Tourney Track

6.170 Final Project Design Document

Team Members: Jamar Brooks, Yachi Chang, Chris Rogers, Katharine Xiao

I. Design Overview

System Overview

TourneyTrack is a web application that organizes small to medium scale tournaments, such as intramurals and dorm tournaments. An administrator can create tournaments with multiple brackets, either of elimination or round robin type, and players can create teams to play in various tournaments. Our app would then create a clear visualization of tournament status, schedule, and stats, allowing players to easily determine who they should play next and admins to easily manage the big picture.

Purposes

Easy Tournament Management

Creating and maintaining tournaments is often difficult and unorganized, especially for informal tournaments such as dorm or intramural level tournaments. The main solutions to this problem are:

- Automated tournament match making
    - Creating matches between players can be cumbersome, since it requires keeping track of individual match outcomes. Our app allows an organizer to be able to specify the type of tournament and its participants, and the system then generates all matches of the tournament.
    - Tournaments are automatically updated and advanced when the results of a match are approved
- Low-latency match outcome updates
    - Often, especially with informal tournaments, match outcomes are reported via email. This creates quite a bit of latency, because it requires the admin to check his/her email and update the tournament spreadsheet accordingly.
    - Our app allows the outcomes of a match to be viewable shortly after the completion of that match.

Easy Access to Accurate Tournament Information

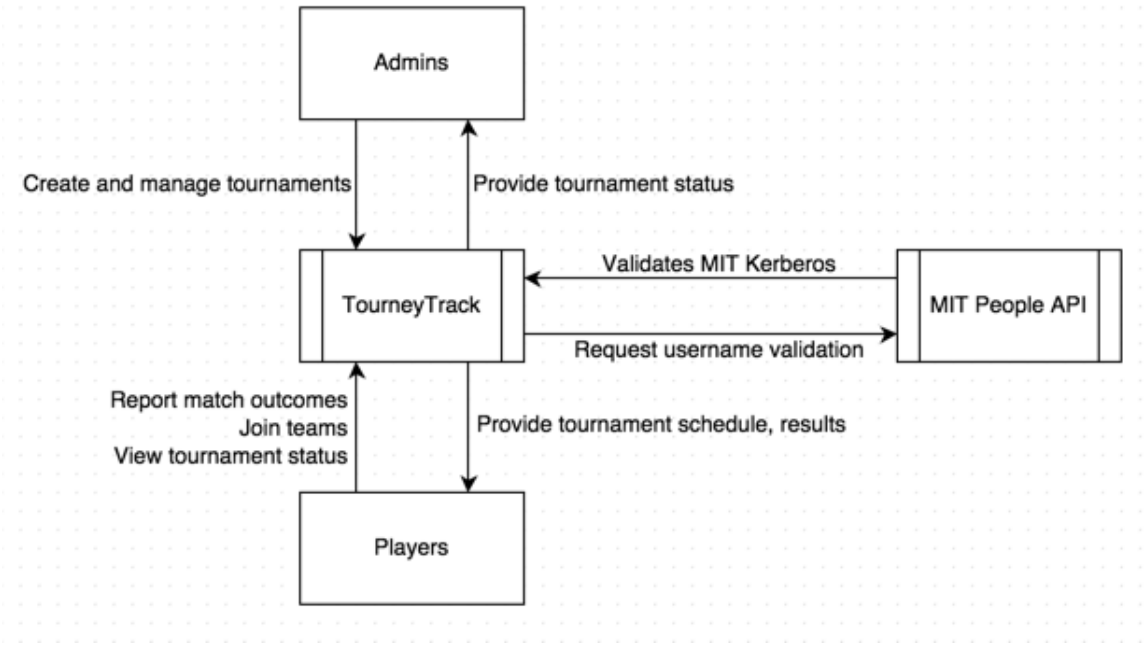
A big problem many participants of informal tournaments face is that big-picture information about the tournament is not immediately clear. Often, participants can only see outcomes to individual matches, making it hard to visualize the standings of all participants in the overall tournament. The following exhibit solutions to this problem:

- Automatically updated team-level statistics
    - Teams should be able to view how they are performing over the course of a given tournament. This information should be available immediately, once match outcome data has been entered into the system.
    - Many solutions provide too many statistics (too fine-grained or too coarse-grained), which causes the stats that many users care about to be lost in a sea of information.

- Big-picture match schedule visualization
    - Users should be able to quickly view a tournament and see what matches they participate in, and how these matches relate to other matches in the tournament hierarchy.
    - Many existing solutions do not provide an intuitive representation of the matches for a tournament.

Context Diagram

Our context diagram is as follows:



https://www.draw.io/#G0B2Mr49HAZeoOMzFHdmtwLU5hdVk

Tournament administrators will be able to create and manage tournaments, while the app provides admins with the current tournament status. Players can use the app to see what matches they still have to play, and also browse tournaments to find teams to join and play. After a match has been completed, each player in that match can report the match outcome on TourneyTrack, and the app will then notify the tournament admin to approve those results.

The scope of our app is currently restricted to the MIT community. All players and admins must register with a valid Kerberos, and upon registration, our app validates the entered username against MIT People API to make sure it's valid.

