Design Document: MIT HackTrack

George Ezenna, Kairat Ashim, Eric Manzi, Favyen Bastani

Overview

MIT HackTrack will be a hacker community where members of the MIT community share their latest projects, get feedback on their ideas, and find interesting projects to contribute to.

Currently, students don't have an effective medium for discussing projects. Although people frequently communicate project ideas over e-mail, Zephyr, or even social networks like Facebook, these platforms aren't tailored for the task. Long-lived project discussions over e-mail and Zephyr are difficult due to the lack of support for different discussion topics; it's also impossible for new users to join in the middle of a discussion since the conversation history won't be visible. Facebook presents a more viable platform: a student can create a Facebook group for their projects and post development updates, demo links, and screenshots in the group. However, Facebook groups still make it hard for other students to find interesting projects and filter projects by their interests; it lacks a browsable project directory. At the same time, Facebook's Orwellian privacy policy drives away many MIT students.

At the same time, students frequently start working on a large project, but shortly find that it's difficult to make progress by themselves in the small amount of free time that MIT students have. HackTrack enables students to share their idea and quickly see if developers in the MIT hacker community are interested in joining in.

Similarly, getting honest feedback on a project is not straightforward. By posting ideas to hackers that the author doesn't necessarily know already, this is mitigated. Simultaneously, opening up feedback to the community allows students to get suggestions from a more diverse group.

Lastly, it's difficult to keep up-to-date with projects: a student might be interested in several projects that are being developed concurrently, and may not be able to keep track of all of the activity.

Concepts

Tag: a key category used to describe the type of the project. Tags make it easy to share and find projects that match a user's interests. Suppose a user has a project consisting of a website where MIT students can share their project ideas. The user may associate the "community" tag with the project to label the project as one that aims to support a particular community. Then, another student can search for the "community" tag and find the project.

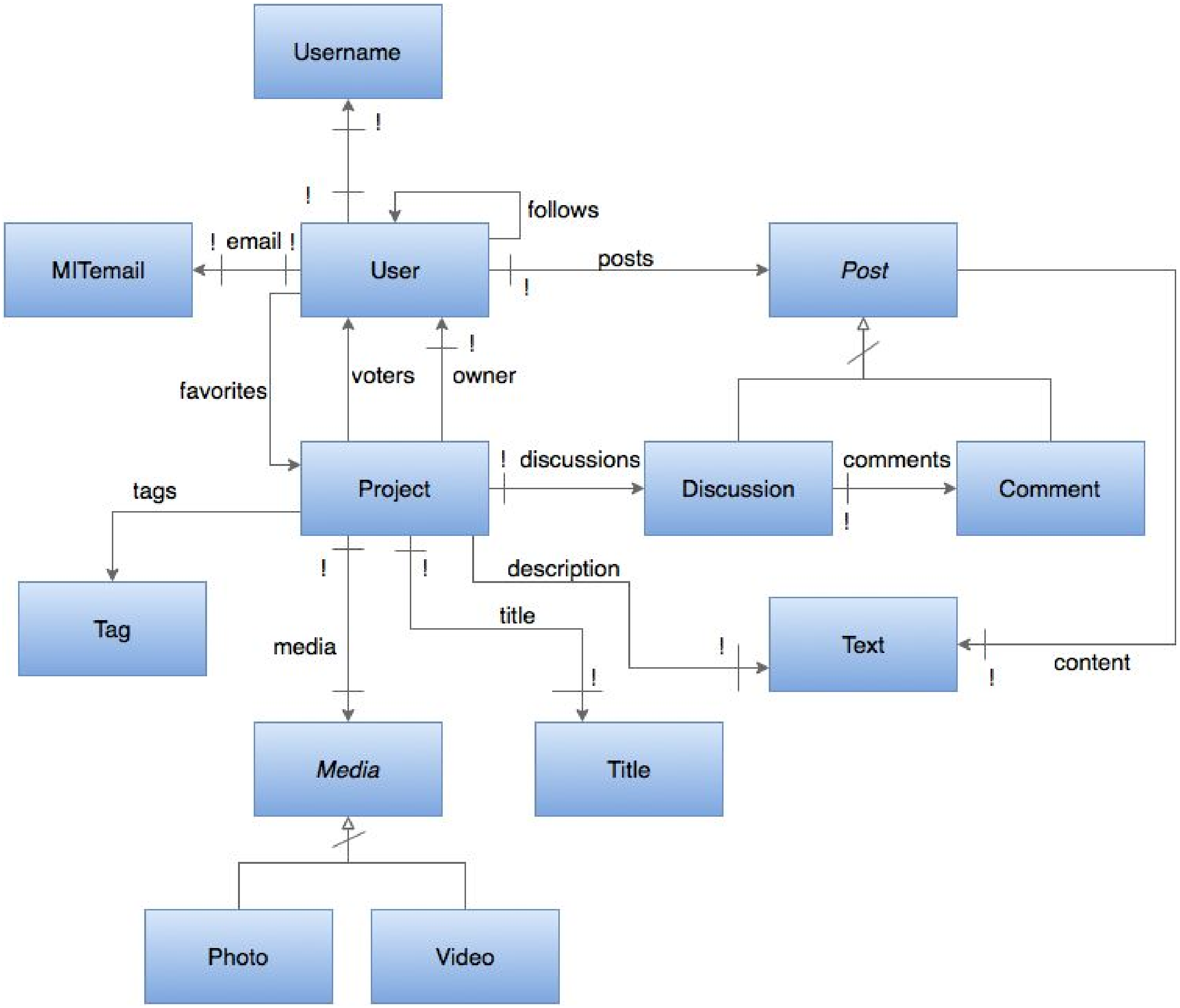
Post: a topic of discussion on a project. Posts enable users to provide feedback for a project, and to discuss other project issues. When a user finds a project and has some feedback, the user may make a post with the feedback. Other users can then visit the project and see the post.

Favorites: a set of projects that a user is interested in. It allows users to save/bookmark the projects they are interested in, so that they can easily navigate to the project when needed. A user selects Favorite on the project dashboard to add the project to the user's set of favorites.

