

StitchHub Design Documentation

By National Badger: Karleigh Moore, Val Healy, Denis Li, James Usrey

Overview (Val)

Motivation

In our project, we have set out to build a social web application that allows users to create, share, save, and remix colorwork charts (design patterns) for knitting, crocheting, and cross stitching.

While many existing chart-making apps allow users to share patterns through other forms of social media, as well as email, none of them really acted as a social network in and of themselves. Many of these sites have clunky or aesthetically displeasing interfaces. Furthermore, only one of the applications allowed users to customize existing patterns on the site. We hope to resolve these issues in our application.

What does our application do?

Our application will focus on two primary purposes: the creation of designs and the creation of derivative designs based on existing designs.

The function of a design is to allow users to plan their stitching projects in advance so that they might use the finished design for reference while they are stitching project. Other sites allow the creation of designs, but we will improve the user interface. We will also introduce some new social features, such as a comment section for each chart and the ability to “like” charts. This will allow other users to give feedback to designers and allow designers to connect with each other through their designs.

We will also allow users to create derivative works based on existing charts. A user can use this feature to create custom designs based on existing designs on the site. Users can mutate designs made by themselves or by other users. They can also trace the genealogy of designs. This feature will supplement the social aspect of our app as well as allow users inspiration for their design and a chance to customize their favorite patterns to suit their own needs.

We also allow users to follow other users. This lets a user filter out to see only the charts of those that he/she follows.

Concepts (All members contributed)

Name	Purpose	Operational Principle
Chart	Provide a visual representation of the design for the user	When a user creates a new chart to share on StitchHub, the user initializes it by specifying a size and title. The user can then select cells in the chart in order to assign colors to the particular squares in the chart and create a design. The user saves the chart when finished and this saves the chart to the database and allows the chart to appear in the public feed so other users can see (and choose to remix) the design. Users can comment on charts and like or save any chart but their own.
Remixing	Provides users a way to take existing designs and edit them to their liking, and then share the new design.	When a user creates a design, it appears in a feed visible to all users. Other users can click to remix an existing design. By remixing an existing design, a user can edit the colors on the original design and make and save a new design (similar to a retweet you can edit).

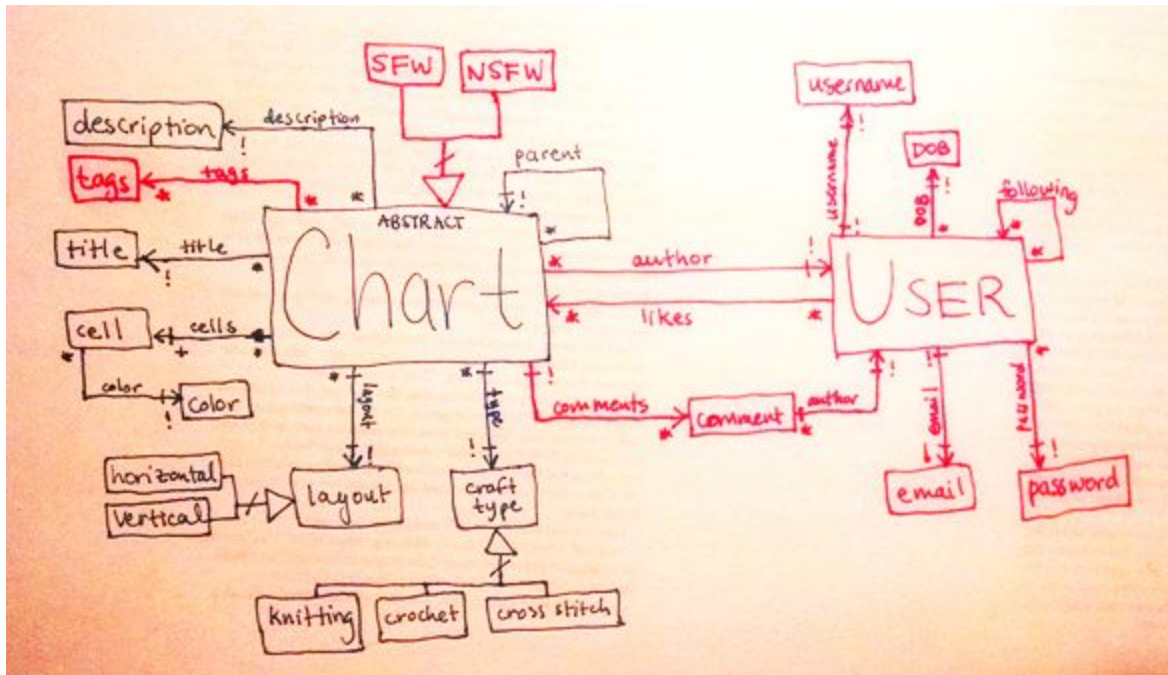
Additional concepts we plan to include that are at least mostly self-explanatory are (we still provide a brief explanation for those that are not entirely clear)