II. Design Model

Data Model and Concepts

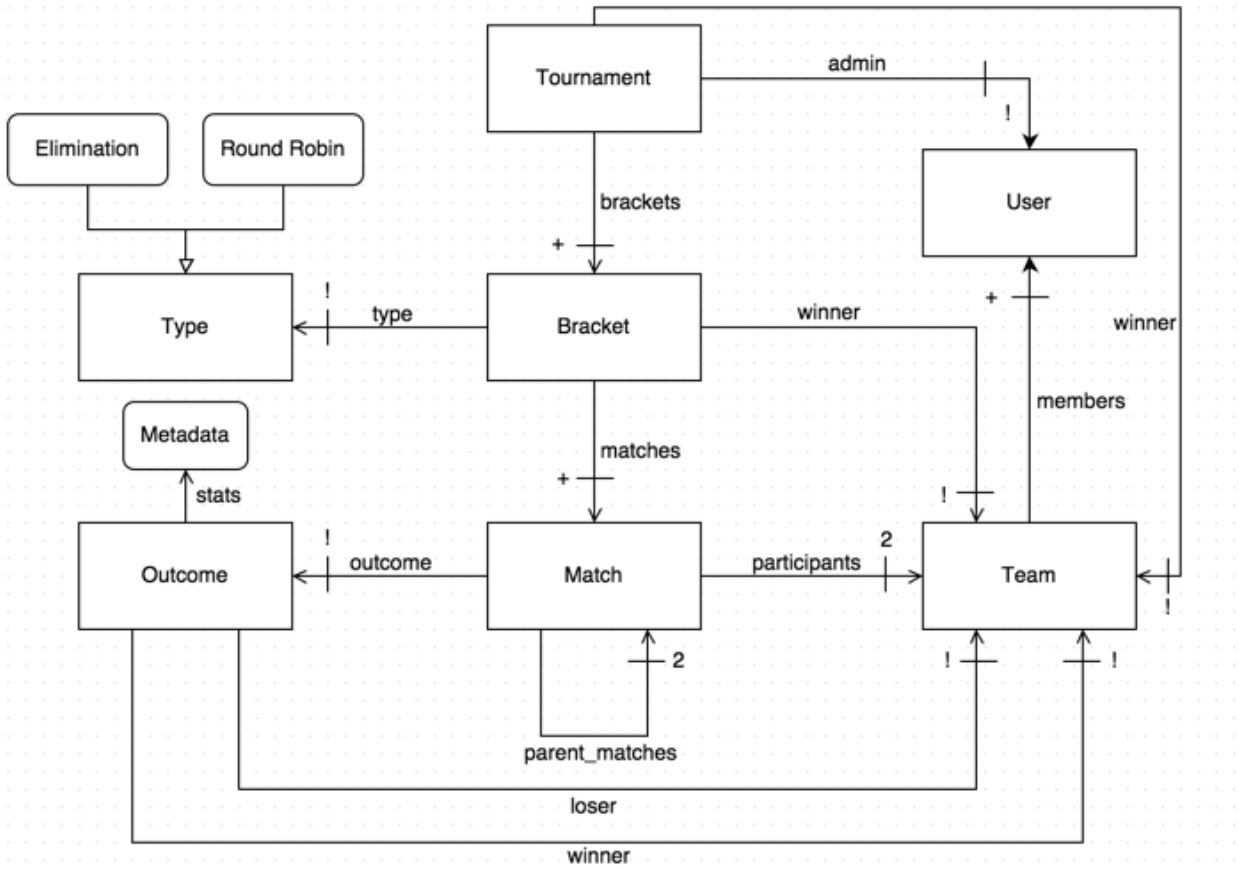
To model our application, we introduce the following concepts:

- **User:** A User is an individual MIT affiliate, and he or she can either create/manage tournaments as an administrator, or join teams in tournaments as a player.
- **Team:** A Team is a list of users that will play together in a specific tournament, and is created by a team captain, which is a User. A specific Team instance will only last through the single tournament that it is part of. If a group of Users wishes to participate in another tournament, they must create a new Team for that tournament. In the case of individual sports, a Team will simply consist of one single member.
- **Match:** A Match is a pairing of two Teams, who will compete against each other. Each Match also has at most 2 parent Matches. The first parent Match is the Match that the winning Team of this Match will participate in for their next round while the second parent Match is the Match that the losing Team will participate for their next round.
- **Outcome:** Each Match will also have an Outcome, and this stores all the metadata and statistics regarding a match. This includes which Team won, what the scores were, etc. Outcomes will be used to calculate Team statistics (for example, by aggregating on how many Matches a specific Team has won or loss).
- **Bracket:** A Bracket is a collection of dependent Matches. Multiple Teams will participate in a Bracket. Matches and their dependencies will be automatically generated based on the Bracket's type (elimination or round robin). When all the Matches in a Bracket have been played, we can aggregate the Outcomes and declare either the Team who won the most matches to be the winner of the Bracket in the case of Round Robin, or the Team who won the final elimination match in the case of Elimination.
- **Tournament:** A Tournament is a collection of related Brackets. For example, a losing Player will move from the main Bracket to the loser's Bracket in a Tournament. A Tournament is automatically generated, given the list of participating Teams and a Tournament type (elimination or round robin).

To illustrate our concepts, in terms of intramural sports, a possible Tournament may be "Soccer Fall 2014 League A", which would be a round robin Bracket of Matches between multiple Teams, such as "Next House", "Conner 4", and etc., and each Team would have a list of User, namely MIT affiliates.

In terms of dorm tournaments, a possible Tournament may be "Next House Ping Pong", which would consist of elimination Brackets "Main Bracket" and "Loser's Bracket". Each Team would start in the Main Bracket, and the first round losers would be filled into the Loser's Bracket. And each Team would simply be a Team of one User.

These relations are further expanded in our data model diagram:



(https://www.draw.io/#G0B2Mr49HAZeoOUmM4VFhTYkxxTGM)

API Specification (Work in progress)

All returned JSON objects have a status code, along with extra data. In cases of a 200 status code, the relevant documents are turned, while a non-200 status code will include a "message" field that indicates the error associated.

URL/Parameters	Method	Behavior	Returned JSON (Success)
USER			
/user/:id	GET	Gets user information	{ statusCode: 200, user: u }
/user/ { _id: un, password: pw }	POST	Creates new user, upon validation with MIT People database	{ statusCode: 200, user: u }

TEAM			
/team/:id	GET	Gets team information	{ statusCode: 200, team: u }
/team/	POST	Create a new team, with the sender as the only member	{ statusCode: 200, team: u }
/team/:team_id/ {add: [user_id: id], remove: [user_id: id]}	PUT	Adds and deletes specified users to the given team	{ statusCode: 200, team: u }
/team/:id	DELETE	Delete a team completely	{ statusCode: 200 }
MATCH			
/match/:id	GET	Gets match information	{ statusCode: 200, match: u }
/match	POST	Creates a new match	{ statusCode: 200, match: u }
/match/:id	PUT	Update match properties	{ statusCode: 200, match: u }
/match/:id	DELETE	Delete a match completely	{ statusCode: 200 }
/match/:id/outcome	PUT	Reports match outcome and update dependent matches accordingly	{ statusCode: 200, match: u, winner_match: u1, loser_match: u2 }
BRACKET			
/bracket/:id	GET	Gets bracket information	{ statusCode: 200, bracket: u }
/bracket/	POST	Creates a new bracket with specified type	{ statusCode: 200, bracket: u }
/bracket/:id	PUT	Updates bracket properties	{ statusCode: 200, bracket: u }
/bracket/:id	DELETE	Delete a bracket completely	{ statusCode: 200 }