Following: a set of users whose activities a particular user wants to keep track of. Following allows a user to get the latest activities of their friends or people they're otherwise interested in. Alice clicks the Follow button on the profile page of another user Bob. Then, when Bob creates a new project or makes a post, Alice can see the activity on Alice's Activity Feed page.

Upvote: a count of how many users have upvoted a project. Upvotes are aimed at increasing visibility of popular projects by providing a rating system for projects that people find interesting. When a user reads about a project and finds it interesting, they upvote it. Projects with the highest number of upvotes appear on the Trending Projects page.

Data Model



Textual constraints:

- Users cannot upvote their own projects

Explanation:

- Discussions are top-level threads that users can open on a project; once a Discussion is opened, users can reply with Comments

Insights:

- Users can have multiple posts within the same project

- We need to restrict email accounts to single users. Multiple users should not share an email. Since one can create multiple mailing lists, however, we can't ensure that users do not have multiple accounts.
- Users can choose whether or not to favorite their own projects.
- Projects can have the same description.

Security Concerns

Security concerns for HackTrack are largely the same as those for standard web applications.

HackTrack will be deployed on a platform-as-a-service (PaaS) provider like Heroku. We assume the provider is trusted, and we additionally consider any platform security issues (e.g. network-level denial of service attacks, physical datacenter security, remote access security) to be the responsibility of the provider (and assume the provider will fulfill that responsibility).

Furthermore, we use HTTPS, and assume that the PaaS provider will ensure that encryption software is up-to-date, so that attackers cannot spoof our service or intercept communications.

We also ignore any physical security on the user-side, where the attacker spies on the user directly.

Thus, in our threat model, the attacker only has the ability to make arbitrary HTTP requests to our application. We then address these general web application security concerns, to prevent an attacker from obtaining privileged access to the web application:

- Cross-site-scripting (XSS) attacks: we use a templating engine that escapes user content when the HTML page is generated. We respond with JSON to asynchronous requests, and ensure that any user content is sanitized in the JSON content on the server.
- Cross-site request forgery (CSRF) attacks: we include a token, stored in the session, to any POST, PUT, and DELETE requests made by the client, and validate the token in the application. We further ensure that GET requests do not affect the server state.
- Database injection attacks: we interact with MongoDB via the Mongoose library; user content is represented as variables in Mongoose model instances, and Mongoose handles data escaping when communicating with the database.
- Access control issues: to avoid accidental cases where access control is missing from some actions, we use routers so that any action that requires user

authentication will be placed under a router that includes an authentication middleware.
- Session management: we use the express-session library to handle sessions and trust the library.
- Outdated components: we subscribe to security updates for node.js modules used in our application, and update the modules routinely.

We also address several concerns in the context of our application, to deter attackers who do not directly target the application but seek to cause direct harm to our users, such as by exposing passwords or gaining access to accounts:

- Spamming: an attacker may make a large number of sign-up requests with various legitimate e-mail address to cause spam complaints; or, users may simply sign-up with someone else's e-mail address, causing that person to receive notifications from our system. We rate-limit sign-ups per IP address, and require e-mail address verification before the activating accounts.
- Username/password guessing: we rate limit login attempts based on IP address to mitigate username enumeration or password guessing.
- Database security: our database may become compromised if our PaaS provider is attacked, or we have a copy of the database in a backup system or on a development machine. We want to mitigate the fallout from an event where an attacker obtains a copy of the database. To do so, we hash passwords using a password hashing algorithm like PBKDF2 before storing the hash and salt in the database.
- Password reset form vulnerabilities: a password reset form is needed in case users forget passwords. To prevent attackers from spamming users by repeatedly issuing password reset requests, we rate limit password reset requests to one per day per username, and also require both username and e-mail address. To prevent attackers from resetting the password, we require e-mail address verification and rely on the security of SMTP infrastructure.

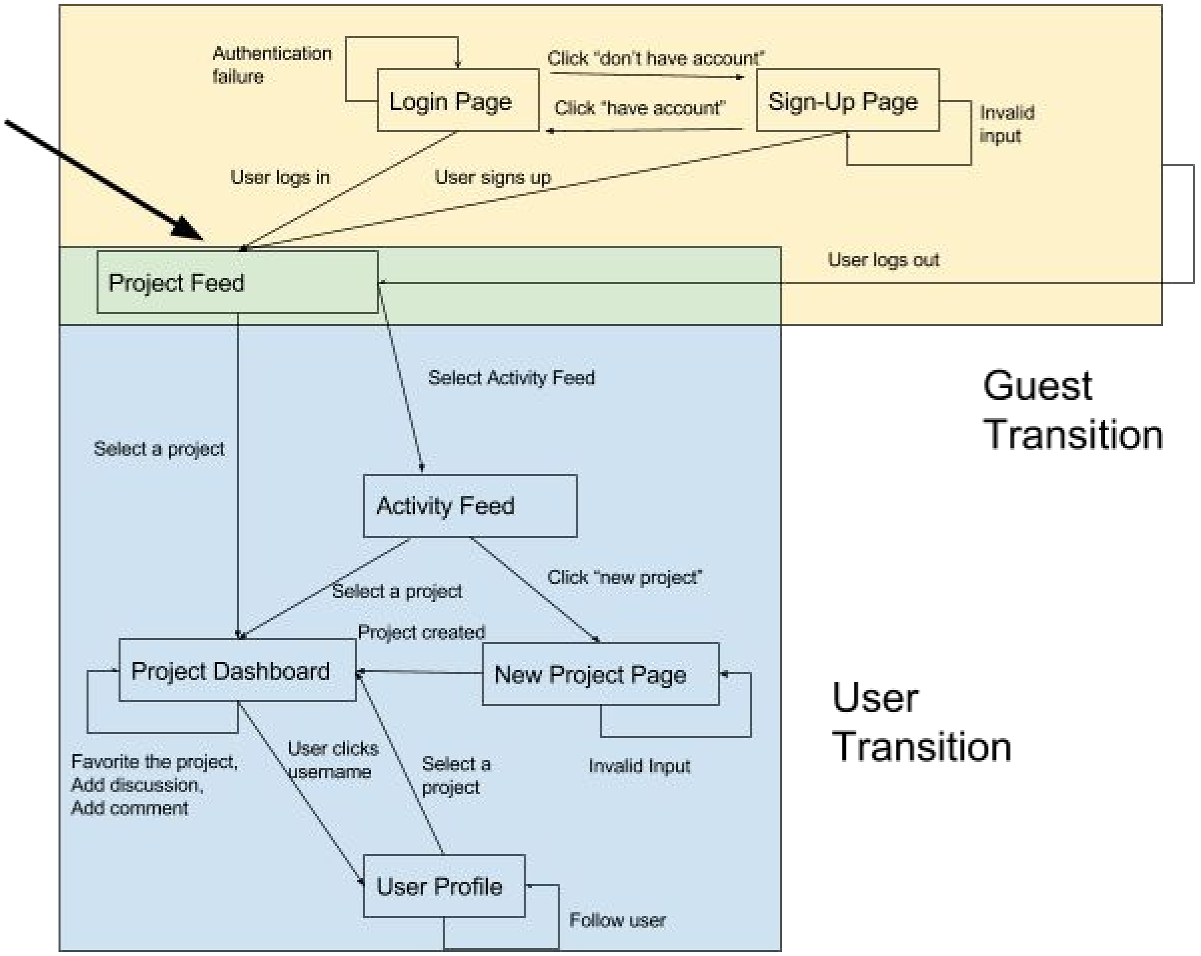
Finally, we address security concerns from attackers with more subtle goals that do not directly violate our application security, but still can indirectly harm our users:

- Some users may intentionally make off-topic offensive posts and comments with the goal of upsetting or pestering other users on the service. To deter such activity, we include a comment policy in the terms of service detailing what types of messages are not permitted. Additionally, we include an administrative interface where moderators can delete posts and comments that violate the comment policy, and warn, silence (i.e., block a user from posting), or ban offending users.

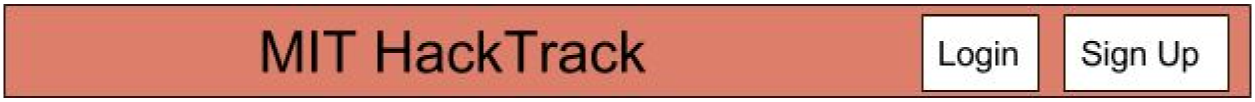
- Recruiters and others may try to collect e-mail addresses from HackTrack and spam our users with offers, advertisements, and possibly spam and phishing attacks as well. We hide e-mail addresses by default to prevent this activity, but users can still opt-in to having their e-mail addresses be displayed on the user profile. If the user opts in, we try to prevent web robots from scraping e-mail addresses from the website by displaying the e-mail address via JavaScript, although humans and some web robots would still be able to bypass this form of protection.
- Spammers may sign up for an account and begin posting advertisements in the form of projects, posts, or comments. We deter spammers by having a moderation time for new accounts, where the first couple projects and first ten or so posts and comments need to be approved by HackTrack staff before it is publicly displayed.

