Full Design

Team: Ham

Overview

Author: Victor Cheng

Brief description:

- Our platform aims to provide a resource for student groups to manage their groups through an announcement and automatic rostering system. The announcement system allows group exec to post announcements for all members to view. The automatic rostering system utilizes a variation of the stable matching algorithm to create matchings between new members and subteams of student groups, if applicable.

Key Purposes:

- Each semester, hundreds of students audition for these groups, and exec members spend hours manually placing people into different dances. Furthermore, during the semester, each choreographer communicates with their dances differently, some using FB groups, others using email chains. Especially for dancers who are in multiple dances, this can get confusing and important announcements may get lost. Our proposed project attempts to solve both these issues, by providing a platform that will have a more efficient, automated rostering system as well as an announcement portal for more consistent communication between members. The automated rostering solution will be applicable to any student group that has subteams. Groups that do not have sub teams can still take advantage of the announcement feature provided by our platform.

Description of Features:

- Display important information to student group members
  - After the sign up phase, dancers will be able to view announcements from the student groups and teams they are a part of
  - The announcements will by default be displayed in time order (most recent to least recent) and users can choose to filter by team or by type (if the announcement contains a video, etc)
  - Announcements will be seperated to group and team announcements so as to not let group announcements spam the team announcement feed
- Automatically match new users to teams based on multiple parameters
  - New members can sign up with their preferences, which includes the teams they want to join, in order of preference.
  - Teams through team leaders can also set their preferences, which include the people they want to join their team, in order of preference

    - Teams will be filled up based on the users preferences and team preferences. The team leads can choose to bypass the automated rostering and instead select members manually.

Deficiencies in existing solutions

- Facebook Messenger
    - Very spammy in nature and important information can get easily lost in all the messages
- Gmail
    - Message threads causes important emails to get lost in long conversations
- Slack
    - Does not have an automatic rostering solution
- Note: One of the benefits that the existing solutions provides is two-way communication. While that is important for much team communication, one product is used for something different. Teams and groups can use applications like Facebook Messenger and Slack for communication, but still use a different application for information (announcements) retrieval and storage. This separation of concerns is not only desirable, but it's also important in our product. We don't want to tackle too many issues at once, and if we feel like the core product (which is the rostering and the announcements) is successful, then we might expand to a faster method of communication on top of that. For our product that we're creating for this class however, we will just stick to our proposed scope.

Conceptual Design

Author: Victor Cheng

Stable Matching Algorithm

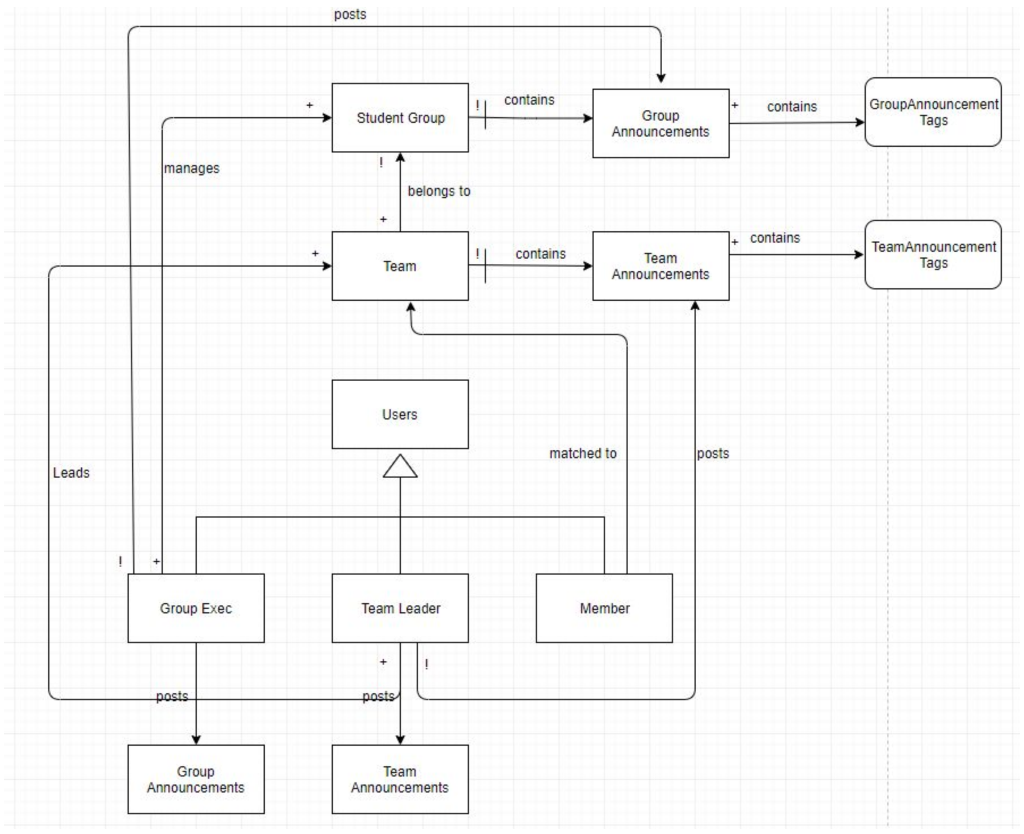
Referenced from: http://mpla.math.uoa.gr/media/theses/msc/Stathopoulos_G.pdf

- Initially, we are presented with the Hospital-Residents problem, where we have two sets of possibly unequal cardinality: the teams and the new members. New members can be matched to at most one hospital and each team has a quota for the maximum number of members that can be matched to it. Each agent (team and new members) has a strict preference list involving some, but possibly not all, of the members of the other set. We notice that an instance of HR where the two sets have equal cardinality and each team has quota of one, yields the Stable Matching problem.
- We define a matching as a partial mapping from the set of the new members, denoted *N*, to the set of the teams, denoted *T*, such that no team's quotas are exceeded and for every matched pair each of its members is acceptable to the other. A matching, *M*, is unstable if there is a pair (*n*, *t*) of a member and a team, not matched in *M* so that:
    - *n* and *t* are acceptable to each other
    - Either *n* is unmatched, or *n* prefers *t* to their assigned team
    - Either *t* has not exhausted its quota or *t* prefers *n* to at least one of its assigned members.

- We can reduce the Hospital-Residents problem to a Stable Matching problem by replacing each team *t* ∈ *T* by the set of teams {*t_1*,...,*t_q*}, all of which have the same preference list as *t*, and also replacing each appearance of team t in and member's *m* ∈ *M*, list by the sequence *t_1*,...,*1_q*. Each instance of the new team has quota of one. Then any stable matching of the reduced HR problem can be transformed into a stable matching for the original HR instance, and vice-versa.
- Example:
    - Member Preferences:
        - Alex: Apple, Banana
        - Billy: Banana, Apple
        - Charlie: Banana, Apple,
        - David: Apple, Banana
    - Team Preferences:
        - Apple (Quota = 2): Alex, Billy, Charlie, David
        - Banana (Quota = 2): David, Charlie, Billy, Alex
- We notice that we have sets of unequal cardinality, so we want to reduce this into a Stable Matching Problem using the method described above to get:
    - Member Preferences:
        - Alex: Apple_1, Apple_2, Banana_1, Banana_2
        - Billy: Banana_1, Banana_2, Apple_1, Apple_2
        - Charlie: Banana_1, Banana_2, Apple_1, Apple_2
        - David: Apple_1, Apple_2, Banana_1, Banana_2
    - Team Preferences:
        - Apple_1 (Quota = 1): Alex, Billy, Charlie, David
        - Apple_2 (Quota = 1): Alex, Billy, Charlie, David
        - Banana_1 (Quota = 1): David, Charlie, Billy, Alex
        - Banana_2 (Quota = 1): David, Charlie, Billy, Alex
- The newly instantiated HR matching results yields:
    - Apple_1: Alex
    - Apple_2: David
    - Banana_1: Charlie
    - Banana_2: Billy
- Converting the newly instantiated HR matching into the original HR matching yields:
    - Apple: Alex, Billy
    - Banana: Charlie: David
- We implemented the HR to SM reduction as well as the Stable Matching algorithm, so we are not relying on a third-party resource to perform the matching for us.

