

March 2014

# ProjectSpot

Anthony Michael Fisher  
*Worcester Polytechnic Institute*

Madalyn E. Coryea  
*Worcester Polytechnic Institute*

Follow this and additional works at: <https://digitalcommons.wpi.edu/mqp-all>

---

## Repository Citation

Fisher, A. M., & Coryea, M. E. (2014). *ProjectSpot*. Retrieved from <https://digitalcommons.wpi.edu/mqp-all/3601>

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact [digitalwpi@wpi.edu](mailto:digitalwpi@wpi.edu).

# ProjectSpot

## A WPI Computer Science MQP Finding Website

A Major Qualifying Project Report:

submitted to the faculty of the

**WORCESTER POLYTECHNIC INSTITUTE**

in partial fulfillment of the requirements for the

Degree of Bachelor Science

by:

---

Madalyn Coryea

---

Anthony Fisher

Date: Friday, March 7, 2014

Approved:



# WPI

---

Professor David C. Brown, Major Advisor

## Abstract

This project is about how we created a website for students in the Computer Science Department at Worcester Polytechnic Institute (WPI). For this project, we gathered information about the process of finding a Major Qualifying Project. We researched how to develop a system that would allow students to interact with each other to find and form project groups. We then constructed a system using PHP, the CodeIgniter framework, a MySQL database, Javascript, HTML, and CSS. We set up our project to work with the Central Authentication Service that WPI provides, in order to allow users to login to the website we created. We conducted user tests and a heuristic evaluation in order to analyze and improve our project. We made our project available to the members of the Computer Science department so that it could be used to coordinate groups for projects.

## **Acknowledgements**

We would like to thank Michael Voorhis for his contributions to this project. We would also like to thank our advisor, Professor David C. Brown; particularly for his guidance and dedication. We are especially thankful to all the students who volunteered to participate in user studies. Lastly, we would like to thank the Computer Science Department at Worcester Polytechnic Institute, for providing us with the resources and the environment to conduct this project.

## Table of Contents

List of Figures .....	6
Terminology.....	7
1 Introduction.....	10
1.1 Problems with the MQP-Finding Process.....	10
1.2 Problem Statement.....	10
1.3 Requirements .....	11
2 Background.....	13
2.1 The MQP at WPI .....	13
2.1.1 The Current Process.....	13
2.1.2 What Resources Exist .....	14
2.2 Recommender Systems.....	14
2.2.1 Algorithms Used in Content-Based Filters.....	15
2.2.2 Content-Based Filtering.....	16
2.3 Matching Systems.....	16
2.3.1 Concerns with Matching Systems.....	16
2.3.2 Matching System Algorithms .....	17
2.3.3 Group Finding Tools for Finding an MQP .....	18
2.4 Web Development – Tools & Technologies Used .....	18
2.4.1 Model View Controller (MVC) .....	18
2.4.2 Client Side – HTML, CSS, Javascript .....	20
2.4.3 Server Side – PHP.....	20
2.4.4 CodeIgniter .....	21
2.5 Databases .....	22
2.5.1 Types of Databases .....	22
2.5.2 Structured Query Language (SQL).....	23
2.5.3 Database Structure Approaches .....	24
2.6 Source Control - Git.....	25
2.7 Central Authentication Service (CAS).....	28
3 Methodology .....	29
3.1 Collecting User Data.....	29
3.2 Designing the Interface (based on User Data) .....	29
3.3 Designing the System .....	30
3.3.1 CodeIgniter: Choosing a Framework.....	30

3.3.2 Database.....	31
3.4 Implementing the Interface.....	31
3.5 Implementing the System .....	32
3.6 Implementing CAS (Central Authentication Service) .....	32
3.7 Making Improvements .....	33
3.7.1 Heuristic Evaluation.....	33
3.7.2 User Study of Tasks through the Interface .....	33
3.8 Group User Study of Functioning System and Marketing ProjectSpot.....	33
4 Analysis of the Requirements .....	35
4.1 Evaluating the Current Process.....	35
4.2 Collecting Initial User Data .....	35
4.4 Design Requirements .....	35
4.3 List of Requirements.....	36
5 Design .....	38
5.1 Interface Design .....	38
5.1.1 Landing Page .....	39
5.1.2 Dashboard .....	40
5.1.3 Profile Page.....	41
5.1.4 Group Page.....	43
5.1.5 Invitations Page.....	46
5.1.6 Find Page .....	47
5.1.7 Admin Pages .....	48
5.2 Architecture Design .....	49
5.3 Database Design.....	49
6 Implementation .....	52
6.1 Implementing the Interface .....	52
6.2 Implementing the System's Architecture.....	52
6.3 Implementing the Database.....	54
6.4 Implementing CAS (Central Authentication Service) .....	55
7 Results & Analysis (Design Evaluation) .....	56
7.1 Interface and Usability .....	56
7.2 Results of User Study of Tasks.....	57
7.3 Final User Study .....	58
7.4 Results of Final User Study .....	59

8 Conclusions.....	60
8.1 Future adoption by the Computer Science Department.....	60
8.2 Future Work.....	60
8.2.1 Recommender System.....	60
8.2.2 Improved Notification System.....	61
8.2.3 The Cycle of ProjectSpot.....	61
8.3 Our Experience with the Project.....	62
8.3.1 Skills and Tools Used.....	63
8.3.2 Skills and Tools Learned.....	63
8.4 Access and Use of the System.....	64
9 References.....	65
Appendix A: Initial Survey Questions.....	68
Appendix B: Initial Survey Responses.....	70
Appendix C: Database Designs.....	76
Appendix D: UML Sequence Diagrams.....	77
Appendix E: User Study on Tasks in Interface Questionnaire.....	80
Appendix F: Results from User Study of Tasks through the System.....	85
Appendix G: Group User Study Questionnaire.....	88
Appendix H: Results from Group User Study.....	92
Appendix I: Available Interests for Users and Groups.....	94
Appendix J: Screen Shots of Admin Pages.....	96

## List of Figures

Figure 1: Model-View-Controller Design Pattern [CodeProject.com, 2008].....	19
Figure 2: A Distributed Source Control Code Base [Chacon, 2009].....	26
Figure 3: A Centralized Source Control Code Base [Chacon, 2009] .....	26
Figure 4: Flow Diagram of ProjectSpot's Tasks .....	36
Figure 5: ProjectSpot Landing Page .....	39
Figure 6: ProjectSpot Dashboard .....	41
Figure 7: The Profile Page for a User .....	41
Figure 8: The Edit Profile Page for a User .....	42
Figure 9: Initial Group Page for a Student User .....	44
Figure 10: Group Page for an Existing Group .....	44
Figure 11: Edit Group Page for a Group.....	45
Figure 12: Invitations Page for a User in a Group .....	47
Figure 13: The Find Page in ProjectSpot.....	48
Figure 14: ProjectSpot's Database Structure.....	76
Figure 15: UML Diagram for Editing a Group.....	77
Figure 16: UML Diagram for Creating New Group.....	78
Figure 17: UML Diagram for Viewing a Group.....	78
Figure 18: UML Diagram for Editing and Viewing a Profile .....	79
Figure 19: Admin Panel Main Page.....	96
Figure 20: Page for Adding and Removing System Administrators .....	96
Figure 21: Admin Page for Deleting Users.....	97
Figure 22: Admin Page for Deleting Groups .....	97
Figure 23: Admin Page for Managing Resources Listed in ProjectSpot .....	98
Figure 24: Admin Page for Editing a Resource .....	98
Figure 25: Admin Page for Adding a New Resource .....	98
Figure 26: Admin Page for Managing Important Dates in ProjectSpot.....	99
Figure 27: Admin Page for Adding a New Important Date.....	99
Figure 28: Admin Page for Editing an Existing Important Date .....	99



## Terminology

**WPI:** Worcester Polytechnic Institute, a four-year private university in Worcester, Massachusetts, USA

**MQP:** Major Qualifying Project, a project done at WPI, usually completed in a student's senior year. It is in the student's major field, and must be completed prior to graduating. [WPI, 2014]

**CS:** Computer Science

**HTML:** Hypertext Markup Language, a markup language that is used to display content on the web. [WC3, 2014]

**Front-End:** The forward facing part of a website. This is what most users will see.

**Back-End:** Management pages of the website. This is what administrators of the website will see.

**Database:** a collection of data organized in such a way that helps developers access the data that they store. [Wikipedia, 2014]

**SQL:** Structured Query Language. A Language that is used to interface with the database. [Encyclopedia Britannica, 2013]

**MVC:** Model View Controller, a way to organize and design programs in an object-oriented fashion. [EllisLab Inc., 2014]

**Model:** Handles the data structures of an MVC Application. The model usually only interacts with the Database.

**View:** Handles the information being presented to the user. Considered to be the "front-end code" that generates a display that the user will see. The views in this system are represented as webpages.

**Controller:** Code that communicates between the view and the models to generate dynamic content.

**Dynamic:** Values or content that changes based on differing scenarios.

**Static:** Values or content that stays the same consistently throughout a system.

**Git:** A form of distributed source control, released in 2005.

**SVN:** A centralized form of source control, released in 2000.

**Repository:** A place where a large amount of source code is stored.

**Source-Control:** Records changes to files so that changes can be recalled at a later date. [Chacon, 2009]

**Distributed:** In the context of source control, this is a method of source control that allows developers to code without being connected to a central point.

**Centralized:** In the context of a source control system, a method of source control where all developers work off of and add their code to a singular, connected codebase.

**Linux:** Free and open source operating system, initially released in 1991. Based on Unix. [LINFO, 2006]

**Production Code:** Finalized code. Code that is considered to be finished and able to be shipped.

**Developer:** A person who designs, writes and tests software.

**Refactoring:** changing already-functioning code to be more flexible and maintainable

**Branch:** A movable pointer to a specific commit in Git [Chacon, 2009].

**Pointer:** In the computer, a point in memory, expressed as a value, that represents the location of another value.

**Commit:** In a version control system, a commit is a specific set of changes made to a code base.

**Schema:** A formal representation of a database, its tables, and how they are interconnected [Oracle, 2014]

**Object-Oriented:** A programming paradigm where items are represented as objects. Objects are generally composed of self-contained values and functions that describe and access the features of the object.

**Web Server:** Special hardware and/or software that handles serving web pages over the internet.

**Apache:** A common, open source web server application [Apache Contributors, 2014]

**Server:** Hardware and software that handles requests over a network.[Wikipedia, 2014]

**Webspace:** The section of a webserver specially allocated for a specific web site.

**Sandbox:** A sectioned off space that separates programs for security purposes.

**HTTP:** HyperText Transfer Protocol. Protocol for application level data transmission over the internet [Fielding et. al, 1999]

**HTTPS:** Hypertext Transfer Protocol Secure. Protocol for secure, application level data transmission over the internet

**Codebase:** Code written for a project, typically stored in a repository. Generally only refers to code that has been written by a developer, not code that is generated by a program. [Wikipedia, 2014]

**Directory:** A folder in a file system. A specific location on a hard drive where files are stored. [LINFO, 2006]

**Root:** The starting point of a file system in a computer [LINFO, 2007]

**Data-Type:** In computer programming, a way of describing what a collection of data represents.

For instance, 1 as a Boolean represents true, but 1 as an integer represents the number 1.

**Boolean:** data-type representing only true or false

**Single Sign-On Protocol:** System where a user uses a single username and password to access many different applications that may or may not be related. [James, 2007]

**Design Pattern:** A solution to a problem that occurs frequently that is reusable. A formalized method that is considered the best way to solve a problem.

**High-Level:** A programming language that is easily understood, usually utilizing elements of natural language and automation of areas in a programming language. Typically runs above system and procedural level. [Wikipedia, 2014]

**SQL Queries:** Commands sent to an SQL database.

**LAMP stack:** Stands for Linux, Apache, MySQL, PHP. A commonly used mixture of tools used to serve web pages.

## 1 Introduction

Our goal in completing this project was to develop a system for users to find their Major Qualifying Projects. Specifically, we wanted this system to promote student-to-student interaction for group-forming, whereas the current process focuses mainly on student-to-advisor group-forming. Our target audience was students in the Computer Science department at Worcester Polytechnic Institute; namely students in their third, or junior, year. We named our system ProjectSpot because it would be a place for students to “spot” and coordinate projects.

### 1.1 Problems with the MQP-Finding Process

The Computer Science Department at WPI holds a session every year to tell students about the MQPs that the professors in the department would like to advise. This “MQP-pitch session” is one of the main events the department does to inform students about the MQP-finding process.

From our initial data collecting, we found that most students in the department do not attend the MQP-pitch session. This is a problem, because the department considers this session to be one of the primary ways to inform students about MQP opportunities with professors. From our data, we found that the main way that RBE, CS, and IMGD students find projects is by contacting professors directly and asking them to be their advisors. This current process is not student-facing, which many of the students we surveyed expressed interest in.

Some students did not know their advisors before they signed up to do an MQP with them. Students we surveyed said that this was intimidating to them, and considered to be a risk. However, students stated that having a professor with a similar subject interest was important enough for them to work with an unfamiliar advisor.

We saw the need to provide students with more opportunities and more information in regards to finding an MQP. Our primary concerns were to change the process to focus on the students’ interests and goals.

### 1.2 Problem Statement

The goal of this project is to develop an online service for WPI students in the Computer Science Department to easily form and find groups for MQP projects. Through working with the

Computer Science Department and the students at WPI, we will determine what students need in order to find an MQP for their senior year. Our system, ProjectSpot, will address the biggest concerns and problems that students have when looking for an MQP. ProjectSpot will be flexible and robust, allowing for future changes and possible expansion into other departments at WPI. As a result of this project, students in the Computer Science Department will have a tool that will allow them figure out their MQP through other students.

### 1.3 Requirements

Our requirements for the system we set out to create would all be focused on what would make the process more robust for students. The system (ProjectSpot) would have to allow for a way for students to login, making sure that only members of the WPI community could access it. Advisors would also need a way to login to the system; as they could post their own projects here that students may be interested in. This would allow for a centralized way of containing all MQP opportunities in the department.

ProjectSpot would need a way for students to browse all available MQPs in the system. Students would need to be able to differentiate between projects offered by student groups and projects offered by advisors. Students should also be able to browse other students in the system who do not yet have MQPs. This is important because students may want to form project-groups with these other students, or a group of students may be using ProjectSpot to look for additional group members.

Every user of ProjectSpot, namely students, project groups of students, and advisors, would all need to be able to create a profile with some basic information. This would be useful for those searching for information on potential projects, partners, or advisors. Every profile will have a description in text of the project, student, or advisor, and “tags” of the area(s) of computer science that the profile pertains to. These tags will be pre-defined terms that users can add to their profiles to better-define what they are interested in and what they are looking for. Each profile will always have at least one contact email, so that project groups can be coordinated. Every profile will have the option to include a picture, so that users of the system will be able to better-recognize fellow users.

There will be additional requirements for the different types of profiles. For a project

group, this information would also include the project title, a description of the project, the names and possibly profiles of the other group members, and the advisor(s) for the project if any. For project groups, the terms that the MQP will be offered should also be included if known. For an individual student, the information would also contain a name, a link to the student's MQP group if he or she has one, and the student's basic skills and experience. For an advisor, there would be the advisor's name, and there would be a link to the advisor's website, where his or her MQPs are usually listed. There would also be links to the MQPs the advisor is currently planning to advise.

Other requirements of ProjectSpot would include a way to search all projects, students, and advisors—filtering these down to a narrower set of results. Users would also need to be able to search by tags, or specific terms that pertain to their specific interests. Our system would also need to have links to other useful resources for finding MQPs. This would include links to the MQP registration page, the Global MQPs page, and the list of available projects as offered by professors. There will be a section of the system for important dates; such as the date of the Computer Science Department's MQP-pitch session.

The goal of our system is to bring users together so they can coordinate MQP groups. For this reason, another requirement of the system was to generate potential matches of students, groups, and advisors, and have a way of suggesting them to each other. To do this, we had to design a matching, or recommender system that would use our different pre-defined tags to suggest projects, advisors, and students to other students and project groups. This would provide a fast and simple way for users to find exactly what they are looking for in ProjectSpot, without having to do any searching or browsing.

## 2 Background

In order to design a system that would allow users to form groups, we had to research other similar systems and the approaches that they took to match users together. We also had to study our target area of users for the system so that we could create an application that matched their needs. For this, we had to consider the way that the current MQP-finding process works at WPI, and determine what could be improved. Since we chose to create a web-application to fulfill the requirements for such a system, we needed to examine the current technologies and tools we could use. In order to coordinate ProjectSpot with the current MQP-finding process at WPI, we had to work with the school to allow students to login to our system with their WPI credentials.

### 2.1 The MQP at WPI

To graduate from Worcester Polytechnic Institute, every student must complete a Major Qualifying Project (MQP) [WPI Gordon Library, 2013]. The MQP is a project that focuses on the student's major. Based on what a student's major is, the process varies from department to department. This report mainly focuses on what Computer Science (CS), Robotics (RBE), and Interactive Media & Game Development Majors (IMGD) have to do in order to obtain an MQP. All three of these majors are part of the Computer Science department at WPI.

#### 2.1.1 The Current Process

Students in the Computer Science department at WPI have to find an advisor, project group, and a project to have an MQP. For IMGD students, student teams must create and pitch an MQP idea on their own, and projects are not provided by advisors like they often are for CS and RBE majors.

Computer Science, Robotics, and IMGD majors also have the opportunity to go abroad and complete their MQPs at an off-campus project center in one or more terms. When students go abroad for MQP, they often do not get to select their project groups, advisors, or projects. Off-campus MQPs usually have sponsors. These sponsors can be companies that fund the project, in order to obtain research or work for their company.

Students who choose to stay on campus can complete their MQPs in one, two, three, or occasionally four terms. On average, a student will take three terms to complete his or her

MQP. It is essential to plan-ahead, and find an MQP at least one term beforehand. If a student waits too long to choose an MQP, he or she will not be able to graduate on-time, and will have to stay at WPI until the MQP is completed.

### **2.1.2 What Resources Exist**

To find MQPs, students in the Computer Science department can attend an annual MQP-pitch session that is held by the professors in the department. The professors in the Computer Science department present short descriptions of the projects that they have chosen to advise for the upcoming year. Some of the professors at the pitch session will also pitch projects that have a tie-in to robotics that RBE majors can do. There is no such pitch day for IMGD students. Some of the projects that are presented at this event have outside sponsors that work with advisors and students to complete a project to the sponsor's specifications.

If a student wants to go abroad for MQP, there are different information sessions for each site that a student can attend. From these sessions, students can get a general idea of the types of projects they may be working on. They also learn about the sponsors that are involved in the MQPs at that site.

There are some online resources that students can access to help them find MQPs. The most prevalent system is the listing of projects available to students, maintained by the WPI Projects Program [WPI, 2013]. This system contains a listing of various proposed projects by professors, as well as information about these projects. This information includes a list of the majors the MQP will service and whether or not the MQP is open or closed to additional students. Users are able to search for projects by discipline and keyword, and users can indicate if they are interested in a project by clicking a button on a specific project's description page. The system is held behind a login wall, and can only be accessed with a valid WPI computer account.

Alternatively, some professors will list the projects that they want to advise on their personal web pages. Students can contact those professors individually in order to find a project.

## **2.2 Recommender Systems**

Recommender systems are a type of information filtering system that are used to help predict what items a user would like to see. This prediction is based upon previously gathered



information about the user. This information could include a list of items the user has liked and a history of previously purchased items. This information is then compared to that of other users, in order to predict what the current user might like. For example, if User A had purchased widgets 1, 2 and 3, and User B had purchased widgets 1, 2, 3 and 4, the recommender system would then suggest widget 4 to the user.

These systems are commonly used in various online stores, such as Amazon or other services like Netflix and Facebook. These systems are very effective at providing relevant information to the user and do so in various ways. We can use the concepts of recommender systems in our product in order to provide more relevant search results to users.

Netflix uses a collaborative approach for its recommendation system. Netflix is a subscription service that lets users watch shows, documentaries, and movies. Users will be recommended items similar to items they have already viewed. Netflix uses user ratings in order to construct a profile of the user and then give the user recommendations. This process of collaborative filtering does not rely solely on ratings, either. It can be expanded to include metrics as such previously viewed items and the time a user spends on a page, along with other pieces of data. The downside to these types of collaborative filtering systems is that they require a user to be proactive in order to work effectively. Few users will rate things, as rating is a time-consuming process, and when a user is relatively new, these systems have very little to base their filtering on.

### **2.2.1 Algorithms Used in Content-Based Filters**

Another approach to recommending items to a user is to use a content-based filter. Content-based filtering relies on information about a specific item and uses that to find similar items based on various criteria [Meteren et. al., 2013]. While collaborative filtering uses a many-to-one approach, content-based filtering relies on one to many. Eventually these systems will grow to be many-to-many systems, but when first starting out there is little to go on in both directions. Because of the nature of our project, a content-based filter would be more appropriate than a recommender system, due to the fact that we will not have time to construct a sophisticated, in-depth user profile.

The most effective content-based filters use various classification algorithms, such as decision trees. Decision trees have proven to be useful in various online storefronts at providing users with related products [Kim et. al., 2013]. By creating a number of decision trees at various

levels of the system when users are searching for projects, we can use then filter these trees out until we have a valid set, providing the users with relevant search results. These can be assisted further by applying weights to various metrics, based upon the needs of the system. For instance, if a user were to be looking for a project and has networking listed as an area of interest, we can use that to weight projects that have a focus on networking.

### **2.2.2 Content-Based Filtering**

A content-based filter will organize search results based on a weighting-system that determines the relevance of results. Separate weights can be applied to each result in a search. In systems where some information is previously known about the user, content-based filtering can be used along with the user's information in order to provide him or her with relevant search results. In this proposed system, a user could search for a specific item, and the results presented would be ordered by relevance to his or her interests. Suppose a user indicates in his or her profile that he or she is quite fond of cheese. This user then searches for the term "Vermont." Because the system knows that he or she likes cheese, it would then prioritize any products listed as cheese, so one of the first results in the search would likely be "Vermont Cheddar." In practice, there would be many more weights involved in such a search, making the system more applicable.

## **2.3 Matching Systems**

By studying matching systems, we can obtain data that can help us design the interface and functionality for our system. Matching systems are generally used in order to match people together based upon certain criteria. The most common uses for these are with online dating or multiplayer video games. These systems rely on information about the user in order to pair them with similar people. This information can be based on user preferences, user performance, or a mixture of both.

### **2.3.1 Concerns with Matching Systems**

The main focus of matching systems is to make sure that matches are compatible. Depending on the type of matching system, compatibility can mean a number of things. For instance, in a matchmaking service used for dating, compatibility is based on numerous factors, from personal quirks to something as simple as eye color [Goldman, 2013]. In the case of

multiplayer games, however, matches are a bit easier. Typically, players are matched together based upon a ranking of skill. This ranking is traditionally calculated by analyzing a player's skill and comparing that to other players, matching players together with the goal of having an equal amount of wins when compared to losses. These systems are usually based on number rankings, and as such, are much easier to match together.

