System Analysis

By Razzmatazz (Preston Chang, James Leonardi, Wei Qi, Kevin Tao)

Executive Summary

MapStudio aims to combine both map creation and community interaction services into one, intuitive piece of software that gives cartography lovers a one-stop-shop for all of their needs. Currently, the map-making world is split between map creating and community meeting places with no service to bridge the gap between them. When individuals want to create custom map-based graphics, they will use general-use graphics editing software like Adobe Photoshop or GIMP, geographic information system (GIS) software like ArcGIS or QGIS, or some combination of the two. However, if they then want to share their custom maps and discuss with other like-minded individuals, they would have to use other services like the subreddit r/MapPorn or other cartography-related forums. MapStudio would allow a more seamless map creation and community experience by not simply offering the services of both, but by integrating these services in a complementary and cohesive manner, fostering a greater sense of camaraderie and collaboration among members of the map-making community.

Objectives

- Allow users to easily create custom map graphics for sharing data geospatially. This includes uploading of map data, editing map data and properties, and exporting graphics.
- Provide a collaborative platform for users to find, edit, discuss, and reuse map data and graphics created by others.
- Offer accounts management with security and access control so users can keep maps private or share them publicly.
- Enable searching maps based on properties so users can discover and reuse existing maps.
- Support common geospatial data formats like Shapefile, GeoJSON, and KML to allow users to bring in map data from various sources.
- Allow custom visual styling and decorating of maps with text, colors, legends etc. to create informative graphics.
- Provide graphics export in standard image formats like PNG/JPG for easy external use.
- Offer convenient editing features, like undo/redo.
- Make the map creation and editing process as seamless and user-friendly as possible.
- Build an engaged user community around collaborative geospatial data mapping.

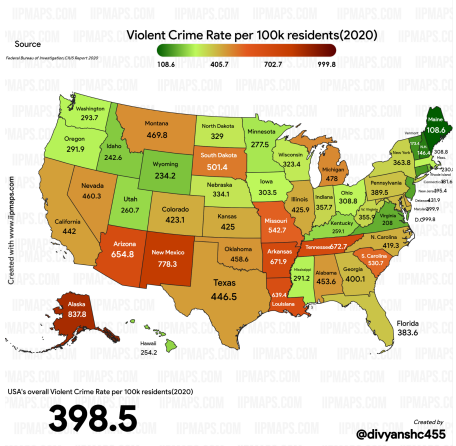
Strategies/Philosophies

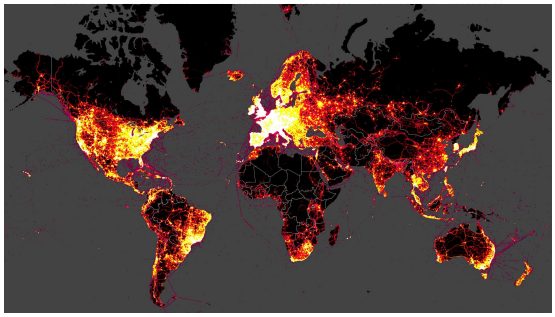
- Intuitive/easy-to-use UI/UX - The user interface should be easy to navigate and common actions should be quick, responsive, and require the least mouse travel and clicks possible. The application's design should be consistent and united.
- Simplicity - The design of the application should be as simple as possible to reduce user confusion and choice overload. Icons and buttons should be labeled whenever possible so that users can easily parse and understand what they are doing at all times.
- Performance Optimization - The web application should maintain a high level of performance, especially when users are editing map graphics.

Constraints

The main constraint of MapStudio would be the different types of maps that it will need to support. In order to keep the application focused and set clear boundaries on the scope of the project, we decided to focus on and support the following five map graphic types:

- Bin Maps - Each subdivision is categorized into one of multiple colored “bins”. In the example given, countries are colored based on if citizens of that country need a visa to enter Germany.
- Gradient Maps - Each subdivision is colored based on a sliding scale and its associated color gradient. In the example given, states are colored based on the violent crime rate with redder states having higher rates and green states having lower states.
- Heat Maps - Areas of intensity are highlighted without being restricted to geographical boundaries. In the given example, different regions on the maps glow more intensely based on the usage of the exercise app “Strava” in that region.
- Point Maps - Select locations are highlighted as a point. In the given example, the state capitals are indicated with points and labeled.
- Satellite Maps - Real-world imagery is displayed to provide a detailed view of geographical features and landscapes. In the example given, the geographical features of the United States are on display.





Actors

There are two groups of actors: guests and members. Guests are defined as users of the web application that are not logged in at time of use. Members are defined as users of the web application who have previously registered an account and are logged in at time of use.

Guests will have fewer privileges and more restrictions compared to members, who will have access to all of the features that the web application provides. Guests will have viewing privileges, but not creation privileges. More specifically, guests will only be able to view the maps that members have created and shared. They will not be able to do anything that will write data to the back-end of the web application. Guests will have the option to either register a new account or log in to an existing account.

Members will have full access to the application. On top of the ability to view maps, members will also be able to create and save their own maps, as well as publish them for others to see. In addition, members will have the privilege of using community features as well. Members will be able to fork the maps of others in order to edit them for their own purposes, vote and comment on the maps of others, as well as participate in discussions of all kinds of cartography-related topics. Members will be able to log out of their account.

Services

- Accounts Management - We will provide a robust and secure account system in which users will be able to make accounts in order to fully utilize all features and privileges offered by the web application. In addition to security, we will also focus on convenience with the ability to recover forgotten passwords.
- Map Graphic Creation - We will provide members the ability to create new map graphics from either uploading SHP/DBF, GeoJSON, or KML files or forking the published maps of other members. This allows members the ability to obtain base maps from third-party sources or directly from provided templates or community creations on MapStudio.
- Map Graphic Editing - We will provide members with the ability to edit map graphics in order to make them more informative and visually appealing. Editing map graphics will involve naming regions, attaching custom data properties to regions, and attaching custom properties to maps.
- Map Graphics Exporting - We will provide members with the ability to export their created map graphics for ease of sharing or use in external applications in the form of rasterized image formats like PNG and JPEG. In addition, members will be able to export their map as a MapStudio custom JSON file so that they can save their map offline.
- Map Classification and Search - We will provide members with the ability to categorize their maps based on the different properties of their maps. These properties would include scope (global vs country specific), type (bin maps vs gradient maps), or subject (industry vs politics). Thus, users of the web application will also be able to search for published maps based on their name and category and sort results by date, views, likes/dislikes, and more.
- Community Interactions - We will provide members with tools to facilitate community interactions and collaboration. Members will be able to submit feedback on maps via likes/dislikes as well as comments. In addition, members will be able to create individual boards, not tied to any map, to discuss any cartography-related topics of their choosing.

Use Case Model

By Razzmatazz (Preston Chang, James Leonardi, Wei Qi, Kevin Tao)

Use Case Descriptions

Use Case #	*UI Context*	*Use Case Name*
1.1	Menu Bar	Log In
1.2	Menu Bar	Register Account
1.3	Menu Bar	Forgot Account?
1.4	Menu Bar	Log Out
1.5	Menu Bar	Browse Maps
1.6	Menu Bar	Create Maps
1.7	Menu Bar	Discuss!
2.1	Menu Bar	Search for Map
2.2	Search Results Screen	Sort Results
2.3	Search Results Screen	Filter Results
2.4	Search Results Screen	Choose Map
3.1	Map View Screen	Like Map
3.2	Map View Screen	Dislike Map
3.3	Map View Screen	Comment on Map
3.4	Map View Screen	Pan Map
3.5	Map View Screen	Zoom in on Map
3.6	Map View Screen	Zoom out on Map
3.7	Map View Screen	Download as JSON
3.8	Map View Screen	Download as PNG
3.9	Map View Screen	Download as JPG
3.10	Map View Screen	Fork Map
4.1	Personal Maps Screen	Search for Personal Map

4.2	Personal Maps Screen	Sort Personal Results
4.3	Personal Maps Screen	Filter Personal Results
4.4	Personal Maps Screen	Choose Personal Map
4.5	Personal Maps Screen	New Map
5.1	Map Edit Screen	Upload Map
5.2	Map Edit Screen	Delete Map
5.3	Map Edit Screen	Publish Map
5.4	Map Edit Screen	Pan Map (Edit)
5.5	Map Edit Screen	Zoom in on Map (Edit)
5.6	Map Edit Screen	Zoom out on Map (Edit)
5.7	Map Edit Screen	Select Subdivision
5.8	Edit Toolbar	Choose Template
6.1	Map Edit Screen	Undo Edit
6.2	Map Edit Screen	Redo Edit
6.3	Edit Toolbar	Edit Map Info
6.3.1	Map Info Sidebar	Rename Map
6.3.2	Map Info Sidebar	Edit Map Description
6.4	Edit Toolbar	Edit Subdivision Info
6.4.1	Subdivision Info Sidebar	Change Subdivision Color
6.4.2	Subdivision Info Sidebar	Edit Subdivision Weight
6.4.3	Subdivision Info Sidebar	Show Satellite View
6.5	Map Edit Screen	Add Point
6.5.1	Map Edit Screen	Edit Point Info
6.5.2	Point Info Sidebar	Edit Point Name
6.5.3a	Point Info Sidebar	Edit Point Weight (Heatmap)

6.5.3b	Point Info Sidebar	Edit Point Weight (Pointmap)
6.5.4	Point Info Sidebar	Delete Point
6.5.5	Point Info Sidebar	Edit Point Location
6.6	Edit Toolbar	Edit Bin Info
6.6.1	Bin Info Sidebar	Create Bin
6.6.2	Bin Info Sidebar	Delete Bin
6.6.3	Bin Info Sidebar	Add Subdivisions to Bin
6.6.4	Bin Info Sidebar	Remove Subdivisions from Bin
6.6.5	Bin Info Sidebar	Change Bin Color
6.6.6	Bin Info Sidebar	Change Bin Label
6.7	Edit Toolbar	Edit Gradient Info
6.7.1	Gradient Info Sidebar	Generate Gradient
6.7.2	Gradient Info Sidebar	Change Gradient Colors
6.7.3	Gradient Info Sidebar	Delete Gradient
7.1	Discussion Board Home	Create New Discussion
7.2	Discussion Board Home	View Discussion
7.3	Discussion Board Home	Search for Discussion
8.1	Discussion Board Screen	Like Post
8.2	Discussion Board Screen	Dislike Post
8.3	Discussion Board Screen	Comment on Post
9.1	Member Name	View Profile
9.2	Member Profile Screen	Choose Map (Profile)
9.3	Member Profile Screen	View Discussion (Profile)

Use Case Section 1 - Menu Bar

Number	1.1
Name	Log In
Actor	Guest
Story	1. Observe the menu bar 2. Click on the account icon and observe the dropdown menu 3. Click on the “Log In” button 4. Observe the login screen 5. Enter a username and password 6. Click the “Log In” button
Scenario	Bob observes the menu bar and clicks on the account icon. He observes the dropdown menu and clicks on the “Log In” button. Bob observes the login screen and enters “Bob123” as the username and “password123” as the password. He clicks the “Log In” button and successfully logs in.
Exceptions	The guest enters incorrect account details. The guest will not be able to log in and will be prompted to try again or use the “Forgot Account?” service.

Number	1.2
Name	Register Account
Actor	Guest
Story	1. Observe the menu bar 2. Click on the account icon and observe the dropdown menu 3. Click on the “Register Account” button 4. Observe the register account screen 5. Enter a username, email, and password 6. Click the “Register Account” button
Scenario	Bob observes the menu bar and clicks on the account icon. He observes the dropdown menu and clicks on the “Register Account” button. Bob observes the register account screen and enters “Bob123” as the username, “bob@gmail.com” as the email, and “password123” as the password. He clicks the “Register Account” button, successfully creates an account, and is automatically logged in.
Exceptions	The chosen username is in use by another account. Account creation will fail and the guest will be prompted to choose another username.

Number	1.3
Name	Forgot Account?
Actor	Guest
Story	1. Observe the menu bar 2. Click on the account icon and observe the dropdown menu 3. Click on the “Forgot Account?” button 4. Observe the forgotten account screen 5. Enter username and email associated with the forgotten account 6. Click the “Recover Info” button
Scenario	Bob observes the menu bar and clicks on the account icon. He observes the dropdown menu and clicks on the “Forgot Account?” button. Bob observes the forgotten account screen and enters “Bob123” as the username and “bob@gmail.com” as the email. He clicks the “Recover Info” button. He then checks his email and recovers his password.
Exceptions	The username and email combination does not exist. The account recovery details will not be emailed to the email address. The guest will not be notified.