Announcements

- Purpose: Display important information to student group members
- Structure:



Behavior:

User actions:

- postTeamAnnouncement(team: Team, announcement: String, user: User, tags: String[])
  - Requires
    - user is a team leader
  - Effects
    - Posts announcement with the attached tags to the team's announcement page

- postGroupAnnouncement(group: Group, announcement: String, user: User, tags: String[])
  - Requires
    - user is part of group exec
  - Effects
    - Posts announcement with the attached tags to the group's announcement page

System actions:

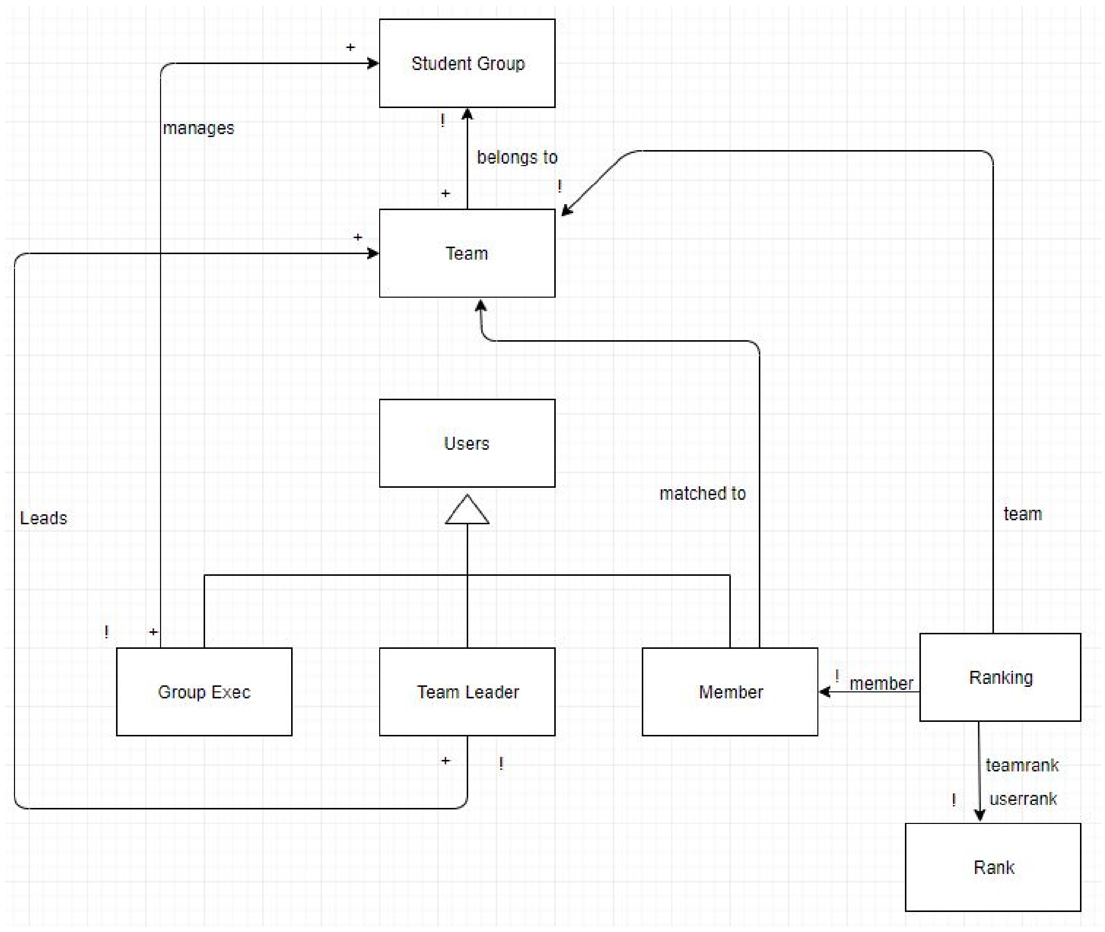
- displayTeamAnnouncement(team: Team, announcement: String, tags: Tags[])
  - Effects
    - Display announcement to team's announcement page. Filters by the tags provided. If no tags are provided, displays all announcements.

- displayGroupAnnouncement(group: Group, announcement: String, tags: Tags[])
  - Effects
    - Display announcement to group's announcement page. Filters by tags provided. If no tags are provided, displays all announcements.

- Operation Principle:
  Users that belong to group exec can post announcements to the group's announcement page. An announcement can have zero or more tags attached to it. Users that are a team leader can post announcements to the team's announcement page. Likewise team announcements can have zero or more tags attached to them. All team members can view the announcements on both the group and team announcement page.

Rostering

- Purpose: Automatically match new users to teams based on multiple parameters
- Structure:



Behavior:

User actions:

- signUp(username: String, password: String)
    - Requires
        - No user logged in.
        - The username is not taken
    - Effects
        - Creates a user with the given username and password

- createGroup(user: User, groupName: String)
    - Requires
        - groupName is not taken
    - Effects
        - Creates a Group with name groupName
        - Makes User the exec of the group

- joinGroup(user: User, group: Group)
    - Requires
        - User not in the exec of the group
    - Effects
        - Makes the user a member of the group

- addTeam(user: User, group: Group, teamName: String)
    - Requires
        - User in the group's exec
        - Not team with name == teamName exist in group
    - Effects
        - Creates a team with name teamName and adds it to the list of teams in group

- prefTeams(user: User, group: Group)
    - Requires
        - user in group
    - Effect
        - Creates rankings by the user of each of the teams in the group

- prefMembers(user: User, group: Group, team: Team)
    - Requires
        - user is teamLead of team and team in Group
    - Effect
        - Creates rankings by the team of each of the members of the group

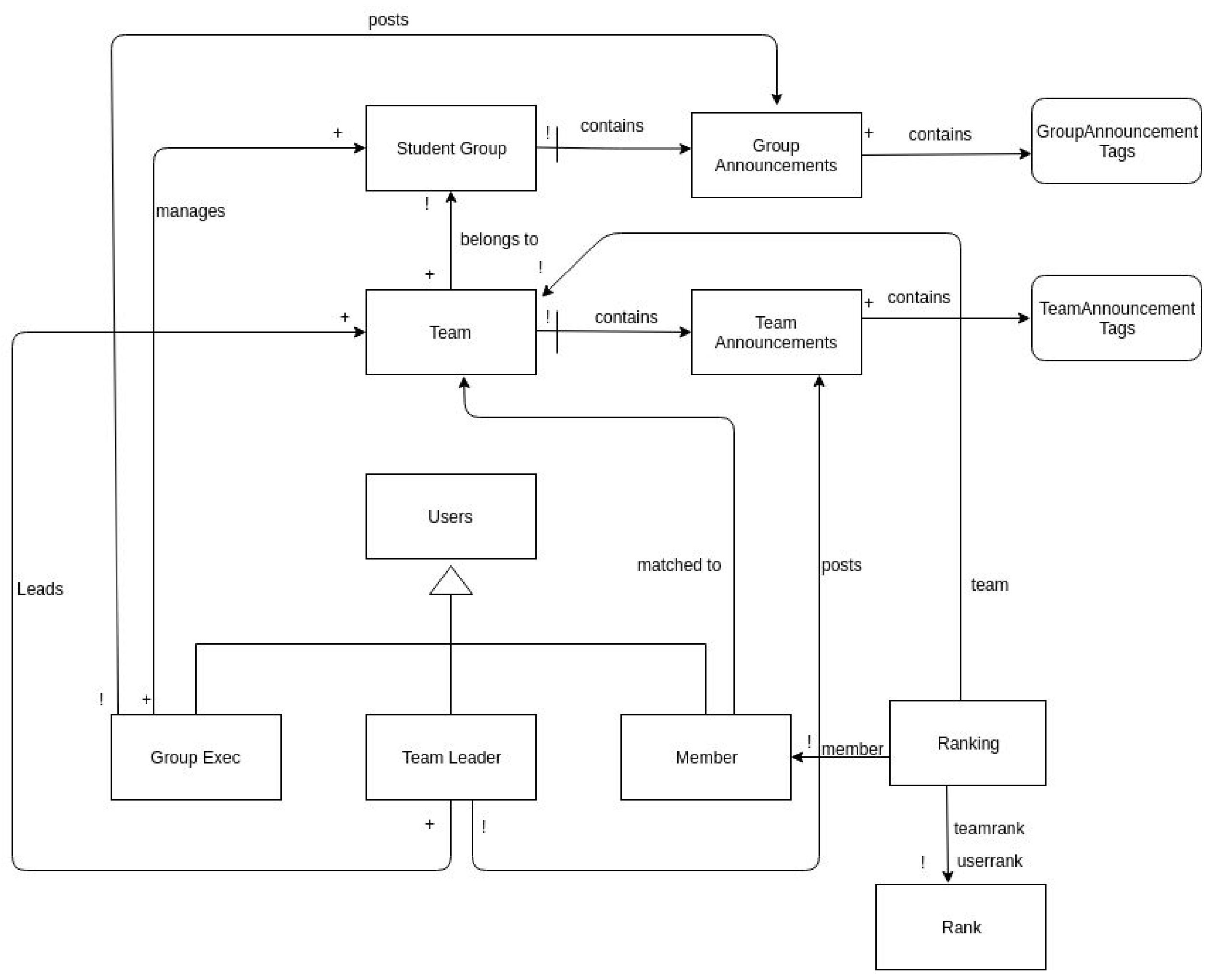
System actions:

