

P(arty)Set

6.170 Project 4 Problem Analysis

Jessica Andersen, Carolyn Holz, Nick Mohr, Hyungie Sung

New Stuff Highlighted in green

PartySet is a service that helps students in the same classes come together to work on their problem sets. These meetings, called *pset parties*, will be shown to relevant students who might wish to join in. PartySet will reduce the efforts needed to meet fellow students to pset with one another.

Current web applications that try to address the same concerns are often unused. These services require a heavy log in process and setting up a profile is often deemed not worth the amount of time and effort it requires. Instead, students find it easier (although more limited) to set up pset groups by word of mouth.

Purposes

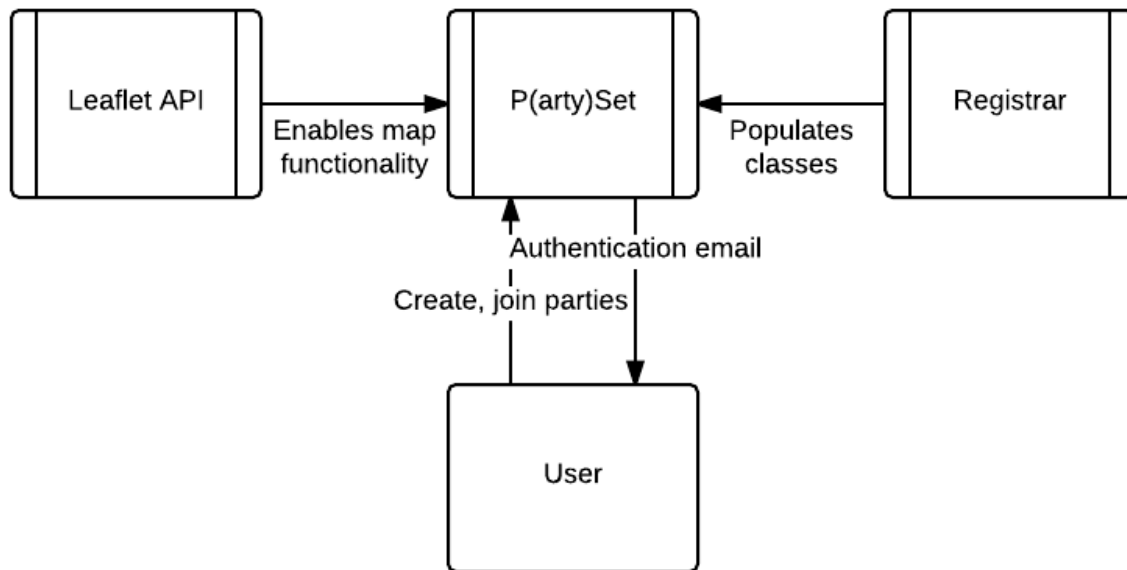
- **Connect people in the same MIT classes to pset together.** Students have missed many opportunities to expand their academic and social circles, not getting the maximum benefits from the environment around them.
- **Help people physically find one another.** It is difficult to plan out where to meet on a campus as big as MIT's. A map with updated locations of events will minimize these efforts any student has to make.
- **Increase effectiveness of MIT students.** Students have wasted much time and efforts in working on psets alone. A combined effort will result in greater efficiency and effectiveness in how each student spends their time.

