# Rush Tracker: Design Document

*By Random Engineers (random-engineers@mit.edu)*

Ben Shaibu
Dennis Smiley
Matt Kerr
Louis Descioli

# Overview

## Purpose and Goals:

The goal of this project is to design and develop a web application called Rush Tracker— an information storage, management, and presentation solution for keeping track of rushees during fraternity rush.

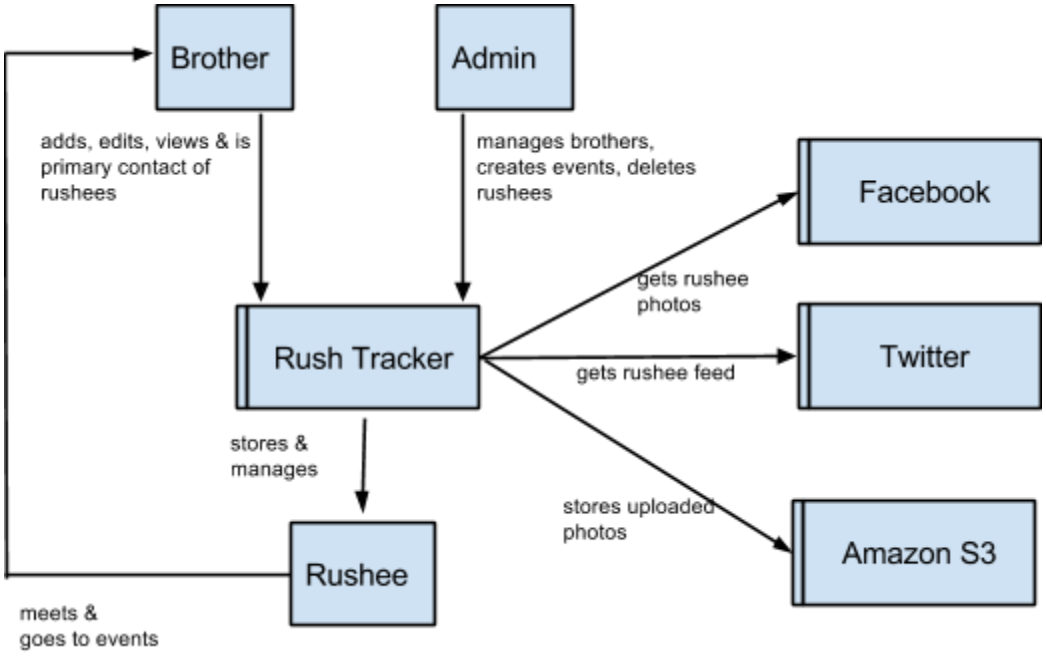
The current method for managing rushee information during rush is slow and inefficient because it involves using two decoupled views of the same data. The first view is the data management/storage view— usually a Google Spreadsheet— and is used to keep track of vital information for potential rushees. The second view is the rush-meeting presentation view— generally a Slideshow that is manually created from that Spreadsheet —and is used during meetings to drive recruitment discussions about each rushee.

Currently there are a variety of Rush Management applications on the market, but none of them has attained widespread appeal due to missing features and a lack of visual appeal. They have a very corporate feel that is unattractive and hard to use. RushTracker offers a friendlier interface along with a unique Presentation Mode that automatically showcases Rushee information in a format perfect for Rush Meetings. Users of alternatives (such as RushTap and ChapterPlan ) still have to manually create presentations using other software, such as PowerPoint.

## Disclaimers of Scope and Goals:

The first iteration of the application is designed to be used by a single fraternity. We have designed it where it will be easily possible to extend the application to support multiple fraternities. This is shown in the data model diagram. Further, our application tracks events, but only for the purpose of seeing which rushees attended them; providing event management and planning tools is **not** part of our goals.

