StartMart Design Document

Victoria Li, Julianna Mello, Angela Zhang

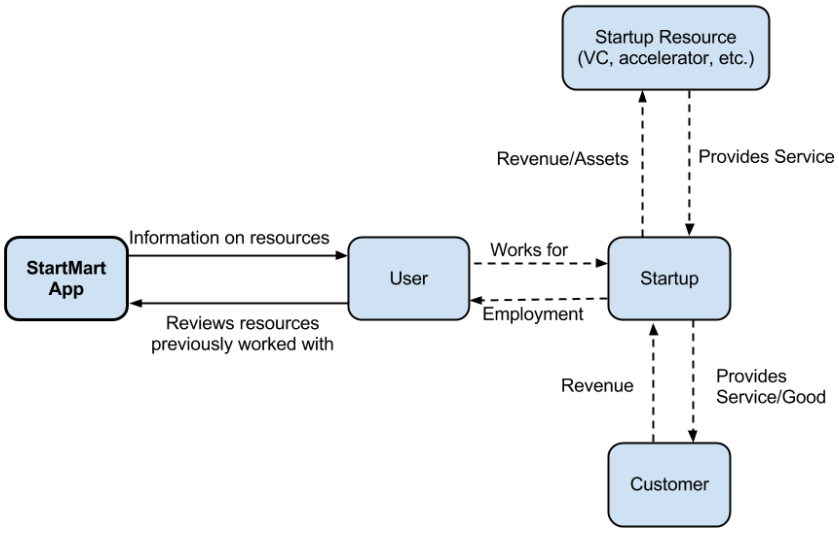
Purpose and Goals

The purpose of this project is to create a platform for MIT students involved in startups to review and discuss startup resources such as venture capital firms, law firms, and their partners. Students who have worked with a company or partner may provide a review and rating out of 5 stars, both publicly and anonymously. Students looking for assistance with startups may read these reviews and start a private or public discussion with the original reviewer and other users, in order to identify the firm and partner that will best assist them in their startup ventures.

We are building this website because there is no sufficient alternative currently existing. The Martin Trust Center for MIT Entrepreneurship has explicitly requested this platform be built, as many students have expressed needs for a place to get in-depth and honest information about these resources before committing to them. StartMart will provide a much needed service to the MIT startup community, and help MIT startups succeed.

Through this project, we hope to accomplish a number of goals we have set for ourselves. First, we hope to build an impressive product that will earn the team members high grades in the 6.170 course. Next, we hope to take this product beyond being the classroom and have it become a widely used resource for MIT students. Finally, we hope to to both use the material we have learned throughout the semester, and extend our knowledge to achieve a sense of accomplishment and pride in our work.