Number	1.4
Name	Log Out
Actor	Member
Story	1. Observe the menu bar 2. Click on the account icon and observe the dropdown menu 3. Click on the “Log Out” button
Scenario	Bob observes the menu bar and clicks on the account icon. He observes the dropdown menu and clicks on the “Log Out” button. Bob is logged out of his account and is returned to the home screen as a guest.
Exceptions	

Number	1.5
Name	Browse Maps
Actor	Guest/Members
Story	1. Observe the menu bar 2. Click on the “Browse Maps” button 3. Observe the search results screen
Scenario	Bob observes the menu bar and clicks on the “Browse Maps” button. He then observes the search results screen.
Exceptions	

Number	1.6
Name	Create Map
Actor	Guest/Members
Story	1. Observe the menu bar 2. Click on the “Create Map” button 3. Observe the map edit screen
Scenario	Bob observes the menu bar and clicks on the “Create Map” button. He then observes the map edit screen.
Exceptions	If the actor is a guest, they should be redirected to the sign-in page instead

Number	1.7
Name	Discuss!
Actor	Guest/Members
Story	1. Observe the menu bar 2. Click on the “Discuss!” button 3. Observe the discussion board home screen
Scenario	Bob observes the menu bar and clicks on the “Discuss!” button. He then observes the discussion board home screen
Exceptions	If the actor is a guest, they should be redirected to the sign-in page instead

Use Case Section 2 - Map Search

Number	2.1
Name	Search for Map
Actor	Guest/Member
Story	1. Observe the menu bar 2. Type a search query in the search text box 3. Press the “Enter” button on the keyboard or click on the search icon 4. Observe the search results screen
Scenario	Bob observes the menu bar and types “Japan” into the search bar. He then clicks on the search icon and is brought to the search results screen which shows the relevant results.
Exceptions	

Number	2.2
Name	Sort Results
Actor	Guest/Member
Story	1. Observe the search results screen 2. Click the dropdown menu that contains the different sorting methods 3. Click on one of the sorting methods
Scenario	Bob observes the search results screen and clicks on the sort dropdown menu. He then clicks on the “Sort by Likes (Highest First)” option. The results are then sorted from most to least likes.
Exceptions	

Number	2.3
Name	Filter Results
Actor	Guest/Member
Story	1. Observe the search results screen 2. Click the dropdown menu that contains the different filters 3. Click on one of the filters
Scenario	Bob observes the search results screen and clicks on the filter dropdown menu. He then clicks on the “Heat Map” option. The results are updated to show only heat maps.
Exceptions	

Number	2.4
Name	Choose Map
Actor	Guest/Member
Story	1. Observe the search results screen 2. Click on one of the cards that contains a map 3. Observe the map view screen
Scenario	Bob observes the search results screen and sees a card containing a map that he thinks has an interesting title and preview image. He then clicks anywhere on the card. He is then sent to the map view screen.
Exceptions	

Use Case Section 3 - Community Actions on Maps

Number	3.1
Name	Like Map
Actor	Member
Story	1. Observe the map view screen 2. Click on the like button
Scenario	Bob observes the map view screen for a particular map that he likes. He clicks the like button.
Exceptions	If the map is already liked, remove the previous like. If the map is already disliked, remove the previous dislike and add a like.

Number	3.2
Name	Dislike Map
Actor	Member
Story	1. Observe the map view screen 2. Click on the dislike button
Scenario	Bob observes the map view screen for a particular map that he dislikes. He clicks the dislike button.
Exceptions	If the map is already disliked, remove the previous dislike. If the map is already liked, remove the previous like and add a dislike.

Number	3.3
Name	Comment on Map
Actor	Member
Story	1. Observe the map view screen 2. Type a comment within the comment text box 3. Click the submit button
Scenario	Bob observes the map view screen for a particular map. He types in the comment text box: “I love this map”. He clicks the submit button.
Exceptions	

Number	3.4
Name	Pan Map
Actor	Guest/Member
Story	1. Observe the map view screen 2. Click on the map and drag in the direction of the intended pan
Scenario	Bob observes the view map screen for a particular map. He wants to look to the east. He clicks on the map and drags to the left.
Exceptions	

Number	3.5
Name	Zoom in on Map
Actor	Guest/Member
Story	1. Observe the map view screen 2. Scroll the mouse wheel up or click on the “+” button on the map screen
Scenario	Bob observes the map view screen for a particular map. He wants to view the area in more detail. He scrolls the mouse wheel up.
Exceptions	

Number	3.6
Name	Zoom out on Map
Actor	Guest/Member
Story	1. Observe the map view screen 2. Scroll the mouse wheel down or click on the “-” button on the map screen
Scenario	Bob observes the map view screen for a particular map. He wants to view the area more broadly. He scrolls the mouse wheel down.
Exceptions	

Number	3.7
Name	Download as JSON
Actor	Guest/Member
Story	1. Observe the map view screen 2. Click the “Download as JSON” button
Scenario	Bob observes the view map screen for a particular map. He wants to download the map data for offline storage. He clicks the “Download as JSON” button.
Exceptions	

Number	3.8
Name	Download as PNG
Actor	Guest/Member
Story	1. Observe the map view screen 2. Click the “Download as PNG” button
Scenario	Bob observes the view map screen for a particular map. He wants to download the map as an image for sharing. He clicks the “Download as PNG” button.
Exceptions	

Number	3.9
Name	Download as JPG
Actor	Guest/Member
Story	1. Observe the map view screen 2. Click the “Download as JPG” button
Scenario	Bob observes the view map screen for a particular map. He wants to download the map as an image for sharing. He clicks the “Download as JPG” button.
Exceptions	

Number	3.10
Name	Fork Map
Actor	Guest/Member
Story	1. Observe the map view screen 2. Click the “Fork Map” button 3. Observe the map edit screen
Scenario	Bob observes the view map screen for a particular map. He wants to use the map shape for his own graphic. He clicks the “Fork Map” button. He is then redirected to the map edit screen for editing.
Exceptions	If the actor is a guest, they should be redirected to the sign-in page instead

Use Case Section 4 - Personal Maps

Number	4.1
Name	Search for Personal Map
Actor	Member
Story	1. Observe personal maps screen 2. Type a search query in the search text box 3. Press the “Enter” button on the keyboard or click on the search icon 4. Observe the search results screen
Scenario	Bob observes the personal maps screen and types “Japan” into the search bar. He then clicks on the search icon and then only his personal map graphics that include “Japan” in the title appear.
Exceptions	

Number	4.2
Name	Sort Personal Results
Actor	Member
Story	1. Observe personal maps screen 2. Click the dropdown menu that contains the different sorting methods 3. Click on one of the sorting methods
Scenario	Bob observes the personal maps screen and clicks on the sort dropdown menu. He then clicks on the “Sort by Likes (Highest First)” option. His maps are then sorted from most to least likes.
Exceptions	

Number	4.3
Name	Filter Personal Results
Actor	Member
Story	1. Observe the personal maps screen 2. Click the dropdown menu that contains the different filters 3. Click on one of the filters
Scenario	Bob observes the personal maps screen and clicks on the filter dropdown menu. He then clicks on the “Heat Map” option. His map collection is updated to show only heat maps.
Exceptions	

Number	4.4
Name	Choose Map to Edit
Actor	Member
Story	1. Observe the personal map screen 2. Click on one of the cards that contains a map 3. Observe the map edit screen
Scenario	Bob observes the personal maps screen and sees a card containing a map that he wants to edit. He then clicks anywhere on the card. He is then sent to the map edit screen.
Exceptions	Published maps cannot be edited, clicking on the card will bring the member to the map view screen.

Number	4.5
Name	New Map
Actor	Member
Story	1. Observe the personal map screen 2. Click on the “Create New Map” button 3. Observe the map edit screen
Scenario	Bob observes the personal maps screen and wants to create a new map. He then clicks on the “Create New Map” button. He is then sent to the map edit screen.
Exceptions	

Use Case Section 5 - Map Creation

Number	5.1
Name	Upload Map
Actor	Member
Story	1. Observe the map edit screen 2. Click on the “Upload Map” button 3. Upload a Shapefile, GeoJSON, or KML file 4. Observe the map on the map edit screen
Scenario	Bob observes the map edit screen and decides to upload a map he procured from a third-party source. He clicks the “Upload Map” button and uploads a Shapefile containing a map of the United States. He observes the newly uploaded map outline of the United States on the map edit screen.
Exceptions	

Number	5.2
Name	Delete Map
Actor	Member
Story	1. Observe the map edit screen 2. Click on the “Delete Map” button 3. Observe the confirmation modal 4. Click on “Confirm” to delete the map or “Cancel” to cancel the operation 5. Observe the personal maps screen
Scenario	Bob observes the map edit screen and decides to delete the map. He then clicks on the “Delete Map” button and the “Confirm” button on the resulting modal. Bob observes the personal maps screen.
Exceptions	

Number	5.3
Name	Publish Map
Actor	Member
Story	1. Observe the map edit screen 2. Click on the “Publish Map” button 3. Observe the confirmation modal 4. Click on “Confirm” to publish the map or “Cancel” to cancel the operation 5. Observe the map view screen
Scenario	Bob observes the personal maps screen and wants to publish the map. He then clicks on the “Publish Map” button and the “Confirm” button on the resulting modal. Bob observes the map view screen.
Exceptions	

Number	5.4
Name	Pan Map (Edit)
Actor	Member
Story	1. Observe the map edit screen 2. Click on the map and drag in the direction of the intended pan
Scenario	Bob observes the map edit screen for a particular map. He wants to look to the east. He clicks on the map and drags to the left.
Exceptions	

Number	5.5
Name	Zoom in on Map (Edit)
Actor	Member
Story	1. Observe the map edit screen 2. Scroll the mouse wheel up or click on the “+” button on the map screen
Scenario	Bob observes the map edit screen for a particular map. He wants to view the area in more detail. He scrolls the mouse wheel up.
Exceptions	

Number	5.6
Name	Zoom out on Map (Edit)
Actor	Member
Story	1. Observe the map edit screen 2. Scroll the mouse wheel down or click on the “-” button on the map
Scenario	Bob observes the map edit screen for a particular map. He wants to view the area more broadly. He scrolls the mouse wheel down.
Exceptions	

Number	5.7
Name	Select Subdivision
Actor	Member
Story	1. Observe the map edit screen 2. Click on a subdivision outlined on the map or click on the subdivision on the subdivision list 3. Observe the highlighted subdivision and its associated info
Scenario	Bob observes the map edit screen for the world. He wants to select a specific country to edit. He clicks on the outline of the United States.
Exceptions	If the subdivision is already selected, deselect the subdivision.

Number	5.8
Name	Choose Template
Actor	Member
Story	1. Observe the map edit screen 2. Click on the “Template” tab on the edit toolbar 3. Observe the five available map templates 4. Choose a map template
Scenario	Bob observes the map edit screen. He clicks on the “Template” tab on the edit toolbar. He observes the five available map templates and clicks on the “Heat Map” option.
Exceptions	

Use Case Section 6 - Map Editing

Number	6.1
Name	Undo Edit
Actor	Member
Story	1. Observe the map edit screen 2. Click the “Undo” button or press CTRL+Z
Scenario	Bob observes the map edit screen. He wants to undo his last edit. He presses the “Undo” button.
Exceptions	If there are no previous edits, the button should be greyed out.

Number	6.2
Name	Redo Edit
Actor	Member
Story	1. Observe the map edit screen 2. Click the “Redo” button or press CTRL+Y or CTRL+SHIFT+Z
Scenario	Bob observes the map edit screen. He wants to redo his last edit. He presses the “Redo” button.
Exceptions	If there are no future edits, the button should be greyed out.

Number	6.3
Name	Edit Map Info
Actor	Member
Story	1. Observe the map edit screen 2. Click on the “Map Info” tab on the edit toolbar 3. Observe the map info edit sidebar
Scenario	Bob observes the map edit screen. He clicks on the “Map Info” tab on the edit toolbar. He observes the map info edit sidebar.
Exceptions	If another sidebar is open, close that sidebar and replace it with this one.

Number	6.3.1
Name	Rename Map
Actor	Member
Story	1. Observe the map info edit sidebar 2. Click on the title of the map 3. Observe the editable text box 4. Type a new name into the text box 5. Press “Enter” on the keyboard or click anywhere outside the text box
Scenario	Bob observes the map info edit sidebar. The map is named “PLACEHOLDER NAME” and he wants to change it. He clicks on the title of the map and a text box appears with “PLACEHOLDER NAME” as a field. He then changes it to “Map of Brazil” and presses “Enter” on his keyboard.
Exceptions	Blank names are not allowed. If entered, the name will not change.