### 2.3.2 Matching System Algorithms

Many matching services consider their algorithms intellectual property and keep them secret. However some matching services are much more open about how they match people together. One popular dating service, okcupid, reveals that it relies on survey questions that users fill out in order to match people together. When comparing two people, okcupid looks at three things: the answer the user provides, the answer the user would like from somebody, and how important the question is to the user [okcupid, 2013]. Point values are then assigned to questions, ranging from 0 to 250, based on how important the questions are to the user. The system then scores each person based upon questions they both have answered. For instance, if person A indicates that something is important, and Person B provides a suitable answer for that question, then Person B gains points. The points each person receives from answering questions that are similar to the other person's answers are then added together. This summation is done for both Person A's and Person B's question-and-answer matches. Person A's total is then divided by the maximum amount any person can receive if he or she answered all of person B's questions the same, and vice versa. These two quotients are then multiplied together and the resulting product is squared, ending with the final match percentage. This match percentage indicates how satisfied the user would be with the other person.

Many popular online games, most notably Massively Multiplayer Online (MMO) games, rely largely on group-based content, where players are to work together towards a common goal. Because of this, many games have implemented various group-finder features that have, over time, been refined.

In the popular MMO game World of Warcraft, there is a version of a group-finding interface named the dungeon finder. Here, users can flag themselves as looking for a group, or flag their groups as looking for players. The former is a simple checkbox while the latter uses a flag, as well as an input box, where a group can write down what they need for their group.

Another tool allows users to look for guilds, which are big collections of players used for

various purposes. A guild officer can put his guild into the guild finder, listing various traits about the guild, such as usual playtimes, guild goals, and a description of the guild. Users can then browse the guilds listed, searching based on various parameters, and then apply to a guild they like.

### **2.3.3 Group Finding Tools for Finding an MQP**

While the dungeon finder tools in World of Warcraft are good to look at while thinking about designing the project, an automated tool would not suit us well. Ultimately, an automated tool would take control away from users and have a high risk of error, especially with so many variables that need to be accounted for. While it works in a game where there are only three skillsets to fulfill that are clearly defined, an MQP can include many different skillsets and needs to be more dynamic. Instead of focusing on an automated tool, we can take notes from other, manual tools inside the game and draw inspiration from them. For ProjectSpot, we can emulate the World of Warcraft guild finder tool, which has the inclusion of group descriptions via a simple textbox. Including descriptions is useful to the guild finder and to ProjectSpot, because it allows users to state exactly what they need in a custom textbox, in order to prevent ambiguity.

## **2.4 Web Development - Tools & Technologies Used**

In order to build a system that will be accessible from the web, the developer has to use a certain set of tools and technologies.

### **2.4.1 Model View Controller (MVC)**

Model-View-Controller is a high-level design pattern that allows for the abstraction of the data in the system, apart from the interface of the system, separated from the functionality of the system [EllisLab Inc., 2014]. Model-View-Controller is popular because it allows for the interface of a system to be developed in an environment that is isolated from the system's functionality, while still having the ability for the two parts to interact and communicate to be a fully-working system.

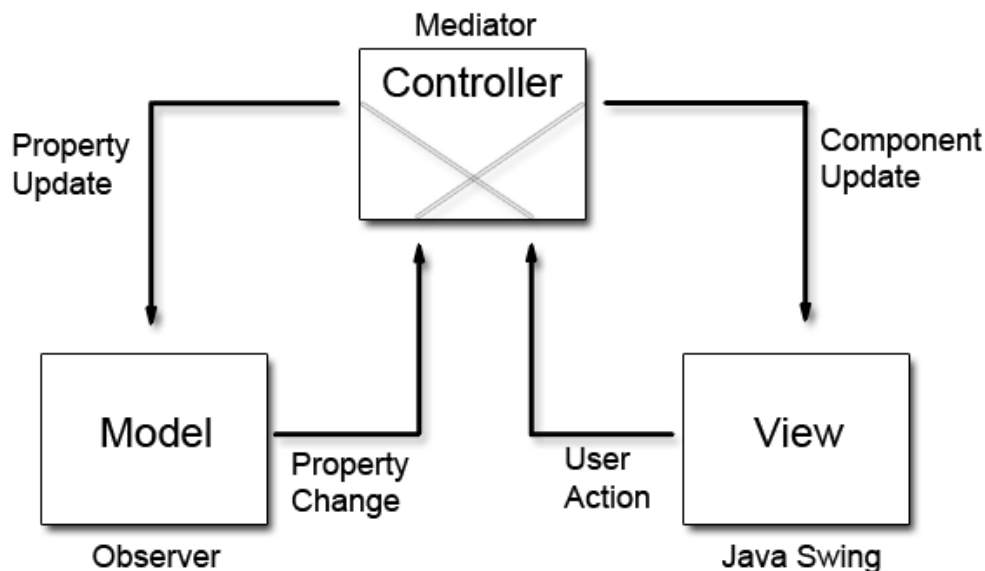


Figure 1: Model-View-Controller Design Pattern [CodeProject.com, 2008]

The MVC design pattern was allegedly invented in Xerox park in the 1970s [Cunningham, 2012]. Its first noted use was in the purely object-oriented language Smalltalk-80. The goal of MVC at the time was to use it to create better Graphical User Interfaces (or “GUIs”) for applications. With a more efficient programming environment, the developer is given the opportunity to focus on developing a better user-experience for the user; less than where the developed user interface is limited by the constraints of the system’s architecture.

Today, the MVC design pattern is used in many web applications. This is mainly because of the structure of the web. The browser serves the user a webpage. This webpage is given to the browser by a server. The browser then parses the HTML of the webpage, to display content to the user. The server that provides the website can be thought of as the “back-end”, and the browser that displays the website can be thought of as the “front-end”. All websites are made available through this process. The front-end is the “View” in the MVC paradigm; and the back-end is the “Model”. The “C” part of MVC here (being the “Controller”) is the code that allows the back-end and the front-end to communicate with each other. This allows the back-end to send information to the front-end, and the front-end to display information from the back-end.

Since the MVC design pattern is so highly-integrated with the web, and because it is a powerful concept that allows for a well-organized system, we chose to follow MVC design principles in implementing ProjectSpot.

#### 2.4.2 Client Side – HTML, CSS, Javascript

All web-based systems have a front-facing component. This is often referred to as the “client side”. In the client-server relationship, there is a back-end that serves the requested information to the client, and the client deals primarily with the front-end of the application. In order to display the information that has been sent from the server, the client must use HTML in some way. W3Schools explains HTML as “a language for describing webpages,” [W3Schools, 2013]. HTML allows the system to display content on the screen. HTML ties in closely with CSS. CSS (Cascading Style Sheets) is a mark-up language that allows for the specified styling of the content that appears on the web. HTML provides the structure of the content with no styling. CSS then adds attributes to the content provided by the HTML to change its appearance.

HTML and CSS deal with static data and content. The dynamic part of web-based system comes from scripting languages. Javascript is a scripting language that is used to update content on the client-side.

Other scripting languages such as PHP and Python deal with accessing the stored data or database for the application. This is known as the “back-end”, whereas client-side is considered to be the “front-end”. The client-side does not generally deal with data-retrieval or manipulation, and is only concerned with updating the front-end of the application—the part that the user sees.

#### 2.4.3 Server Side – PHP

PHP, also known as *PHP: Hypertext Processor*, is a server side scripting language that is used mainly for web development. As of January of 2013, PHP is in use on over 244 Million sites around the world [Ides, 2013]. The main attraction of the language is that it is free to use, and can be installed on almost any computer. Documentation is largely handled by the PHP Group, and the core build of PHP contains a number of free, open source modules, that are mainly devoted to dealing with problems that face web developers, such as support for accessing databases, as well as handling file uploads, amongst many others.

While the language is quite robust, and has been around for some time, there are a number of issues plaguing it. To start with, it is generally considered to be insecure. Upon

searching the Common Vulnerabilities and Exposures system, maintained by MITRE, around 29% of all vulnerabilities in code are linked to PHP [Coelho, 2014]. More often than not, this is linked to sloppy developers not following standards, leading to insecure code. At the same time, the language has been developed rather organically, leading to functions being named inconsistently, as well as many built in functions that most programs simply do not need.

Despite the drawbacks of the system, there are a number of benefits for using PHP. First, it is rather easy to learn. It is a dynamically typed programming language that has a lot of flexibility and power. In addition, most of the documentation for PHP is handled by the community, and tends to be thorough. Chances are, if a developer is having a problem, the solution is readily accessible with a few Google searches. Finally, PHP is everywhere and can be installed on anything. While the language of PHP may not be consistent with regards to standard naming conventions, PHP itself is still relatively lightweight and can be easily expanded upon to add more functionality. Couple this with the plethora of frameworks that can be utilized with PHP, and you have one of the most powerful languages designed specifically for handling web applications.

#### **2.4.4 CodeIgniter**

CodeIgniter is a PHP framework, developed by Ellis Labs, that focuses on making web developers' lives easier [EllisLabs Inc., 2013]. Common tasks, such as interfacing with databases, handling form submissions, amongst other things, are achieved through the use of a robust set of libraries. This allows developers to focus on coding the larger parts of their web applications, instead of the more tedious tasks that plague PHP developers.

Compared to other available frameworks, CodeIgniter does not take long to set up. CodeIgniter is very flexible, allowing developers to have full control of the system. While the framework has a standardized way of doing things, modeled after the Model-View-Controller design pattern, developers are able to ignore these guidelines if they prefer. The CodeIgniter framework has even been lauded by the creator of PHP, Rasmus Lerdorf, saying that compared to other frameworks, CodeIgniter is the one that is "least like a framework" [Peterson, 2008].

Because CodeIgniter does not contain many additional features in its system, and because it focuses on extensibility, CodeIgniter proves to be a powerful framework on which to base a web application. It is rather easy to use, and is extremely well-documented. With CodeIgniter, the functionality of the system itself is not dependent on the CodeIgniter framework. For this

reason, the developers are not confined by the constraints of the framework, and the system remains flexible.

## 2.5 Databases

Databases are collections of organized data. There are a number of database systems available to developers, which differ in ways from the method for data storage to the organization and retrieval of data. Generally, most databases are accessed by a specific language, such as Structured Query Language (SQL). Each type of database has its own set of pros and cons, and many different programming languages work well with specific database systems. Many PHP web applications rely on what is called a LAMP Stack. A LAMP Stack simply means a Linux system, with an Apache webserver, using a MySQL database, hosting PHP for the back-end of the web site.

### 2.5.1 Types of Databases

There are many different types of design philosophies available when deciding how to organize and build a database's table structure, all with their own strengths and weaknesses. Some databases are designed with speed in mind, while others are designed under the assumption that lots of data will be handled by the database. Regardless of how a database is organized, there really is no answer that will satisfy everybody, so programmers tend to need to choose a database design that best suits their needs. One of the bigger points of contention concerning database design is a problem that faces many programmers when choosing what programming language to use: the issue of strongly typed languages versus weakly typed ones.

A strongly typed programming language is one that has its syntax rigidly defined [Hanenburg, 2010]. One of the most prevalent distinctions of a strongly typed programming language is enforcing data types. For instance, if you were to have a variable that represented an integer, you would need to explicitly state that this variable was an integer. In addition, only certain kinds of operations can be performed on that variable, because it is of type integer. While this can make code verbose, it helps to reduce ambiguity in code. This helps developers when trying to ascertain what exactly a function is doing, allowing them to get to work faster.

On the other hand, a weakly typed language is one that is a lot more flexible with what can be done with variables. Traditionally, most weakly typed languages do not have explicitly



stated data types. Instead, the data type is inferred at run time. For instance, in some languages, if you were to issue the command “1 + ‘1’”, some languages would infer that you would like to do number addition, and return the number 2. Other languages might instead infer that the desired result is string concatenation, and instead return the string “11.” While this makes coding a bit easier, code can become rather ambiguous, making it hard for code to be maintained. In addition, it is harder to figure out where some bugs may be, due to syntactically valid code not working the way it is expected to.

With regards to databases, and how they are structured, a database that is organized like a weakly typed language, with lots of inferring of data types, can be quite flexible, depending on how you organize it. For instance, the popular blogging tool, Wordpress, has a rather weakly typed database, in order to allow developers to easily expand on the database without having to change the database schema at all [Wordpress, 2013]. On the other hand, this leads to a database being somewhat slower with searches, due to all of the various data types for each module in the system being contained in a single table. Splitting modules up into separate tables allows a new developer of the system to see exactly where things are stored, at the cost of having to change the database schema when models need to be expanded. These factors are ones we needed to consider when deciding how we wish to organize our database structure.

### 2.5.2 Structured Query Language (SQL)

Structured Query Language, or SQL is a programming language designed specifically for interfacing with relational databases. SQL allows users to insert, update, delete, and query a relational database, as well as modify the database’s schema and other aspects of the database. Originally designed in 1970 by IBM, it was created to replace the current ‘standing processes’ that required programmers to create specially designed programs whenever data needed to be stored [Microsoft, 2013].

Relational Databases are a specific type of database that consists of tables containing data. A set of data in a table is known as a relation. Each table requires at least one column be used as a unique identifier for each row of data. This allows users or other entries in the database to point to specific entries in the database, eliminating the need for duplicated data. Relational databases can even be further refined in various ways in order to increase flexibility as well as performance.

The most common operations handled by SQL are queries. These are performed using

the “SELECT” command. These commands do not modify the database, but simply initiate a search within the database for information they wish to find. The most basic queries are quite simple, telling the program to search for values within specific fields and then return (“select”) the data from rows with values matching the search. Generally, the user selects a specific table from several within the database. These queries can be expanded upon, allowing the user to filter out things they do not need, sort the results, or even expand or limit the result set using data from other tables through joins. Queries can even be recursive—meaning that they can contain queries inside of them, allowing the language to be quite flexible.

There are many other commands that can be issued through SQL, such as adding, removing, or editing existing entries, adding entire new tables, changing the structure of existing tables, or even dropping an entire table from the database. This much versatility means that SQL is an extremely powerful but potentially dangerous tool and developers must exercise care when issuing commands, especially when dealing with user input.

### 2.5.3 Database Structure Approaches

WordPress is a highly popular, open source blogging tool and Content Management system, developed using the scripting language PHP and a MySQL database. The attraction of this system is its extensibility and flexibility in design, allowing developers to tear down the system and rebuild it to suit their needs. This has led to it being the most widely used blogging system on the World Wide Web, with more than 60 million websites using the framework as of August of 2012 [Colao, 2012].

One of the major benefits of WordPress is its database design, which is a big factor in how extensible it is. Every object in WordPress is considered a “post”, where a post is the term given to any collection of data that is stored [WordPress, 2013]. While each post has its own set of attributes, the beauty of the system is how it handles custom entries. Suppose, for example, that you are running an online store using WordPress. Each product that you sell is a post with a special post type of “product.” While each product can have a description and a name, which is handled by the post, extra values that might not be used in a blog post, such as price, are stored in a separate table, containing metadata. This table contains only four attributes, the id of the metadata, the id of the post it belongs to, the key of the value, and the actual value. Because of how this is set up, developers can easily add another field to the product without having to modify the database’s schema. This saves development time and reduces the chance of data-loss

when adding new attributes to a data-type.

This flexibility comes with some drawbacks in terms of efficiency. It may be helpful that attributes are centralized in one flexible place, but with this schema, everything is always saved as a block of text, even if it is something as simple as a checkbox on a form. Some items can be as simple as a Boolean data-type but would still be stored as plain text in the database. This leads to data taking up more space than it actually needs in some situations. This also runs into the issue of the database being “tall”; as opposed to being spread out over more tables, inflating search time. On top of this, comparisons of strings are much slower than comparisons of integers in a MySQL database, due to differences in size.

For these reasons, it is important to consider other database structures aside from those proposed by Wordpress. We studied another database design that is more commonly used in web development, basing it off of the Boyce-Codd Normal Form for databases [Wikipedia, 2014]. The data is normalized, which means that redundant values are never stored. If values need to be updated, they are only updated in one place. With this database structure, each type of data, such as groups and user profiles, has its own database table that is strictly defined. While some tables may have shared fields, it is easier to see where data is stored and how it is structured. Instead of having to look through the code, a developer can simply look at the database schema in order to figure out how the database is laid out. At the same time, we can have the database enforce properly structured data, because we can explicitly state what data types are needed and where. With this structure, we can refuse data that is malformed, thus increasing security and stability with our system.

The only real drawback to this design is that if a developer wanted to add a new field to a table, he or she would need to go in and change the database’s schema. This is a problem with many databases. While changing a database’s schema sounds rather complicated, if done correctly, it is a simple process. Developers that work in databases are already familiar with this process. Because of how standardized it is, we elected to go with this design based on the Boyce-Codd Normal Form for databases.

## 2.6 Source Control - Git

Source control, also called version control, is a system that tracks changes to files. With source control, multiple people are able to work on the same code base, while different

developers make changes to the code on different machines [Chacon, 2009]. This allows programmers to maintain one organized code base. There are different types of source control such as SVN and Git [Chacon, 2009]. The most important difference to note is that Git is a form of distributed source control and SVN is a centralized form of source control. This means that with Git you can have one or more versions of the main code base that exist locally on your own computer. With SVN, you must always be connected to the server with the main code base to save any changes.

With source control, you can have multiple versions of the main code base that can be merged back in. This is used frequently with Git, but not as much so with SVN, because with SVN any change made will always affect the centralized code base. We chose to use Git because of our familiarity with the tool, as well as for its efficiency, speed, reliability, and flexibility

Git was developed in 2005 by the Linux community to be a fully-distributed form of source/version control that would be able to handle large projects [Chacon, 2009]. Some of the best aspects of Git is that it is fast and highly flexible.

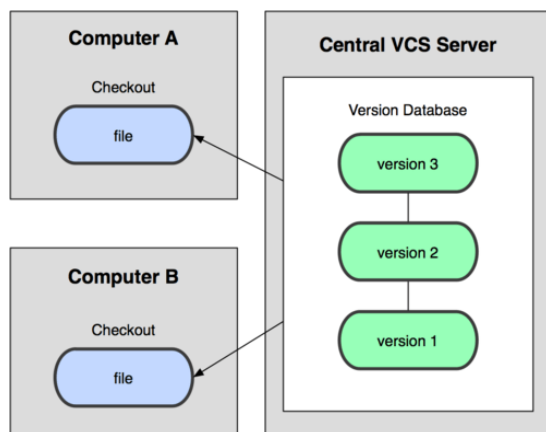


Figure 3: A Centralized Source Control Code Base [Chacon, 2009]

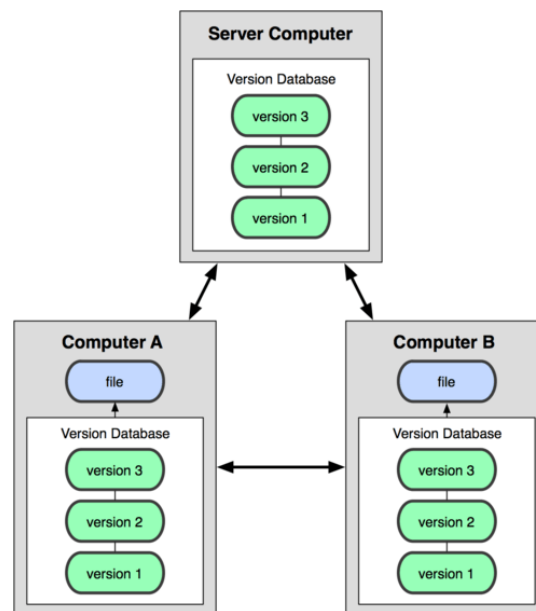


Figure 2: A Distributed Source Control Code Base [Chacon, 2009]

Git works in a tree-system with different “branches” on the tree. The main branch is traditionally called “master” and can be thought of as the “trunk” of the tree. The master branch is often the production code, and is kept in a bug-free state at all times if possible. Then, any number of developers can create child branches off of the master branch. When a developer creates a child branch, that branch starts out as a clone of the master branch. Then the developer can “commit” changes to the new branch without affecting the master branch.

With Git, the developer can work remotely; meaning he or she does not have to be connected to a main server to work off of the main code base. By working remotely, developers can have their own versions of the code base on their computers. Whenever a developer wants to update his or her local repository with the latest version of master, one only needs to “pull” the code from the remote server via an internet connection.

Git works well because master can be updated at any time by any developer. The way that code is added when using Git is through a process called “merging”. What happens is that once a developer makes a new branch off of master, that developer will code on that branch in order to make changes. When a developer adds code, he or she can be writing new features for the software, fixing bugs, or refactoring the code. Once a developer is satisfied with the changes made, she can “merge” her branch into the master branch, and those changes will be seamlessly added into master. This can be thought of as replacing the old code in master with the new code of the developer’s branch.

One frustrating thing when dealing with source control is merge conflicts. A merge conflict occurs when two developers change the same lines of code on different branches, and then both developers try to merge that new code into the same location. What happens is that Git does not know which version of the altered code is correct, so it cannot add either. The repository remains in an almost “paralyzed” state until one of the developers can resolve the issue.

The more developers working on a single code base, the more frequently merge conflicts will occur. However, if the developers are careful and remember to keep master up-to-date by pulling from the remote repository on a frequent basis, much of this can be avoided. A good rule-of-thumb is to always pull master right before you attempt to merge your branch into it. It is also important to not work off of another developer’s branch without his or her permission.

## **2.7 Central Authentication Service (CAS)**

For WPI authentication, WPI uses a service called Central Authentication Service, or CAS. It is a single sign-on protocol that has been developed for a number of years, and has been rolled out to nearly every WPI web service that uses a user account. The major benefit of using CAS is that it has already been written, and is in use all around WPI. A user from WPI would already know how to use such a system, and this helps make the ProjectSpot feel more official. By using CAS, we avoid the issues that come with choosing or writing an authentication service. With CAS, we do not have to worry about protecting user data or securing passwords, as it is already handled for us.

## 3 Methodology

Our goal was to design a system that would allow students to find MQPs. We collected user data that would allow us to determine the needs of the users for the system. We organized a list of user tasks and subtasks from the collected data. We took our list of tasks and subtasks and figured out how a user would want to move between processes in the system. With that determined, we designed a flow diagram that would describe the user's steps through the software in correlation with the user's mental model. From our flow diagram, we designed a potential user interface via wireframes that we tested on WPI students for usability. From the success in our studies, we implemented the designed user interface. We designed a back-end that would complement this user interface and work in a Model-View-Controller design pattern.

### 3.1 Collecting User Data

We collected data from six students in our target area. We focused on senior WPI students in the Computer Science Department who had already found their MQPs, whether they had completed them or not. We asked them questions that would help us to learn what features they would need in an MQP-matching system. The data collection was structured in an interview format. Each student interviewee signed a Consent Form. The interviews were held privately and with discretion. The Institute Review Board was not used. The data collection was conducted in a calm and generally quiet setting. The interviewer would ask the student each question, and then have the student elaborate when necessary. The interviewer would type up the student's response as he or she was talking. Some of the questions asked had the students rate something on a scale from 1 to 10. The questions asked can be seen in Appendix A.

### 3.2 Designing the Interface (based on User Data)

We took the students' needs into consideration when designing the user interface for ProjectSpot. We based our design on the survey data that we had collected (this data is available in Appendix B). Most students expressed the need for viewing a list of all available projects, advisors, and students in a list. We made this the layout of our main screen for searching the system in ProjectSpot .