    - match(group: Group, userRanking: UserRanking, teamRanking: TeamRanking)
        - Requires
            - userRanking and teamRanking
        - Effects
            - Utilizes matching algorithm to place users on teams using both userRanking and teamRanking

- Operation Principle:
  User can signUp for a group if they have not previously signed up. User will enter in preferences in form of ranking for teams they want to be on. Team admins can also enter preferences (also in form of ranking) of the people they want in their team. After all users have signed up, the system will use a matching algorithm to optimally assign each user to a team based on the user's preferences and team preferences.

Data Model

Author: Lesian



Schema Design

Author - Lesian

Steps followed to obtain schema

- Added attributes to data model
- Reversed one to many to make them point many to one
- Combined tables for many to many relationships.

```
CREATE TABLE IF NOT EXISTS users(
    userID INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(30) NOT NULL UNIQUE,
    password TEXT NOT NULL
```

```
   )


CREATE TABLE IF NOT EXISTS groups(
    groupID INT PRIMARY KEY AUTO_INCREMENT,
    groupname VARCHAR(30) NOT NULL UNIQUE
   )


CREATE TABLE IF NOT EXISTS teams(
    teamID INT PRIMARY KEY AUTO_INCREMENT,
    teamname VARCHAR(30) NOT NULL,
    teamquota INT,
    teamgroupID INT,
    FOREIGN KEY (teamgroupID) REFERENCES groups(groupID)
    ON UPDATE CASCADE
    ON DELETE CASCADE
   )


CREATE TABLE IF NOT EXISTS groupAdmins(
   groupID INT,
   adminID INT,
   FOREIGN KEY (groupID) REFERENCES groups(groupID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
   FOREIGN KEY (adminID) REFERENCES users(userID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
   PRIMARY KEY (groupID, adminID)
   )
CREATE TABLE IF NOT EXISTS teamLeads(
   teamID INT,
   leadID INT,
   FOREIGN KEY (teamID) REFERENCES teams(teamID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
   FOREIGN KEY (leadID) REFERENCES users(userID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
   PRIMARY KEY (teamID, leadID)
   )
```

```
CREATE TABLE IF NOT EXISTS groupMembers(
      groupID INT,
      memberID INT,
      FOREIGN KEY (groupID) REFERENCES groups(groupID)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
      FOREIGN KEY (memberID) REFERENCES users(userID),
         ON UPDATE CASCADE
         ON DELETE CASCADE,
      PRIMARY KEY (groupID, memberID)
      )

/* Many to many matched to relation table */
CREATE TABLE IF NOT EXISTS teamMembers(
        teamID INT,
        memberID INT,
      FOREIGN KEY (memberID) REFERENCES users(userID)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
      FOREIGN KEY (teamID) REFERENCES teams(teamID)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
      PRIMARY KEY (teamID, memberID)
      )

CREATE TABLE IF NOT EXISTS teamAnnouncements(
        announcerID INT,
        announcement TEXT NOT NULL,
        teamID INT,
      FOREIGN KEY (teamID, announcerID) REFERENCES teamLeads(teamID, leadID)
        ON UPDATE CASCADE
        ON DELETE CASCADE
      )

CREATE TABLE IF NOT EXISTS groupAnnouncements(
        announcerID INT,
        announcement TEXT NOT NULL,
        groupID INT,
        FOREIGN KEY (groupID, announcerID) REFERENCES groupAdmins(groupID, adminID)
        ON UPDATE CASCADE
        ON DELETE CASCADE
        PRIMARY KEY (groupID, announcementID)
      )
```

```
CREATE TABLE IF NOT EXISTS rankings(
	teamID INT,
	userID INT,
	userRankingByTeam INT,
	teamRankingByUser INT,
	FOREIGN KEY (teamID) REFERENCES teams(teamID)
	ON UPDATE CASCADE
	ON DELETE CASCADE,
	FOREIGN KEY (userID) REFERENCES users(userID)
	ON UPDATE CASCADE
	ON DELETE CASCADE
    )

CREATE TABLE IF NOT EXIST groupAnnouncementsTags(
	groupID INT,
	announcementID INT,
	announcementTag VARCHAR(30),
	FOREIGN KEY (groupID, announcementID) REFERENCES
groupAnnouncements(groupID, announcementID)
	ON UPDATE CASCADE
	ON DELETE CASCADE
    )

CREATE TABLE IF NOT EXIST teamAnnouncementsTags(
	teamID INT,
	announcementID INT,
	announcementTag VARCHAR(30),
	FOREIGN KEY (teamID, announcementID) REFERENCES teamAnnouncements(teamID,
announcementID)
	ON UPDATE CASCADE
	ON DELETE CASCADE
    )
```

