CSE 416 Section 2 Team Purple: Jenny Bao, Aaron Li, Jonathan Ng, Angelo Panopio

Executive Summary

Existing applications such as https://www.mapchart.net/ already provide a lot of the services we would like to offer in our application. However, existing applications do not have a dedicated section for community on their websites. Instead, it seems to be the case that users export their created maps and then share images on social media such as Reddit.

Our application aims to bridge that gap by allowing users to more easily share their map creations with each other in order to allow for easy editing and building upon each others' ideas. Furthermore, community interaction (e.g. comments, likes, etc) can provide feedback and help other users better visualize their data.

Objectives

- Foster a community of cartographers through publication and sharing
- Empower map enthusiasts with customizable and personalized map editing tools
- Offer learning opportunities for researchers, authors, and educators

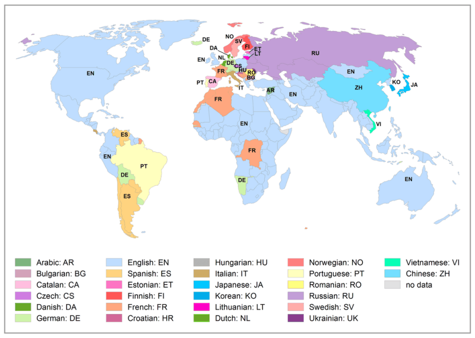
Strategies/Philosophies

- Agile Development:
    - We will develop our application in a feature-by-feature process, with a group of 2 people working on any given feature at a time.
    - If no features are currently listed on the "to-do" list, this will be a good sign for a team meeting on what to do next.
- Code Review:
    - Changes to master must be approved by another member
- User-Centric Design:
    - Prioritize user experience and usability in the application's design
- Security:
    - Implement robust security measures to protect user data and privacy

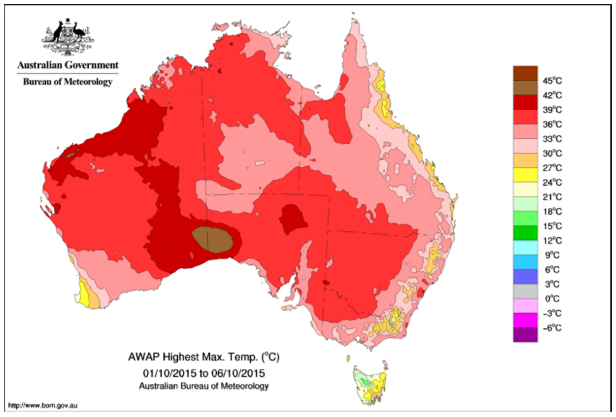
Constraints

- Map Graphic Types:

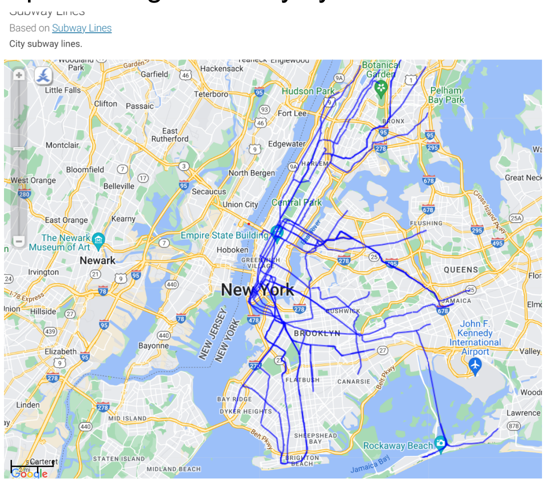
1. Bin Map (coloring) - Template that creates a blank colored world map with borders drawn between countries. Allows users to fill in regions with specific colors, or change what type level of administrative borders are drawn.



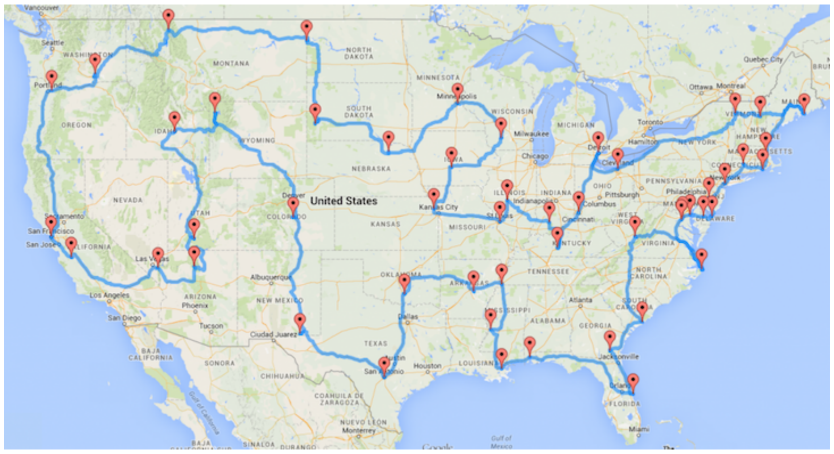
2. Heat Map (gradient coloring) - Similar to bin map, but allows users to provide region data such that they can be colored based on that data.



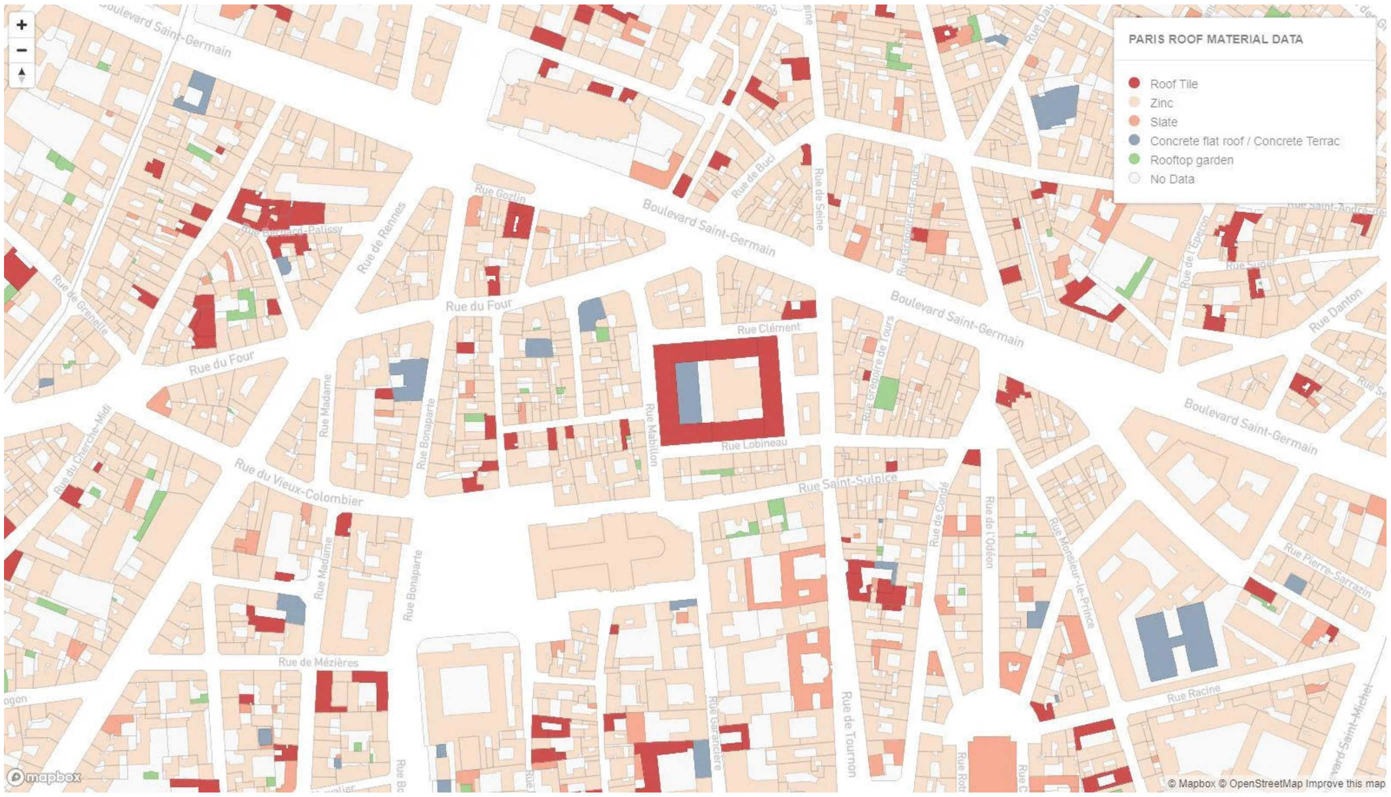
3. Subway - Template that overlays a set of vectors onto a blank region (e.g. NYC) representing a subway system.


Subway Lines
Based on Subway Lines
City subway lines.

4. Landmark - Template that creates a map with labeled points of interest (e.g. restaurants in a city, historical landmarks, buildings of significance, etc.)



5. Cadastral - Template that creates a map with borders around property / buildings in a given region.



- Device Compatibility:
    - Web application optimized for desktop use only and many not provide an optimal experience on mobile devices.

Actors

- Registered Users:
    - View/Fork/Comment/Export existing published maps
    - Create/edit/delete/Export owned maps
- Guest Users:
    - View/Export public maps
    - Sign up for an account
- Administrators:
    - Manage user accounts, maps, comments

Services

- Account Management:
    - Secure account creation and login
    - Authentication and authorization
    - “Forgot Password” mechanism
- Map Graphics Creation:
    - Upload SHP/DBF, GeoJSON, or KML files
    - Fork existing maps on the forum
- Map Graphics Editing:
    - Create new map from template
    - View and navigate maps
    - Attach custom properties to map regions
    - Undo/redo changes
    - Decorate maps with texts, colors, and legends
- Map Graphics Exporting:
    - Export map graphics as PNG, JPG, or JSON format
        - Zip file containing geojson and our own proprietary json files
        - Allow users to import previously exported maps
- Map Classification & Search
    - Classify maps with properties/tags
    - Search functionality based on properties for public maps
- Community Interactions:
    - Public commenting feature on maps
    - Like button on maps

Use Case #	UI Context	Use Case Name
2.1	Sign In Screen	Create Account
2.2	Sign In Screen	Login to Account
2.3	Any Screen	Logout of Account
2.4	Home Screen	Use as Guest
2.5	Sign In Screen	Recover Password
2.6	My Maps Screen	View Own Map
2.7	My Maps Screen	Upload Map
2.8	Edit Maps Screen	Edit Map
2.9	My Maps or Edit Maps Screen	Publish Map
2.10	My Maps Screen	Delete Map
2.11	Home Screen	View All Posts (Threads & Maps)
2.12	Home Screen	View Maps Only
2.13	Home Screen	Search Maps
2.14	Post Screen	Fork Existing Map
2.15	Post Screen	Comment on Post
2.16	Post Screen	Delete Comment
2.17	Home Screen	Delete Post
2.18	Post Screen	Like Post
2.19	Post Screen	Export Map

Use Case #	2.1
Use Case Name	Create Account
Actors	Guest
Story	The user arrives on the home screen and wants to start making maps. The user clicks the “Log In” button on the top right and a modal with log in options will appear. Since the user does not have an account, they click the “Sign Up” button at the bottom of the modal. The user can then enter their username, email, and password in the fields on the modal and press “Create Account”. This will add the user to the database and automatically log them in and redirected back to the home screen.
Scenario	David Lin is a user that would like to make a map. David arrives on the homepage after entering the website on his browser. He clicks on “Log In” which brings up the log in modal. Since he doesn’t already have an account, he clicks on the “Sign Up” button. He enters “david” in the username field, “david@david.com” in the email field, and “david123” in the password field. He confirms his password and clicks “Create Account” and is now logged in and back on the home screen.
Exceptions	A user cannot create an account using an email address that is already associated with another account. In the case that this happens, the application should show styled feedback with an appropriate message. The same can be applied if improper passwords are provided. The same is true for usernames as they should also be unique.

Use Case #	2.2
Use Case Name	Login to Account
Actors	Guest
Story	The user arrives on the home screen and wants to start making maps. The user clicks the “Log In” button on the top right and a modal with log in options will pop up. Since the user has an account, they click on the “Log In” button on the bottom of the modal. The user can then enter their username and password in the fields on the modal and press “Log In”. This will log in the user and redirect them.
Scenario	David Lin is a user that would like to make a map. David arrives on the homepage after entering the website on his browser. He clicks on “Log In” which brings up the log in modal where he enters his username and password. He then clicks “Log In” and is redirected to the home page.
Exceptions	A user cannot log in if they provide incorrect credentials. In this case, a pop up will alert the user of the error.

Use Case #	2.3
Use Case Name	Logout of Account
Actors	Logged-In User, Admin
Story	The user is logged into the application. The user thinks about wanting to log out. The user puts his hands on the mouse and moves the cursor to the drop-down menu. The user selects the “Logout” button located in the drop-down menu and clicks it. The user is then logged out and redirected to the home screen.
Scenario	David is logged into the application. David thinks about wanting to log out. David puts his hands on the mouse and moves the cursor to the drop-down menu. David selects the “Logout” button located in the drop-down menu and clicks it. David is then logged out and redirected to the home screen.
Exceptions	A guest cannot see the logout button because they are not logged in.

Use Case #	2.4
Use Case Name	Use as Guest
Actors	Guest
Story	The user arrives on the home screen and does not want to login. The user can view all public posts on the home screen and the view maps only screen. The user can search for maps made by others by their properties or names. They can view posts, view comments and export maps. The user cannot make maps, so they cannot navigate to the my maps page and the fork maps button is disabled. They cannot like or comment on posts so those buttons will not show up.
Scenario	David is on the home screen and wants to use the site but doesn't have an account and doesn't feel like making an account. David is fine with only viewing public posts. David clicks on the search bar and enters "Antarctica" in the text field and hits enter. This brings up the list of maps that includes "Antarctica" in the title. David sees one called "AWESOME map of Antarctica" and clicks on the expand button to view the rest of the map. David reads through the first 3 comments but wants to see more so he clicks the view more button.
Exceptions	We should make sure that guests can't navigate to screens that they don't have access to and don't see buttons that they cannot use. Make sure we have a foolproof design to ensure these conditions.

Use Case #	2.5
Use Case Name	Recover Password
Actors	Guest
Story	The user lands on the home page and is browsing the public maps. The user sees an interesting map and wants to leave a comment. The user enters the username with an incorrect password. A pop-up comes up indicating that the password is wrong. After many attempts, the user decides to click on the “Recover Password” button. The email field replaces the password field. The user enters their username and email and clicks on “Send.” The user receives an email from Cartistry prompting them to reset their password. After clicking on the reset password link they are redirected to a password reset screen. The user enters their new desired password and confirms.
Scenario	David lands on the home page and is browsing the public maps. David sees an interesting map and wants to leave a comment. David enters the username with an incorrect password. A pop-up comes up indicating that the password is wrong. After many attempts, David decides to click on the “Recover Password” button. David enters his username and email and clicks on “Send.” David receives an email from Cartistry prompting them to reset their password. After clicking on the reset password link they are redirected to a password reset screen. David enters their new desired password and confirms.
Exceptions	Only a guest has the option to recover their password. This is only available on the login screen. If the credentials entered do not match what is on the database, an error message pops up.

Use Case #	2.6
Use Case Name	View Own Map
Actors	Logged-In User, Admin
Story	The user is logged in and clicks on the avatar icon on the top-right, which will display a dropdown, and they click on “My Maps”. This will navigate them to the my maps screen where they can see all the maps they’ve already made.
Scenario	David is logged in and wants to look at his maps. He clicks on his avatar icon to click the “My Maps” button. After navigating to the my maps screen, he sees that his maps are sorted by recently opened. He wants to sort his maps alphabetically, so he clicks on the SORT BY button and selects the “By Name (A-Z)” criteria. Now he can view his maps sorted alphabetically by title.
Exceptions	The user might not have nay maps, but only the container listing all the maps should be empty, everything else should render the same (create map button, sort by, etc.)

Use Case #	2.7
Use Case Name	Upload Map
Actors	Logged-In User, Admin
Story	The user is logged in and is on the My Maps screen. The user wants to create a new map. The user clicks on the upload icon, which opens a modal for the user to select the map file they want to upload. Upon confirming the upload, a new thumbnail appears on the My Maps screen. Clicking this icon will allow the user to begin editing their map
Scenario	David navigates to the My Maps screen and wants to create a new map. He previously downloaded a file called “EU.json” containing geojson data for all the administrative borders corresponding to the countries in Europe. He clicks the upload button, chooses his file to upload, then confirms the upload. He then clicks on the generated thumbnail to begin editing his map
Exceptions	.kml, .json, and .zip files (containing .shp files) are the only types allowed.

Use Case #	2.8
Use Case Name	Edit Map
Actors	Logged-In User, Admin
Story	The user is logged in and is on the My Maps screen. The user clicks on one of their existing non-published maps. This transitions the user to a new screen containing the user’s map and the UI controls for editing. The user can click on the map to perform editing actions (depending on the template, different actions are possible), as well as zoom and pan around with the mouse button. When the user is done, they can either click the “Save Changes” button to confirm their edits, or “Publish” in order to create a new post with the map. This follows the same procedure as creating a new post, where the user can check off what they want to tag their post with.
Scenario	David navigates to the My Maps screen and clicks on one of his unpublished maps titled “GDP of European nations”. The map defaults to being centered at North America, so he uses his mouse to pan over to Europe. He then clicks on an area of the map containing the bordered region corresponding to Germany. Since the map was originally templated as a heatmap, he can edit the associated GDP data for Germany via the modal that popped up. Once done, he clicks on “Save Changes” to confirm his edits to the map.
Exceptions	The undo and redo transaction stack should be reset on each open of the map.

Use Case #	2.9
Use Case Name	Publish Map
Actors	Logged-In User, Admin
Story	The user is logged in and is viewing a map they have created which they are ready to publish. The user clicks on the “File” options in the header which opens a drop-down menu. The user selects the “Publish” button located in the menu. After clicking on the button, a modal appears prompting the user to provide a “title” and text description” for the post. After the user provides the information, the user clicks the “Submit” button. The user is then redirected to the home page where they can view their newly published post. Note that a user is able to publish the same map multiple times. Doing so would allow the users to publish a map, perform edits on it, and then publish again, creating a new post.
Scenario	David is ready to publish his map. He navigates to the My Maps screen and clicks on his unpublished map titled “GDP of European nations”. This opens the map editor screen where he has access to the “Publish” button. He clicks on the button, which opens a modal prompting him to submit more information for the post he is about to create. He enters in the “title” and “text description” for the post and clicks “Submit”. This creates a new post on the site, allowing other users to view his map.
Exceptions	If a user if unhappy with their map, they can fork it to create a duplicate and then delete the published version.

Use Case #	2.10
Use Case Name	Delete Map
Actors	Logged In User
Story	The user navigates to the “My Maps” screen and finds a map they no longer wish to be there. Upon hovering the thumbnail of the image of the map, a trashcan icon appears. Upon clicking the trashcan, a modal pops up, asking the user for confirmation if they want the map to be deleted. Upon clicking yes, the map is deleted from the user’s account and will no longer appear on the MyMaps screen.
Scenario	David navigates to “My Maps” and sees one of his old maps “Best Restaurants in Flushing”. He wants to delete it because his friends have made fun of his food opinions. He hovers over the map icon, clicks the trashcan icon that appears, and then confirms deletion.
Exceptions	Deleting a map will not delete the post(s) associated from all the time(s) that map was published. The user can click the cancel button, but once a map is deleted it is permanent.

Use Case #	2.11
Use Case Name	View All Posts (Threads & Maps)
Actors	Logged In User, Guest, Admin
Story	The actor’s browser loads the correct URL for the web application. The home page loads where the default view upon loading the page is the view of all posts (threads & maps). Thumbnails, titles, brief descriptions and a 3 dot menu for other options are provided for each post The actor can navigate through all the posts and filter them by name, tag, date and click on any post they wish to view to see its full content.
Scenario	David has loaded the web application on his desktop browser where he can view all posts (threads & maps). David can navigate through all the posts, filter them by preset categories and can click on any post to see its full content.
Exceptions	The menu associated with each post contains varying options depending on the current actor. Logged-In users and Admins are presented with delete/edit options.

Use Case #	2.12
Use Case Name	View Maps Only
Actors	Guest, Logged-In User, Admin
Story	The user is on the home screen and viewing all posts, but they want to only look at maps. The user navigates to the right part of the screen where the filters are and checks the “Maps Only” checkbox. They click apply, and now are viewing only posts with maps.
Scenario	David is scrolling on the home screen and is getting annoyed with the multiple general discussion posts since he’s here to see some cool maps. He moves his mouse to the right of the screen and clicks the checkbox for “Maps Only”. Now only maps appear on the screen and David is very satisfied.
Exceptions	To go back to the View All screen, the user can either click the logo (home icon) or uncheck “Maps Only” and hit apply.

Use Case #	2.13
Use Case Name	Search Maps
Actors	Guest, Logged-In User, Admin
Story	The user can type into the search bar and submit a query to filter all posts on the site. A drop down menu in the search bar specifies what type of query it will be, either by post title or by tags. Queries for post title will return all posts that contain the query in the title. If a user chooses to search by tags, they must choose from a pre-existing set of tags. Upon selecting the tags, the search will filter all posts containing any of the tags, sorted by whichever posts contain the most tags.
Scenario	David is looking for maps related to the video game Valorant. He is on the search bar and chooses to search posts by tag. He selects the following tags: “videogames”, “esports”, “geek”. His search returns a total of 10 posts, the top of which contains all tags, and the bottom of which only contain 1 tag.
Exceptions	

Use Case #	2.14
Use Case Name	Fork Existing Map
Actors	Logged-In User, Admin
Story	The user has the option to take an existing post containing a published map and fork it. By forking it, they essentially import the geoJSON associated with the post into their own account. It is equivalent to having the geoJSON data already downloaded and creating a new map by the “Upload Map” functionality. Upon forking, the user’s MyMaps screen will contain the forked map.
Scenario	David sees a post containing a map titled “Best Restaurants in Flushing”. He likes the author’s opinion, but wants to add some suggestions of his own. Instead of starting from scratch, David clicks on the fork button for the map which creates a copy of the map in his account. At this point, David can edit the map to his liking, adding his own favorite restaurants to it.
Exceptions	If the user already forked a map with the same name, a number should be added to the map title to make it unique.

Use Case #	2.15
Use Case Name	Comment on Post
Actors	Logged-In User, Admin
Story	The user sees an interesting post. The post has an interesting conversation going on in the thread. The user wants to join in on the conversation. The user clicks on the text box in the comment section and begins typing “I love McKIllaGorilla.” The user then clicks the post/submit button for the comment. The comment is then posted and visible to other users.
Scenario	David sees an interesting post. The post has an interesting conversation going on in the thread. David wants to join in on the conversation. David clicks on the text box in the comment section and begins typing “I love McKIllaGorilla.” David then clicks the post/submit button for the comment. The comment is then posted and visible to other users.
Exceptions	Guests cannot post comments.