### 3.3 Designing the System

We had to design a system that would be robust and extendable. This was important because our goal was to have the WPI Computer Science Department adopt ProjectSpot after we graduated. If the school were ever to incorporate ProjectSpot as part of the MQP-finding process, then our system would have to allow future system administrators to make changes. That notion drove many of the design decisions for ProjectSpot.

In order to have this system used by WPI, we had a number of requirements we needed to meet. One such requirement was that our application had to be flexible. We had to account for the fact that somebody would inherit this project after us, and we wanted to make sure it was something that person could use and understand. In addition, we needed something that was safe and secure, as we had to handle user data. We did not want anybody to be able to break into our system and get that data. While the data we obtained for each user was already publicly visible in WPI's system, we still wanted to keep our system as secure as possible. Finally, we wanted our system to be fast and efficient, in case WPI as a whole, and not solely the Computer Science Department, eventually wanted to use ProjectSpot. We needed to be able to handle a large number of users accessing ProjectSpot all at the same time.

#### 3.3.1 CodeIgniter: Choosing a Framework

We thought long and hard about what framework we wanted to use at the core of our project. We considered using many different frameworks, including one that was an offshoot of the CodeIgniter project. Ultimately, we felt that CodeIgniter was the best choice for us. One of the main reasons for us choosing CodeIgniter was that we wanted to design our project with MVC in mind, which CodeIgniter was built around. We also wanted something fast and flexible, and when compared to other frameworks, CodeIgniter has consistently ranked higher in many different tests on speed [Kujawa, 2013].

Another point of contention was that we wanted our system to be extensible, so by proxy, our framework needed to be flexible as well. Because it is built with MVC in mind, and comes with a rather extensive set of documentation, CodeIgniter was a good fit. This is coupled with the fact that the framework has been around for a number of years, and includes a rather robust community that is generally very helpful when running into issues. We did not want to create extra work for ourselves or any future developers, so we needed something that was rather mature, and solidly developed.



Finally, we wanted something that had a feature set that would suit our needs. Specifically, we needed something that would interface with a database and handle MVC well. Not only does CodeIgniter do all this, it also has libraries that handle stored sessions, improving security, and automatically sanitizes user input, so we do not have to worry about SQL injections. Finally, the framework has a number of security features that protect against other attacks, such as Cross Site Scripting (XSS) attacks already enabled, saving us a lot of time.

### **3.3.2 Database**

When designing our system's database, we had to keep a number of things in mind. Our biggest issues were mainly making sure we were able to handle lots of data effectively, as well as making sure when another team picks up the project, they will be able to easily modify and work with it. This wasn't a simple task, and our database went through a number of design changes, before we settled on one that suited our needs. In the end, we came up with two designs that would suit our needs, and after analysis, we decided on the one that would allow for the most flexibility and scalability, as well as being the most clear. This proved to be useful, as through the development of the system, some changes needed to be made for adding new features, and our organization of the database made it quite simple to do so.

### **3.4 Implementing the Interface**

We implemented the interface based on our designs. We created a template for the header and footer of the website, so that each page could have the same top and bottom, but have different content.

We created blank content pages for each required screen of the system. We then created content for each screen using HTML. Some information on the screens was generated by pulling data from the database using PHP and SQL queries. The content of each page was styled using CSS. Different themes were used to keep aspects of the site such as buttons, headings, and hyperlinks consistent.

We designed the system around the design for the interface, and we implemented the interface in accordance with the structure of our system.

### 3.5 Implementing the System

When implementing the system the fact that we were using an MVC design pattern helped us build the application. We were able to work on the views and models at the same time separately, before working on each individual controller. In this case, we will look at how the groups system was designed.

To start, the model was created with a straightforward approach. We started with four pieces that would be core functionality, getting a single group by its id, getting all the groups in the system, adding a new group, and updating an existing group.

At the same time, the interface was being developed for the various pages being served by the groups section. Once these pages were built, we would know how the information would be displayed, and know exactly what the controller needed to do.

Once the models and views were finished, we were able to focus on the controllers, where most of the processing of data takes place. The controllers are what communicate between the views and the models, asking models for the data needed, processing it, and then presenting it to the view in a way it can be processed. Building these was relatively simple. Each view had its own function in the controller, allowing us to keep things separate and organized. Each of these functions were organized in the same way, starting with retrieving the data we needed, formatting it in such a way that the views could process it, and then passing that information to the views. The views would then parse that data, and then present it to the user. We added functionality to the controllers as we went on, if the need for extra functionality was warranted.

### 3.6 Implementing CAS (Central Authentication Service)

To implement CAS to work with our system, we had to work closely with the Computer Science Department's system administrator, Michael Voorhis. Mike configured our server and set-up our system so that it was compatible with CAS. Mike created a directory for our project in the Computer Science Department's webspace. In our webspace, Mike set-up a sandbox so that we could run Apache on our server. This allowed us to use our development tools such as PHP and Git, while still using the department's servers. WPI then required us to change ProjectSpot from HTTP to HTTPS so that it would be more secure. Mike set this up for us.

Through his hard work, one of the biggest, most challenging pieces of our project was made easy, which saved us a lot of development time.

### **3.7 Making Improvements**

After implementing all the components of ProjectSpot, we continued working to improve aspects of the site such as the interface and the architecture.

#### **3.7.1 Heuristic Evaluation**

After our system was implemented so that we could go through the system acting as a test user, we conducted a heuristic evaluation with an expert to make sure a user could navigate through the system. Notes were taken at this session, and the comments that the evaluator made were written down. After the heuristic evaluation, changes were made to the system to improve the flow, and the look-and-feel of ProjectSpot.

#### **3.7.2 User Study of Tasks through the Interface**

We tested the interface on six users. We documented the path each user took through the interface to complete a task that was given to him or her. We had each user fill out a questionnaire after each session. The user ranked different parts of the interface on a 1-5 scale. The users were asked to describe difficulties they had with the system, and they were asked to point out which parts of the system they thought worked well. This data was evaluated and used to make improvements to the interface.

### **3.8 Group User Study of Functioning System and Marketing ProjectSpot**

Once CAS was in place, we conducted a user study with groups of WPI students logging into ProjectSpot with their real WPI credentials. Now with CAS, the system was able to pull the student user's actual data. The users were then told that they were in an MQP group together, and that they had to use the system. This final user study had three main goals. The first goal was to evaluate the improvements we made to the system from the last study. The second goal was to observe the practical application of ProjectSpot with groups of students working together, and collect information for our final evaluation of ProjectSpot. The third goal was to show students the potential of our system, and in a way, market it to students for future use.

We wanted to incorporate ProjectSpot with the Computer Science Department at WPI, so we decided that this user study would be the beginning of marketing our project to students. We also created a “Group” for ProjectSpot in the system, in the hopes that a future group of students would continue the project and improve it, in order to make ProjectSpot even more useful to the school.

## **4 Analysis of the Requirements**

Before we could design ProjectSpot, we had to figure out what our users' main needs would be. To do this, we figured out what the current MQP-finding process in the Computer Science Department at WPI was lacking. Then we interviewed students from our target audience and asked those students questions to better understand their needs. Once we had collected data from these users, we analyzed that data to come up with a set of design requirements for the interface of ProjectSpot.

### **4.1 Evaluating the Current Process**

We felt that the MQP-finding process could be improved by making the process more student-facing. By student-facing, we mean that students should be able to see what projects are available (namely MQP-groups formed by other students), rather than just going to a Professor and asking him or her to be an advisor.

We recognized the current system for finding an MQP is for students to approach professors to obtain a project, but in order to improve it we had to find out what users in our target area needed for a better system. For this reason, we conducted a study of users in our target area.

### **4.2 Collecting Initial User Data**

To thoroughly assess the needs of the users in our target area, we set up private interviews with six WPI seniors in the CS department. The student users contained a mix of CS, RBE, and IMGD-tech majors. Each student was asked a series of questions (found in Appendix A). One student had plans to go abroad for MQP, one student had not yet found an MQP, and the other four students were all currently in their respective MQPs.

Each student's responses were typed up, and then analyzed to come up with a set of requirements for the system.

### **4.4 Design Requirements**

From the initial data that we collected, we were able to determine what users would need

from our system. Our initial data allowed us to come up with a set of requirements and design our system around those requirements. We created a diagram of how the users would navigate through our system's interface. This allowed us to see the different tasks and subtasks that would play into our design.

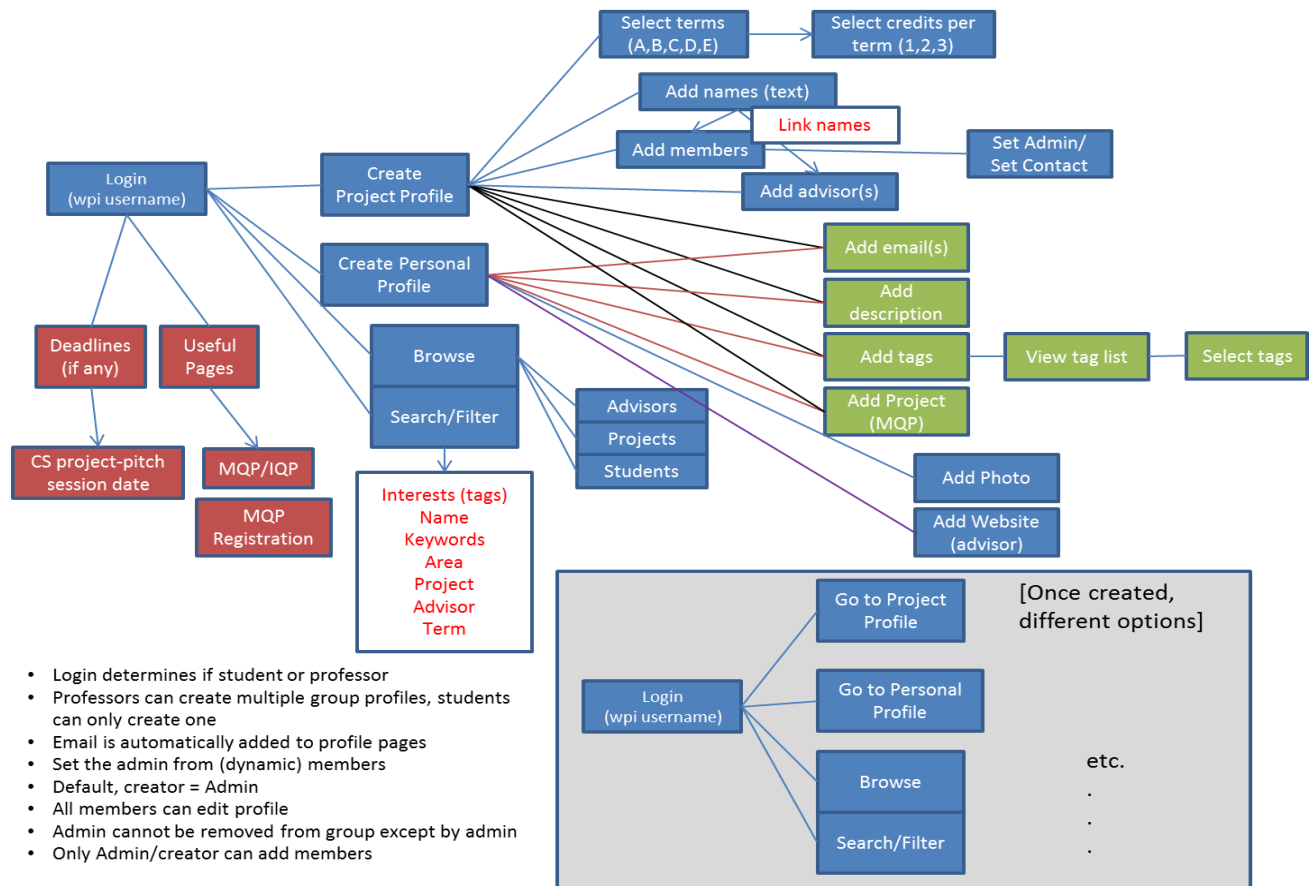


Figure 4: Flow Diagram of ProjectSpot's Tasks

### 4.3 List of Requirements

We came up with this list of requirements for our system:

- View students without projects yet
- View currently-formed project groups
- View projects offered by advisors

- Coordinate groups with other users
- View the topics an MQP is on before joining it
- View the interests a user has before forming a group with that user
- Provide links to WPI resources on the MQP
- Provide information about important dates pertaining to the MQP (such as the Project Pitch Session the Computer Science Department holds)
- Be coherent with the theme of WPI's webpages

## 5 Design

As we discussed in our Methodology, a good design was essential for our project if we ever wanted ProjectSpot to be adopted for practiced use. Our design for the Interface, Architecture, and Database of our system was thought out with care, and implemented according to our models, diagrams, sketches, and plans.

### 5.1 Interface Design

The interface design of our system went through many versions. To create all the functionalities that our system needed to be useful to its users, ProjectSpot had to contain a certain list of components.

First, we wanted our system to be cohesive with the current theme of WPI's webpages. For this reason, we decided to use the same font as on [www.wpi.edu](http://www.wpi.edu), which is "trebuchet ms". To stay with the WPI theme, we also chose to use black text on a white background for our content, which stayed with a theme of simplicity. We decided that the principle colors in our site would be red and gray; the main colors of WPI.

In order to make navigating the site easy for users, we created a top-banner with links to the primary parts of the site: Dashboard, Find, Profile, Group, Invitations, and Logout. From our diagrams of the user's tasks through the system, we deduced that it would be important for users to have quick and easy access to these pages, which is why we included them in the banner.

For ProjectSpot, we had two main task scenarios that we had to account for. The first was that our users would be students without MQPs. These users would need to start-from-scratch and would use ProjectSpot to find whatever information they could. These users would look for already-formed MQP groups, and they may also look for other students and advisors without MQPs to enlist in a group.

Our second scenario was that our users would be groups of students in already-formed MQP groups. These users would be looking for other students to join their group. They also might be interested in finding one or more advisors to advise their group.

For both of our main task scenarios, we determined that all users of ProjectSpot would be interested in knowing what areas of Computer Science the other users were interested in. We also had to design for professors who would want to create multiple projects that they wanted to



advise. We also had to design for system administrators who would need to update the system.

After going through many different scenarios, we came up with all the different pages we would need in ProjectSpot.

### 5.1.1 Landing Page

Before logging in, users would need access to some basic information about ProjectSpot. For this purpose, we created a landing page, which would let users know that they needed to login with their WPI credentials to get into ProjectSpot.

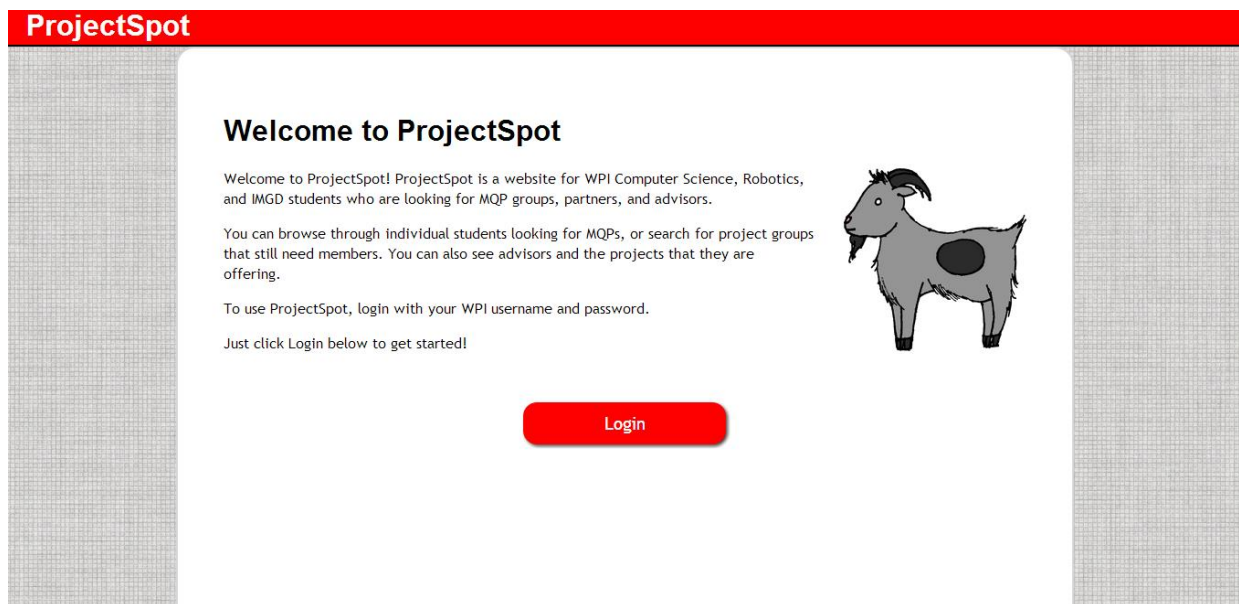


Figure 5: ProjectSpot Landing Page

The main goal of the landing page is to let users know that ProjectSpot is the application that they are about to access. It is also a way to give users some preliminary information about ProjectSpot so that they are better-equipped to begin using the system before they login. This helps us adjust the user's mental-model from the start to be in-line with the model that we designed ProjectSpot around.

After clicking the login button, users are redirected to the CAS server, which prompts users to login with their WPI credentials. This CAS page is one that WPI community members are familiar with, which made it an important part of our design. Users would know that our system is not only secure, but is affiliated with the school. This will help us gain the trust of our

users. After users fill out the form on the CAS login page, CAS handles the authentication, and if the user information is valid, the user is redirected to the dashboard.

### 5.1.2 Dashboard

We eventually came up with the need for a dashboard in our design. In the design we settled on originally for the User Interface, there was no concept of a “dashboard”. We had the “Find page” as the main screen of our system, since finding other students, MQP groups, and advisors is the most important aspect of ProjectSpot.

After some thought, we realized that immediately presenting users with the Find page did not make much sense. If this were the case, users would not know about any other aspects of the system. By being presented with the Find page right-away, it would be difficult for users to realize that there are other aspects of ProjectSpot, such as filling out one’s profile and creating a group. Now by first showing the Dashboard page, we can give the user a ToDo list of tasks to complete in ProjectSpot; which better alerts the user to the different aspects of the site. By having the ToDo list on the Dashboard, users have a better chance of completing their respective tasks successfully; whether that means finding a group, more group members, or an advisor.

We also decided that we needed the Dashboard page to give the user a way to get to any WPI MQP-resources (namely, MQP-related websites), as well as to show any upcoming MQP-related events on-campus. System administrators can change the WPI resources and add important dates to ProjectSpot. These will then be on the Dashboard page for every user. Important dates can be anything from the last day of the term, to the day of the project-pitch session held by the Computer Science Department. It is helpful to have extra information like this in ProjectSpot, because it helps students with finding their MQPs; and the Dashboard is the perfect place to put that information. This was worked into the design also because the header on each page in ProjectSpot was beginning to have too many items on it. We needed links to the Dashboard, the user’s Profile, the Group page, the Find page, and the Invitations page from all locations in the system. Adding Important Dates and WPI Resources would make the banner cluttered and much too large.

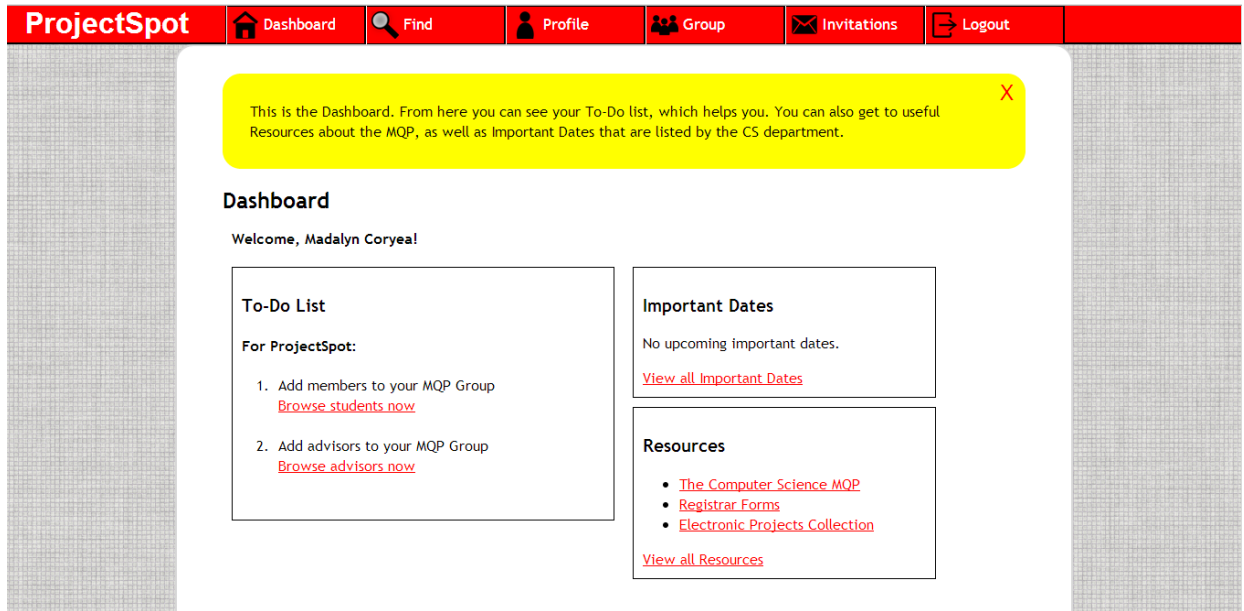


Figure 6: ProjectSpot Dashboard

In the current version of ProjectSpot, the first thing the user sees when he or she logs in is the Dashboard. This provides an informative “buffer” to the rest of the site that improves workflow and aids the user in completing his or her tasks in ProjectSpot.

### 5.1.3 Profile Page

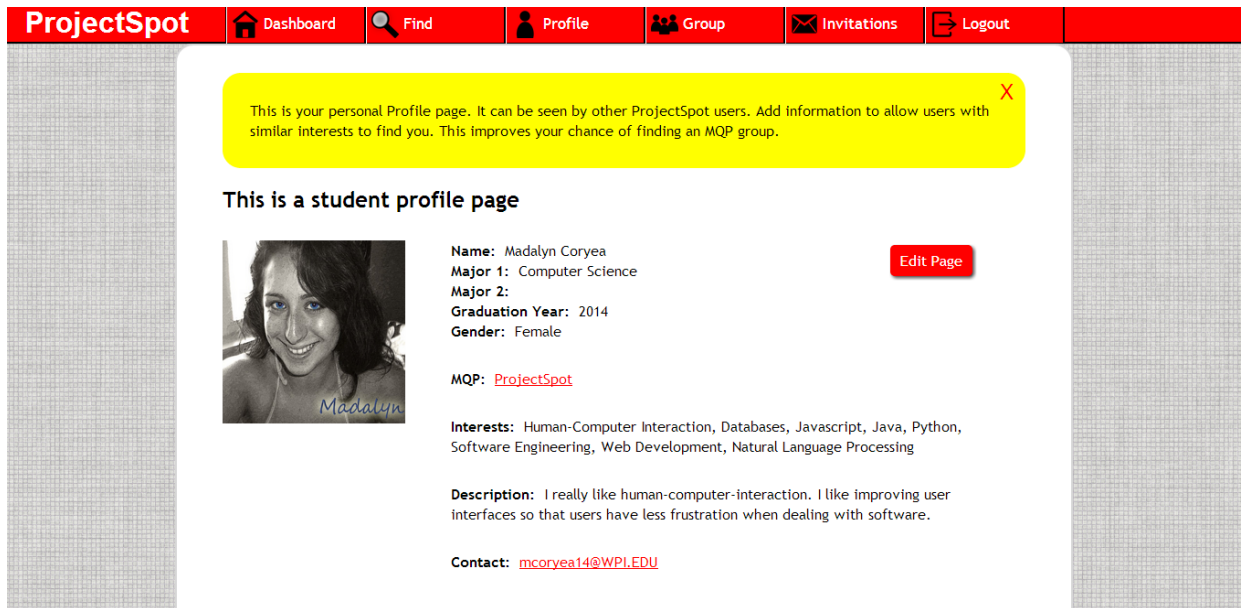


Figure 7: The Profile Page for a User

Each user needs an individual profile to display the user's skills, interests, and general information to other users. To serve this purpose, we designed the "profile" page. The profile page starts out with some general information provided by WPI on the user whose page it is. This information comes from WPI's records and cannot be changed by the user. This information is the user's name, major(s), WPI email, and graduation year. We believe that the user should not be allowed to modify this information with false data; which is already accurate since it comes from WPI. If this information was ever incorrect, the user would need to correct it through WPI, not ProjectSpot.