- Description
- Tags
- Following
- Liking
 - The standard like. However, users may also see a list of their liked charts.
- Searching (by title, tags, etc.)
- Sorting (by date, most liked, etc.)
- Comments

Anticipated Misfits:

- One misfit found in almost all of the existing apps is incorrect cell aspect ratio for different chart types. Knitted stitches have a different aspect ratio from crocheted stitches or cross stitches. We will remedy this by allowing for different aspect ratios, each paired to a chart type (knitting, crocheting, cross stitching).

Data Model (Val)



Our data model is pictured above. The blue elements represent the sets and relations of our MVP. The pink elements will be added for our final project.

NOTE: we may change password to be mutable, if time allows. (see [related design decision](#))

Textual Constraints:

- A User cannot follow themselves
- A Chart cannot be its own parent

Design Choices (we have updated this whole section)

Concern: Can two users have the same email address?

Decision Made

This concern is roughly the same as asking whether or not we allow for multiple accounts with a single email address. We do not feel that there is a need for a user to have multiple accounts. We also believe that having multiple accounts can cause rigging in the number of likes. As such, we mitigate the issue by deciding on *not* allowing two users to use the same email address.

Concern: How can we handle plagiarism?

Choices Considered

1. We considered having a reporting system where a user could make a report on a chart they thought was plagiarized. However, we spoke with our TA and realized that we weren't sure how to define "plagiarism" in certain edge cases. For example, if a user copies an entire chart but then adds one row of empty squares to the bottom — these are technically distinct charts, but one still feels that something is wrong.
2. A second option we considered to handle plagiarism was to simply have all users agree to have their charts listed as open source. This is part of the site's terms of service. This way, plagiarism is mitigated because users forfeit their right to claim plagiarism.

Decision Made

As suggested by the TA (Vinay), we will make every user of StitchHub agree to making their work open source. Therefore there is no plagiarism within our site. However, it is still possible that a user plagiarism another site/infringe on copyright issues from external content. For these scenarios, we still allow companies to report to us such violations for manual confirmation. Charts with violations will be removed.

Concern: Should a user be allowed to edit his/her chart after finishing?

Choices Considered

- User is allowed to edit their chart after creating/remixing it. This gives more freedom to the user. However, if the Chart was remixed by another user beforehand, then the remix now has a misleading parent.

- User cannot edit their chart. Users have less freedom, but we can now guarantee that parents of a remix accurately reflect what the remix was based on.

Decision Made

Users are not allowed to edit their chart after finishing it. We reason that if a user wants to edit their chart, then they might as well remix their own chart.

Concern: Can a users modify the tags of another user's chart?

Choices Considered

- User can modify another user's chart's tags. This is useful in cases when the creator of the chart forgot/was unaware of certain tags. Having another user modify it could help improve the searchability of a chart.
- User cannot modify another user's chart. This is a safer alternative. A troll could just as easily reduce the searchability of a chart or provide incorrect search results by tampering with the tags of a popular chart.

Decision Made

A user cannot modify another user's chart's tags. We believe that we should not take risks to make the application more "powerful". If a user wants the same chart to have better tags, then they will just remix the chart and update those tags. A troll can still provide irrelevant tags to the chart they just remixed. However, it is unlikely that a chart with ridiculous tags will end up having a high like count; therefore it will not be high up in the search results filtered by number of likes and will not be exposed to many users.

Concern: What types of information about a Chart can a user change after posting it?

Choices Considered

Allowing a user to edit chart titles, descriptions, and tags will allow the users more freedom. However, this could also cause some confusion when users search for titles of charts that they remember, but whose title has been changed in the interim.

Another disadvantage of this is the potential for trolling by users. A troll user could create a chart that gains a high popularity. The troll user could then change the various metadata about the chart to be offensive. This would make it appear that users liked an offensive post, (although this could be mitigated by through the use of the un-like button).

Decision Made

After lengthy argument on the mutability of each of these fields, we have agreed on the “Reddit mutability”. This means that the title is immutable, description is mutable, and tags are mutable. There is no particularly good reason for this decision. We have simply arrived at the conclusion that it would be difficult for the team to agree on which fields should be immutable or not. As an example, one argument for making title mutable is user freedom, and argument for title being immutable is that users would be able to search charts by title without worrying that it changed. Both of these were reasonable arguments. Thus, we simply agreed to the mutability schema that a popular site has (in this case Reddit).

