6.170 Software Studio
Fall 2014

TeamMotivate: Problem Analysis

POST /users
POST /sessions

Motivation

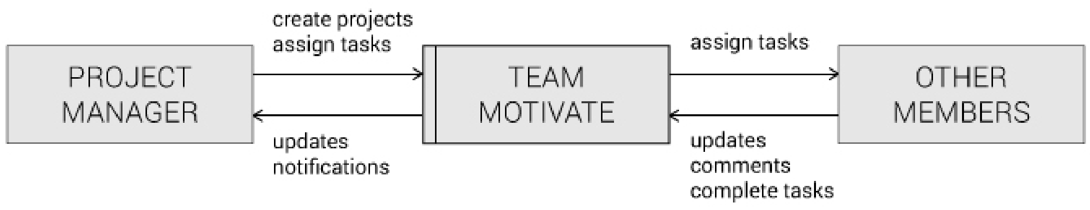
TeamMotivate is an online app with the intent of providing a user friendly, and simple interface for members of a large-scale project to monitor and track progress of the project as a whole. It allows for members to create new projects and add or assign them to other members. It also lets the user keep track of and prioritize their individual tasks

Purpose

- **Provide a means for different departments of a company to keep track of progress of a project**. In the case of large scale projects, it is essential to be able communicate between departments to efficiently gauge and monitor the progress across disciplines.
- **Help workers to keep track of pending tasks and to prioritize these tasks**. By having an interface that allows users to keep track of what tasks they have to complete within different projects, TeamMotivate seeks to clarify each user's responsibilities and help users prioritize their work.
- **Allow members to see how their work fits in to the rest of the project**. The app helps users understand what part of the project they are contributing to and how the completion of their part allows other parts of the project to progress.

Context Diagram

The scope of this project would be a large company setting, with multiple departments that need to communicate effectively to accomplish a task.



Concepts

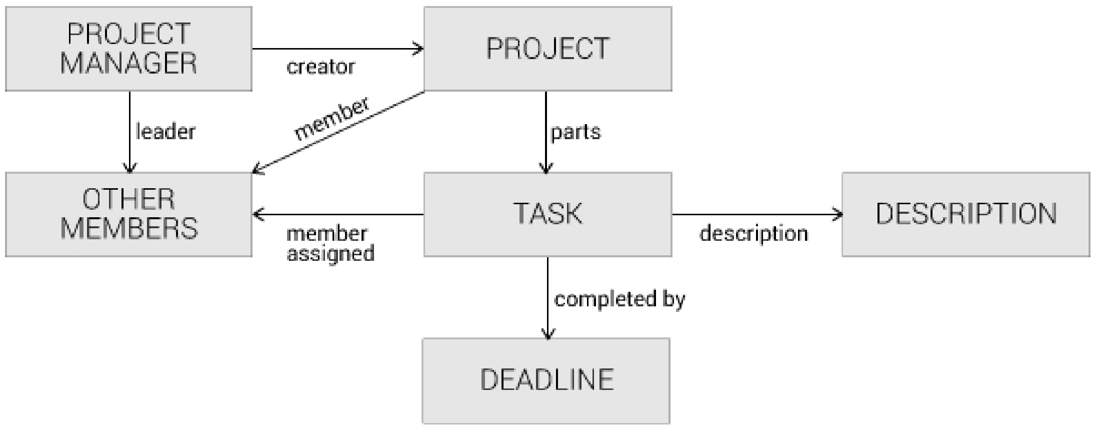
Project: This is the key concept our application is based on. Our application runs on the existence and completion of large scale projects. A project keeps the tasks organized and allows users to see how their work fits in with everything else.

Tasks: Each project can be split up in to multiple tasks that may be assigned and kept track of individually. Tasks help members stay organized.

Deadline: A deadline may simply be thought as an attribute of a project or task. However, in the case of large scale projects, deadline is a pivotal concept that determines the success of a project. The idea of a strict deadline is utilized: if a project is not completed by this deadline, repercussion will be enforced. Deadlines help users keep organized by knowing which projects need to be done first.

Progress: The progress is a value that ranges from 0 to 100%, showing the progress on an action item. This can be set by team members, and when taken together with an action item's estimated time to completion, can be used to calculate the total progress on the entire project. This fulfills the purpose of tracking project progress.