Security Concerns

Author: Faaya Fulas

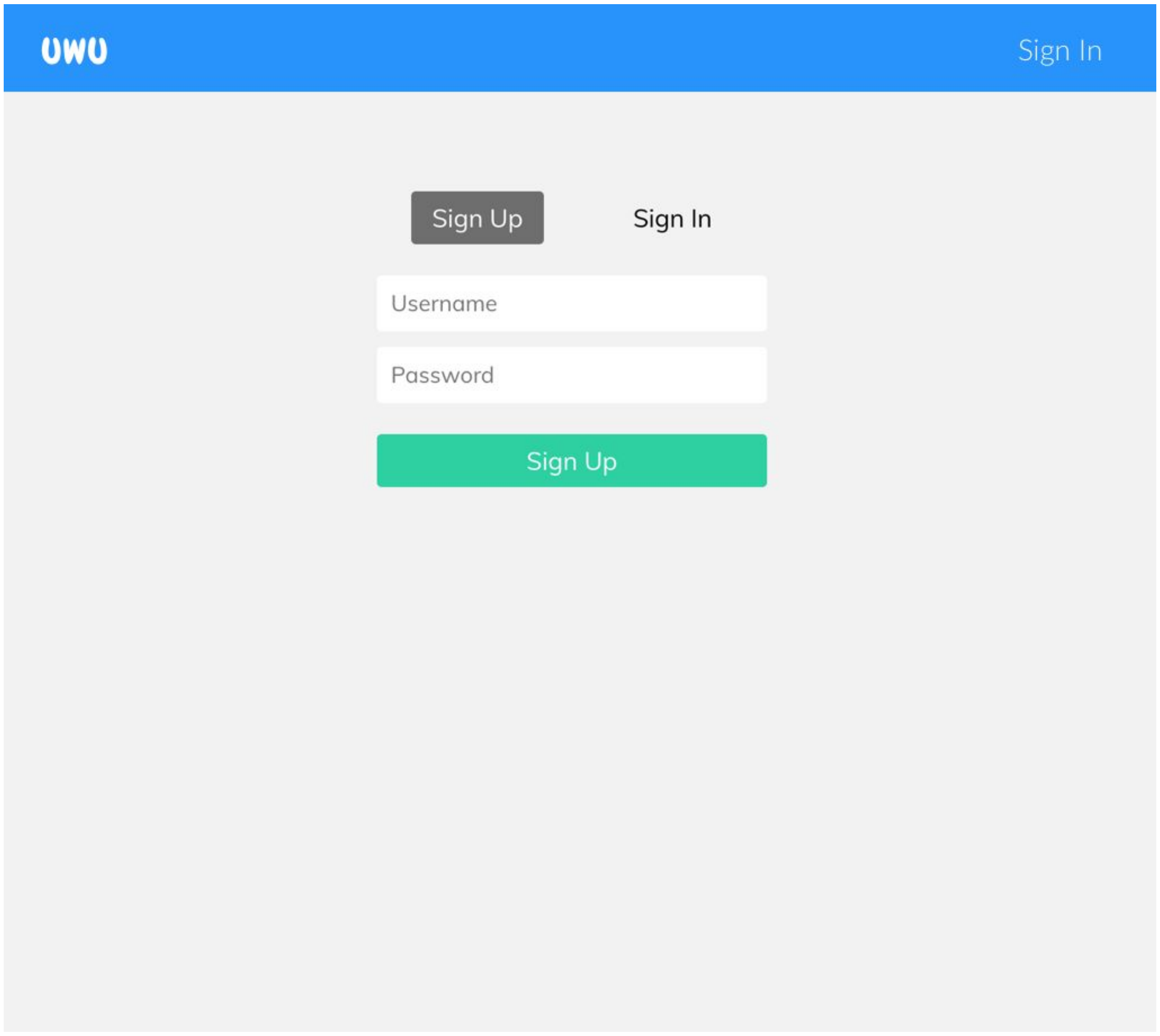
- Key security requirements:
    - The system will have a password protected sign-in page to make sure that users identify and authenticate themselves before accessing restricted resources.
    - During the signup phase, a user can opt to either join an existing student group, or create a new student group. The creator of a student group automatically becomes a Group Exec, and has the power to grant the Group Exec privilege to one or more members of the group.
    - The Team Lead privilege will be granted to one or more members of the student group by one of the Group Execs.
    - Users with similar privileges cannot override each other's actions.
    - Some system features will only be visible to users with special privileges (Group Execs and Team Leads).For example, the 'Create Teams' feature will only be visible to a user with a Group Exec privilege.

- Mitigations to protect the system against standard web attacks:
    - User input supplied via forms will be sanitized to prevent XSS and SQL injection attacks.
    - All forms will include CSRF tokens to prevent CSRF attacks.
    - The system will only store hashed passwords to prevent unauthorized logins even if the 'users' table in the database is compromised.
    - Data integrity will be maintained across tables in the database via appropriate foreign key constraints.

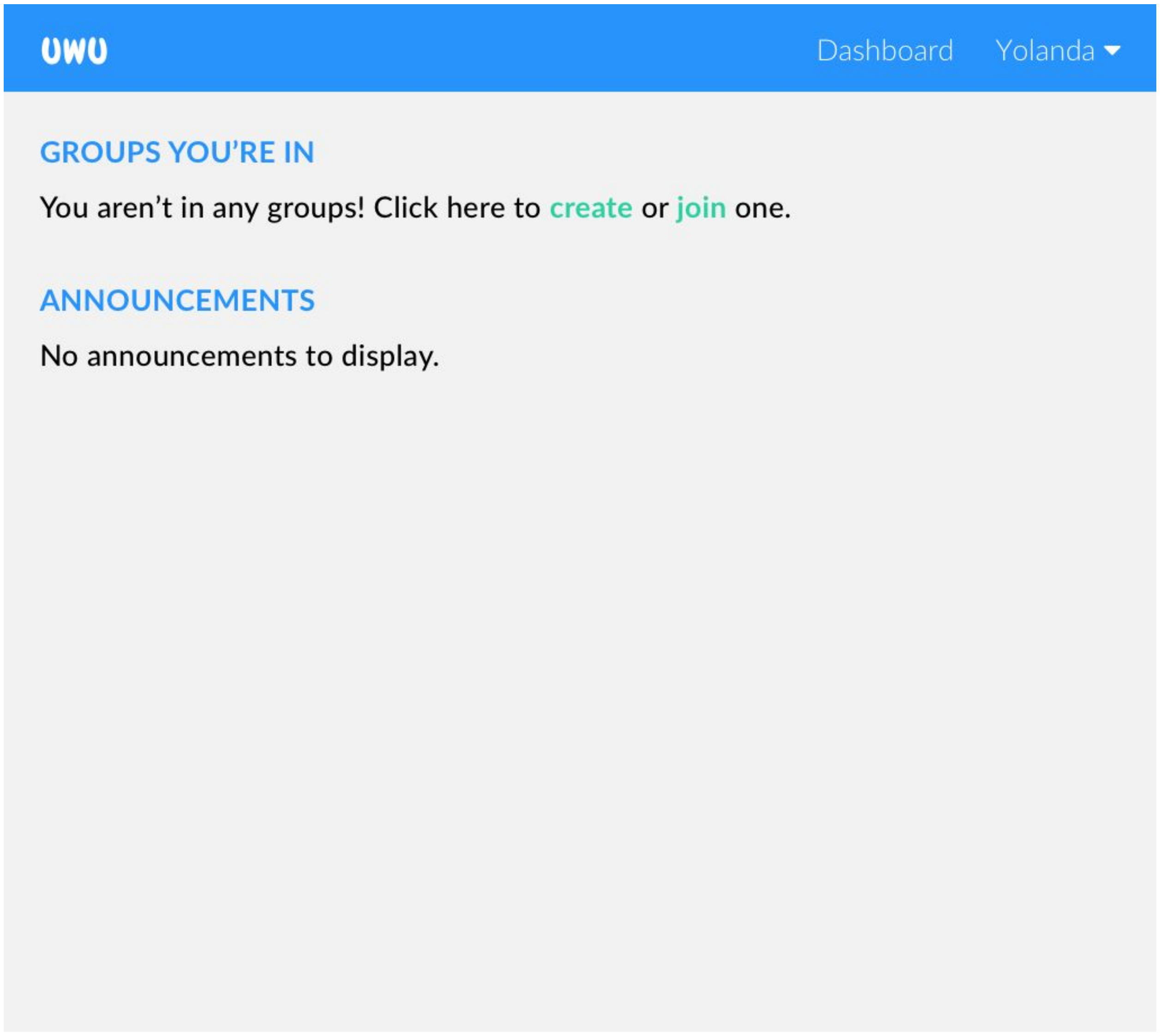
Wireframes

Author: Yolanda Zhou

Users initially start on the sign up / sign in page (presumably after a generic home page with information about the app)