Concepts

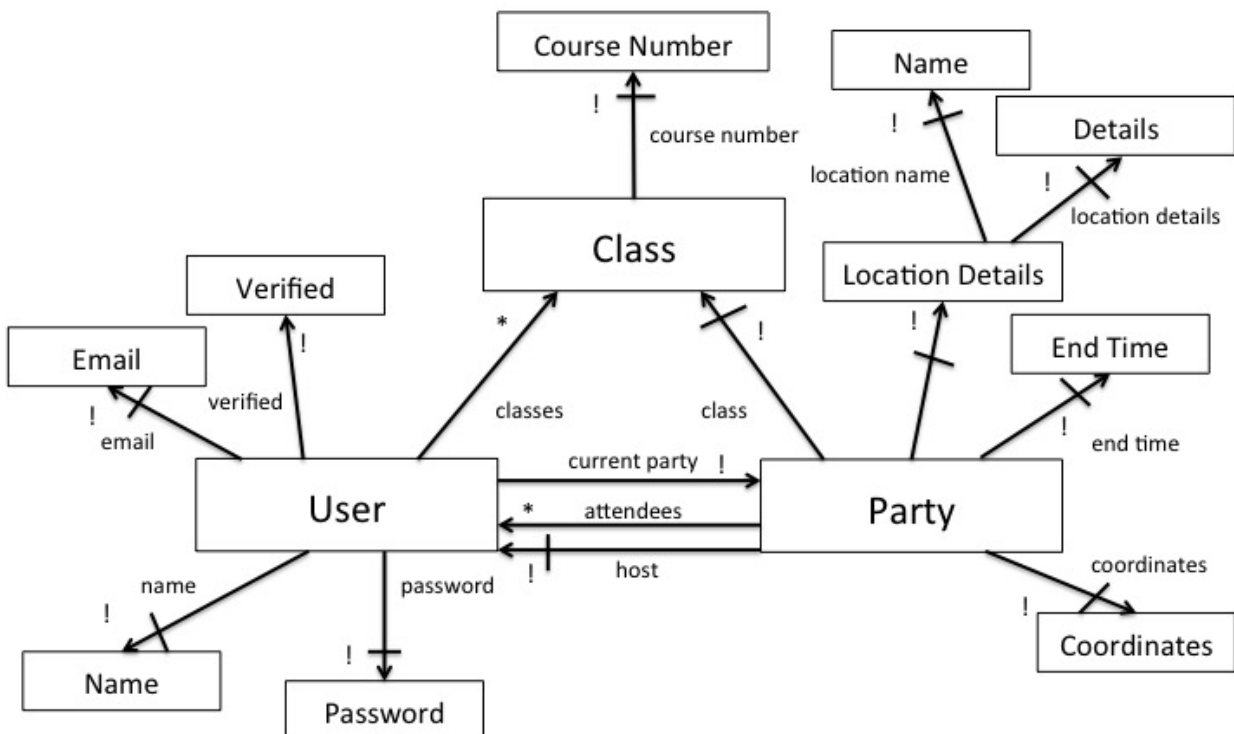
- **Party** - A party is a group of MIT students psetting together at a certain location for a certain class. By having parties, we allow MIT students to search for the classes they are in, fulfilling the first purpose of connecting students in the same classes, and to figure out where people are, fulfilling the purpose of helping people physically find each other. Parties are necessary in order to help people physically find one another.
- **Attendees** - An attendee is someone who is going to a party. This concept is important because we want people to use this as a social networking site, and be able to choose to attend a party based on how many people are there or which people are there. Attendees are necessary in order to connect people in the same MIT classes to pset together.
- **Class** - The main criterion by which a user finds a party is which class it is for. They will not want to view parties that are not for the classes they are taking. Therefore,

each user has the classes they're currently taking at MIT that they can delete or add classes to. They'll also be able to filter which classes they see on the map further from that list. Classes are necessary in order to connect people in the same MIT classes to pset together.