Use Case #	2.16
Use Case Name	Delete Comment
Actors	Logged-In User, Admin
Story	The admin sees an inappropriate comment regarding McKillaGorilla. As a good admin, the admin moves the cursor to the delete button. The admin clicks on the button and the comment is then deleted.
Scenario	David sees an inappropriate comment regarding McKIllaGorilla. As a good admin, David moves the cursor to the delete button. David clicks on the button and the comment is then deleted. David then sees one of his own comments that he does not like. He proceeds to click on the delete comment button and the comment is discarded.
Exceptions	You cannot delete other users' comments unless you are an admin.

Use Case #	2.17
Use Case Name	Delete Post
Actors	Logged-In User, Admin
Story	The user just made a post. The user does not like the post they just made. The user wants to delete the post they just made. The user moves the cursor to the delete post button. The user clicks the button. The post is then discarded and no longer visible to other users.
Scenario	David just made a post. David does not like the post they just made. David wants to delete the post they just made. David moves the cursor to the delete post button. David clicks the button. The post is then discarded and no longer visible to other users.
Exceptions	You cannot delete other users' posts unless you are an admin.

Use Case #	2.18
Use Case Name	Like/Unlike Post
Actors	Logged-In User, Admin
Story	The user is on the home screen and sees a post they like. They expand the post to see the rest of it and see the like icon on the bottom with the number of likes next to it. They click the like icon and the number of likes increases.
Scenario	David is scrolling on the home screen and sees a map of Antarctica that intrigues him. He expands it to see the rest of the map and finds it very nice. He looks at the bottom right and sees that this map already has 103 likes next to an icon of an outlined heart. David navigates his mouse to the like icon and clicks it, making the like count 104 and making the icon a filled-in heart. However, he realizes that the author accidentally spelled Antarctica as “Antartica” and finds it unforgivable. So, he clicks on the filled-in heart button to unlike which makes the like count go back down to 103 and makes the icon back to an outlined heart.
Exceptions	The user should only be able to like once, as clicking the button again should unlike the post. If other users are liking the post, the number should only change on reload. The user can only see the count dynamically change for their own like.

Use Case #	2.19
Use Case Name	Export Map
Actors	Logged-In User, Guest User, Admin
Story	The user is on the home screen and sees a map they like. They expand the post to see the rest of the map and the export icon on the bottom left. They click the button and a modal appears to let them choose how to download the map (PNG, JPEG, or JSON). Once they select one, the user can click the “Export Map” button and a download will begin.
Scenario	David is scrolling on the home screen and sees a map of Antarctica that intrigues him. He clicks on the expand button to see the rest of the map and likes it very much. He wants to send it to his friends so he clicks on the export button on the bottom left of the post. A pop up appears, and David clicks on the dropdown to choose the ‘PNG’ option. He clicks “Export Map” and the download begins.
Exceptions	The user can clicks cancel to cancel the action, so a download will not begin.

Use Case #	2.20
Use Case Name	View Profile
Actors	Logged-In User, Admin
Story	
Scenario	
Exceptions	

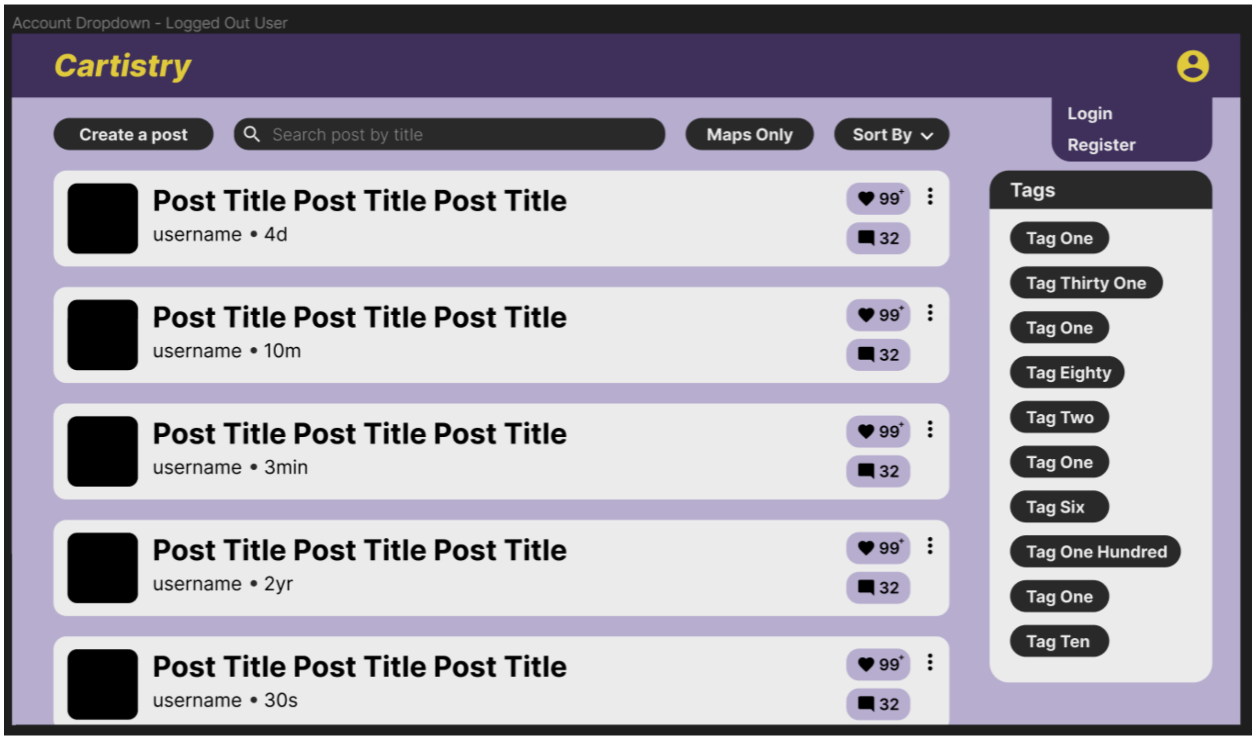
Use Case #	2.21
Use Case Name	Reset Password
Actors	
Story	
Scenario	
Exceptions	

Cartistry User Interface View Listing - Team Purple

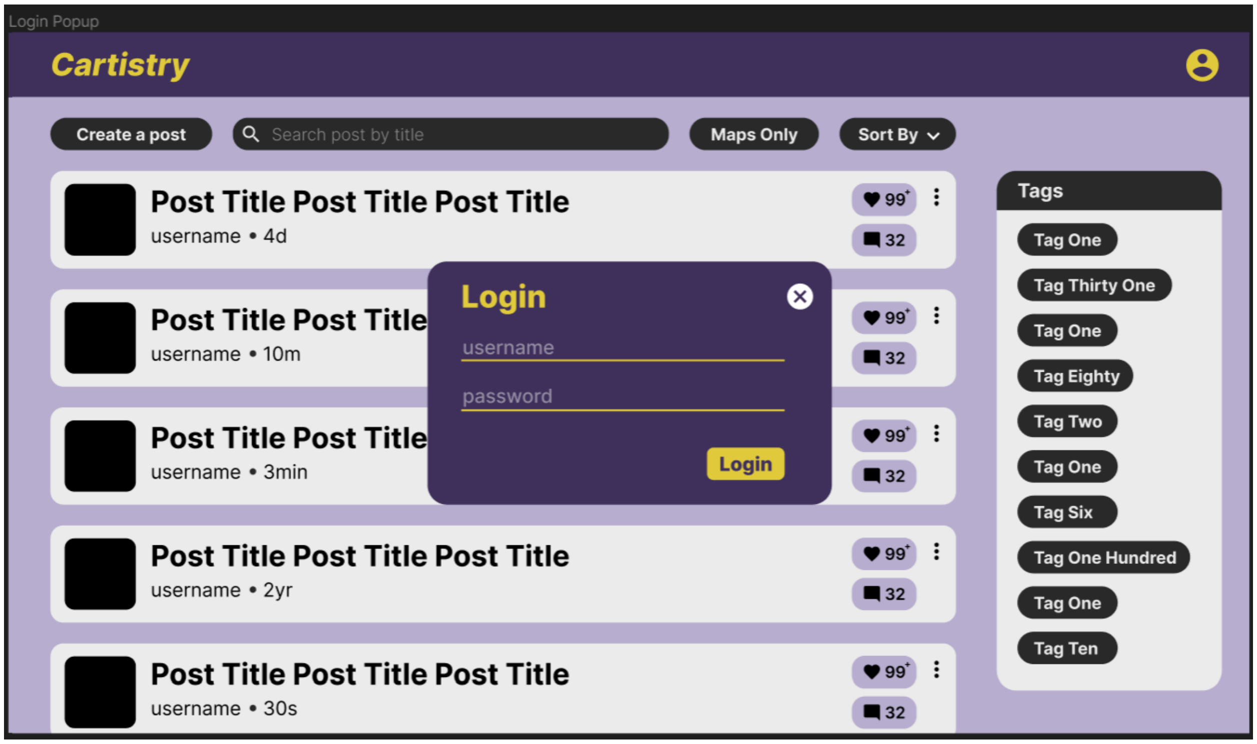
View #	Name	Description
0	Profile Icon To Login/Register	Navigates to Login/Register Modal
1	Login Modal	Login to Account
2	Register Modal	Create Account
3	Clicking Profile Icon as Logged In User	Logged In User can navigate to “My Maps”, “Reset Password” or “Logout”
4	Home Screen	Contains Top Posts, containing both text and map posts. Users can use the search bar to search for posts by title.
4.1	Home Screen Maps Only Filter	Users can filter top posts to filter out text only posts
4.2	Home Screen Sort	Users can sort posts by newest, oldest, and most liked.
4.3	Home Screen Filter By Tag	Users can filter posts to be ones only containing tags they select.
4.4	Home Screen Like Post	Users can like any post they haven’t already liked via the home screen.
5	Reset Password Modal	Logged In User Can Reset Password
6	My Maps Screen	View of logged in user’s maps after clicking on profile icon and “My Maps”
6.1	My Maps Screen (Creating Map Menu)	My Maps screen view with “Create A Map” menu open with options “Use default” and “Import”
6.2	My Maps Screen (Import Map Popup)	Modal popup after clicking “Import” from “Create A Map” menu
6.3	My Maps Screen (Sort By Menu)	My Maps screen view with “Sort By” Menu expanded with options “Name”, “Edit/Publish/Create Date”
6.4	My Maps Screen (Map Card Menu)	My Maps screen view with “Map Card” menu expanded with options “Export”, “Publish”, “Fork”, “Rename”, “Delete”

6.5	My Maps Screen (Delete Confirm Popup)	Modal popup after clicking “Delete” from “Map Card” menu
6.6	My Maps Screen (Publish Confirm Popup)	Modal popup after clicking “Publish” from “Map Card” menu
7	Edit Maps Screen	Users can edit their map graphics through this screen and the UI controls.
7.1	Edit Maps Screen Pan Select	Users can enter Pan mode where they can click and drag to view around the map.
7.2	Edit Maps Screen Color Mode	Users can enter Color Mode where clicking on a bordered region on the map will color it accordingly.
7.3	Edit Maps Screen Pin Mode	Users can enter Pin Mode where clicking on the map drops a point which they can label and assign a color
7.4	Edit Maps Screen Color Selection Menu Open	Users can select a color to be used in color mode.
7.5	Edit Maps Screen Menu	Users can choose to save, export, publish, fork, or delete their map
8	Create Post Screen	Default view after clicking create post button on home screen
8.1	Create Post Screen (Added Attachment and Tag)	Create a Post screen containing optional attachments. If the User came here via Publishing a map, their map will be added as an image attachment.
9	Post Screen	Contains full post title description, comments and attachments. Users always have the option to like and comment. If map data is attached, users can fork or export it.
9.1	Post Screen Comment Menu	Users can edit or delete their own comments on a post.

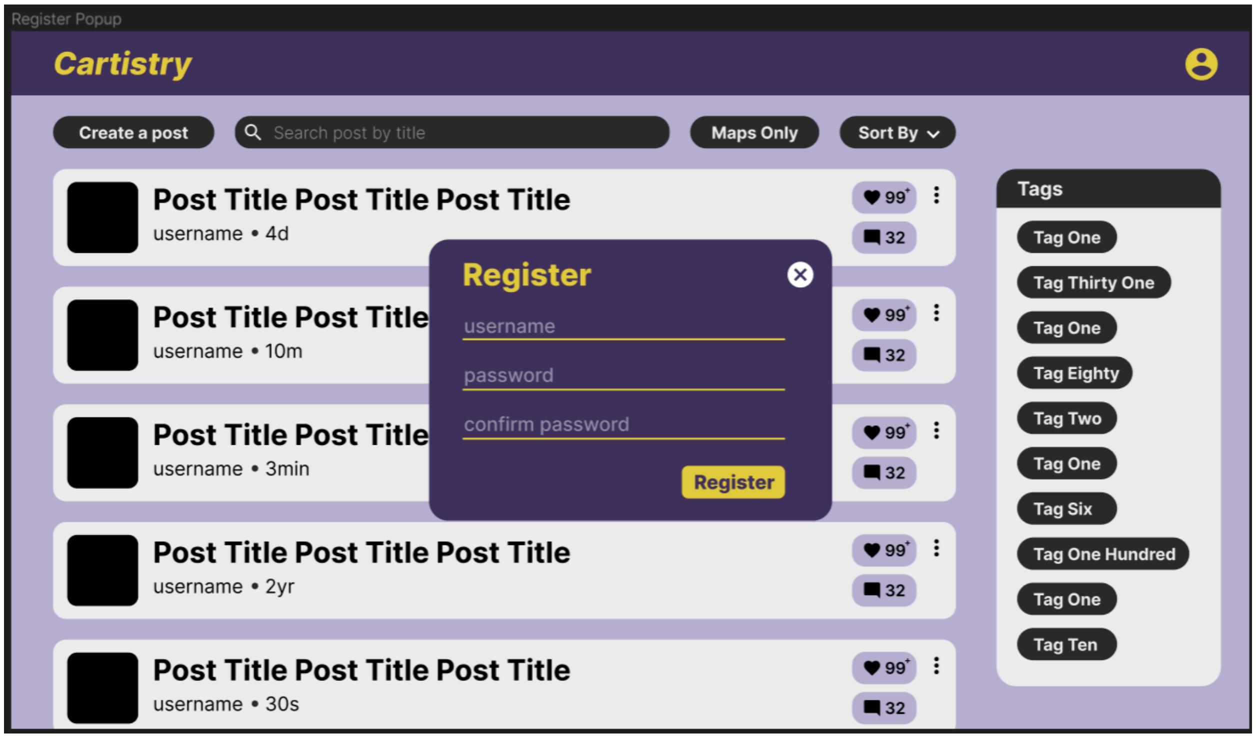
0


Account Dropdown - Logged Out User
Cartistry
Login
Register
Create a post
Search post by title
Maps Only
Sort By
Post Title Post Title Post Title
username • 4d
username • 10m
username • 3min
username • 2yr
username • 30s
99+
32
Tags
Tag One
Tag Thirty One
Tag One
Tag Eighty
Tag Two
Tag One
Tag Six
Tag One Hundred
Tag One
Tag Ten

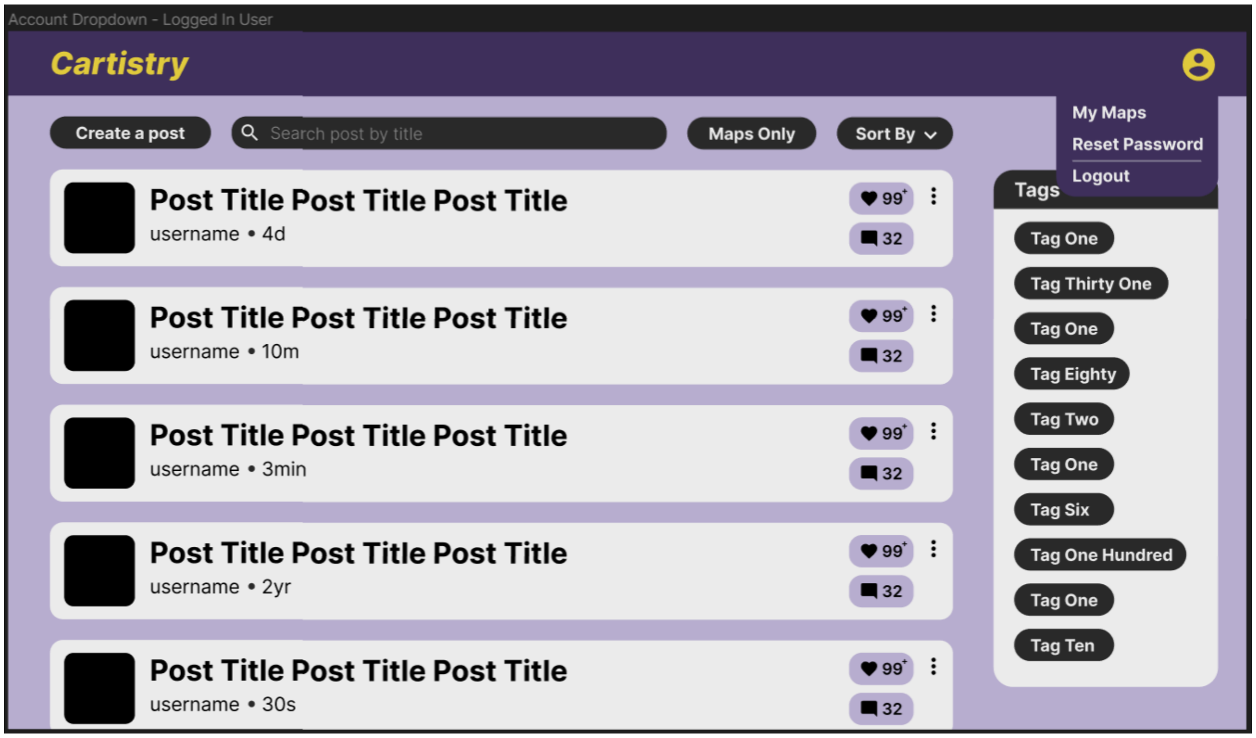
1


Login Popup
Cartistry
Create a post
Search post by title
Maps Only
Sort By
Post Title Post Title Post Title
username • 4d
username • 10m
username • 3min
username • 2yr
username • 30s
Login
username
password
Login
Tags
Tag One
Tag Thirty One
Tag One
Tag Eighty
Tag Two
Tag One
Tag Six
Tag One Hundred
Tag One
Tag Ten

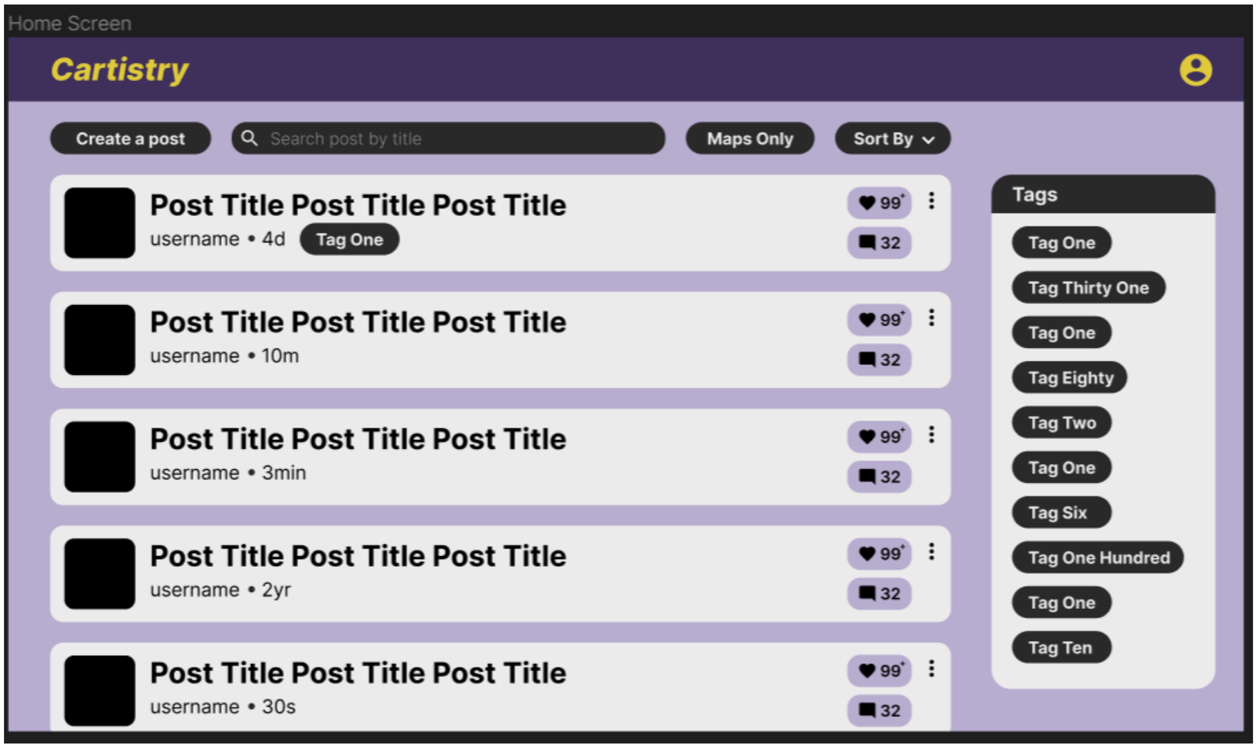
2


Register Popup
Cartistry
Create a post
Search post by title
Maps Only
Sort By
Post Title Post Title Post Title
username • 4d
Post Title Post Title
username • 10m
Post Title Post Title
username • 3min
Post Title Post Title Post Title
username • 2yr
Post Title Post Title Post Title
username • 30s
99+
32
Register
username
password
confirm password
Register
Tags
Tag One
Tag Thirty One
Tag One
Tag Eighty
Tag Two
Tag One
Tag Six
Tag One Hundred
Tag One
Tag Ten