Concern: Should we include nicknames for users? If so, which will we display?

Note that the notion of nicknames is that nicknames for two different users can be the same. Otherwise users might as well use their username as nickname and hope that nobody else took it first.

Choices Considered

- We allow for nicknames. This gives users the option to display their own name. In this case, nickname must be displayed (because otherwise what’s the use). This means either only the nickname is displayed for a user, or the nickname and the username is displayed.
- We do not allow for nicknames. We already have usernames, so nicknames are sort of an unnecessary bonus.

Decision Made

We decided that users would not have nicknames. Suppose that users do have nicknames. Then we would either need to display just the nickname or both username and nickname for when a user comments. If only nickname is displayed, then it can become confusing to users when two share a nickname. Thus, using usernames eliminates this problem, as usernames must be unique.

Concern: Should we allow remixes to be exactly the same as the parent?

Choices Considered

If we allow remixes to be exactly the same as parents then this would result in “plagiarism”. Even if we do not consider this to be plagiarism, it seems strange that a user would remix to design exactly the same pattern.

Decision Made

We will allow remixes to be exactly the same as the parent. As per the decision we made regarding plagiarism, an exact-copy remix would not be considered plagiarism. Even though an exact-copy remix is generally quite useless, we cannot find a good reason to disallow it.

Concern: What is the maximum size of a chart (if any)?

Choices Considered

There are really many choices to consider here. But there are two main ones to consider: limited size and unlimited size. We want to avoid unlimited size since that could result in huge memory consumption and performance issues. Still, we want to allow for a reasonable maximum size so that users can freely design their chart.

Decision Made

We have decided on a maximum size of 512 x 512. This is a somewhat arbitrary size, but chosen so that there would be few performance issues as well as being big enough to design most patterns.

Concern: Charts can be too large to easily fit on one screen, or too small to easily edit for some users. How should we rectify this?

Choices Considered

- No zooming
- Buttons for zooming in and out
- Slider with zooming (many discrete options)

Decision Made

We have decided to allow for many discrete zoom states via a slider. More options here seem to be better than fewer. We will limit the minimum (~0.2x) and maximum zoom (~3.0x) so that a chart cannot be too small or too big. If the user requires further zoom settings, then they can use the browser's built-in zoom-in/zoom-out settings.

Concern: What types of search/sort/filter options should we allow for in Chart search and Dashboard?

Choices Considered

- Sorting by date
- Sorting by number of likes
- Sorting by color
- Sorting by recent popularity (some metric based on the number of likes and date)
- Filtering by size
- Filtering by craft type
- Filtering by color
- Search by title
- Search by author
- Search by description
- Search by tags

Decision Made

For searching: We will allow users to search by both author, title, and tags. Searching for description could be useful, but we feel that it would be too non-performant to do. It would require that we go through each Chart and split the description into words.

For sorting: We decided to allow for sorting by date, number of likes, and recent popularity.

For filtering: We will allow for filtering by size and craft type. Filtering by color was excluded in our decision since it would be too difficult to filter on the colors.

Concern: Which user information (username, password, and DOB) should a user be allowed to change?

Decision Made

We decided that username and DOB should not be mutable. This seems to be fairly standard.

There does not seem to be a good reason for the user to be able to change them.

We believe that a mutable password is a good idea. If a password is too insecure, then a user should be allowed to make it more secure. Also, if a password ends up being too complicated, then the user should be able to make it easier for him/her to remember. Still, it is not entirely trivial to implement password changing. Therefore we decide to implement immutable password first. If time permits, we will aim for mutable password.

Concern: For charts that are remixed from another, what information should we display regarding the “ancestor” charts?

Choices Considered

We considered two options: linking only the parent chart, and listing links to all charts in the “ancestry” of the chart.

Decision Made

We decided to link only to the parent chart. Listing links to all charts in the “ancestry” is certainly a nice feature, but it seems impractical to implement. Either we make multiple queries to MongoDB to fetch the entire ancestry, or we store the list of ancestors into each chart. This becomes a tradeoff between runtime and space. Furthermore, providing a link to just the parent chart still allows us to see the entire ancestry as long as the user goes through each link.

Concern: What should happen when a user deletes their chart?

Choices Considered

- User cannot even delete the chart.
- All charts are derived from the deleted chart via remixing would be deleted.
- Descendant charts of a deleted chart would still mention it as an ancestor with the name “[Deleted]”. However, there is no link associated with that chart.
- A reference to the deleted chart is still there. However, the chart’s page will not have much information. In fact, the only information on it will just be a link to that chart’s parent (if any). The title of the chart will be renamed to “[deleted]” to indicate that it was removed.