Number	6.3.2
Name	Edit Map Description
Actor	Member
Story	1. Observe the map info edit sidebar 2. Click on the text box labeled “Description” 3. Click outside of the text box to automatically save the description
Scenario	Bob observes the map info edit sidebar. He types into the “Description” text box, “This is a map of the GDP of the different states of Brazil”. He clicks outside of the text box to save the change.
Exceptions	

Number	6.4
Name	Edit Subdivision Info
Actor	Member
Story	1. Observe the map edit screen 2. Select a subdivision as outlined in 5.8 3. Click on the “Data” tab on the edit toolbar 4. Observe the data edit sidebar
Scenario	Bob observes the map edit screen. He clicks on a subdivision he wants to edit the information of. He clicks on the “Data” tab on the edit toolbar. He observes the data edit sidebar.
Exceptions	If another sidebar is open, close that sidebar and replace it with this one.

Number	6.4.1
Name	Change Subdivision Color
Actor	Member
Story	1. Observe the data edit sidebar 2. Click on the color picker icon 3. Observe the color picker 4. Choose the desired color via Hex, RGB, or manually choosing on the palette
Scenario	Bob observes the map edit screen with the subdivision info sidebar. He clicks on the color picker icon. A color picker widget shows up. He chooses a new color via the color picker.
Exceptions	

Number	6.4.2
Name	Edit Subdivision Weight
Actor	Member
Story	1. Observe the data edit sidebar 2. Adjust the weight of the subdivision by either clicking + or -, or by manually entering a weight
Scenario	Bob observes the map edit screen with the subdivision info sidebar. He clicks on the weight field to manually enter a new weight. He adjusts it further using the + or - buttons.
Exceptions	Weight must be within 0.0 to 1.0

Number	6.4.3
Name	Show Satellite View
Actor	Member
Story	1. Observe the data edit sidebar 2. Toggle the satellite view
Scenario	Bob observes the map edit screen with the subdivision info sidebar. He clicks on the “Show satellite view” button to toggle the satellite view underlay.
Exceptions	

Number	6.5
Name	Add Point
Actor	Member
Story	1. Observe the map edit screen 2. Click on the “Add Point” tool 3. Click on places on the map to add points 4. Click on the “Add Point” tool again to stop
Scenario	Bob observes the map edit screen. He clicks on the “Add Point” tool button. He then clicks several points on the map. He finishes by clicking the “Add Point” tool button again.
Exceptions	

Number	6.5.1
Name	Edit Point Info
Actor	Member
Story	1. Observe the map edit screen 2. Click on a point 3. Observe the point info sidebar
Scenario	Bob observes the map edit screen. He clicks on a point to show the point info sidebar where he can edit this point’s information.
Exceptions	If another sidebar is open, close that sidebar and replace it with this one.

Number	6.5.2
Name	Edit Point Name
Actor	Member
Story	1. Observe the point info sidebar 2. Type text into the name text box
Scenario	Bob observes the map edit screen with the point info sidebar. He types “Point 1” into the name text box.
Exceptions	Blank names are not allowed. If entered, the name will not change.

Number	6.5.3a
Name	Edit Point Weight (Heatmap)
Actor	Member
Story	1. Observe the point info sidebar 2. Adjust the weight of the point by either clicking + or -, or by manually entering a weight
Scenario	Bob observes the map edit screen with the point info sidebar. He clicks on the weight field to manually enter a new weight. He adjusts it further using the + or - buttons. The heatmap updates to reflect this new weight.
Exceptions	Weight must be within 0.0 to 1.0

Number	6.5.3b
Name	Edit Point Weight (Pointmap)
Actor	Member
Story	1. Observe the point info sidebar 2. Adjust the weight of the point by either clicking + or -, or by manually entering a weight
Scenario	Bob observes the map edit screen with the point info sidebar. He clicks on the weight field to manually enter a new weight. He adjusts it further using the + or - buttons. The point on the map changes size to reflect this new weight.
Exceptions	Weight must be within 0.0 to 1.0

Number	6.5.4
Name	Delete Point
Actor	Member
Story	1. Observe the point info sidebar 2. Click “Delete Point”
Scenario	Bob observes the map edit screen with the point info sidebar. He clicks the “Delete Point” button to remove the point from the data set. The map updates to reflect this change
Exceptions	

Number	6.5.5
Name	Edit Point Location
Actor	Member
Story	1. Observe the map edit screen 2. Click “Move Point” 3. Drag the point to its new location or enter its position manually 4. Click “Move Point” again to stop
Scenario	Bob observes the map edit screen with the point info sidebar. He clicks the “Edit Point” button. He drags the point to a new location. He then fine-tunes the location by editing its position manually.
Exceptions	

Number	6.6
Name	Edit Bin Info
Actor	Member
Story	1. Observe the point info sidebar 2. Click on the “Bin Info” tab on the edit toolbar 3. Observe the bin info sidebar
Scenario	Bob observes the map edit screen. He wants to edit the info of the bins he created. He clicks on the “Bin Info” tab on the edit toolbar. He observes the bin info sidebar.
Exceptions	If another sidebar is open, close that sidebar and replace it with this one.

Number	6.6.1
Name	Create Bin
Actor	Member
Story	1. Observe the bin info sidebar 2. Click on the “Create Bin” button
Scenario	Bob observes the bin info sidebar. He wants to create a new bin to categorize subdivisions. He clicks on the “Create Bin” button. A new bin is created.
Exceptions	

Number	6.6.2
Name	Delete Bin
Actor	Member
Story	1. Observe the bin info sidebar 2. Click on the “X” next to the name of bin to be deleted
Scenario	Bob observes the bin info sidebar. He wants to delete the bin “Anglophone”. He clicks on the “X” next to the bin name “Anglophone”.
Exceptions	

Number	6.6.3
Name	Add Subdivisions to Bin
Actor	Member
Story	1. Observe the bin info sidebar 2. Click on “Add” underneath the name of the bin 3. Click on the subdivisions on the map to add to the bin 4. Click on “Add” underneath the name of the bin to finish
Scenario	Bob observes the bin info sidebar. He wants to add subdivisions to the bin “Anglophone”. He clicks on “Add” under the bin name “Anglophone”. He then clicks on the United States on the map. He clicks on “Add” under the bin name “Anglophone” to finish.
Exceptions	

Number	6.6.4
Name	Remove Subdivisions from Bin
Actor	Member
Story	1. Observe the bin info sidebar 2. Click on “Remove” underneath the name of the bin 3. Click on the subdivisions on the map to remove from the bin 4. Click on “Remove” underneath the name of the bin to finish
Scenario	Bob observes the bin info sidebar. He wants to remove subdivisions from the bin “Anglophone”. He clicks on “Remove” under the bin name “Anglophone”. He then clicks on China on the map. He clicks on “Remove” under the bin name “Anglophone” to finish.
Exceptions	

Number	6.6.5
Name	Change Bin Color
Actor	Member
Story	1. Observe the bin info sidebar 2. Click on the colored square next to the name of the bin 3. Observe the color picker 4. Choose the desired color via Hex, RGB, or manually choosing on the palette
Scenario	Bob observes the bin info sidebar. He wants to change the color associated with the bin “Anglophone” to red. He clicks on the colored square next to the bin name “Anglophone”. He observes the color picker and enters #ff0000.
Exceptions	If the subdivisions in the bin are already colored, overwrite their colors with the color of the bin.

Number	6.6.6
Name	Change Bin Name
Actor	Member
Story	1. Observe the bin info sidebar 2. Click on the name of the bin 3. Observe the editable text box 4. Type a new name into the text box 5. Press “Enter” on the keyboard or click anywhere outside the text box
Scenario	Bob observes the bin info sidebar. He wants to change the name of the bin “Anglophone”. He observes the editable text box and types “English Speaking Nations” into it. He presses “Enter” on his keyboard to save the name.
Exceptions	Blank names are not allowed. If entered, the name will not change.

Number	6.7
Name	Edit Gradient Info
Actor	Member
Story	1. Observe the map edit screen 2. Click on the “Gradient Info” tab on the edit toolbar 3. Observe the gradient info edit sidebar
Scenario	Bob observes the map edit screen. He clicks on the “Gradient Info” tab on the edit toolbar. He observes the gradient info edit sidebar.
Exceptions	If another sidebar is open, close that sidebar and replace it with this one.

Number	6.7.1
Name	Generate Gradient
Actor	Member
Story	1. Observe the gradient info sidebar 2. Choose the lower and upper gradient values 3. Press the “Generate Gradient” button
Scenario	Bob observes the gradient info sidebar. He chooses two colors for the gradient category, corresponding to the lower bound and the upper bound color, respectively. He presses “Generate Gradient” to color in subdivisions based on their weight.
Exceptions	The gradient will only apply to subdivisions with an associated weight. If subdivisions are already colored, overwrite their colors with the colors corresponding to the gradient. Only one gradient can exist at a time.

Number	6.7.2
Name	Change Gradient Color
Actor	Member
Story	1. Observe the gradient info sidebar 2. Click on the right colored square 3. Observe the color picker 4. Choose the desired first color via Hex, RGB, or manually choosing on the palette 5. Click on the left colored square 6. Observe the color picker 7. Choose the desired second color via Hex, RGB, or manually choosing on the palette
Scenario	Bob observes the gradient info sidebar.
Exceptions	The gradient will only apply to subdivisions with an associated weight. If subdivisions are already colored, overwrite their colors with the colors corresponding to the gradient.

Number	6.7.3
Name	Delete Gradient
Actor	Member
Story	1. Observe the gradient info sidebar 2. Click on the “X” button
Scenario	Bob observes the gradient info sidebar. He wants to delete the existing gradient and clicks on the “X” button.
Exceptions	

Use Case Section 7 - Discussion Creations

Number	7.1
Name	Create New Discussion
Actor	Member
Story	1. Observe the discussion board home 2. Click the “New Discussion” button and observe the discussion board creation screen. 3. Type a title into the title text box and a post into the main body text box 4. Click the “Post Button” 5. Observe the discussion board screen
Scenario	Bob observes the discussion board home. He clicks on the “New Discussion” button and observes the discussion board creation screen. He types “Ancient Maps of the World” into the title text box and types “Anyone have cool ancient maps detailing the world?” into the main body text box. He clicks the “Post Button” and observes the discussion board screen.
Exceptions	

Number	7.2
Name	View Discussion
Actor	Member
Story	1. Observe the discussion board home 2. Click on a card with the title of the discussion board 3. Observe the discussion board screen
Scenario	Bob observes the discussion board home. He clicks on a discussion board card with the title “Ancient Chinese Maps”. He then observes the discussion board screen.
Exceptions	

Number	7.3
Name	Search for Discussion
Actor	Member
Story	1. Observe the discussion board home 2. Type a query into the discussion search bar 3. Press the “Enter” button on the keyboard or click on the search icon 4. Observe the filtered discussions on the discussion board home
Scenario	Bob observes the discussion board home and types “china” into the discussion search bar. He then clicks on the search icon and is presented with the relevant discussion boards.
Exceptions	

Use Case Section 8 - Discussion Participation

Number	8.1
Name	Like Post
Actor	Member
Story	1. Observe the discussion board screen 2. Click on the like button on the initial post
Scenario	Bob observes the discussion board screen for a particular post that he likes. He clicks the like button.
Exceptions	If the post is already liked, remove the previous like. If the post is already disliked, remove the previous dislike and add a like.

Number	8.2
Name	Dislike Post
Actor	Member
Story	1. Observe the discussion board screen 2. Click on the dislike button on the initial post
Scenario	Bob observes the discussion board screen for a particular post that he dislikes. He clicks the dislike button.
Exceptions	If the post is already disliked, remove the previous dislike. If the post is already liked, remove the previous like and add a dislike.