Data Model



Design Challenges

Creation and priority assignment of tasks

How are projects and tasks created? Similarly, how is priority assigned? The project manager should keep track of all projects created under his main project. However, it is useful for members to create projects that suit their organizational needs in implementation.

Potential solutions:

1. *Only project manager can create projects or tasks and assign priority*: power is solely in the hands of the project manager.
2. *All users can create projects or tasks and assign priority*: project manager oversees the whole project, so it is important for them to have the final say on the choice of projects and tasks to be completed.
3. *Other members can propose new projects or tasks and priority assignments for the project manager to approve. Project manager can create projects and assign priority*: encourages other members to actively participate in the overall process of completing the project, while giving the ultimate control to the project manager.

Chosen solution: 1. Only project managers can create projects and tasks so that the project manager is aware of all upcoming todos. This is also the simplest approach.

Dropping out of a project

If a member drops out of a project, he is removed from the database, and steps are taken to ensure that the tasks do not disappear with him. The tasks must be reassigned to another member without breaking the application.

Potential Solutions:

1. One way to prevent this is when a member decides to drop out, the project manager reassigns the tasks to people with proper qualifications and are least busy before actually removing the member from the database.
2. Another less feasible way is to prevent a member from dropping out before completing all his assigned tasks. Doing this will conveniently result in not needing to change anything within our application, but there are probably legal reasons why this cannot be implemented.

Chosen solution: 1. We allowed the leader the authority to remove a member from a projects. When a user leaves, it makes sense to allow the leader to redistribute tasks and remove that member individually.

Completion of tasks and projects

After tasks and projects are completed, should they be deleted right away or archived?

Potential solutions:

1. *Delete completed tasks and projects*: keeps the pending tasks and projects free of clutter, but at the same time it becomes hard for users to determine which tasks are done and which have yet to be added.
2. *Remove tasks and projects from the hierarchy and store them in an archived section*: keeps the pending tasks and projects free of clutter, but users would have to check the archived section to verify the completion of a project.
3. *Update their status to be completed but keep them in the hierarchy*: users can continue to keep track of what has been done for a project, but pending tasks and projects list will be harder to navigate through.

Chosen solution: 1. Deleted tasks and projects are removed from the database and not archived. This is the simplest approach and also keeps the database clean.

Owner of Tasks in Data Model

One design challenge was to determine what should own tasks in the database.

Potential solutions:

1. *Have tasks owned directly by projects, and have the tasks store which users are assigned to them.* This approach places the project at the highest level of importance.
2. *Let tasks be its own top-level collection, and to have everything that uses tasks store a list of task id's.* This approach places tasks at the highest level of importance.

Chosen solution: 2, because many different things depend on tasks, so giving tasks its own collection provides the best protection against performance bottlenecks and explicitly states its importance.

Progress Tracking

How should the tracking of progress be done? Many different ways were considered for project progress to be tracked.

Potential solutions:

1. *Qualitative progress tracking by allowing team members to write short progress reports.* This is the approach that allows for the most detail, but is also the most complex feature.
2. *Allow team members to directly edit the estimated time to completion*. One disadvantage of this approach is that it causes the original size of the task to be lost.
3. *Allow team members to exercise their own judgment and assign a percentage completion rate to tasks.* This is a small additional feature that relies on a user's intuition in order to be useful.

Chosen solution: 3. This quantitative approach allows the overall completion to be calculated by aggregating the progress for individual tasks. At the same time, this choice allows for a vivid depiction of task progress.

Summary Given to User

How should the user receive a summary of his situation, in terms of projects he is a member of and tasks he is managing?

Potential solutions:

1. *Display all information on projects and tasks. For project leader, display all tasks of project.* This option allows the user to see all information on the tasks assigned to the user and the project the tasks is part of. The project leader will be able to see tasks assigned to everyone for the project they are leading. This option gives the user the most comprehensive information, but it may overwhelm the user with dense content.
2. *Display all information on the tasks assigned to user. Name the project they are part of*. This option allows the user to focus on the tasks that they have to work on while being aware of what projects they are part of.
3. *Display all information on the projects the user is part of.* This option gives the user an overview of the projects they are part of, but does not give them an idea of what they need to work on, which is what is most immediately important to their individual contribution to the project.