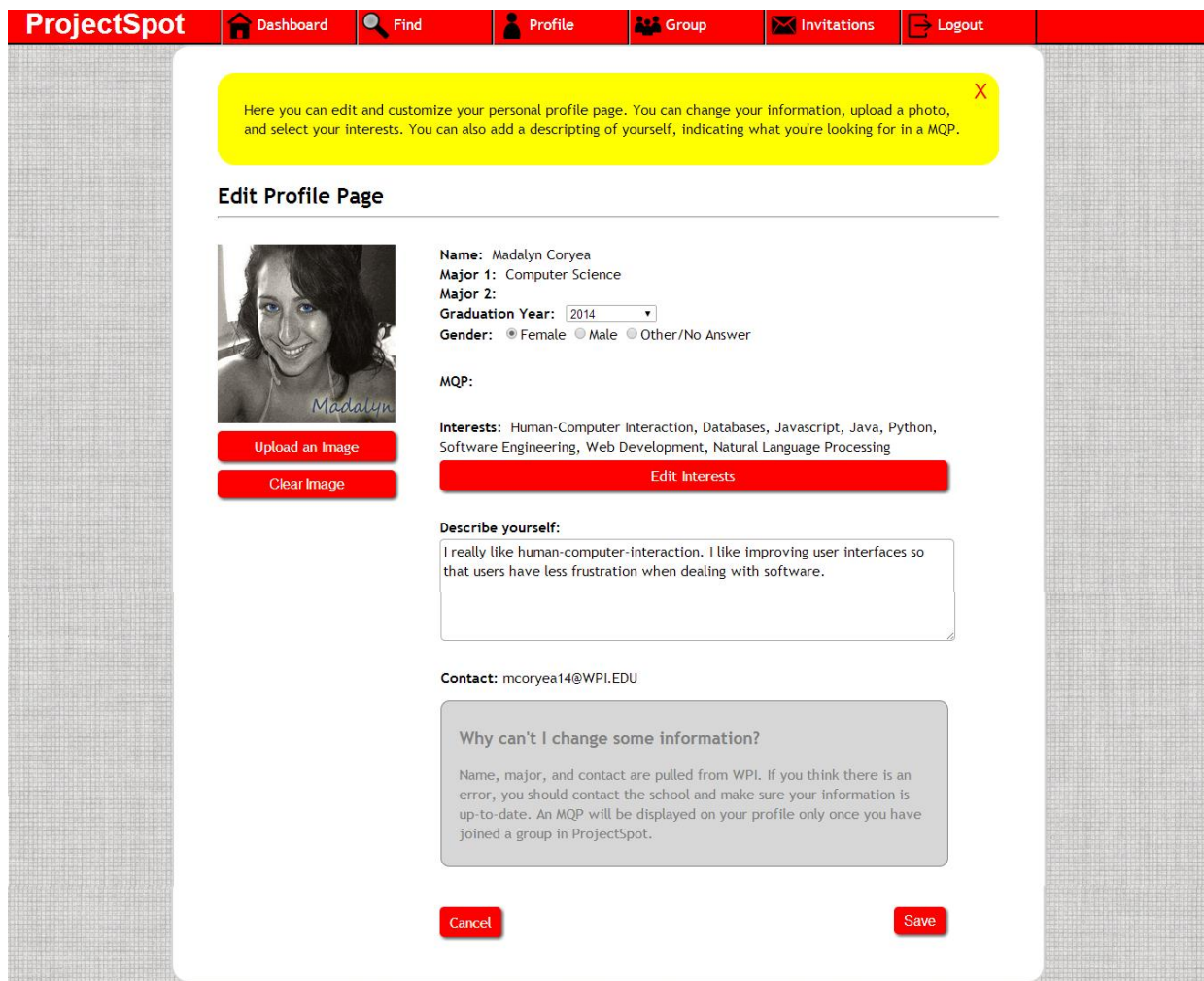


Figure 8: The Edit Profile Page for a User

The user can also indicate his or her gender on the profile page. If the user is in an MQP that is on ProjectSpot, that MQP will appear on the user's profile page. On the profile page, each user can customize his or her interests. There is a separate "edit interests" page that the user can get to when editing his or her profile page. The edit interests page allows the user to check or uncheck boxes that indicate specific interests in various areas of computer science. A list of these interests is available in Appendix I. The user can also add a description of his or herself if desired.

By having a profile page for each user, we make it easier for other users in the system to find out basic information on each person in ProjectSpot. This allows users to make more-informed decisions about who they work with on their MQPs, which will hopefully lead to more effective, better-matched MQP groups. This is the main goal of ProjectSpot.

#### **5.1.4 Group Page**

Since ProjectSpot is based around finding project groups, we decided we needed a main group profile page for each user. This page would be called "group". The Group page appears differently to each user based on several different scenarios.

The first scenario is when a user is already part of a group in ProjectSpot. When this user navigates to the Group page, he or she will view the group to which that specific user belongs. The next scenario is for when a user is not yet part of a group on ProjectSpot. When the user goes to the Group page in this situation, the user is presented with a page with two options. Here, the user can either create a Group page from scratch, or the user can go to the Find page to browse through other groups in ProjectSpot. The last case that can appear for the Group page, is in the scenario where the user is an advisor. Then, this user is shown a list of the groups that he or she is a part of, along with the option to create a new group.



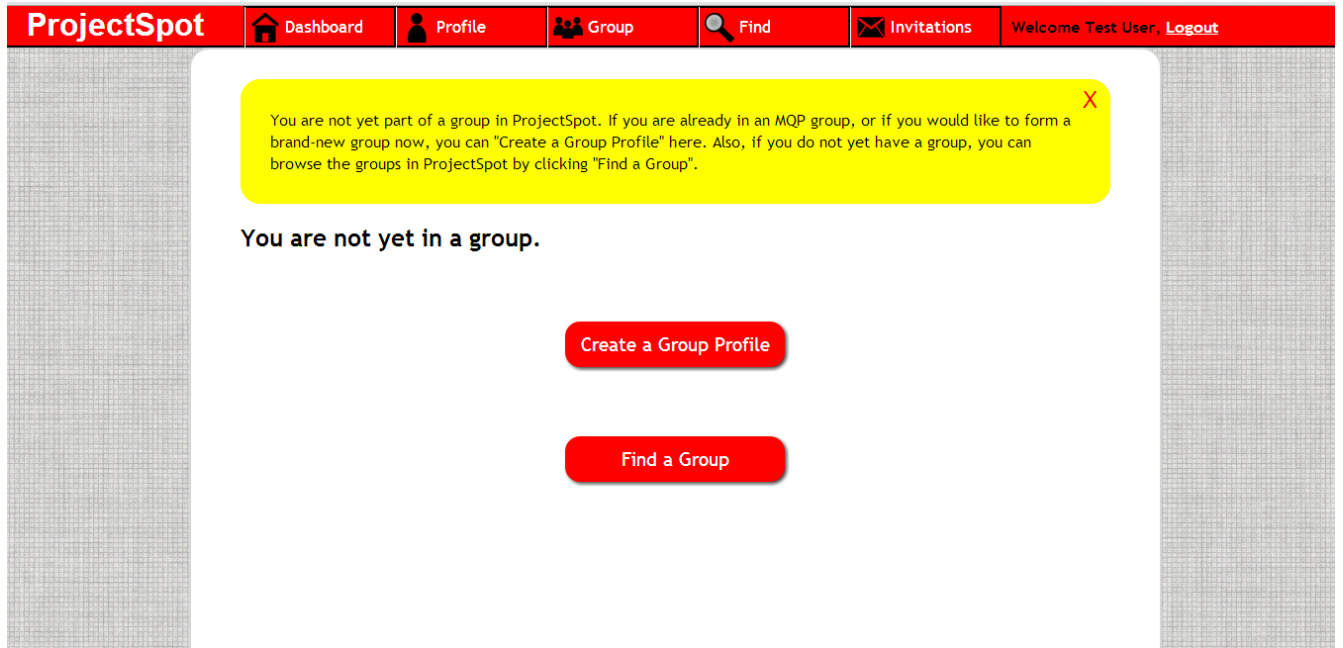


Figure 9: Initial Group Page for a Student User

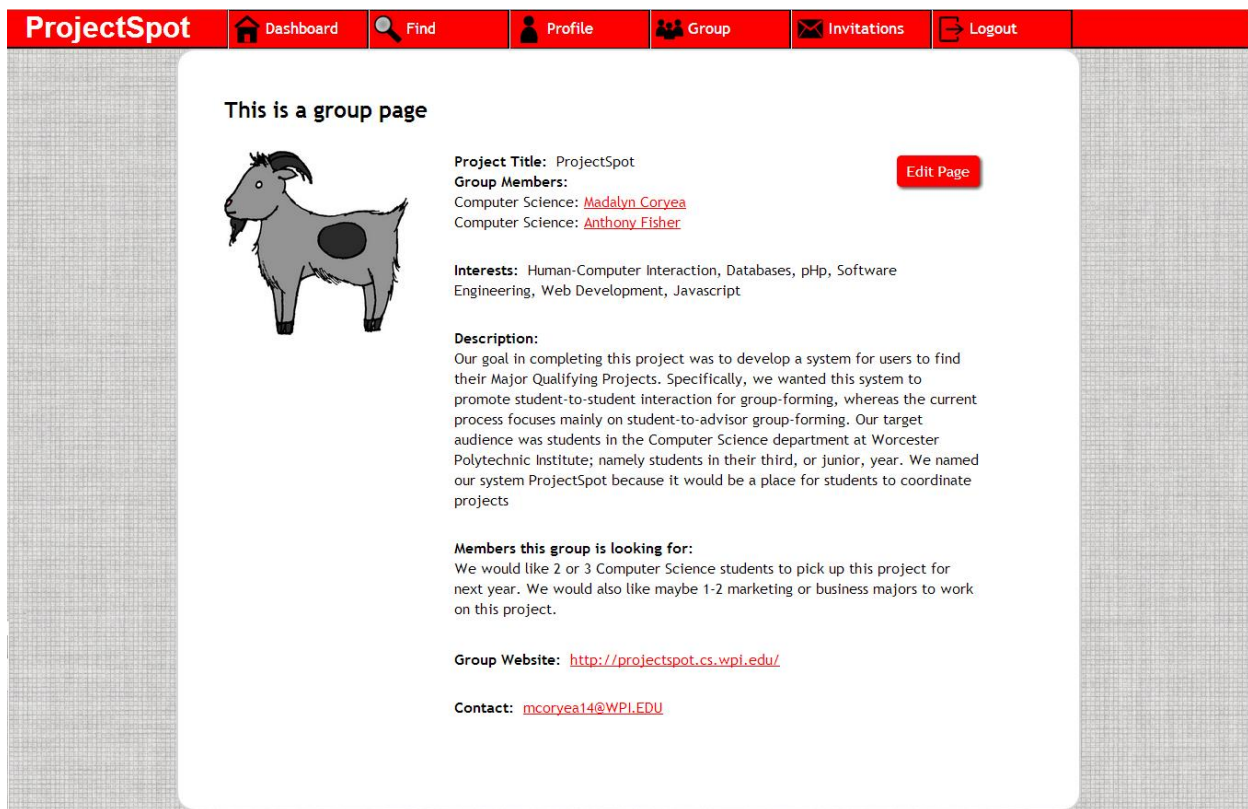


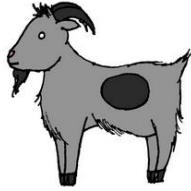
Figure 10: Group Page for an Existing Group

Student users in ProjectSpot may only belong to exactly one MQP group, or not be in a group at all. However, advisor users can be a part of no MQP groups, or they can advise any number of MQP groups. This required a Group page for advisors with multiple groups that allows these types of users to see all the MQP groups that they are a part of at a glance. From this version of the Group page, the user can then click on any one of the groups on this page, and then be directed to the more “traditional” Group page for that specific group.

On a traditional Group page, the user will see the name of that group, its members, a description of the project, what new members the group is looking for, and the areas of Computer Science that that project pertains to. There will also be a contact email for the group listed. From the Group page, a user can click on the names of all the members of the group to view those users’ user-profiles.

**ProjectSpot** Dashboard Find Profile Group Invitations Logout

### Edit Group Page

  
Upload an Image  
Clear Image

**Project Title:** ProjectSpot

**Group Members:**  
To delete this group, all group members must be removed.  
Computer Science: Madalyn Coryea [Remove](#)  
Computer Science: Anthony Fisher [Remove](#)

**Interests:** Human-Computer Interaction, Databases, pHp, Software Engineering, Web Development, Javascript  
[Edit Interests](#)

**Give a description of your group:**  
Our goal in completing this project was to develop a system for users to find their Major Qualifying Projects. Specifically, we wanted this system to promote student-to-student interaction for group-forming, whereas the current process focuses mainly on student-to-advisor group-forming. Our target audience was students in the Computer Science department at

**Majors, skills, and requirements for new members:**  
We would like 2 or 3 Computer Science students to pick up this project for next year. We would also like maybe 1-2 marketing or business majors to work on this project.

**Group Website:** http://projectspot.cs.wpi.edu/

**Contact:** mcoryea14@WPI.EDU

[Cancel](#) [Save](#)

Figure 11: Edit Group Page for a Group

### 5.1.5 Invitations Page

We decided that we needed a page where users could see which groups they were invited to join, and see which users they had invited to their groups. This stemmed from the requirement of users needing the ability to use ProjectSpot to coordinate project groups with other users. To fulfill this requirement, we created the Invitations page.

The main reason for creating the Invitations page was that we needed a way for already-existing group members to form their groups in ProjectSpot's system. We needed some form of regulation. If any user could simply add any other user to a group in ProjectSpot, users would be added to groups against their will constantly. This would be a major problem, because ProjectSpot only lets a user be a part of one MQP group at a time to prevent confusion. For this reason, ProjectSpot does not allow users to be added to more than one group at a time. This is the motivation for having our invitations system.

A user in a group can request a ProjectSpot user to join his or her group; provided that that user is not part of another group. Conversely, a user without a group can request to join any group in ProjectSpot. The key factor here, is that a student must *accept* an invitation to join a group, or a group must *accept* a request for a student to become a group member in ProjectSpot. The same goes for requesting to join and sending invitations for advisors. User can also reject invitations from groups that they do not wish to join. Group members can reject invitations from users that they do not want to be in their group. This provides a suitable buffer between users and groups. The buffer allows for this process to be regulated in a manner where the user is in control.

Other members in a group can revoke invitations that a member has sent out. Students and advisors can also revoke requests to join that they have sent to groups. This is so that actions are "un-doable"; and since every member in a group is considered to be on the same level, each group member needs full-control over all group actions.



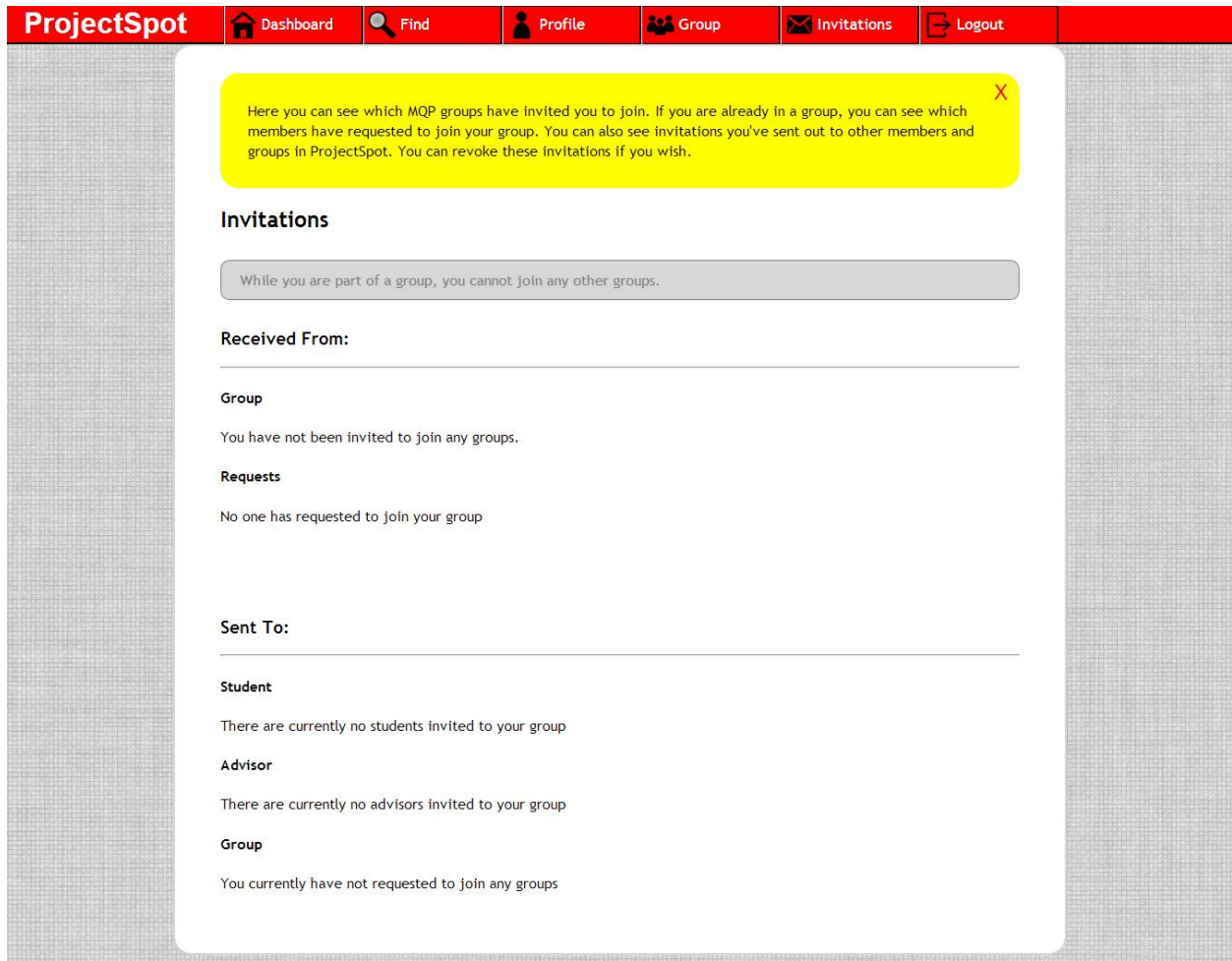


Figure 12: Invitations Page for a User in a Group

### 5.1.6 Find Page

Arguably the most important page in the ProjectSpot web-app is the Find page. From here, users can view all users (both students and advisors) and all MQP groups in ProjectSpot's system.

Users can then further refine the content of this page, which initially displays all of the information in ProjectSpot's system. There are checkboxes on this page that allow the user to filter the information by category: Students without MQPs, Students with MQPs, Advisors, and MQPs. Users can refine results even more by typing into the filter "search bar". The text the user types is matched-up automatically, and the information is filtered to show only results that match what the user has typed. All of this happens instantly, so as soon as the user types a character in the search bar, the list of results is updated.

**ProjectSpot** [Dashboard](#) [Profile](#) [Group](#) [Find](#) [Invitations](#) [Welcome Test User, Logout](#)

This page lets you browse all students, advisors, and MQP groups. Use the search bar to filter even more by typing in a name, major, or area of computer science that interests you. Use the checkboxes below the search bar to set which kind of results you want to see.

### Find students, advisors, and groups

Filter:

Students without MQPs  Students with MQPs  Advisors  MQPs

#### Students

Status	Name	Major 1	Major 2	Interests	MQP
Student	<a href="#">Anthony Fisher</a>	Computer Science		Algorithms	<a href="#">ProjectSpot</a>
Student	<a href="#">Madalyn Coryea</a>	Computer Science			
Student	<a href="#">Test User</a>	Computer Science			

#### Advisors

Status	Name	Department	Interests	MQPs
Advisor	<a href="#">David Brown</a>	Computer Science		<a href="#">ProjectSpot</a>
Advisor	<a href="#">Gary Pollice</a>	Computer Science		

#### MQP Groups

Status	Title	Department	Areas	Advisors	Members
MQP Group	<a href="#">ProjectSpot</a>	Computer Science	Algorithms		<a href="#">Anthony Fisher</a>


 **WPI** WPI  
ProjectSpot © 2013

Figure 13: The Find Page in ProjectSpot

All of the user names and group names in ProjectSpot are hyperlinks. Because of this, the Find page makes it easy to see all the users and groups and click on whichever one the user wishes to explore. The user can easily get to all of the different profiles in ProjectSpot, giving the user the ability to learn more about the available options when searching for an MQP.

### 5.1.7 Admin Pages

The final pieces of ProjectSpot are the admin pages, where users who are flagged as administrators can make modifications to the system, such as adding new upcoming dates,

important links, as well as promoting users to admin status, and removing users who are no longer needed. These are all simple forms that insert data into the database, and act as a layer of abstraction from modifying the database directly. Screen shots of each Admin page in ProjectSpot can be found in Appendix I.

## 5.2 Architecture Design

The main goal of the design for our system architecture was to allow for an efficient way to communicate between our user interface and our database. Because we used CodeIgniter as our framework, most of the basic functionality and security for ProjectSpot was provided for us.

When the user visits the site, the system will process the URL. Using this data, it routes to the proper controller, and then the proper function inside that controller. Once inside the controller, the URL and any sessions stored hold all the information needed in order to obtain data from the models. This has been designed in such a way to limit how many database calls are needed to be done. In addition, joins are used so that instead of having to constantly query the database to build the data structure the view needs, we can limit it as much as we can.

Once all the data required is obtained, the controller will then process it, forming an array (which can be thought of as a “list” of data) that the view can then use when displaying the data, as specified by CodeIgniter’s documentation. Once the data has been fully processed and organized, we pass it on to the view, which then processes that information, displaying it to the user.

## 5.3 Database Design

Our initial database design was based on how Wordpress organizes their database. In this design, everything revolved around one core table, the profile table. Because the core data structures of our system were all different types of profiles, we would use a single table that holds all those profiles. Each data entry in this table would contain information that every profile needed, such as a name, a description and other attributes. In addition, each item would contain a unique id, as well as an identifier of what type of profile that entry was. This would then be connected to a secondary table, which would hold any other attributes a profile might have. Each entry in this table would represent a specific piece of data about the item it is tied to.