Decision Made

We have decided to keep a reference to the deleted chart. We think it would be unfair to a user if they could not delete an embarrassing chart that they made. Not to mention, we need to delete charts that violate copyright. Therefore we allow deletion. It would also be unfair if derived charts of a deleted chart are also deleted. Therefore we do not allow that either. We also would like to keep a trace of where a chart came from as much as possible. Therefore children of a deleted chart will still have a link to the deleted chart's page, which should then provide a link to the deleted charts' parent's page (if one exists).

Concern: Should we have separate save and like features?

Choices Considered

- Have separate save and like features. In this case, save feature allows users to revisit a collection of saved charts. The like feature simply measures how many users thought the chart was good (similar to "upvoting" on Reddit or "liking" on Facebook).
- Consolidate save and like features into one feature (which we will call like). Basically, whenever a user likes a chart, it gets added to a collection of charts that the user can revisit.

Decision Made

We decided to just have a single merged feature-- "likes". It is entirely possible that a user could want to save a chart that they do not like. Still, we believe that most users will tend to only want to revisit charts that they like. We do not want to make it a hassle for our users to have to click twice. Furthermore, this simplifies the user interface a bit as well.

Concern: Should we provide consistency between the number of likes a chart has and the number of users that have the chart in their collection of liked charts?

Background

This concern is a direct result of our decision to [merge the save and like features](#). We have thought through the implementation consequences and believe that it would require significant work to actually guarantee that the number of likes a chart has matches with the number of users that have the chart in their collection of liked charts.

There are many ways to implement the feature.

1. Have a Chart and User schema. The Chart model has a pointer to a list of Users that liked it. This would be easy to count the number of likes that a chart has. However, it would be slow to retrieve a list of charts that single user liked.
2. Have a Chart and User schema. The User model has a pointer to a list of Charts that the user liked. Here the list of charts a user liked is easy to fetch, but the number of likes that a chart has is hard to count.
3. Have a Chart and User schema. The Chart model has a pointer to a list of Users that liked it and the User model has a pointer to a list of Charts that the user liked. Here, it would be easy to count the number of likes a chart has as well as fetch the list of charts a user likes. Note, however, that the Chart model no longer actually needs to store a list of Users. The better alternative is the next implementation

The 3rd implementation is more efficient than the first two implementations. However, in the first two implementations, a user like corresponds to only a single MongoDB operation. In the 3rd implementation, two MongoDB operations are required (add a User to list of users who like the chart and adding a Chart to list of charts a user likes). This is the issue. It means that a failed operation on one would require the other to not occur. As it turns out, there is no good way to do this other than to rollback. This means that we need to implement a two-phase commit algorithm which [turns out to be possible](#) in MongoDB.

Decision Made

We would not like to have the inefficiency of the first two implementations. They do not scale well with the number of users or charts. Therefore we decide on the 3rd implementation. Note that, from our experience, it is unlikely that MongoDB will have an error anyway. Also a slight miscalculation is hardly detrimental.

Our decision is therefore dependent on how far we can get. On the first implementation, we will stick with the 3th implementation, but do no correction. With more time, we will try to implement the two-phase commit algorithm to ensure consistency.

Features We Might Implement If We Have Time

These are additions that would be nice to have, but are not necessary for the application. Given more time, we might choose to implement these features.

Disallowing Profanity in Usernames

We want to avoid profanity as much as possible. Therefore we would like to disallows users from choosing usernames that contain profanity. This is a nontrivial task since there are many profane words and many more ways to disguise them.

Note that we do not disallow profanity in descriptions since we can mark such charts as NSFW.

We do not disallow profanity in comments since we do not want to take away a user's right to make such comments. Our site is not strictly for children.

Fancy Drawing Tools

We can implement additional drawing features to facilitate pattern design as much as possible. Some features we can include if/when we have time are: pencil (mouse-down to color the cell), flood-filling, and dragging a filled-rectangle. We might also allow users to import an image file to initialize a chart for designing.

Advanced Searching

This includes a variety of additions to the search feature. We could include fuzzy word search (to handle typos). Also autocomplete for quick input of tags.

Chart Liking Consistency

This feature is a direct consequence of the [consistency concern](#). To fix the inconsistency, we can implement the [two-phase commit algorithm](#) suggested to get rid of the issue completely.