**Context Diagram:**

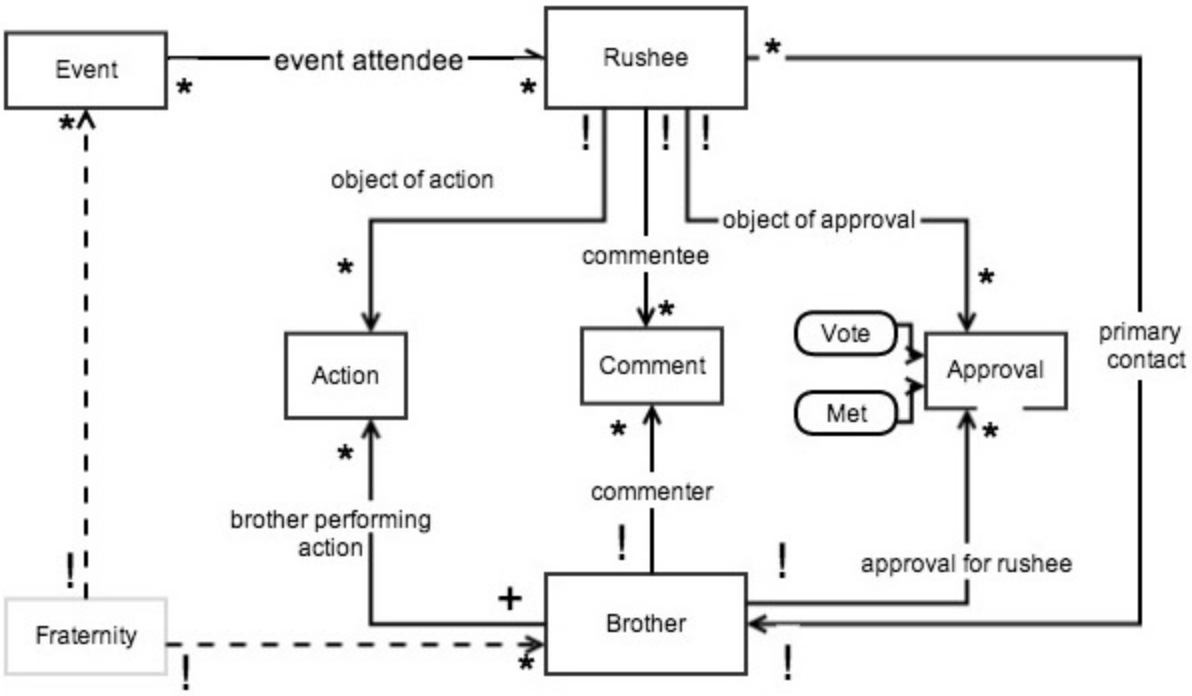


# Concepts

## Key Concepts:

- Rushee
    - A Rushee is an individual who is a potential Brother of a Fraternity. As such, besides their basic information (Name, Age, Potential Major, Hometown, Contact Information, etc.), it is also important to keep track of which Events a Rushee has attended and their current rushee status.
    - Rushees have an Action Status, based on the Fraternity's plan of action for recruiting the Rushee. An Action Status is either 'Stay the Course', 'Push Harder', or 'Repudiate'.
    - Rushee's also have a Bid Status. A Bid Status is either 'None', 'Offered', 'Accepted', or 'Rejected'.
- Brother
    - A user of the system and a member of the Fraternity. Brothers can also serve as a primary contact for a Rushee. Most of the contact with a Rushee is usually through their primary contact.
- Event
    - An event is a representation of an activity the Fraternity holds in order to recruit Rushees. If a Rushee attends an activity, they are added to the attendees of the respective Event.

### Data Model:




The Fraternity entity is included to show how the system would be extended to support multiple fraternities using the system. It will not be included in the MVP.

## Behaviour

### Features:

- Rushee Profiles
    - All rushees are given profiles that contain their basic information and any additional information a Brother feels is important.
    - The profiles are integrated with a Rushee's social media accounts to display photos
    - Track which brothers have met which Rushees
    - Keep track of Rushee Action Status and Bid Status (described more in the Rushee key concept)
    - Brothers can write comments on a Rushee's profile.
- Administrator Accounts
    - Create and remove Admin accounts
    - Create and delete brother accounts
    - Change status of Rushees
    - Delete Rushees
- Brother Accounts