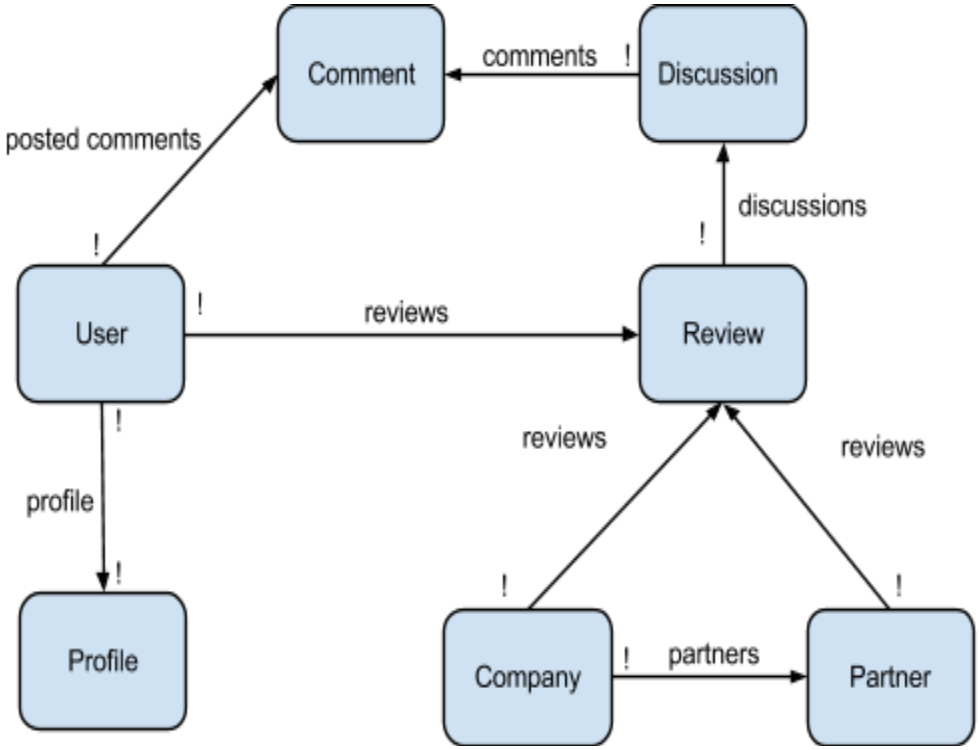
Context Diagram



Key Concepts

A **user** is an individual who has an @mit.edu address and who is either searching for resources for a startup, or who has worked with companies that provide resources for a startup and wishes to write an review on the experience.
A **company/firm** that provide resources for startups. The companies include Venture Capital firms, Angel investors, Law firms, Finance firms (banks), Accelerators & Incubators, and Others.
An **admin** is an administrative user who moderates content and may test features through an admin interface.
A **review** is a description of a user's experience working with a company or company contact.
A **partner** is an individual who works at a specific company and who will work closely with a startup.
A **category** is a company or contact's specialized field, for example law, accelerator, or finance.
A **discussion** a thread on which many comments are posted
A **comment** is a message sent between users and may be public, in which all users can view it, or private, in which only invited users may view it.
A **profile** is a description of a user account where personal information can be added.
A **notification** is an indication to the user that a discussion or a comment has been added to his/her review or a review that he/she had previously commented on.

Object Model



Features

1. User Profile:
    a. A user may create an account using their MIT email
    b. A user may edit their profile information, upload a profile picture, etc.
2. Admin Interface:
    a. An admin interface to allow easy management for administrators
3. Company/Partner:
    a. A user may create a company, or partner if it does not already exist (the company description can be auto-populated by the Crunchbase Database, or entered manually)
4. Reviewing:
    a. A user may create, edit, and destroy a review on either a company or a partner, either publicly or privately/anonymously
    b. A user may edit a company or partner's information
    c. A user may start a discussion on a review
    d. A user may add comments to existing discussions, either publically or privately/anonymously.
    e. A feedback/credibility system, upvoting/downvoting certain reviews and/or

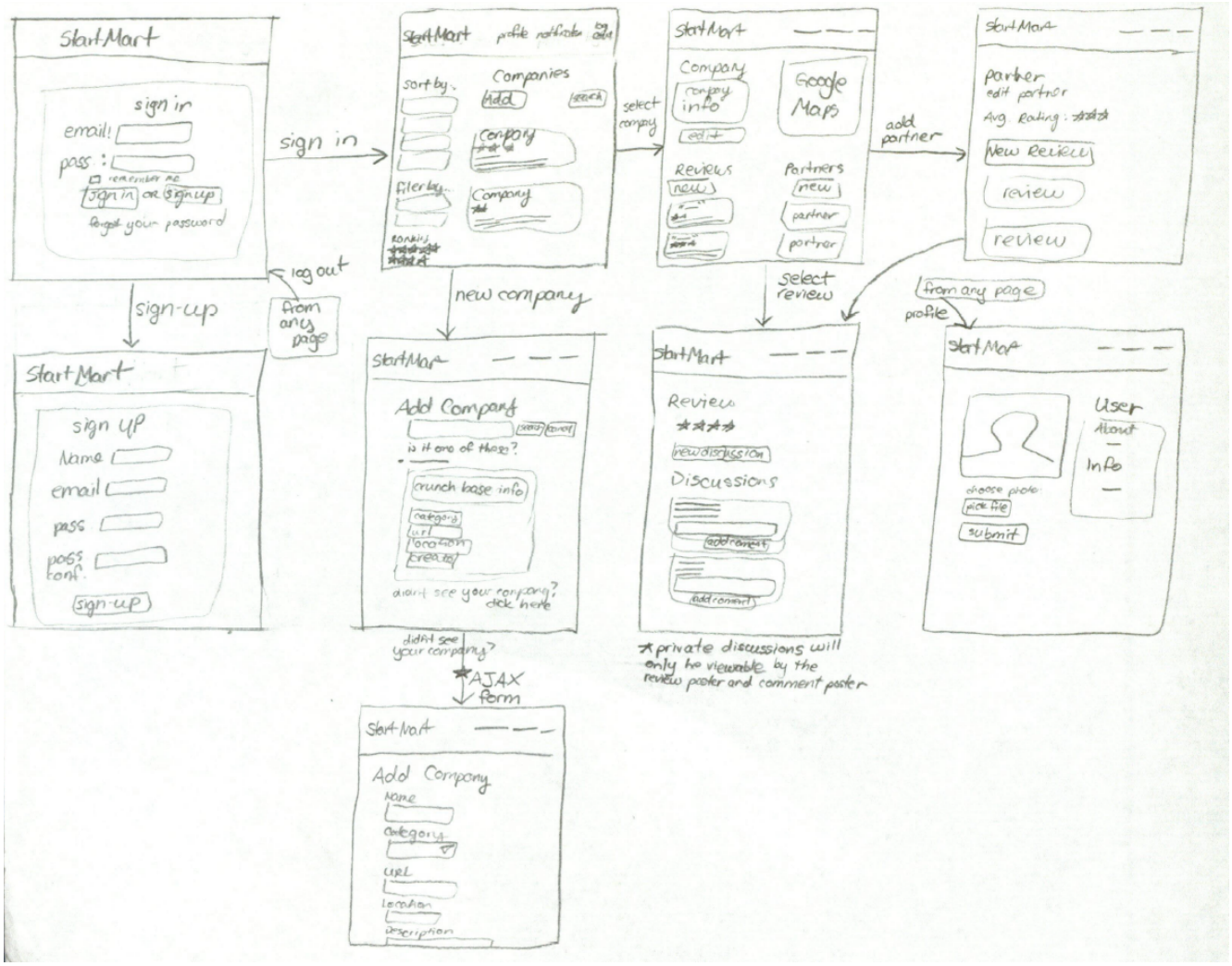
discussions.