TOURNAMENT			
/tournament/:id	GET	Gets tournament information	{ statusCode: 200, tournament: u }
/tournament/	POST	Creates a new tournament	{ statusCode: 200, tournament: u }
/tournament/:id	PUT	Updates tournament properties	{ statusCode: 200, tournament: u }
/tournament/:id	DELETE	Delete a tournament completely	{ statusCode: 200 }

III. Design Behavior

Security Concerns

Policy

Requirements

- Availability
  - The information stored and calculated in our site should be easily and readily accessible to all users.
  - Since a large problem of the existing solutions to tournament management is latency, we must alway provide data to users in a quick and coherent manner.
- Integrity
  - The information recorded for each match outcome should be an accurate representation of the actual outcome of that match
  - Statistics of players and tournaments must be reflective of aggregate match outcomes

Non-Requirements

- Privacy
  - All player statistics should be viewable from all other users of the system

Threat Model/What the attacker can do

- Modify tournament outcomes
- Send false match outcome data to the admin for approval
- Send bogus information to our system directly

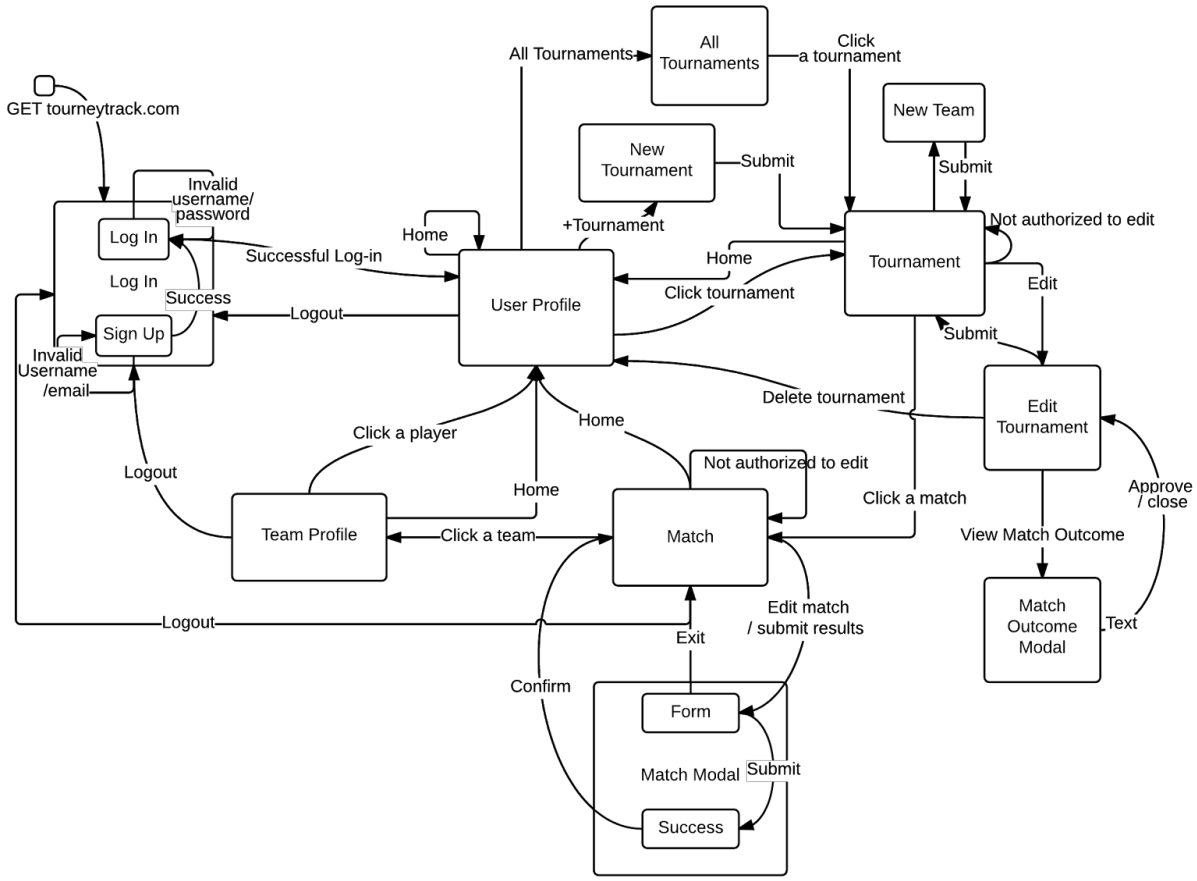
Mechanism/How attacks will be mitigated

- Using validator.js and AngularJS $sanitize service to sanitize inputs

- Use authentication to prevent unauthorized tournament modification
- Use a two-layered match data approval scheme, where the creators of a tournament have the final say regarding match outcomes and creators of tournaments must approve new teams that join and creators of teams must approve new team members

User Interface

Action Flow and State Transitions:

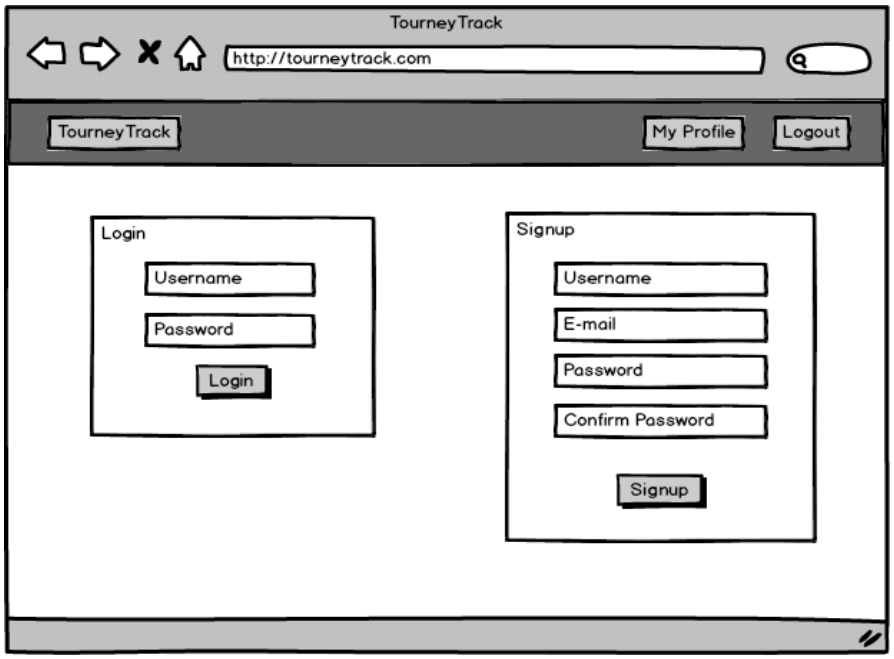


*On any view except the modal windows and Log In, clicking the TourneyTrack logo or “My Profile” will point to User Profile, and clicking Logout will point to Log In. Clicking either of those on the Log In page will redirect to itself.

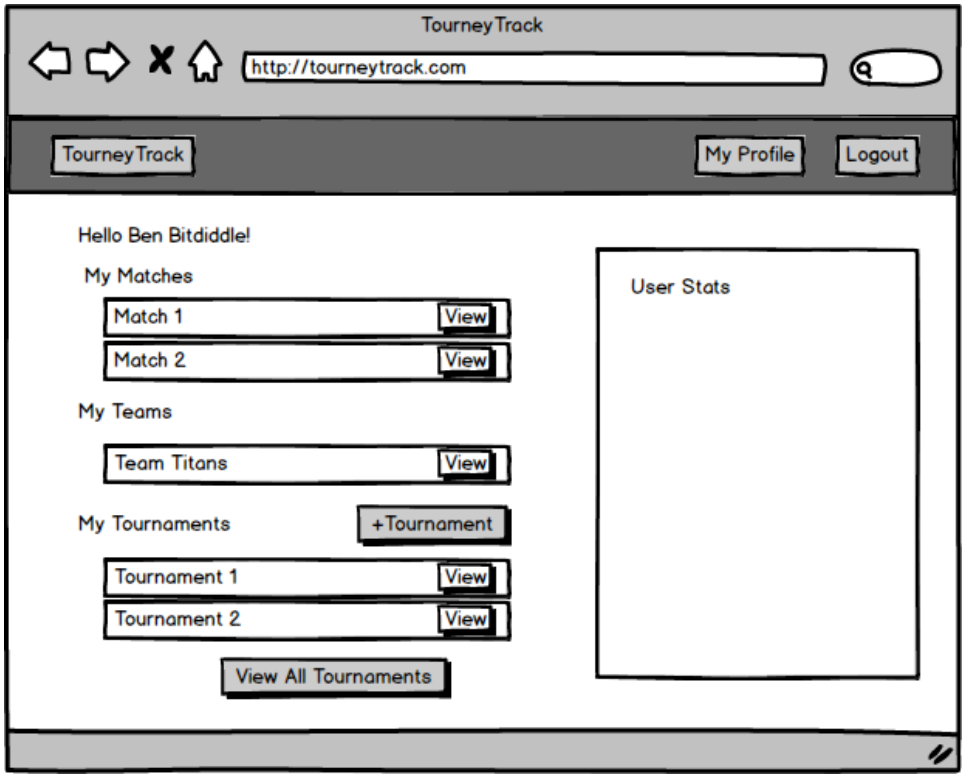
Wireframes (for MVP)

Below are the wireframes for our MVP:

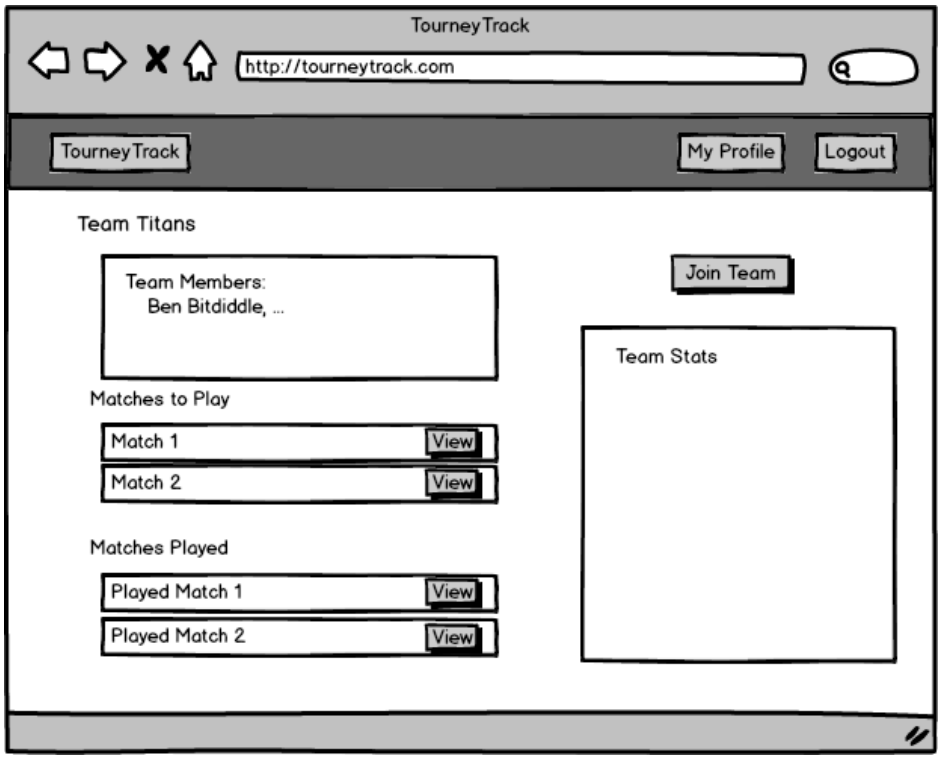
Login Page



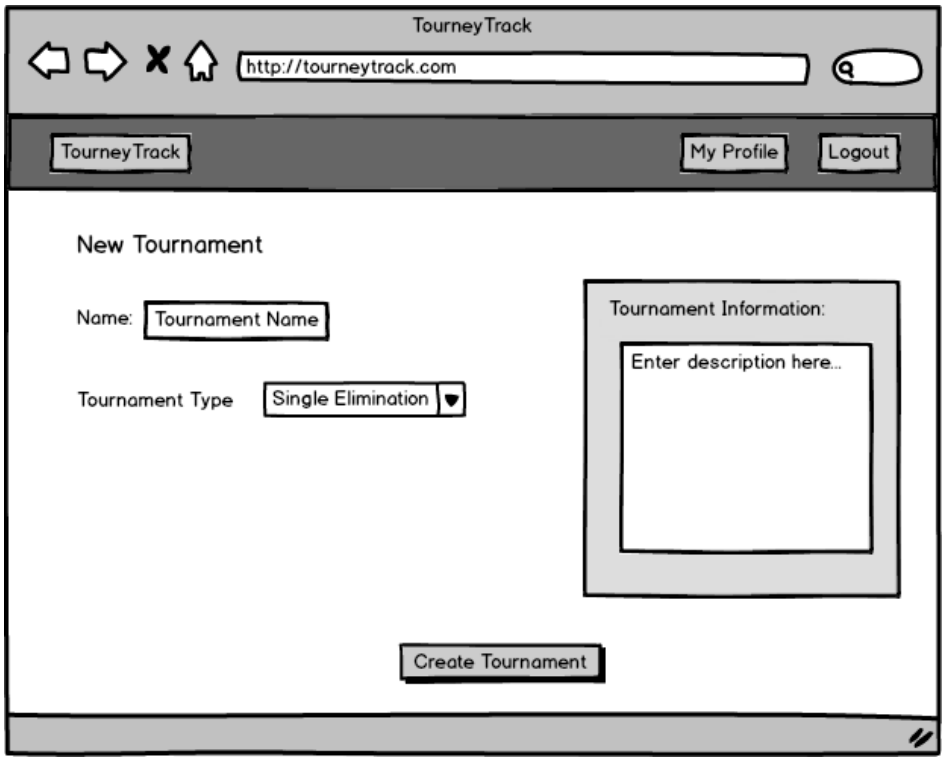
User Profile Page



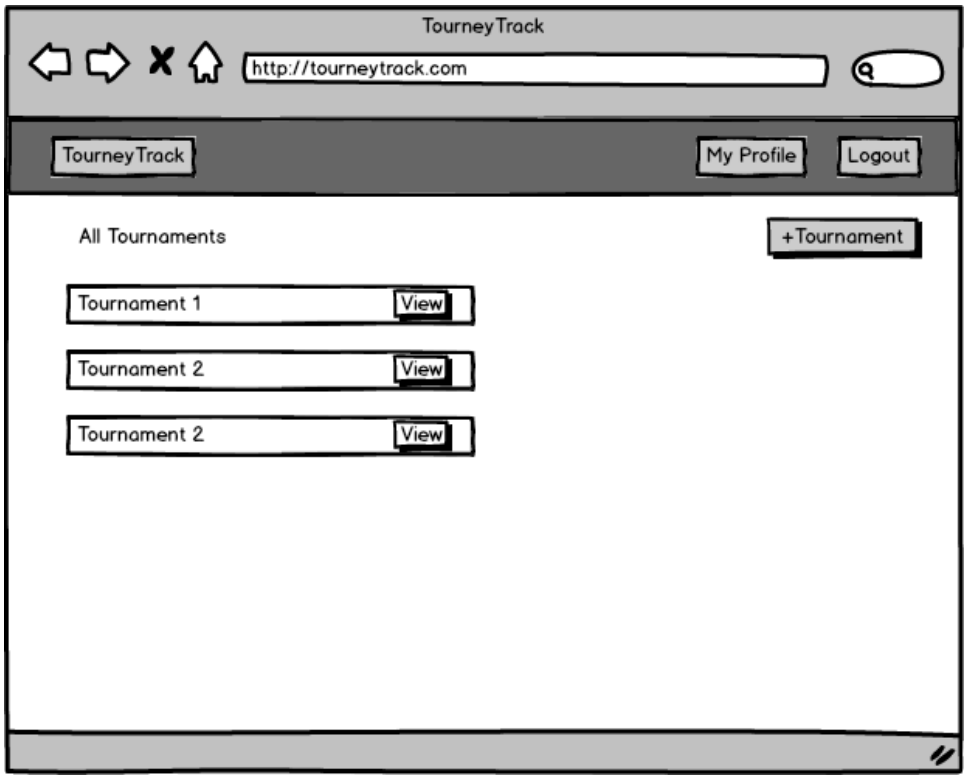
Team Profile Page