They would include the id of the item it describes, what piece of data the entry specifically refers to, and the data itself. This would allow programmers who inherit the project to be able to quickly and easily add different types of profiles to the system, while also allowing them to add more fields to existing models, without having to change the database schema. On the other hand, the system would quite possibly run into issues when it started handling more users than our initial estimates, because every single profile attribute is stored in a single table. On top of that, because everything was contained in one table, organization of our data structures could very easily become confusing, and maintainability would become an issue.

Our final design was one that was based on how databases are traditionally designed. Each profile type, currently user profiles and group profiles, would have their own tables, with explicitly defined attributes. It also would follow the Boyce-Codd Normal Form, where the database is normalized. A normalized database is one that does not include redundant data. This means that when something is changed, it only needs to be changed in one place. This prevents orphaned data from remaining in the database when something is deleted, and makes entering data into the database simpler. Each type of data, such as a group, or a profile has its own table, with columns describing it explicitly defined.

With the database more rigidly defined, future system administrators of the project would be able to quickly look at the schema and know exactly where things were located in the database. In addition, because type checking would be done on the database side, rather than client or server side, security would be increased, as the system would refuse data that was deemed unacceptable. Searching would also be much faster, because string matching is much slower than other types of matching in a MySQL database, and the previous design was made entirely of strings. Unfortunately, this would make our system harder to expand upon, as any changes to models would require the database to be refactored, increasing the chance of data loss if done incorrectly.

In the end, we decided on using the second database design for our system. While this would make our system less flexible when being extended, the pros far outweigh the cons. The first database design, while flexible, was a little bit over-engineered. We do not expect future system administrators to make many large changes to the database, other than adding new tables; and as long as backups are made before making any changes, the chance of data loss would be greatly reduced. In addition, we decided that it would be better if developers were able to clearly

see how the system is designed when they inherit the project. Finally, we wanted our system to be fast, which the second design assists with. We decided that the first design solved one problem while making three others, while the second design caused one problem and solved three others. For the sake of clarity, having a database that was slightly more rigid in its design would be an acceptable tradeoff. It would also be faster when the system started handling more users.

## 6 Implementation

Since we stayed true to MVC (Model-View-Controller) in the implementation of ProjectSpot, we were able to separate-out the main components of our system and implement them in environments that were isolated from one-another.

### 6.1 Implementing the Interface

The interface for ProjectSpot was implemented piece-by-piece. First, pages were made for each of the main parts of the system. HTML mockups and wireframes were made for group profiles, student profiles, and advisor profiles. Then, pages were made that would allow the user to edit each of these pages. Once all of these worked and functioned according to the interface designs, they were uploaded to the ProjectSpot server and placed in the appropriate directories.

In the file structure of our system, we followed the MVC design pattern. This means, that we have folders for models, views, and controllers in our system. All of the pages for the interface are in the views folder. The views folder is set-up to work with the CodeIgniter framework, and it has separate folders for the main parts of the interface, such as group, invites, and profile. This works well because users need to access different pages depending on where they are in the system. CodeIgniter knows to pull the correct template page out of each folder and show it to the user correctly in these situations. This also keeps our code and file structure organized in a way in which it is easy to do development.

The main process of implementing the interface was to get the basic pages working with the back-end of the system, and then go back and write the CSS that would alter the appearance of the site into a cohesive theme. All of the CSS (or styling) for our site, is kept in the “stylesheets” folder that is one level down from our root directory. All of the views in the view folder access the CSS files in the stylesheets folder, in order to obtain the appropriate styling when displayed to the user.

### 6.2 Implementing the System’s Architecture

The core of our program’s architecture is the CodeIgniter Framework. CodeIgniter comes preconfigured to handle an MVC design paradigm, which greatly sped up development time for features. Because everything was already organized how we were going to organize it,

we did not have to worry about inheritance and including proper files. When we needed load another module, it was handled by a function call native to CodeIgniter.

The models of our system were handling data manipulation. Most of the models only handled getting, adding, setting, or deleting data in the database. When CodeIgniter loads, it loads the various configuration files, and retrieves information for connecting to the database. This way, we can use the included functions to interface with the database, instead of connecting to the database manually.

Specifically, CodeIgniter comes preconfigured with a library called ActiveRecord, which handles database interactions. Instead of having to manually write out SQL queries, we can call functions to build them for us. Once a query is built, it is sent to the database, where it is executed and the result is returned. One of the best features of ActiveRecord is that it automatically sanitizes data that gets put into the database, increasing security and saving time.

The controllers are the largest part of our program, and they were coded with a specific pattern that each controller shares. Every main section of the site, such as the dashboard and group profiles, has its own controller. These controllers are designated to each view to control the dynamic content that is displayed in that view. Inside each controller are functions that handle specific actions, such as viewing a specific group, or adding a new one. These functions largely share the same structure. Initially, we load any models or helper functions that we need for obtaining data. After that, we make queries to the models in order to retrieve any information that is needed. For instance, when displaying a specific group to the user, a function is called that takes in the id of the group being viewed. This function then retrieves of all the information about that group. Once obtained, that information is stored in an array that is correctly formatted so that the view can use that data. The array in question is an associative array that is sectioned off by each piece of data. In this case, the array has an index called 'group\_item', which is where the information is stored about the specific group being viewed. Once the array is fully built, it is passed to the view, where it can be processed.

The views are what the users will see, and as such, are mostly HTML markup. We wanted to try and keep as much PHP out of the views as we could, as the MVC paradigm dictates. In theory, the code for a view should not do any operations that require more computation than that for accessing the information in an array. When the array from the controller is passed to the view, it is processed and each root index is converted into its own

variable for the view to use.

### 6.3 Implementing the Database

The design of our database is quite simple, and it follows a design schema that many web applications use. Traditionally, PHP web applications are hosted on what is called a “LAMP Stack” which stands for Linux running an Apache web server, with a MySQL database, serving pages using PHP. It is standardized, so we know it works, and many web developers will know how to work in such an environment. Because of this, we knew we wanted to use a MySQL database from the beginning of the project, due to its ease of use, and how widespread it was. It also was suitable for our needs, in terms of speed and how much data we would be holding, which, compared to other commercial web applications, is not that much.

One thing we had to keep in mind with our database design was how extendable it was. One of the initial designs we considered was built around there being one large table where we stored every piece of data the system used. This would have been simpler to expand upon when adding new features, as developers would not have to modify the database schema. Unfortunately, it proved to be inefficient, and if something needed to be changed, it would prove to be rather complicated for someone not familiar with the system to understand. In the end, we decided to go with a design where every different collection of data, such as groups, users, etc. would have their own tables. If features needed to be added, new tables would have to be made, or existing tables would need to be modified.

With that in mind, each table in our database has an id field that will automatically increase as new data is entered, so that we can always access specific pieces of data. The largest tables are “ps\_groups” and “ps\_users.” As expected, these hold the data for groups and users respectively. Contained within these tables are values that represent the various attributes tied to those sections of the web application, such as descriptions, user images, and other bits of data. We also have a few smaller tables, “ps\_tags” and “ps\_majors,” that hold the values for interests and majors.

In order to tie these types of data together, we have a number of relational tables that allow us to have one to many connections between types. For instance, if you wanted to invite a user “Jimmy” to the “Spongification” MQP group, you would add an entry to “ps\_group\_user\_rel” table, consisting of the group id, user id, as well as his invitation status.



Tying majors and interests to users and groups happen much in the same way.

## 6.4 Implementing CAS (Central Authentication Service)

In order to handle authentication with users, we had a number of options at our disposal. We could have written an authentication system ourselves, but due to the nature of software security, it is considered bad form to write something custom. It is better to utilize a system that is already proven to be secure. At the same time, it would be confusing for users to have a separate login for a WPI service. After discussing with the System Administrator in the WPI CS department, we figured out a way to utilize WPI User Accounts with our system, all with a small amount of coding.

Setting up the authentication was a simple process. The codebase was set up with a landing page outside of our web application. This page would be accessible to everyone. One directory down from the root is protected by an .htaccess file, which requires that a user be authorized first. If the user is not authorized, he or she is redirected to the CAS server. The user is presented with the CAS login page, where he or she must enter a username and password.

While the actual steps needed in order to implement CAS were quite small, ultimately being only three lines of code, we encountered many issues with getting it to work. There was a great deal of interaction between us and the department handling CAS. The amount of interaction required was a source of much frustration. Ultimately, through the use of a .htaccess file, we were able to enable user authentication with our system. Once the authentication was working, the system was able to retrieve the user name of the authenticated individual and look up the correct information in order to populate that user's ProjectSpot account.

## 7 Results & Analysis (Design Evaluation)

After ProjectSpot was implemented to the point where all the basic functionality was in place, we conducted a user study to ensure that users could make their way through our system. We wanted to verify that the user's mental model lined up with the user interface that we had created. It was also important to make sure our project's goal was attainable to our users: find information on projects, students, and advisors in the Computer Science Department at WPI.

By observing users in these studies, and by collecting data from those users, we were able to evaluate the effectiveness of our design. We were also able to come up with a set of metrics that users could rank different segments of our design with. These metrics provided us some insight into the usefulness of our system, as we were able to analyze the data obtained through the use of those metrics. The details of this analysis are presented below.

### 7.1 Interface and Usability

After designing the interface based on user needs from our first study, and then conducting the heuristic evaluation on our completed interface, we were ready to test the interface on potential users in order to understand the path that users would take through the system to complete their tasks.

A total of six users participated in the study. There were two scenarios that we chose to test on students. The first scenario was based on a persona "Test User" who was looking for an MQP group. The second scenario was based on the persona "Test User" who, this time, had one groupmate and was looking for an additional groupmate as well as an advisor.

Out of the six users, three users were evaluated while they acted out case one. The remaining three users were evaluated while they acted out case two. These are the two cases presented to users:

#### **CASE 1:**

You will be playing the role of a fictional student in this scenario. Test User is a WPI junior Computer Science major. Test User has no idea what to do for his/her MQP at WPI. Test User likes Databases, Graphics, and Web Design. Test User does not know much about ProjectSpot, but wants to use it to find an MQP group.

Your job is to act as Test User. Login to the system, create a profile for Test User, and see if you can find out how to find an MQP.

### **CASE 2:**

You will be playing the role of a fictional student in this scenario. Test User is a WPI junior Computer Science major. Test User and his/her friend Chelsea Fishwall have already formed an MQP group called “Android App Maker”. Test User and Chelsea want to find one other student that knows about java. They also want to find an advisor for their project. Test User does not know much about ProjectSpot, but wants to use it to find an advisor and team member for his/her MQP group.

Your job is to act as Test User. Login to the system, create a profile for Test User, and create a group profile for Android App Maker. Try to use the system to find an advisor and a team member.

The users were studied as they navigated through the system. They were asked to simply explain what they were doing as they were doing it and to think out loud if they felt comfortable doing so. At the end of each user session, the participating user was asked to fill out a questionnaire; allowing them to rank various parts of the interface on a 1-5 scale. There were also some open-response questions at the end of the study. This questionnaire along with the open-response questions can be found in Appendix E.

## **7.2 Results of User Study of Tasks**

From our user study, we received useful suggestions from users, as well as some raw data that came from the users ranking different parts of ProjectSpot’s interface. This data was then put into Microsoft Excel so that it could be analyzed. All of this raw data can be seen in its entirety in Appendix F.

Since each part of the interface was ranked on a 1-5 scale (where in every case, “1” was considered “worst” interface and “5” was considered “best” interface), it can easily be seen by the data how “good” the users felt each part of the system was.

The overall feeling toward ProjectSpot was positive. Users ranked the system highly in all categories. For this reason, our study became less about finding out what was drastically wrong with the system, and more about where were the minor areas that needed to be tweaked.

### 7.3 Final User Study

Before we concluded our project, it was important to do our final user study to determine the feasibility of ProjectSpot being adopted by the WPI Computer Science Department. We designed a user study in which groups of three students would work together to create one Group in ProjectSpot that they all must join. Similar to our previous user study, users were given a specific task scenario. They were also given a list of tasks that they needed to complete. However for this study, users were able to login with their real WPI credentials, so each user's experience was customized to him or her. The previous user study had been performed with each user using the same test account, whereas this study gave us a better idea of the real scenario for a user using ProjectSpot. This also allowed us to see users login to ProjectSpot with CAS, and observe how the final system worked.

The users were told to work together and communicate with each other as they used the system, in order to emulate how group-forming with ProjectSpot would likely work in a practical application. Below is the scenario and task list that users were presented with:

You are friends. You all need an MQP and have decided to work together. You have come up with an idea for an MQP (you may change this if you like). You want to build an application that will measure the performance of any software. You have decided to call it "The Performance Calculator".

Tasks (in any order):

- Create a group in ProjectSpot for your group idea (The Performance Calculator)
- Update your group page to reflect information on the project
- Each of you must Login to ProjectSpot and update your user profiles
- Have all group members listed as a part of this project group

Users were told that they could spend as long as they desired looking through the system. The users were asked to indicate when they felt that they had finished completing the tasks that were given to them.

We studied two groups, each with three users. Each user was asked to do a separate analysis of his or her experience with the study in the form of a questionnaire (see Appendix G). We then took the questionnaires from each of our six users and analyzed the results. Again, we elected to use a ranking system on a 1-5 scale in combination with an open-response section at the end. Although the questions asked for the ranking part of the questionnaire were different from those asked in our previous user study, the open-response questions (Did anything overwhelm you? What was straightforward? What was not straightforward? What did you like? What could have been better?) remained the same, as we felt they provided valuable feedback in all situations.

#### **7.4 Results of Final User Study**

From our final user study, we found that groups were able to work together to create project groups in ProjectSpot's system. While we recognize that there is room for improvement, we deduced from this study that ProjectSpot provides the necessary functionality for its users to form project groups. Both of the groups we studied were able to create a Group page in the system and have all of the users join the group. The process made sense to the users, and users even got excited when they saw the other users' "requests to join" appear on their respective invitations pages.

Five out of the six users ranked a 4 or a 5 for "Did the process of creating a group make sense?". This result was the same for the question "Did the process of adding members to the group make sense?". Users also ranked ProjectSpot highly for questions pertaining to whether or not users enjoyed using the system, and questions that asked users to indicate how likely they would be to use ProjectSpot if it was available for use. This makes us confident that users will understand the system well enough to use it successfully, which, in turn, will help the students in the Computer Science Department coordinate MQPs. The full results of this study can be found in Appendix H.

## 8 Conclusions

### 8.1 Future adoption by the Computer Science Department

We based our project around the need for a system that would allow for students to better-coordinate their senior projects. Since the MQP is such an important capstone of one's educational experience at WPI, finding the right project with the best-suited group and advisor is vital. When we began our project, we believed that students should be given the chance to review all their options for an MQP in order to make an informed decision about who they will be working with for the majority of the academic year.

We studied users, we designed the system, we implemented the system, and we studied users again. We set-out to create the best possible system within our time constraints of three terms to fill this need. However, none of this would directly benefit any of the students in the department unless we obtained the Computer Science Department's commitment to ProjectSpot after our graduation. This is why we told the department head and the professor in-charge of coordinating Computer Science MQP groups about our project. Our goal was to get their approval for the department to adopt ProjectSpot as a tool available to students.

### 8.2 Future Work

In three terms, we were able to create a fully-functioning system that users could login to and look for project groups, students, and advisors. However, if we had an unlimited amount of time, there are many more features we would have added to the system. There are also several improvements we would have made. This is why we feel that this project could be continued by future groups at WPI. In this section, we aim to discuss future work for ProjectSpot. We hope that our remaining, unfinished goals will eventually be made a reality by another group.

#### 8.2.1 Recommender System

When searching for an MQP group, it would be useful to use a weighting system where certain results are displayed first, driven by information held in the various profiles, to help sort search results. A similar system can be used when searching for more people to add to a group. When a user is searching for something in the system, he or she should be able to choose to filter in different ways, including this weighted search. For instance, say a user has indicated in the

individual profile that he or she is skilled in web development and database design. In this scenario, two projects are in the system. Both are web development projects, but one focuses on databases, while the other focuses on interface design. When the user sorts his or her search results by this specific algorithm, the system will see what skills the user has, rank the two projects according to those skills, and then present the user with the two projects in the system, with the project focusing on databases in a higher spot than the project focusing on interface design.

### **8.2.2 Improved Notification System**

Originally we had wanted to incorporate ProjectSpot with email. We had even thought about creating a messaging system in ProjectSpot, where users could send messages back and forth. We thought that this would be especially useful to students without MQPs who wanted to coordinate with individual students in the system; instead of just students coordinating with project groups, which is how the system currently works.

In user studies, users have mentioned that if they were to use ProjectSpot, they would likely not login every day to check their invitations. They indicated that having ProjectSpot send them an email as soon as an invitation was received would be useful. Future groups that work on this project should consider implementing this, as it may encourage users to keep returning to ProjectSpot. It will also help users coordinate projects in a faster, more efficient manner. However it is important to remember, that the overuse of email is becoming a burden on many people. So despite the advantage of immediate notification, it is important to consider the disadvantage of email overload and people ignoring any emails sent from the system. It is important to find a suitable balance between sending email notifications, and not sending any emails at all.

### **8.2.3 The Cycle of ProjectSpot**

One of the most important parts of any system, especially one in education, is that it remains up-to-date. Since students cycle through college every four years or so, and thus it is necessary to remove these students from the system. Also, since ProjectSpot has project groups that end after a year, these groups need to either be removed, or updated for new members to take them on. Currently in place are “Admin Pages” that will allow system administrators to delete both users and groups from the system. Unfortunately, we did not have time to design and

implement a sophisticated system that would update groups and users in ProjectSpot automatically. One of the most important things for the next group to work on this project would be to develop an intelligent way to clear out old groups and old users.

This is a delicate issue, since not every MQP group is the same. It would be dangerous to have the system automatically clear out old groups at the end of the academic year. Some MQPs will be active for three terms, but some may go over this and even take more than a year to complete. There is also the opposite problem where a group may only be active for one term, and then it sits idle in the system for the remainder of the year. This could confuse students that wanted to join the group. Students would request to join the group in ProjectSpot, and never hear a response for the project would already be over.

There is a similar problem for students in the system. Even though students normally graduate in four years, some make take five years or even more to get their degrees. There is also currently no way to “close” a project group. The group members should be able to indicate that they do not want any new members to join. Groups can be deleted once all members of the group leave, but we thought it might be an important for ProjectSpot to have a state where groups can be archived. This way, groups could be re-opened if new member wanted to join in subsequent years, or if the current group members decided to change the project.

If this situation is not dealt with in a suitable manner, it could mean the downfall of ProjectSpot. While the system may work well now as there is no information yet in the system (new or old), after this year it could be cluttered with old projects and graduated students if the system administrators do not remove them. It is unrealistic to have the system depend so much on the system administrator for this, which is why this issue should be carefully evaluated and solved by the next project group.

### **8.3 Our Experience with the Project**

For the past year, we have committed ourselves to this project. As we have said from the beginning, we wanted to leave a system behind that would help our fellow students in the Computer Science Department. That is why we set out to develop this piece of software that became ProjectSpot. It took a great deal of time and effort, but we now have a completed system that is fully functional. In order to reach this great achievement, we had to apply many of the skills that we learned here during our time here at WPI. We also gained new skills through the



process of researching, evaluating, and creating ProjectSpot.

### **8.3.1 Skills and Tools Used**

From our Computer Science courses here at WPI, we learned about the importance of design patterns, and with that, the importance of developing software in a structured, organized, and well-documented manner. Since we were both working on the codebase multiple times a day, we had to make sure the code was well-structured and understandable to all who would see it.

One important skill that we applied to this project was the use of source control. Source control is so important, because it tracks the changes to the system so developers can experiment with new features without having to jeopardize the working version of the system. It also allowed both of us to work on the codebase at the same time, which was crucial to the success of our project. By use of our previous knowledge, we were able to set up a git repository and have it hosted on GitHub. This initial set up of source control was one of the fundamental skills we exercised when working on this project.

### **8.3.2 Skills and Tools Learned**

We learned that it is important to understand what requirements you have for a system before you begin to build it. We found this when we designed our entire system around a database that followed the Wordpress database structure. Once we found this would not suit our needs, we had to stop our project to redesign a new database. While it was interesting to try out a novel database design structure, it would have been better if we had just followed the conventional methods for designing a database from the start.

From our Advisor, Professor David C. Brown, we learned the importance of obtaining measurable information from users. While asking users questions such as “What was not straightforward [about the system]?” (See Appendix E) gave us good information, questions that had users rank the software on a 1-5 scale really gave us the most useful information. By analyzing these results, we could determine which parts of the system we needed to focus on improving. We learned that user study questionnaires were much more than just a series of questions. We had to spend a great deal of time planning and thinking out our questions. We learned a very important fact: decide exactly what you want to learn from users. Then plan the study. Everything has to be based around what information you want to learn. This knowledge

will help us in our careers when assessing the needs of our customers.

With this experience, we learned how to work on a large-scale software project. It was important for us to communicate effectively and work together. There were many different pieces that went into this project, and we had to keep a schedule throughout the course of the year so that we had enough time to complete everything. It was definitely difficult to divide the work equally between us with such a large scope of tasks we needed to accomplish. We learned about effective time management skills, and we learned how to force ourselves to adhere to strict deadlines.

## 8.4 Access and Use of the System

The system is currently available for use. It is hosted at the domain <https://projectspot.cs.wpi.edu/>.

All of the code for ProjectSpot is available from GitHub. It can be accessed from <https://github.com/irreama/ProjectSpot>. The code can also be accessed by logging into projectspot from the cs.wpi.edu servers.

New members should add themselves to the alias [projectspot@wpi.edu](mailto:projectspot@wpi.edu) for maintenance of the system. This alias will be accessible through Michael Voorhis, the System Administrator and Lab Manager of the Computer Science Department at WPI.