Number	8.3
Name	Comment on Post
Actor	Member
Story	1. Observe the discussion board screen 2. Type a comment into the comment text box 3. Click the submit button
Scenario	Bob observes the discussion board screen for the “Historical Chinese Maps” discussion board. He types “Anyone have one for the Ming Dynasty?” into the comment text box. He clicks the submit button.
Exceptions	

Use Case Section 9 - Member Profile Screen

Number	9.1
Name	View Profile
Actor	Member
Story	1. Observe the name of a member (under maps, discussion posts, etc.) 2. Click on the name 3. Observe the member profile screen
Scenario	Bob observes the discussion board screen for a particular post that he likes. He clicks the like button.
Exceptions	

Number	9.2
Name	Choose Map (Profile)
Actor	Member
Story	1. Observe the member profile screen 2. Click on one of the cards that contains a map 3. Observe the map view screen
Scenario	Bob observes the member profile screen for Alice. He clicks on one of the cards for a map that Alice made. He observes the map view screen.
Exceptions	

Number	9.3
Name	View Discussion (Profile)
Actor	Member
Story	1. Observe the member profile screen 2. Click on one of the cards that contain a discussion board post 3. Observe the discussion board screen
Scenario	Bob observes the member profile screen for Alice. He clicks on one of the cards for a discussion post that Alice made. He observes the discussion board screen.
Exceptions	

User Interface Model

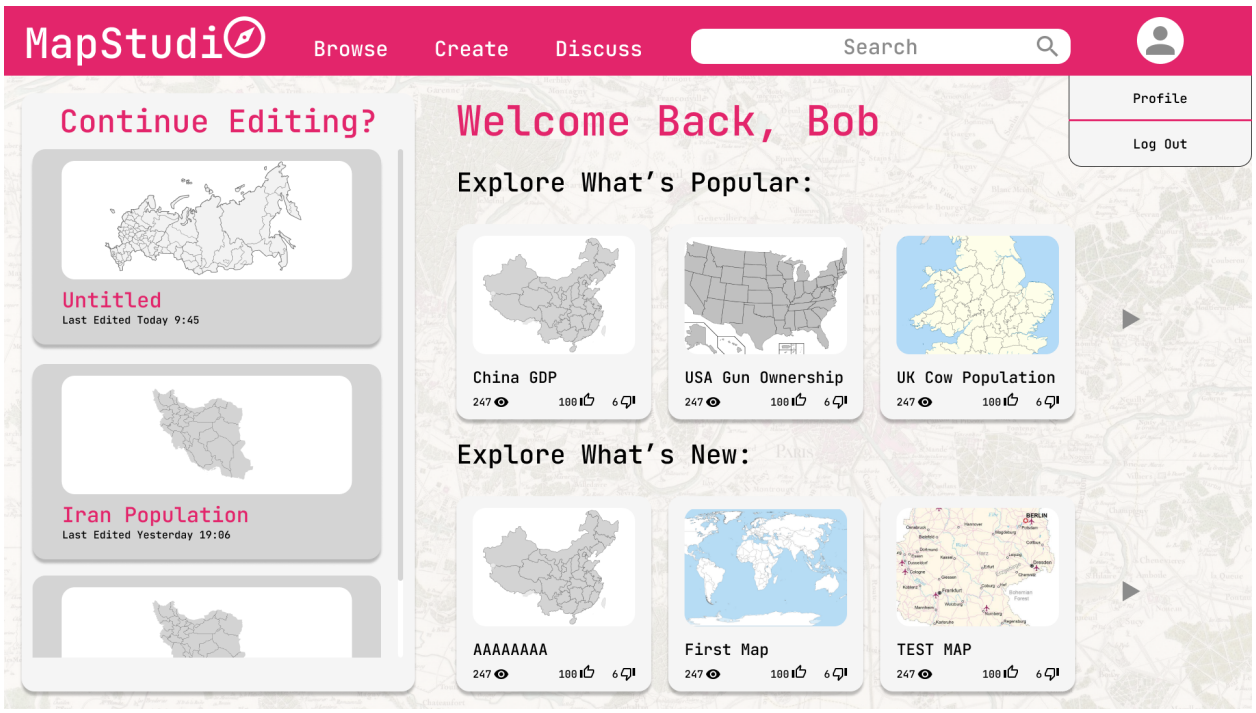
By Razzmatazz (Preston Chang, James Leonardi, Wei Qi, Kevin Tao)

User Interface Views

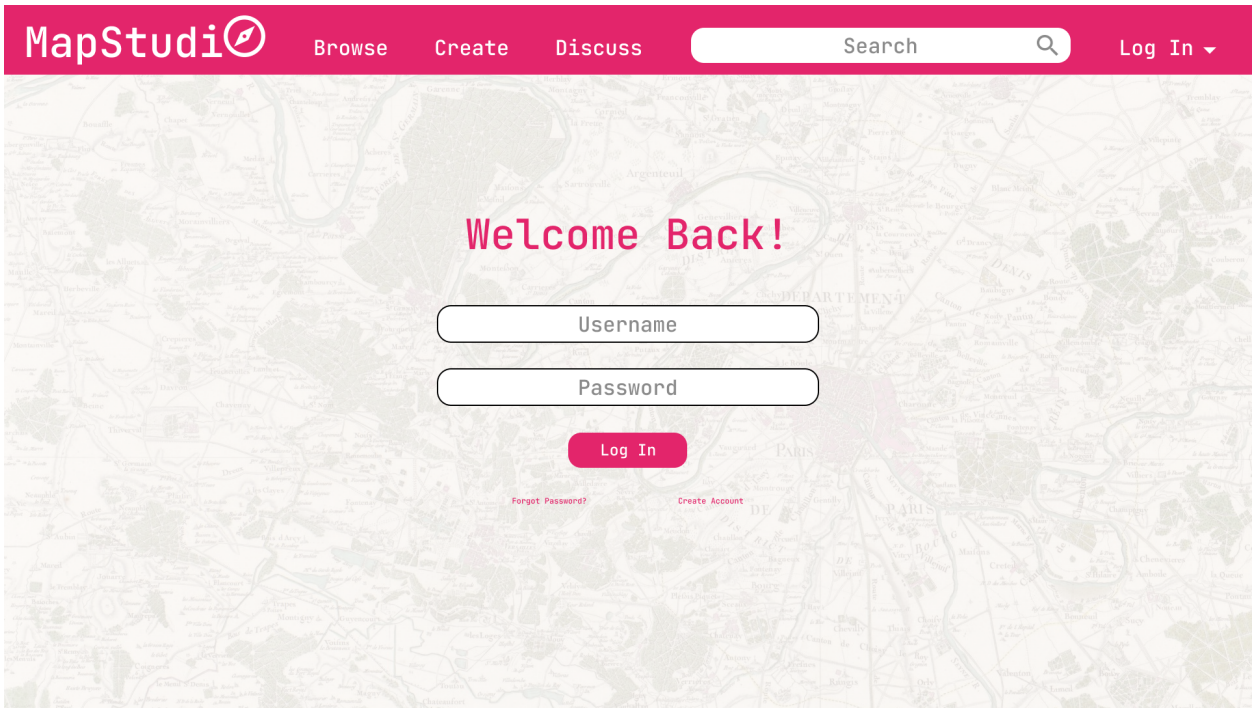
View #	*View Name*	*Description*
1	Guest Landing Page	Landing page for users not currently logged in
2	Member Landing Page	Landing page for members logged in
3	Login	Page to login for existing members
4	Forgot Password	Page to recover a forgotten password for members
5	Register	Register for a new account
6	Profile Screen	User profile, which shows the user's published maps and discussion posts
7	Map View	View an existing map's details, including its title, description, author, and comments
8	Map Search	Search for an existing map on the site
9	Personal Maps	List of all maps created by the member
10	Discussion Board	Board of discussion threads
11	Discussion Thread	Detailed view of a discussion thread
12	Create Thread	View of creating a new discussion thread
13	Map Editing	Map editing view that allows for editing of generic map properties, like name and description
14	Subdivision Editing	Map editing view that allows for editing of subdivisions and their properties
15	Point Editing	Map editing view that allows for editing of individual points and their properties
16	Bins Editing	Map editing view that allows for editing of the map's bins, including the ability to add/subtract from them
17	Gradient Editing	Map editing view that allows for editing of the map's gradients.
18	Template Editing	Map editing view that shows a list of the 5 available map types as templates

