Marissa Stephens
Nick Mohr
Danielle Man
6.170 Software Studio: Project 3.2

GroupFinder

Overview

Motivation

We seek to make it easy for students to form teams for group projects

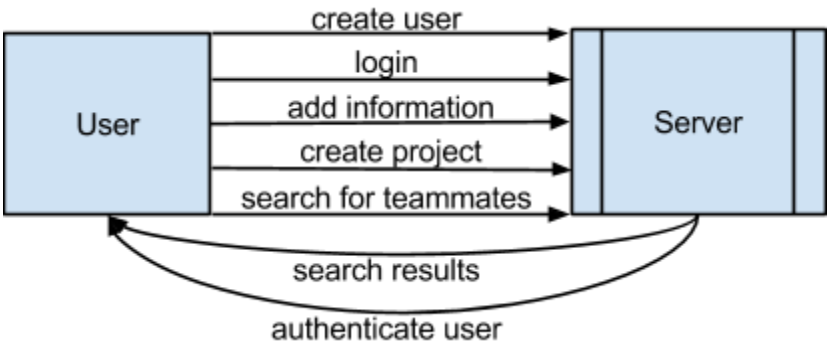
Key purposes

1. The main purpose of our app is to allow MIT students to easily find teammates for group projects. From what we've noticed, MIT students (especially in course 6) often have difficulty forming groups for team projects due to the vast size of classes and restrictions imposed on the projects - like not being able to work with people with whom you've worked before. As the compatibility of the team often affects the grade a team receives, students want teammates that they can rely on. Students usually don't want to work with strangers, and in large classes, it's hard to know how many of your acquaintances are in the class (and which of those are actively looking for partners). By providing the names of people who are looking for partners for a particular project, this app attempts to rectify that issue.
2. We seek to ensure team compatibility. There are many factors that affect team cohesion, such as location, workstyle, and schedule. Often, students want to share things in common with their teammates, like classes, extracurriculars, and dorms/FSILGs. Students can also have very different styles of working- ome are procrastinators; some like to get things done early; some like to work in a group; some like to work alone; some like to commit 20 hours a week; some only want to commit 5 hours a week. If students can find partners with similar workstyles, then the team as a whole will have a better understanding of how it can be most efficient.
3. We want to help students form teams with complementary skill sets. Each student is a unique person with a unique set of skills. Some are natural leaders and organizers whereas others can work hard as long as they have some direction first. Some students prefer frontend, others backend, still others testing (although very rare). A good team has a balanced set of skills, and allowing users to know their teammates' strengths can makes dividing the workload a lot easier.

Deficiencies of existing solutions

Currently there is no commonly used system for creating teams other than speaking to students who are in the class. For the most part, students prefer to work with people they have had some previous interactions with outside of the class, rather than complete strangers. In large classes however, it's often hard to know which of your acquaintances are also taking the class. Previously, there has been an app called Setup that allowed users to find pset partners that have the similar study styles. In a group project, you need a variety of skillsets to be optimally productive. Classes also allow students to fill out a google form (if they were not able to find a team on their own) and allowing the class staff to assign them to other teams. We witnessed this in 6.005: people were writing Kerberos names on the board and yelling to find other people who they had never met before, but were now about to work on a big project with. For this project we also saw many sponge-talk emails to our dorm, where people were asking for one or two more group members for this project and just hoping that someone would respond. The deficiencies of existing solutions mainly boils down to leaving something extremely important (which is group dynamics for a big class project) to chance.