5. Exploring Reviews:
    a. A user may search reviews based on keywords in company name, company category, company location, or company description
    b. A user may filter reviews by category or by rating.
6. Discussing:
    a. A user may post a public or private discussion. Public discussions are viewable by everyone and private discussions can only be viewed by individuals it is shared with.
7. Notifications:
    a. Each user who posts a review will be notified of all activities (future discussions, and comments on those discussions) related to the review.
    b. Each user who posts a discussion or a comment on a discussion will be notified of all activities (comments) on the discussion
8. Overall Experience:
    a. Many functionalities are implemented with AJAX for asynchronous experience

Security Concerns

- Session Hijack. Attackers might sniff packets and could get a user's session ID and hijack their session.
    - For the purpose of this project, we will not use HTTP Secure, as it is beyond the scope of the project (if we have time left, we will implement it). However, we can protect a user's identity by setting a time limit on the cookie.
- Session Fixation. The attacker could create and maintain a valid session id, then force the user's browser into using this session id, thereby forcing the user to use this session.
    - We will issue a new session identifier and declare the old one invalid after a successful login by calling reset_session.
- CSRF. Attacher includes malicious code on a link to a page that accesses a web application the user has authenticated and then execute unauthorized commands as the user.
    - We will generate a security token, calculated from the current session and the server-side secret, in all forms and Ajax requests generated by Rails.We will do that by setting protect_from_forgery :secret => "<insert_secret_here?".
- SQL Injection.
    - We will sanitize all input and output strings, and use Rails database query defaults instead of writing our own SQL statements whenever possible.
- XSS.
    - We will escape all user input before displaying it on the screen. Fortunately, this is done automatically by Rails. We will avoid using raw html.
- Ensuring that each user is only allowed to view content they are authorized to view.
    - For each view method, we will check the authorization before displaying the content. We can do that by either setting a before_filter for checking whether the current user has permission to view this content, or querying only within reviews and discussions that belong to this user.
- Ensuring anonymity for posting reviews and comments.
    - We will have a display_name attribute to each review and comment. When a user checks "private" when posting a review.comment, the display_name attribute will be set to "Anonymous." Otherwise, the display_name attribute will be set to the current user/reviewer's name.

User Interface



Design Challenges (1 - JM, REST - VL)

1. Private vs Public discussions

There are several interesting design challenges that must be considered with regard to private vs. public discussions.

1.1 First, how will the two be distinguished in the schema?

Option 1: A private discussion will be a separate model from a public discussion.
Option 2: Single Table Inheritance will be used to separate the two. There will be a single discussion model, and a 'type' attribute will be used to distinguish them.
Option 3: There will be a single discussion model and a 'private' attribute that may be true or false that will distinguish the two.

Option 1 has the advantage of being relatively easy to implement. However, it does not make much sense as the two types of discussions are very similar and should not be separated from one another completely.