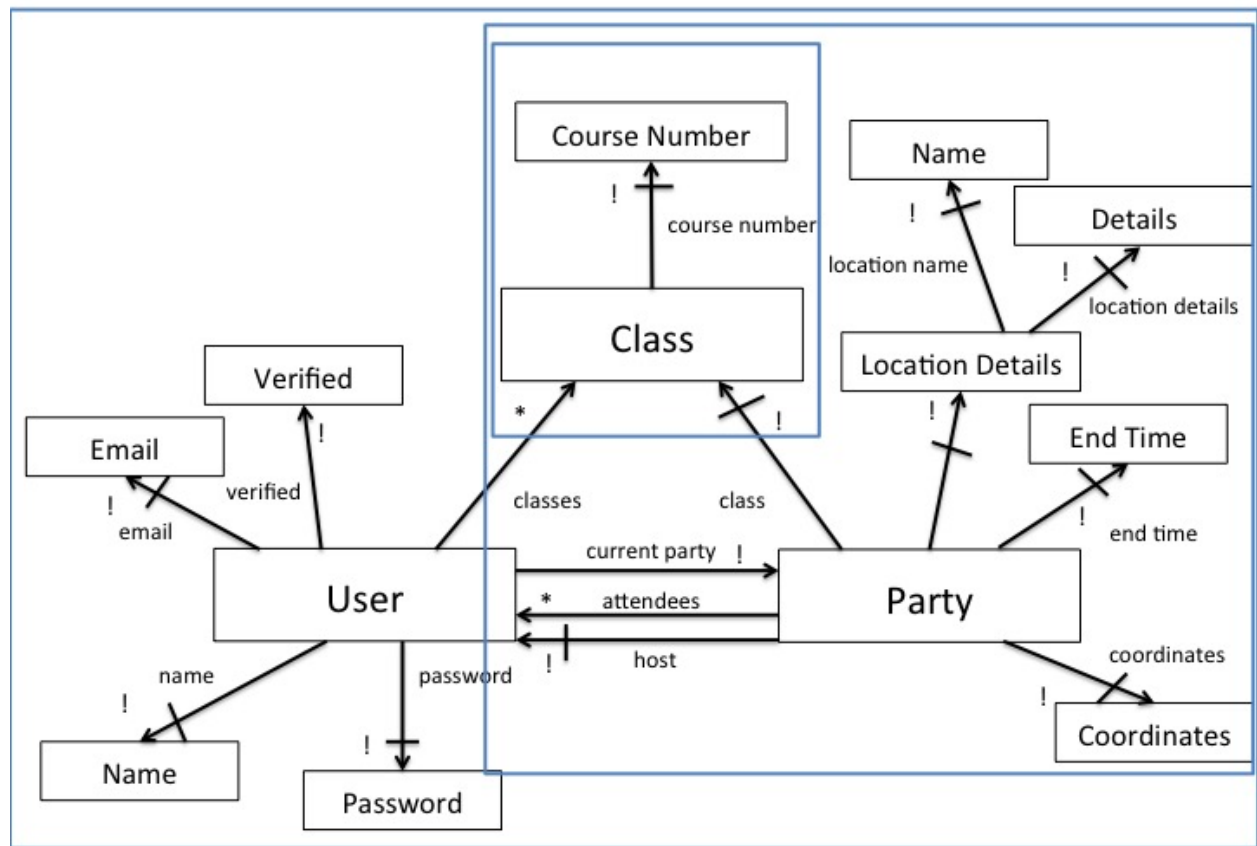
Context Diagram



Data Model



Data Design



Security Concerns

The security requirements are that:

- Only MIT students are able to view existing pset parties.
- Personal information about users are kept hidden from other users. The only information that should be displayed regarding a user is possibly the location of where they are (and even then, the location is not directly mapped to a specific user).

Potential risks include:

- Fake users signing up for the service for purposes aside from setting up pset parties.
- Malicious users who create nonexistent pset parties in order to distract and deter fellow users.
- Using information about popular locations on campus for evil purposes.

Threat model:

- Can assume little to no interest from criminal figures as only minimal personal information (name and email) is stored.
- Fake users are very likely.
- Impersonating another user is unlikely because there is no benefit to being one user over another.

Mitigations:

- The only communication directly between users is in person. The only other communication is in communicating the location of the party.
- An email authentication system will be established to make sure that users are valid users through a confirmation email.
- Prevent alterations of parties once they are established.

Design Challenges

Show Users names vs. Only Show Numbers:

- **Show Users Names:** This would allow users to see who is in a party that they were considering going to. This might give users extra motivation to go and it would give us some extra work to do.
- **Only Show Numbers:** Simpler view, not necessary to have another modal to see the rest of the people in the party. Doesn't give friends the opportunity to specifically meet up.
- **Decision: Only show numbers.** If we show names people will only look at the names of people they know and if they know no names they won't go to the party. Working with people you don't know is also possibly more efficient because of the lack of unrelated conversation. Our first purpose: "Connect people in the same MIT Classes to one another". If we show names we aren't connecting new people, we're doing something a text message could do just as well and people will likely stop using our service and start just texting each other.

HTML map element vs. Map API vs. Textual representation:

- **HTML Map Element:** This will most likely give us the most ability to do whatever we want with the map without extra bulk on top of exactly what we want to do. The main con for the map element is that we don't really have any base functionality and we're building from nothing.
- **Map API:** This could be good because a lot of the features we want to make are already built, we would just need to change them slightly for our uses. The main con is that it has that extra bulk on top of it like zooming in and out when we just want to show the MIT campus (just as one example).
- **Textual Representation:** The textual representation would be the simplest when it comes to design/coding time for us. It also kind of defeats the purpose of helping people physically find each other since maps are particularly good for that.
- **Decision: map API.** We felt that while we couldn't do an overlay well in a map api, we could do pins very well and we decided that was what was much better for readability of users. We also think that letting the API do the pins part of the map makes our implementation job really simple.

Dynamically updating party info on the map: Refresh button vs. Sockets sending updates whenever available:

- **The Refresh Button** would be the simplest to implement because it would basically just refresh the map part of the website. The really bad part about it is that this slows down the users a lot and the user never knows when refreshing is useless or not. This would lead to an incredibly discouraging user flow for the user (refreshing over and over until they find something).