Chosen solution: 2. This option allows the user to understand what project they are part of while still keeping focus on the tasks they are assigned to. Option 2 effectively presents the user with only the most important information and prevents information overload.

Tasks Shown to User

Given that users have access to tasks that they are not responsible for, now should tasks be displayed to a user?

Potential solutions:

1. *Show all tasks to the user.* This allows the user to get a good idea of what work is required for the project, at the expense of keeping the UI clean. Good for project leaders.
2. *Show only tasks the user is responsible for.* This is a simple solution, but causes decreased transparency.
3. *Only show tasks the user is responsible for on the homepage, and show all tasks on a separate page.* Increases the difficulty of seeing all tasks while still keeping it an option.

Chosen solution: 3. Transparency and a clean UI are both important to this application's purposes. The increased complexity and friction of the page layout is acceptable. The reason is that it is desirable to give the user easy access to tasks he is responsible for, and it is expected that the user will not want to see all the tasks for a project very often.

Deletion of Projects and Tasks in UI

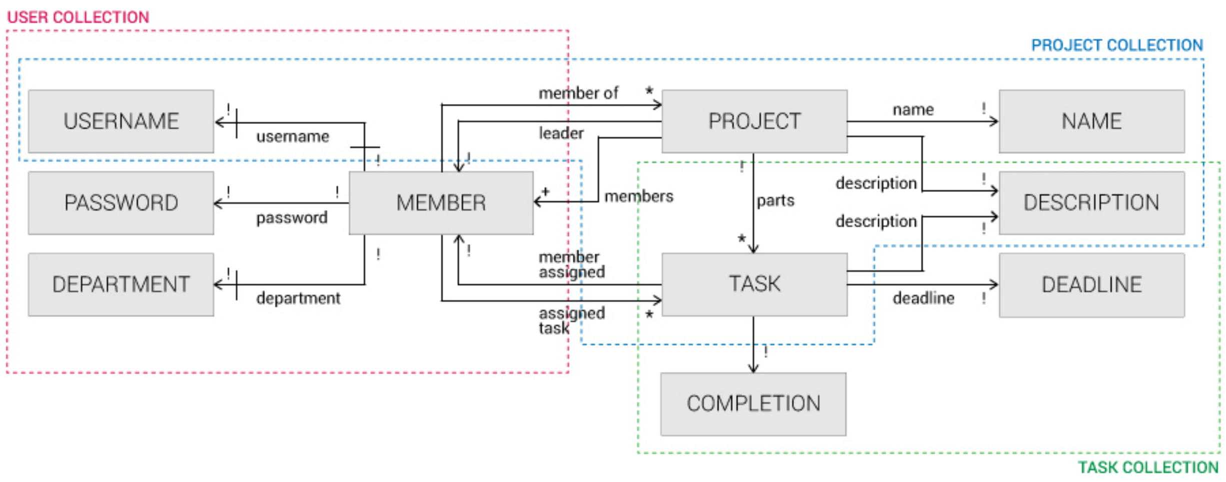
On what page should the deletion of projects and tasks be?

Potential solutions:

1. *Put the deletion of projects and tasks on the respective edit pages.* Makes sense because deletion is a form of editing.
2. *Put the deletion of projects and tasks on same pages they are displayed on.* Can be done in the style of Windows by adding X buttons to each project and task. Makes deletion easier and keeps it separate from plain editing.

Chosen solution: 2. We want to minimize the friction for deletion and other operations because maximizing the time that a user spends on the site or a particular page is not one of our goals. Moreover, the use of an X to signify deletion does not add clutter to the applicable pages, and its meaning is easily understood.