Chart Ancestry Feature

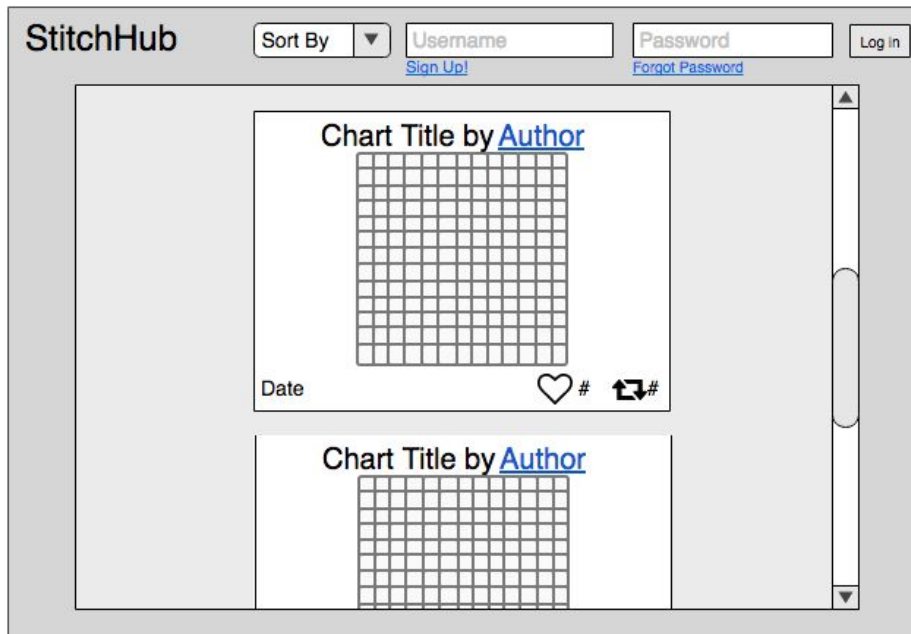
We are considering implement a chart ancestry tree for each chart, which would show parents and children (and possibly other generations) of a chart. Because a key feature of our product is remixing, we see value in showcasing it; this is one way of doing so. However, this feature is not necessary for the app to function. We are already including a link to a chart's parent on its page (if applicable). We also recognize that this feature is a real challenge in edge cases (what happens if a chart has 50 children? How do we display that?).

Design Risks (Denis)

Risk	Solution
Plagiarism	We will have in our terms of service that once a user posts a chart on the site, it becomes open source for all other uses mitigating claims of plagiarism right off the bat.
Copyright (e.g. creating content like Pokemon)	Companies that find copyright-infringing material will report the material to us.
Inappropriate content	Users can flag content as inappropriate in which case we will not display it to younger users.

User Interface (Karleigh)

Landing Page (Not logged in)



The landing page features a header with the "StitchHub" logo, a "Sort By" dropdown menu, and input fields for "Username" and "Password". Below these are links for "Sign Up!" and "Forgot Password", and a "Log In" button. The main content area displays two identical chart entries. Each entry has a title "Chart Title by [Author](#)", a 10x10 grid, a "Date" label, and icons for a heart and a share symbol, each followed by a hash symbol (#). A vertical scrollbar is on the right side of the content area.

If a user clicks "Sign up!" they're redirected to the following page.

Sign Up Page

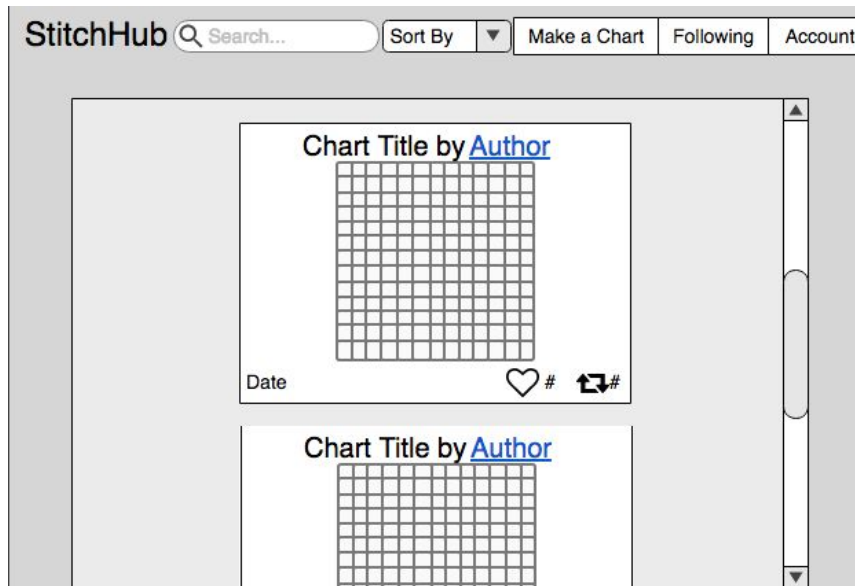


The sign-up page has a title "Sign Up For StitchHub". On the left, there are six input fields: "First Name", "Last Name", "Username", "Email Address", "Password", and "Confirm Password". Below these is a "Sign me up!" button. On the right, there is a text area with the placeholder text "Text about what you can do with a StitchHub account".