- **Sockets** would take more development time but would make it much easier for a given user to find what they are looking for (or quickly find that there is not what they are looking for.)
- **Decision:** Since the refresh button is so easy to implement, we heavily considered it but in the end we decided it's crucial for the user experience to have the data be pushed to them. We decided on using **sockets for the final implementation** but keeping the **refresh button for the MVP** in order to keep the code simple.

Having users enter their classes when they join vs. Having it as an option:

- **Why Important?** This is a traction issue that will be essential to gaining new users and making sure they stay with us. We want to streamline the process so that a user can get in quickly but also use the better features of our app quicker.
- **Having users enter their classes when they join:** As a new user no one wants to fill out a long profile (adding all of their classes) but with these data the map we show will be much more specific to what our user actually wants to see.
- **Have it as an option:** If we have it as an option our users may never even notice the option and think that our app is really shallow (opposite of deep here) and will not be actually useful to them.
- **Decision: Compromise.** Rather than making the user slog through making their classes all at once without any results until you submit all your classes, we're going to have a tab on the main page to add classes. When you first load the page no pins will show up but we will point you towards the plus button which you can use to add a class you are a member of.

Satellite pictures vs. Google map pictures vs. Map distributed by MIT:

- **Satellite pictures** would not be labelled nicely and they wouldn't have any separation from building to building making it hard to use for new students who don't know where all the buildings are.
- **Google Map pictures** have labelling and would be easy to use but the labelling is arbitrary. Some buildings are labelled with building name some are labelled with numbers.
- An **MIT map** would be very nice but would take the extra work of converting it to whatever form our map api likes. We would also have to tile any picture we want to use if we want zooming to be fast.
- **Decision: MIT map** because it makes the labelling of buildings much better which is important for the user because often they need to know what building to label their party as being in. It is also possible to use the google api without using the google maps which is nice.

Authentication - Certificates vs. Confirmation Email vs. Trusting users with @mit.edu emails:

- **Certificates** have the potential to be easier for the user. One ok button and we know who you are. One downside is that it's much harder to use it on mobile. It may also be harder to implement because we have to figure out how to identify certificates
- Having a **Confirmation Email** to confirm identity would be a bit harder, because the user has to navigate away to even use our service. However it is easier for us to implement and makes the data model simple because we just have a small field of whether or not the user is confirmed.
- **@mit.edu emails** would be the simplest of all of these options. It doesn't offer much security because anyone can just enter an @mit.edu email to get into our system.
- **Decision: @mit.edu emails.** While it would be nice to use certificates, it's not essential to our goals to make absolutely sure people are who they say they are. Much more important to us is that we lower the barrier to entry for users. If we have a confirmation email or require them to have certificates on their browser this might slow them down.

Code Design Challenges

Google Maps vs Leaflet

- We started out trying to use **Google Maps**, but our original hesitations to use a map API were fulfilled: it was too bulky for the simple things we were trying to complete. In order to use an MIT map, we would've had to tile the map, put the tiles on the server, and make API calls to retrieve them. Adding markers (especially in different colors) would've been similarly complicated.
- **Leaflet** is pretty lightweight and doesn't require much to start using. If we need to add more complicated things later in the project leaflet may make it much harder. It also does not require tiling of the map so it is easy to use but might have a slow load time for the user
- **Decision: Leaflet** for our map API. It was much more lightweight, and using a custom image was much simpler. We were also able to find an external library that you can use to generate custom Leaflet marker icons.

Finding a party on the map

- **Challenge:** To fulfill the purpose of people being able to physically find each other, users needed to be able to find a party on the map after identifying one in the list. However, Leaflet doesn't provide a way to tag then find markers with identification.
- **Solution:** We maintain dictionaries that map HTML marker elements to their party ID, then another dictionary that maps those tables to the course the parties are for. Later

when we're looking for the marker for a specific party, we can look it up directly if we know the course, or find it by iterating through all the courses.

Mockups

P(arty)Set

Email

Text

Password

Text

Sign in

Login

More parties, less problems

Make connections,
Finish problem sets

Email

Text

Name

Text

Password

Text

Sign up

P(arty)Set

Welcome, user Logout

6.034

6.006

6.005

+

Location

Attendees

Reading room

8

Until: 6pm

Back left table, all welcome!

Join

Reading room

8

Reading room

8

Reading room

8

Reading room


8

Reading room

8

Reading room

8

A map of the MIT campus showing various locations and markers. The map includes labels for 'Kilian Court', 'Lowell Court', 'du Pont Court', 'W13', 'W11', '2A', '3', '4', '6B', '1', '2', '5', '7', '1A', 'MEMORIAL D', and '3'. There are several blue location pins and green location pins scattered across the map. A black plus sign is visible in the top right corner of the map area.

