StartMart Design Document

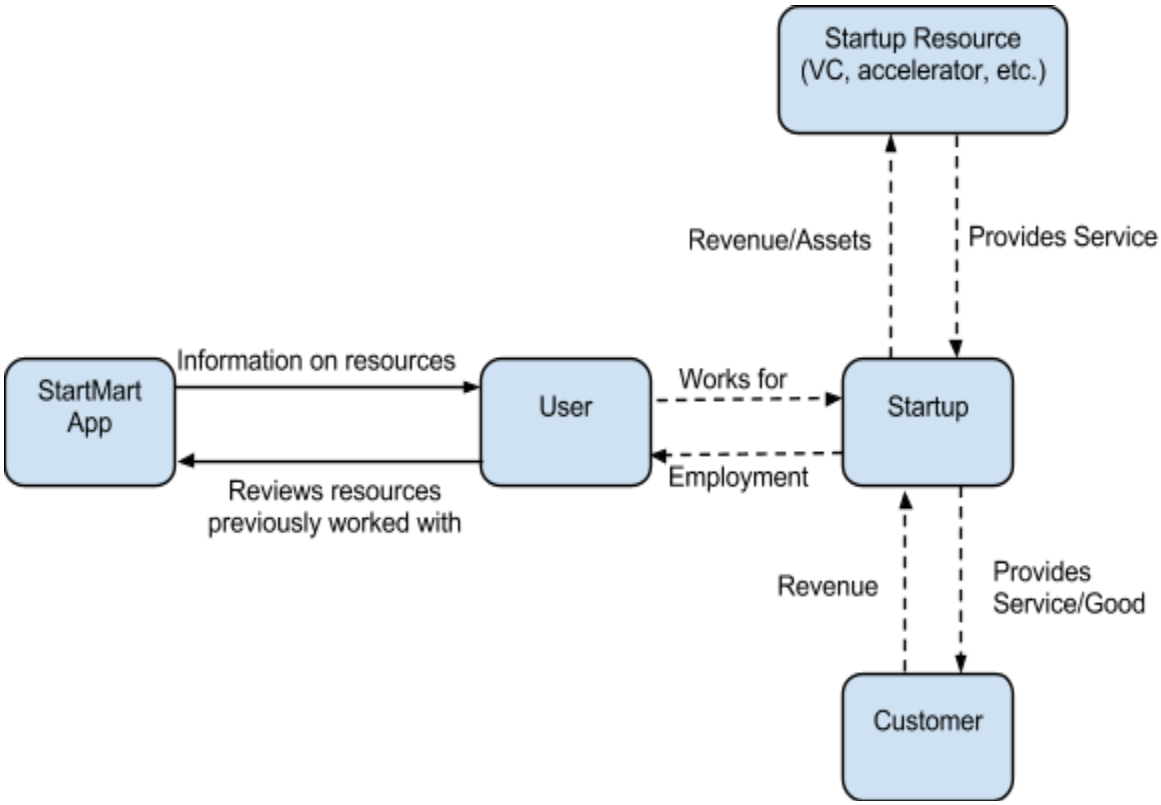
Victoria Li, Julianna Mello, Angela Zhang

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.

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.