    - Assign Brother as the Primary Contact for several Rushees
    - Receive Action Reminders for Rushees the Brother is the primary contact for, via email and a view on the Brother's account page
- Presentation Mode
    - A summary of all the Rushees and their profiles
    - Highlight Overall Status of Recruitment
- Create Events
    - Date, Name, and Event Details
    - Monitor Which Rushees are Attending Which Events
- Reminders
    - Daily messages are sent to Brothers (via email) reminding them to invite Rushees to events, meals, etc.
    - The messages contain information about the status of the Rushee and what are the next actions the Brothers need to take
    - Brothers will be able to view reminders of their pending actions on their Homepage
- Profile Search
    - Filter Rushee Profiles by Names, Activities, Potential Major, Hometown, etc.
- Voting System
    - Brothers can upvote Rushees
    - A brother has one vote per Rushee. It is either present, or not. Only upvotes, à la Hacker News, or Facebook likes.
- Mobile Version of the Site
    - Allows quick information updating
    - Easy Access to Rushee Contact Information
    - Simplified, focused layouts
- Social Media Integration
    - Our application will pull Rushee images from their Facebook profile for use as a profile picture
    - Can display a Rushee's Twitter feed on their profile page

## Security Concerns:

Security Policies

- Only authenticated Brothers (and Admins) can access the system.
- Only Admin accounts can authenticate new users.
- Only Admins can remove Rushees, Brothers, and Events.
- Users passwords are encrypted and stored as a hashed digest.

Threat Model

- An unauthenticated user can try to construct requests to the application.
- A user with basic credentials can construct requests or make requests via forms.
- A third party website can attempt to load data from application.

Specific Concerns

- An anticipated concern is that of malicious users impersonating Brothers in order to gain access to the system.
    - **Mitigation**: In order to prevent this scenario from occurring, all new accounts need to be verified by Admins before they are granted access.
- Another concern is that of a malicious Brother, or a malicious user who has gained control of a Brother's account, corrupting the data stored in the application.
    - **Mitigation**: In order to prevent data from being maliciously removed, only Admins are allowed to delete important information (Rushees, Events, Users) from the application.
- There is also the potential that an Admin's account can be compromised. In this scenario, short of rolling back the database, it is very difficult to mitigate the fallout.
    - **Mitigation**: Encourage users to secure their admin accounts and enforce passwords are a secure length.