Context diagram



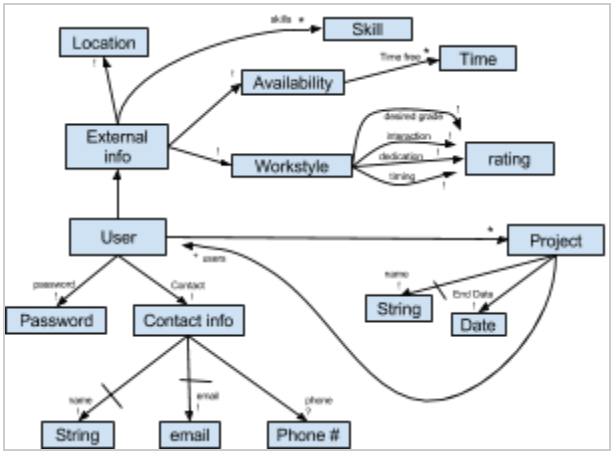
Design Model

Concepts

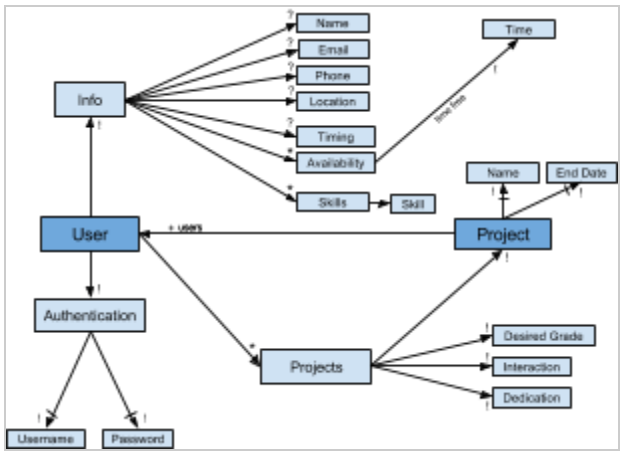
1. **Project**: created by a user such that it can be accessed by other students (helps fulfill purpose 1)
2. **Prioritization** by external factor: Users can prioritize other users that have specific external factors (helps fulfill purpose 2)
3. **Matches**: Users that have a desired external factor (helps fulfil purpose 3)
4. **External Factor**: something such as location, workstyle, availability, or skillset that a user might want their teammates to have. (helps fulfil purpose 2 and 3)
5. **Skill**: A student can have some specific skills that will help other people select based on skills (helps fulfil purpose 3)

Data Model

Phase 1



Phase 2

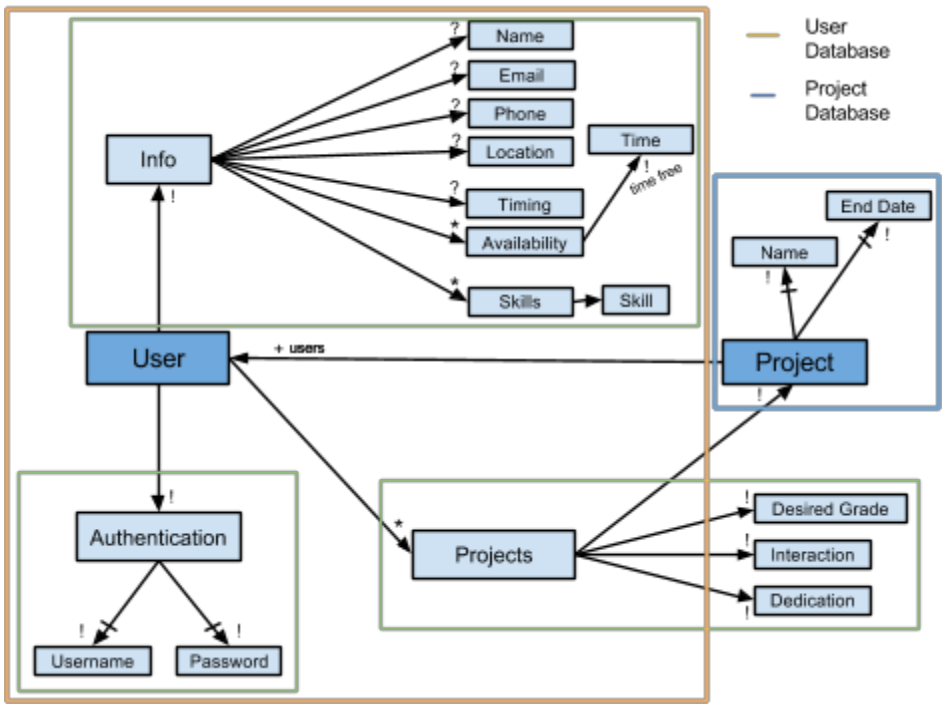


Here we show two data models, the one on the left being the data model from phase 1 of the project, and the one on the right being our updated data model for phase 2 of the project.