## 9 References

- Apache Contributors (2014), "Frequently Asked Questions," [http://wiki.apache.org/httpd/FAQ#What\\_is\\_Apache.3F](http://wiki.apache.org/httpd/FAQ#What_is_Apache.3F), Accessed 2/28/2014
- S. Chacon (2009), "1.1 Getting Started - About Version Control", *Pro Git*, <http://git-scm.com/book/en/Getting-Started-About-Version-Control>, Accessed 2/28/2014
- S. Chacon (2009), "1.2 Getting Started - A Short History of Git", *Pro Git*, <http://git-scm.com/book/en/Getting-Started-A-Short-History-of-Git>, Accessed 2/20/14
- S. Chacon (2009), "About," *Pro Git*, <http://git-scm.com/about/distributed>, Accessed 2/28/2014
- S. Chacon (2009), "What a Branch Is," *Pro Git*, <http://git-scm.com/book/en/Git-Branching-What-a-Branch-Is>, Accessed 2/28/2014
- CodeProject.com (2008), "Simple Example of MVC (Model View Controller) Design Pattern for Abstraction", <http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design>, Accessed 2/21/14
- F. Coelho, "PHP-related vulnerabilities on the National Vulnerability Database", Coelho, [http://www.coelho.net/php\\_cve.html](http://www.coelho.net/php_cve.html) , Accessed 2/15/2014
- J.J. Colao (2012), "With 60 Million Websites, WordPress Rules The Web. So Where's The Money?", Forbes, <http://www.forbes.com/sites/jjcolao/2012/09/05/the-internets-mother-tongue/>, Accessed 10/17/13
- Cunningham (2012), "Model View Controller History", <http://c2.com/cgi/wiki?ModelViewControllerHistory>, Accessed 2/21/14
- EllisLab Inc., "CodeIgniter at a Glance," [http://ellislab.com/codeigniter/user-guide/overview/at\\_a\\_glance.html](http://ellislab.com/codeigniter/user-guide/overview/at_a_glance.html), Ellis Labs, Accessed 11/26/2013
- EllisLab Inc., "Model-View-Controller," EllisLab, inc., <http://ellislab.com/codeigniter/user-guide/overview/mvc.html>, Accessed 2/28/2014
- Encyclopedia Britannica (2013), "SQL," <http://www.britannica.com/EBchecked/topic/569684/SQL>, Accessed 2/28/2014
- R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee (1999), "Hypertext Transfer Protocol -- HTTP/1.1," <http://tools.ietf.org/html/rfc2616>, Accessed 2/28/2014
- A. Goldman, *The Heart of the Matter*, <http://alumni.berkeley.edu/news/california-magazine/winter-2010-inside-out/heart-matter>, Accessed 10/15/13

- S. Hanenburg (2012), *An Experiment About Static and Dynamic Type Systems*, <https://courses.cs.washington.edu/courses/cse590n/10au/hanenberg-oopsla2010.pdf>, Institute for Computer Science and Business Information Systems, Accessed 11/26/2013
- A. Ide (2013), "PHP Just Grows & Grows", Netcraft, <http://news.netcraft.com/archives/2013/01/31/php-just-grows-grows.html>, Accessed 2/15/2014
- S. James (2007), "Web Single Sign-On Systems," <http://www.cs.wustl.edu/~jain/cse571-07/ftp/websso/>, Accessed 2/28/2014
- J. Kim & S. Lee & H. Lee, *Decision Tree Induction Techniques for E-Commerce Recommendation Systems*, <http://isi.cbs.nl/iamamember/CD2/pdf/440.PDF>, Accessed 10/15/13
- L. Kujawa (2013), "Performance benchmark of popular PHP frameworks", <http://systemsarchitect.net/performance-benchmark-of-popular-php-frameworks/>, Systemsarchitect, Accessed 2/22/2014
- LINFO (2006), "Directory Definition," <http://www.linfo.org/directory.html>, Accessed 2/28/2014
- LINFO (2006), "Linux Definition," <http://www.linfo.org/linuxdef.html>, Accessed 2/28/2014
- LINFO (2007), "Root Directory Definition," [http://www.linfo.org/root\\_directory.html](http://www.linfo.org/root_directory.html), Accessed 2/28/2014
- P. Melville & V. Sindhvani, *Recommender Systems*, <http://www.prem-melville.com/publications/recommender-systems-eml2010.pdf>, Accessed 10/14/13
- R. Meteren & M. Someren, *Using Content-Based Filtering for Recommendation*, [http://users.ics.forth.gr/~potamias/mlnia/paper\\_6.pdf](http://users.ics.forth.gr/~potamias/mlnia/paper_6.pdf), Accessed 10/14/13
- Microsoft, "Structured Query Language (SQL)", [http://msdn.microsoft.com/en-gb/library/windows/desktop/ms714670\(v=vs.85\).aspx](http://msdn.microsoft.com/en-gb/library/windows/desktop/ms714670(v=vs.85).aspx), Accessed 10/15/13
- okcupid, "Match Percentages", <http://www.okcupid.com/help/match-percentages>, Accessed 10/15/13
- Oracle, "MySQL Glossary," [http://dev.mysql.com/doc/refman/5.6/en/glossary.html#glos\\_schema](http://dev.mysql.com/doc/refman/5.6/en/glossary.html#glos_schema), Accessed 2/28/2014
- D. Peterson (2008), "Rasmus Lerdorf: PHP Frameworks? Think Again.", Sitepoint, <http://www.sitepoint.com/rasmus-lerdorf-php-frameworks-think-again/>, Accessed 2/16/2014
- E. Rescorla (2000), "HTTP Over TLS," <https://tools.ietf.org/html/rfc2818>, accessed 3/1/2014

Wikipedia contributors (2014), "Codebase," Wikipedia, the Free Encyclopedia, <http://en.wikipedia.org/w/index.php?title=Codebase&oldid=585720865>, Accessed 3/7/2014

Wikipedia contributors (2014), "Database," Wikipedia, The Free Encyclopedia, <http://en.wikipedia.org/w/index.php?title=Database&oldid=598462722>, Accessed 3/7/2014

Wikipedia contributors (2014), "Database normalization," Wikipedia, The Free Encyclopedia, [http://en.wikipedia.org/w/index.php?title=Database\\_normalization&oldid=595742341](http://en.wikipedia.org/w/index.php?title=Database_normalization&oldid=595742341), Accessed 3/7/2014

Wikipedia contributors (2014), "High-level programming language," Wikipedia, the Free Encyclopedia, [http://en.wikipedia.org/w/index.php?title=High-level\\_programming\\_language&oldid=5980366866](http://en.wikipedia.org/w/index.php?title=High-level_programming_language&oldid=5980366866), Accessed 3/7/2014

Wikipedia contributors (2014), "Server (computing)," Wikipedia, the Free Encyclopedia, [http://en.wikipedia.org/w/index.php?title=Server\\_\(computing\)&oldid=598383715](http://en.wikipedia.org/w/index.php?title=Server_(computing)&oldid=598383715), Accessed 3/7/2014

WordPress, "Database Description", [http://codex.wordpress.org/Database\\_Description](http://codex.wordpress.org/Database_Description), Accessed 10/17/13

WPI, "Major Qualifying Project," <http://www.wpi.edu/academics/ugradstudies/mqp.html>, Accessed 2/28/2014

WPI, "Project Program", <http://www.wpi.edu/academics/Projects/available.html>, Accessed 10/15/13

WPI Gordon Library, "IQP and MQP Reports", <https://www.wpi.edu/academics/library/find/iqp-mqp.html>, Accessed 10/6/13

W3C, "HTML & CSS," <http://www.w3.org/standards/webdesign/htmlcss>, Accessed 2/28/2014

W3Schools, "HTML Introduction", [http://www.w3schools.com/html/html\\_intro.asp](http://www.w3schools.com/html/html_intro.asp), Accessed 10/15/13

## Appendix A: Initial Survey Questions

- What is your Major?
- Are you going away for MQP?
- What resources did you use to find your MQP?
- Did you find you Project, Group, or Advisor first?
- Did someone refer you to your group, project, advisor & who?
- Who came up with the Project? (You, Advisor, Group/member, Company)
- How many people are in your MQP group?
- How did you find them & How did they find you?
- Are you working with any different majors/what are their majors?
- Was your partner(s) skill set important information to you?
  
- Rank how important it was to know your partner(s) skill sets before joining group
  - 0 - didn't matter --- 10 - super important
  
- What resources did you use to find your advisor?
- Did you know your advisor before the project?
- Did your other group members (# & how many) know the advisor before the project?
- Why/based on what criteria did you choose your advisor?
  
- Was your advisor's subject interest important to you (rank)
  - 0 - didn't matter --- 10 - super important
  
- Did you attend the CS MQP projects day last year by the advisors?
  
- Rate the usefulness of that day with regards to finding an MQP
  - 0 – not useful --- 10 – super-useful
  
- How would you rate the MQP-finding experience
  - 0 - no effort/ easy --- 10 - difficult/almost didn't find an MQP

- Name 3 reasons you chose to go away for MQP/Name 3 reasons you chose to stay on campus for MQP
- What would have made the process easier? (easy to find a project, knowing group, better skill sets in group, etc.)

## Appendix B: Initial Survey Responses

- **What is your Major**  
User 1: CS  
User 2: IMGD-tech  
User 3: RBE/CS  
User 4: CS  
User 5: CS  
User 6: CS
- **What resources did you use to find your MQP**  
User 1: mostly emails from faculty saying where centers were  
User 2: outlook email, personal networking (knowing people on campus)  
User 3: looked on website a little, but wasn't much info, mostly just emailed out to professors looking for projects (didn't have any ideas) RBE project website is really out of date (when to 3 different professors, looking for kinds of project they would have)  
User 4: talked to a professor (only one), went to prof. proj. presentation day  
User 5: went to global fair, MQP in Budapest  
User 6: currently in the process of looking for one (no advisor, no project, potential partner), have been looking on prof websites, decided I would need to come up with own projects, want to find something related to programming language theory (no profs around), freaking out about grad school admissions, talking to profs
- **Did you find you Project, Group, or Advisor first**  
User 1: advisor  
User 2: group  
User 3: advisor  
User 4: advisor  
User 5: advisor (apply MQP process)  
User 6: partner
- **Did someone refer you to your group, project, advisor & who**  
User 1: No, the advisor sent an email to all the majors (Finkel)  
User 2: Yes, [student: name omitted], first partner helped [student: name omitted] find others  
(group as a whole decided on proj and advisors)  
User 3: A few different people told me I should talk to my advisor (wanted to focus more on RBE + software), just friends/other professors (academic advisor)  
User 4: both talked to same professor, set us up, knew we wanted to work together, gave us some project options  
User 5: previous academic advisor [advisor: name omitted]  
User 6: knew him from outside CS, don't remember, just talking about MQPS and he didn't have one, a friend
- **Who came up with the Project? (You, Advisor, Group/member, Company/Sponsor)**  
User 1: Don't have a project yet, probably by the company, typical with project center



User 2: a group member (pitched idea), were other ideas as well  
User 3: [advisors: names omitted] Advisor had an idea for a project, wasn't concrete until A Term (was away D Term on IQP) – robot learns to play wack-a-mole  
User 4: Our advisor had the basics of what we wanted (continued past MQP), picked a lot of details ourselves w/ general ideas, formed MQP to our interests  
User 5: the Budapest people  
User 6: don't know how to find an MQP-sized problem in this field, posted on forum looking for project ideas, going to email [advisor: name omitted] (lambda-the-ultimate.org) community of programming language researchers

- **How many people are in your MQP group**

User 1: (not sure) 3?  
User 2: 5 (including)  
User 3: 2 (including) (other is RBE/IMGD)  
User 4: 2 (including)  
User 5: 2 (including) – didn't like that person  
User 6: probably 2, maybe 3, advised against one by self

- **How did you find them & How did they find you**

User 1: they were also going away  
User 2: I know a guy who knows a guy...some people who were just looking for group that were referred, accidentally added to someones mqp email, [name omitted] offered her a spot, just people knowing other people in the program  
User 3: they also went to the same advisor separately [name omitted] advisor asked if you knew him, met up  
User 4: found each other through professor, knew each other from soft eng, but didn't think of working together until professor suggested it  
User 5: you apply, the only two who went (only 4 spaces, usually more people go)  
User 6: friend, know each other, both looking for MQP, his previous proposed MQP fell through

- **Are you working with any different majors/what are their majors**

User 1: (not sure) ECE majors are going, don't know if in group  
User 2: IMGD-CS double majors (some art some tech)  
User 3: RBE/IMGD double major (tech)  
User 4: just CS  
User 5: CS  
User 6: CS

- **Was your partner(s) skill set important information to you?**

User 1: Not in the decision, because going away, didn't pick group, would be reassuring (would have requested one who is sure of skills to be in group)  
User 2: Yes, because most IMGD projects need a mix of tech and art (wanted it to be even)  
User 3: it's important in our process of doing the project, felt a little nervous because signed up on the project w/ advisor w/o a team or idea. you didn't know who you'd be

working with, felt a little scary. (thought might be a good partner b/c like professor c.) had had some classes with him before

User 4: Yes, but not extremely so. Knew already that he was more organized & had strengths I didn't have; good to have balance in a group (didn't need 2 people coding constantly)

User 5: of course

User 6: sort of, been in some classes before, had a feeling he's pretty good, area of specialization isn't the same as mine, both looking for something, so he would do in my area. would have liked to work with someone else in PL research, those people don't exist!

- **Rank how important it was to know your partner(s) skill sets before joining group**

0 - didn't matter --- 10 - super important

User 1: 8 (important in planning/doesn't want to have to do all the work)

User 2: 8

User 3: 2.5

User 4: 6

User 5: 10 think it's really important

User 6: fairly good programmer, 3

- **What resources did you use to find your advisor**

User 1: email, IQP advisor is the ECE advisor going, helped provide info [advisor: name omitted]

User 2: I didn't actually find our advisor, discussed as a group who would be "top-picks", sent emails to those, whoever responded first was your advisor. 3/4 (1 no, too many other groups, 2 yes)

User 3: emailed them & word of mouth told him who/who to email (skyped from Australia)

User 4: had classes with him (knew he would be a good professor to work with), just went to him

User 5: only one advisor (one WPI, one Budapest)

User 6: aren't a whole lot of professors maybe [professors names omitted]

- **Did you know your advisor before the project**

User 1: Did not, just knew name

User 2: Knew of him, hadn't had any classes with either beforehand [advisor: name omitted]

User 3: Not really, no (didn't have a class with her before)

User 4: Yes, took multiple classes (knew each other fairly well)

User 5: No

User 6: yes, taken courses with both [professors names omitted]

- **Did your other group members (# & how many) know the advisor before the project**

User 1: maybe?

User 2: both art students knew the art advisor beforehand, and one tech knew tech advisor

beforehand

User 3: I'm not sure, but he had a summer internship with her, got to know her better

User 4: Yes, both met him in the same class

User 5: No [advisor: name omitted]

User 6: not sure

- **Why/based on what criteria did you choose your advisor**

User 1: he was the only prof going to silicon valley, heard good things, if it was a professor I'd heard of who was bad I wouldn't have good

User 2: chose it on the criteria that he teaches a lot of CS courses. knew that he had technical experience, has done MQPs before, knew what he was doing, Art kids really liked art guy

User 3: her project ideas & research areas seemed interesting

User 4: Knew he was easy to work with, got along well, knowledge in areas wanted to do project in, knew he had a list of projects (from presentation) that I would be interested in.

User 5: didn't get to choose, chose project center oversea research experience

User 6: that they would advise the kind of project that I wanted to have, deciding the project, then building the project around that (why didn't want to go off campus bc forced to work on that)

- **Was your advisor's subject interest important to you, specific area of cs (rank)**

0 - didn't matter --- 10 - super important

User 1: nice to know, wouldn't have affected decision, 7

User 2: 6.5 (more important that knew how to run an MQP) !# of mqps done!!

User 3: 7 (was important factor)

User 4: 8 (yes!)

User 5: Yes 10

User 6: 7 ultimate criteria was willingness

- **Did you attend the CS MQP projects day last year by the advisors**

User 1: No (would have gone but couldn't)

User 2: no (got email & deleted)

User 3: No (don't even know when that was) Knew that most of the CS projects wouldn't apply to RBE, really needed an RBE project (no idea when the RBE's is, RBE dept sucks at doing things on time/organizing & planning)

User 4: Yes (when I was junior)

AL : No

User 6: No, scheduling conflict

- **Rate the usefulness of that day with regards to finding an MQP**

0 – not useful --- 10 – super-useful

\*IMGD HAVE to come up with own project pitch to professor

User 4: 5

- **How would you rate the MQP-finding experience**

0 - no effort/ easy --- 10 - difficult/almost didn't find an MQP

User 1: 3 (thinks MQP are easy to find/come up with, a lot of opportunities)

User 2: 5 (once had a foot in the door, it was easier to accrue ppl, getting the process started was the hardest part, finding the initial person/group)

User 3: 6 (hate the process for finding things, hard to make decisions, wasn't necessarily hard to find MQPs to begin with) would be easier to make a decision if more upfront info on the options! (like numbers, they mean something! statistics)

User 4: 6

User 5: 0 (really easy, wasn't hard to find)

User 6: 9 this has been pretty hard! doing the way has been hard, hard to find who would work on it

- **Name 3 reasons you chose to go away for MQP**

- **/Name 3 reasons you chose to stay on campus for MQP**

User 1: 1.) cost/subsidized housing 2.) location, being able to go and explore different part of the country 3.) networking opportunities (going into the CS mecca out there)

User 2: 1.) Money 2.) didn't know Japanese, might of wanted to go to Silicon 3.)

Girlfriend 4.) courses need to graduate (only offered one per term)

User 3: 1.) Had already gone away for IQP/cost 2.) Wanted to have the time to spend w/ friends after being gone, wanted 4 terms 3.) Cost

User 4: 1.) already went away for IQP 2.) knew there were really cool projects on campus (word of mouth) you can do CS anywhere & it's still the same 3.) didn't want to lose whole term of classes/do ID2050 again

User 5: 1.) want new perspective from oversea experience 2.) graduate school at end of year, wanted to have MQP on application for apply 3.) wanted to travel

TB 1.) wanted to choose what I would get to work on (nothing else mattered)

- **What would have made the process easier**

- easy to find
- know them
- skill sets

User 1: When I was attempting to pitch my own MQP, I wished there was an MQP-checklist form (this is my advisor, this is what I'm doing, abstract, partners, have someone sign it)

User 2: A central place where everyone in the major could go "looking for group", flurry of email is ridiculous, forum, email alias to send to find others who aren't in a group yet. Electronically would be easier than important, meeting night might work, but be harder for people to go and have to worry

User 3: Would have made it better would be a unified place (like a website or something) WPI's/RBEs websites suck, some way to find unified list of all projects, ideas that advisors have, ideas that students have or just looking for a group (instead of spamming undergrads at cs.wpi.edu!) a big list! something that everyone would actually use it would be the key thing (especially advisors) is a page for RBE professors to put their

MQPs, but don't usually post it, and rarely post it early enough

User 4: it would have been nice if after the presentation there was a centralized list of all projects, because no way to see them after that day (luckily my partner wrote them all down) – I didn't have time to write them down. (Centralized listing overall). Couldn't "see" what projects, always had to talk to someone. I got lucky with a partner. Would have been nice to see who was available/looking for MQPs (students) : MQP seems like this ambiguous thing that you just have to DO. would be easy if you had a list to look at and compare.

User 5: project pitch day, good idea, should market it more, should be on the website (didn't know about it).; would have been better to pick a partner, picking a partner can be really hard. sometimes you know the ppl, but don't always know what they're good at. better to pick a working partner, not just a friend

User 6: some sort of resource how to find a project in your sub-field, having Professor [name omitted] on campus

#### COMMENTS:

-can't make money off of what you do –Professor [name omitted] going away, deterred from choosing own MQP –difficulty worried about finishing –worried about finding people to do it with (wasn't sure if people would want to do, foundations area/turing machine simulator, type of state machine) wanted to give it to TAs, feels like would have had trouble finding partners & didn't want to work alone, search by types of interests, have to had taken the certain classes to want to work with! (foundations!), partner is what makes/breaks MQP experience, they should at least like what they're doing so they'll do work

-overwhelming thing I've heard, (I'm going into academia) HOW DO YOU GET STARTED how do you start doing research – find a mentor (what I've been told), not a lot of resources for finding, networking researchers don't have problems finding mentors.

searching online for advise, friend in academia

## Appendix C: Database Designs

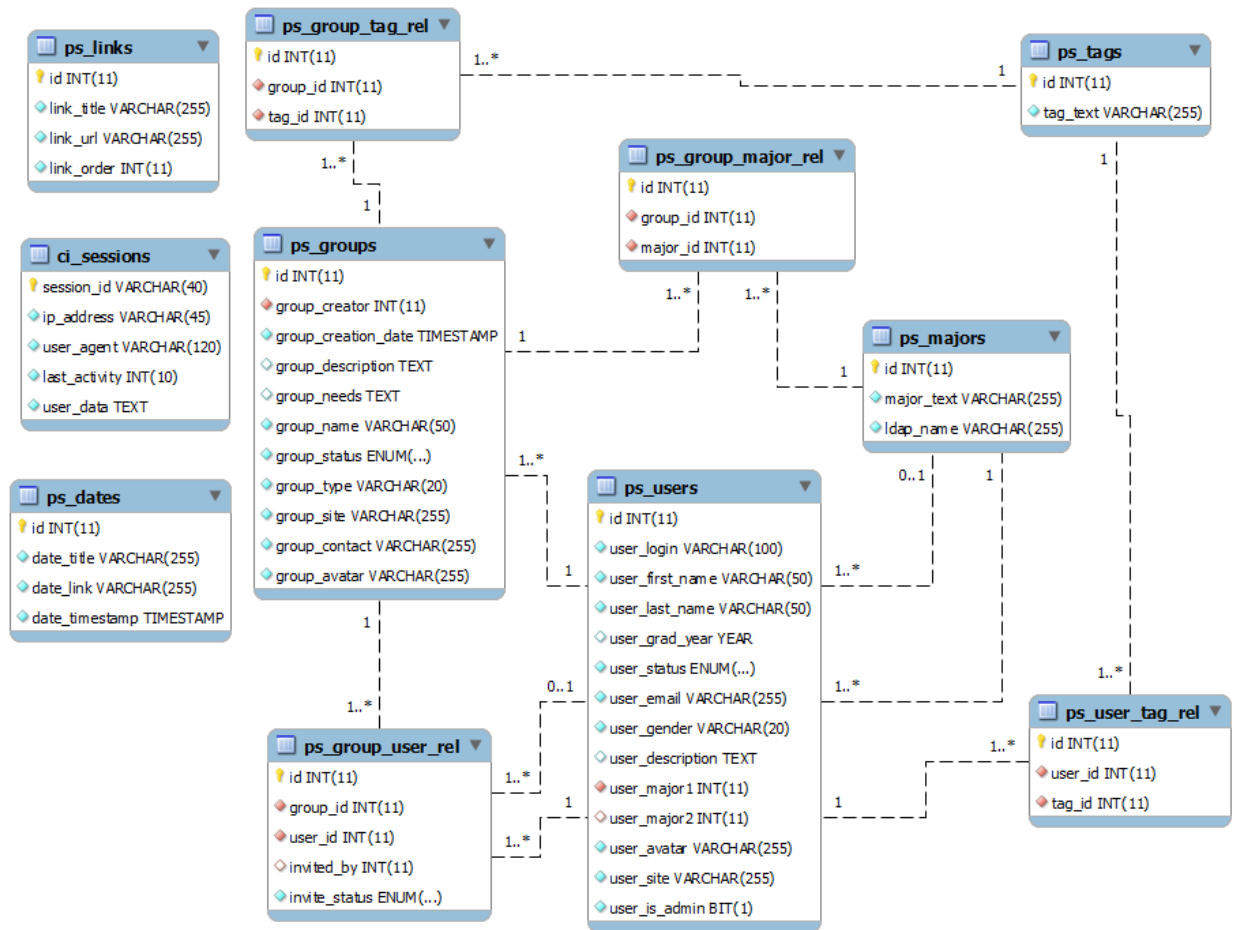


Figure 14: ProjectSpot's Database Structure

This diagram represents the final design of our database.

Each large box represents a table in the database.

Every value inside the box represents a field in that database.

The lines connecting the boxes signify a relation between those tables.

For example, ps\_users has a relation between itself and the ps\_user\_tag\_rel table, which has a relation between itself and the ps\_tags table.