Guest Landing Page

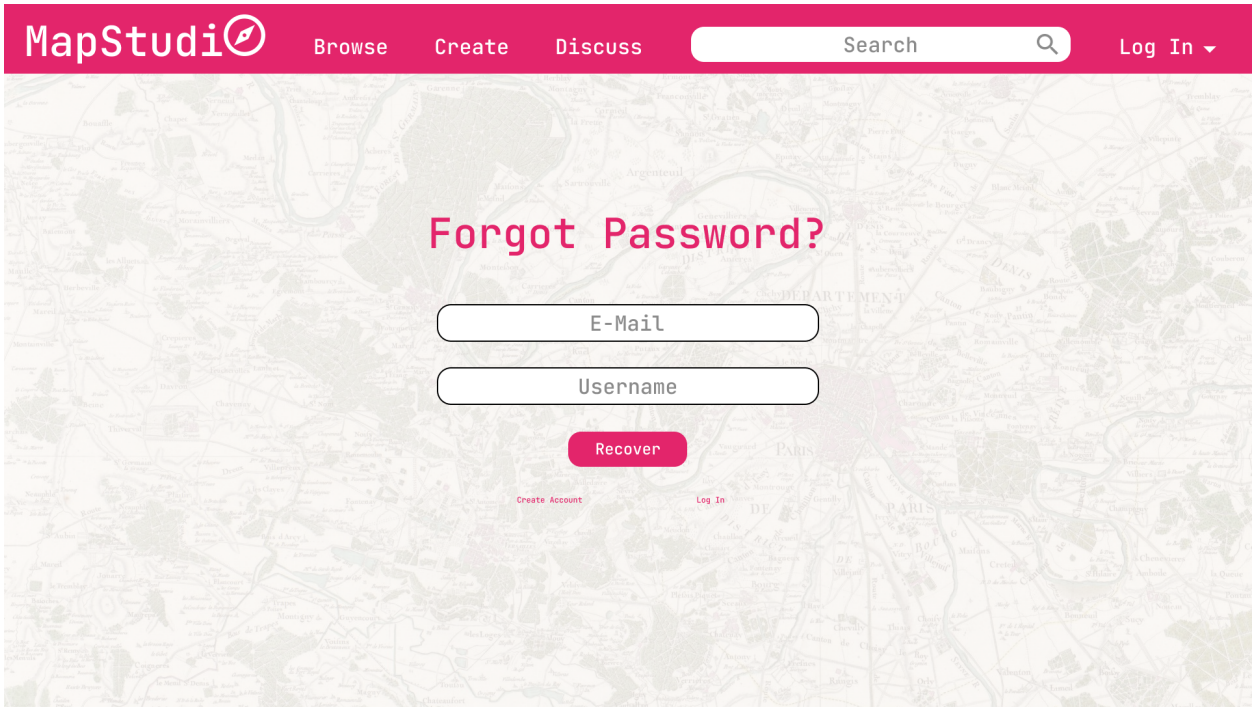
Member Landing Page



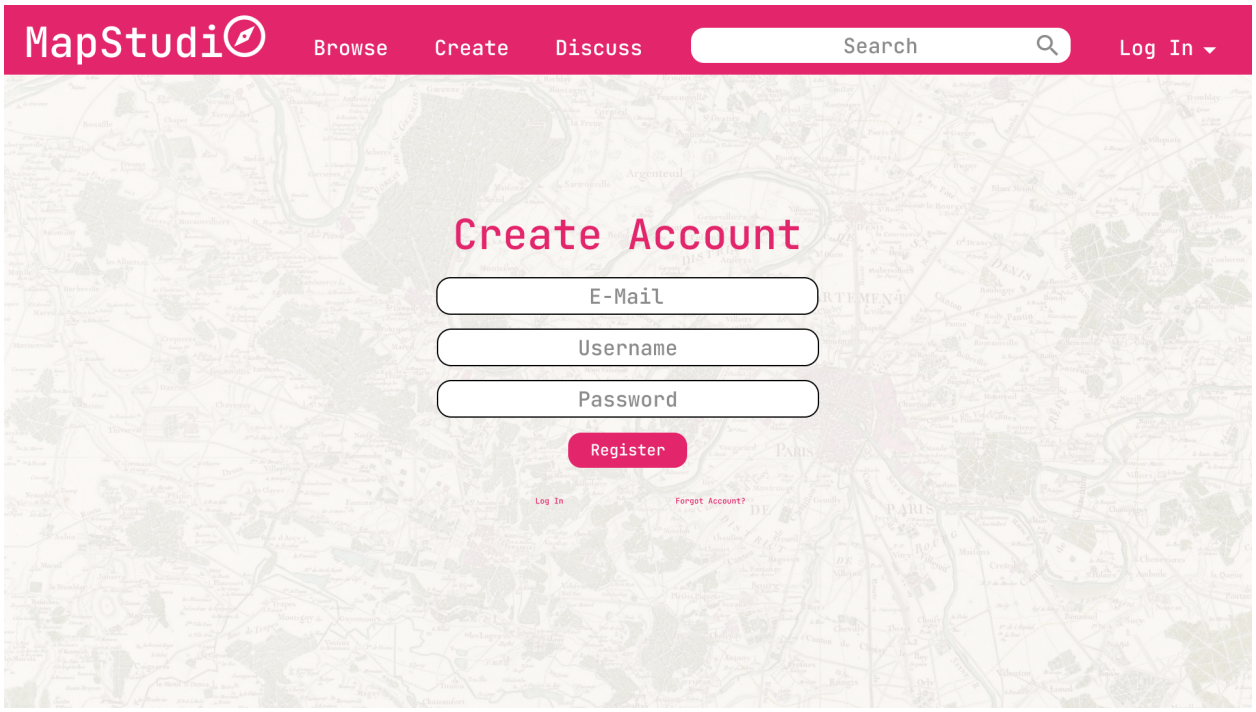
Login



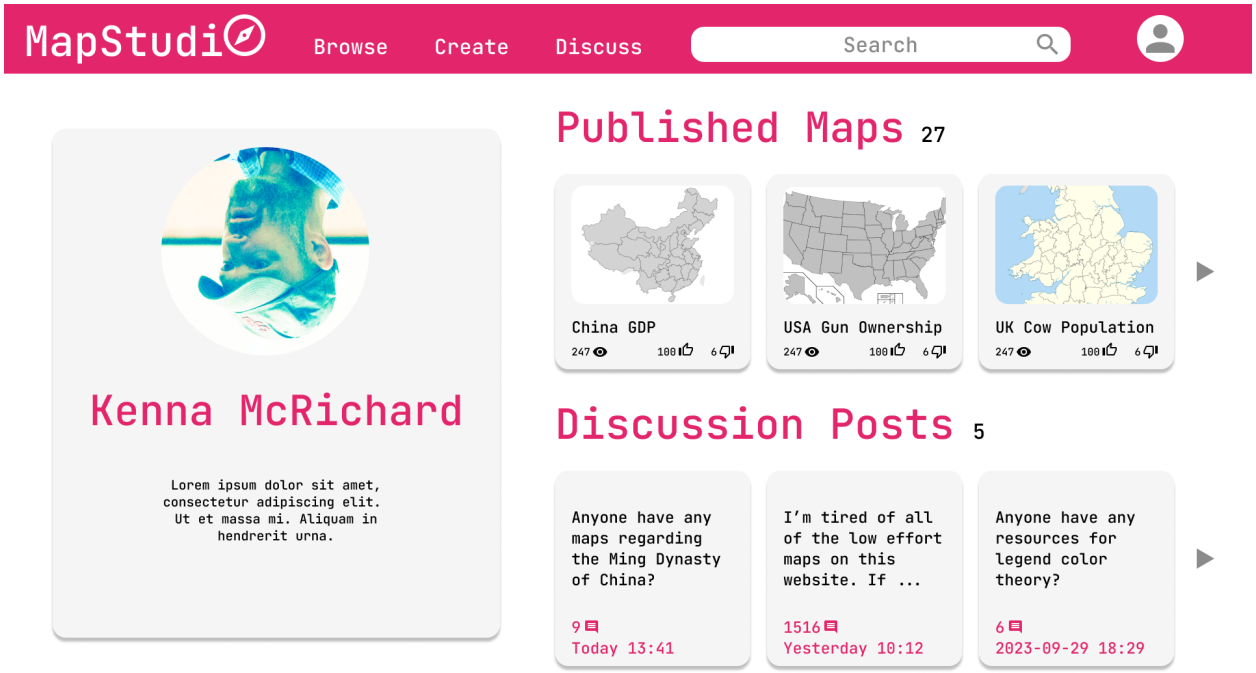
Forgot Password



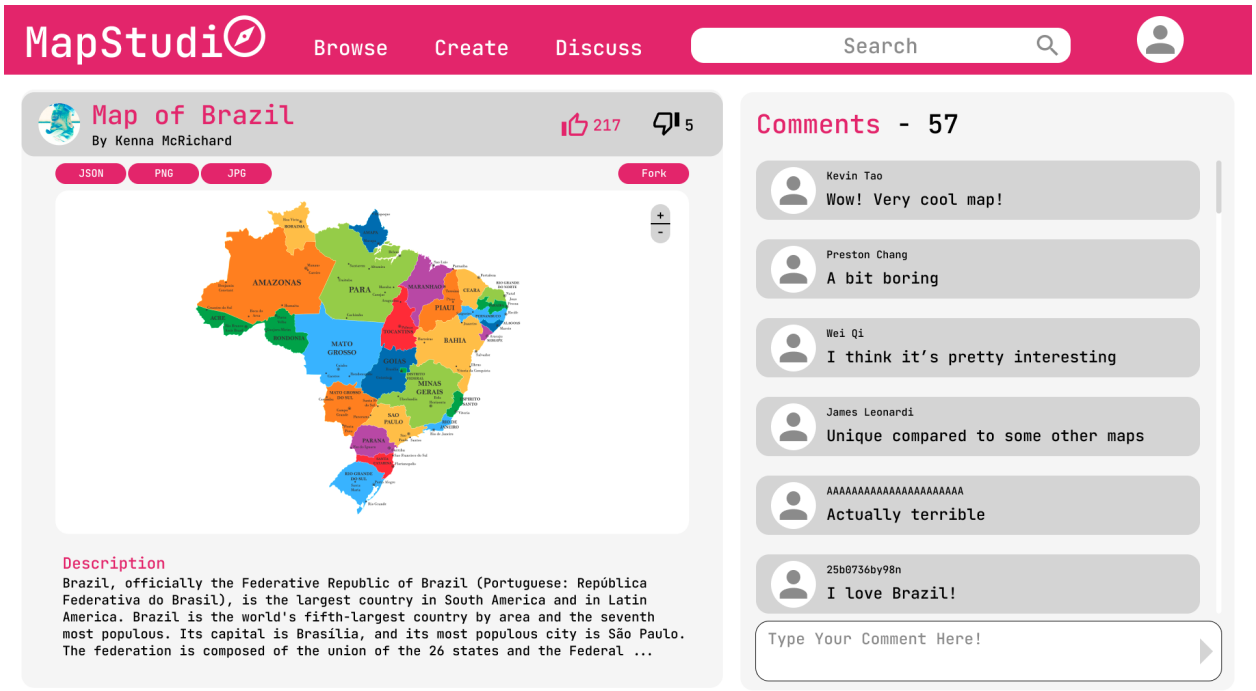
Register



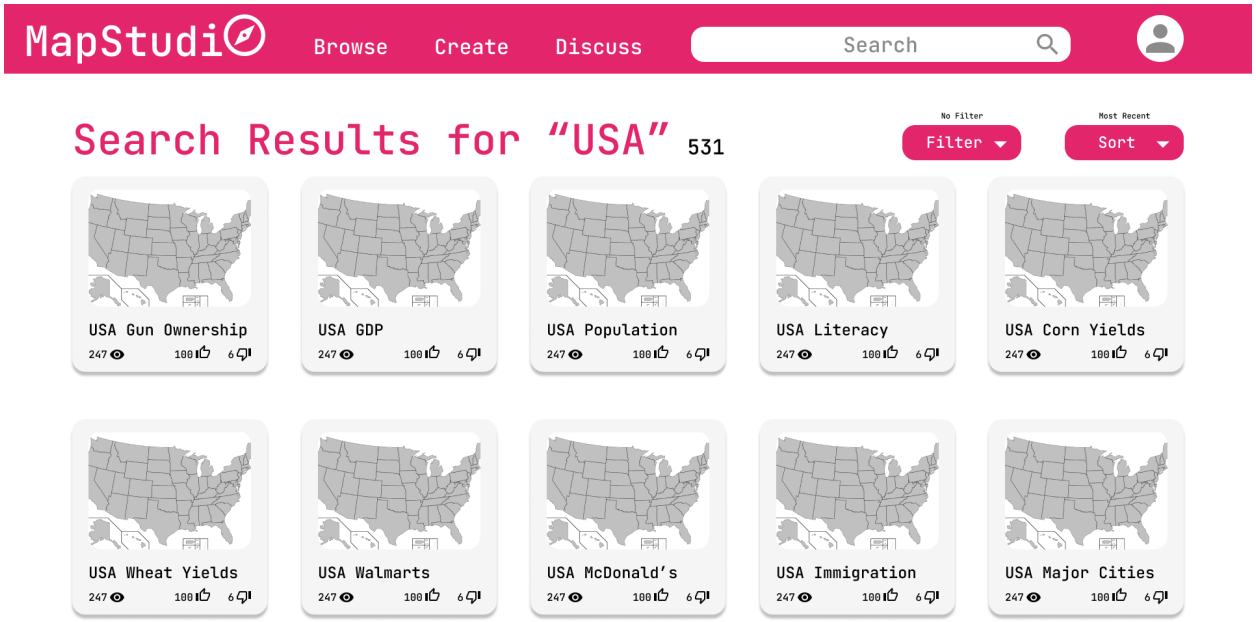
Profile Screen



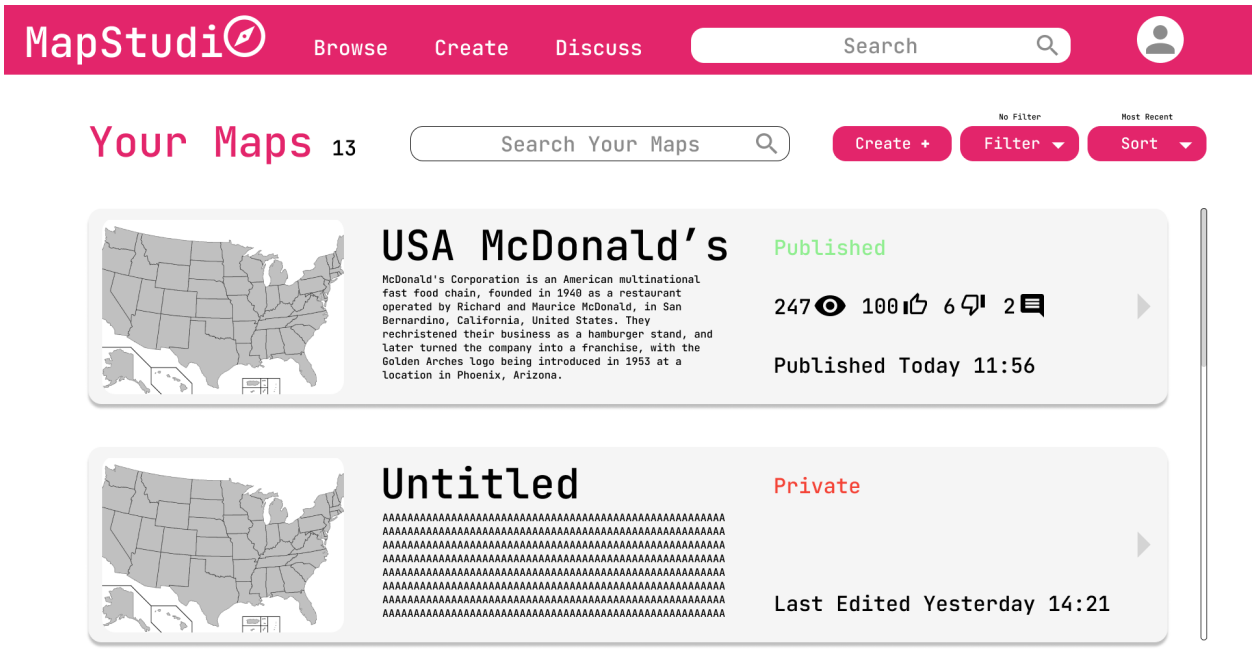
Map View



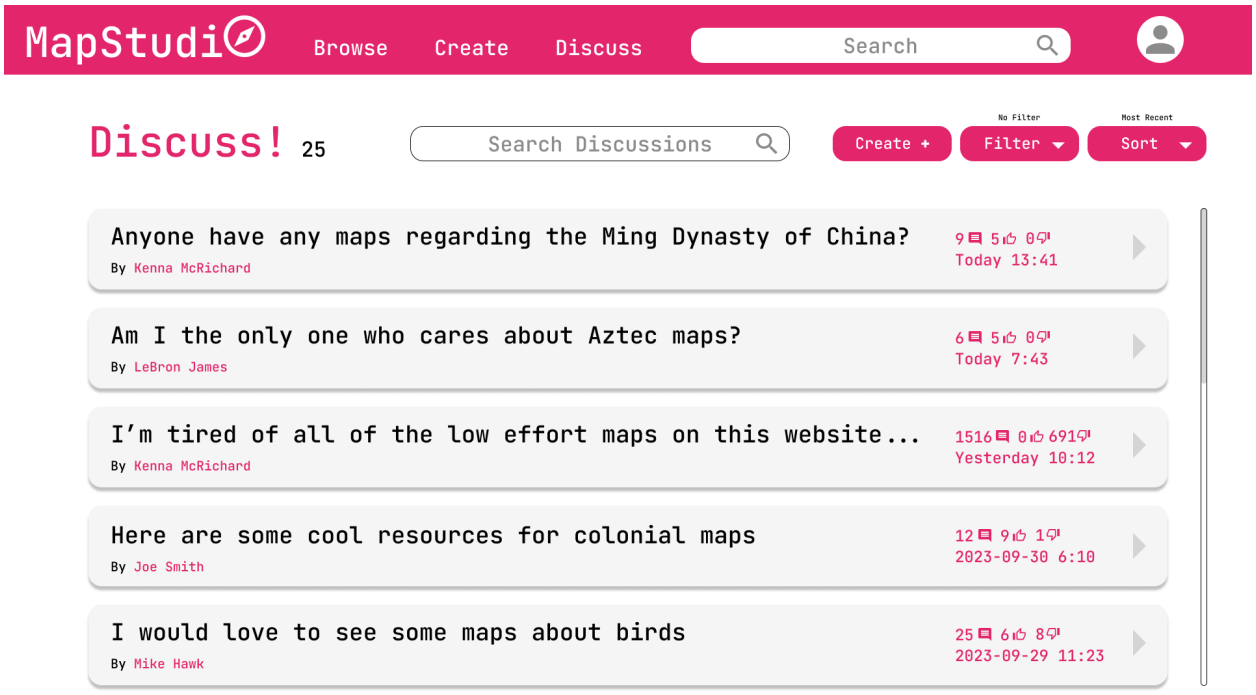
Map Search



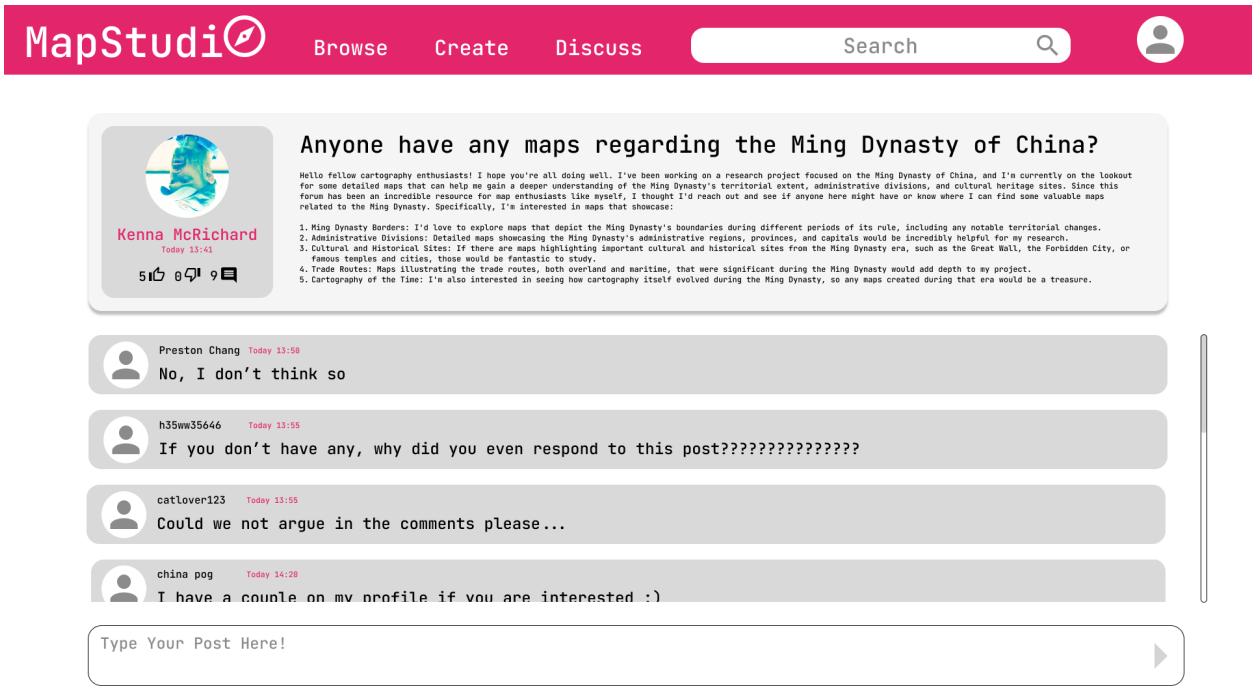
Personal Maps



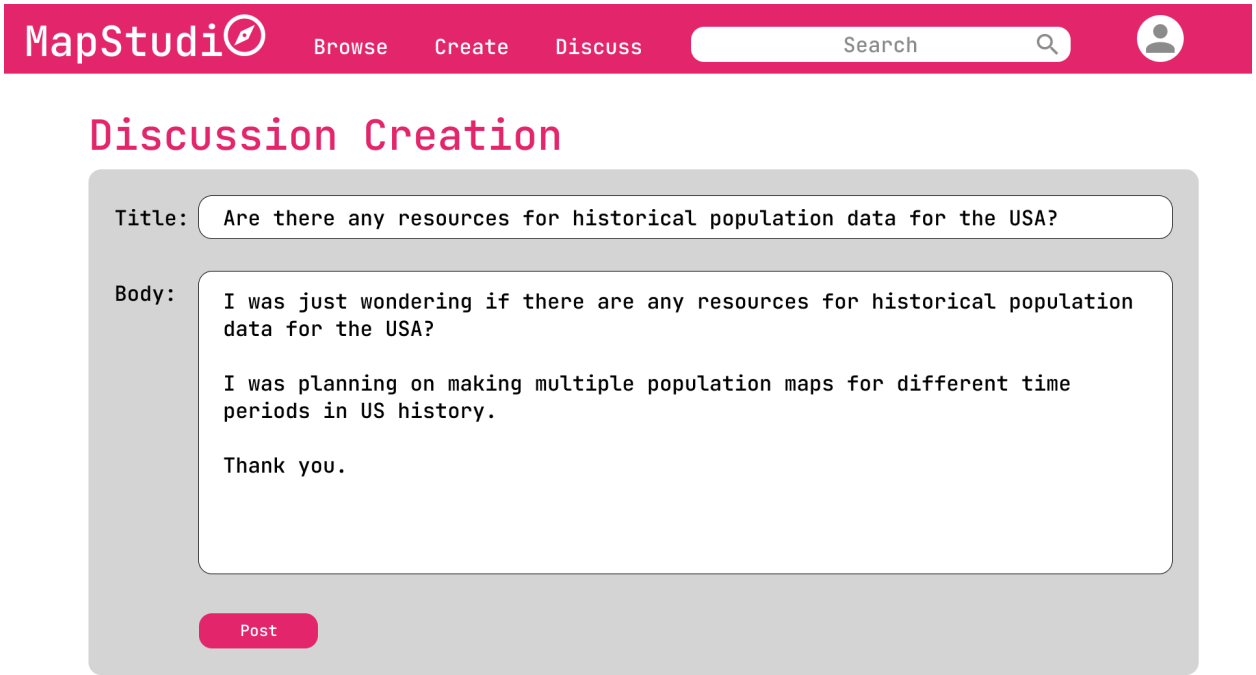
Discussion Board



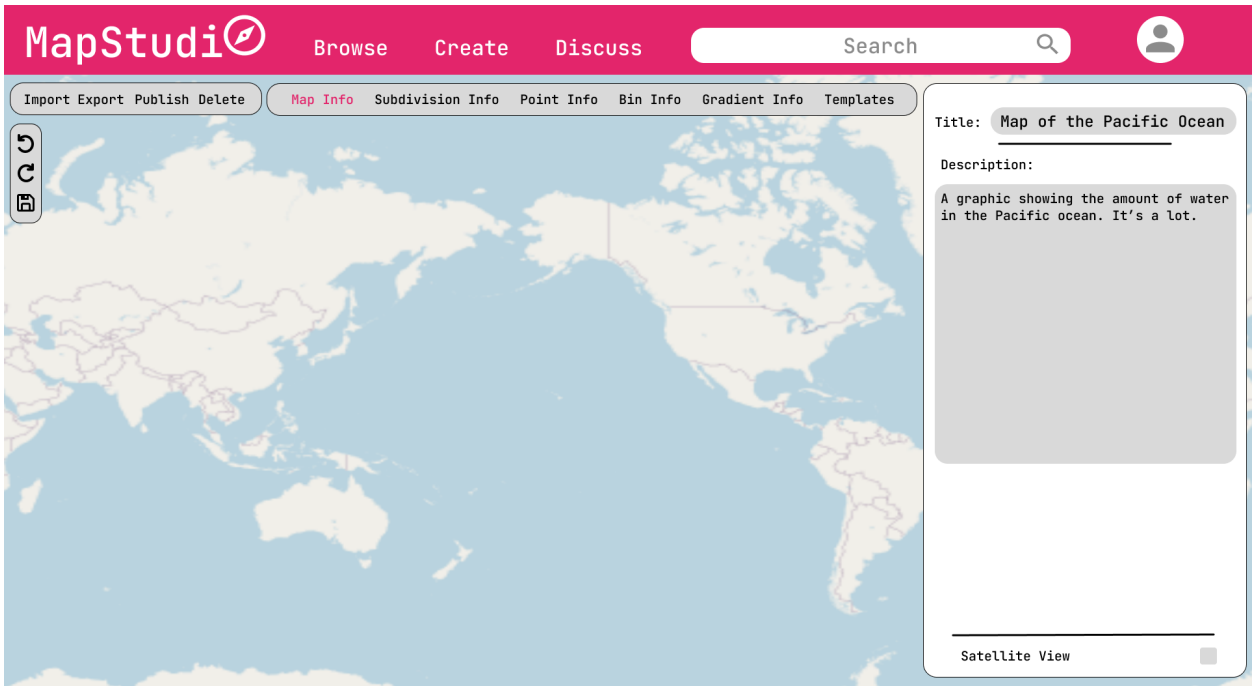
Discussion Thread



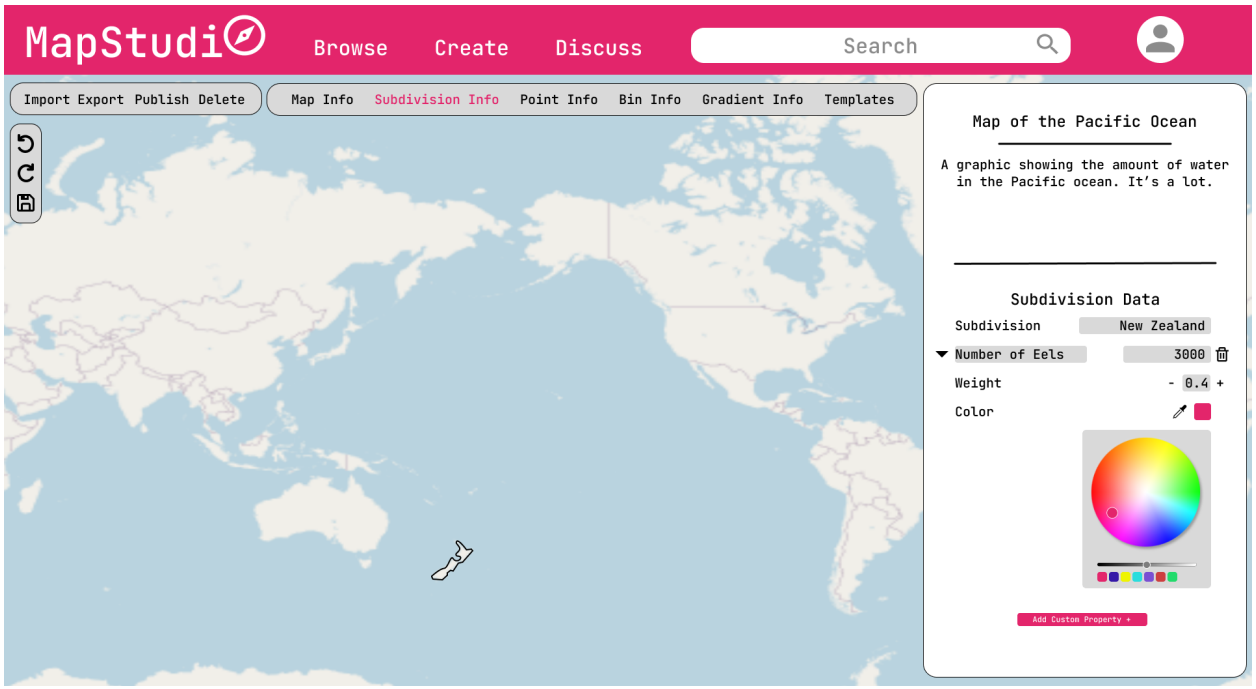
Create Thread



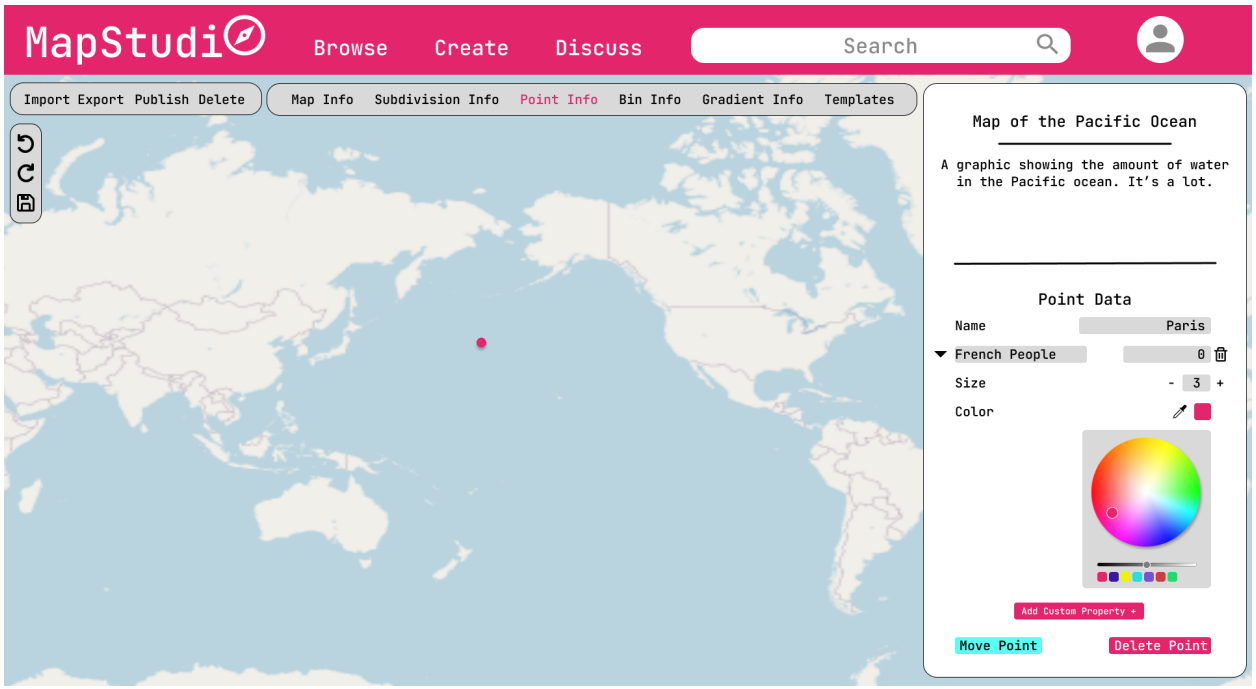
Map Editing



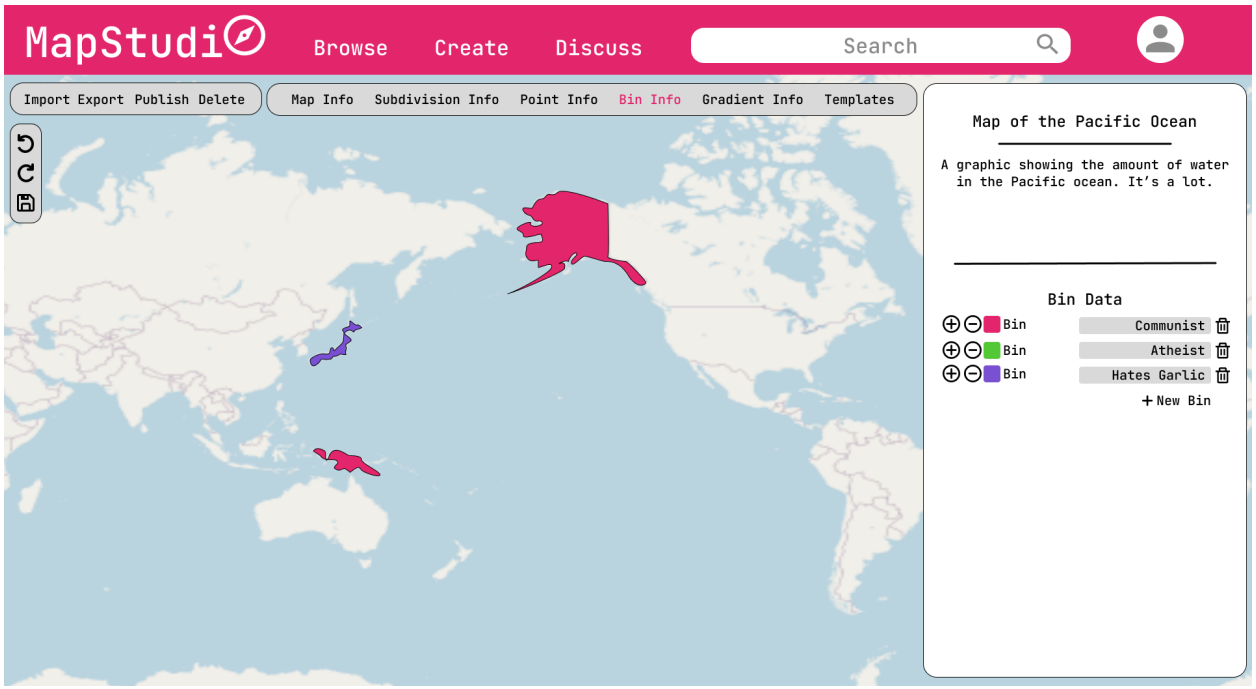
Subdivision Editing



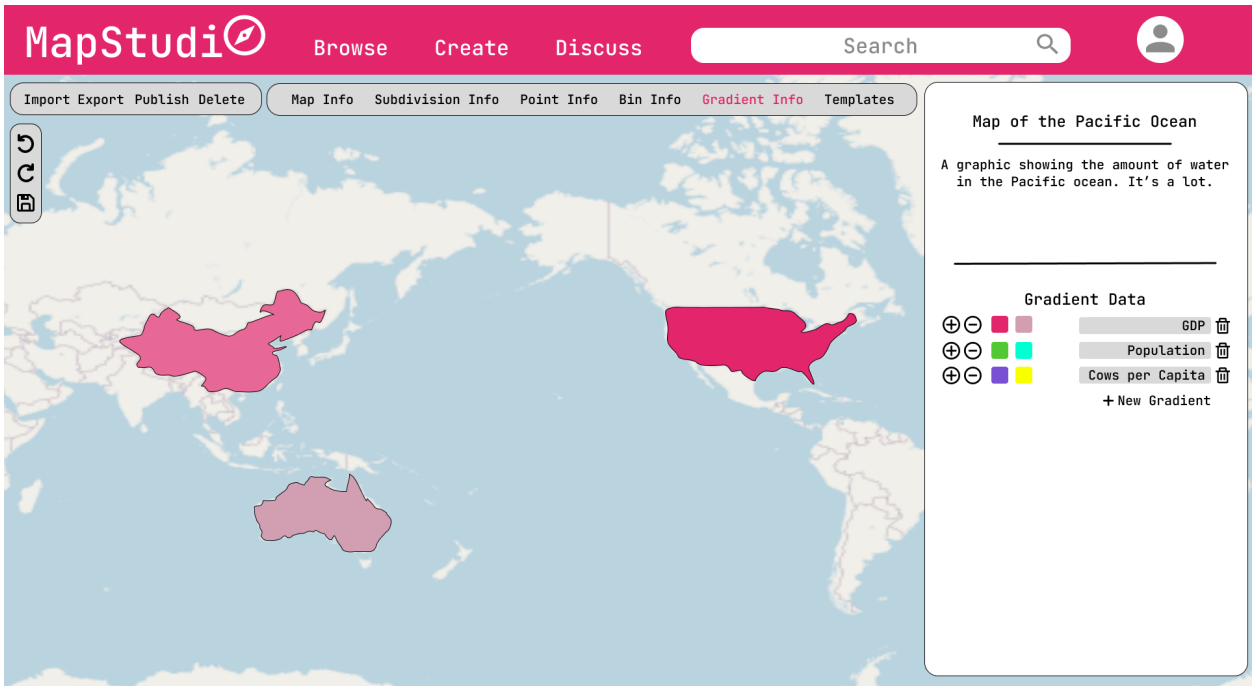
Point Editing



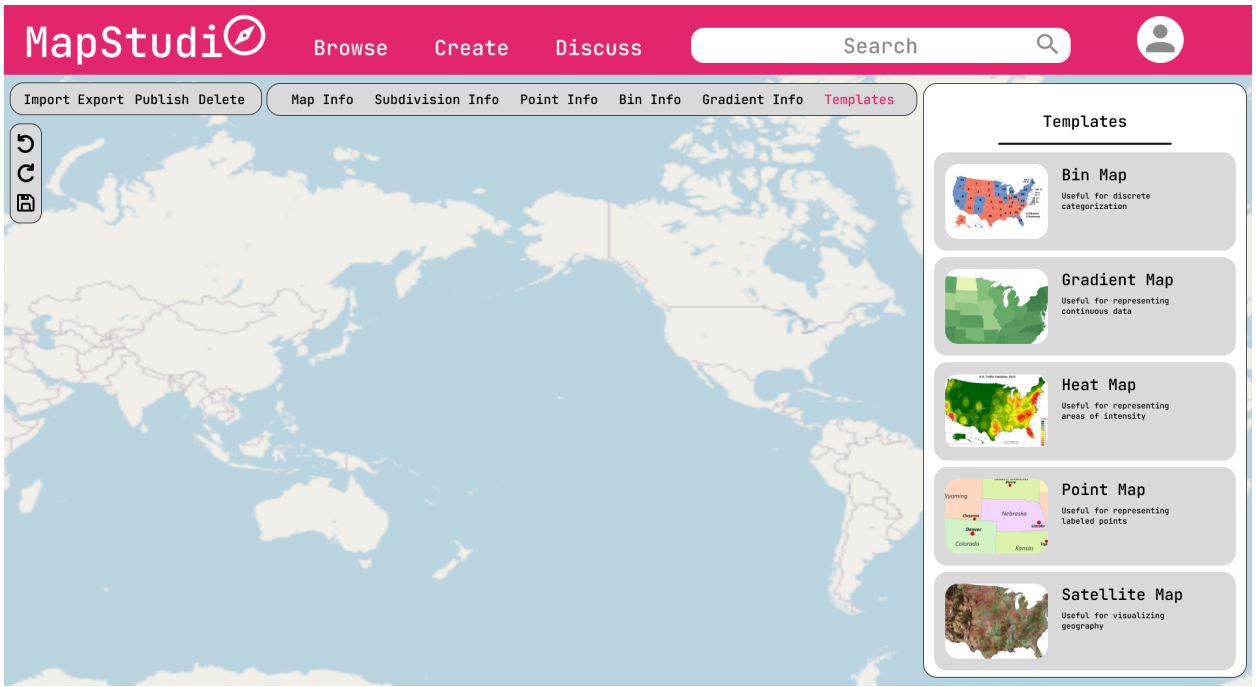
Bins Editing



Gradient Editing



Template Editing



Data Model

By Razzmatazz (Preston Chang, James Leonardi, Wei Qi, Kevin Tao)

Persistent Data Model - MongoDB/Mongoose