One can see that the model has changed quite significantly, the underlying change being that we altered how a User's information is stored. For organizational (and security and readability) purposes, we segmented the Users metadata into 3 categories: Authentication, Information, and Projects. There were many reasons to do this, the foremost being the improvements to security that it brings. When we return a user's JSON in our API, we don't want to expose a user's password - thus we only ever allow access to user.projects or user.info - user.authentication is solely used by the signup/login/logout methods, and is never exposed as JSON. We also realized that 'Contact Info' and 'External Info' could be combined into an overarching category of 'Info', because when we do want information about the user (mainly in our filter), we want access to all of that information at one time. Lastly, a user's projects are in their own category, separate from the authentication of the user's account and from the profile information that a user has. This did not change from our first iteration of the data model; however, we have moved the fields *desired grade*, *interaction*, and *dedication* to be specific to each of the users projects, as opposed to being set globally. The change was intuitive because a user might have a different desired grade for his/her projects in different classes (depending on the user's status in the class), so it didn't make sense to have only one overarching *desired grade* information field; the same logic applies for a user's *interaction* and *dedication* to a project.

There are a few other minor edits to the data model. We decided to authenticate our users based on usernames (as opposed to email), so email is no longer an immutable field and the field of username has been added. A project's end date is now also immutable, because this is how we will be removing expired projects from the app. For clarity, the distinction between *timing* and *availability* is that timing refers to the times of day when a user prefers to work (and is a number that represents this), and availability is an array that represents the user's weekly schedule.

Data Design



Challenges

Design Challenges

1. How should we represent the external factors? There are factors that are universal across all projects for a user such as availability, when they like to work on projects, their skills, and their location. On the other hand there are factors that may vary from project to project such as desired grade, dedication, and amount of interaction. We could have users define each factor for each new project, but that would take up a needless amount of the user's time. Alternatively, we could set these factors to be the same for any of the projects the user is a part of. This saves user time but sacrifices user's desired functionality. To compromise, we set some factors on a per-project basis and others just universally for the given user.
2. How do we test? For this section, we just used a series of ajax calls that can be found in the public folder in unit_tests.js and viewed in the routes/views of the app at /tests. While we could have used some better testing frameworks, the lack of instruction and time to learn on our own prevented us from using complex frameworks. Additionally, since JavaScript runs functions asynchronously, we needed all of our test to cascade, which made traditional frameworks like Qunit much more difficult to use, because those testing suites usually test individual functions and our tests depend on a series of AJAX calls *in order*, and their corresponding responses. It needs to be guaranteed for example that you sign up a user before adding them to a project, and those two events might not necessarily happen in order if they are not cascaded.
3. How do we filter? We want to be able to display best matches with the current student without too much difficulty. We could do some machine learning, or we could sort based on how good the match is in one category, or we could do somewhat arbitrary weightings on what is important in a group. We decided to go with the latter because it is somewhat simple to implement (unlike machine learning) and also allows for users to create their own weightings via the URL.
4. Should a user contain a list of projects or should a project contain a list of users? A user containing a list of projects makes it easier to sort users based on their projects. However, a project containing a list of users makes it easier for a group of users to be associated with the same project, so if project information changes then it only has to change in one place, which is why we decided to give project a list of users.
5. How do we get rid of old projects? One solution is to never get rid of them. That was unfavorable since that could lead to a lot of unused, expired projects clogging up the database. Another idea was to have a specific end date, say the date the project is due. This was obviously preferable so that led to the question of who gets to set the end date. Either the first user who created the project gets to set the end date, which could lead to an unfortunate case where the user sets it too early or too late, or we automatically delete a project if it has no users actively searching for a partner. This is problematic since if the users do not all notify the database that they are no longer looking for a partner, then we have the same issue as if we never got rid of any project. Therefore we decided to allow the first user to set a project due date at which time the project is deleted.
6. How do we create projects? One solution was to get a list of all MIT courses and then allow users to select the course for the specific project they were trying to build a team for. However this does not take into account the fact that students might have several group projects in a course or the fact that not all group projects will be associated with a course-such as a hackathon team. Therefore, we decided to let the users create projects, or select a pre-existing project. While this does allow users the chance to