# Appendix D: UML Sequence Diagrams

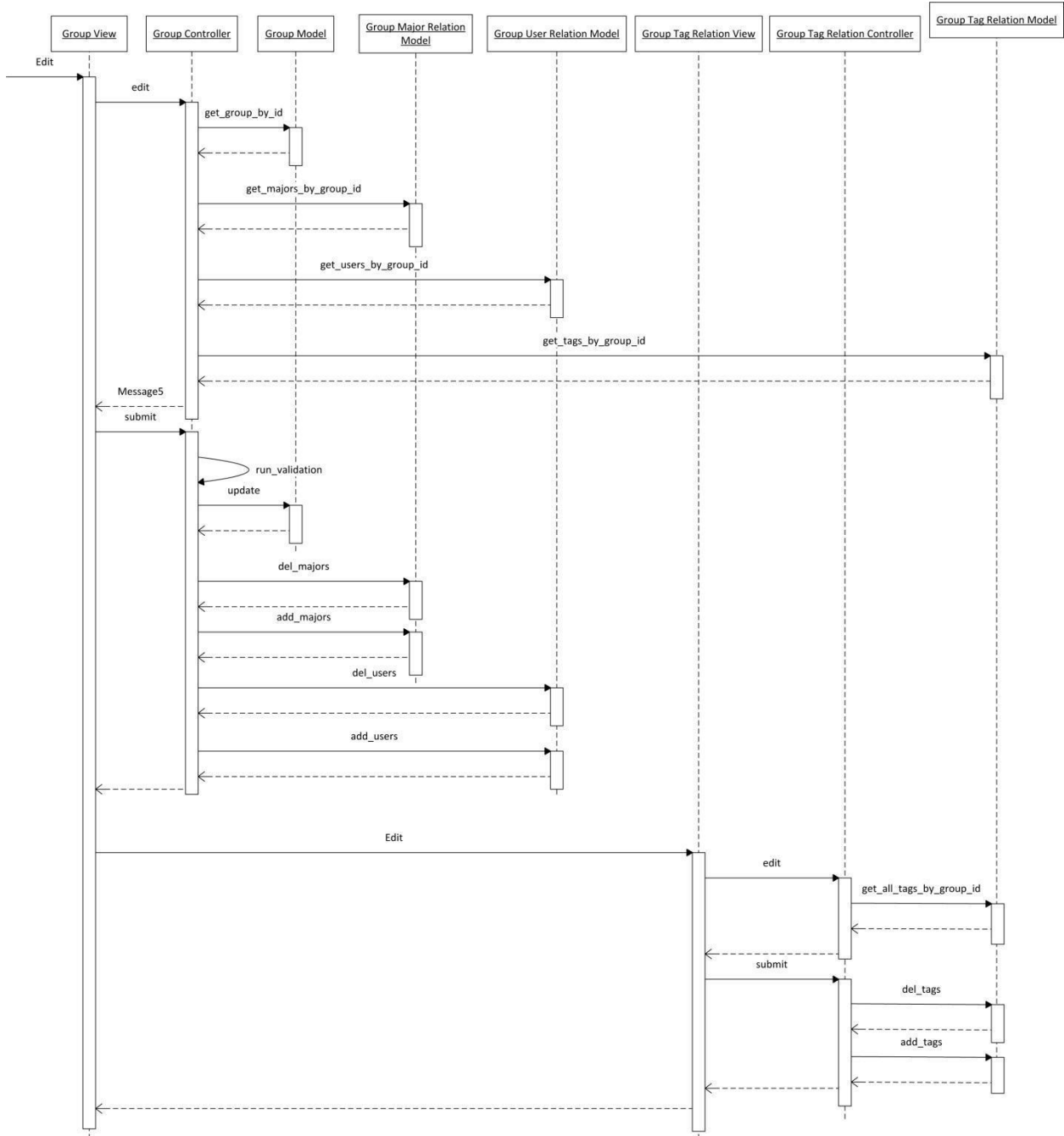


Figure 15: UML Diagram for Editing a Group

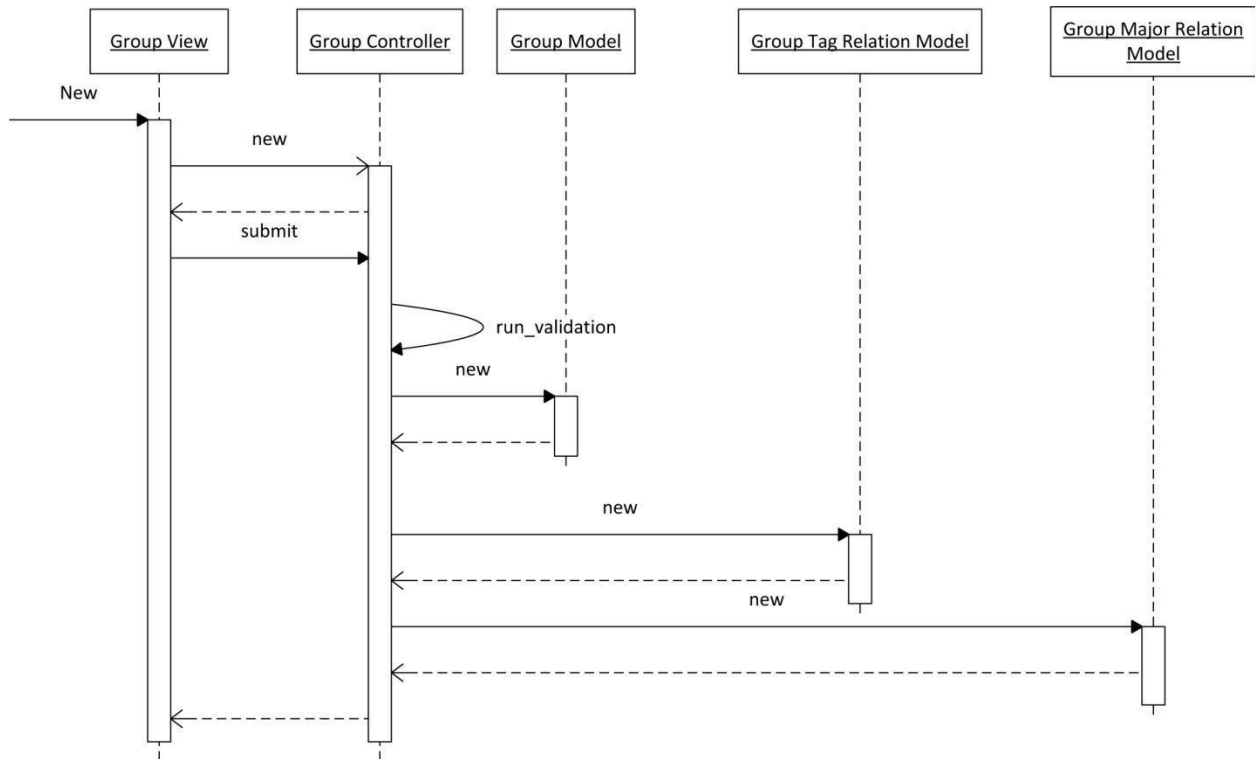


Figure 16: UML Diagram for Creating New Group

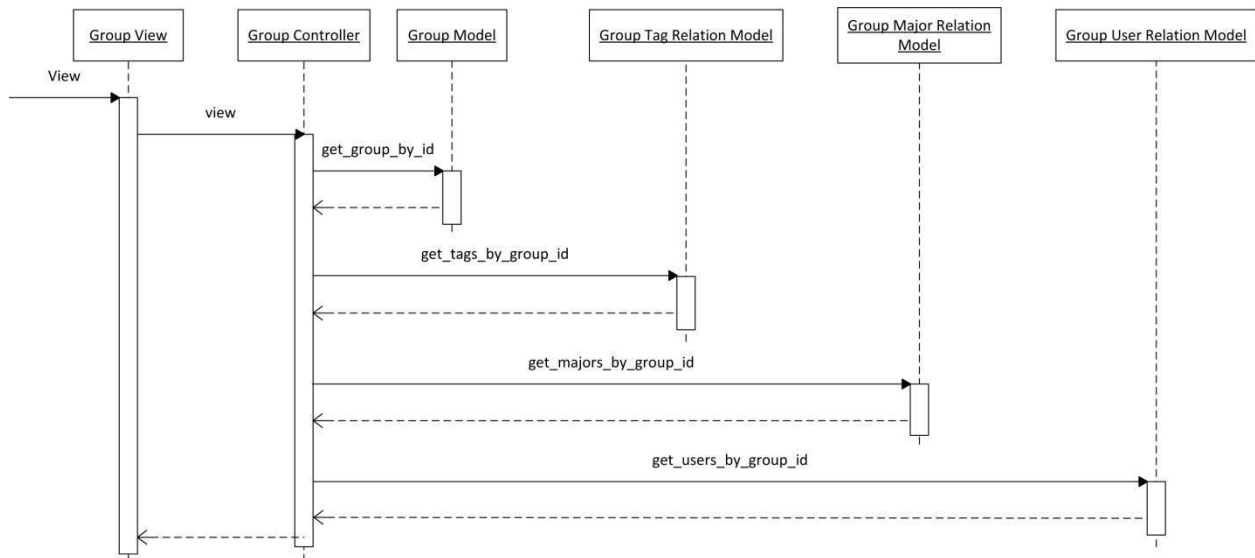


Figure 17: UML Diagram for Viewing a Group



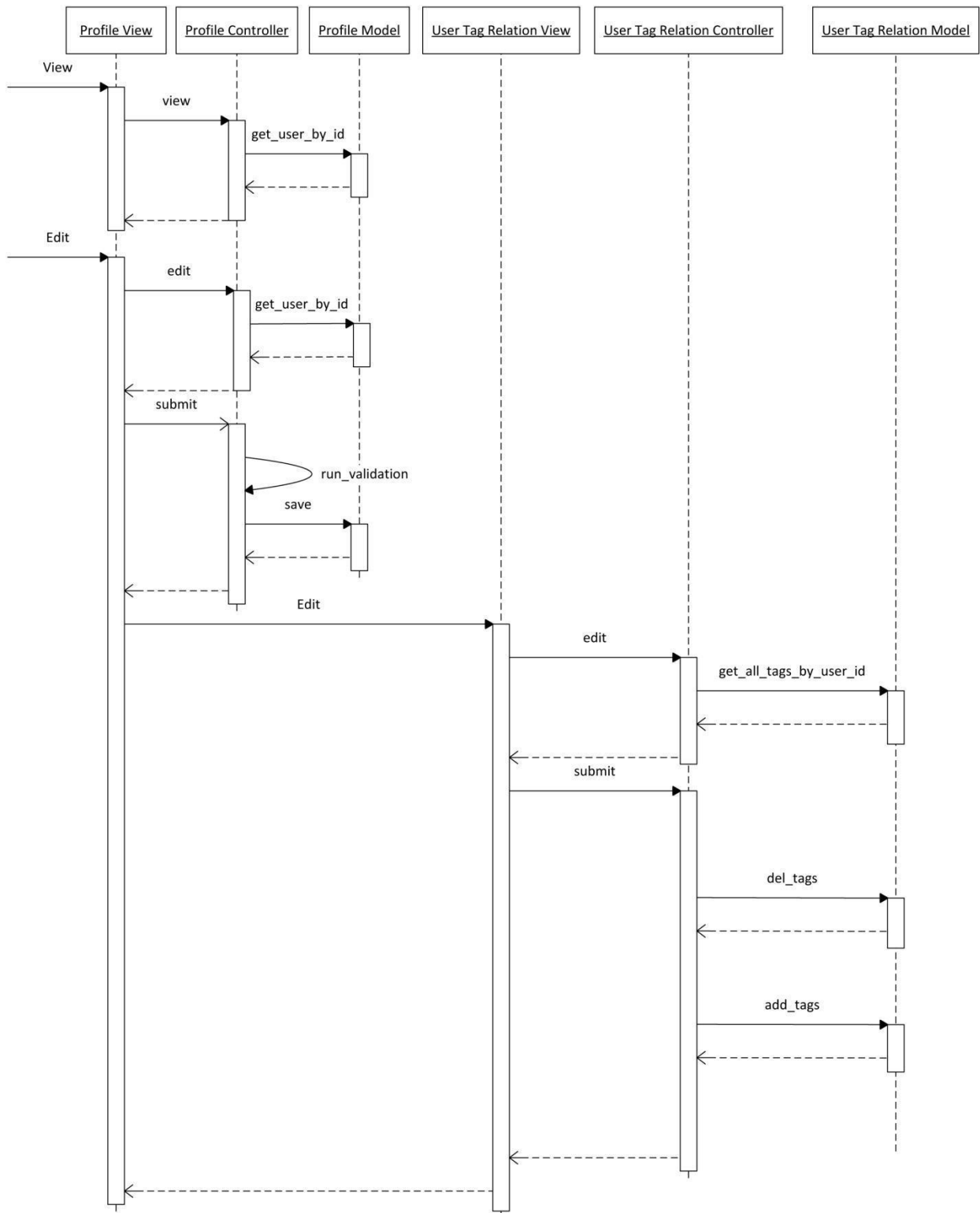


Figure 18: UML Diagram for Editing and Viewing a Profile

## Appendix E: User Study on Tasks in Interface Questionnaire

[This was a printout for each user to fill-in by hand. Some of the formatting here was changed]

### Questionnaire

User: \_\_\_\_\_

Case: \_\_\_\_\_

You may look at the interface while you answer these questions.

Rank the following parts of the site (in your opinion) for layout/organization, use of color, ease of understanding, and usefulness to your objective on the following scale:

1 – cluttered/confusing layout, disliked/confusing color, couldn't understand, not useful

2 – a bit better, but overall was poor

3 – no opinion, indifferent, was OK

4 – good overall

5 – well-spaced/understandable layout, good use of color, very understandable, useful to user

### Dashboard

<b>Layout/Organization:</b>	1	2	3	4	5
<b>Use of Color:</b>	1	2	3	4	5
<b>Ease of Understanding:</b>	1	2	3	4	5
<b>Usefulness to Objective:</b>	1	2	3	4	5

### Profile

<b>Layout/Organization:</b>	1	2	3	4	5
<b>Use of Color:</b>	1	2	3	4	5
<b>Ease of Understanding:</b>	1	2	3	4	5
<b>Usefulness to Objective:</b>	1	2	3	4	5

### Group

<b>Layout/Organization:</b>	1	2	3	4	5
<b>Use of Color:</b>	1	2	3	4	5
<b>Ease of Understanding:</b>	1	2	3	4	5
<b>Usefulness to Objective:</b>	1	2	3	4	5

**Find**

<b>Layout/Organization:</b>	1	2	3	4	5
<b>Use of Color:</b>	1	2	3	4	5
<b>Ease of Understanding:</b>	1	2	3	4	5
<b>Usefulness to Objective:</b>	1	2	3	4	5

**Banner**

<b>Layout/Organization:</b>	1	2	3	4
5				
<b>Use of Color:</b>	1	2	3	4
5				
<b>Ease of Understanding:</b>	1	2	3	4
5				
<b>Usefulness to finding MQP:</b>	1	2	3	4
5				

**Impression of the Website overall**

<b>Layout/Organization:</b>	1	2	3	4
5				
<b>Use of Color:</b>	1	2	3	4
5				
<b>Ease of Understanding:</b>	1	2	3	4
5				
<b>Usefulness to finding MQP:</b>	1	2	3	4
5				

In terms of the Banner;

**✚ How do you feel about the number of items on the banner?**

- no opinion
- it's fine
- too sparse
- too crowded
- it's perfect

In terms of the "Find" page;

**How time consuming was the process to find what you needed?**

took too long		average time spent		was quick to find
1	2	3	4	5

**Did filtering work the way you expected?**

no, was confusing		not sure		exactly as expected
1	2	3	4	5

**Was the layout of the "Find" page useful to you?**

needs a better layout		not sure		great layout for use
1	2	3	4	5

**Did viewing other profiles (user and group) from the Find page make sense?**

no, was confusing		not sure		made perfect sense
1	2	3	4	5

**Was viewing the interests of other users useful to your objective?**

no, wasn't useful		indifferent		was the most useful
1	2	3	4	5

In terms of the "Edit Profile" page;

**How time consuming was the process to find what you needed?**

took too long		average time spent		was quick to find
1	2	3	4	5

**Was the layout of the "Edit Profile" page useful to you?**

needs a better layout		not sure		great layout for use
1	2	3	4	5

**Did uploading a picture work the way you expected?**

no, was confusing		not sure		exactly as expected
1	2	3	4	5

**Did adding interests make sense?**

no, was confusing		not sure		made perfect sense
1	2	3	4	5

**Did you find it useful to have interests listed as part of your profile?**

no, wasn't useful		indifferent		was very useful
1	2	3	4	5

General Notes/Comments:

## Open-Ended Questions

User: \_\_\_\_\_

Case: \_\_\_\_\_

✚ Did anything overwhelm you?

✚ What was straightforward?

✚ What was not straightforward?

✚ What did you like?

✚ What could have been better?

## Appendix F: Results from User Study of Tasks through the System

User Study Results - Based on Case 1 & Case 2 Users										
Madalyn Coryea - ProjectSpot										
**For these questions, users were asked to "rank" different aspects of the system on a 1-5 scale. With (for each category) 1 being the most negative option, and 5 being the most positive.										
Category	User A	User B	User C	Case 1 Average	User D	User E	User F	Case 2 Average	Total Average	
<b>Dashboard</b>	Layout/Organization	4	5	5	4.667	3	4	5	4	4.333
	Use of Color	5	5	5	5	5	3	3	3.667	4.333
	Ease of Understanding	5	5	5	5	4	4	5	4.333	4.667
	Usefulness to Objective	4	5	5	4.667	4	5	5	4.667	4.667
<b>Profile</b>	Layout/Organization	4	5	5	4.667	5	4	5	4.667	4.667
	Use of Color	4	4	5	4.333	5	3	4	4	4.167
	Ease of Understanding	5	5	5	5	5	5	5	5	5
	Usefulness to Objective	5	5	5	5	5	4	5	4.667	4.833
<b>Group</b>	Layout/Organization	3	5	5	4.333	5	4	5	4.667	4.5
	Use of Color	4	4	5	4.333	5	3	4	4	4.167
	Ease of Understanding	3	5	5	4.333	4	3	5	4	4.167
	Usefulness to Objective	5	5	5	5	5	4	5	4.667	4.833
<b>Find</b>	Layout/Organization	4	5	5	4.667	4	5	5	4.667	4.667
	Use of Color	4	5	5	4.667	5	4	5	4.667	4.667
	Ease of Understanding	5	4	5	4.667	4	5	5	4.667	4.667
	Usefulness to Objective	5	5	5	5	5	5	5	5	5
<b>Banner</b>	Layout/Organization	5	5	5	5	4	5	5	4.667	4.833
	Use of Color	5	5	5	5	5	5	5	5	5
	Ease of Understanding	5	5	5	5	4	5	5	4.667	4.833
	Usefulness to Objective	5	5	5	5	5	5	5	5	5
<b>Website Overall</b>	Layout/Organization	4	5	5	4.667	4	4	5	4.333	4.5
	Use of Color	4	4	5	4.333	5	3	4	4	4.167
	Ease of Understanding	5	5	5	5	3	4	5	4	4.5
	Usefulness to Objective	5	5	5	5	5	4	5	4.667	4.833





**Notes:**

Most users did not like the gray buttons. Felt they looked "disabled"

Many users in Case 2 were confused about adding their group member to the group. Felt the "Edit Group" page needed an option for adding members

Users overall did not like the way that uploading a photo worked. The word "avatar" was confusing, and there were 2 buttons to click.

Users generally liked the banner

Most users did not like the LOOK of the Dashboard, but almost all found it very useful. (In studying users they all used links on the Dashboard before anything else)

Users in Case 1 overall didn't have negative comments about the system. Users in Case 2 had more negative comments and a harder time navigating the system

All 6 users were able to complete their respective tasks.

Feedback on the system was positive overall

One user stated that "Invites" should be called "Notifications" instead

Limits on the Photo upload should be clearer

## Appendix G: Group User Study Questionnaire

[This was a printout for each user to fill-in by hand. Some of the formatting here was changed]

### Questionnaire

User: \_\_\_\_\_

Case: \_\_\_\_\_

You may look at the interface while you answer these questions.

#### How difficult was it for you and your partners to create a group for your project?

impossible		neither easy nor difficult		extremely easy
1	2	3	4	5

#### Did the process of creating a group make sense?

made no sense		made some sense		made perfect sense
1	2	3	4	5

#### How difficult was it to get all partners listed as members of the group?

impossible		neither easy nor difficult		extremely easy
1	2	3	4	5

#### Did the process of adding members to the group make sense?

made no sense		made some sense		made perfect sense
1	2	3	4	5

#### How difficult was it to use the Invitations page?

impossible		neither easy nor difficult		extremely easy
1	2	3	4	5

#### Did the process of using the Invitations page make sense?

made no sense		made some sense		made perfect sense
1	2	3	4	5

#### Did you enjoy using the system?

disliked using it		using it was ok		loved using it
1	2	3	4	5

#### If you needed an MQP, how likely would you be to use the system for real to find a group?

unlikely		somewhat likely		very likely
1	2	3	4	5

#### If you were in an MQP, how likely would you be to use the system for real to find project partners?

unlikely		somewhat likely		very likely
1	2	3	4	5

**How likely would you be to use the system for real to find an advisor for a project?**

unlikely		somewhat likely		very likely
1	2	3	4	5

In terms of the “Find” page;

**How time consuming was the process to find what you needed?**

took too long		average time spent		was quick to find
1	2	3	4	5

**Did filtering work the way you expected?**

no, was confusing		not sure		exactly as expected
1	2	3	4	5

**Was the layout of the “Find” page useful to you?**

needs a better layout		not sure		great layout for use
1	2	3	4	5

**Did viewing other profiles (user and group) from the Find page make sense?**

no, was confusing		not sure		made perfect sense
1	2	3	4	5

In terms of the “Edit Group” page;

**How time consuming was the process to find what you needed?**

took too long		average time spent		was quick to find
1	2	3	4	5

**Was the layout of the “Edit Group” page useful to you?**

needs a better layout		not sure		great layout for use
1	2	3	4	5

**Did adding interests make sense?**

no, was confusing		not sure		made perfect sense
1	2	3	4	5

**Do you find it useful to have interests listed as part of your group profile?**

no, wasn't useful		indifferent		was very useful
1	2	3	4	5

In terms of the “Edit Profile” page;

**How time consuming was the process to find what you needed?**

took too long		average time spent		was quick to find
1	2	3	4	5

**Was the layout of the “Edit Profile” page useful to you?**

needs a better layout		not sure		great layout for use
1	2	3	4	5

**Did adding interests make sense?**

no, was confusing		not sure		made perfect sense
1	2	3	4	5

**Do you find it useful to have interests listed as part of your profile?**

no, wasn't useful		indifferent		was very useful
1	2	3	4	5

## Open-Ended Questions

User: \_\_\_\_\_

Case: \_\_\_\_\_

✚ Did anything overwhelm you?

✚ What was straightforward?

✚ What was not straightforward?

✚ What did you like?

✚ What could have been better?