Map

```
const map = new Schema({
     author: { type: ObjectId, ref: 'User', required: true },
     isPublished: { type: Boolean, required: true },
     title: { type: String, required: true },
     description: { type: String, default: '' },
     likes: { type: Number, required: true },
     dislikes: { type: Number, required: true },
     comments: [{ type: ObjectId, ref: 'Comment' }],
     mapFile: { type: Buffer, required: true },
     creationDate: { type: Date, required: true, default:
     Date.now },
     updateDate: { type: Date, required: true, default: Date.now
     }
});
```

User

```
const user = new Schema({
     username: { type: String, required: true },
     email: { type: String, required: true },
     pfp: { type: String }, // file path
     bio: { type: String, default: '' },
     passwordHash: { type: String, required: true },
     maps: [{ type: ObjectId, ref: 'Map' }],
     posts: [{ type: ObjectId, ref: 'Post' }],
     joinDate: { type: Date, default: Date.now },
     likedMaps: [{ type: ObjectId, ref: 'Map' }] ,
     dislikedMaps: [{ type: ObjectId, ref: 'Map' }],
     likedComments: [{ type: ObjectId, ref: 'Comment' }],
     dislikedComments: [{ type: ObjectId, ref: 'Comment' }],
     likedPosts: [{ type: ObjectId, ref: 'Post' }],
     dislikedPosts: [{ type: ObjectId, ref: 'Post' }],
});
```

Comment

```
const comment = new Schema({
     author: { type: ObjectId, ref: 'User', required: true },
     content: { type: String, required: true },
     likes: { type: Number, required: true },
     dislikes: { type: Number, required: true },
     publishedDate: { type: Date, required: true }
});
```

Community Post

```
const post = new Schema({
     author: { type: ObjectId, ref: 'User', required: true },
     title: { type: String, required: true },
     content: { type: String, required: true },
     likes: { type: Number, required: true },
     dislikes: { type: Number, required: true },
     comments: [{ type: ObjectId, ref: 'Comment' }],
     publishedDate: { type: Date, required: true }
});
```