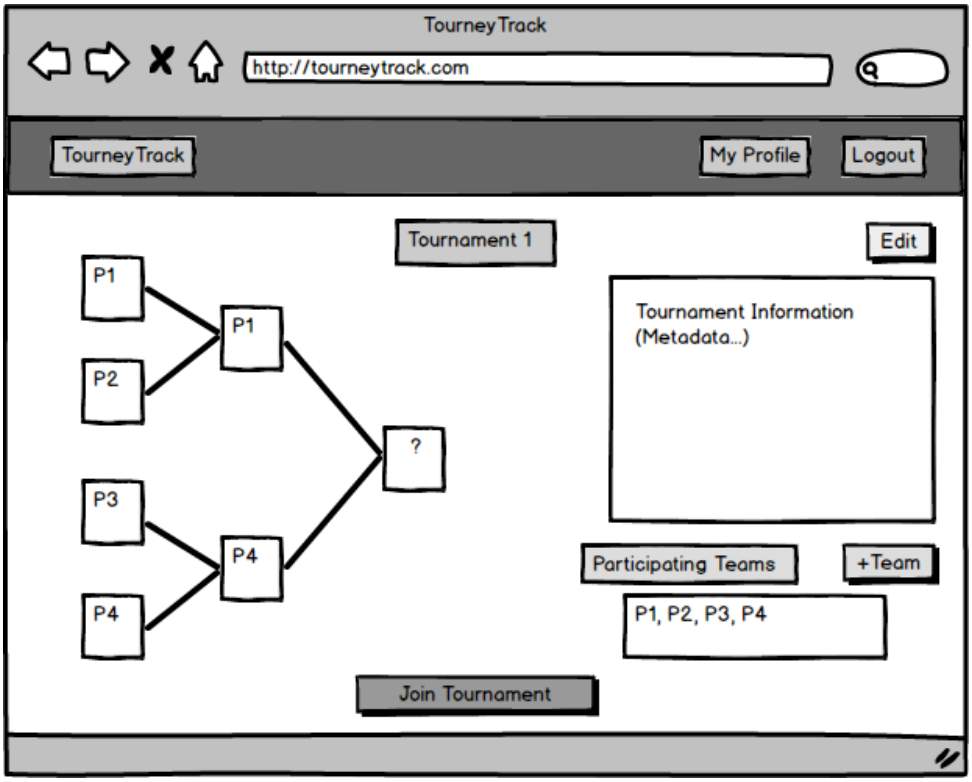
New Tournament Page



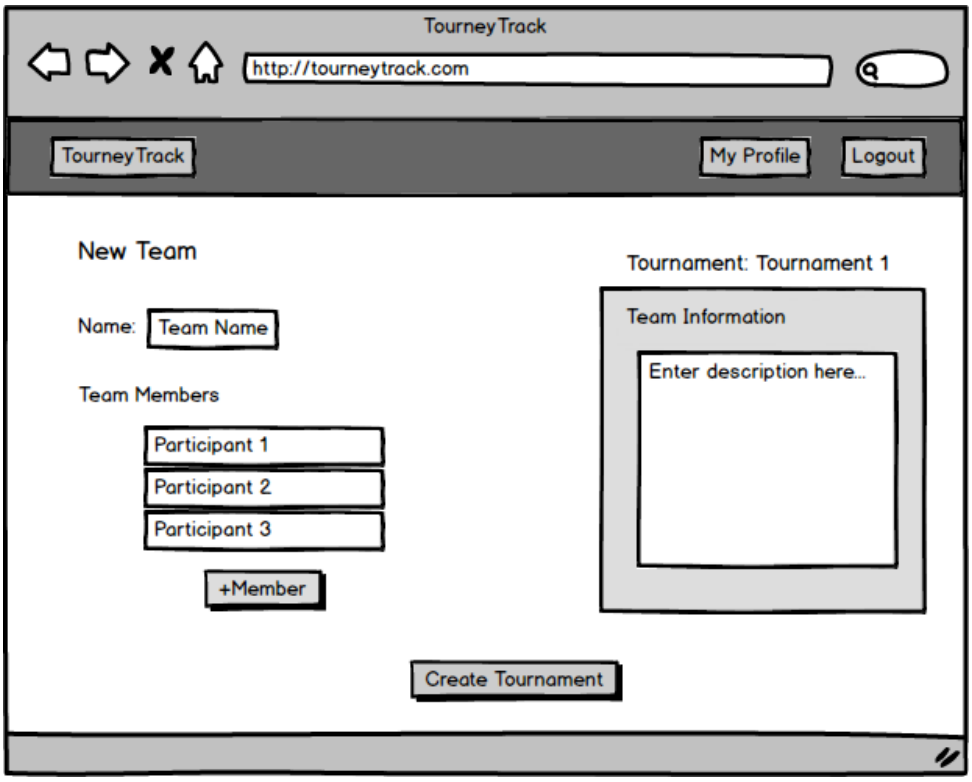
All Tournaments Page



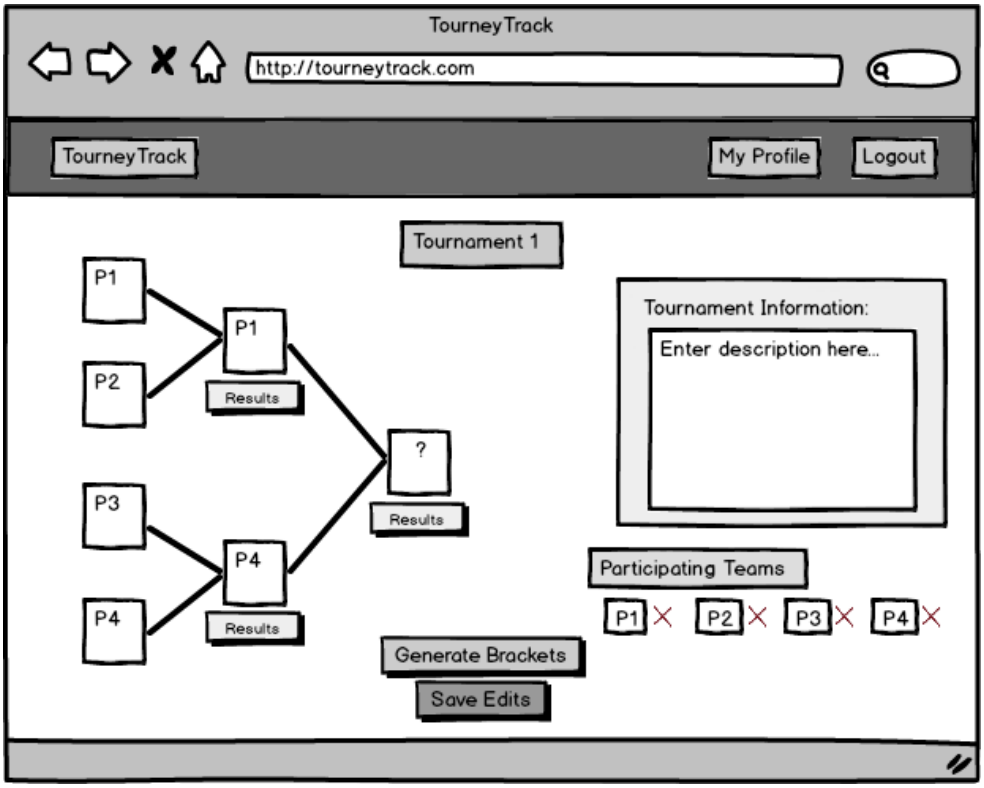
Tournament Page