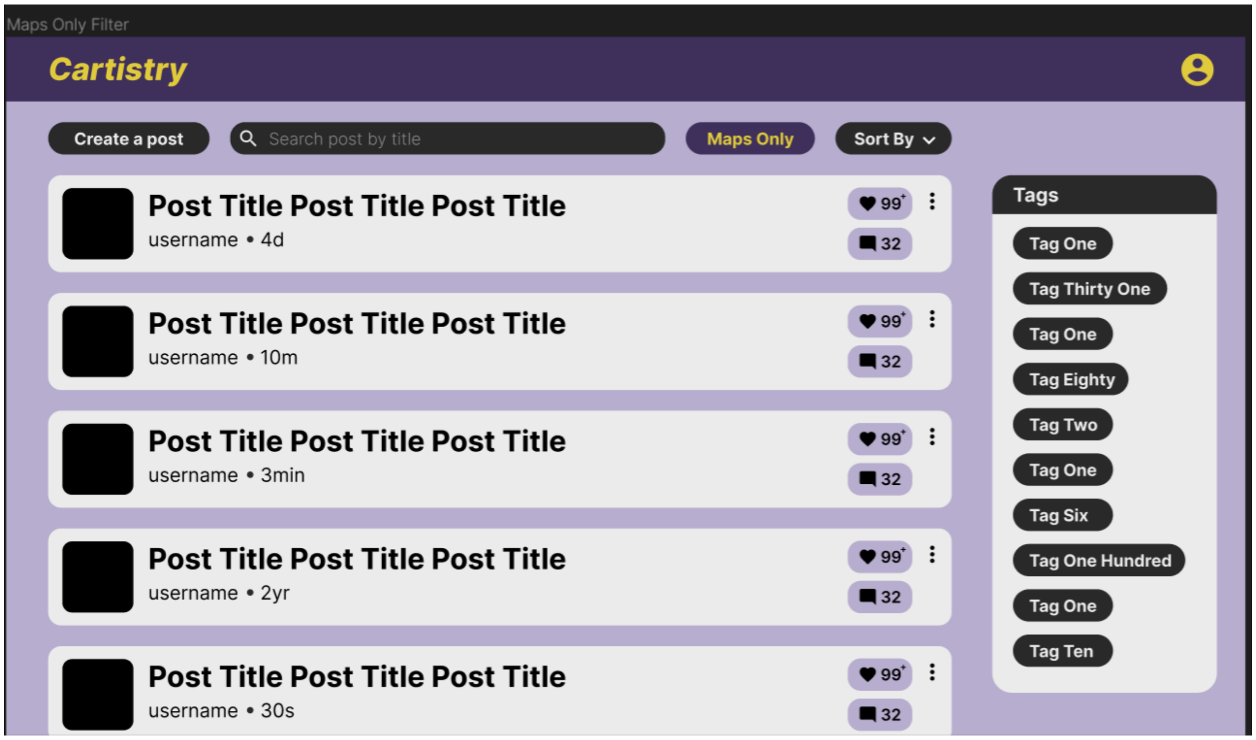
3


Account Dropdown - Logged In User
Cartistry
My Maps
Reset Password
Logout
Create a post
Search post by title
Maps Only
Sort By
Post Title Post Title Post Title
username • 4d
Post Title Post Title Post Title
username • 10m
Post Title Post Title Post Title
username • 3min
Post Title Post Title Post Title
username • 2yr
Post Title Post Title Post Title
username • 30s
99+
32
Tags
Tag One
Tag Thirty One
Tag One
Tag Eighty
Tag Two
Tag One
Tag Six
Tag One Hundred
Tag One
Tag Ten

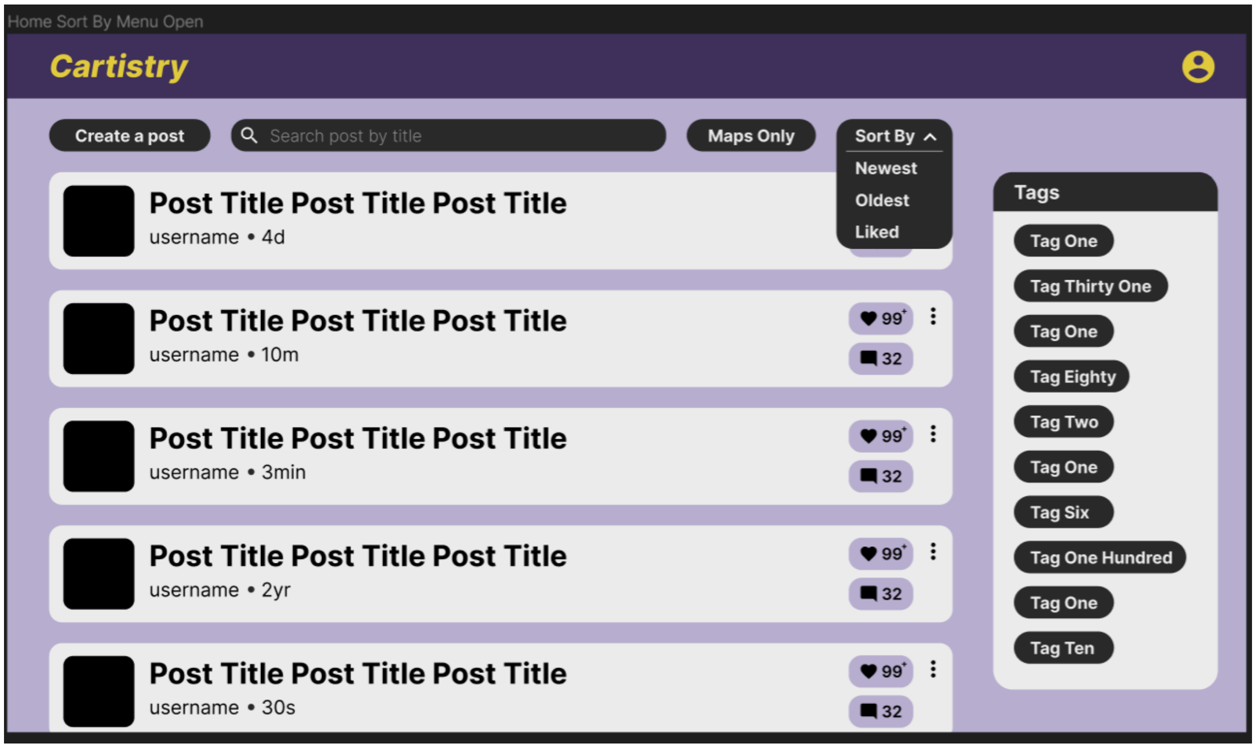
4


Home Screen
Cartistry
Create a post
Search post by title
Maps Only
Sort By
Post Title Post Title Post Title
username • 4d
Tag One
99+
32
Post Title Post Title Post Title
username • 10m
Post Title Post Title Post Title
username • 3min
Post Title Post Title Post Title
username • 2yr
Post Title Post Title Post Title
username • 30s
Tags
Tag One
Tag Thirty One
Tag One
Tag Eighty
Tag Two
Tag One
Tag Six
Tag One Hundred
Tag One
Tag Ten

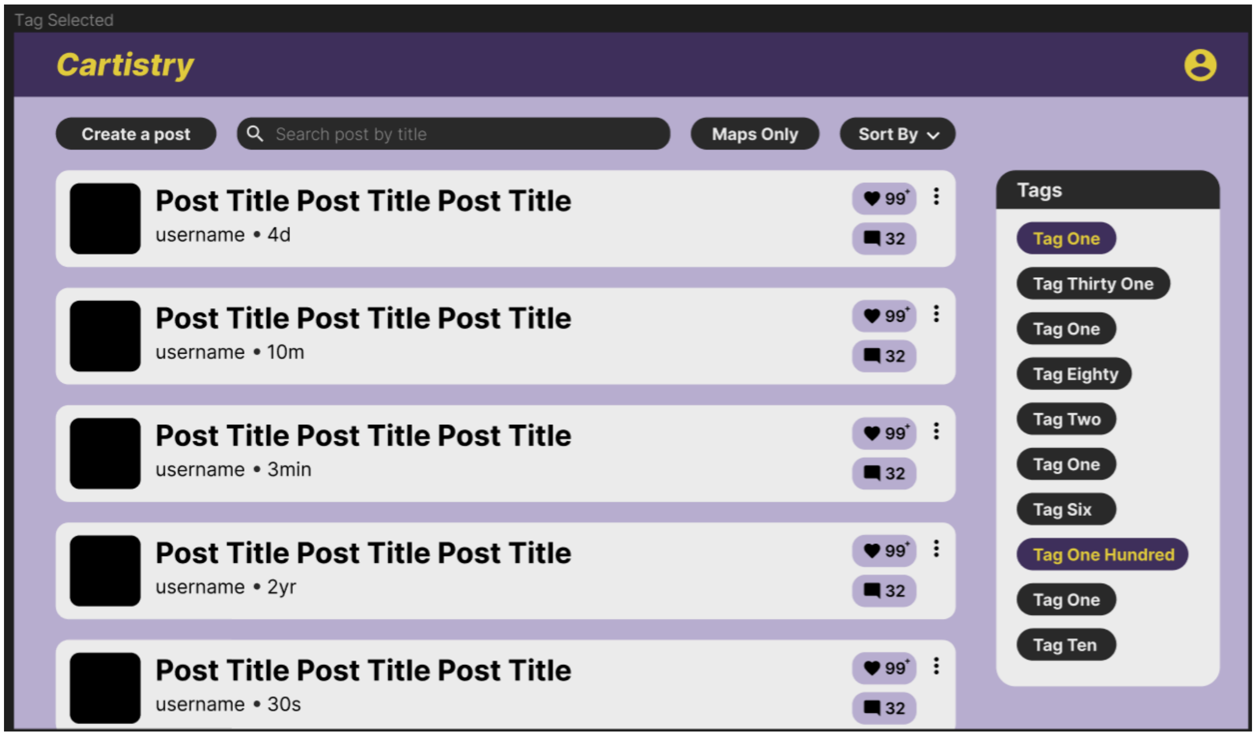
4.1


Maps Only Filter
Cartistry
Create a post
Search post by title
Maps Only
Sort By
Post Title Post Title Post Title
username • 4d
99+
32
Post Title Post Title Post Title
username • 10m
Post Title Post Title Post Title
username • 3min
Post Title Post Title Post Title
username • 2yr
Post Title Post Title Post Title
username • 30s
Tags
Tag One
Tag Thirty One
Tag One
Tag Eighty
Tag Two
Tag One
Tag Six
Tag One Hundred
Tag One
Tag Ten

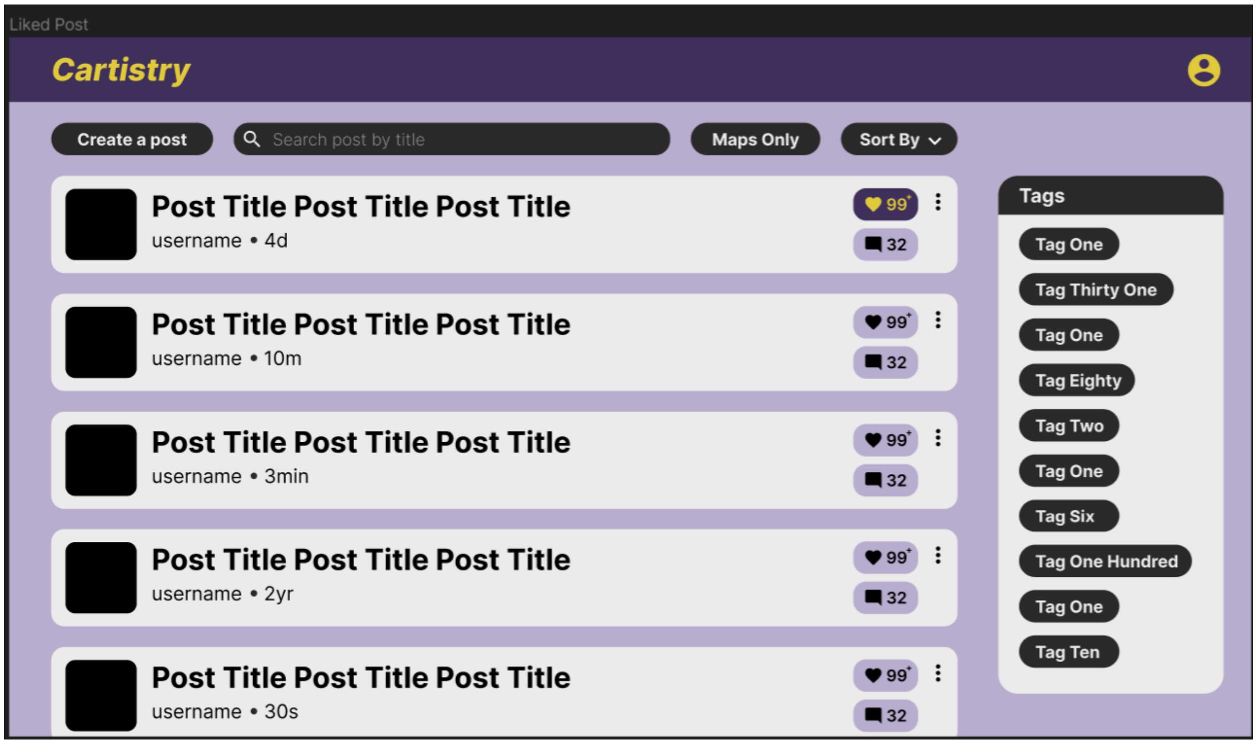
4.2


Home Sort By Menu Open
Cartistry
Create a post
Search post by title
Maps Only
Sort By
Newest
Oldest
Liked
Post Title Post Title Post Title
username • 4d
Post Title Post Title Post Title
username • 10m
99+
32
Post Title Post Title Post Title
username • 3min
99+
32
Post Title Post Title Post Title
username • 2yr
99+
32
Post Title Post Title Post Title
username • 30s
99+
32
Tags
Tag One
Tag Thirty One
Tag One
Tag Eighty
Tag Two
Tag One
Tag Six
Tag One Hundred
Tag One
Tag Ten

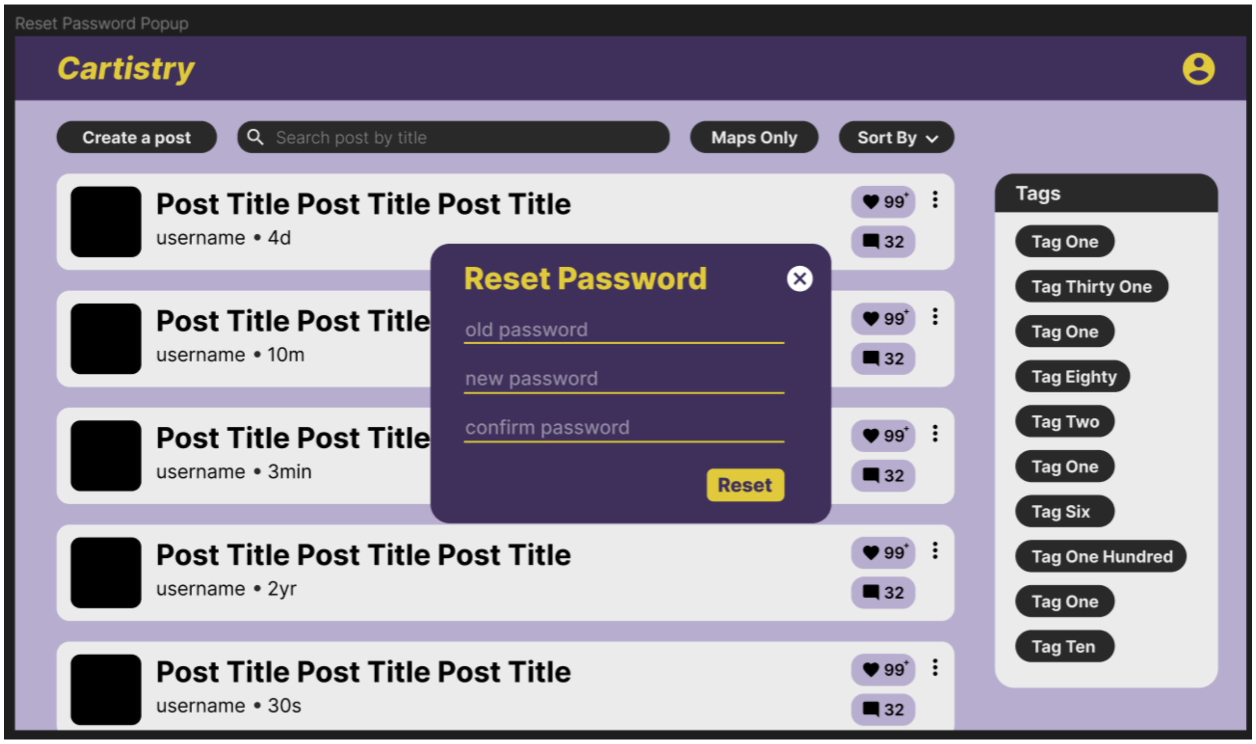
4.3


Tag Selected
Cartistry
Create a post
Search post by title
Maps Only
Sort By
Post Title Post Title Post Title
username • 4d
99+
32
Post Title Post Title Post Title
username • 10m
99+
32
Post Title Post Title Post Title
username • 3min
99+
32
Post Title Post Title Post Title
username • 2yr
99+
32
Post Title Post Title Post Title
username • 30s
99+
32
Tags
Tag One
Tag Thirty One
Tag One
Tag Eighty
Tag Two
Tag One
Tag Six
Tag One Hundred
Tag One
Tag Ten

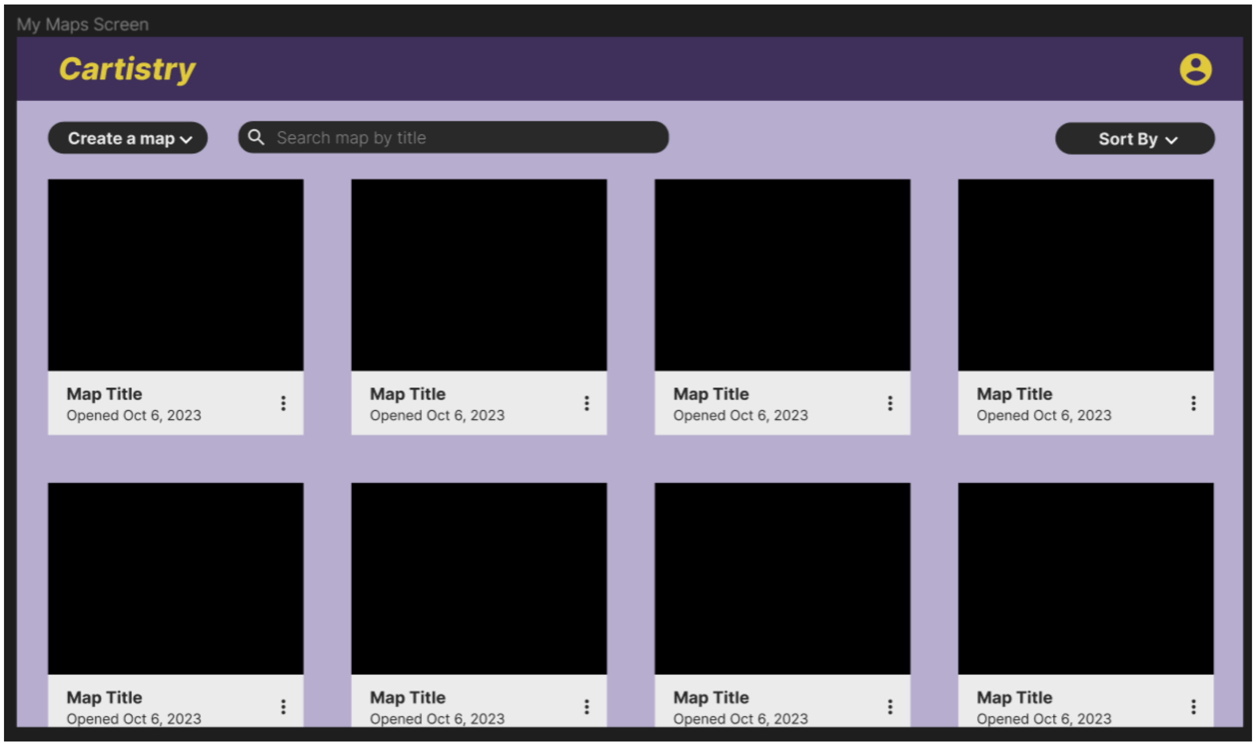
4.4


Liked Post
Cartistry
Create a post
Search post by title
Maps Only
Sort By
Post Title Post Title Post Title
username • 4d
99+
32
Post Title Post Title Post Title
username • 10m
Post Title Post Title Post Title
username • 3min
Post Title Post Title Post Title
username • 2yr
Post Title Post Title Post Title
username • 30s
Tags
Tag One
Tag Thirty One
Tag One
Tag Eighty
Tag Two
Tag One
Tag Six
Tag One Hundred
Tag One
Tag Ten

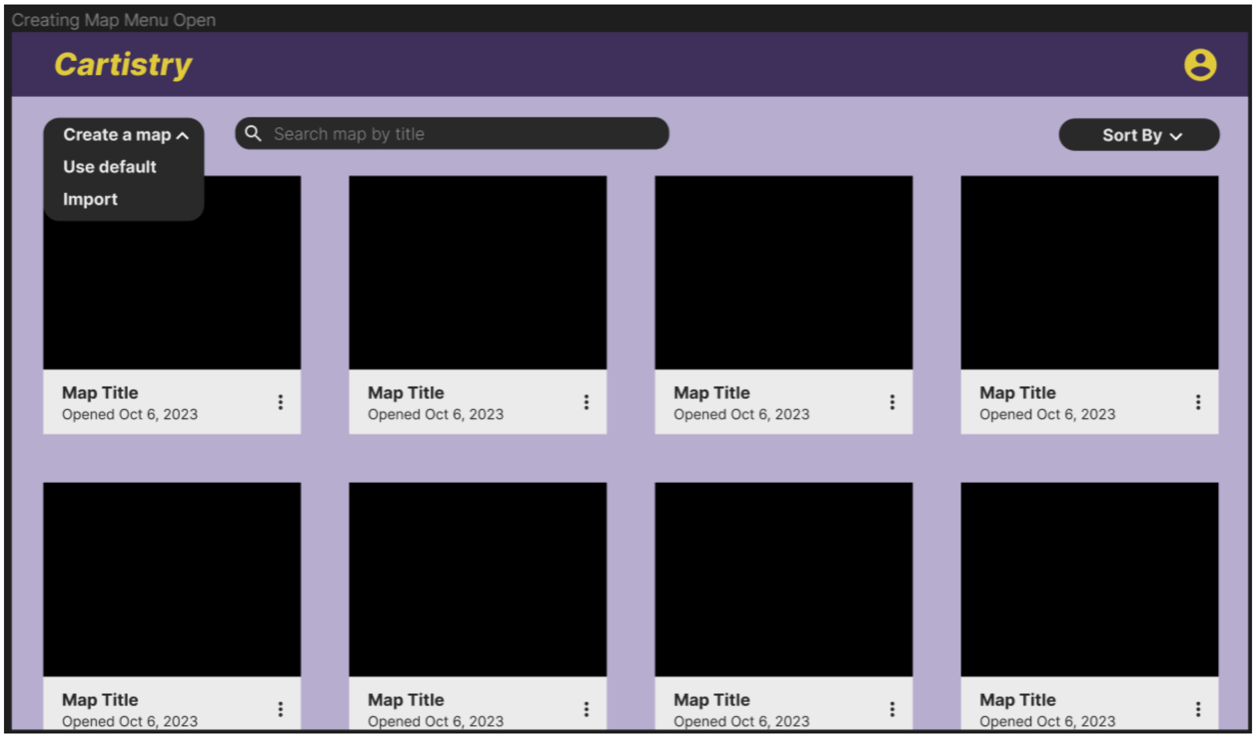
5


Reset Password Popup
Cartistry
Create a post
Search post by title
Maps Only
Sort By
Post Title Post Title Post Title
username • 4d
username • 10m
username • 3min
username • 2yr
username • 30s
99+
32
Reset Password
old password
new password
confirm password
Reset
Tags
Tag One
Tag Thirty One
Tag One
Tag Eighty
Tag Two
Tag One
Tag Six
Tag One Hundred
Tag One
Tag Ten