Data Design

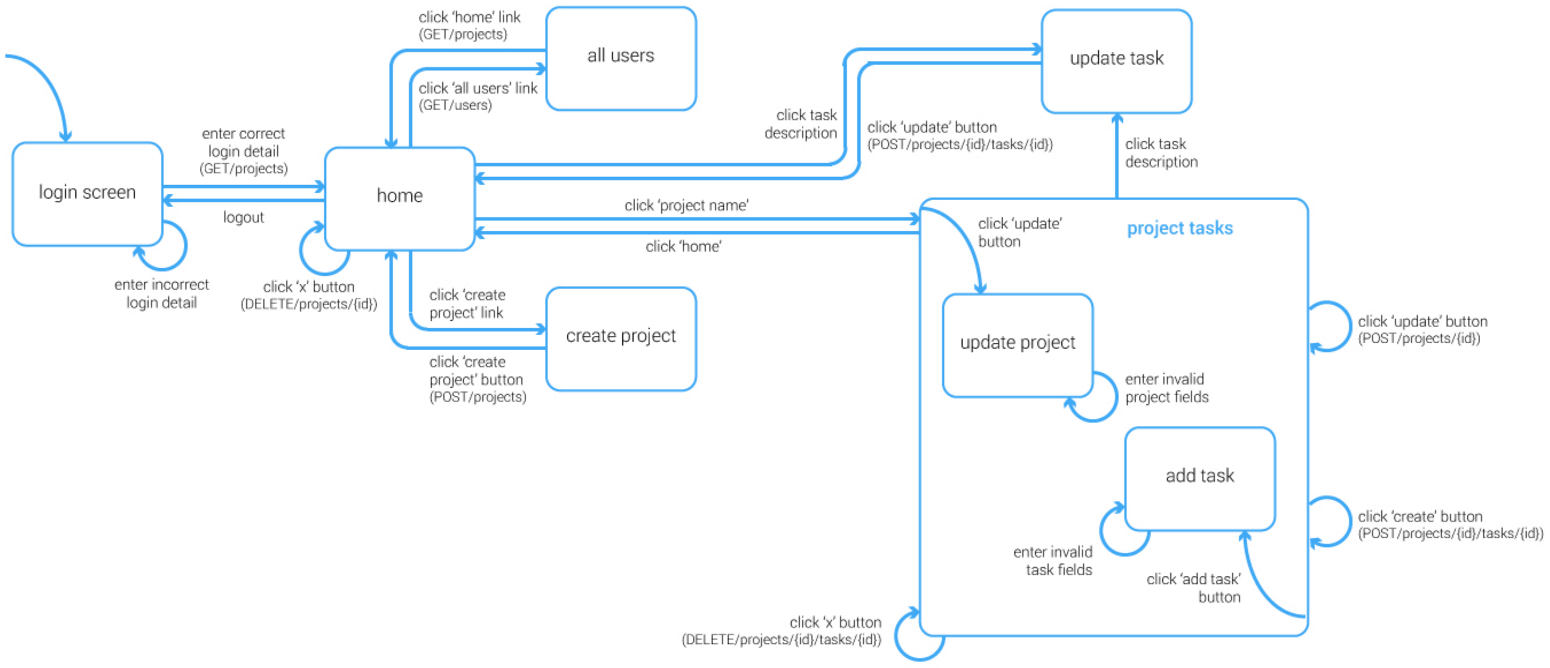


One subtle design element is that a department attribute was included in a member. This is a key focus for the problem that is addressed, because the problem that large scale projects often have is the lack of communication between departments. The department attribute is set here as a reminder of our purpose.

A project should have a deadline, but since project is comprised of tasks, and each task has a deadline, it is redundant to attribute a deadline to the project too. A task's deadline should never be later than a project's deadline.

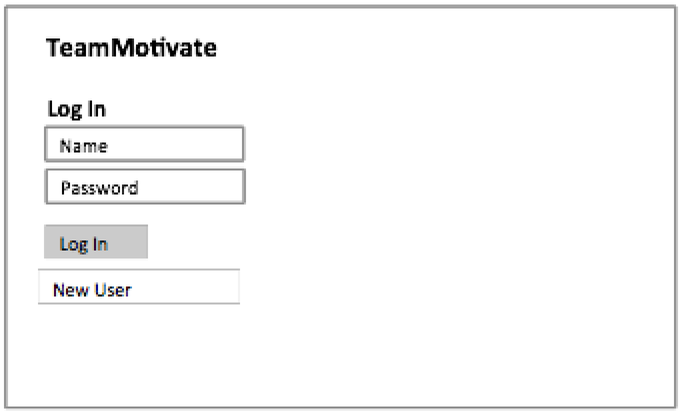
A task has a completion attribute that keeps track of the percentage completion of the task. This is updated by the assignee's judgement to how complete the task is.