Upon signing up, users have the option to join or create a group



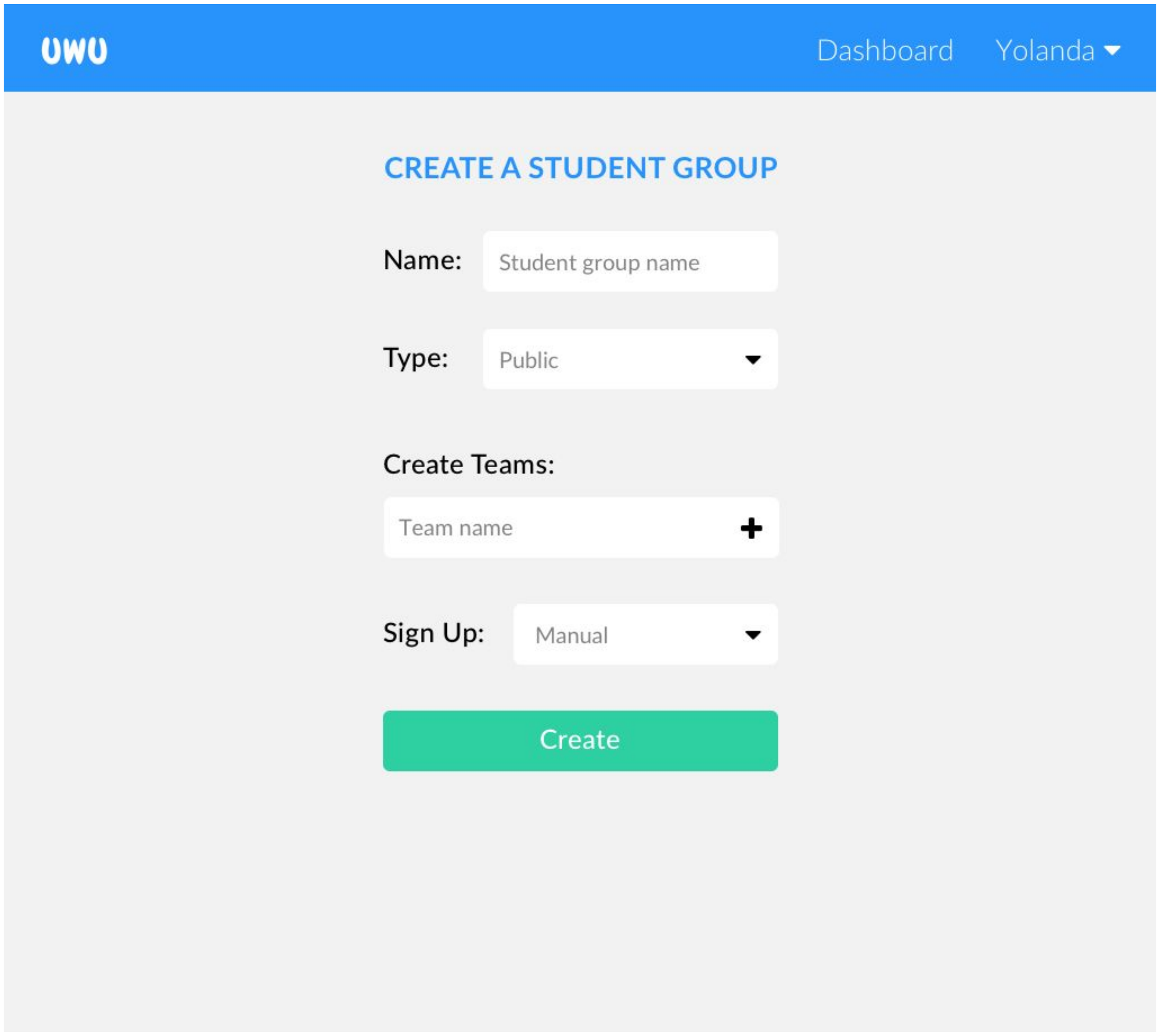
GROUPS YOU'RE IN

You aren't in any groups! Click here to create or join one.

ANNOUNCEMENTS

No announcements to display.

The rough page for creating a student group would look like this:


UWU Dashboard Yolanda

CREATE A STUDENT GROUP

Name: Student group name

Type: Public

Create Teams:

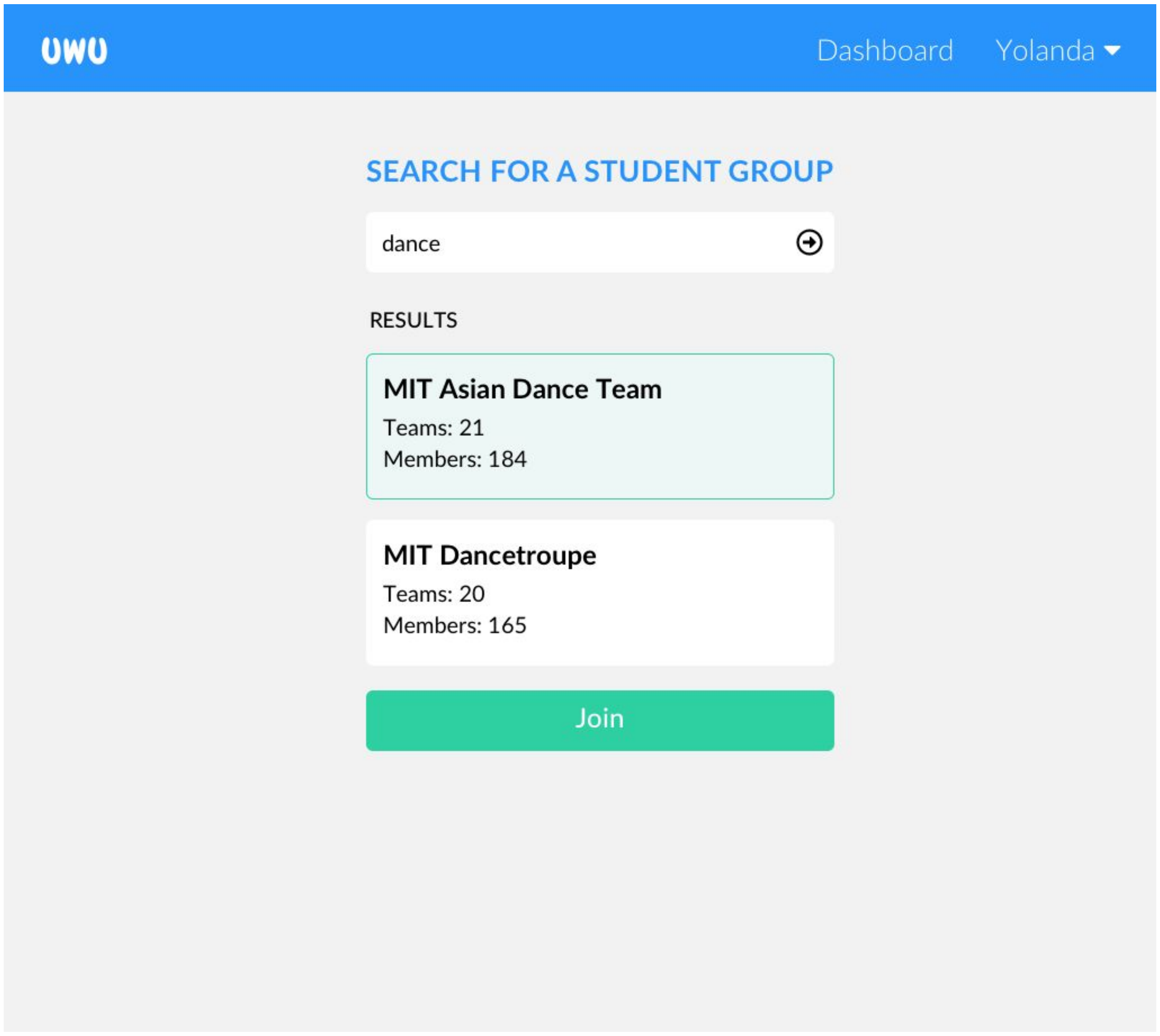
Team name

Sign Up: Manual

Create

After creating the group, there will be a page to manage it (create new teams, manage the team leaders, manage the members of the group, invite team leaders, change the status of the student group sign up phase).