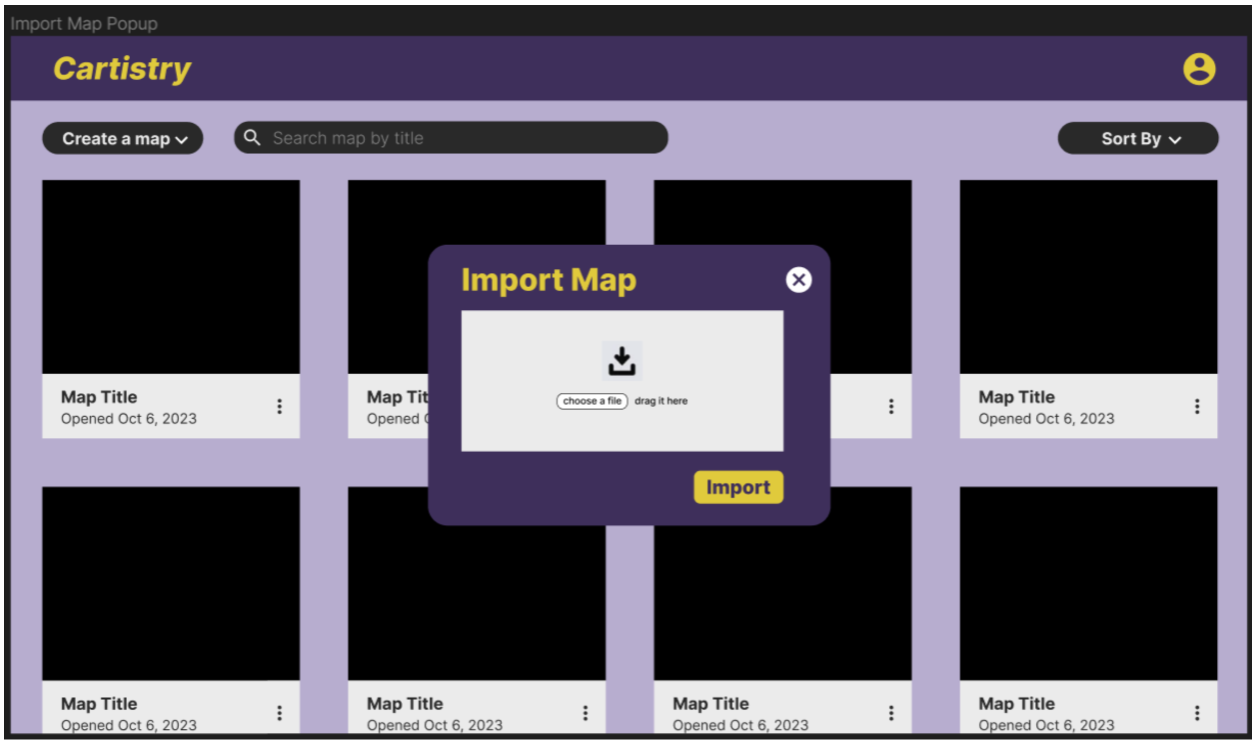
6


My Maps Screen
Cartistry
Create a map
Search map by title
Sort By
Map Title
Opened Oct 6, 2023

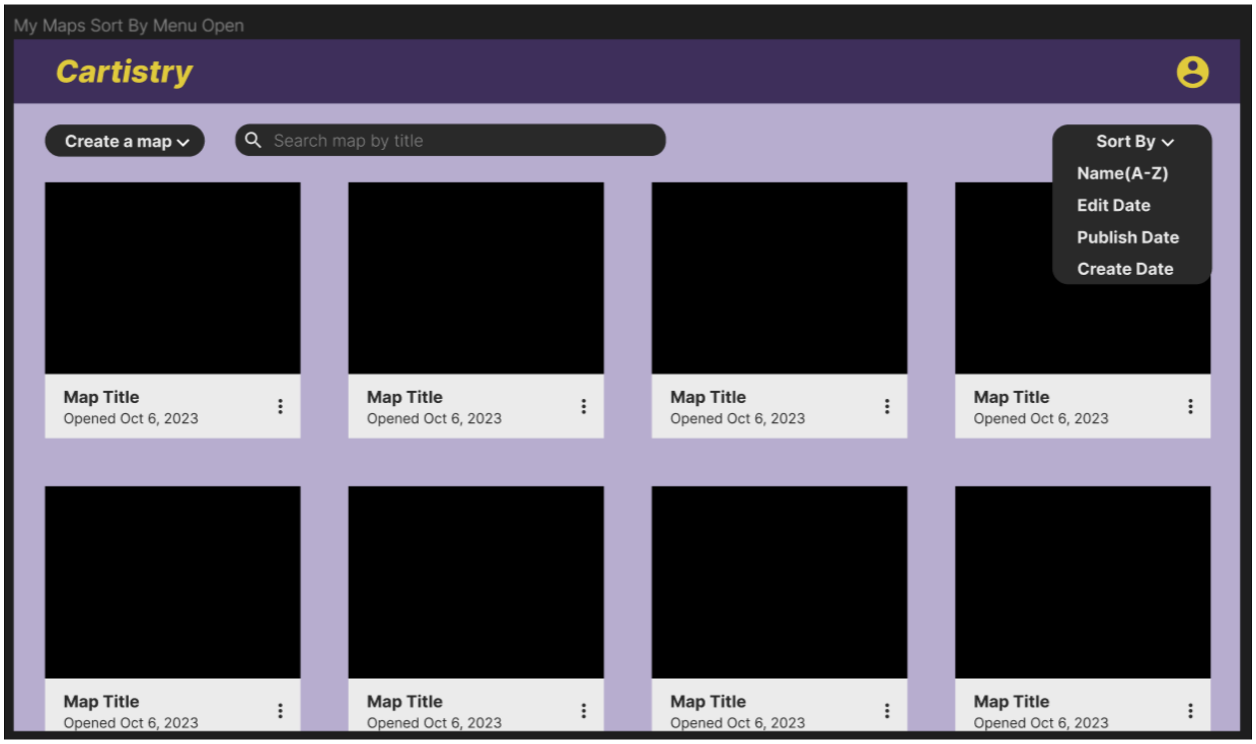
6.1


Creating Map Menu Open
Cartistry
Create a map
Use default
Import
Search map by title
Sort By
Map Title
Opened Oct 6, 2023

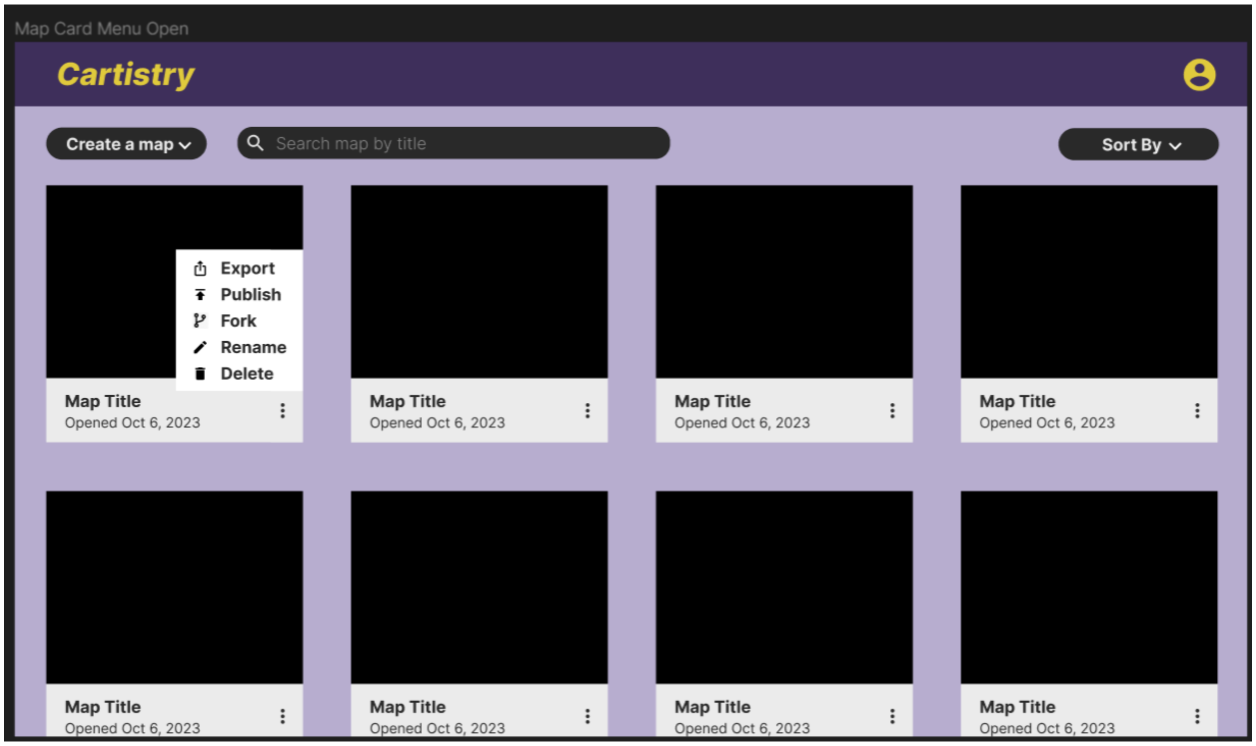
6.2


Import Map Popup
Cartistry
Create a map
Search map by title
Sort By
Import Map
choose a file
drag it here
Import
Map Title
Opened Oct 6, 2023

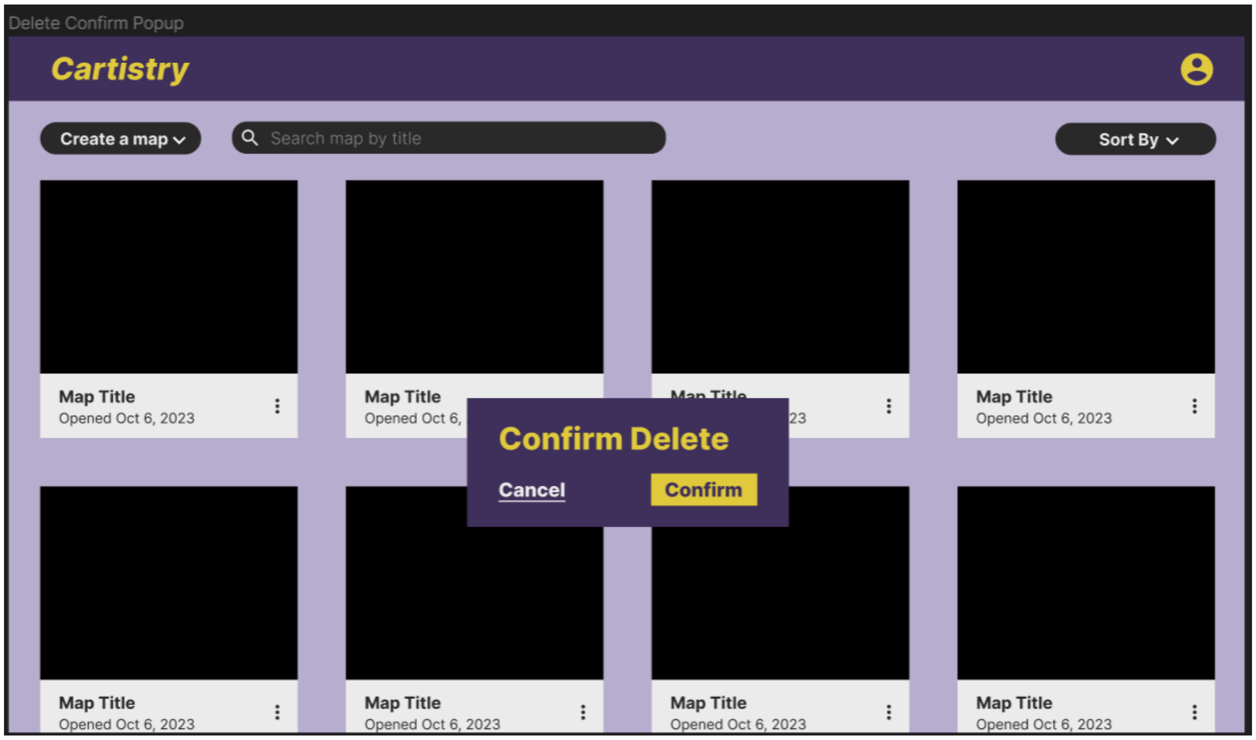
6.3


My Maps Sort By Menu Open
Cartistry
Create a map
Search map by title
Sort By
Name(A-Z)
Edit Date
Publish Date
Create Date
Map Title
Opened Oct 6, 2023

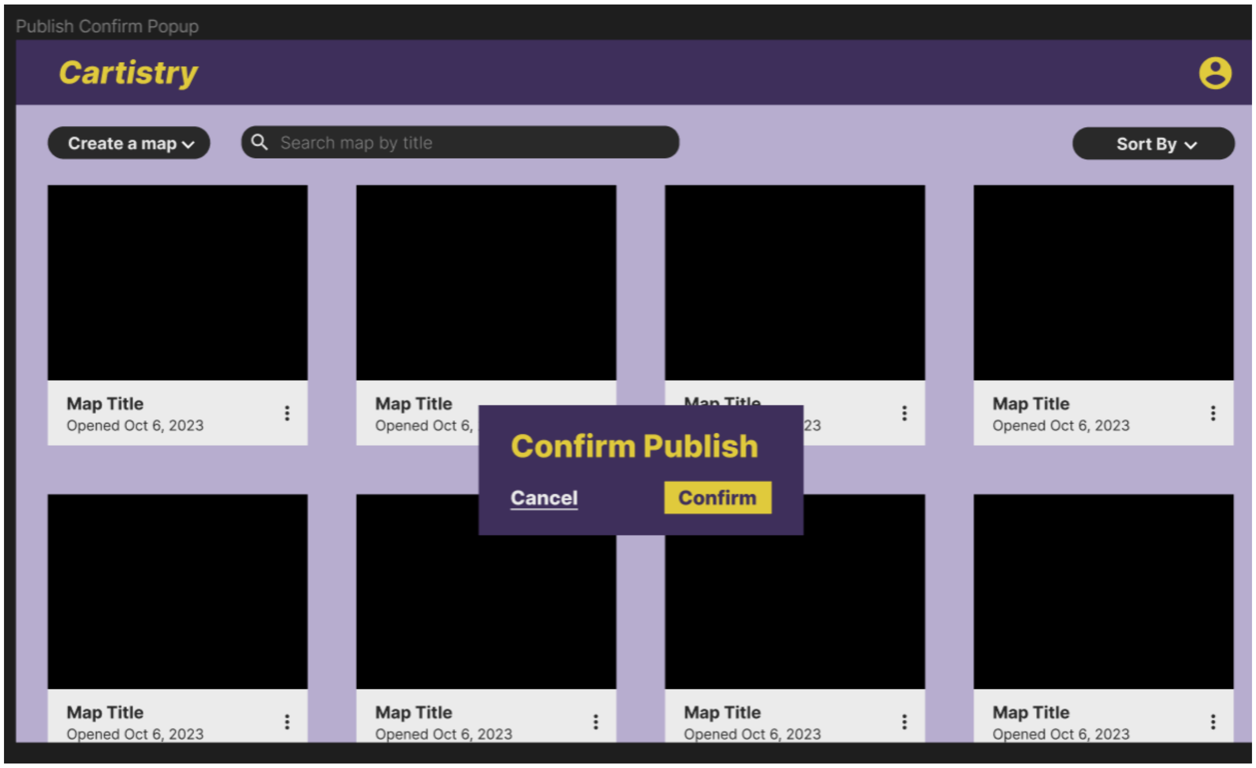
6.4


Map Card Menu Open
Cartistry
Create a map
Search map by title
Sort By
Export
Publish
Fork
Rename
Delete
Map Title
Opened Oct 6, 2023

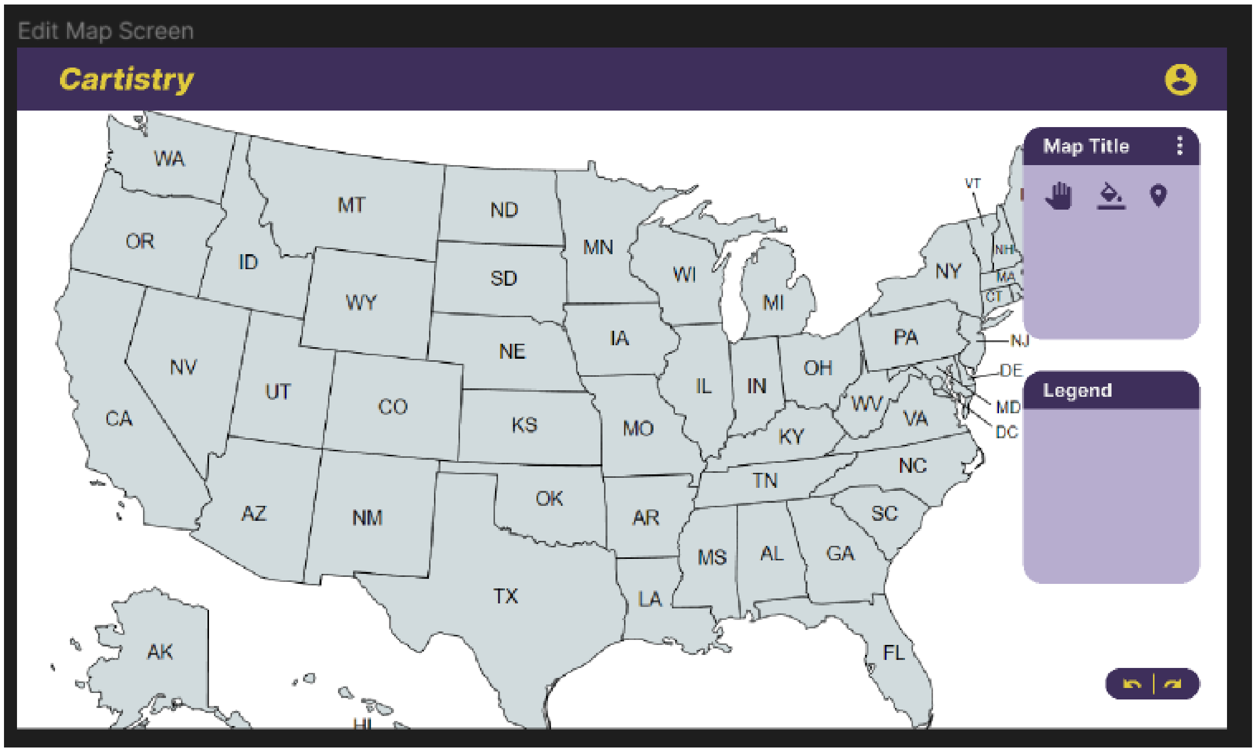
6.5


Delete Confirm Popup
Cartistry
Create a map
Search map by title
Sort By
Map Title
Opened Oct 6, 2023
Confirm Delete
Cancel
Confirm

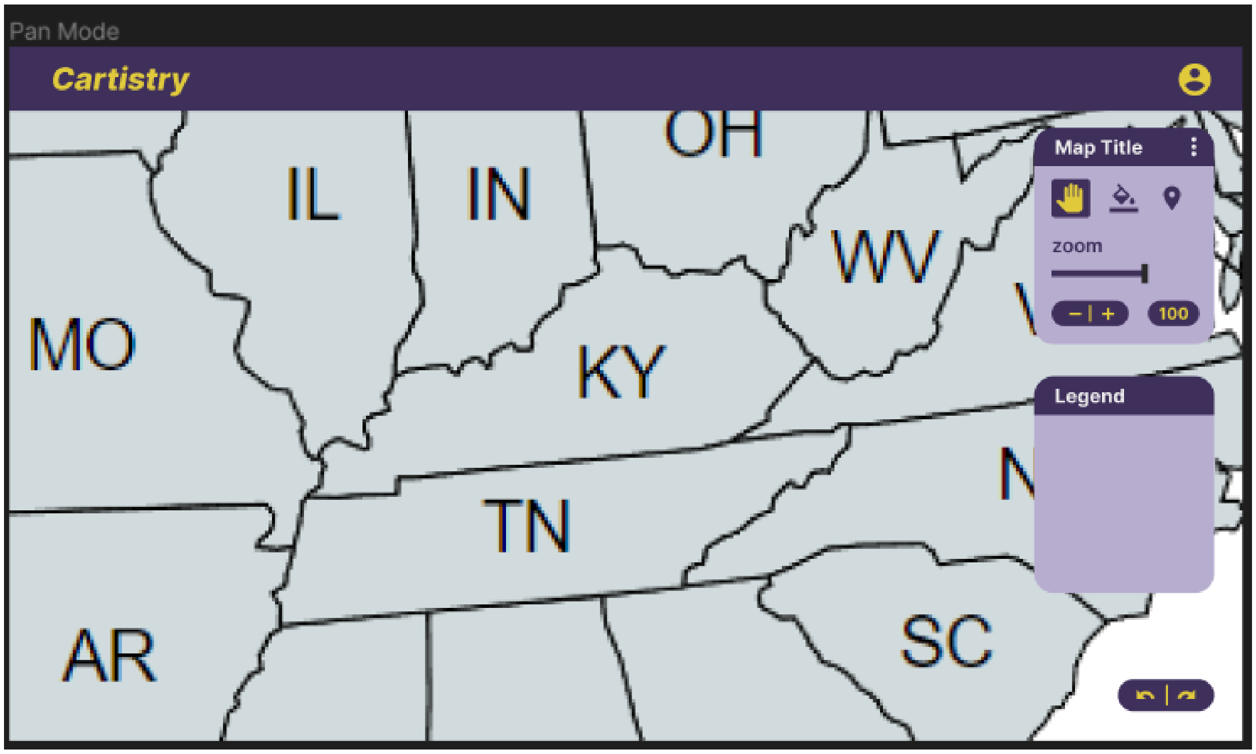
6.6


Publish Confirm Popup
Cartistry
Create a map
Search map by title
Sort By
Map Title
Opened Oct 6, 2023
Confirm Publish
Cancel
Confirm