Wireframe for user workflows

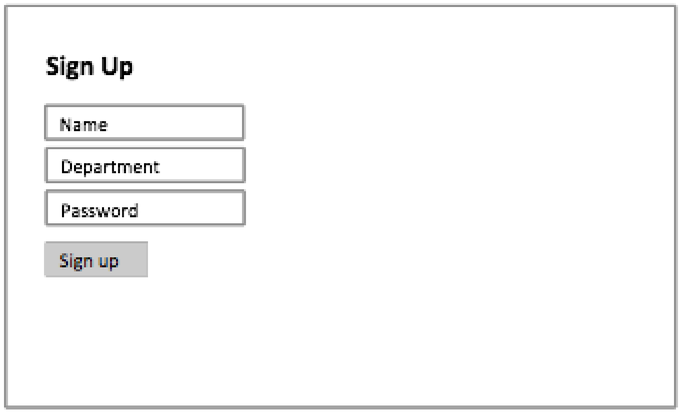

login screen
enter correct login detail (GET/projects)
logout
enter incorrect login detail
home
click 'x' button (DELETE/projects/{id})
click 'home' link (GET/projects)
click 'all users' link (GET/users)
all users
click 'create project' link
click 'create project' button (POST/projects)
create project
click task description
click 'update' button (POST/projects/{id}/tasks/{id})
update task
click task description
click 'project name'
click 'home'
project tasks
click 'update' button
update project
enter invalid project fields
click 'update' button (POST/projects/{id})
add task
enter invalid task fields
click 'add task' button
click 'create' button (POST/projects/{id}/tasks/{id})
click 'x' button (DELETE/projects/{id}/tasks/{id})