## Appendix H: Results from Group User Study

Question	Group 1					Group 2				
	User A	User B	User C	Average	User D	User E	User F	Average	Overall Average	
How difficult was it for you and your partner to create a group for your project?	2	4	5	3.67	5	5	5	5.00	4.33	
Did the process of creating a group make sense?	3	4	5	4.00	5	4	5	4.67	4.33	
How difficult was it to get all partners listed as members of the group?	5	4	4	4.33	5	4	5	4.67	4.50	
Did the process of adding members to the group make sense	5	4	4	4.33	4	4	3	3.67	4.00	
How difficult was it to use the Invitations page?	3	4	3	3.33	5	4	5	4.67	4.00	
Did the process of using the Invitations page make sense?	4	4	2	3.33	5	3	4	4.00	3.67	
Did you enjoy using the system?	5	5	5	5.00	5	3	5	4.33	4.67	
If you needed an MQP, how likely would you be to use the system for real to find a group?	4	4	4	4.00	5	4	5	4.67	4.33	
If you were in an MQP, how likely would you be to use the system for real to find project partners?	3	5	5	4.33	4	3	5	4.00	4.17	
How likely would you be to use the system for real to find an advisor for a project?	4	5	5	4.67	4	4	5	4.33	4.50	
<b>In terms of the "Find" page:</b>										
How time consuming was the process to find what you needed?	4	5	4	4.33	5	4	5	4.67	4.50	
Did filtering work the way you expected?	4	3	5	4.00	5	4	3	4.00	4.00	
Was the layout of the "Find" page useful to you?	3	3	4	3.33	5	5	1	3.67	3.50	
Did viewing other profiles (user and group) from the Find page make sense?	4	3	4	3.67	5	5	5	5.00	4.33	

<b>In terms of the "Edit Group" page;</b>										
	3	4	5	4.00	5	5	5	5	5.00	4.50
Was the layout of the "Edit Group" page useful to you?	4	3	5	4.00	5	3	3	3	3.67	3.83
Did adding interests make sense?	4	3	5	4.00	5	4	5	4	4.67	4.33
Do you find it useful to have interests listed as part of your group profile?	4	3	3	3.33	5	4	5	4	4.67	4.00
<b>In terms of the "Edit Profile" page?</b>										
How time consuming was the process to find what you needed?	3	5	5	4.33	5	4	5	4	4.67	4.50
Was the layout of the "Edit Profile" page useful to you?	3	4	5	4.00	5	4	5	4	4.67	4.33
Did adding interests make sense?	1	5	4	3.33	5	5	5	5	5.00	4.17
Do you find it useful to have interests as part of your profile?	4	3	4	3.67	4	5	5	5	4.67	4.17

## **Appendix I: Available Interests for Users and Groups**

Algorithms

Networks

Network Security

Operating Systems

Programming Languages

Artificial Intelligence

Java

Javascript

pHp

Python

Ruby

C/C++

Web Development

Human-Computer Interaction

Software Security

Computation

Databases

Theoretical Computer Science

Applied Computer Science

Graphics

Visualization

Architecture

Software Engineering

Parallel Systems

Concurrent Systems

Distributed Systems

Code Theory

Data Structures

Formal Methods

Cryptography



Compilers

Pattern Recognition

Machine Learning

Evolutionary Computation

Natural Language Processing

Data Mining

Image Processing

Information Security

Information Retrieval

## Appendix J: Screen Shots of Admin Pages

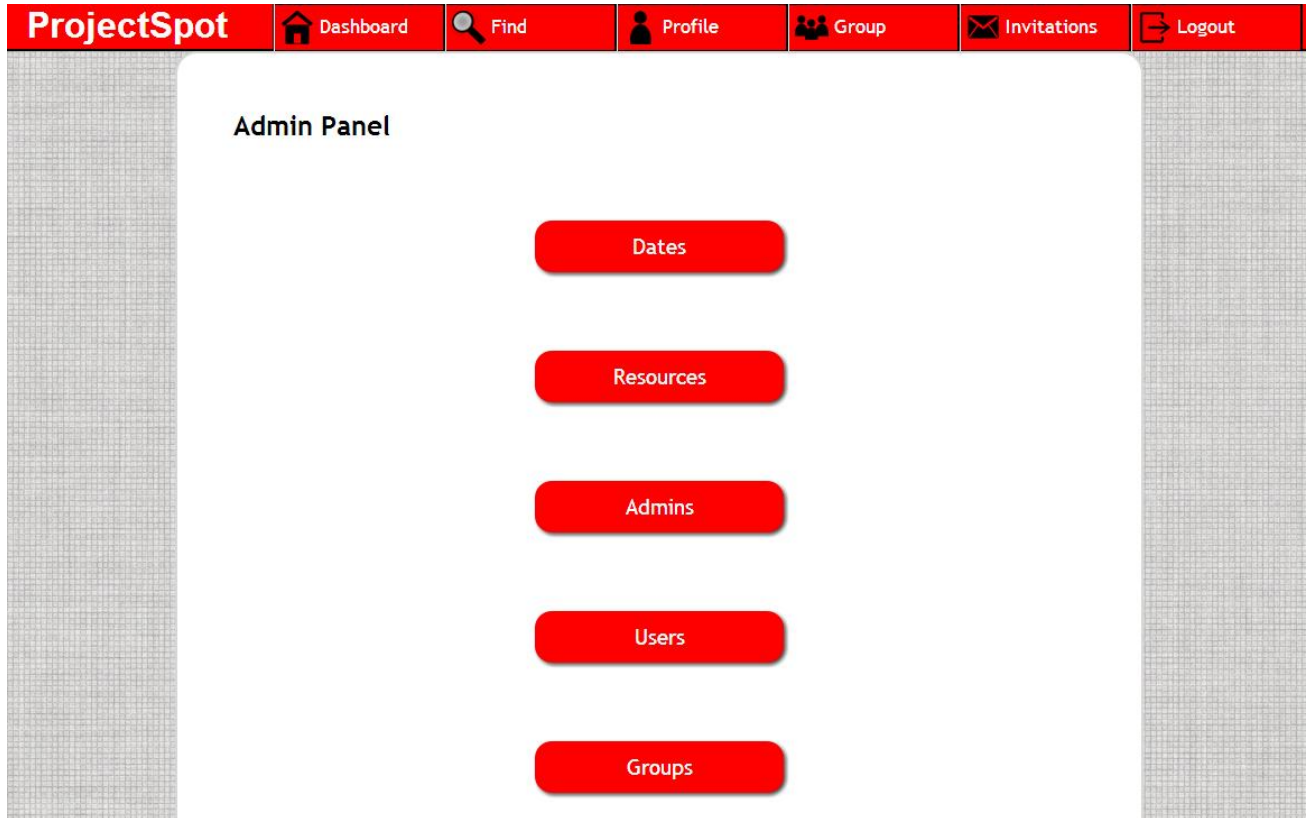


Figure 19: Admin Panel Main Page

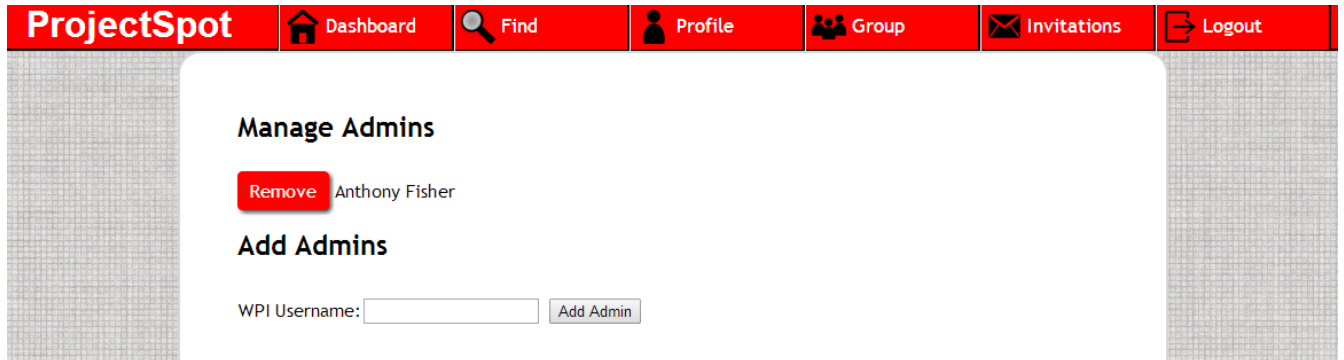


Figure 20: Page for Adding and Removing System Administrators

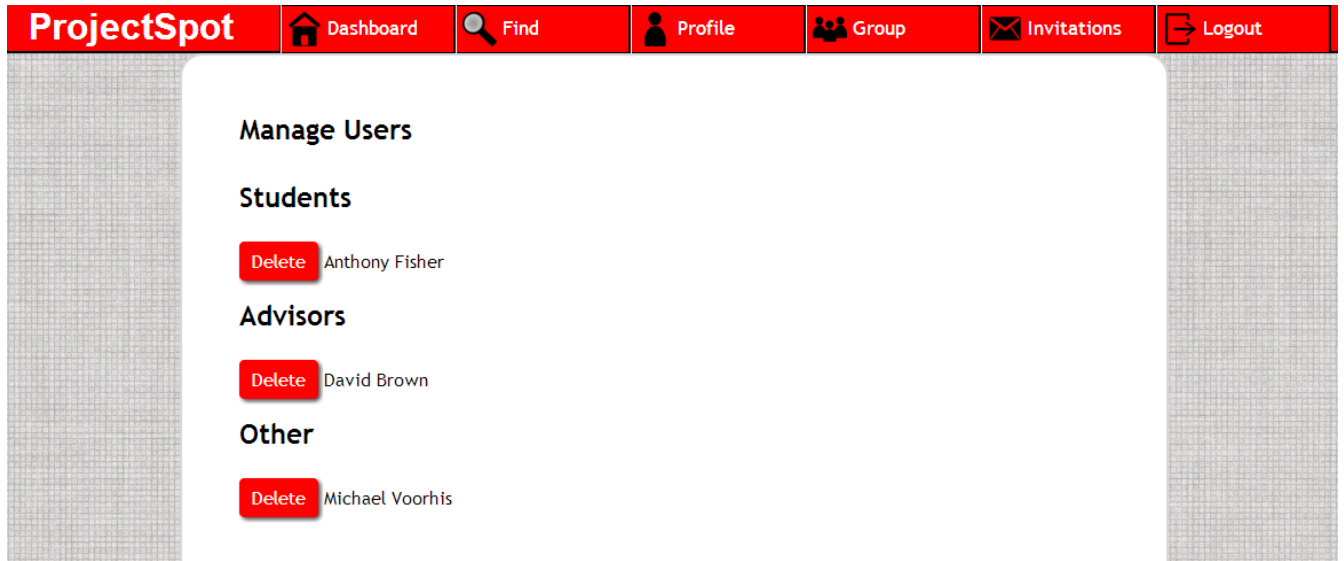


Figure 21: Admin Page for Deleting Users

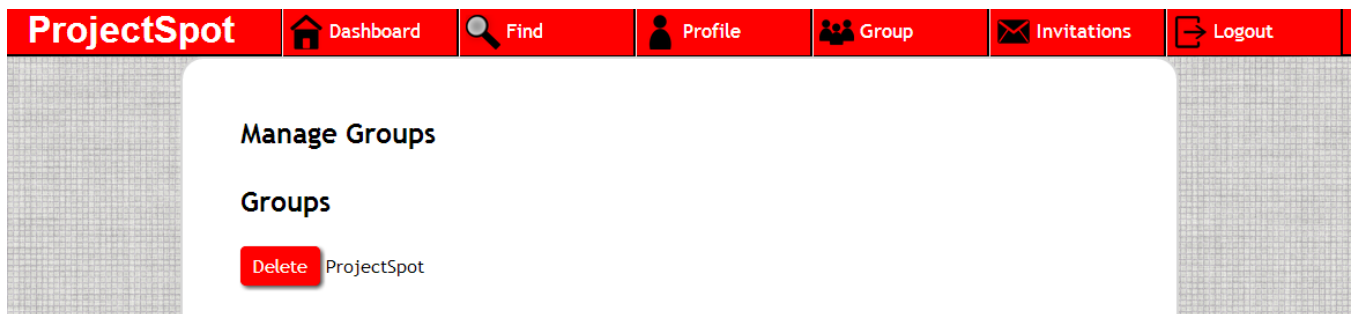


Figure 22: Admin Page for Deleting Groups

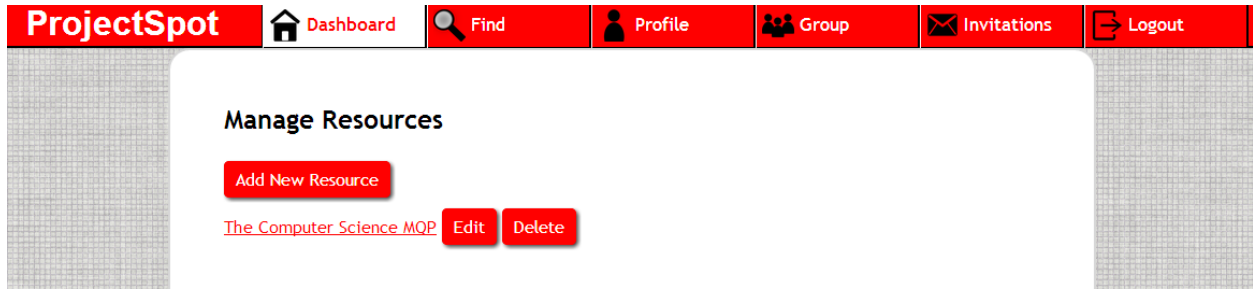


Figure 23: Admin Page for Managing Resources Listed in ProjectSpot

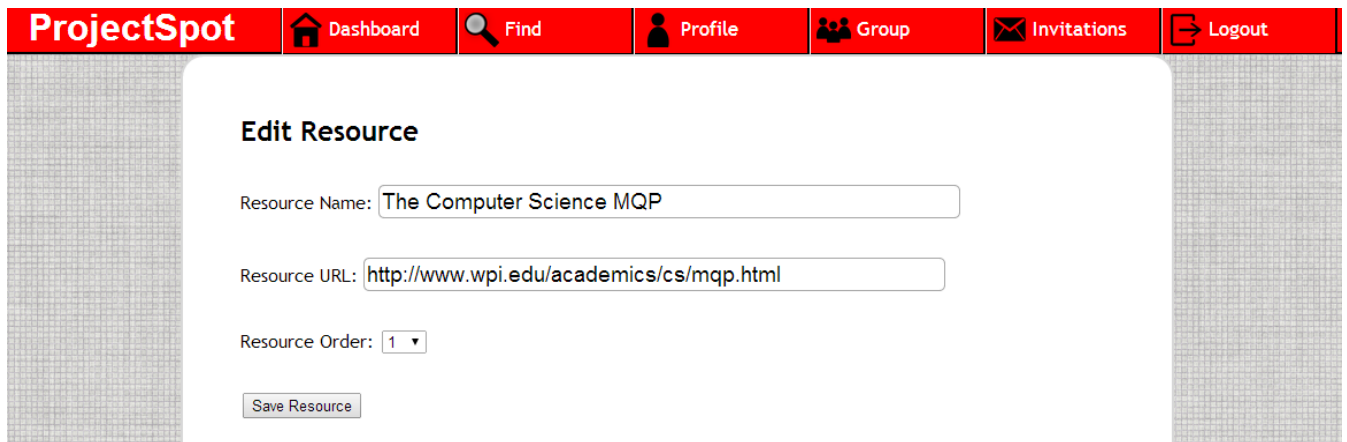


Figure 24: Admin Page for Editing a Resource

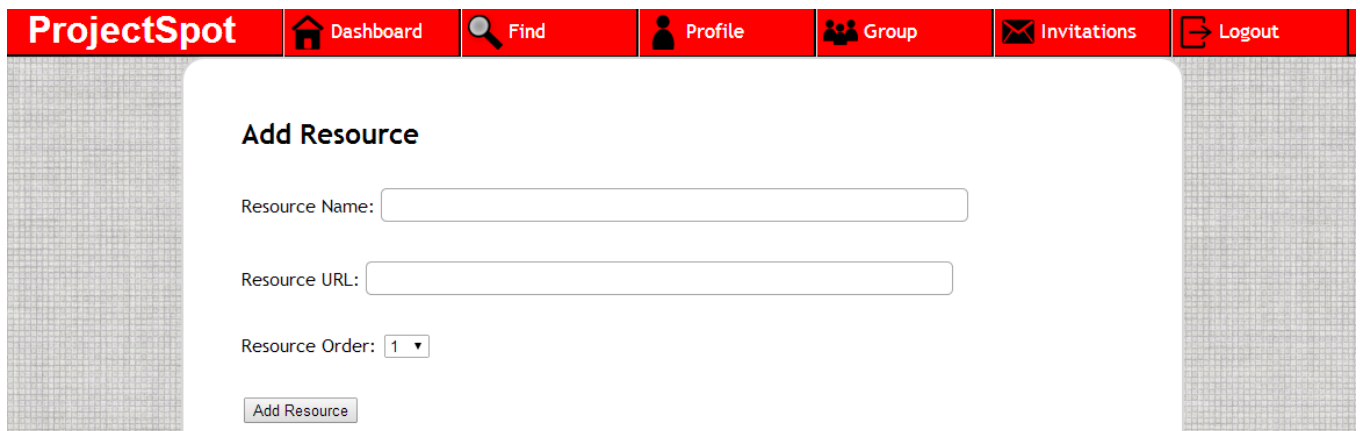


Figure 25: Admin Page for Adding a New Resource

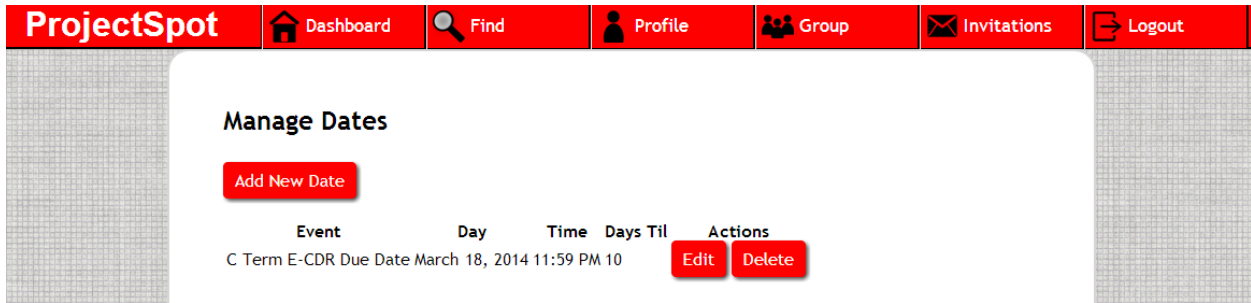


Figure 26: Admin Page for Managing Important Dates in ProjectSpot

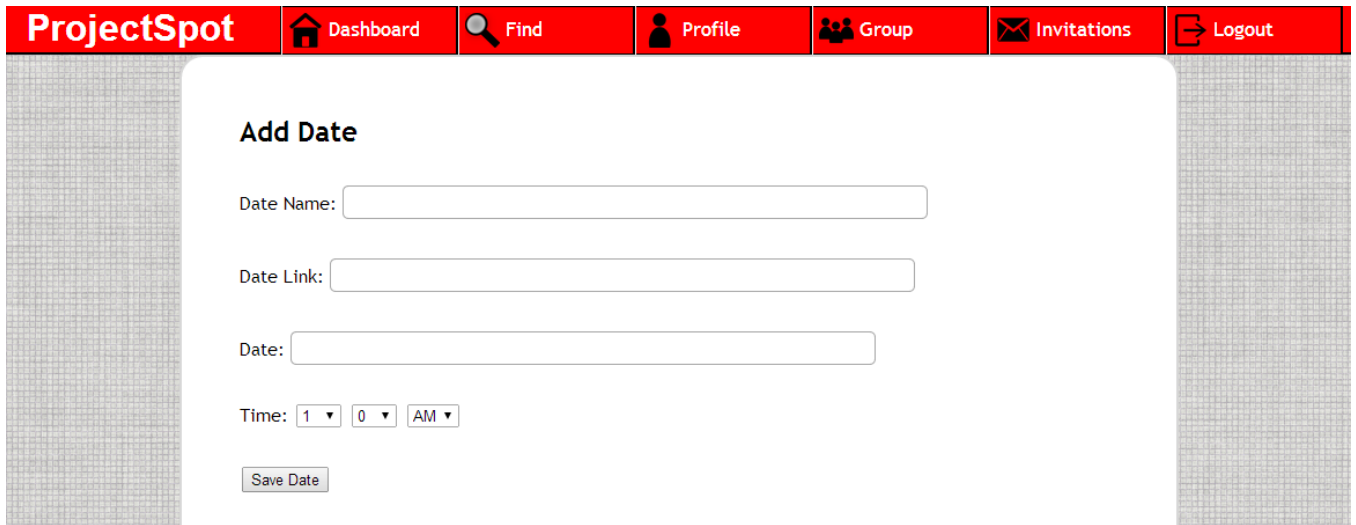


Figure 27: Admin Page for Adding a New Important Date

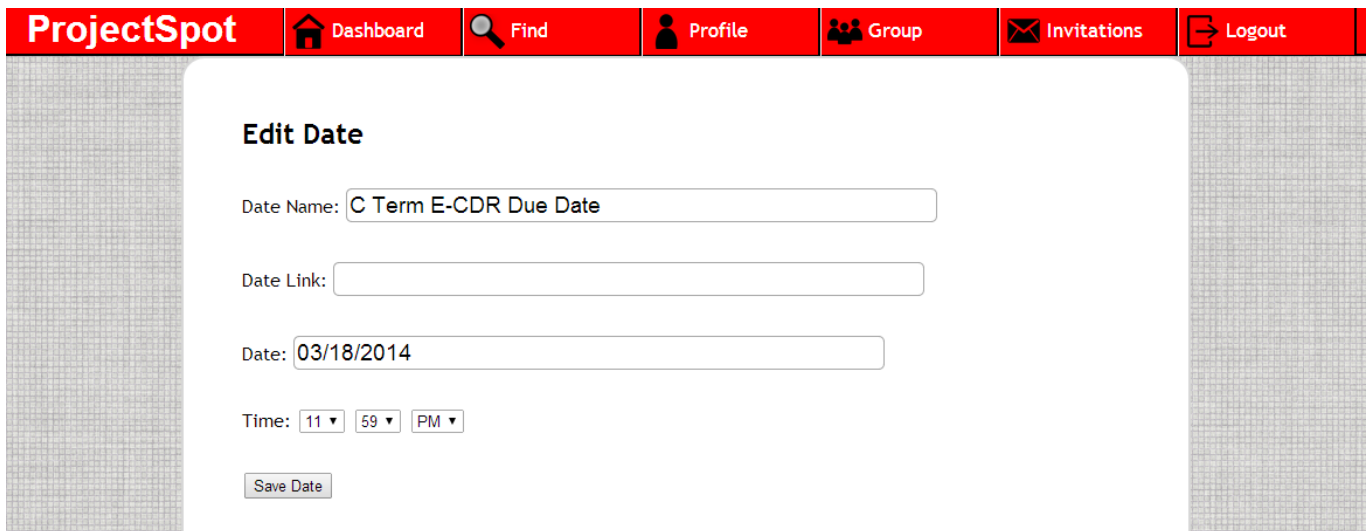


Figure 28: Admin Page for Editing an Existing Important Date