If the user chooses to join a student group, they will see a screen like this:



SEARCH FOR A STUDENT GROUP

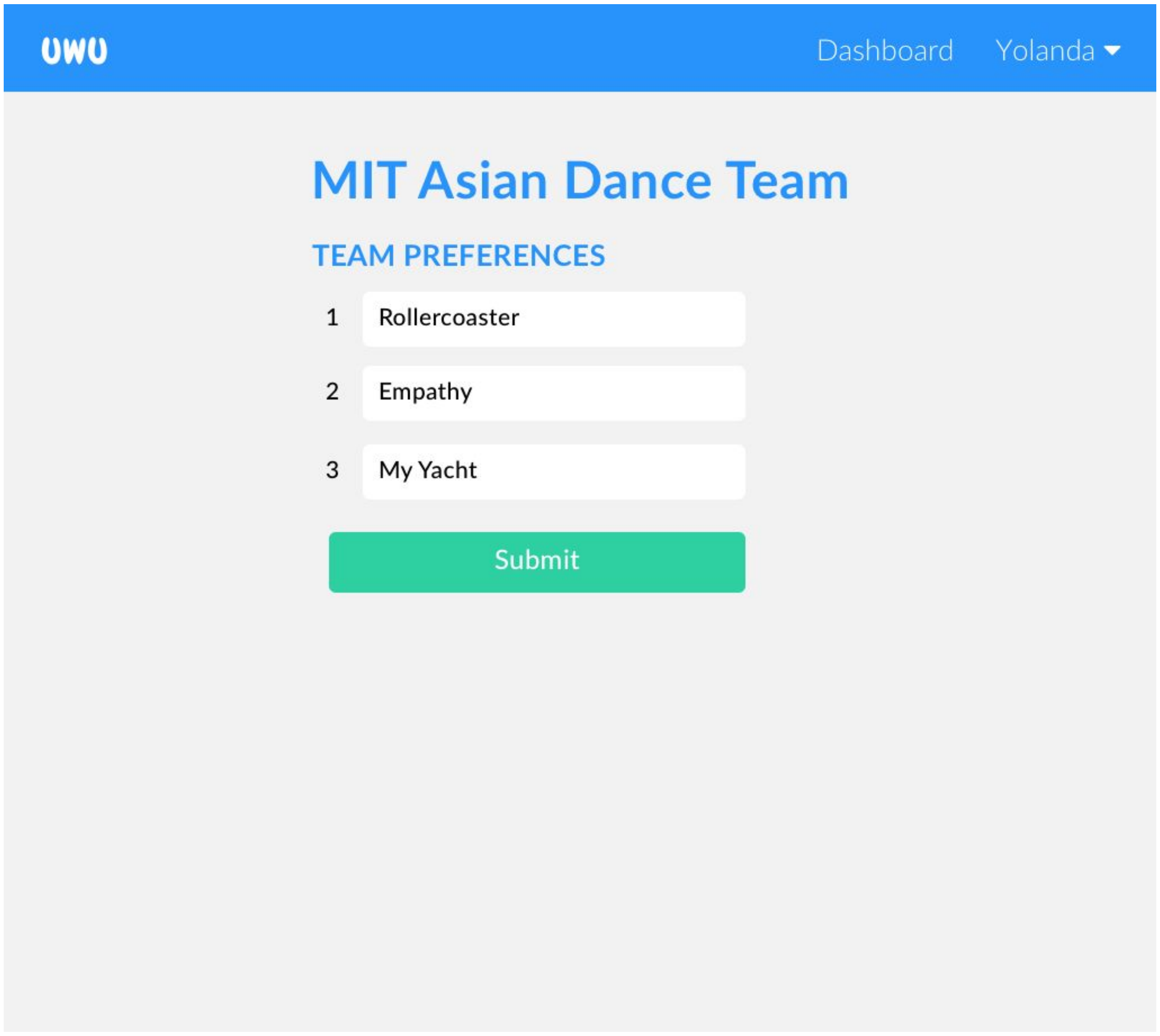
dance

RESULTS

MIT Asian Dance Team
Teams: 21
Members: 184

MIT Dancetroupe
Teams: 20
Members: 165

After joining the student group, if the student group is in its sign up phase, the user will be able to pref teams.

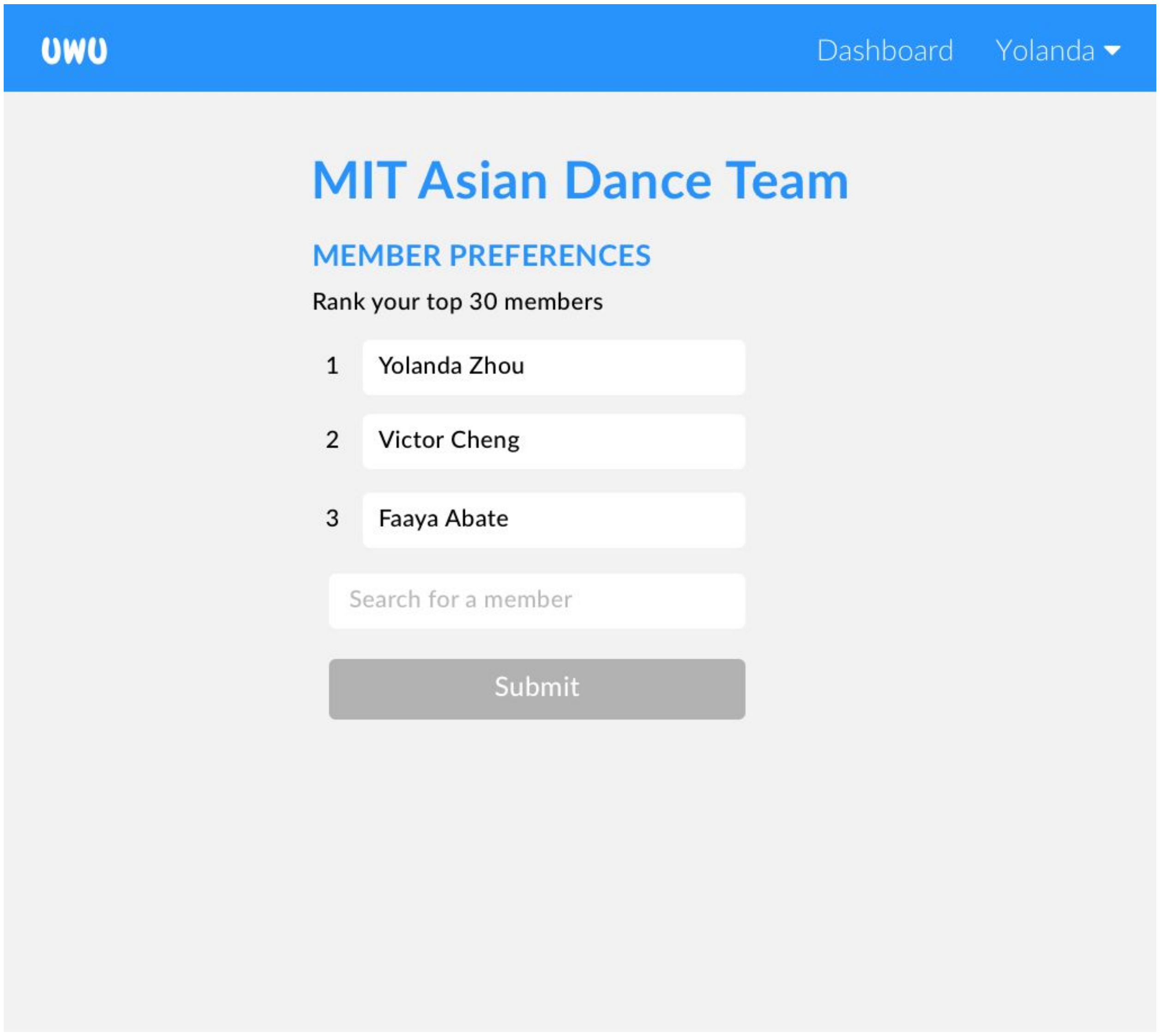


MIT Asian Dance Team

TEAM PREFERENCES

1	Rollercoaster
2	Empathy
3	My Yacht

Similarly, team leaders can pref members.