User Interface

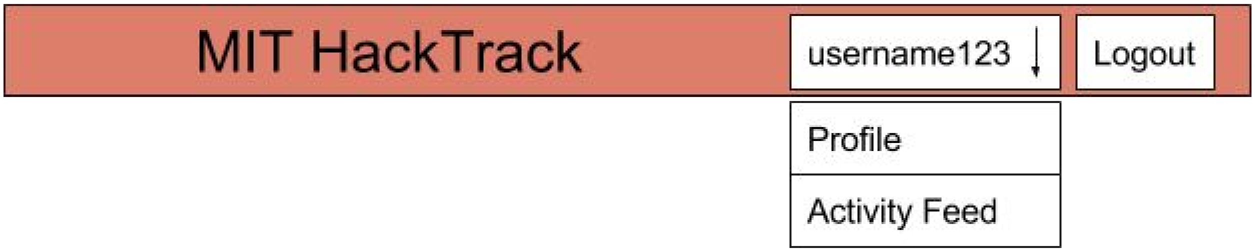
Transition Diagram



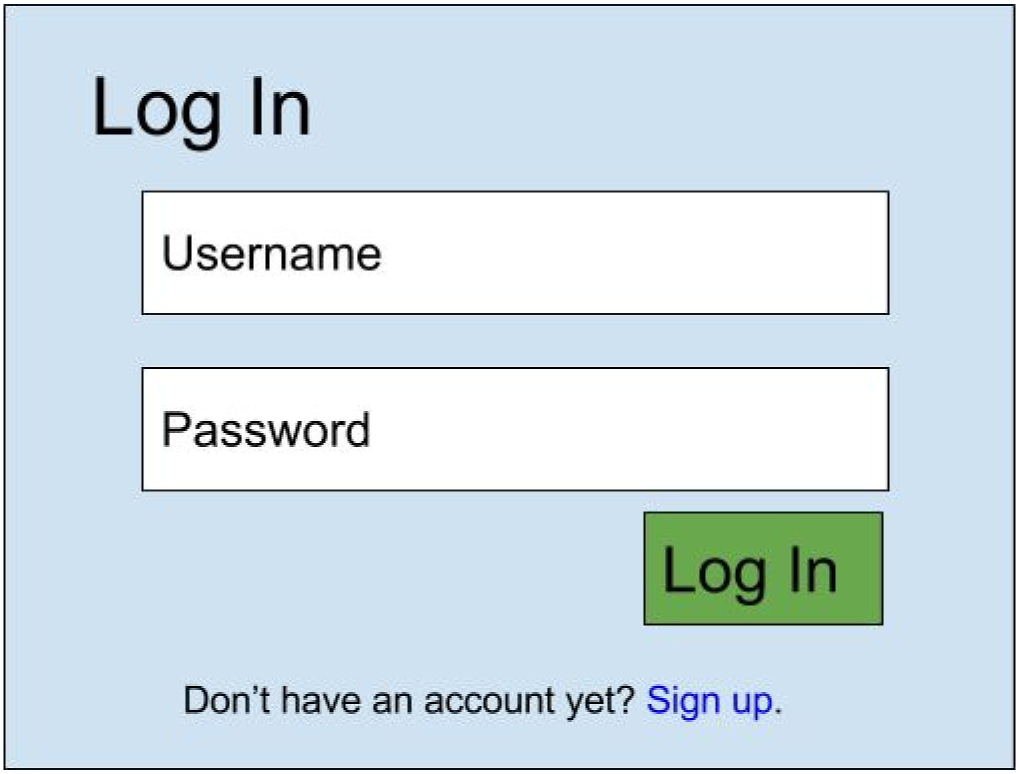
Guest Navigation Bar



Logged In Navigation Bar

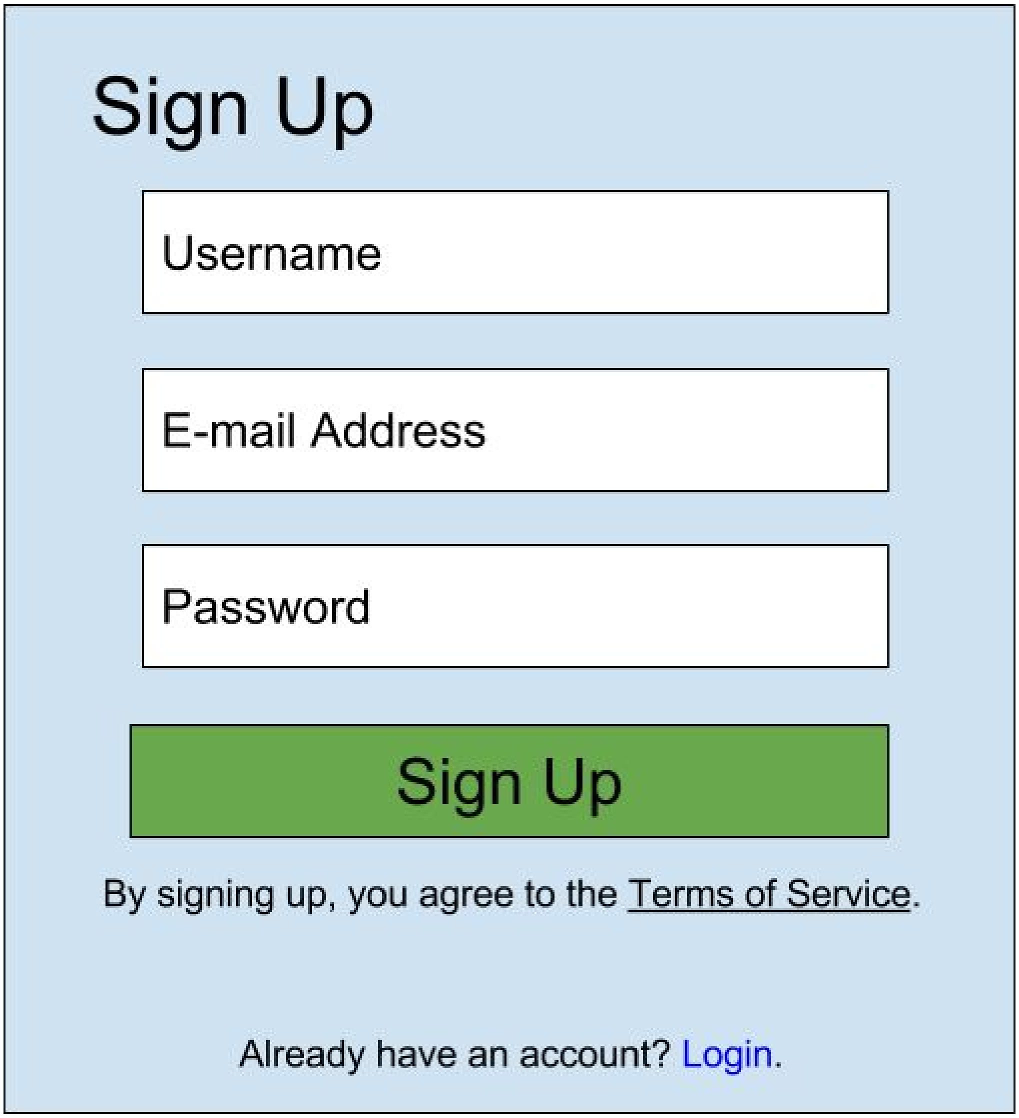


Login Page



Don’t have an account yet? Sign up.