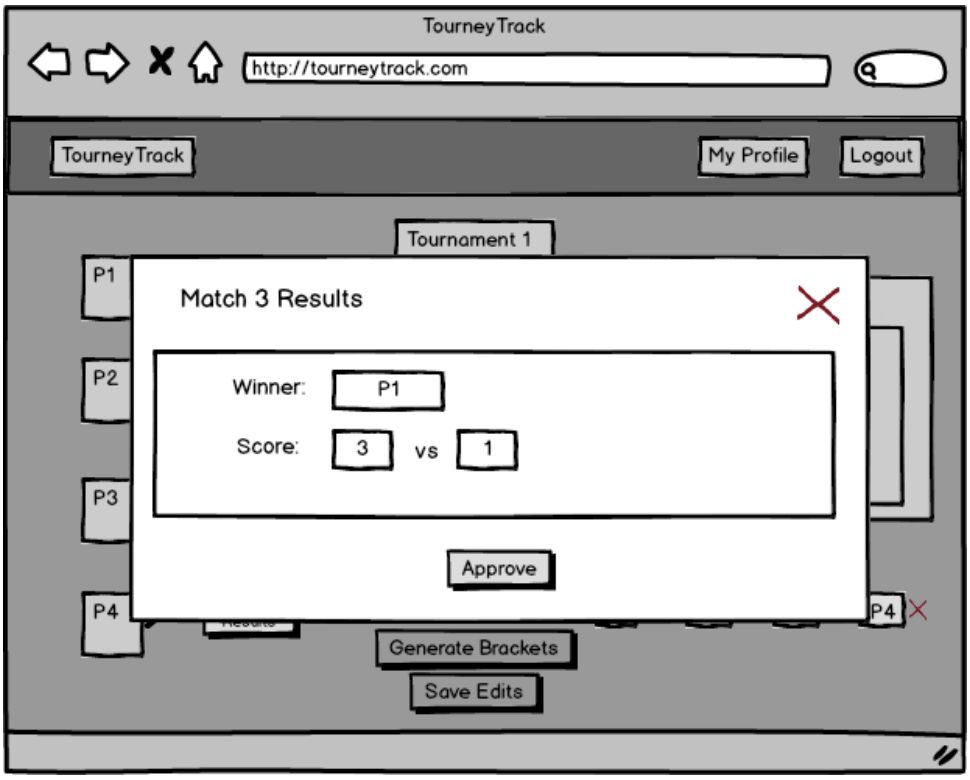
New Team Page



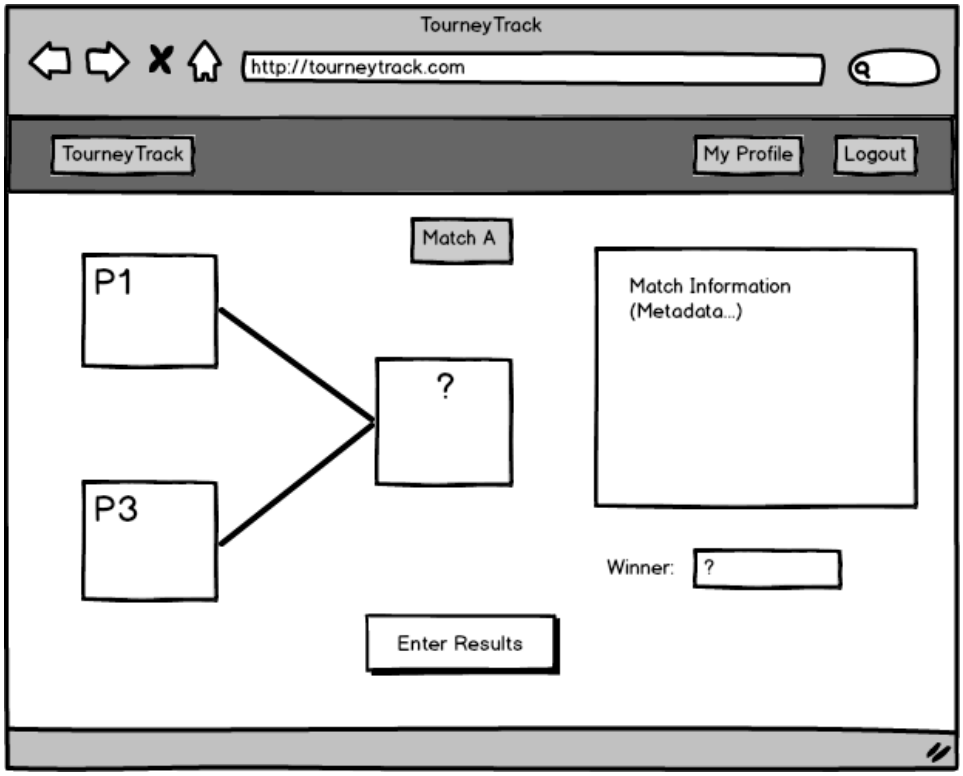
Edit Tournament Page



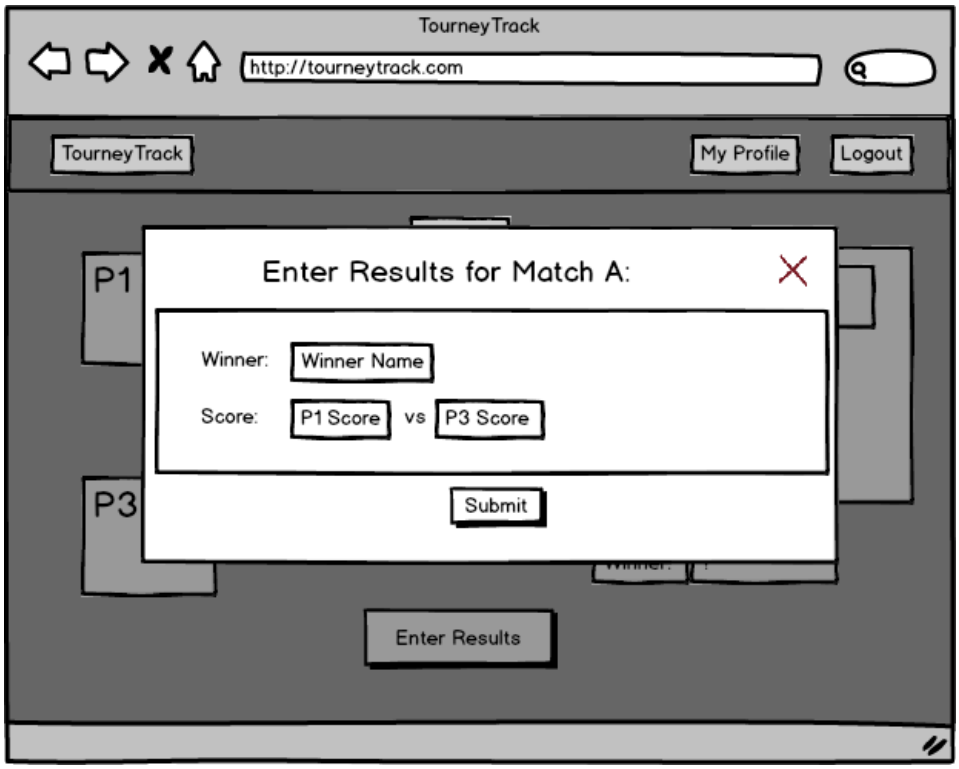
Tournament Modal Page (for result approval)