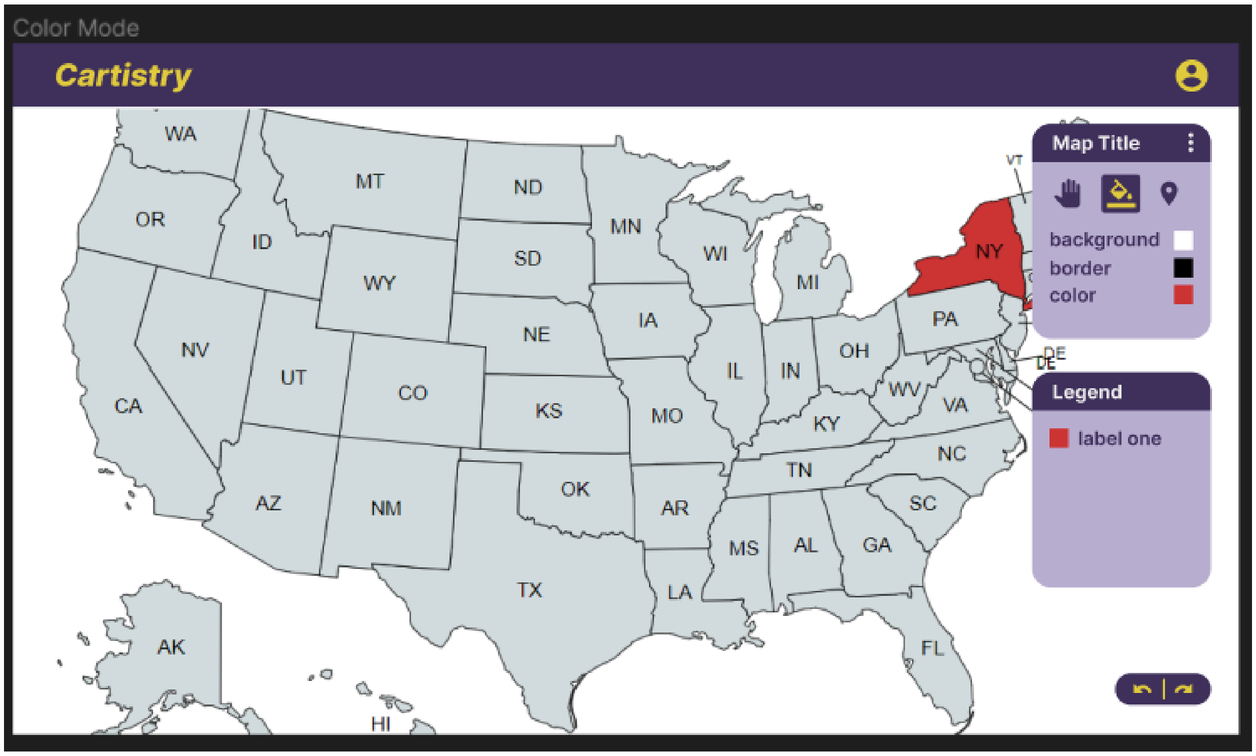
7


Edit Map Screen
Cartistry
Map Title
Legend
WA
OR
CA
NV
ID
MT
WY
UT
AZ
CO
NM
ND
SD
NE
KS
OK
TX
MN
IA
MO
AR
LA
WI
IL
MI
IN
OH
KY
TN
MS
AL
GA
FL
SC
NC
VA
WV
PA
NY
VT
NH
MA
CT
NJ
DE
MD
DC
AK
HI

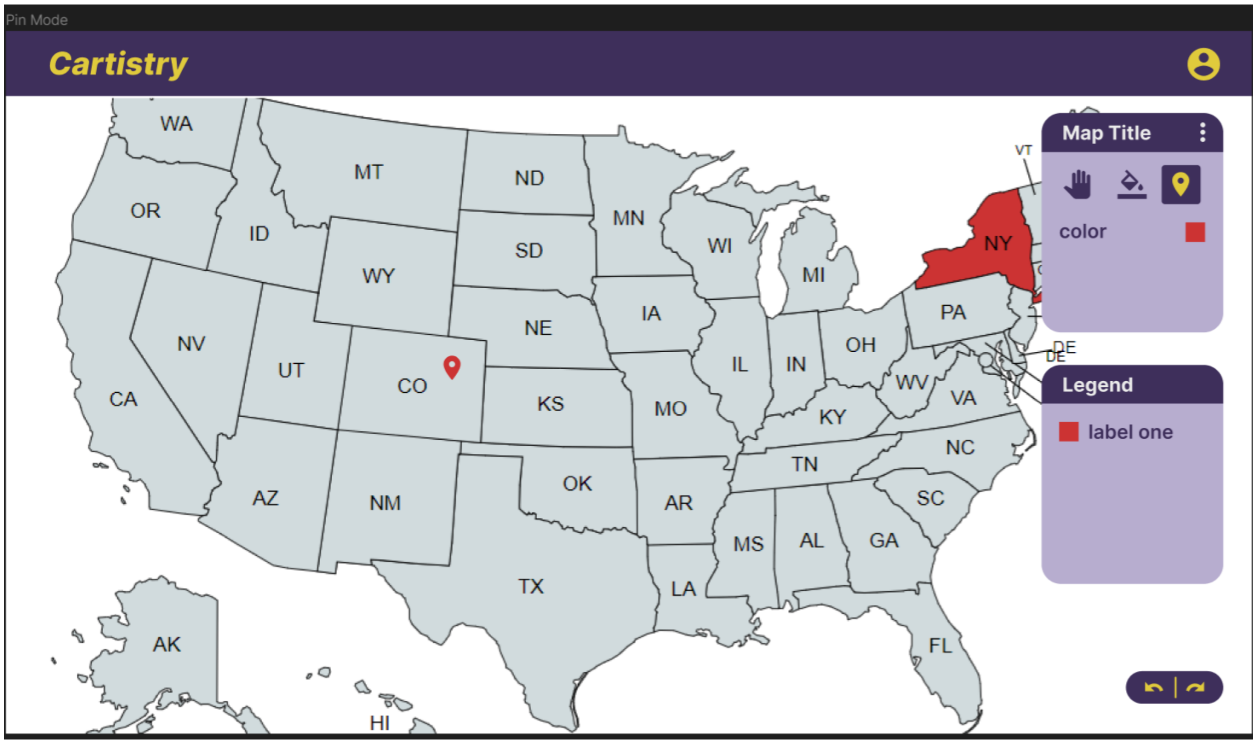
7.1


Pan Mode
Cartistry
Map Title
zoom
− | +
100
Legend
MO
IL
IN
OH
WV
KY
TN
AR
SC

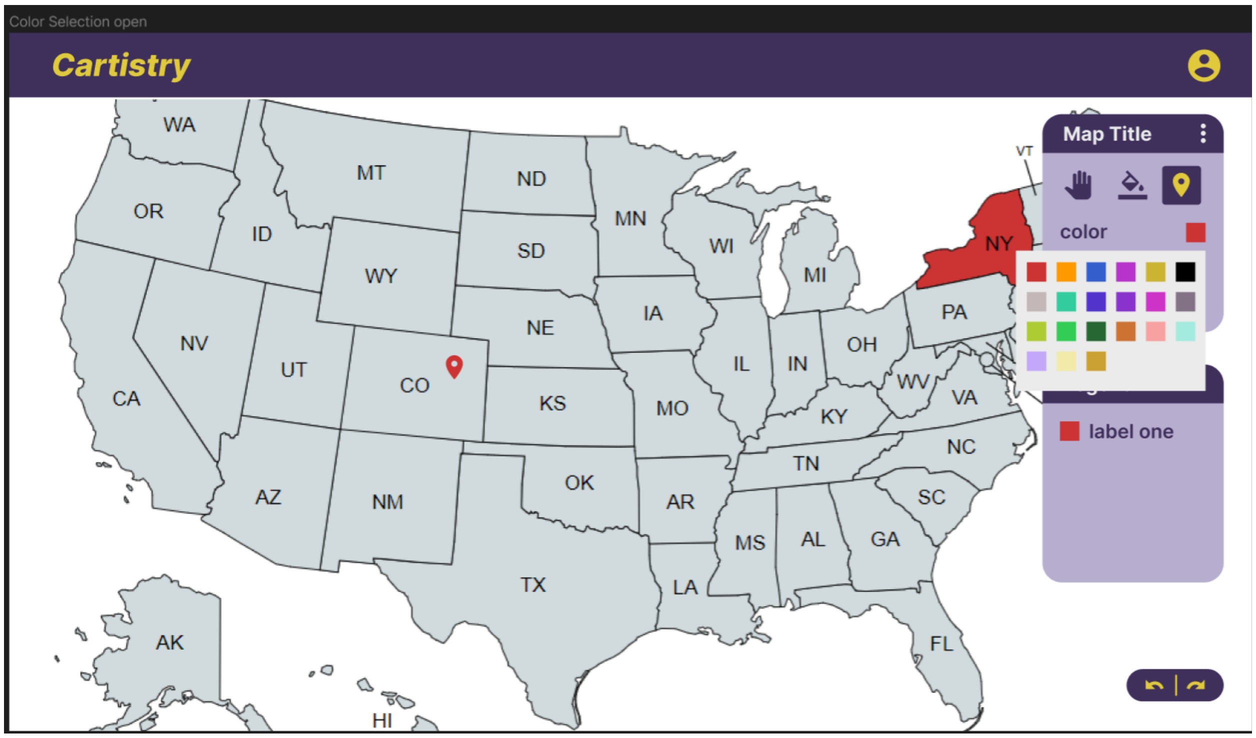
7.2


Color Mode
Cartistry
Map Title
background
border
color
Legend
label one
WA
OR
CA
NV
ID
MT
WY
UT
AZ
CO
NM
ND
SD
NE
KS
OK
TX
MN
IA
MO
AR
LA
WI
IL
MI
IN
OH
KY
TN
MS
AL
GA
FL
SC
NC
VA
WV
PA
NY
VT
DE
AK
HI

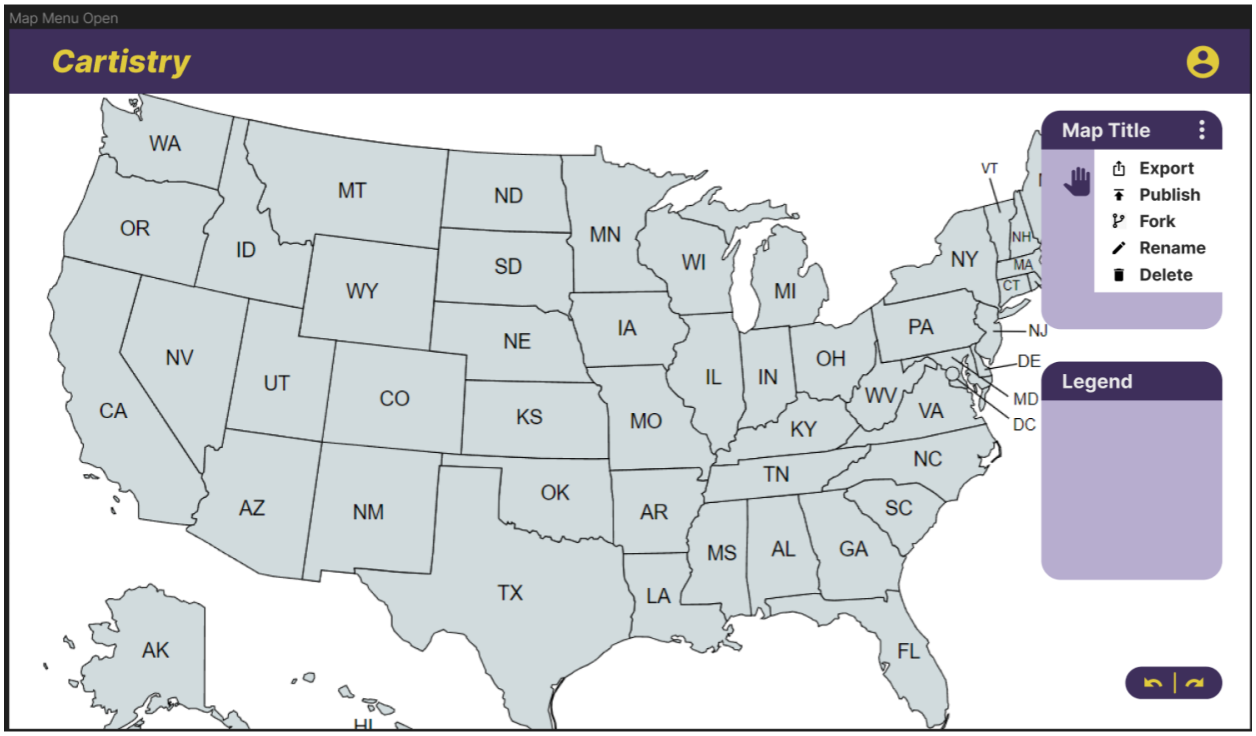
7.3


Pin Mode
Cartistry
Map Title
color
Legend
🟥 label one
WA
OR
CA
NV
ID
MT
WY
UT
AZ
NM
CO
ND
SD
NE
KS
OK
TX
MN
IA
MO
AR
LA
WI
IL
MS
MI
IN
KY
TN
AL
OH
GA
FL
WV
VA
NC
SC
PA
NY
VT
DE
AK
HI

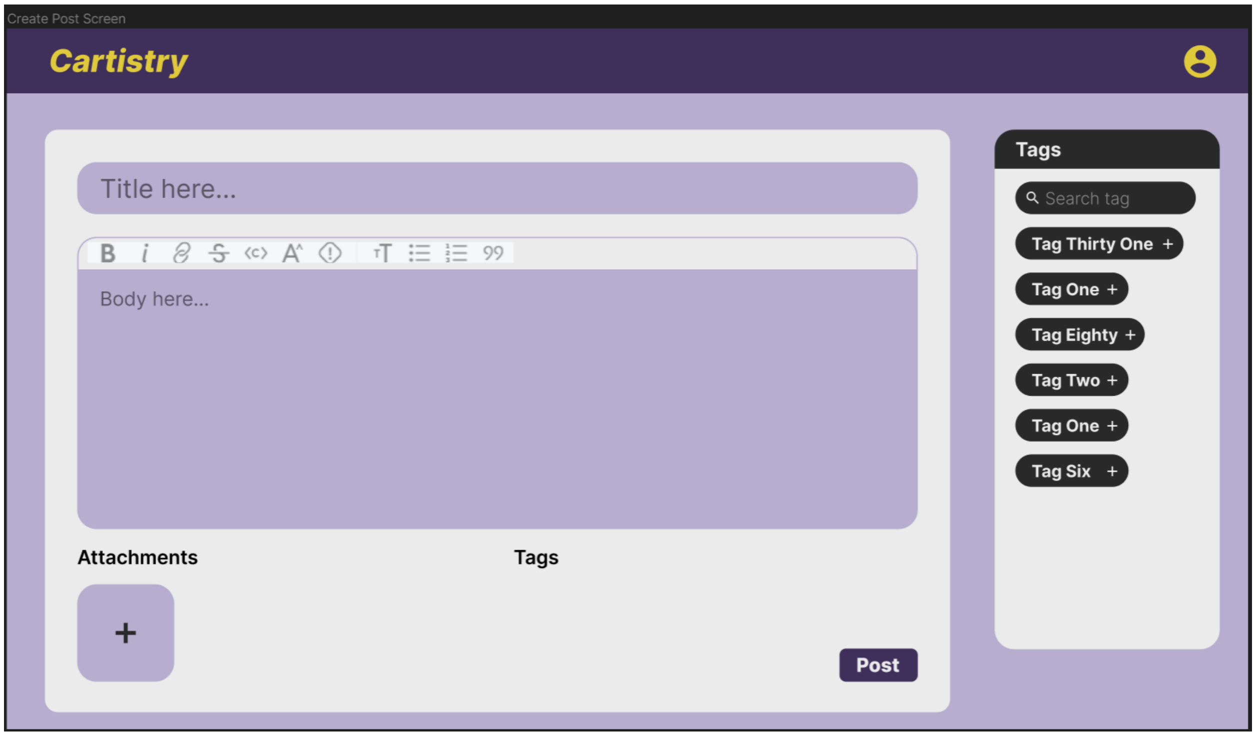
7.4


Color Selection open
Cartistry
WA
OR
CA
NV
ID
MT
WY
UT
CO
AZ
NM
ND
SD
NE
KS
OK
TX
MN
IA
MO
AR
LA
WI
IL
MI
IN
OH
KY
TN
MS
AL
GA
FL
SC
NC
VA
WV
PA
NY
VT
AK
HI
Map Title
color
label one

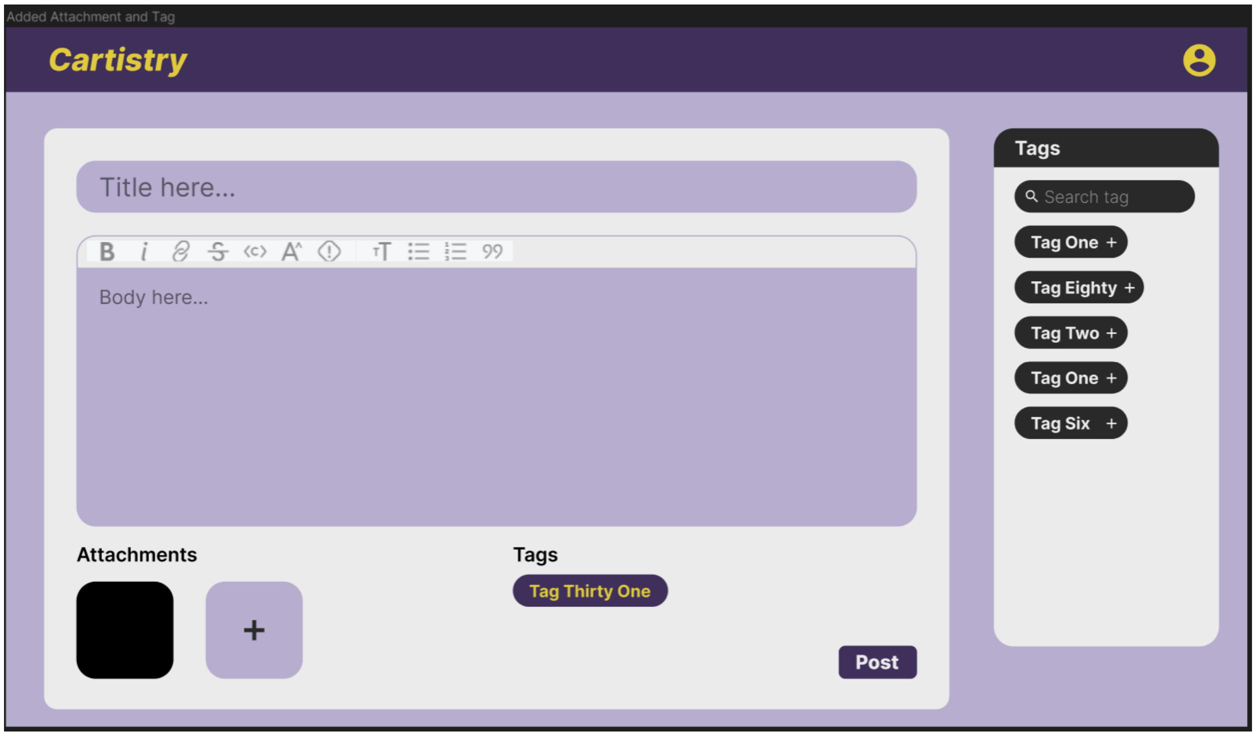
7.5


Map Menu Open
Cartistry
WA
OR
CA
NV
ID
MT
WY
UT
CO
AZ
NM
ND
SD
NE
KS
OK
TX
MN
IA
MO
AR
LA
WI
IL
MI
IN
OH
KY
TN
MS
AL
GA
FL
SC
NC
VA
WV
PA
NY
VT
NH
MA
CT
NJ
DE
MD
DC
AK
HI
Map Title
Export
Publish
Fork
Rename
Delete
Legend

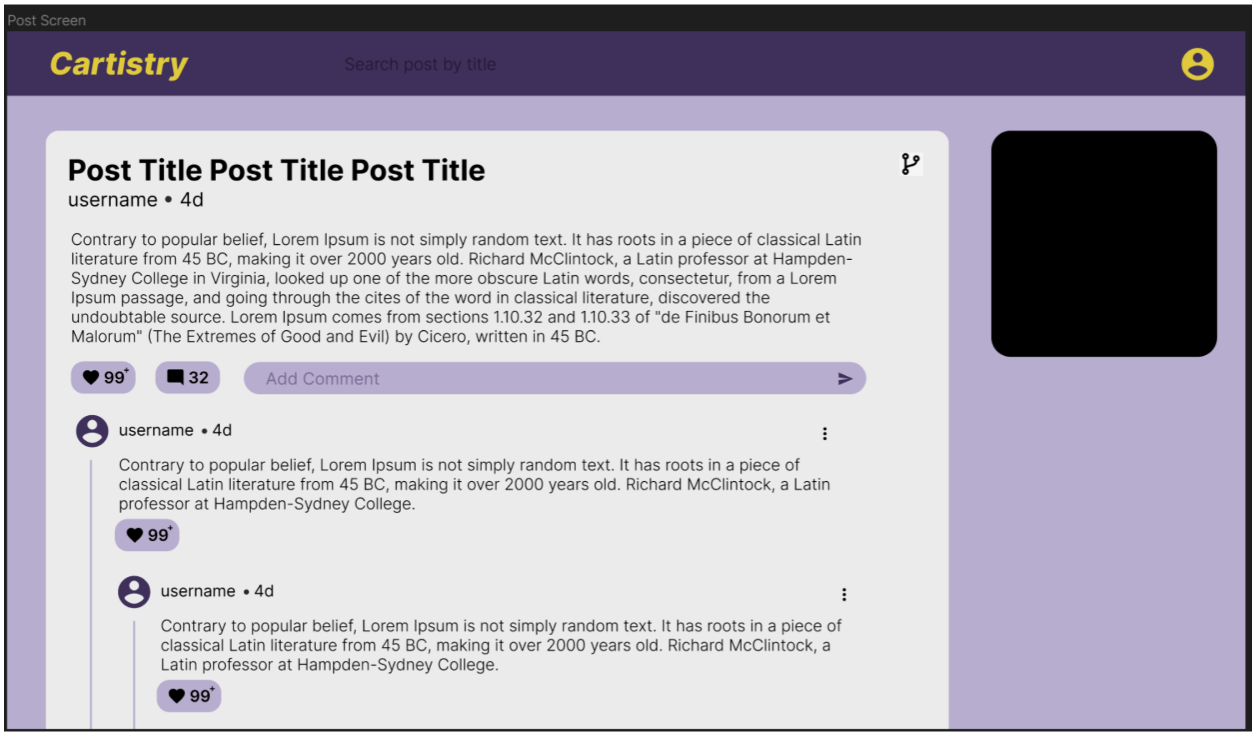
8


Create Post Screen
Cartistry
Title here...
Body here...
Attachments
Tags
+
Post
Tags
Search tag
Tag Thirty One +
Tag One +
Tag Eighty +
Tag Two +
Tag One +
Tag Six +

8.1


Added Attachment and Tag
Cartistry
Title here...
Body here...
Attachments
Tags
Tag Thirty One
+
Post
Tags
Search tag
Tag One +
Tag Eighty +
Tag Two +
Tag One +
Tag Six +

9



Post Screen

Post Title Post Title Post Title

username • 4d

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC.