Once the user logs in, they are directed to the home feed.

A user will immediately see the public feed but can click on a particular user to see only that user's posts by going to that user's profile.

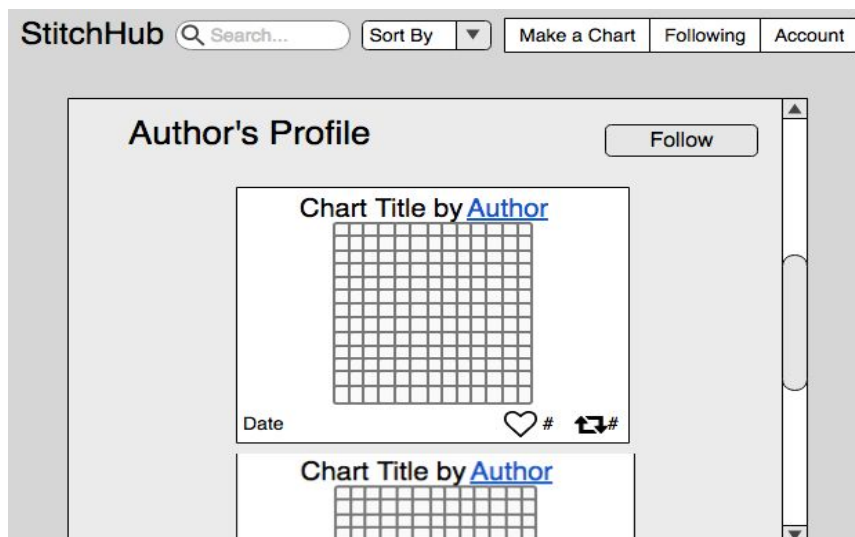
Landing Page (Logged in)



Clicking on “Following” will show users a feed populated by posts made only by users they are following. This will look similar to the home feed view-wise — we will just have some backend stuff filtering which posts make it to this feed.

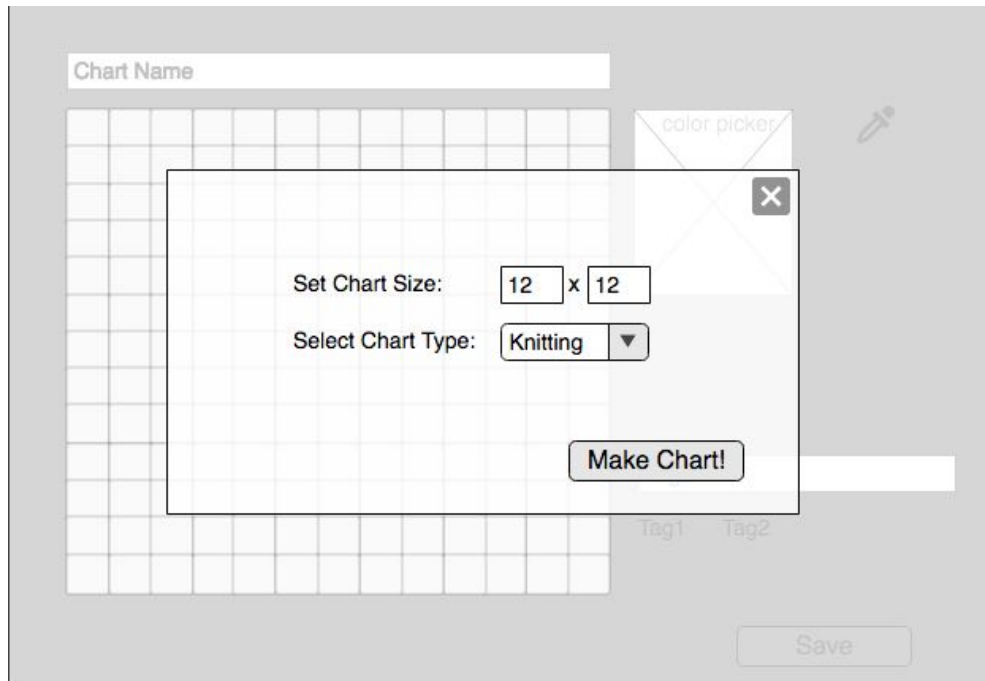
We can get to the user profile page by clicking on the author’s name.