Wireframes for page layout

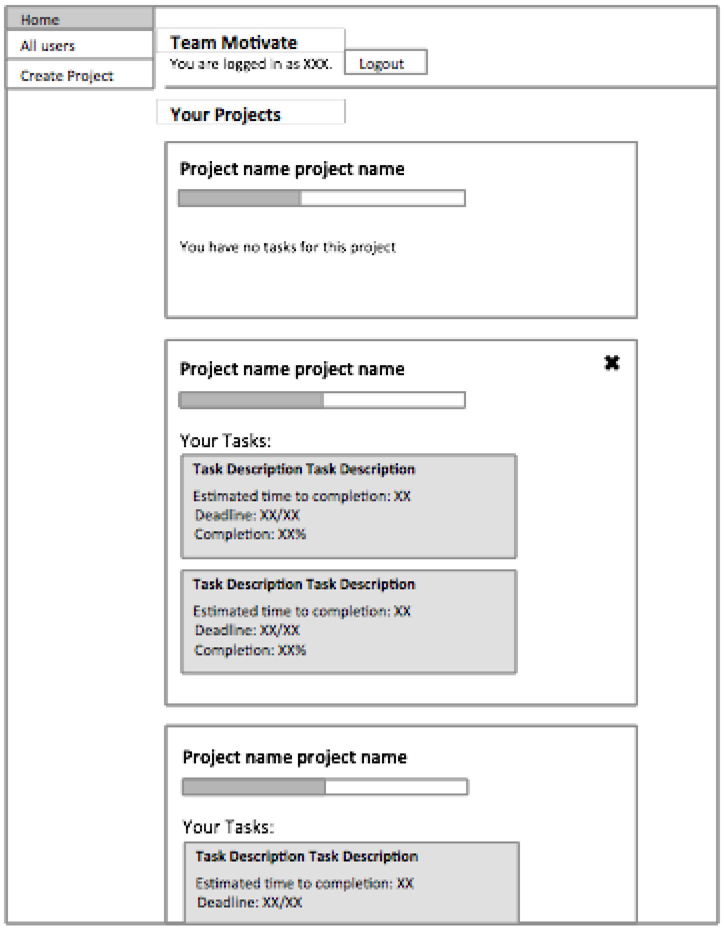
Login


TeamMotivate
Log In
Name
Password
Log In
New User

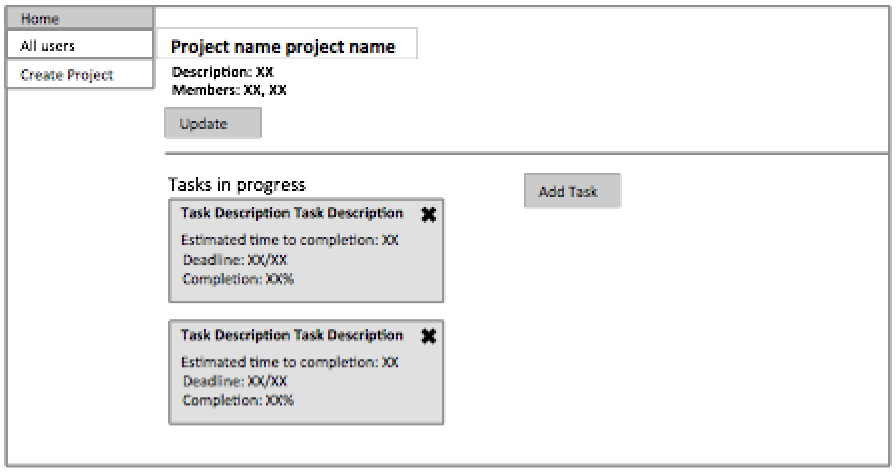
Signup


Sign Up
Name
Department
Password
Sign up

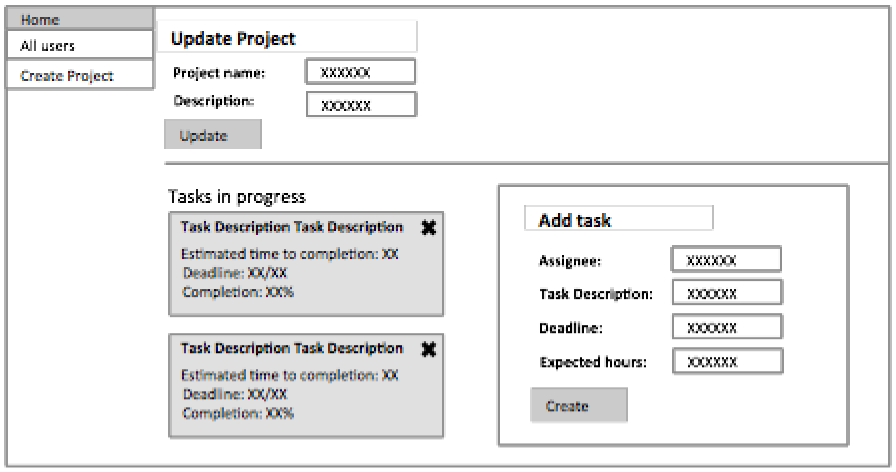
Homepage


Home
All users
Create Project
Team Motivate
You are logged in as XXX.
Logout
Your Projects
Project name project name
You have no tasks for this project
Project name project name
Your Tasks:
Task Description Task Description
Estimated time to completion: XX
Deadline: XX/XX
Completion: XX%
Task Description Task Description
Estimated time to completion: XX
Deadline: XX/XX
Completion: XX%
Project name project name
Your Tasks:
Task Description Task Description
Estimated time to completion: XX
Deadline: XX/XX

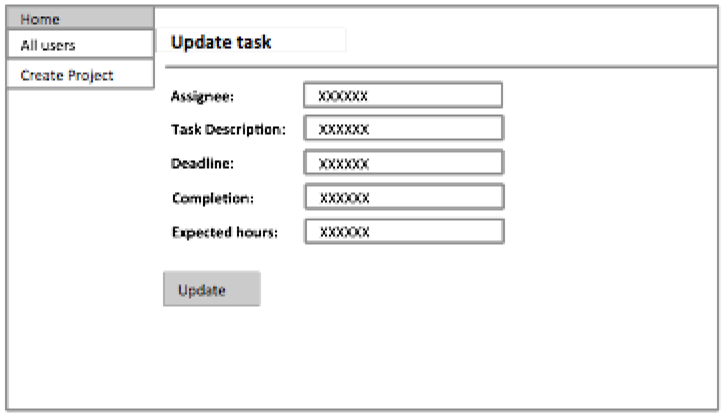
Individual Project


Home
All users
Create Project
Project name project name
Description: XX
Members: XX, XX
Update
Tasks in progress
Add Task
Task Description Task Description
Estimated time to completion: XX
Deadline: XX/XX
Completion: XX%
Task Description Task Description
Estimated time to completion: XX
Deadline: XX/XX
Completion: XX%