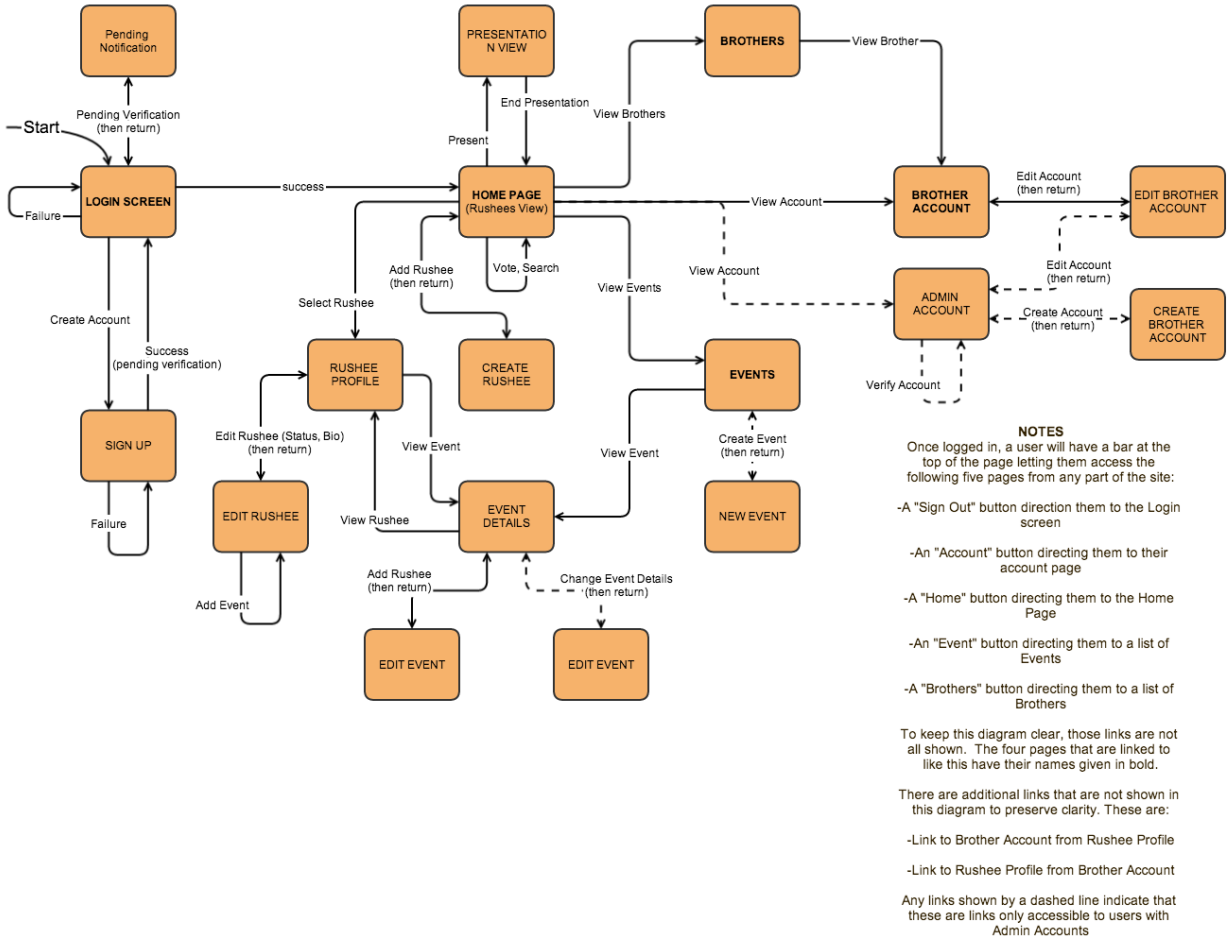
Common Concerns

- SQL Injection
    - Mitigation:
        - The application avoids calling an interpreter and only builds on the fly commands with expression objects, not strings.
        - The application relies on Rails' ActiveRecord ORM and uses only parameterized queries.
        - All used methods format and sanitize passed inputs for any queries.
- Cross Site Scripting
    - Mitigation:
        - We primarily rely on the defenses built in to Rails 4 to prevent XSS attacks.
        - We can prevent XSS by implementing Strong Parameters/Mass Assignment. Throughout the application, we use *require* and *permit* to whitelist what arguments can be passed.
        - We are sure to escape all HTML and stings rendered in the view. We do not disable this or use raw.
- Cross Site Request Forgery
    - Mitigation:
        - The application uses Rail's built-in CSRF Security Token to prevent against forgery.
- Packet Sniffing
    - Mitigation:
        - In order to prevent Packet Sniffing (as well as defend against XSS), we implement Transport Layer Security along with HTTP

Secure. This encrypts the packets and prevents attackers from reading the transmissions.

# User Interface:

## Website Flow Diagram:




*A file containing a larger image is saved in the docs folder under 'Workflow Diagram v2.png'.
Please refer to the notes section for a more detailed explanation of the diagram.

## Wireframes:

Please refer to the Wireframe Document.

# Challenges

## Design Challenges and Decisions:

- ➢ **Implementing search/filtering:**
  - A feature of the application will allow users to filter Rushee profiles. In order to implement this feature, we considered either using a gem or using Javascript to dynamically hide profiles. Due to the limited number of potential Rushees (this number is usually less than 40), we feel that using Javascript will be sufficient.

- ➢ **Rushees without social media:**
  - **Problem:** Another feature of this application is the ability to integrate Rushee profiles with their social media accounts. However, we anticipated that some Rushees may not have social media account or their accounts will be difficult to find.
  - **Solution:** In order to mitigate the problems that can be caused by this, users are able to upload their own images to a profile.
  - **Effects:** This will increase the storage resources required by the application.