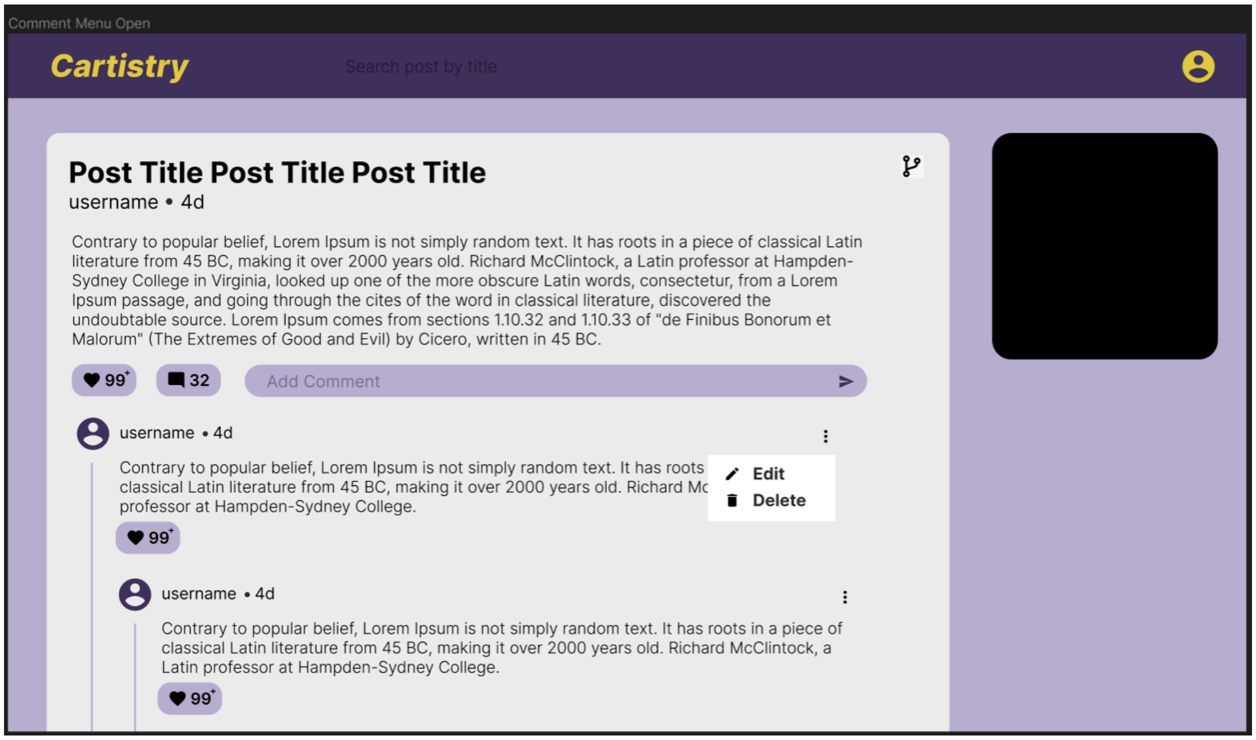
username • 4d

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College.

username • 4d

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College.

9.1



Comment Menu Open

Post Title Post Title Post Title

username • 4d

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC.

username • 4d

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots classical Latin literature from 45 BC, making it over 2000 years old. Richard Mc professor at Hampden-Sydney College.

username • 4d

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College.

Cartistry Data Model

DATA DICTIONARY BEGINS ON PAGE 3.

User

```
const userSchema = new Schema(
    {
        userName: { type:String, required: true, unique: true},
        email: { type: String, required: true, unique: true},
        passwordHash: { type: String, required: true },
        posts: [{type: ObjectId, ref: 'Post'}],
        mapsMetdadata: [{type: ObjectId, ref: 'MapMetadata'}],
        likedPosts: [{type: ObjectId, ref:'Post'}],
        untitledCount: {type:Number, default: 0},
        duplicateCount: {type:Number, default: 0},
        isAdmin: {type: Boolean, default: false},
         timeOfLastPasswordResetRequest: {type: Date}
    },
    { timestamps: true },
)
```

Post

```
const postSchema = new Schema(
    {
        title: { type: String, required: true },
        owner: { type: ObjectId, ref:'User', required: true },
        ownerUserName: {type: String, required: true},
        thumbnail: {
            imageData: Buffer,
            contentType: String,
        },
        comments: { type: [{
            authorUserName: String,
            text: String,
             publishDate: {type: Date},
        }], default: []},
        likes: {type: Number, default: 0},
        publishDate: {type: Date},
    },
    { timestamps: true },
)
```

Map Metadata

```
const mapMetadataSchema = new Schema(
    {
        title: { type: String, required: true },
        owner: { type: ObjectId, ref:'User', required: true },
        ownerUserName: {type: String, required: true},
        thumbnail: {
            imageData: Buffer,
            contentType: String,
            required: true
        },
        lastSaved: {type: Date},
        ownerFavorited: {type: Boolean, default: false},
        forks: {type: Number, default: 0},
        mapData: {type: ObjectId, ref:'MapData', required: true},
        isPrivated: {type: Boolean, default: true},

    },
    { timestamps: true },
)
```

Map Data

```
const mapDataSchema = new Schema(

    {
        mapMetadata: {type: ObjectId, ref:'MapMetadata', required: true},
        geoJSON: {type: Object, required: true},
        proprietaryJSON: {
            templateType: {type: String, required: true},
            legend: {
                title: {type: String},
                keyValueLabels: {type: [{
                    key: {type: String},
                    value: {type: String}
                }]}
            },
            gradientData: {
                primaryColor: {type: Number},
                minScale: {type: Number},
                maxScale: {type: Number},
                sections: {type: Number},
            }
        }
    }
)
```

Data Dictionary

This Data Dictionary has been sorted lexicographically first by Schema Name, and then by fields and subcategories within those fields.

Map Data

FIELDNAME	DESCRIPTION	CONSTRAINTS
geoJSON	JSON used to represent the collection of features corresponding to the map data that will be rendered in the editor.	Backend server must verify that this is valid geoJSON before storing it as map data. Required
mapMetadata	ObjectId of the document that contains the metadata about this map (e.g. title, ownerUserName, etc)	ObjectId. Required.
proprietaryJSON	JSON used to represent information necessary	

	for loading all other components of the map editor aside from the map itself, e.g. the legend.	
proprietaryJSON.gradientData	Object that corresponds to the information required to display the gradient on the map and on the legend labels.	
proprietaryJSON.gradientData.maxScale	Number that corresponds to the maximum value for the gradient's scale.	Can be any valid float value, except for "-inf" or "+inf
proprietaryJSON.gradientData.minScale	Number that corresponds to the minimum value for the gradient's scale.	Can be any valid float value, except for "-inf" or "+inf"
proprietaryJSON.gradientData.primaryColor	Number that represents the 32 bit color value for the primary color of the gradient	Number must represent a valid color.
proprietaryJSON.gradientData.sections	Number that corresponds to how many sections the gradient will be divided into for the legend. In other words, how many shades of colors the gradient will be divided into.	Integer greater than 0
proprietaryJSON.legend	Object that contains information about the map legend	
proprietaryJSON.legend.keyValueLabels	Array of Objects that corresponds to the labels of the legend	
proprietaryJSON.legend.keyValueLabels.key	String that corresponds to the key of a given label (e.g. color)	
proprietaryJSON.legend.keyValueLabels.value	String that corresponds to the value of a given	

	label	
proprietaryJSON.legend.title	String that corresponds to the title of the map legend	
proprietaryJSON.templateType	String used to represent the type of template this map originates from.	String. Required. Can only take on the following possible values: “bin”, “heat”, “subway”, “cadastral”, “landmark”

Map Metadata

forks	Number used to represent the amount of times this map has been forked by other users.	Integer value greater than or equal to 0
isPrivated	Boolean value representing whether or not this map is private and available for other users to view.	Boolean
lastSaved	Date corresponding to when the last time the MapData was saved via the save button in the Map editor.	Is a number representing the date and time in milliseconds since the Unix epoch (January 1, 1970, 00:00:00 UTC)
mapData	ObjectId corresponding to the MapData containing geoJSON and other data used to render the map graphics.	ObjectId. Required
owner	The ObjectId corresponding to the owner of the post, which is a User.	ObjectId. Required
ownerFavorited	Boolean value to represent whether or not the owner has favorited the map corresponding to this metadata. Used for filtering favorited cards in the	Boolean

	MyMaps page.	
ownerUserName	String value corresponding to the userName of the owner of the post.	String. Required
thumbnail.contentType	String that specifies what type of imageData the thumbnail is	“image/jpeg” and “image/png” are the only valid values.
thumbnail.imageData	Buffer of bytes corresponding to image data.	Bytes must be a valid png or jpeg image.
title	String value corresponding to the title of the map	String. Required. Must be unique in that user’s collection (e.g. Untitled0, Untitled1)

Post

comments	Array of objects corresponding to comments	Initially empty
comments[i].authorUserName	String value corresponding to the userName of the author of the comment	String
comments[i].text	String value corresponding to the text contents of the comment	String
likes	Number value corresponding to the number of likes on the post	Integer greater than or equal to 0
owner	The ObjectId corresponding to the owner of the post, which is a User.	ObjectId. Required
ownerUserName	String value corresponding to the userName of the owner of the post.	String. Required

publishDate	Date corresponding to the time when this post was published and made publicly available to other users.	Is a number representing the date and time in milliseconds since the Unix epoch (January 1, 1970, 00:00:00 UTC)
thumbnail.contentType	String that specifies what type of imageData the thumbnail is	“image/jpeg” and “image/png” are the only valid values.
thumbnail.imageData	Buffer of bytes corresponding to image data.	Bytes must be a valid png or jpeg image.
title	String value corresponding to the title of the post.	String. Required.

User

email	Unique email specified by the user during account creation. Used for the purpose for logging in and recovering forgotten passwords.	String. Required. Cannot create a User with an email that already exists on another account
isAdmin	Boolean value referring to whether or not the user is an Admin user.	Can only be set manually, not through a public API endpoint.
likedPosts	Array of ObjectIds corresponding to posts that the user has liked.	Can contain Objectids from posts of any user.
mapsMetdadata	Array of ObjectIds corresponding to map metadata that the user has created. Usage will mainly be for displaying thumbnails/cards for the user’s “MyMaps” page.	Array of ObjectId
passwordHash	The hash of the user’s	String. Required. Must only

	password that will be used for verification against the provided password whenever the user attempts to login.	be generated and stored on the backend server
posts	Array of ObjectIds corresponding to posts that the user has published.	Must only contain ObjectIds from posts that the user has created.
timeOfLastPasswordResetRequest	Date representing the last time a user has requested a password reset. Used to check for whether or not the password reset link sent to the user's email has expired.	Is a number representing the date and time in milliseconds since the Unix epoch (January 1, 1970, 00:00:00 UTC)
userName	Unique name specified by the user during account creation. Will be displayed on public posts/maps.	String. Required. Cannot create a User with a userName that already exists on another account

CARTISTRY SOFTWARE MODEL

Similar Problems

Mapchart | mapchart.net

Overview:
MapChart.net is a web-based application tailored for crafting custom geographical charts. By delving into MapChart.net's features and capabilities, we can derive actionable implementation strategies for our map editing web app.

Key Features and Implementation Strategies:

Basic Map Creation:

- Rationale: Users select and color countries or states using a chosen palette.
- Implementation: Develop a responsive color-picking tool and integrate it with the app's geographical database. Use vector graphics to ensure smooth rendering of countries and states when colored.

Legend Customization:

- Rationale: Legends are auto-generated based on user color choices.
- Implementation: Implement a dynamic legend generator that reacts to user color choices.

Advanced Interactivity:

- Rationale: Context menus appear upon right-clicking, offering customization options.
- Implementation: Design a context-sensitive menu system that presents options based on the current state and selection on the map.

Keyboard Shortcuts and Efficiency Tools:

- Rationale: MapChart uses keyboard shortcuts like undo or color removal.
- Implementation: Embed a keyboard event listener and create a shortcut repository, allowing users to edit efficiently.

Save and Load Functionality:

- Rationale: Work can be saved as a local .txt file and later uploaded for continuation.
- Implementation: Design a serialization process to save map configurations into custom json files and a deserialization method to load and reconstruct saved maps.

Zooming and Focusing:

- Rationale: Users can zoom in/out and navigate using both the mouse and keyboard.
- Implementation: Integrate a zoom library (like D3.js) and develop custom navigation controls for precise zooming and panning.

Pattern Usage:

- Rationale: Patterns can be chosen for nuanced data representation.
- Implementation: Extend the color-picking tool to include pattern selections. Store pattern graphics as SVGs for scalability.

Excel Integration:

- Rationale: For extensive maps, an Excel tool aids in fast configuration creation.
- Implementation: Design an Excel parser to read configuration files, converting them to the app's data format for quick bulk editing.

Conclusion:
MapChart.net's rich feature set provides a blueprint from which our map editing web app can benefit. Adapting these into tailored implementations will ensure a robust and user-friendly mapping platform.

Playlister | CSE 316 FALL 2022

MERN stack Web App

Playlister, built to create, manage, and share music playlists, its intricate functionalities and user-focused design provide essential lessons for our map editing application. By understanding Playlister's successful methodologies, we aim to enhance the development of our map tool, drawing parallels where appropriate and adapting best practices.

Dynamic User Interface:

- Rationale: The Playlister uses an AppBanner for interactive menu options and modal pop-ups (CurrentModal). Similarly, our map editing app benefits from an intuitive and dynamic interface for seamless user interaction with map elements.
- Implementation: Incorporate a toolbar in the map editor, reminiscent of the Edit Toolbar, to house tools for actions such as drawing routes, placing markers, and adjusting map layers.

Contextual Actions:

- Rationale: Playlister manages playlist content through GlobalStoreActionType that defines distinct actions (e.g., CREATE, DELETE). In a parallel fashion, our map editor requires defined actions to manage its map elements.

- Implementation: Introduce actions like ADD_MARKER, DELETE_ROUTE, ZOOM_IN, and others. Structuring these actions this way facilitates the management and triggering of functionalities based on user inputs.

Advanced User Interactions:

- Rationale: Playlister incorporates the jTSPS system for transactional actions, offering users the ability to undo or redo actions, enriching the user experience and minimizing errors.
- Implementation: Embed a transaction system in the map editor, allowing users to undo a recently placed marker or redo a removed route, proving essential for intricate map editing tasks.

Server Interactions and Data Management:

- Rationale: In Playlister, the client communicates with the server using the GlobalStoreHttpRequestApi to retrieve or modify playlist data. In a similar vein, efficient server interactions are needed in our map editor to fetch or store map data.
- Implementation: Design APIs that facilitate users in saving their current map state, retrieving saved maps, or updating existing ones, ensuring robust communication between the client and server.

User Authentication and Management:

- Rationale: Playlister effectively manages user authentication via AppBanner and AuthContextProvider. For our map editing app, user accounts are essential to store and manage personalized maps.
- Implementation: Integrate authentication protocols into the map editor. This integration should allow users to establish accounts, preserve their maps, and retrieve them from any device.

By evaluating the Playlister application's features and structures, we've garnered valuable insights and best practices that will inform the development and enhancement of our map editing web app.

Fake Stack Overflow | CSE 316 FALL 2023

MERN stack Web App

In a previous project, a Stack Overflow clone, we developed a community-driven platform where users could post, edit, and interact with questions and answers. This project shares several similarities with the current web app project focused on map editing and community sharing. The following aspects from the Stack Overflow clone can be utilized in the current project:

Community Engagement:

- The Stack Overflow clone included features for user profiles, content posting, and community interaction. These elements can be leveraged for the current project to facilitate community sharing and engagement with maps.

Content Management:

- The project allowed users to create, edit, and share questions and answers. This content management system can be adapted to suit the needs of the map editing application, enabling users to create, edit, and share maps.

User Authentication and Profiles:

- User authentication and profiles were integral to the Stack Overflow clone, tracking each user's content and interactions. This system can be reused for the current project to manage user-created maps and interactions.

Search and Filter:

- A search and filter functionality was implemented to help users find specific questions and answers. This functionality can be adapted and incorporated into the map editing application, allowing users to easily search and filter through maps.

Complete Technology Set

Front-end APIs

axios	Used to send HTTP requests and receive HTTP responses to/from our back-end server
jsondiffpatch	Used to calculate the difference between the geoJSON after edits in order to have efficient saving of users' map data
jstps	Used to keep track of the users transactions while performing map editing in order to enable undo/redo
jszip	Used to send uploaded mapData to backend server as a zipfile, as well as receive zipped data from the backend server.
react	Used to render our front-end application view
react-leaflet	Used to render the users' geoJSON map data in the application, as well as providing the api for the user to interact with the map (e.g. via

	clicking)
react-router	Used to abstract different sections of our single page application into multiple routes specified by url path.
turfjs	Library used for map graphics editing and creating geoJSON data corresponding to lines, points, polygons, etc.

Back-end APIs

adm-zip	Used to receive zipped map data from the client during a map upload
archiver	Used to send zipped mapdata to the client during a map export
bcryptjs	Used for user registration and authentication, providing capabilities such as generating password hashes and comparing hashes.
body-parser	Used to parse the body of an HTTP request into a javascript object
cookie-parser	Used to parse the cookie sent in an HTTP request into a javascript object
cors	Used to control the cors policy for access to routes on our backend server
dotenv	Used to parse .env files stored on our backend server, usually used to store private information such as API keys for mongoDB
express	Used as the framework for building the routes and controllers for our backend server
jsondiffpatch	Used for saving the edited version for a users' geoJSON by applying the diff data provided by the user to their geoJSON
jsonwebtoken	Used to verify the user has logged in by reauthenticating them via verifying their given json web token.

mapbox/togeojson	Used to parse kml files in order to convert them to geoJSON
mongoose	Used for communicating and maintaining a connection with our backend database, mongoDB. Also used to create schemas for the data stored in the database, as well as having the model API used to retrieve, update, and delete documents from the database.
multer	Middleware responsible for handling multipart/form-data, which mainly deals with image uploads for Posts
shpjs	Used to parse zipped shapefiles in order to convert them to geoJSON
xmldom	Used for parsing .kml files sent by the client so that mapbox/togeojson can convert it to geoJSON

Training Verification

Angelo Panopio

Background:
- Angelo successfully created a clone of StackOverflow in the spring semester of 2023 during the class CSE 316 . This experience has equipped him with hands-on knowledge about React and related technologies.

Deployment & Work Demonstration:
- Angelo's StackOverflow clone is a testament to his proficiency with React and related technologies. You can view his implementation at the following GitHub repository: https://github.com/angelopanopio/cse316spr23.

Technologies Used:
- React.js for the frontend UI components.
- Node.js and Express.js for backend implementation.
- MongoDB for database operations.

Jonathan Ng

Background:

- Jonathan successfully created the Playlister application in Fall 2022 semester, giving him the experience required for building a MERN stack application. Furthermore, during his internship in summer 2023 at Capital One, he got hands-on experience in deploying an API to AWS Lambda used for load generation and performance testing other internal company applications.

Deployment & Work Demonstration:
https://github.com/JWaibong/Playlister-final

Technologies Used:

- React.js for the frontend UI components
- Express for backend server implementation
- MongoDB for database

Aaron Li

Background:

- Aaron created the Playlister application in Fall 2022 semester, giving him the experience required for building a MERN stack application. This along with other projects has given him hands-on experience. He has an in-depth understanding and knowledge of HTML and CSS which allows him to work quickly.
- wowie

Deployment & Work Demonstration:
https://github.com/aaronli03/final-project

Technologies Used:

- React.js for the frontend UI components
- Express for backend server implementation
- MongoDB for database

Jenny Bao

Background:

- Jenny successfully created the Playlister application in Fall 2022 semester, giving her the experience required for building a MERN stack application. Furthermore, during her internship in Summer 2023 at Citi, she got hands-on experience with creating features using React while being on the frontend team. Also, she has experience with using an Express backend with a React frontend during her Summer 2022 internship where she worked fullstack.