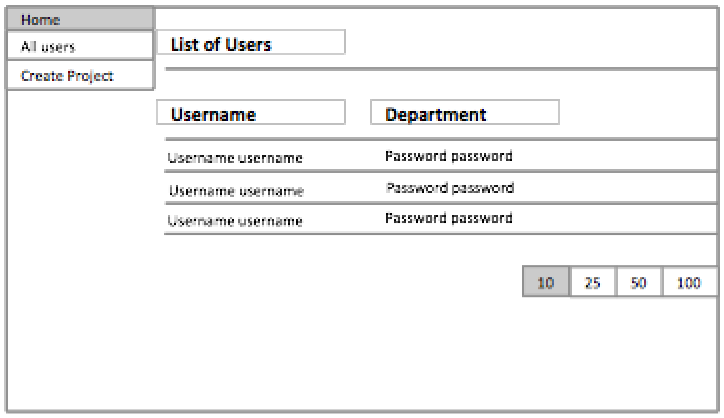
Individual Project (Dynamic Content Shown)



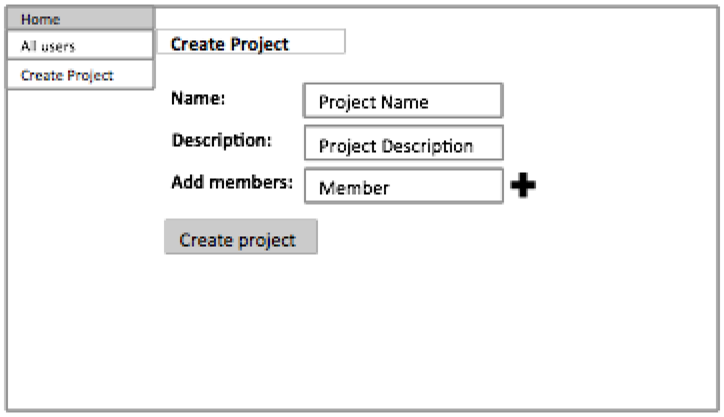
Task Update



User List



Project Creation



Lessons Learned

One thing we learned was that it is important to maintain a separation of models, views, and controllers. We experimented with using angular.js, a framework that provides client-side MVC, for phase 3 of the project. We found that it allowed us to write clean, organized code, and the initial learning curve required to get started with angular.js was more than justified by the amount of work saved by using it.

Another lesson learned was that when designing a RESTful API, one should consider concrete use cases for the client. When writing code from the client's perspective, we found some of our API calls to be missing some desired functionality, which was returning actual objects instead of object id's. In some cases we ended up going back and changing the API. We learned that there is a tradeoff between ease of programming for the API, and ease of programming for the client.

API Specification

GET /projects

Returns the list of projects associated with a user.

```
arguments:
  username: (String) The username of the user you would like to retrieve projects for

returns: {'success': boolean, 'projects': [Object], ‘message’: String}
  success: (boolean) true if a possibly empty set of projects was returned, else false
  projects: (array) Array of projects associated with user. Is undefined when success is
  false. Each project contains:
  {
    leader: (string) the user in charge of the project
    name: (string) name of the project
    description (string) description of the project
    users: (array) all the users in the project
    tasks: (array) all the task id’s associated with the project
  }
  message: (String) error message. Is undefined when success is true

returns: 401 Not Authorized if no user is logged in
```

POST /projects

Create a project with a given name, leader, description, and users.

```
arguments:
  name: (String) name of the project
  leader: (String) The person in charge of the project
  description: (String) description of the project
  users: (array) all the users in the project

returns: {‘success’: boolean, ‘id’: String, ‘message’: String}
  success: (boolean) true if a project was created, else false
  id: (String) the id of the newly created project. Is undefined when success is false
  message: (String) error message. Is undefined when success is true

returns: 401 Not Authorized if no user is logged in
```

GET /projects/{id}

Returns the project specified by the id. User must be a member of the project.

```
arguments: None

returns: {'success': boolean, ‘project’: Object, ‘message’: String}
   success: (boolean) true if the project exists and the user is a member, else false
   project: (Object) a project object. Is undefined when success is false. Contains the
   following fields:
   {
      leader: (string) the user in charge of the project
      name: (string) name of the project
      description (string) description of the project
      users: (array) all the users in the project
      tasks: (array) all the tasks associated with the project
   }
   message: (String) error message. Is undefined when success is true

returns: 401 Not Authorized if no user is logged in
```