- ➢ **Handling Image Storage:**
  - **Option 1**: Share a link to a Rushee's photo on Facebook
    - Pros: Reduce the amount of resources the application needs to provide storage
    - Cons: Problematic if a Rushee does not have a Facebook account or has limits public access to their photos
  - **Option 2:** Store a Rushee's images on an external server dedicated to storing static assets
    - Pros: Allows brothers to upload photos - a Rushee may not have Facebook, or may not have public photos
    - Cons: Forces the application to provide storage resources
  - Decision: Both. A Rushee's Facebook photos are integrated into their profile on Rush Tracker if possible. If not, Brothers can upload photos to a Rushee's profile.

- ➢ **Reminder as a Data Model Object:**
  - **Option 1: A** Reminder is just a text field belonging to a Rushee and does not have a relationship with any other object in the Data Model
    - Pros: Simplifies the Data Model.
    - Cons: Difficult to track the relationship between the primary contact and the reminders belonging to the primary contact.
  - **Option 2**: A Reminder is an object that has a one-one relationship with a Rushee/Primary Contact pair
    - Pros: Easy to lookup the reminders for a particular primary contact

        - Cons: Introduces more complexity to the Data Model
    - **Decision**: Option 2. An explicit relationship between a Reminder and a Rushee/Primary Contact pair allows for the easy lookup of a primary contact's reminders.

➢ **Variety of Use Cases**
- An interesting question to be considered for the application is how varied the use cases actually are. While we focused entirely on the Rush use case from the very beginning of the design process, it is possible to imagine this application being used in other scenarios where there is an initial time-period of potential members meeting existing members of a society. Keeping this in mind, a possible future extension to the application is perhaps making the design (layouts/visuals) and names more universal.
- **Decision:** We will focus on creating the best possible application for our initial use case; if we feel more use cases are necessary, we will focus on extension at that time (outside the scope of this project).

➢ **Brother - Rushee Relation:**
- An important design consideration is the relationship between a Brother and a Rushee. We would like to provide brothers the option to view and add comments to all of the Rushee profiles, while being able to keep track of who the primary contact is for a Rushee. In order to provide this flexibility, two possible options were considered.
- **Option 1:** Either the primary contact would just be a field and there would be no explicit relation between a Brother and a Rushee,
- **Option 2:** There would be a one to many relationship between a Brother and Rushees.
- **Decision:** By choosing the latter, it provides us with more flexibility and lays the groundwork for the Reminder feature.

➢ **The Nature of Votes:**
- Votes are a way of expressing the sentiment of the Fraternity towards each Rushee. A design decision regarding this is the definition of what a Vote is and what are its subtypes.
- **Option 1:** A Vote only has one type - an Upvote. This is similar to a Facebook Like.
    - Pros: Simplifies the voting process. Comparing Rushee's through their respective vote count is trivial - Rushee's are ordered from lowest amount of votes to highest.
    - Cons: There is no manner in which a Brother can express a negative opinion towards a Rushee. Perhaps one negative vote from a Brother is enough to dissuade the Fraternity from offering a bid. The presence of

negative votes may also allow for more complex opinions to be formed regarding a Rushee.

- **Option 2:** Votes may be either positive or negative - there are Upvotes and Downvotes.
  - Pros: Allows Brothers to share negative opinions towards a Rushee, which may be as equally important or even more important than a positive opinion.
  - Cons: Introduces extra complexity and challenges regarding the ordering process of Rushees. For example, how would you compare a Rushee with three positive votes only and a Rushee with 10 positive votes but who also has 5 negative votes.
- **Decision:** Option 1. Brothers may only upvote Rushees, and the absence of votes will also influence a Fraternity's bid decision.