P(arty)Set

34

☒ 6.006

☐ 6.005

+

LocationAttendees

Reading room8

Until: 6pm

Back left table, all welcome!

Join

Reading room8

Reading room8

Reading room8

Reading room8

Reading room8

Reading room8

Reading room8

Welcome, user

Logout

New Party

Class

6.034

▼

Location

More info

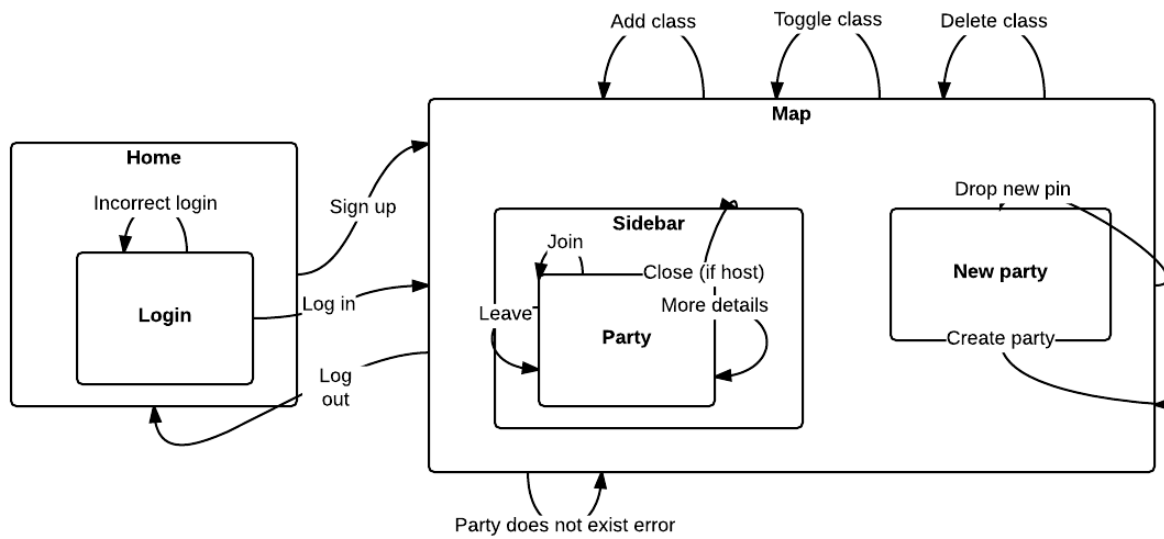
Working on lab 4

Duration

Please enter a location like 4-124

Create

Wireframe



API Calls

User

POST /

Adds a new user

name: String representing the name of the user

password: String representing the encoded password of the user

email: String representing the MIT email address of the user

verified: Boolean whether user's email address is valid (for future implementation)

party: Party that the user is currently attending

courses: Array of Courses representing classes the user is taking

POST /login

Logs in user

email: user's email

password: user's password

POST /logout

Logs out current user

PUT /{course_id}

Adds course to course list for current user

course_id: id of a specific course

courses: array of the courses current user is taking

PUT /delete/{course_id}

Removes course from course list for current user

course_id: id of a specific course

courses: array of the courses current user is taking

GET /loggedin

Returns current user (for internal use only)

Party

POST /

Creates new party and add current user to it

location: String representing location of party so users can find the party

details: String representing any further necessary information about the party

latitude: Number representing latitude of location of party

longitude: Number representing longitude of location of party

endTime: Date representing when the party will expire

attendees: Number representing how many users are currently attending the party

course: Course which gave the problem set

host: User who created the party

GET /

Returns all ongoing parties

GET /{party_id}

Returns information about specified party

party_id: id of specific party

PUT /{party_id}

Adds current user to specified party

party_id: id of specific party

party: party that current user is currently attending

DELETE /{party_id}

Removes current user from specified party

party_id: id of specific party

party: party that current user is currently attending

PUT /terminate/{party_id}

Ends party by the request of the host (to be implemented in the final product)

party_id: id of specific party

endTime: date when the party will expire

party: party that user is currently attending

PUT /{party_id}/invited/{user_id}

Add user to party through an emailed invite (to be implemented in the final product)

party_id: id of specific party

user_id: encoded id of specific user

party: party that user is currently attending

Course

POST /

Adds a new course

courseNumber: String representing the name of a course

GET /

Returns all courses

GET /{course_id}

Returns all ongoing parties for specified course

course_id: id of specific course