It is implemented with a simple SQL query to grab all of the names of topics in the ‘Topic’ table. Should the user desire to add a new topic, they would navigate to the ‘Create’ tab and create a new topic.

The ‘Scheduled’, ‘Pending’, and ‘Past’ sections are all variations of the same formula. These tabs display meetings that the user is associated with based on the status of the request. Requests that are pending will appear in ‘Pending’, requests that have been scheduled but have not yet taken place will be in ‘Scheduled’, and requests for meetings that have completed will appear in ‘Past’. In ‘Pending’ and ‘Scheduled’, the user has the option to delete requests. In ‘Scheduled’ and ‘Past’, the user can view the other users in their meeting group by clicking on rows of the table.

In order to allow for our matching algorithm to handle larger topic sizes (such as students taking introductory Economics courses), we decided to make the algorithm greedy. For each topic, the algorithm ranks all time slots by how many people wish to study for that topic during that time slot and are willing to be a large study group. It gets this information from the views LargeRequestedTimeSlots, MediumRequestedTimeSlots, and SmallRequestedTimeSlots, which are created with createStudyGroups.sql. Note that each of these views makes use of the view PersonBusyDuringTimeSlot, which records when someone is already scheduled for a meeting in a particular time slot. This allows for the three views to make sure that they don’t count someone as being available to study during a time slot when they’re actually in a meeting.

If the time slot with the greatest number of people satisfying these conditions has enough people to form a large group, the algorithm will then form a large group out of a subset of these people. If the time slot with the greatest number of people does not have enough to form a group, then it must be that no time slot has enough people and the algorithm moves on. After all possible large groups for a topic have been scheduled the algorithm moves on to medium groups and finally small groups.

Here are the database tables with keys declared and brief descriptions:

* Topic(tid, name, description)
  + contains topics for studying with descriptions
* Person(pid, first\_name, last\_name, salt, password, email)
  + contains tuple for each user with their user data
  + passwords are salted
* Request(rid, pid, tid, time, status)
  + contains a lists of requests for studying. Each request consists of the person doing the requesting, the topic requested, the time that the request was made, and the status of the request (i.e. whether or not the request has been matched up with others to create a meeting).
* TimeSlot(tid, time\_slot\_date, time\_slot\_time)
  + contains the time slots available for studying
* RequestTimes(rid, tid)
  + records the time slots that each request lists as available for meeting during
* Meeting(mid, topic, meeting\_time)
  + lists the meetings that have been scheduled
* PersonAttendingMeeting(pid, mid)
  + records which people are attending the meetings

**Assumptions**

We’re assuming that enough people use our application with enough frequency for the algorithm to be able to regularly match up groups of all three sizes. Since we’re targeting our application at those wishing to study particular topics, we’re assuming that a particular topic will never have more people who wish to study it than the number of people in one of Duke’s largest classes. Finally, we’re assuming that “Morning”, “Afternoon”, etc. are specific enough for our purposes and that one time slot is sufficient for a meeting (e.g. nobody will want to meet for both the morning and afternoon).

**Evaluation**

Though rudimentary, our application effectively serves the niche we targeted. Our goal was to create a system that allows users to simultaneously find new study partners and arrange meeting times, and our application performs this function well. Furthermore, to our knowledge there is no other software available to Duke students that allows for this.

All of our code runs efficiently for data sets according to the assumptions of our project, in particular assuming that class sizes do not become too large and timeslots are limited to morning, afternoon, evening, and night over the next year. One weakness of our design, however, is that we would have to make changes to our design to accommodate for certain expansions beyond our original assumptions. Although our matching algorithm would be able to accommodate larger class sizes due to its greedy nature, we would likely have to make changes in order to allow for more timeslots. Another weakness is that since our matching algorithm is greedy, it is possible for it to match non-optimally in such a way that a small number of people are not matched up, but this is very likely to occur under our assumption that our application has sufficiently many users in each topic.

Altogether we are happy to have create an application that effectively serves an original function, and we believe there are a number of exciting directions to expand our application beyond its original goals.

**Conclusions and Future Work**

We were successful in creating the application that we set out to create. Although the Study Group Scheduler may not be extensively used, the proof of concept and the process of creating it was very enlightening. Our application simply solves the problem of how to connect with other students in a class that the user is unfamiliar with initially. However, there are many other features that we would have liked to implement given more time.

First of all, we wanted to perform login validation using school email addresses and then separate the application based on the university the user attends. Also, we also considered the idea of having professors or admin users being the ones to create the topics for the classes. Along with this, students would need to be registered in a class to create study requests for that particular topic. Next, another feature is to add more extensive study preferences when creating a request. This could include variables such as preferred study locations, academic year, study habits, etc. Similarly, this also includes more specific requested study times instead of how we break it up to morning, afternoon, evening, and night currently. Finally, if more variables were added in the requests, our matching algorithm would need to be updated to account for and prioritize these different variables. The efficiency of our matching algorithm could also be improved.