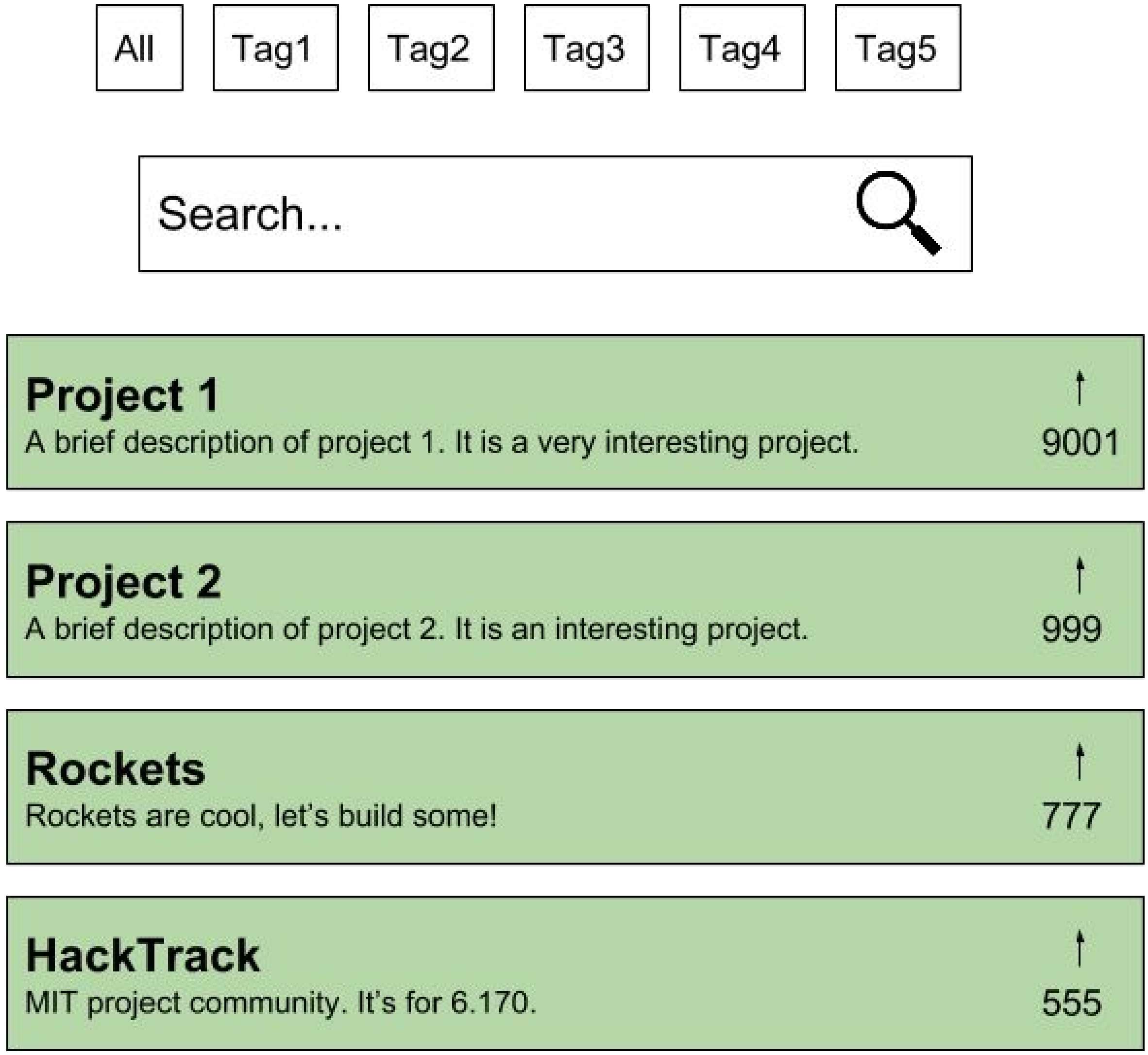
Sign Up Page



By signing up, you agree to the Terms of Service.

Already have an account? Login.

Project Feed



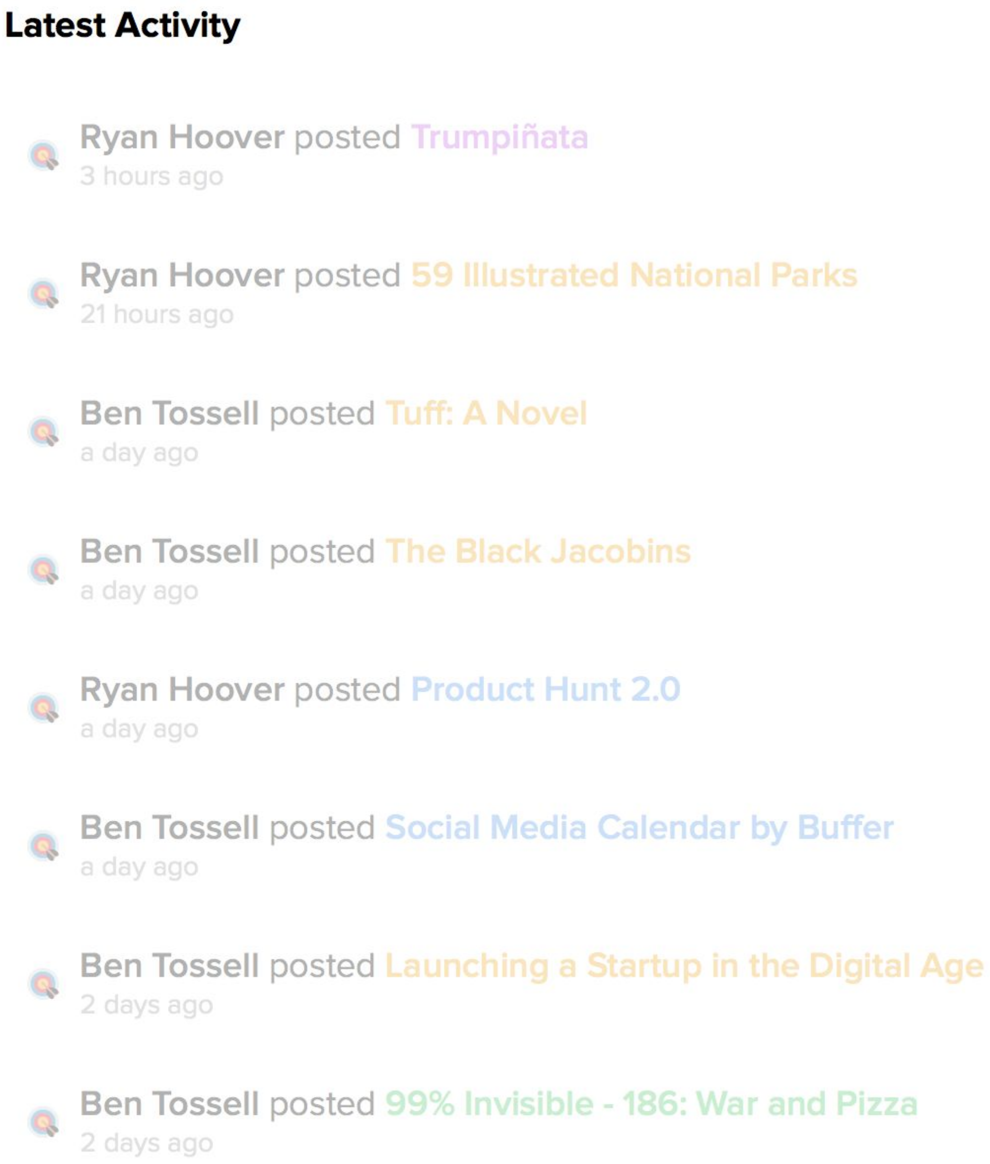
Project 1
A brief description of project 1. It is a very interesting project.
↑ 9001