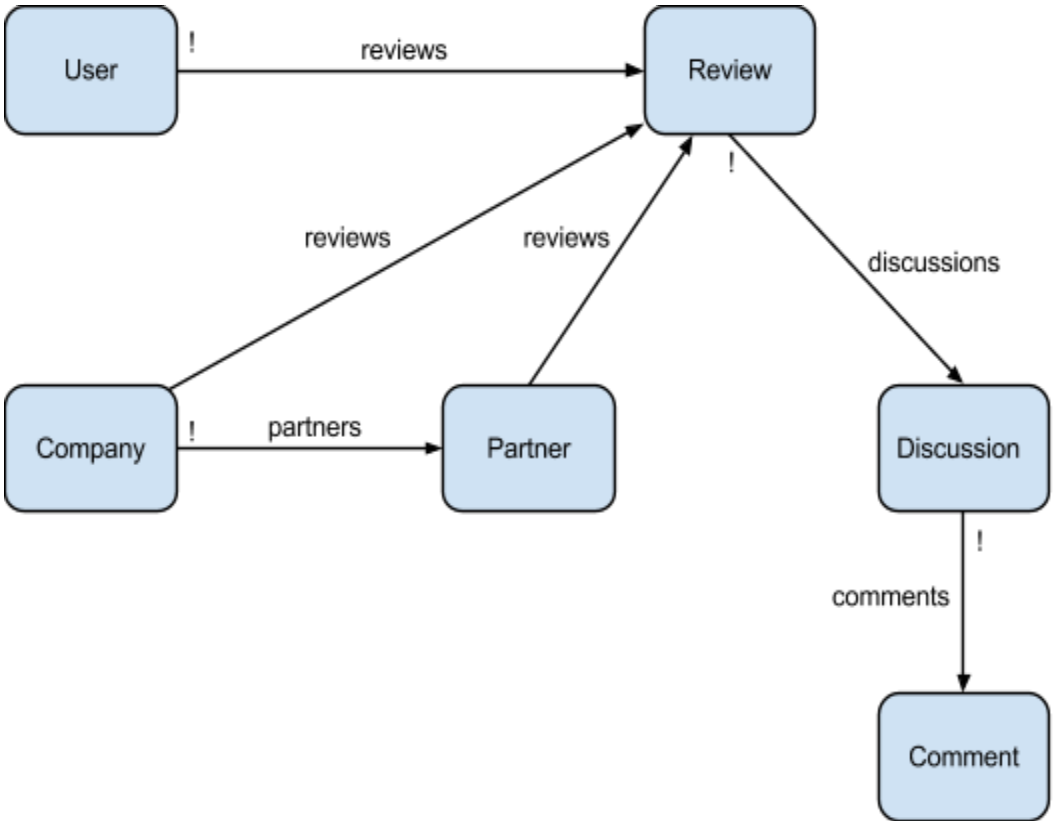
Context Diagram



Key Concepts

A **user** is an individual affiliated with MIT who is either searching for resources for a startup, or who has worked with **companies** or contacts on a startup before and wishes to provide information on the experience.
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** s 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.

Object Model



Basic Features

The descriptions of each feature of the site is described below. The basic features describe the minimal viable product whereas the advanced features are lower priority and will be added after the MVP is perfect.

1) User profile:
- A user may create an account using their email
- A user may edit their profile information
- A user may destroy their account

2) Reviewing:
- A user may create, edit and destroy reviews
- A user may create a company, or partner if it does not already exist
- A user may edit a company or partner's information
- A user may start a discussion on a review
- A user may add comments to existing discussions

3) Exploring Reviews:
- A user may search reviews based on a keyword, by category, or by rating

4) Discussing:
- 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.