➢ **Security and Admin Accounts:**

- The design of the application introduces several security challenges that needed to be addressed through the use of Admin accounts. In order to prevent 'fake' Brothers from signing up, or to preserve the integrity of the application's information, we only allow Admin accounts to verify new users and only Admin accounts are allowed to delete any data from the application.
- **Differentiating between different account types:**
  - **Option 1:** An Admin is a difference type of data object.
    - Pros: Having a new object makes it very explicit what an Admin is. Also allows for the introduction of Admin only relationships in the Data Model.
    - Cons: May introduces unnecessary complexity to the Data Model.
  - **Option 2:** The Brother object has a boolean field that represents whether or not the Brother is an Admin.
    - Pros: Extremely simple to implement.
    - Cons: May not accurately reflect the privileges and relationships an Admin has.
  - **Decision:** Option 2. In this system, an Admin is merely a sub type of a Brother object with only a few more privileges.
- **The Amount of Admin Accounts:**
  - **Option 1**: Each fraternity only has one super Admin account with administrator privileges, without the ability to create new Admin accounts.
    - Pros: Secure in that only one 'trusted' individual will be able to administer the system.
    - Cons: There is no backup plan if the credentials for the Admin account are lost. The presence of only one Admin account also makes it difficult for the system to be inherited by a new group of Brothers. If the credentials are lost or are not communicated to the next administrator, the next generation will not be able to

administrate the system. Finally if a malicious user is able to obtain the credentials for the Admin account, it is not possible to mitigate the adverse effects the malicious user may have - changing the credentials, deleting information from the system.

- **Option 2**: Each fraternity has multiple Admin accounts that can create new Admin accounts.
  - Pros: Creating new administrators for the next generation is simple. Also allows for more flexibility in case multiple individuals should have administrator access. Finally, there are multiple backup Administrators in case a single Admin account is compromised.
  - Cons: There is a chance that an admin may purposely or accidentally give administrator privileges to another User who should not have the privileges.
- **Decision**: Option 2. In order to provide Admin accessibility to multiple types of Brothers (Fraternity President, Rush Chairman, Event Planners), multiple Admin accounts are allowed, and Admin accounts are able to designate new accounts to become Admins.

➢ **How do we create new brother accounts?**

- **Option 1**: Admin creates account and sends to brothers somehow
  - Pros: All new users are easily verified by the fact the Admin created them.
  - Cons: Admin must manually enter Basic User information and passwords.
- **Option 2**: Brothers create accounts and cannot do anything until their account is verified by Admin
  - Pros: Allows brothers to enter their own individual information.
  - Cons: Forces users to wait on Admin. Admin may have to check spontaneously/maliciously generated users.
- **Decision**: Both. This will allow the application to create users with both methods.

➢ **Initial Administrator Account and Creating Fraternities**

- **Option 1:** Users specify which fraternity they belong to on the signup page. If a user elects to join an existing fraternity he will need to be verified by the fraternity's existing administrator. If the user elects to join a fraternity that does not exist, his account is created and he is made administrator of the newly created fraternity.
  - Pros: This flow allows users to join existing fraternities, and allows for the creation of new fraternities with an initial administrator.
  - Cons: A user may sign up for an account and specify that a new fraternity should be created for him, but he may actually not belong to the fraternity that he claims he should be in charge of.
- Decision: Option 1. To address the drawback of users falsely claiming that they are in charge of a given fraternity, the sign-up page will support joining a fraternity based on the fraternity's administrator's email address, so there will be support

for multiple fraternities with the same name.

## Initial Database Schema Design Choices:

- Database Tables
    - **Brother Model** (Devise gem takes care of password / email )
        - STRING firstname
        - STRING lastname
        - INT fraternity_id
        - BOOLEAN is_admin
        - BOOLEAN is_verified
    - **Rushee Model**
        - STRING firstname
        - STRING lastname
        - STRING email
        - STRING cellphone
        - STRING facebook_url
        - STRING twitter_url
        - STRING profile_picture_url
        - STRING dorm
        - STRING room_number
        - STRING hometown
        - STRING sports
        - STRING frats_rushing
        - INT primary_contact_id
        - STRING action_status
        - STRING bid_status
        - INT fraternity_id (not implemented in MVP)
    - **Approval Table**
        - INT brother_id
        - INT rushee_id
        - BOOLEAN vote
        - BOOLEAN met
    - **Comment Table**
        - INT brother_id
        - INT fraternity_id (not implemented in MVP)
        - INT rushee_id
        - DATETIME timestamp
        - TEXT text
    - **Event Model**
        - STRING name
        - DATE date

        - INT fraternity_id
    - **Attendance Table**
        - INT event_id
        - INT rushee_id
    - **Action Reminder Model**
        - INT brother_id
        - INT rushee_id
        - DATE date
        - TEXT description