Project 2
A brief description of project 2. It is an interesting project.
↑ 999

Rockets
Rockets are cool, let's build some!
↑ 777

HackTrack
MIT project community. It's for 6.170.
↑ 555

Activity Feed



Latest Activity

Ryan Hoover posted Trumpiñata
3 hours ago

Ryan Hoover posted 59 Illustrated National Parks
21 hours ago

Ben Tossell posted Tuff: A Novel
a day ago

Ben Tossell posted The Black Jacobins
a day ago

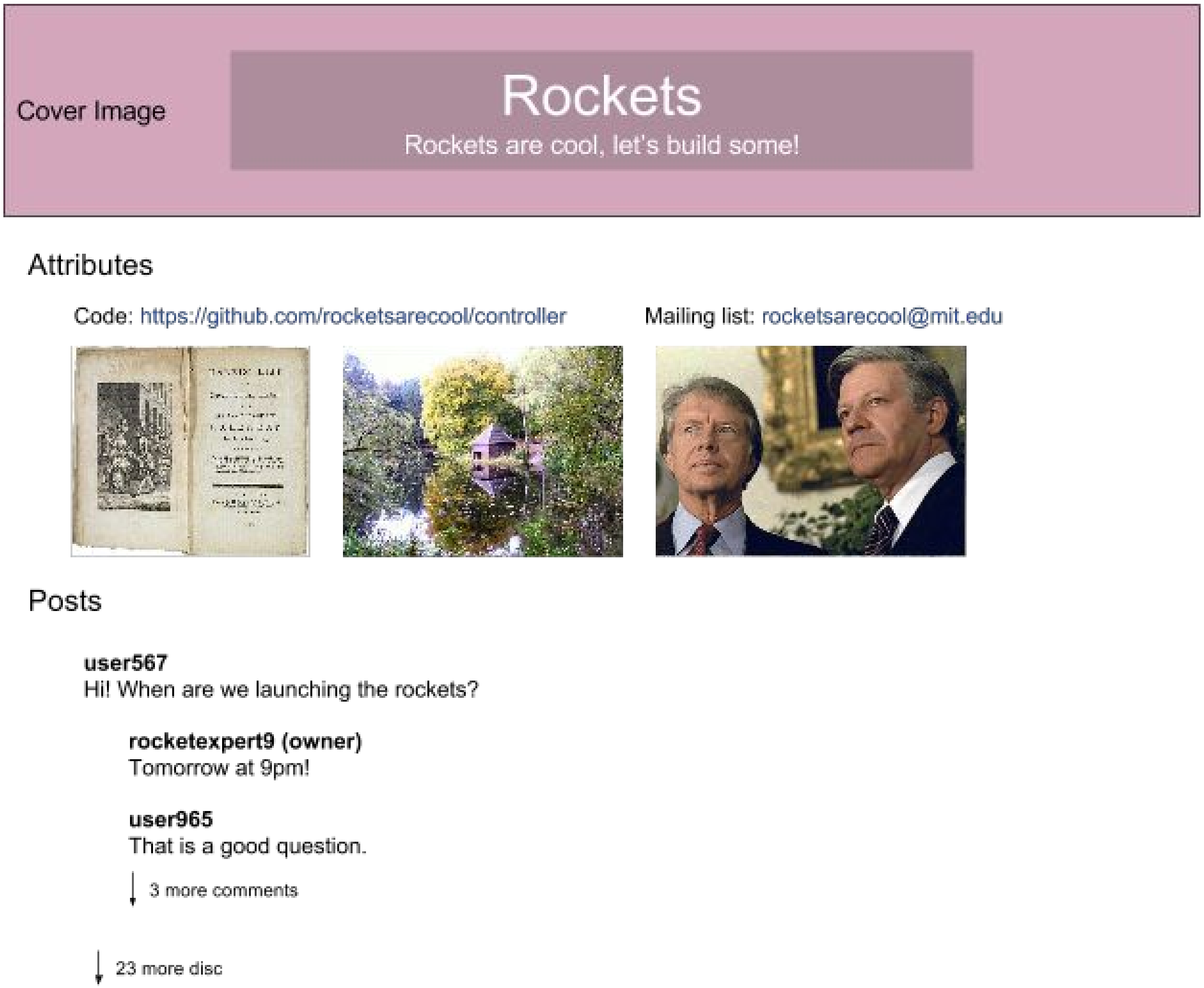
Ryan Hoover posted Product Hunt 2.0
a day ago

Ben Tossell posted Social Media Calendar by Buffer
a day ago

Ben Tossell posted Launching a Startup in the Digital Age
2 days ago

Ben Tossell posted 99% Invisible - 186: War and Pizza
2 days ago

Project Dashboard



Cover Image

Rockets

Rockets are cool, let's build some!

Attributes

Code: https://github.com/rocketsarecool/controller

Mailing list: rocketsarecool@mit.edu

Posts

user567
Hi! When are we launching the rockets?