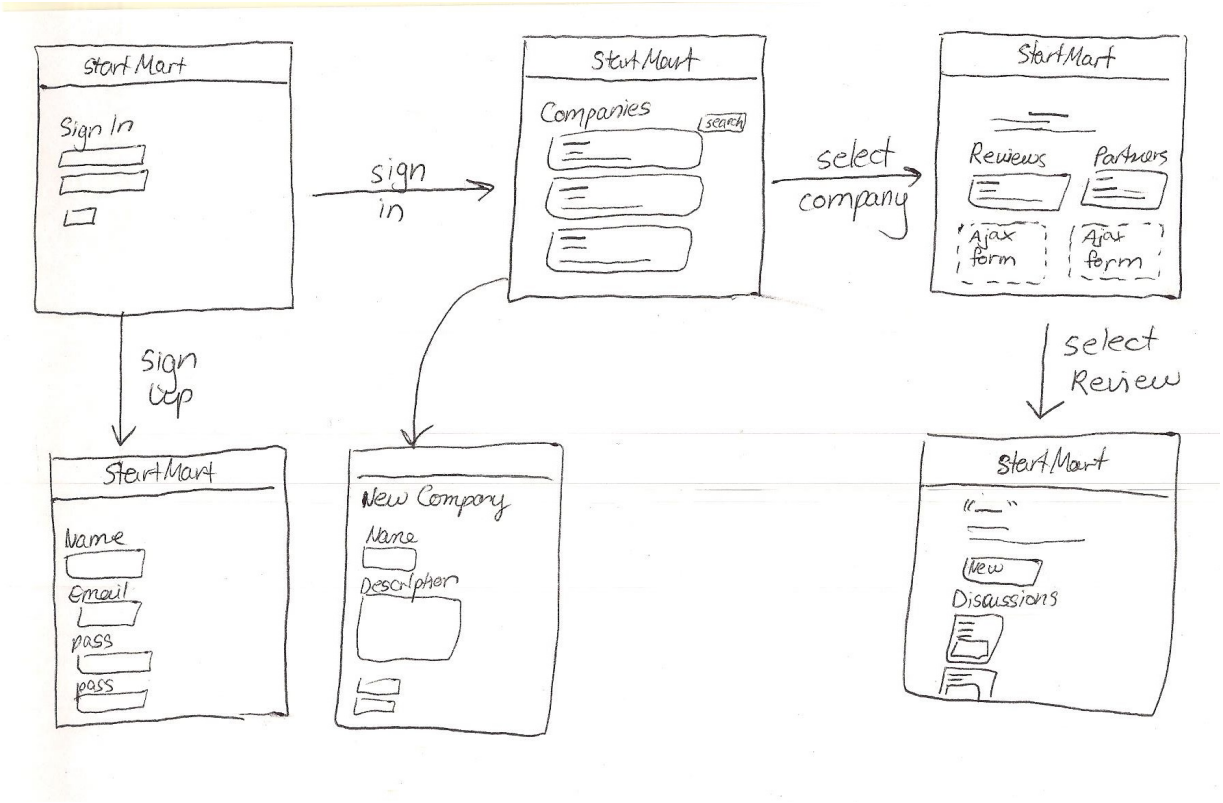
Advanced Features

1) Search suggestion based on search history, as well as auto-fill for searches.
2) Search by any combination of keywords, categories and ranks
3) Verify provided company information by cross referencing with some sort of external database
4) Allow integration with social media to "share" reviews --> this may be tricky to implement as security concerns will have to be addressed
5) Extended AJAX for asynchronous experience
6) A feedback/credibility system, upvoting/downvoting certain reviews and/or discussions.
7) An admin interface to allow easy management for administrators
8) Access limited to only MIT students through a web certificate or @mit.edu email address

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. However, we can protect a user's identity by setting a time limit on the cookie. We might consider using ActiveRecord::SessionStore to store the session data on the server rather than in a 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 an anonymity flag for both reviews and discussions. Before we display any related contents, we will check for the flag and display author as "Anonymous" whenever the flag is set to true.

User Interface



Design Challenges

1. Filtering

The key component of this site is being able to quickly and effectively filter through data to retrieve the relevant reviews. We are allowing users to filter based on multiple options, which means that scoping, ordering and finding reviews of a certain type will be very important. Attention should be given to ensuring the filtering is doing via helper methods and/or several methods to avoid lengthy methods and obscure code. Care also must be taken to ensure that results are not being repeated, the scope isn't so narrow such that no results are returned, or so wide such that irrelevant results are also seen.

We can use scopes for filtering, as there is a set amount of filters with more or less the same structure. Scope allows use of all rails query methods (where, joins etc). Scopes are also chainable, which is ideal in our case as multiple filters can be joined together. This is optimal as it allows basic queries with one parameter, as well as complex queries with many parameters.

2. Search

Due to the vast amount of data that would be available on this site, search is important to ensure

quick data access for users. Ideally, search will be implemented asynchronously for a smooth user experience. Search should also work as the user is typing. This can be done via simple javascript hidden fields, and simple methods in the model. The key challenge is to ensure that search also provides the relevant results.

Simple implementation should be easy, but there is also the option of implementing a “smarter” search that also searches for similar items, to offset potential user typos. This may or may not be feasible for the scope of this project.

3. AJAX

For smooth user experience, it’ll be best if the majority of the website was asynchronous. Discussions and review postings would when asynchronous to ensure that a natural flow in conversation can happen. This would be implemented in the MVP, as it is crucial to how the site functions.

Another use of ajax is in the filtering and/or searching, where searching through various companies will allow for an instant change in results. While this will not be the priority for the MVP, we will look to making this asynchronous for our extended implementation.

The problem with using a large amount of AJAX may end in messy and/or confusing code. If done wrong, this could result in a lot of bugs and undocumented changes to the database. A way to mitigate this would be to look into various js frameworks that may make it slightly easier, as well as implementing an extensive testing suite that would allow us to confirm if the right requests are being made. The javascript should be organized in an external file, and everything should be thoroughy Another way would to organize the javascript in an external file and ensure that everything is thoroughly commented and ordered in a logical manner.

4. Private and Public Discussion

An important part of the application is the ability to discuss reviews and/or thoughts between users. These discussions can be public for more general questions, or private if a user has a specific question for the reviewer. A big challenge would be addressing how these discussions are structured and shown on the application. We want to minimize the amount of “clutter” on the page by using a visually appeal layout, but also ensure that the user experience for the discussion is intuitive.

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.