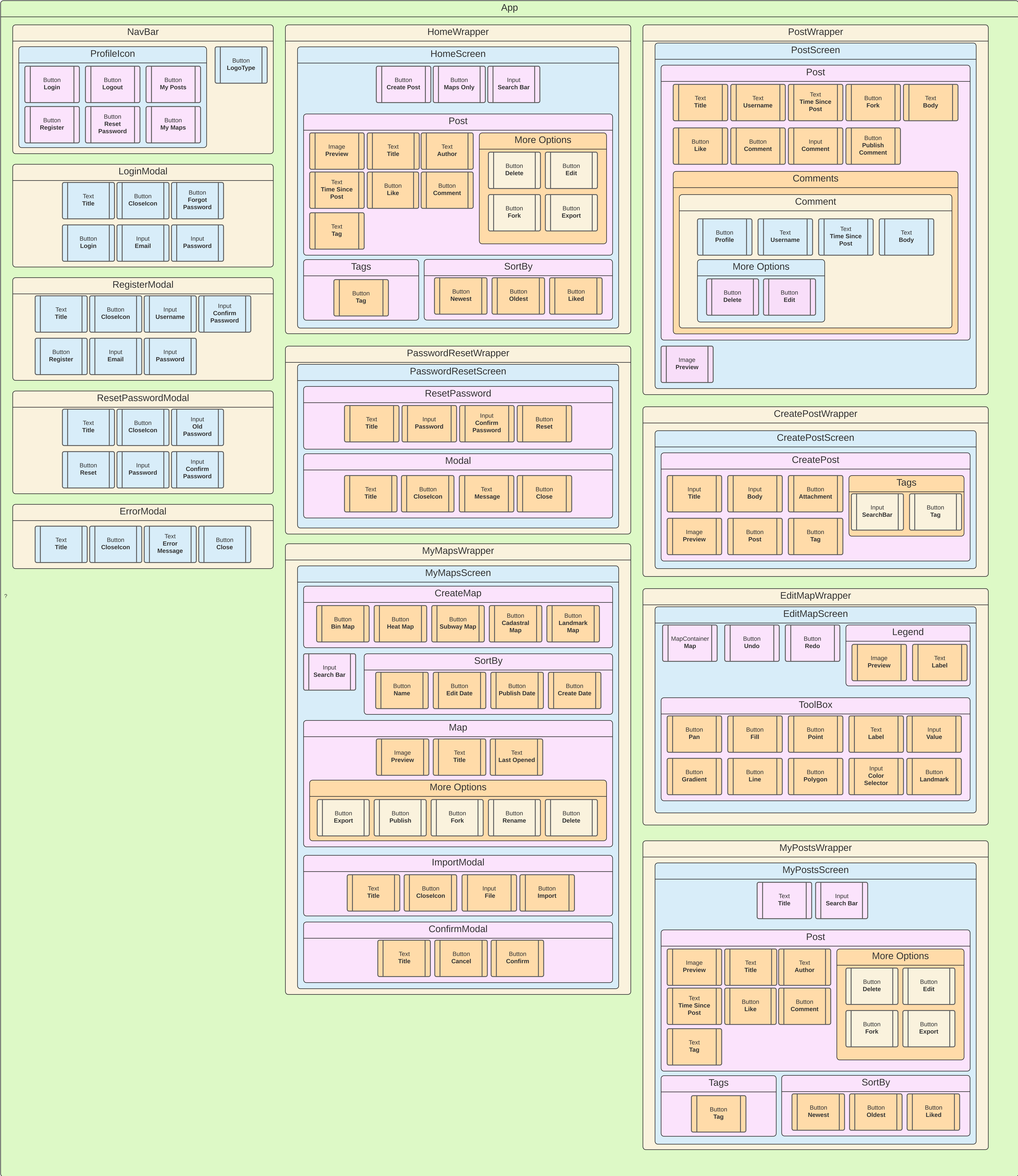
Deployment & Work Demonstration:
https://github.com/blueskies0038/cse316-playlister-project

Technologies Used:

- React.js for the frontend UI components
- Express for backend server implementation
- MongoDB for database

UML DIAGRAMS BEGIN ON NEXT PAGE

Note: Auth Diagram remains mostly unchanged from CSE 316 Playlister


App
NavBar
ProfileIcon
Button Login
Button Logout
Button My Posts
Button Register
Button Reset Password
Button My Maps
Button LogoType
LoginModal
Text Title
Button CloseIcon
Button Forgot Password
Button Login
Input Email
Input Password
RegisterModal
Text Title
Button CloseIcon
Input Username
Input Confirm Password
Button Register
Input Email
Input Password
ResetPasswordModal
Text Title
Button CloseIcon
Input Old Password
Button Reset
Input Password
Input Confirm Password
ErrorModal
Text Title
Button CloseIcon
Text Error Message
Button Close
?
HomeWrapper
HomeScreen
Button Create Post
Button Maps Only
Input Search Bar
Post
Image Preview
Text Title
Text Author
Text Time Since Post
Button Like
Button Comment
Text Tag
More Options
Button Delete
Button Edit
Button Fork
Button Export
Tags
Button Tag
SortBy
Button Newest
Button Oldest
Button Liked
PasswordResetWrapper
PasswordResetScreen
ResetPassword
Text Title
Input Password
Input Confirm Password
Button Reset
Modal
Text Title
Button CloseIcon
Text Message
Button Close
MyMapsWrapper
MyMapsScreen
CreateMap
Button Bin Map
Button Heat Map
Button Subway Map
Button Cadastral Map
Button Landmark Map
Input Search Bar
SortBy
Button Name
Button Edit Date
Button Publish Date
Button Create Date
Map
Image Preview
Text Title
Text Last Opened
More Options
Button Export
Button Publish
Button Fork
Button Rename
Button Delete
ImportModal
Text Title
Button CloseIcon
Input File
Button Import
ConfirmModal
Text Title
Button Cancel
Button Confirm
PostWrapper
PostScreen
Post
Text Title
Text Username
Text Time Since Post
Button Fork
Text Body
Button Like
Button Comment
Input Comment
Button Publish Comment
Comments
Comment
Button Profile
Text Username
Text Time Since Post
Text Body
More Options
Button Delete
Button Edit
Image Preview
CreatePostWrapper
CreatePostScreen
CreatePost
Input Title
Input Body
Button Attachment
Image Preview
Button Post
Button Tag
Tags
Input SearchBar
Button Tag
EditMapWrapper
EditMapScreen
MapContainer Map
Button Undo
Button Redo
Legend
Image Preview
Text Label
ToolBox
Button Pan
Button Fill
Button Point
Text Label
Input Value
Button Gradient
Button Line
Button Polygon
Input Color Selector
Button Landmark
MyPostsWrapper
MyPostsScreen
Text Title
Input Search Bar
Post
Image Preview
Text Title
Text Author
Text Time Since Post
Button Like
Button Comment
Text Tag
More Options
Button Delete
Button Edit
Button Fork
Button Export
Tags
Button Tag
SortBy
Button Newest
Button Oldest
Button Liked

HTTP Request
HTTP Response
jsTPS
jsTPS_Transaction
Server

NavBar
```
const { auth } = useContext(AuthContext);
const { store } = useContext(GlobalStoreContext);

// The menu's anchor element, i.e. where it will appear
const [ anchorEl, setAnchorEl ] = useState(null);

// Keeps track of if the menu is open or not
const isMenuOpen = Boolean(anchorEl);
```
```
// Responds to logotype or home button being pressed
const handleHome = () => {...

// Responds to click on avatar to open drop-down menu
const handleProfileMenuOpen = (event) => { ...

// Responds to click away from menu, which closes it
const handleMenuClose = () => { ...

// Responds to click on Create New Account menu item
const handleRegister = () => { ...

// Responds to click on Login menu item
const handleLogin = () => { ...

// Responds to click on Logout menu item
const handleLogout = () => { ...

// Responds to pressing Reset Password
const handleResetPassword = () => {...

// Responds to pressing My Maps
const handleMyMaps = () => {...

// Responds to pressing My Posts
const handleMyPosts = () => {...
```

HomeWrapper
```
const { auth } = useContext(AuthContext);
```

HomeScreen
```
const {store} = useContext(GlobalStoreContext);
const [search, setSearch] = useState("");
```
```
// Respond to clicking on create post button (only
available when logged in)
const handleCreateNewPost = () => {...

// Responds to clicking on Maps only button
const handleMapsOnly = () => {...

// Responds to Sort By being pressed
const handleSortByOpen = () => {...

// Responds to a sort being chosen
const handleSort = () => {...

// Responds to typing in the search bar
const handleSearchChange = () => {...

// Responds to pressing enter on search bar
const handleSearch = ()  => {...
```

Post
```
const { store } = useContext(GlobalStoreContext);
const [likes, setLikes] = useState(0)
const [comments, setComments] = useState({});
const [comment, setComment] = useState("");
```
```
// Responds to like button being pressed
const handleLike = () => {...

// Responds to comment text being changed
const handleCommentChange = () => {...

// Responds to comment text being submitted
const handleCommentSubmit = () => {...

// Responds to username being pressed
const handleProfileClick = () => {...

// Responds to export button
const handleExport = () => {...

// Responds to fork button
const handleFork = () => {...

// Responds to delete button (admin/owner)
const handleDelete () => {...

//Responds to edit button
const handleEdit = () => {...
```

PostCard
```
const { store } = useContext(GlobalStoreContext);
const [likes, setLikes] = useState(0)
```
```
//Responds to like button being pressed
const handleLike = () =>

//Responds to username bring pressed
const handleProfileClick = () => {...

//Responds to card being pressed
//Called when comment button clicked as well
const handleOpenPost = () =. {...
```

Comment
```
const { store } = useContext(GlobalStoreContext);
const [likes, setLikes] = useState(0);
const [editActive, setEditActive] = useState(false);
const [comment, setComment] = useState("");
```
```
// Responds to like button being pressed
const handleLike = () => {...

// Responds to edit button being pressed
const handleEdit = () => {...

// Responds to comment text being changed
const handleCommentChange = () => {...

// Responds to comment being submitted
const handleCommentSubmit = () => {...

// Responds to delete button (owner/admin)
const handleDelete = () => {...

//Responds to username being pressed
const handleProfileClick = () => {...
```

PostEditor
```
const { store } = useContext(GlobalStoreContext);
const [title, setTitle] = useState("");
const [body, setBody] = useState("");
const [attachments, setAttachments] = useState({});
const [tags, setTags] = useState({});
const [tagsSearch, setTagsSearch] = useState("");
```
```
// Responds to title text change
const handleTitleChange = () => {...

// Responds to body text change
const handleBodyChange = () => {...

// Responds to add attachment
const handleAddAttachment = () => {...

// Responds to delete attachment
const handleDeleteAttachment = () => {...

// Responds to add tag
const handleAddTag = () => {...

// Responds to delete tag
const handleDeleteTag = () => {...

// Responds to change in input for tags search
const handleSearchChange = () => {...

// Responds to submitting tags search
const handleSearchSubmit = () => {...

// Responds to post button being clicked
const handlePostSubmit = () => {...
```

MyMaps
```
const { store } = useContext(GlobalStoreContext);
const [maps, setMaps] = useState({});
const [search, setSearch] = useState("");
const [showModal, setShowModal] = useState(false);
```
```
// Responds to create map button
const handleCreateMap = () => {...

// Responds to search text being changed
const handleSearchChange = () => {...

// Responds to submitting maps search
const handleSearchSubmit = () => {...

// Respnds to sort criteria being selected
const handleSort = () => {...
```

MapCard
```
const { store } = useContext(GlobalStoreContext);
const [showOpts, setShowOpts] = useState(false);
const [editActive, setEditActive] = useState(false);
const [name, setName] = useState("");
const [showModal, setShowModal] = useState(false);
```
```
// Responds to delete
const handleDelete = () => {...

// Responds to rename text change
const handleRenameChange = () => {...

// Responds to submitting rename text
const handleRenameSubmit = () => {...

// Responds to fork
const handleFork = () => {...

// Responds to publish
const handlePublish = () => {...

// Responds to export
const handleExport = () => {...

// Responds to opening the map (for editing)
const handleOpenMap = () => {...

// Responds to changing privacy
const handleTogglePrivacy = () => {...
```

MyPosts
```
const { store } = useContext(GlobalStoreContext);
const [search, setSearch] = useState("");
```
```
// Responds to typing in the search bar
const handleSearchChange = () => {...

// Responds to pressing enter on search bar
const handleSearch = ()  => {...

// Responds to tag being pressed
const handleToggleTag = () => {...

// Responds to a sort being chosen
const handleSort = () => {...
```

ResetForgotPasswordScreen
```
const { store } = useContext(GlobalStoreContext);
const [formData, setFormData] = useState({
    password: "",
    confirmPassword: "",
});
const [showError, setShowError] = useState(false);
```
```
// Responds to form change
const handleFormChange = () => {...

//Responds to form submit
const handleFormSubmit = () => {...
```

MapEditor
```
const { store } = useContext(GlobalStoreContext);
const [mapData, setMapData] = useState(null);
const [currentTool, setCurrentTool] = useState(null);
const [legendTitle, setLegendTitle] = useState(null);
const [legendKV, setLegendKV] = useState([]);
const [currentColor, setCurrentColor]  = useState(null)
const [minScale, setMinScale] = useState(0);
const [maxScale, setMaxScale] = useState(0);
const [sections, setSections] = useState(1);
// used for line tool, polygon tool
const [currentPoints, setCurrentPoints] = setState([ ]);
```
```
// Rsponds to undo button being pressed
const handleUndo = () => {...

// Responds to redo button being pressed
const handleRedo  = () => {...

// Responds to label text being double clicked
const handleLabelEdit = () => {...

// Responds to label text being updated
const handleLabelChange = () => {...

//Responds to label text change submit
const handleLabelSubmit = () => {...

const handleSaveMap = () => {...

const handlePanButtonClick = () => {...

const handleColorButtonClick = () => {...

const handlePinButtonClick = () => {...

const handleGradientButtonClick = () => {...

const handleLineButtonClick = () => {...

const handlePolygonButtonClick = () => {...

const handleIconButtonClick = () => {...

const handleSetMin = () => {...
const handleSetMax = () => {...
const handleSetSections = () => {...
const handleSetColor = () => {...
const handleSetLegendTitle = () => {...

const handleMapClick = () => {...
// Responds to file being updated
const handleFileChange = () => {...

// Responds to import button being pressed
const handleImport = () => {...
```

Confirm Modal
```
const { store } = useContext(GlobalStoreContext);
```
```
// Responds to close button being pressed
const handleClose = () => {...

// Responds to confirm button being pressed
const handleConfirm = () => {...
```

Alert Modal
```
const { store } = useContext(GlobalStoreContext);
```
```
// Responds to close button being pressed
const handleClose = () => {...
```

GlobalStoreActionType
```
DELETE_MAP
DELETE_COMMENT
DELETE_POST
UPDATE_COMMENT
UPDATE_POST
LOAD_MAP
LOAD_MAP_CARDS
LOAD_POST
LOAD_ALL_POSTS
LOAD_MY_POSTS
CREATE_COMMENT
CREATE_POST
```

GlobalStoreContextProvider
```
const [store, setStore] = useState({

      currentModal : CurrentModal.NONE,
      mapCardsInfo: [ ],
      postCardsInfo: [ ],
      currentPostInfo: null,
      mapCardIndexMarkedForDeletion: null,
      mapCardMarkedForDeletion: null
      postCardIndexMarkedForDeletion: null,
      postCardMarkedForDeletion: null
      commentIndexMarkedForDeletion: null,
      commentMarkedForDeletion: null
});

const { auth } = useContext(AuthContext);
const tps : jsTPS;
const history = useHistory();
```
```
// Reducer function to update store state
const storeReducer = (action) => { ...


// Functions to call apis


//  to delete a specific map
store.DeleteMap = function(mapId) {...}

//  to delete a specific comment
store.DeleteComment = function(commentId) {...}

//  to delete a specific post
store.DeletePost = function(postId) {...}

//  to edit or update a comment's text
store.EditComment = function(commentId, newText) {...}

//  to edit or update a post's content
store.EditPost = function(postId, newContent) {...}

//  to load a specific map's details
store.LoadMap = function(mapId) {...}

//  to load all maps associated with a user
store.LoadMapCards = function(userId) {...}

//  to load a specific post's details
store.LoadPost = function(postId) {...}

//  to load all available posts
store.LoadAllPosts = function(filterOptions) {...}

//  to load all posts associated with a user
store.LoadMyPosts = function(userId) {...}

//  to create a new comment with specified details
store.CreateComment = function(newCommentDetails) {...}

//  to create a new post with specified details
store.CreatePost = function(newPostDetails) {...}


store.addEditFeaturePropertiesTransaction = function(newProperties,
oldProperties, featureIndex)

store.addCreateFeatureTransaction = function(newFeature, featureIndex)

store.addDeleteFeatureTransaction = function(feature, featureIndex)

...
```

EditFeaturePropertiesTransaction
```
store : GlobalStoreContext
newProperties: Object
oldProperties: Object
featureIndex : Number
```
```
constructor(initStore, initIndex, initFeature)
doTransaction()
undoTransaction()
```

CreateFeature_Transaction
```
store : GlobalStoreContext
index: Number
feature: Object
```
```
constructor(initStore, initIndex, initFeature)
doTransaction()
undoTransaction()
```

DeleteFeature_Transaction
```
store : GlobalStoreContext
index: Number
feature: Object
```
```
constructor(initStore, initIndex, initFeature)
doTransaction()
undoTransaction()
```

GlobalStoreHttpRequestApi
```
searchPostsByTitle(username) {
   method: GET
   route : /posts-api/posts/search-title/:title
   body : {
        limit: Number | undefined
   }
}

searchPostsByTags(tags) {
   method: GET
   route : /posts-api/posts/search-tags
   body : {
        tags: [String]
        limit: Number | undefined
   }
}

getPostsOwnedByUser(userId, limit) {
   method: GET
   route : /posts-api/posts/user/:userId
   body : {
        limit: Number | undefined
   }
}

getPost(postId) {
   method: GET
   route : /posts-api/posts/:id
   body : { }
}

getMostRecentPosts(limit) {
   method: GET
   route : /posts-api/posts/most-recent
   body : { limit: Number | undefined}
}

getMostLikedPosts(limit) {
   method: GET
   route : /posts-api/posts/most-liked
   body : { limit: Number | undefined }
}

createPost(title, textContent, images) {
   method: POST
   route : /posts-api/posts
   body : { // images will be sent via FormData
object
        title: String,
        textContent: String
   }
}

editPost(title, textContent, images) {
   method: PUT
   route : /posts-api/posts/:id
   body : { // images will be sent via FormData
object
        title: String,
        textContent: String
   }
}

updatePostLikes(postId) {
   method: PUT
   route : /posts-api/posts/:id/likes
   body : { }
}
commentOnPost(postId) {
   method: PUT
   route : /posts-api/posts/:id/comment
   body : {
          comment: String
   }
}

editComment(postId, index, comment) {
    method: PUT
    route: /posts-api/posts/:id/edit-comment
    body: { index: Number, comment: String }
}

deletePost(postId) {
   method: DELETE
   route : /posts-api/posts/:id/delete
   body : { }
}

deleteComment(postId, index) {
   method: DELETE
   route: /posts-api/posts/:id/comment
   body: { index: Number }
}


MAPS

exportMap(mapId) {
    method: GET
    route: /maps-api/maps/:id/export
    body: { }
}

getMapMetadataOwnedByUser() {
    method: GET
    route: /maps-api/map-metadata/:userId
    body: { }
}

getPublicMapMetadataOwnedByUser() {
    method: GET
    route:
/maps-api/public-map-metadata/:userId
    body: { }
}

getMapData(mapId) {
   method: GET
   route: /maps-api/maps/:id
   body: { }
}

uploadMap(formData) {

   method: POST
   route: /maps-api/maps/upload
   body: {
         // contains zipFile blob and actual
extension type (.shp, .json, .kml)
         formData: FormData
   }
}


forkMap(mapId) {
   method: POST
   route: /maps-api/maps/:id/fork
   body: { }
}

publishMap(mapId) {
    method: POST
    route: /maps-api/maps/:id/publish
    body: { }
}

favoriteMap(mapId) {
   method: PUT
   route: /maps-api/maps/:id/favorite
   body: { }
}

renameMap(mapId, name) {
   method: PUT
   route: /maps-api/maps/:id/rename
   body: { }
}

updateMapPrivacy(mapId, privacyStatus) {
    method: PUT
    route: /maps-api/maps/:id/change-privacy
    body: {
            privacyStatus: String
     }
}

deleteMap(mapId) {
    method: DELETE
    route: /maps-api/maps/:id/rename
    body: { }
}
```

PostsRouter
/posts-api
```
router.get('/posts/search-title/:title', PostsController.searchPostsByTitle)
router.get('/posts/search-tags', PostsController.searchPostsByTags)
router.get('/posts/user/:userId', PostsController.getPostsOwnedByUser)
router.get('/posts/:id', PostsController.getPost)
router.get('/posts/most-recent', PostsController.getMostRecentPosts)
router.get('/posts/most-liked', PostsController.getMostLikedPosts)


router.post('/posts', auth.verify, PostsController.createPost)

router.put('/posts/:id', auth.verify, PostsController.editPost)
router.put('/posts/:id/likes', auth.verify, PostsController.updatePostLikes)
router.put('/posts/:id/edit-comment', auth.verify,
PostsController.editComment)
router.put('/posts/:id/comment', auth.verify,
PostsController.commentOnPost)

router.delete('/posts/:id', auth.verify, PostsController.deletePost)
router.delete('/posts/:id/comment', auth.verify,
PostsController.deleteComment)
```

MapsRouter
/maps-api
```
router.get('/maps/:id/export', MapsController.exportMap)

router.get('/maps/map-metadata/:userId', auth.verify,
MapsController.getMapMetadataOwnedByUser)

router.get('/maps/public-map-metadata/:userId/',
MapsController.getPublicMapMetadataOwnedByUser)

router.get('/maps/:id', auth.verify, MapsController.getMapData)


router.post('/maps/upload', auth.verify, MapsController.uploadMap)
router.post('/maps/:id/fork', auth.verify, MapsController.forkMap)
router.post('/maps/:id/publish', auth.verify, MapsController.publishMap)

router.put('/maps/:id/favorite', auth.verify, MapsController.favoriteMap)
router.put('/maps/:id/rename', auth.verify, MapsController.renameMap)
router.put('/maps/:id/change-privacy', auth.verify,
MapsController.updateMapPrivacy)


router.delete('/maps/:id', auth.verify, MapsController.deleteMap)
```