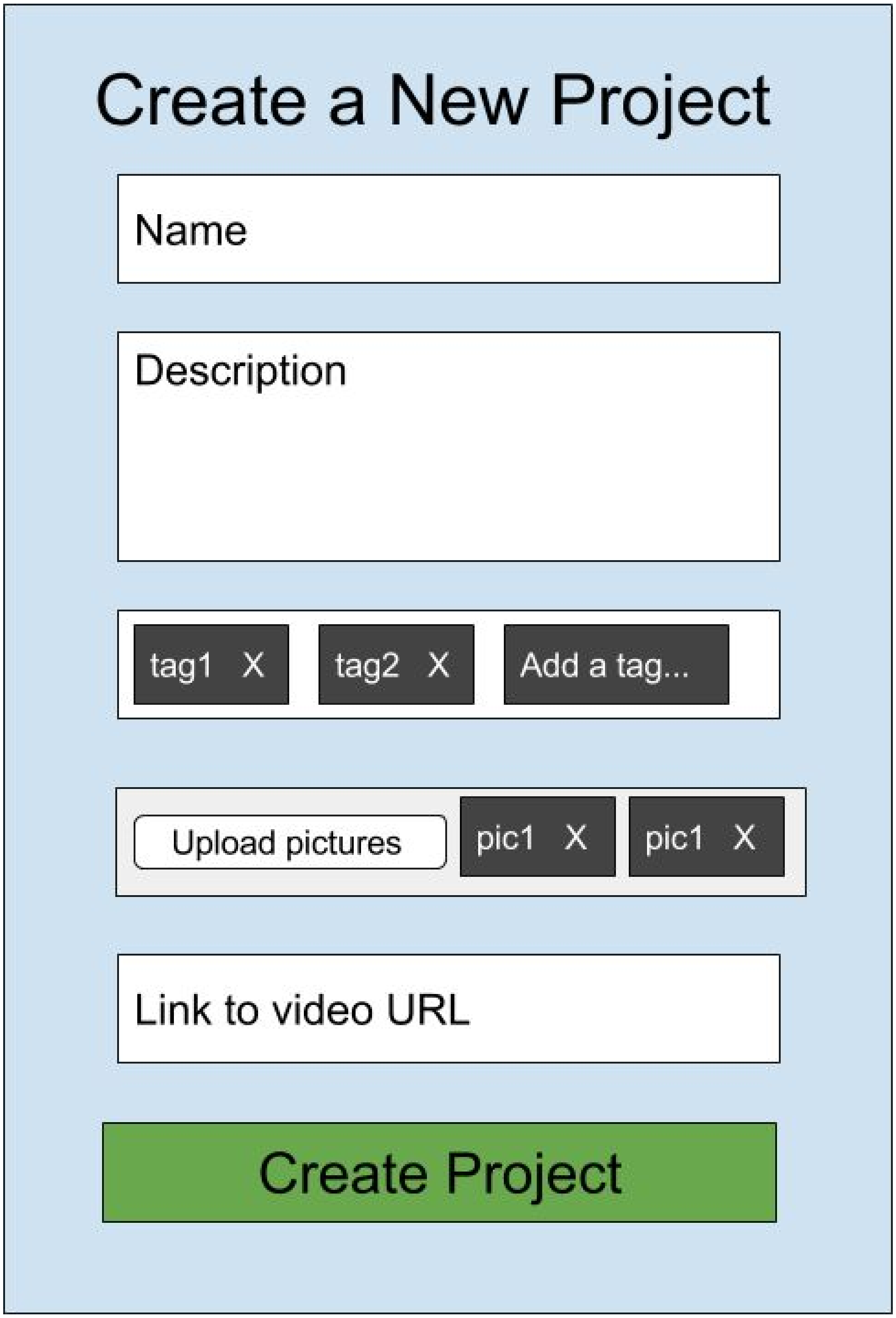
rocketexpert9 (owner)
Tomorrow at 9pm!

user965
That is a good question.

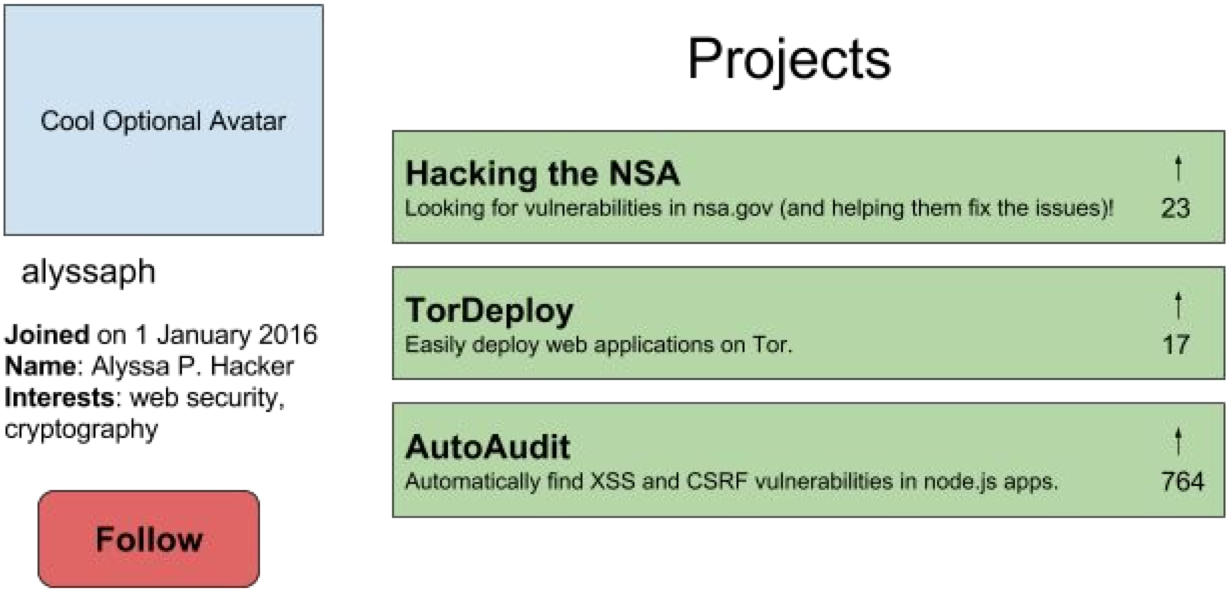
↓ 3 more comments

↓ 23 more disc

New Project Page



User Profile



alyssaph

Joined on 1 January 2016
Name: Alyssa P. Hacker
Interests: web security, cryptography

Projects

Hacking the NSA
Looking for vulnerabilities in nsa.gov (and helping them fix the issues)! 23

TorDeploy
Easily deploy web applications on Tor. 17

AutoAudit
Automatically find XSS and CSRF vulnerabilities in node.js apps. 764

Design Challenges

1. How to authenticate users as members of the MIT community?

Since we are limiting the use of this app to members of the MIT community, we think it's necessary to authenticate users as MIT members when they sign up. We considered the following options:

A. MIT certificates
   - ○ **Pro:** Authentication by MIT certificates would be a robust authentication method which is guaranteed to work so long as the user has a valid MIT certificate.
   - ○ **Con:** However, to implement MIT's certificate authentication, one would need an SSL packet and approval from IS&T for a signed server certificate.