Match Page



Match Outcome Modal Page



Design Challenges

1. **How big should we scale our app / what is the scale of the tournaments we wish to serve?**
   One of the first questions we had to ask ourselves was who our target audience is that we want to serve. For example, do we want to make an app for just small dorm tournaments, or do we want to be able to support the World Cup?
   We had considered generalizing our app to fit all sized tournaments for any type of sport. Having the ability to support a wider audience seemed very appealing, but it also comes with a couple of subtle disadvantages. First of all, having a one-size-fits-all approach would increase the complexity of the app, which may discourage small-tournament administrators from using our app. If there are too many ways to customize the information inputted into our system, simple use cases quickly become unnecessarily difficult to do. Secondly, the more formal a tournament is, the more likely it is that they have access to a tournament-management app that is already tailored to their needs. This being the case, they would most likely not want to use our generalized app over their existing system.
   Since less-formal tournaments tend to lack sophisticated management tools, we chose to focus our efforts towards this audience. By cutting our scope, we are able to tailor this system to the needs of informal-tournament admins, allowing us to provide a

simpler, more intuitive experience.
We chose to limit the scope even further to cover only the MIT community because this allows us to gain a user-base more quickly, since fellow MIT students would trust us to use our app. Also, this allows us to tailor the experience to MIT students, which we can do because we are more aware of the problems faced by MIT students than all students in general. In the future, we could generalize the app to be for all informal tournaments.

2. **How do we keep track of match outcome data?**
   By supporting many different types of sporting events, we run into the question of how to keep track of all the match outcomes. Most sports have their own rules for determining the winners of a match, given the final scores. Many sports have certain statistics that would be necessary to keep track of, like number of goals scored, number of strikes, etc.
   One solution to this would be to support a fixed number of sports and tailoring the system to handle those sports in their specific ways. While this may be better for the user, it doesn't allow our system to grow. For example, if MIT created a new IM Quidditch league, we wouldn't be able to support them, unless we manually go into the system and create a new model for them.
   Our solution to the problem is to generalize the concept of match outcomes to "winning" and "losing." Every match will have a winner that moves on to some parent match, and a loser who may move on to some other parent match. This model fits most sports, since for most sports, a match has a winning and a losing team. The disadvantage of this setup is that we would lose the ability to track specific data regarding match outcomes, like points scored. To solve this, we would allow each tournament to specify additional statistics that can be stored with every match. These stats will not affect the outcome of the match, since winning and losing should ultimately decide this. But this allows statistics to be aggregated along the entire tournament.
3. **How do we protect the integrity of match data entered into our system?**
   One challenge we may face is that teams may not report accurate data into the system. Specifically, if a team lost, but they report to the system that they have won, this would have major implications to the rest of the tournament.
   We could just settle on trusting our users to be truthful and update the match outcome themselves. This approach would however ignore the concerns of data integrity.
   Another approach could be that the admin is responsible for inputting all match data. The problem with this is that often, the admin of a tournament is not present at all matches in the tournament, so they would not know the scores to input. At this point, the admin would ask the teams for the outcome, which reduces back down into the problems with the current way informal tournaments are managed.
   Our solution to this problem is to allow users to input match outcome, but to have the admin have the final say in the match outcome. Our system has to trust someone to input correct data, and the most logical person to trust is the person who created the tournament. Our system will allow teams to post their scores, which will be submitted

to the admin for approval. If both competing teams submit conflicting outcomes for a given match, the admin can contact the teams, make the proper changes, and submit the agreed-upon outcome. Since we want our system to focus on low-latency updates of team outcome data, we don't want this extra-step validation process to slow things down too much. To handle this, we will send notifications to both the opposing team and the admin when a team submits outcome data. This reminds the opposing team to report their outcome data, so the admin will have both outcome data statements to check against each other. The notification also serves as a reminder to the admin, so he/she can quickly approve the submission, allowing the update to surface in the system.

4. **Should Teams be within the context of a tournament?**
   One question we asked ourselves is that should a team be specific to a tournament? Or should a team be able to compete over multiple tournaments?
   We had originally considered teams to span outside of the context of a tournament. This seemed to be the natural option. In the example of dorm tournaments, we thought that a particular wing of a dorm might want to form a team, in which they can compete in many different dorm tournaments. This approach had many disadvantages. First off, not everyone in a given dorm wing might want to compete in a specific tournament. For example, if Bob is a member of the 5th floor ping pong team, he might not want to compete with the 5th floor in the basketball tournament. Also this approach would make it difficult to aggregate statistics across different tournaments, since different tournaments might have disjoint data that wouldn't make sense to aggregate across the two tournaments.
   Our solution was to have teams be specified under the context of a tournament. This makes more sense in terms of keeping track of team statistics, since the data entered for matches will be uniform over all matches in a given tournament. Also, this solution encourages groups of people to make separate teams for different events. So the 5th floor can have a separate team for ping pong and basketball. Another benefit to this solution is that it simplifies the process of adding participants to a tournament. By creating a team under the context of a tournament, a user is also registering this team as a participant of the tournament.
5. **Should Teams be immutable?**
   One decision we were faced with is whether or not players can be added to a team after a team is created.
   We wanted to allow players to be added after team creation, because this allows individual players to join a team themselves. The problem with this however is that players shouldn't be able to join a team mid-tournament, as this wouldn't be fair in the context of a competition. We could make teams immutable, but It would be a bit of a hassle for the team creator to have to input all members of the team at one time.
   Our solution is to combine these two notions. Players can join a team themselves up until the tournament starts. Once the tournament starts, players can no longer join the team.