POST /projects/{id}

Edit the project’s leader, name, description, and the set of users assigned to the project. User must be the project leader.

```
arguments:
   leader: (string) the new username in charge of project
   name: (string) new name of project
   description: (string) description of the updated project
   users: (array) all the users in the updated project

returns: {'success': boolean, ‘message’: String}
   success: (boolean) true if the update was successful, false otherwise
   message: (String) error message. Is undefined when success is true

returns: 401 Not Authorized if no user is logged in
```

DELETE /projects/{id}

Delete the project. User must be the project leader.

```
arguments: None

returns: {‘success’: boolean, ‘message’: String}
   success: (boolean) true if the deletion was successful, else false
   message: (String) error message. Is undefined when success is true
```

```
returns: 401 Not Authorized if no user is logged in
```

GET /projects/{id}/tasks

Returns the set of tasks assigned to the user for the project with the given id.

```
arguments:
  username: (String) username of the user logged in

returns: {‘success’: Boolean, ‘tasks’: [Object], ‘message’: String}
  success: (Boolean) true if the set of tasks is successfully retrieved, else false
  tasks: (array) the tasks assigned to the user for the project. Is undefined when
  success is false. Each task contains the following fields:
    assignee: (String) username of the user assigned to the task
    description: (String) a description of the task
    completion: (number) degree of completion of the task
    deadline: (date) the date at which the task is due
  message: (String) error message. Is undefined when success is true

returns: 401 Not Authorized if no user is logged in
```

POST /projects/{id}/tasks

Adds a new task to the project with the given id. User must be the project leader.

```
arguments:
  assignee: (String) username of the user assigned to the task
  description: (String) a description of the task
  etc: (number) expected time in hours that it will take to complete the task
  deadline: (date) the date at which the task is due

returns: {‘success’: Boolean, ‘message’: String}
  success: (Boolean) true if task is successfully added, else false
  message: (String) error message. Is undefined when success is true

returns: 401 Not Authorized if no user is logged in
```

GET /projects/{id}/tasks/{id}

Returns the task specified by the identifiers. User must be a member of the project.

```
arguments: None

returns: {‘success’: Boolean, ‘task’: Object, ‘message’: String}
  success: (boolean) true if task is returned, else false
```

```
   task: (Object) a task object. Is undefined when success is false. Contains the
   following fields:
     assignee: (String) username of the user assigned to the task
     description: (String) a description of the task
     completion: (number) degree of completion of the task
     deadline: (date) the date at which the task is due
   message: (String) error message. Is undefined when success is true

returns: 401 Not Authorized if no user is logged in
```

PUT /projects/{id}/tasks/{id}

Change the assignee, description, degree of completion, and deadline of the task at id. User must be either the project leader or the assignee to update the task.

```
arguments:
   assignee: (String) username of the user assigned to the task
   description: (String) a description of the task
   completion: (number) degree of completion of the task
   etc: (number) expected time in hours that it will take to complete the task
   deadline: (date) the date at which the task is due

returns: {‘success’: Boolean, ‘message’: String}
   success: true if task is successfully updated with the field values, else false
   message: (String) error message. Is undefined when success is true

returns: 401 Not Authorized if no user is logged in
```

DELETE /projects/{id}/tasks/{id}

Delete task specified by the project id and task id. User must be the project leader.

```
arguments: None

returns: {‘success’: Boolean, ‘message’: String}
   success: true if task is successfully deleted, else false
   message: (String) error message. Is undefined when success is true

returns: 401 Not Authorized if no user is logged in
```

GET /users

Returns all the users.

```
arguments: none

returns: [Object] array of users; each user contains the following fields:
   username: (String) username of the user
   department: (String) department of the user

returns: 401 Not Authorized if no user is logged in
```

POST /users

Creates a new user.

```
arguments:
   username: (String) username of the user
   password: (String) password of the user
   department: (String) department of the user

returns: {‘success’: Boolean, ‘message’: String}
   success: true if user was successfully created, else false
   message: (String) error message. Is undefined when success is true
```

POST /sessions

Attempts to start a new session with the given username and password.

```
arguments:
   username: (String) username of the user
   password: (String) password of the user

returns: Unauthorized if the login was unsuccessful, or {success: true} otherwise
```