B. Nodemailer
   - ○ **Pro:** Relatively easy to use and does not modify the User model. This method of authentication asks the user for a valid MIT email and sends an email with a verification link to the user's email. Clicking this link would then authenticate the user as an MIT student
   - ○ **Con:** Need to store users who have not yet verified their accounts and delete them after a certain deadline if they are not verified and otherwise move them to the user collection when their accounts are verified.

Team verdict: In the interest of saving time, we decided to go with the latter option, Nodemailer. We decided that the time it would take to get approval from IS&T could be unboundedly long. Nodemailer avoids dependence on external factors. To sign up, a user must enter their MIT email, a username and a password. The email they entered is then parsed to verify that it is an MIT email (alum email also allowed.) If it passes, an activation link is then sent to that email and the user must click on that link within 24 hours to be registered to our app. This ensures that the MIT email they entered is in fact their own.

2. Best way to present updates on projects that users are interested in

Our options:

A. If a user likes a project, they favorite it, which adds it to their list of favorite projects. New activity on that project will be shown in that user's activity feed. Additionally, users follow other users, which adds all of the followee's projects to that user's favorites.
    - **Pros**: If user A likes user B's projects, they don't have to individually favorite all of that user B's projects and if user B creates a new project that user A misses in projects feed, then they still get notified of that new project.
    - **Con**: Following users might form a conflicting mental model with favoriting projects and users might find it confusing. For instance, if user A follows user B, that implies that user A favorited user B's projects but if user A unfavorites one of user B's projects, it's not clear whether user A no longer follows user B.

B. If a user likes a project, they favorite it, which adds it to their list of favorite projects. When user A follows user B, posts made by user B will be shown in user A's activity feed.
    - **Pros**: External consistency. This model of following is similar to that of other popular apps.
    - **Con**: A user following another user still has to favorite all that user's projects if they want to bookmark them.

We decided on option B because it greatly improves usability in terms of creating a clear mental model for the user and have external consistency with popular apps. The only disadvantage that this has over option A is that the follower doesn't get all his followee's projects in his favorites list but this is mitigated by having the followee's posts appear in the follower's activity feed.

3. Where to store votes in model

Users vote on projects that they like. In designing the model, we considered the following options:

A. Have the User model store a list of all the posts the user has upvoted.
B. Have the Project model store all the users that upvoted it
C. Have the User store a list of their upvoted projects and have the Project only store its vote count to minimize number of queries made while displaying project.

After some deliberation, we decided to go with the option B: In terms of implementation, B is quite similar to option A but will be faster/easier to implement since the developer who will be working on Project won't require the API from the developer who will be working on User and so the two can work in parallel. Option C was eliminated because there runs the risk of inconsistency between the Project's vote count and the User's upvoted projects list. These risks stem from the fact that incrementing the vote count is an atomic operation that might be disturbed by asynchronous AJAX calls. Furthermore, option C was only considered because it has the highest performance but performance isn't a priority in this class.

4. Order in which to display projects

One main design challenge of this app is how to display projects to the users on the project feed. If projects are displayed in a sorted manner, i.e. by date and/or votes, some projects will never get their time in the sun. For instance, old projects with low votes will get limited visibility. We considered two possible solutions:

A. Sort all the projects by date (most recent first)
   - **Pros**: New projects are most visible. Projects with low votes still get shown at the top of the feed
   - **Cons**: Old projects get limited visibility
B. Sort all the projects by votes.
   - **Pros**: Most voted on project are the most visible. Old projects still get shown at the top of the feed
   - **Cons**: Low vote projects get limited visibility. This is slightly better than A
C. Projects are sorted by date (most recent first.) Additionally have a "Trending projects" page which shows x posts with the highest vote count, sorted by votes. Also have a "Featured projects" page which shows x posts with the lowest vote count, reverse sorted by date so oldest posts are shown first.
   - **Pros**: This method ensures the most fair visibility to all projects.

    - **Cons**: No incentive to visit “Featured projects” page. Implementation is more time consuming

D. Compound sort projects by date then by vote count such that all posts from one day are shown before posts from a previous day and within each day, posts are sorted by votes.
    - **Pros**: New projects are most visible. Projects with low votes still get shown at the top of the feed
    - **Cons**: Old projects get limited visibility

We went with option C because it performs the best in distributing visibility as equally as possible among the projects despite the fact that it will take the longest time to implement among the other options. For the MVP, we will not be implementing the “Featured projects” since the app’s core functionality does not require it.

5. Project owners vs Collaborators

Projects typically have more than one person involved. If someone is working on a project with other people and then posts the project to our app, should the collaborators on that project have edit permissions on that project’s page? How should the poster attribute credit to his teammates? We considered the following options:

A. Users add collaborators to a project which gives them the same access rights to a project’s description, features… as the original poster.
    - **Pros**: All collaborators receive credit. All are notified about comments
    - **Cons**: Potential asynchronous errors. If two collaborators are editing the same project at the same time, it is not defined whose edits will take effect.

B. Users optionally write down names of collaborators. The app doesn’t check if that user exists in the system.
    - **Pros**: All collaborators still receive credit.
    - **Cons**: Collaborators not notified about feedback on the project.

After some deliberation, we decided to go with option B. There’s only one project owner. This greatly simplifies implementation while still attributing credit to a project’s collaborators. We imagine that only in rare cases will all the collaborators on a project be signed up for the app which means implementing ‘collaborators’ would have taken a considerable amount of time while it would only be useful in very few cases.