MIT Asian Dance Team

MEMBER PREFERENCES

Rank your top 30 members

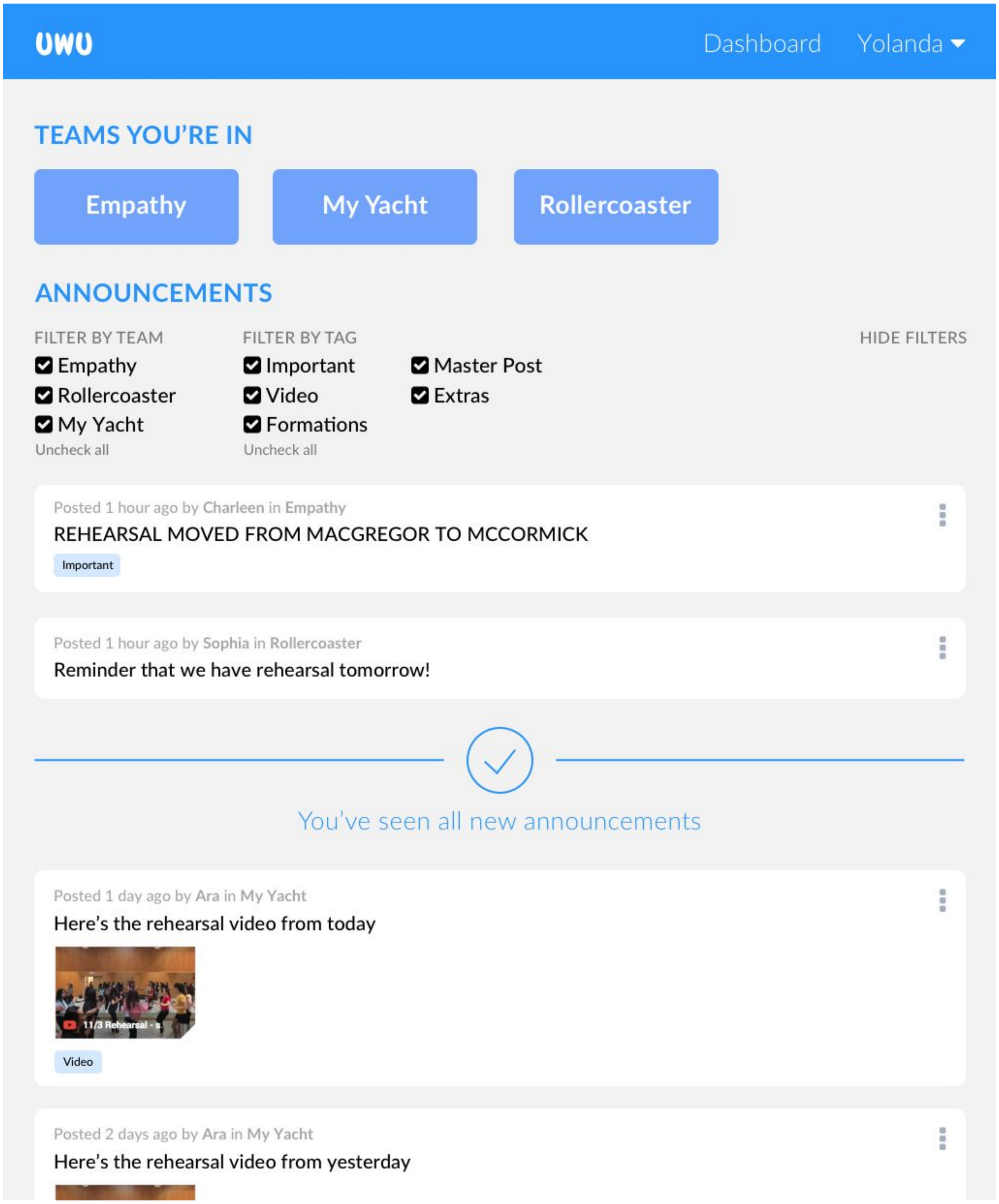
1 Yolanda Zhou

2 Victor Cheng

3 Faaya Abate

After these preferences are done, the admin of the student group will run stable matching or some other kind of matching process from students to groups. Then, the app functions more as an announcements app to display relevant information to users.

This is what a typical member's view might look like:



UWU Dashboard Yolanda

TEAMS YOU'RE IN

Empathy | My Yacht | Rollercoaster

ANNOUNCEMENTS

FILTER BY TEAM
- ☑ Empathy
- ☑ Rollercoaster
- ☑ My Yacht

Uncheck all

FILTER BY TAG
- ☑ Important
- ☑ Video
- ☑ Formations
- ☑ Master Post
- ☑ Extras

Uncheck all

HIDE FILTERS

Posted 1 hour ago by Charleen in Empathy
REHEARSAL MOVED FROM MACGREGOR TO MCCORMICK
Important

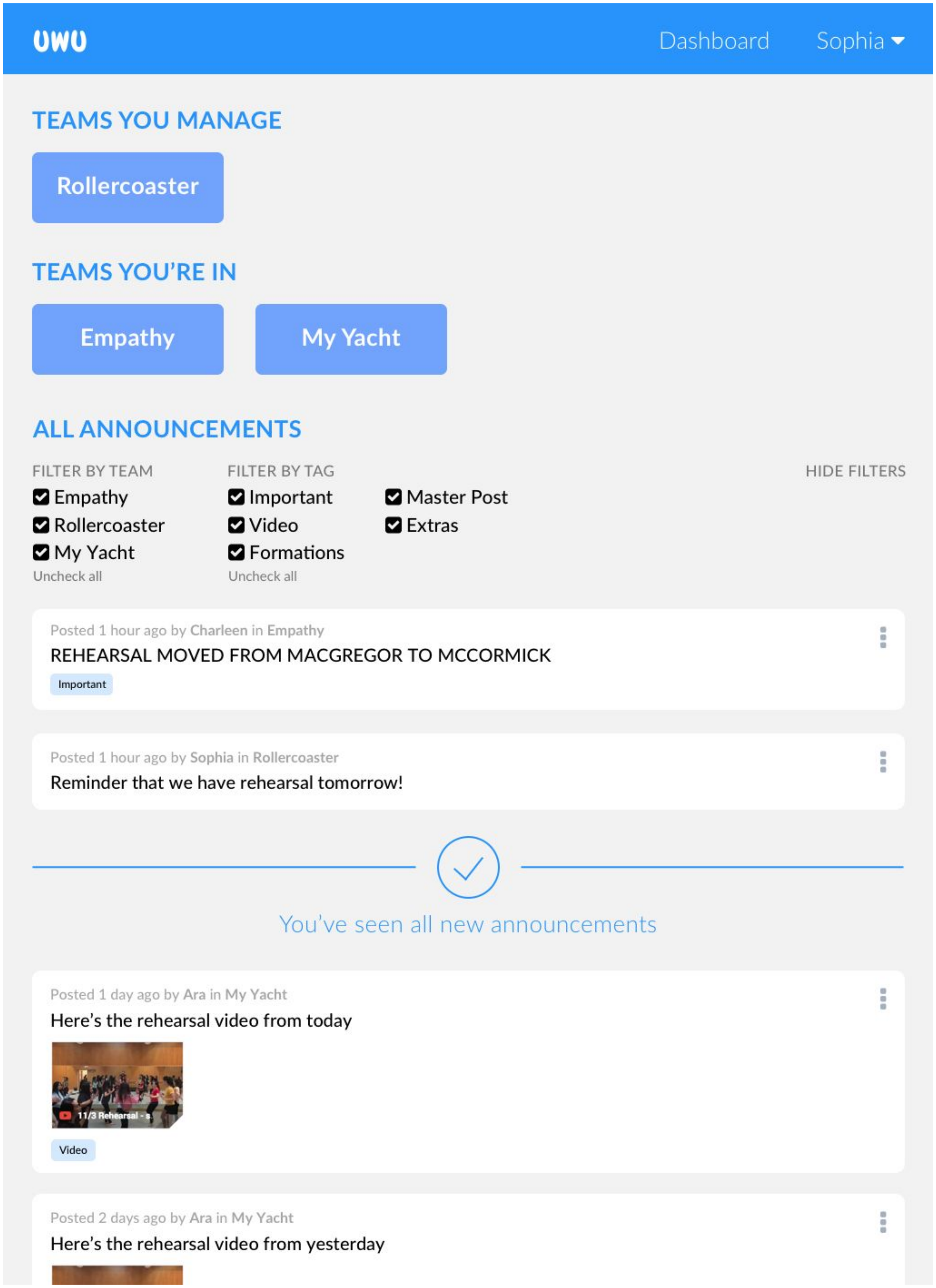
Posted 1 hour ago by Sophia in Rollercoaster
Reminder that we have rehearsal tomorrow!