User Profile Page



When users click “Make a Chart”, they are taken to the following page where they need to specify the type and size of the chart they want.

Chart Creation Page

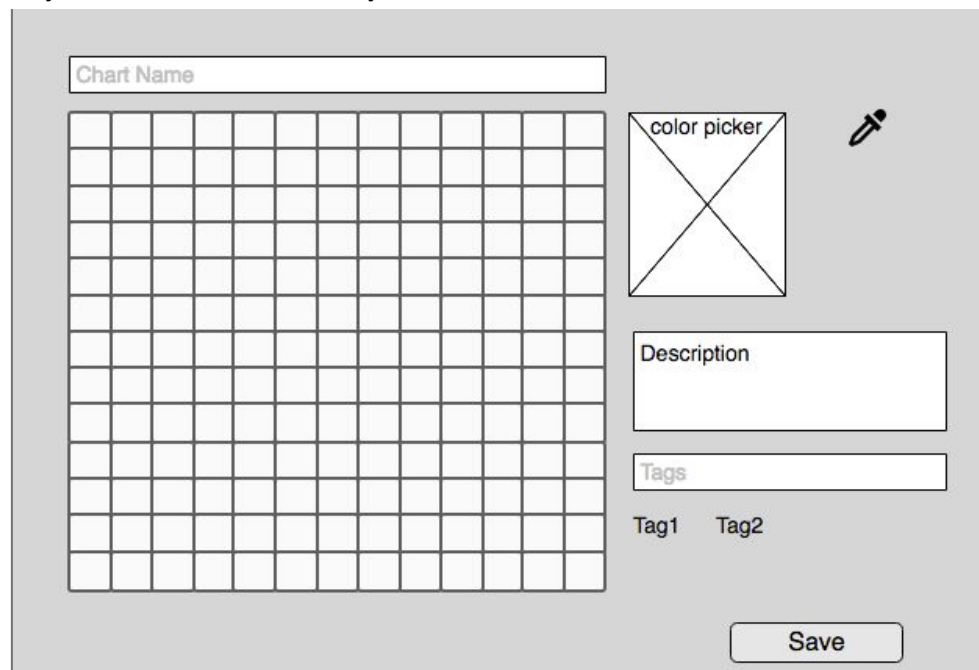


The image shows a web interface for creating a chart. At the top, there is a text input field labeled "Chart Name". Below it is a large grid. To the right of the grid is a "color picker" icon (a square with an 'X') and a pencil icon. A modal dialog box is open in the center, containing the following elements:

- A close button (X) in the top right corner.
- Text: "Set Chart Size:" followed by two input fields, both containing "12", separated by an "x".
- Text: "Select Chart Type:" followed by a dropdown menu showing "Knitting".
- A "Make Chart!" button at the bottom.

Below the grid, there are labels "Tag1" and "Tag2" and a "Save" button at the bottom right.

Once the user specifies the size and type of their chart, they are taken to the following screen. If they click the “X” button, they are taken back to the home feed.



The image shows the same web interface as before, but the modal dialog is closed. The "color picker" icon now has a large 'X' over it. The rest of the interface remains the same:

- "Chart Name" input field.
- Large grid.
- "color picker" icon with an 'X' and a pencil icon.
- "Description" input field.
- "Tags" input field.
- "Tag1" and "Tag2" labels.
- "Save" button at the bottom right.

The user can modify the chart by selecting colors from a color picker and using an eyedropper-type tool to determine existing colors they've used (in case they forget). Once a chart is posted, the user is taken to the chart's page.

If the user clicks on an existing chart (that they find on someone's profile or in their feed), they are taken to a page where they can see the chart more closely and add comments.

Chart Page

StitchHub

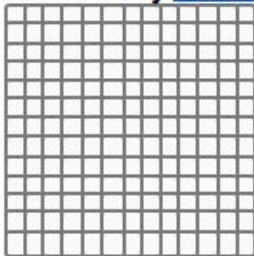
Sort By ▼




Make a Chart

Following

Account

Chart Title by [Author](#)