create several similarly named projects, it was decided that the benefit of extending the definition of group project far surpassed the error that could occur between the mouse and the keyboard.

7. How do we authenticate users are from MIT? While it would be nice to use certificates, some students might not want the app to access their certificate if it is not safe, which would reduce the appeal of the app. Instead, we chose to allow users to login with thier MIT email, which would then also be used as contact information and still checks that they are related to MIT. We decided to use passport to login users and not authenticate that they are from MIT.
8. How do we retrieve MIT data, like class numbers and dorm locations? We could hard code this ourselves in the app, we could ask users to input it as they go, or we could pull from one of IS&Ts resources. The cleanest way would probably be to pull from IS&Ts resources, but that would expand the scope of the app quite significantly. Else we would have to ask for a lot of information from users, which significantly takes away from the experience of the user. Ultimately we decided to forgo this step and allow users to simply create any unique project they desire. This makes the app more applicable to non-mit based projects.

GroupFinder: API Documentation

AJAX References can be found in the public/unit_tests.js method calls

---------- Accounts ----------

POST /signup
Adds a new user
username: a string representing a user's username
password: an object representing an user's encoded password.

POST /login
Logs in user
email: user's email
password: user's password

POST /logout
logs out current user

---------- Users ----------

GET /users/{username}
Returns the user info associated with the requested user, if that user is logged in.
username: id of a specific user

GET /users/{username}/projects
Returns a list of projects that the user with the requested username is part of, if that user is logged in.
username: username of a specific user

PUT /users/
Updates the information of the logged in user.
password: String representing a user's password
name: String representing a user's name
email: String representing a user's email
phone: Number representing a user's phone number
location: String representing user's location
availability: Array of Strings representing times a user is available to work
skills: Array of Strings representing different skills of the user
timing: Number between 0 and 1 representing when a user likes to work on his/her projects

DELETE /users
Deletes the logged-in user. Otherwise throws an error

---------- Projects ----------

GET /projects
Returns a list of all the projects created thus far

GET /projects/{project_name}/users
Returns a list of all username for a specific project
project_name: String representing the name of a project

GET /projects/{project_name}/users/filter
returns a list of users that are a part of project that are ranked with respect to how well they match with the user given a user-inputted weight system
project_name: String representing the name of a project
URL Variables:
location: Number, default value of 1, representing weight of matched location
availability: Number, default value of 1, representing weight of matched hours available
grade: Number, default value of 1, representing weight of matched desired grade
interaction: Number, default value of 1, representing weight of matched type of interaction between teammates
dedication: Number, default value of 1, representing weight of matched hours dedicated to project
timing: Number, default value of 1, representing weight of matched timing
skills: Number, default value of 1, representing weight of matched skills
skillset: array of strings, default to empty, representing different skills

POST /projects/
Adds a new project to the list of projects
URL Variables:
project_name: string representing the name of the project
project_due: date representing when the project will be deleted

POST /projects/{project_name}/users/
Adds the logged in user to a specific project
project_name: String representing the name of a project
grade: Number, default value of 1, representing weight of matched desired grade
interaction: Number, default value of 1, representing weight of matched type of interaction between teammates
dedication: Number, default value of 1, representing weight of matched hours dedicated to project

DELETE /projects/{project_name}/users/
Deletes the logged in user from a project
project_name: String representing the name of a project

DELETE /projects/{project_name}
Deletes a project
project_name: String representing the name of a project