PostsController
```
searchPostsByTitle(req, res) {
    method: GET
    route: /posts-api/posts/search-title/:title
    response: {
       status: 200
    // all posts info except for comments
       data: { posts: [ {
              _id: String,
             title: String,
             ownerUserName: String,
             ownerId: String,
             thumbnail: { imageData: Buffer,
contentType:String},
             likes: Number,
             forks: Number,
             tags: [String],
             mapMetadata: String
             publishDate: Date
          }]
      }
    // no search results for request
    response: {
             status: 404
             data: {errorMessage: String}
    }
}

searchPostsByTags(req, res) {
    method: GET
    route: /posts-api/posts/search-tags
    response: {
       status: 200
    // all posts info except for comments
       data: { posts: [ {
              _id: String,
             title: String,
             ownerUserName: String,
             ownerId: String,
             thumbnail: { imageData: Buffer,
contentType:String},
             likes: Number,
             forks: Number,
             tags: [String],
             mapMetadata: String
             publishDate: Date
          }]
      }
    // no search results for request
    response: {
             status: 404
             data: {errorMessage: String}
    }
}

getPostsOwnedByUser(req, res) {
    method: GET
    route: /posts-api/posts/user/:userId
    response: {
       status: 200
    // all posts info except for comments
       data: { posts: [ {
              _id: String,
             title: String,
             ownerUserName: String,
             ownerId: String,
             thumbnail: { imageData: Buffer,
contentType:String},
             likes: Number,
             forks: Number,
             tags: [String],
             mapMetadata: String
             publishDate: Date
          }]
      }
    // no search results for request
    response: {
             status: 404
             data: {errorMessage: String}
    }
}

getPost(req,res) {
    method: GET
    route: /posts-api/posts/:id
    response: {
       status: 200
       data: { post: Post }
    }
    // no search results for request
    response: {
             status: 404
             data: {errorMessage: String}
    }
}

getMostRecentPosts(req, res) {
    method: GET
    route: /posts-api/posts/most-recent
    response: {
       status: 200
    // all posts info except for comments
       data: { posts: [ {
              _id: String,
             title: String,
             ownerUserName: String,
             ownerId: String,
             thumbnail: { imageData: Buffer,
contentType:String},
             likes: Number,
             forks: Number,
             tags: [String],
             mapMetadata: String
             publishDate: Date
          }]
      }
    // no search results for request
    response: {
             status: 404
             data: {errorMessage: String}
    }
}

getMostLikedPost(req,res) {
    method: GET
    route: /posts-api/posts/most-liked
    response: {
       status: 200
    // all posts info except for comments
       data: { posts: [ {
              _id: String,
             title: String,
             ownerUserName: String,
             ownerId: String,
             thumbnail: { imageData: Buffer,
contentType:String},
             likes: Number,
             forks: Number,
             tags: [String],
             mapMetadata: String
             publishDate: Date
          }]
      }
    // no search results for request
    response: {
             status: 404
             data: {errorMessage: String}
    }
}


createPost(req, res) {
   method: POST
   route: /posts-api/posts
   response : {
      // properly formatted legal request
      status : 200,
      data : { postId: String}
   }
   response :  {
      // improperly formatted request
      status : 400,
      data : { errorMessage : String }
   }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
    }
}


deletePost(req, res) {
   method: DELETE
   route: /posts-api/posts/:id
   response: {
      // for properly formatted legal request
      status : 200,
      data : { }
   }
   response : {
       // for improperly formatted request
       status : 400,
       data : { errorMessage : String }
   }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
}

editPost(req, res) {
   method: PUT
   route: /posts-api/posts/:id
   response: {
      // for properly formatted legal request
      status : 200,
      data : {}
    }
    response : {
       // for improperly formatted request
       status : 400,
       data : { errorMessage : String }
   }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}

updatePostLikes(req, res) {
   method: PUT
   route: /posts-api/posts/:id
   response: {
      // for properly formatted legal request
      status : 200,
      data : {}
    }
   response: {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
commentOnPost(req, res) {
   method: PUT
   route: /posts-api/posts/:id
   response: {
      // for properly formatted legal request
      status : 200,
      data : {}
    }
    response : {
       // for improperly formatted request
       status : 400,
       data : { errorMessage : String }
   }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}

editComment(req, res) {
   method: PUT
   route: /posts-api/posts/:id/edit-comment
   response: {
      // for properly formatted legal request
      status : 200,
      data : {}
    }
    response : {
       // for improperly formatted request
       status : 400,
       data : { errorMessage : String }
   }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
```

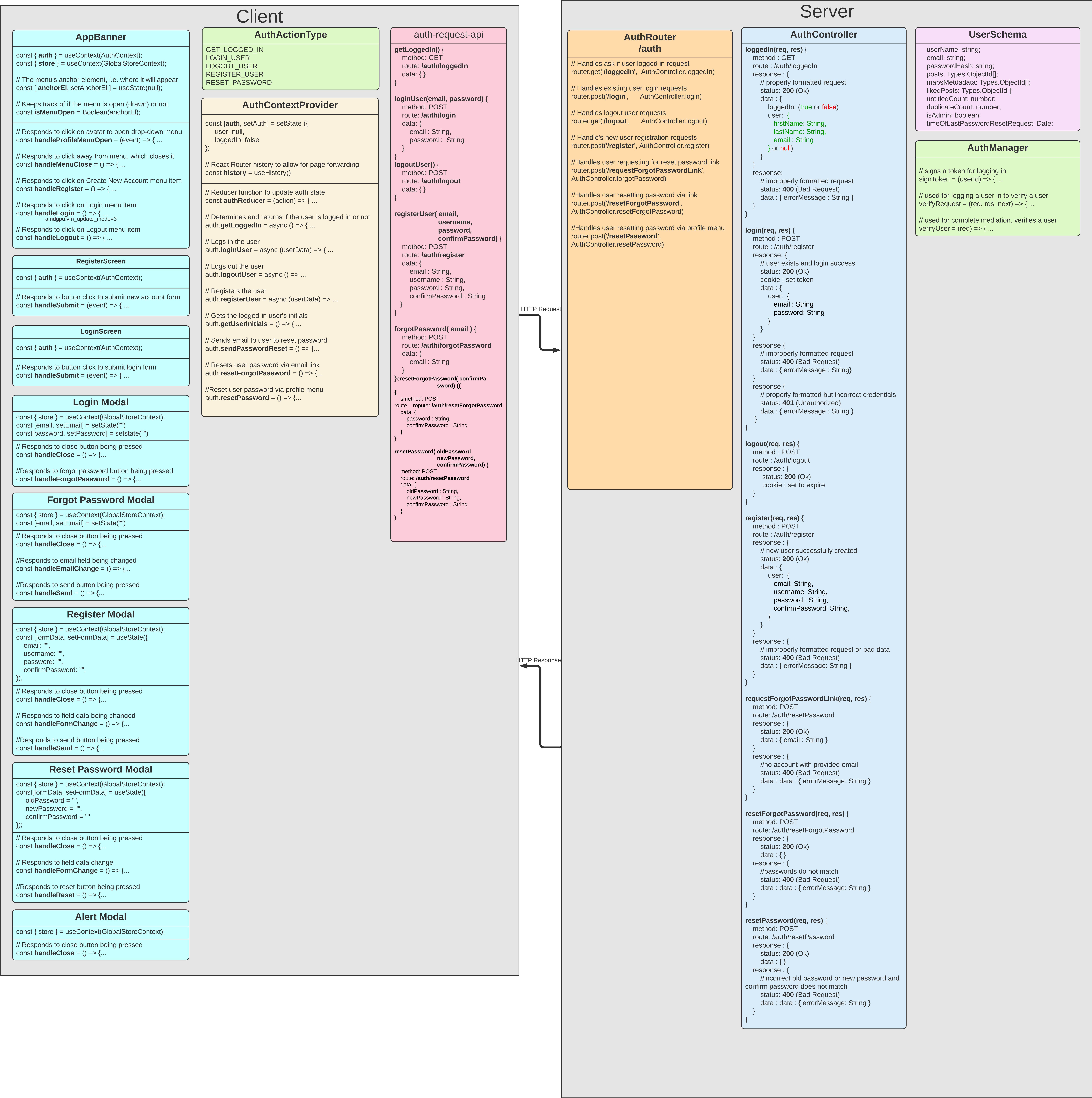
MapsController
```
exportMap(req, res) {
   method: GET
   route: /maps-api/maps/:id/export
   response: {
      // for properly formatted legal request
      status : 200,
      data : { } // there's no data because we will send the
map data to the client as a zip file.
    }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
getMapMetadataOwnedByUser(req, res) {
   method: GET
   route:  /maps-api/maps/map-metadata/:userId/
   response: {
      // for properly formatted legal request
      status : 200,
      data : { mapMetadata: [MapMetadata] }
    }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
getPublicMapMetadataOwnedByUser(req, res) {
   method: GET
   route:  /maps-api/maps/public-map-metadata/:userId/
   response: {
      // for properly formatted legal request
      status : 200,
      data : { mapMetadata: [MapMetadata] }
    }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
getMapData(req, res) {
   method: GET
   route:  /maps-api/maps/:id
   response: {
      // for properly formatted legal request
      status : 200,
      data : {mapMetadata: MapMetadata, mapData:
MapData}
    }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
uploadMap(req, res) {
   method: POST
   route:  /maps-api/maps/upload
   response: {
      // for properly formatted legal request
      status : 200,
      data : { mapId: String }
    }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
forkMap(req, res) {
   method: POST
   route: /maps-api/maps/:id/fork
   response: {
      // for properly formatted legal request
      status : 200,
      data : {}
    }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
favoriteMap(req, res) {
   method: PUT
   route:  /maps-api/maps/:id/favorite
   response: {
      // for properly formatted legal request
      status : 200,
      data : {}
    }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
publishMap(req, res) {
   method: PUT
   route:  /maps-api/maps/:id/publish
   response: {
      // for properly formatted legal request
      status : 200,
      data : {}
    }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}

updateMapPrivacy (req, res) {
       method: PUT
       route: /maps-api/maps/:id/change-privacy
       response: {
      // for properly formatted legal request
      status : 200,
      data : {}
    }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
deleteMap(req, res) {
   method: DELETE
   route: /maps-api/maps/:id/delete
   response: {
      // for properly formatted legal request
      status : 200,
      data : {}
    }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
renameMap(req, res) {
   method: PUT
   route: /maps-api/maps/:id/rename
   response: {
      // for properly formatted legal request
      status : 200,
      data : {}
    }
   response : {
      // for unauthorized request
      status : 401,
      data : { errorMessage : String }
   }
    response: {
             status: 404
             data: {errorMessage: String}
    }
}
```

Post
```
  title: string;
  owner: Types.ObjectId | UserDocument;
  ownerUserName: string;
  thumbnail: Image;
  comments: Comment[];
  images: Image[];
  likes: number;
  forks: number;
  tags: string[];
  publishDate: Date;

interface Comment {
  authorUserName: string;
  comment: string;
  publishDate: Date;
}
interface Image {
  imageData: Buffer,
  contentType: string,
}
```

User
```
   userName: string;
   email: string;
   passwordHash: string;
   posts: Types.ObjectId[];
   mapsMetdadata: Types.ObjectId[];
   likedPosts: Types.ObjectId[];
   untitledCount: number;
   duplicateCount: number;
   isAdmin: boolean;
   timeOfLastPasswordResetRequest: Date;
```

MapMetadata
```
  title: string;
  owner: Types.ObjectId;
  thumbnail: Image;
  lastEdited: Date;
  ownerFavorited: boolean;
  forks: number;
  mapData: Types.ObjectId;
  isPrivated: boolean;

interface Image {
  imageData: Buffer,
  contentType: string,
}
```

MapData
```
  geoJSON: object;
  proprietaryJSON: {
    templateType: string;
    legend: {
      title: string | undefined;
      keyValueLabels: Array<{
        key: string;
        value: string;
      }>;
    };
    gradientData: {
      primaryColor: number;
      minScale: number;
      maxScale: number;
      sections: number;
    };
  };
```



Client

AppBanner

```
const { auth } = useContext(AuthContext);
const { store } = useContext(GlobalStoreContext);

// The menu's anchor element, i.e. where it will appear
const [ anchorEl, setAnchorEl ] = useState(null);

// Keeps track of if the menu is open (drawn) or not
const isMenuOpen = Boolean(anchorEl);
```

```
// Responds to click on avatar to open drop-down menu
const handleProfileMenuOpen = (event) => { ...

// Responds to click away from menu, which closes it
const handleMenuClose = () => { ...

// Responds to click on Create New Account menu item
const handleRegister = () => { ...

// Responds to click on Login menu item
const handleLogin = () => { ...
          amdgpu.vm_update_mode=3
// Responds to click on Logout menu item
const handleLogout = () => { ...
```

RegisterScreen

```
const { auth } = useContext(AuthContext);
```

```
// Responds to button click to submit new account form
const handleSubmit = (event) => { ...
```

LoginScreen

```
const { auth } = useContext(AuthContext);
```

```
// Responds to button click to submit login form
const handleSubmit = (event) => { ...
```

Login Modal

```
const { store } = useContext(GlobalStoreContext);
const [email, setEmail] = setState("")
const[password, setPassword] = setstate("")
```

```
// Responds to close button being pressed
const handleClose = () => {...

//Responds to forgot password button being pressed
const handleForgotPassword = () => {...
```

Forgot Password Modal

```
const { store } = useContext(GlobalStoreContext);
const [email, setEmail] = setState("")
```

```
// Responds to close button being pressed
const handleClose = () => {...

//Responds to email field being changed
const handleEmailChange = () => {...

//Responds to send button being pressed
const handleSend = () => {...
```

Register Modal

```
const { store } = useContext(GlobalStoreContext);
const [formData, setFormData] = useState({
   email: "",
   username: "",
   password: "",
   confirmPassword: "",
});
```

```
// Responds to close button being pressed
const handleClose = () => {...

// Responds to field data being changed
const handleFormChange = () => {...

//Responds to send button being pressed
const handleSend = () => {...
```

Reset Password Modal

```
const { store } = useContext(GlobalStoreContext);
const[formData, setFormData] = useState({
    oldPassword = "",
    newPassword = "",
    confirmPassword = ""
});
```

```
// Responds to close button being pressed
const handleClose = () => {...

// Responds to field data change
const handleFormChange = () => {...

//Responds to reset button being pressed
const handleReset = () => {...
```

Alert Modal

```
const { store } = useContext(GlobalStoreContext);
```

```
// Responds to close button being pressed
const handleClose = () => {...
```

AuthActionType

```
GET_LOGGED_IN
LOGIN_USER
LOGOUT_USER
REGISTER_USER
RESET_PASSWORD
```

AuthContextProvider

```
const [auth, setAuth] = setState ({
    user: null,
    loggedIn: false
})

// React Router history to allow for page forwarding
const history = useHistory()
```

```
// Reducer function to update auth state
const authReducer = (action) => { ...

// Determines and returns if the user is logged in or not
auth.getLoggedIn = async () => { ...

// Logs in the user
auth.loginUser = async (userData) => { ...

// Logs out the user
auth.logoutUser = async () => ...

// Registers the user
auth.registerUser = async (userData) => ...

// Gets the logged-in user's initials
auth.getUserInitials = () => { ...

// Sends email to user to reset password
auth.sendPasswordReset = () => {...

// Resets user password via email link
auth.resetForgotPassword = () => {...

//Reset user password via profile menu
auth.resetPassword = () => {...
```

auth-request-api

```
getLoggedIn() {
    method: GET
    route: /auth/loggedIn
    data: { }
}

loginUser(email, password) {
    method: POST
    route: /auth/login
    data: {
        email : String,
        password :  String
    }
}
logoutUser() {
    method: POST
    route: /auth/logout
    data: { }
}

registerUser( email,
                          username,
                          password,
                          confirmPassword) {
    method: POST
    route: /auth/register
    data: {
        email : String,
        username : String,
        password : String,
        confirmPassword : String
    }
}

forgotPassword( email ) {
    method: POST
    route: /auth/forgotPassword
    data: {
        email : String
    }
}eresetForgotPassword( confirmPa
                          sword) {{
{
    smethod: POST
route    ropute: /auth/resetForgotPassword
    data: {
        password : String,
        confirmPassword : String
    }
}

resetPassword( oldPassword
                          newPassword,
                          confirmPassword) {
    method: POST
    route: /auth/resetPassword
    data: {
        oldPassword : String,
        newPassword : String,
        confirmPassword : String
    }
}
```

HTTP Request

HTTP Response

Server

AuthRouter /auth

```
// Handles ask if user logged in request
router.get('/loggedIn',  AuthController.loggedIn)

// Handles existing user login requests
router.post('/login',      AuthController.login)

// Handles logout user requests
router.get('/logout',      AuthController.logout)

// Handle's new user registration requests
router.post('/register', AuthController.register)

//Handles user requesting for reset password link
router.post('/requestForgotPasswordLink',
AuthController.forgotPassword)

//Handles user resetting password via link
router.post('/resetForgotPassword',
AuthController.resetForgotPassword)

//Handles user resetting password via profile menu
router.post('/resetPassword',
AuthController.resetPassword)
```

AuthController

```
loggedIn(req, res) {
    method : GET
    route : /auth/loggedIn
    response : {
        // properly formatted request
        status: 200 (Ok)
        data : {
            loggedIn: (true or false)
            user:  {
               firstName: String,
               lastName: String,
               email : String
            } or null)
        }
    }
    response:
        // improperly formatted request
        status: 400 (Bad Request)
        data : { errorMessage : String }
    }
}

login(req, res) {
    method : POST
    route : /auth/register
    response: {
        // user exists and login success
        status: 200 (Ok)
        cookie : set token
        data : {
            user:  {
               email : String
               password: String
            }
        }
    }
    response {
        // improperly formatted request
        status: 400 (Bad Request)
        data : { errorMessage : String}
    }
    response {
        // properly formatted but incorrect credentials
        status: 401 (Unauthorized)
        data : { errorMessage : String }
     }
}

logout(req, res) {
    method : POST
    route : /auth/logout
    response : {
         status: 200 (Ok)
         cookie : set to expire
    }
}

register(req, res) {
    method : POST
    route : /auth/register
    response : {
        // new user successfully created
        status: 200 (Ok)
        data : {
            user:  {
               email: String,
               username: String,
               password : String,
               confirmPassword: String,
            }
        }
    }
    response : {
        // improperly formatted request or bad data
        status: 400 (Bad Request)
        data : { errorMessage: String }
    }
}

requestForgotPasswordLink(req, res) {
    method: POST
    route: /auth/resetPassword
    response : {
        status: 200 (Ok)
        data : { email : String }
    }
    response : {
        //no account with provided email
        status: 400 (Bad Request)
        data : data : { errorMessage: String }
    }
}

resetForgotPassword(req, res) {
    method: POST
    route: /auth/resetForgotPassword
    response : {
        status: 200 (Ok)
        data : { }
    }
    response : {
        //passwords do not match
        status: 400 (Bad Request)
        data : data : { errorMessage: String }
    }
}

resetPassword(req, res) {
    method: POST
    route: /auth/resetPassword
    response : {
        status: 200 (Ok)
        data : { }
    response : {
        //incorrect old password or new password and
confirm password does not match
        status: 400 (Bad Request)
        data : data : { errorMessage: String }
    }
}
```

UserSchema

```
userName: string;
email: string;
passwordHash: string;
posts: Types.ObjectId[];
mapsMetdadata: Types.ObjectId[];
likedPosts: Types.ObjectId[];
untitledCount: number;
duplicateCount: number;
isAdmin: boolean;
timeOfLastPasswordResetRequest: Date;
```

AuthManager

```
// signs a token for logging in
signToken = (userId) => { ...

// used for logging a user in to verify a user
verifyRequest = (req, res, next) => { ...

// used for complete mediation, verifies a user
verifyUser = (req) => { ...
```

Meeting Minutes
Date: 09/26/2023

Screens:
- Home Screen (View all discussions threads and public maps)
- Maps Screen (View all public maps only)
- Post Screen (View current post opened)
- Profile Screen (View account details and settings)
- My Maps Screen (View user owned maps)
- Edit Map Screen (Edit current map opened)

Actors:
- Guest
- Logged-In User
- Admin

When the user forgets a password, they should be prompted to enter their username and email. The system will validate if these match any existing data. If not, an error message will be shown to the user. Otherwise, a password reset email will be sent to the user

Avatar icon on top right will open a dropdown menu with the options “My Maps”, “Settings”, and “Logout.” The user will have the option to change their profile icon to an image from their device.

Maps on My Maps Screen are sorted by recently viewed by default. We can also add the option to sort alphabetically or by recently edited.

Publishing a map creates a new post with the current state of the map (JSON data). This way, anyone who forks the map basically does an “automatic import” of that JSON data into their own MyMaps.

A user can publish their map multiple times (possibly after making edits in between each publish). This would create a new post each time they publish.

Options for search:
Post title,
Tag(s),
Users

We decided on having the search bar have 2 options. To either search by post title or tag(s)

11/1 - Build 1 Meeting Notes:

- Version Control: 2 Repositories:
  - cartistry (frontend)
  - cartistry-express (backend)
- Deployed Platforms:
  - AWS Amplify (frontend)
    - Try to set up custom domain?
  - Vercel (backend)
    - Serverless and free!!!
- CI/CD:
  - Github Actions: live logs and multi-container testing
  - Create ci.yml
- Cypress IO
  - Documentation: https://docs.cypress.io/guides/overview/why-cypress
  - Refer to E2E testing
- Jest and Supertest
  - Jest Documentation: https://jestjs.io/docs/configuration
- Monday:
  - Basic Schedule:
    - Build 2: All frontend screens and routes
    - Build 3: All basic auth (login, register, logout, etc.), My Maps screen (CRUD operations) and loading posts (homescreen & view post screen)
    - Build 4: Posts (CRUD operations), sorting maps & posts, get most recent post
    - Build 5: Comments (CRUD operations), Map transactions, & Map editing templates
    - Build 6: Finishing touches
  - Make sure to add frontend and backend testing for each build and new feature
- TODO List:
  - Set up deployed platforms (Jonathan)
  - Set up Version control and CI/CD (Angelo)
  - Create frontend and backend tests (Aaron)
  - Set up Monday tasks (Jenny)

11/15 - Build 3 Meeting Notes:

- TODO
    - Figure out how to setup backend and database for E2E testing for CI/CD
    - Fix reset password through forgot password
    - Fix bugs from progress check
        - Input fields should be the correct type
        - Add loading states
        - Guests should not be able to comment
        - Foolproof design for modal buttons
        - Renaming and edit post fields update
    - Sizing on map containers for post and edit map screen
    - Fix exit current post action
    - Add error and success messages for user feedback
    - Initial setup for sorting posts functionality
    - Make sure publish map works
    - Handle redirection on post card clicks
    - Handle redirection on map upload
    - Work on map and post contexts
    - Dropdowns should close when clicking away
    - Finish import modal

11/22 - Build 4 Meeting Notes

- TODO
    - Fix view post bugs
    - Fix edit post bugs
    - Foolproof create post button for guests
    - Update post card menu styling
    - Routing with mapid and postid params should work
    - Implement searching in home and my maps screens
    - Likes update on post screen
    - Comment input clears after submission
    - Maps only sort should work on home screen
    - Fix upload map bugs
    - Add isSubnitting to post, login, and register
    - Fix my posts screen view
    - Add thumbnails to postcard view
    - Add image uploads to edit post screen

11/29 - Build 5 Meeting Notes

- TODO
  - Add error handling to popup form
  - Finish profile screen
  - Make usernames clickable
  - Update time since posted
  - Undo and redo features
  - Add frontend tests for posts
  - Edit and delete comments
  - Add backend to ci/cd
  - Show comments from new commentList field
  - Fix loadmaps on my posts screen
  - Fix auth routing

12/6 - Build 6 Meeting Notes

- TODO
  - Add legend
  - Add sorting by tags
  - Post and comment timestamps fix
  - Fix image not showing
  - Fix saving for all map types
  - Fix undo/redo for all map types
  - Add color picker
  - Add Geoman controls
  - Foolproof Geoman tools
  - Folder restructure
  - Dynamic toolbox
  - Update handling icons for re renders