Date   #  #

Description

Tag1 Tag2

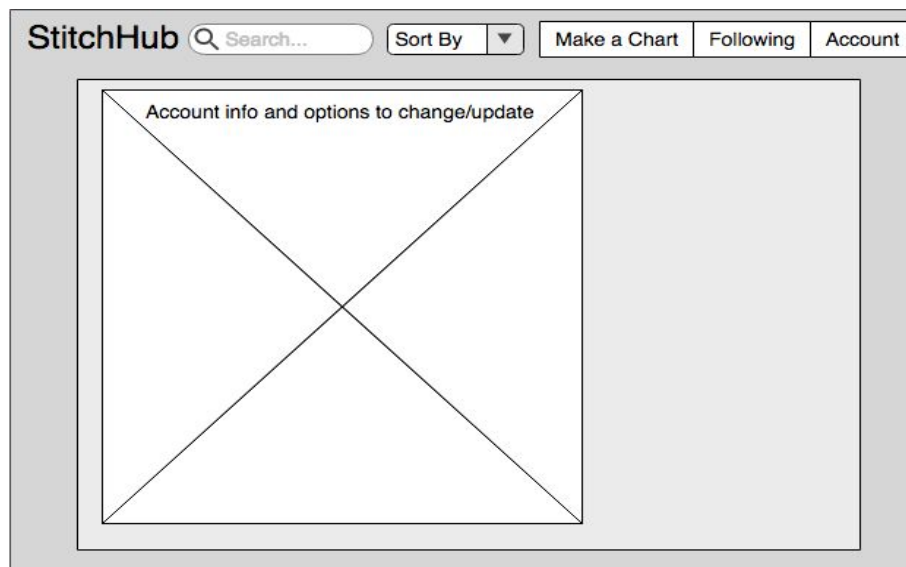
[Report Chart](#)

Comment Feed

Make a comment

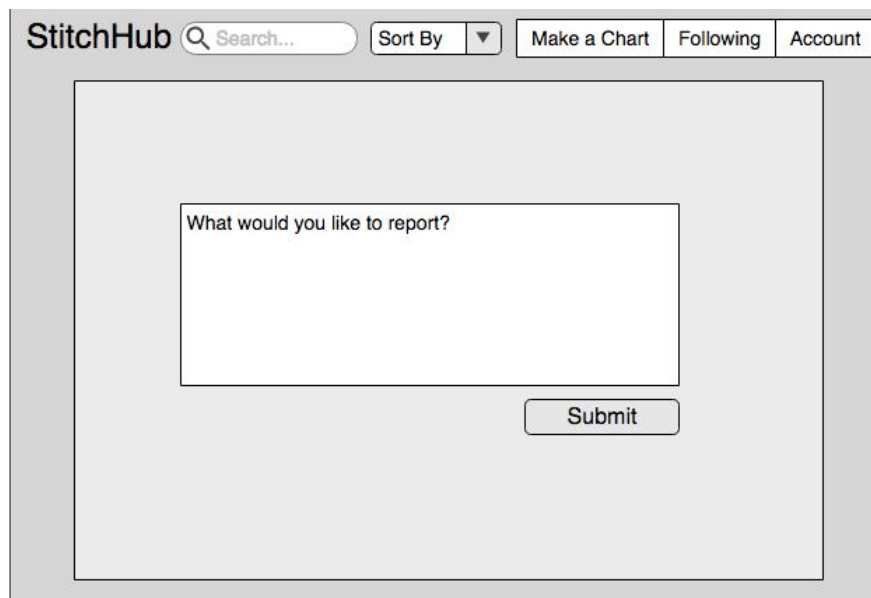
Post Comment

Account page



Report page

If a user clicks to report a chart, they are taken to the following page.



Security Concerns (James)

Key Security Requirements

Requirement	How to satisfy requirement
Protect user information like email, DOB, and password	Users must be logged in to access private information or make posts. This will be enforced by authentication checks during the necessary requests. To prevent members of the team from stealing password information, we will hash and salt them via libraries like crypto. However, the email and DOB are unprotected from us. We will have the user agree to providing us with this information.
Users should be able to log in using the username and password they specified when they registered their account.	The server should properly authenticate these credentials, and use them to enforce security constraints.

Mitigating Standard Web Attacks

Injection attacks are mitigated through the sanitization of user-generated input prior to construction of server requests. Malicious users will not be able to inject code into queries to obtain private information.

Cross-Site Scripting (XSS) attacks are mitigated through the sanitization of user-generated input prior to adding it to the web page. This will prevent malicious users from embedding javascript into our pages via the various user-generated input (comments, titles, etc.).

Cross-Site Request Forgery (CSRF) attacks will be mitigated through the use of [csrf](#) node package. We use the library to generate an x-csrf-token. We store this token in the client-side HTML. For every POST, PUT, or DELETE request sent, the secret x-csrf-token would be sent and verified by the server--allowing us to prevent requests from unauthorized locations.

Threat Model

What assumptions we're making about attackers

- Assume they want passwords to take over user accounts.
- Assume they want to delete the database.
- Assume they want access to email addresses (can be used to spam, or used in addition with passwords to potentially hack other sites).
- Assume that members of the team want the passwords of users.