Option 2 may be good because it establishes a clear relationship between public and private comments. However, the differences in a private and public are not so great as to require separate methods for them, so STI may not be required.

Option 3 is ideal for this project. The two types of discussions are contained within the single entity, "Discussion". They are distinguishable and may be displayed differently, but are contained within the same model and controller and share the same methods.

1.2 Next, how will private discussions be implemented?

Option 1: A private discussion will be like a personal message, and all private discussions will be in an inbox.

Option 2: A private discussion will be on the review page but will only be viewable to the original poster of the review and the poster of the discussion.

Option 1 has the advantage of being intuitive in that many websites are implemented in this manner. However, the review itself would need to be included in the personal message for reference. Option 2 is a good option because the review itself and all discussions will be viewable for reference.

2. Exclusion of Companies

This site can allow companies to be included, where they can see the reviews and perhaps monitor/ edit their company information. However, we have decided to exclude company access from our site. While this may mean more moderative work (which has been made significantly more easy with the use of scraping databases for relevant companies), we believe that this would benefit the users of the website and allow for complete honesty.

3. Category Implementation

One interesting design problem we ran into was how categories were related to companies → did companies belong to a category, or did companies have a category attribute? While it sometimes made sense that companies should belong to categories so that companies can be grouped into their representative categories, this made for a tricky relationship where companies had to be accessed or referenced to through a category.

For a more straightforward relationship, we decided that companies should have a category attribute, and filter based on this attribute. This made relations a lot simpler and sufficient for our purposes, but had the tradeoff where companies only belonged to one category. However, this tradeoff wasn't extremely significant in this context, as a VC company would not also be a sales and marketing firm.

4. Notifications

4.1 How do users receive notifications?

There are many ways in which notifications can be implemented: a newsfeed/ live stream of sorts, a pop up or a indication on the navigation bar, or through email messaging.
Option 1: A newsfeed / live stream is an interesting option as it allows users to view all notifications at once on a separate page, similar to that of a facebook feed. However, this may not be the most appropriate option for the purpose of the website, as there isn't a lot of information that needs to be displayed, other than "x has done y on your review". This would add unnecessary clutter.
Option 2: A notification integrated on the navigation bar seems to be a clean solution, as it allows users to quickly see how many notifications he/she has, and be quickly linked to the relevant reviews. A user can click on the notifications, and see the list as a dropdown menu no matter where they are located on the site.
Option 3: While we don't anticipate a high amount of traffic in terms of notifications, it generally seems bothersome to send an email to the user each time something happens on the site. Thus, we decided to move away from email notifications and keep the notifications local to the site itself.

We've decided to choose Option 1 for its simplicity, and for its ease of use.

4.2 *What notifications should they be able to recieve?*
While there is the opportunity to make notifications complex by allowing users to "follow" a comment or choose to receive notifications when comments are added to certain reviews, we have decided to follow a simpler model.
A user will be able to receive notifications when someone comments on his/her review OR when someone adds a new discussion. A user will also receive notifications of reviews that he/she has commented on.

Code Design

- User Authentication
    - We will be authenticating users using the rails gem Devise. It is a simple addon that will allow us to easily add features such as database authentication (encrypt and store passwords), token authentication, send confirmation emails to @mit.edu emai laddresses, track user analytics data such as sign in counts and ip addresses, set timeouts, and set account locks after certain number of invalid attempts.
- Testing
    - We will be testing with RSpec, because it provides richer command line options, a textual description of examples and groups, flexible reporting, and built-in mocking and stubbing framework.
    - We are also looking into factory_girl to simplify the test setup and define the object networks.
- Pagination
    - We will be using Kaminari for pagination. It is a scope and engine based paginator.
    - Alternatively, we are looking into implementing infinite scroll with AJAX.
- Clean HTML
    - We are considering using HAML on top of HTML to allow for a non-repetitive, elegant, and easy way to embedding Ruby.
- Authorization and access control for content.
    - We are looking at CanCan, which is a an authorization library for Ruby on Rails to restrict what resources a given user is allowed to access, so that we can store all permissions in one location instead of duplicating across controllers, views, and databases.
- Searching, Sorting, or Filtering Results
    - We will be using the will_paginate gem for searching, sorting, and paginating with AJAX. Specifically, we will be following this example: http://railscasts.com/episodes/240-search-sort-paginate-with-ajax
    - We are also considering Has Scope gem to leverage existing model scopes and keep the controller lean without conditional statements for each possible filter query.