You've seen all new announcements

Posted 1 day ago by Ara in My Yacht
Here's the rehearsal video from today
11/3 Rehearsal - s
Video

Posted 2 days ago by Ara in My Yacht
Here's the rehearsal video from yesterday

An team leader will get a similar page but with the option to view the teams they manage.



UWU   Dashboard   Sophia ▾

TEAMS YOU MANAGE

Rollercoaster

TEAMS YOU'RE IN

Empathy   My Yacht

ALL ANNOUNCEMENTS

FILTER BY TEAM	FILTER BY TAG		HIDE FILTERS
☑ Empathy	☑ Important	☑ Master Post	
☑ Rollercoaster	☑ Video	☑ Extras	
☑ My Yacht	☑ Formations		
Uncheck all	Uncheck all		

Posted 1 hour ago by Charleen in Empathy
REHEARSAL MOVED FROM MACGREGOR TO MCCORMICK
Important

Posted 1 hour ago by Sophia in Rollercoaster
Reminder that we have rehearsal tomorrow!

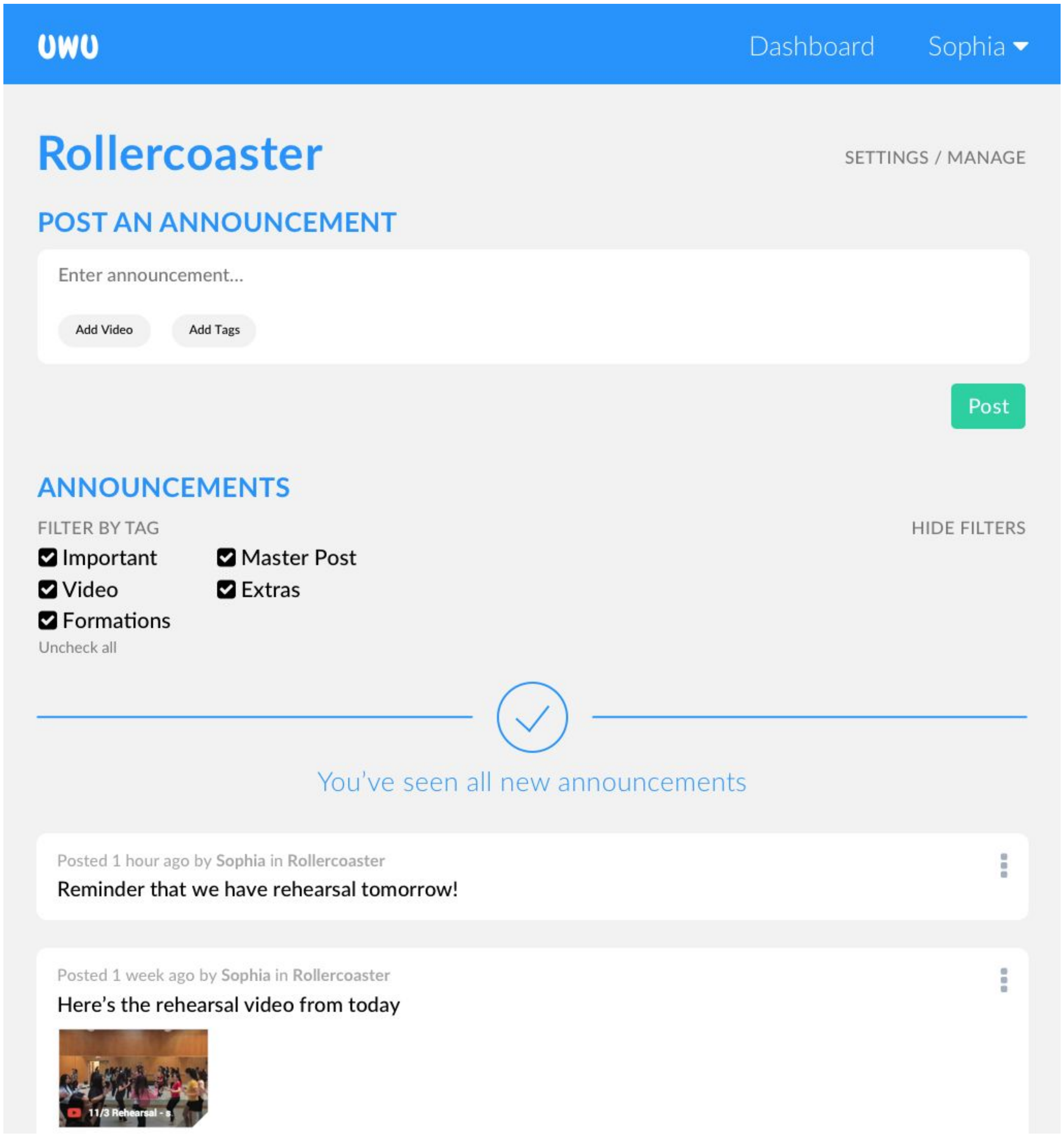
✓
You've seen all new announcements

Posted 1 day ago by Ara in My Yacht
Here's the rehearsal video from today

Video

Posted 2 days ago by Ara in My Yacht
Here's the rehearsal video from yesterday

Lastly, the team leader can go on their team page to post announcements or access other settings such as the team roster to view or change



Rollercoaster

POST AN ANNOUNCEMENT

ANNOUNCEMENTS

- [✓] Important
- [✓] Video
- [✓] Formations
- [✓] Master Post
- [✓] Extras

You've seen all new announcements

Posted 1 hour ago by Sophia in Rollercoaster
Reminder that we have rehearsal tomorrow!

Posted 1 week ago by Sophia in Rollercoaster
Here's the rehearsal video from today

Design Commentary

Author: Faaya Fulas

- During the conceptual design phase for the Announcements concept, we experimented with the idea of having a single set for both group and team announcements. However, we realized that we will need to differentiate between announcements targeted to the entire student group and announcements relevant to specific teams to avoid spamming users with team announcements irrelevant to them . Ultimately, we decided to have seperate sets for each announcement types.

- During the conceptual design phase for the Rostering concept, we had extensive discussions about how the system would automatically match teams with unassigned users. We initially came up with the idea of having a 'Qualifications' set that would contain a list of attributes the user 'has' and the team 'requires'. This proved to be problematic during the schema design phase since the list of attributes that the Qualifications table should contain would be different depending on the type of student groups.

- Ultimately, we came up with the idea of Ranking - each user will rank the team he/she want to join. Likewise, each team will rank the users who requested to join the team. After ranking is done, a stable matching algorithm will assign users to teams. This design choice not only resolved the issue of having inconsistent attributes in the Qualifications table , it also made the matching process more straightforward since users and teams can be effectively sorted and filtered based on the rankings.

- The revised design document is modified to addresses the issues raised by our mentor in the Full Design feedback. We have modified our data model and schema to support announcement tagging, and included a section explaining the stable matching algorithm which is the essence of the Rostering concept.