MapStudio JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "definitions": {
    "bin": {
      "type": "object",
      "properties": {
        "name": { "type": "string" },
        "color": { "type": "string", "default": "#000000" },
        "subdivisions": {
          "type": "array",
          "items": { "type": "string" },
          "uniqueItems": true
        }
      },
      "required": [ "name" ]
    },
    "subdivision": {
      "type": "object",
      "properties": {
        "name": { "type": "string" },
        "coordinates": {
          "type": "array",
          "items": {
            "type": "array",
            "items": {
              "type": "number"
            },
            "minItems": 2,
            "maxItems": 2
          }
        }
      },
      "required": [ "name" ],
      "additionalProperties": { "type": "number"  }
    },
    "point": {
      "type": "object",
      "properties": {
        "location": {
          "type": "object",
          "properties": {
            "lat": { "type": "number", "minimum": -90,
"maximum": 90 },
```

```
            "lon": { "type": "number", "minimum": -180,
"maximum": 180 }
          },
          "required": [ "lat", "lon" ]
        },
        "weight": {
          "type": "number",
          "maximum": 1,
          "minimum": 0,
          "default": 1
        }
      },
      "required": [ "location" ]
    },
    "gradient": {
      "type": "object",
      "properties": {
        "dataField": { "type": "string" },
        "minColor": { "type": "string", "default": "#000000" },
        "maxColor": { "type": "string", "default": "#E3256B" },
        "affectedBins": {
          "type": "array",
          "items": { "type": "string" },
          "uniqueItems": true
        }
      },
      "required": [ "dataField" ]
    }
  },
  "type": "object",
  "properties": {
    "type": {
      "type": "string",
      "enum": [ "bin", "gradient", "heatmap", "point",
"satellite" ]
    },
    "bins": {
      "type": "array",
      "items": { "$ref": "#/definitions/bin" },
      "uniqueItems": true
    },
    "subdivisions": {
      "type": "array",
      "items": { "$ref": "#/definitions/subdivision" },
```

```
      "uniqueItems": true
    },
    "points": {
      "type": "array",
      "items": { "$ref": "#/definitions/point" },
      "uniqueItems": true
    },
    "gradients": {
      "type": "array",
      "items": { "$ref": "#/definitions/gradient" },
      "uniqueItems": true
    },
    "showSatellite": {
      "type": "boolean",
      "default": false
    }
  },
  "required": [ "type" ]
}
```

Data Dictionary

Map Schema

author	An ObjectID that references an instance of the user schema belonging to the map's author. This field is required; all maps have owners.
comments	An array of ObjectIDs that each reference an instance of the comment schema that belongs to each of the comments of this map. This field is not required.
creationDate	A date object that details the exact timestamp when the map is created. This field is automatically filled when the map is created.
description	A string that contains the map's description. This field is not required.
dislikes	An integer that contains the number of dislikes that a map has. This field is required. This field must be non-negative.
isPublished	A boolean that details whether or not a map is published. This field is required; all maps are either published or not. New maps by default are not published.
likes	An integer that contains the number of likes that a map has. This field is required. This field must be non-negative.
mapFile	A custom JSON file that holds the data of the map. This is required; all maps have an associated map file.
title	A string that contains the map's title. This field is required; all maps need titles.
updateDate	A date object that details the exact timestamp when the map is saved. This field is automatically updated when the map is saved.

User Schema

bio	A string that details the user's profile description. This field is not required.
dislikedComments	An array of ObjectIDs that each reference an instance of the comment schema that belongs to the comments that the user has disliked. This field is not required.
dislikedMaps	An array of ObjectIDs that each reference an instance of the map schema that belongs to the maps that the user has disliked. This field is not required.
dislikedPosts	An array of ObjectIDs that each reference an instance of the post schema that belongs to the posts that the user has disliked. This field is not required.
email	A string that details the user's email. This field is required; all users have an associated email. At the time of creation, the entered email should be checked to make sure it adheres to a proper email format.
joinDate	A date object that details the exact timestamp when the user account was created. The field is automatically filled upon account registration..
likedComments	An array of ObjectIDs that each reference an instance of the comment schema that belongs to the comments that the user has liked. This field is not required.
likedMaps	An array of ObjectIDs that each reference an instance of the map schema that belongs to the maps that the user has liked. This field is not required.
likedPosts	An array of ObjectIDs that each reference an instance of the post schema that belongs to the posts that the user has liked. This field is not required.

maps	An array of ObjectIDs that each reference an instance of the map schema that belongs to each of the user's maps. This field is not required.
passwordHash	A string that contains the hashed password of the user. This field is required; all users have passwords.
pfp	A string that describes a path to the user's profile picture. This image should be a 128px square image in JPG or PNG format. This is not required; users will have a default pfp.
posts	An array of ObjectIDs that each reference an instance of the post schema that belongs to each of the user's posts. This field is not required.
username	A string that details the user's username. This field is required; all users have a username.

Comment Schema

author	An ObjectID that references an instance of the user schema belonging to the comment's author. This field is required; all comments have owners.
content	A string that contains the comment's content. This field is required; all comments need content.
dislikes	An integer that contains the number of dislikes that a comment has. This field is required. This field must be non-negative.
likes	An integer that contains the number of likes that a comment has. This field is required. This field must be non-negative.
publishedDate	A date object that details the exact timestamp when the comment was posted. The field is automatically filled upon the comment's posting.

Post Schema

author	An ObjectID that references an instance of the user schema belonging to the post's author. This field is required; all posts have owners.
comments	An array of ObjectIDs that each reference an instance of the comment schema that belongs to each of the comments of this post. This field is not required.
content	A string that contains the post's content. This field is required; all posts need content.
dislikes	An integer that contains the number of dislikes that a post has. This field is required. This field must be non-negative.
likes	An integer that contains the number of likes that a post has. This field is required. This field must be non-negative.
publishedDate	A date object that details the exact timestamp when the post was posted. The field is automatically filled upon the post's posting.
title	A string that contains the post's title. This field is required; all posts need titles.

JSON Schema

$schema	States that this schema complies with v7 of the IETF standard
definitions	Defines the structure of each object type used in the schema
definitions/bin	Bin is an object that has a “name” field of type string, “color” field of type string, and a “subdivisions” field that is an array of unique strings. The strings in the “subdivisions” field reference the names of subdivisions
definitions/bin/properties/color	A string that details the color of the bin, in hex format. The default value is "#000000".
definitions/bin/properties/name	A string that details the name of the bin. This field is required; all bins have names.
definitions/bin/properties/subdivisions	An array of unique strings that reference the names of subdivisions contained within the bin.
definitions/gradient	Gradient is an object that has a “dataField” field that corresponds to the data field on which this gradient describes, a “minColor” field and “maxColor” field that describe the gradient’s bounds, and an “affectedBins” field that is an array of unique strings, referencing the names of the bins that are affected by this gradient.
definitions/gradient/properties/affectedBins	An array that contains strings referencing the bins that this gradient applies to. The strings are unique.
definitions/gradient/properties/dataField	A string that details the subdivision property that this gradient is associated with. This field is required; all gradients have dataFields.
definitions/gradient/properties/maxColor	A string that details the color of the upper bound of the gradient, in hex format. The default value is "#E3256B".
definitions/gradient/properties/minColor	A string that details the color of the lower bound of the gradient, in hex format. The

	default value is "#000000".
definitions/point	Point is an object that has a “location” field that is an object containing a latitude and longitude, and a “weight” field that is a number from 0 to 1.
definitions/point/properties/location	An object that contains two properties: lat, a number which represents the latitude of the point, and lon, a number which represents the longitude of the point
definitions/point/properties/weight	A number that represents the weight of the point. This can be a value between 0 and 1, with the default being 1. The weight determines either the point’s size, in non-heat maps, or intensity, in heat maps.
definitions/subdivision	Subdivision is an object that has a “name” field of type string, an array “coordinates” of coordinates that compose the geometry of the subdivision, and a variable amount of additional fields to store numerical data correlating to this subdivision.
definitions/subdivision/properties/additionalproperties	Additional properties are able to be added to the subdivision. Each additional property has a user chosen name and an associated number value.
definitions/subdivision/properties/coordinates	An array of coordinate pairs that represent the subdivision’s geometry. The array contains arrays of length 2, with the first entry being a number that represents the longitude and the second entry being a number that represents the latitude. The points will create the outline of the subdivision.
definitions/subdivision/properties/name	A string that details the name of the subdivision. This field is required; all subdivisions have names.
properties	Defines the properties of the given instance of the schema.
properties/bins	An array of unique bin objects belonging to the map.

properties/gradients	An array of unique gradient objects belonging to the map.
properties/points	An array of unique point objects belonging to the map.
properties/showSatellite	Whether or not to show the satellite layer on the map.
properties/subdivisions	An array of unique subdivision objects belonging to the map.
properties/type	A string that describes what base type of map this is, choosing from the options of bin map, gradient map, heat map, point map, and satellite map.
type	This JSON schema is of type object.

Software Model

By Razzmatazz (Preston Chang, James Leonardi, Wei Qi, Kevin Tao)

Similar Problems

We have encountered similar problems and worked on similar projects to MapStudio in the past. Specifically, we have all worked on MERN stack applications in the past as part of our CSE 316 curriculum. Details regarding the applications that we have developed are detailed in the Training Verification section of this document. In addition, we were provided with the source code for two previous CSE 316 projects: The Todo Tracker and Top 5 Lister. From these MERN stack applications, we can analyze and study the overall design and architectural decisions that went into said applications, observe the different APIs and libraries they used to solve certain problems, and how certain functions were written and utilized. Another resource that we will heavily use and draw upon is the documentation from the various APIs, libraries, and frameworks that we will be using throughout the application.

Complete Technology Set

MongoDB - MongoDB is a NoSQL, document-oriented database. It stores data in a flexible, JSON-like format called BSON (Binary JSON). MongoDB is well-suited for handling large volumes of unstructured or semi-structured data.

Express.js - Express.js is a web application framework for Node.js. It simplifies the process of building robust, scalable, and performant web applications and APIs. It provides middleware for handling HTTP requests, routing, and other backend functionalities.

React - React is a JavaScript library for building user interfaces. It's designed for creating interactive, single-page applications (SPAs) with a component-based architecture. React is responsible for rendering the user interface and handling the client-side logic of your application.

Node.js - Node.js is a runtime environment for executing JavaScript code on the server-side. It allows you to build server-side applications using JavaScript. In the context of the MERN stack, Node.js is used as the server to run the Express.js backend.

MUI - Material UI is an open-source React CSS component library that implements Google's Material Design. It includes a comprehensive collection of prebuilt components that are ready for use right out of the box.

Leaflet - Leaflet is an open-source JavaScript library for mobile-friendly interactive maps. Leaflet supports GeoJSON files natively.

toGeoJSON - toGeoJSON is an API that allows the conversion of KML files to GeoJSON files. This fulfills the need of our application supporting KML files, despite the fact that Leaflet only supports GeoJSON files.

shapefile - shapefile is an API that allows the conversion of SHP files or SHP/DBF files to GeoJson files. This fulfills the need of our application supporting SHP/DBF files, despite the fact that Leaflet only supports GeoJSON files.

jsonwebtoken - jsonwebtoken is an implementation of JSON Web Tokens which are compact, URL-safe means of representing claims to be transferred between two parties as a JSON object. When a user logs in, the server issues a JWT that can be stored on the client side (e.g., in a cookie or local storage). For subsequent requests, the JWT is included in the HTTP header, allowing the server to identify and authenticate the user.

Axios - Axios is a popular JavaScript library that is used for making asynchronous HTTP requests to web servers. It provides a simple and easy-to-use API for performing HTTP operations, such as GET, POST, PUT, and DELETE requests. Axios supports promises, allowing work with asynchronous data in a more readable and organized manner. It can be used in both browser-based JavaScript applications and Node.js environments, making it a versatile choice for handling network requests in web development.

bcrypt - bcrypt is a hashing function specifically designed for hashing passwords. In addition to incorporating a salt to protect against rainbow table attacks, it also allows for its hashing speed to be changed, slowing down brute-force attacks.

emailjs - emailjs allows our web application to send emails. This would be used to send users a temporary password in the scenario in which they forgot their password.

Training Verification

Preston Chang - Took CSE 316 in the Fall 2022 semester with Professor McKenna. In the class, built the Playlister application, a MERN stack web application that allowed users to create, share and play custom song playlists. In addition to the MERN stack, jsonwebtoken and bcrpyt were used for account security, MUI was used for UI, and Axios for HTTP requests.

https://github.com/InfraredCookie/Playlister_FinalProject

Wei Qi - Took CSE 316 in the Spring 2022 semester with Joydeep Mitra. Developed a StackOverflow clone.Designed and implemented RESTful API for the application using Express and Node.js. Implemented real-time communication and group chat functionality using Socket.IO.Built user authentication and authorization features using MongoDB and Mongoose. Designed and implemented the user interface using HTML,CSS, Pug, FontAwesome, and Bootstrap.

Kevin Tao - Took CSE 316 in the Spring 2023 semester with Joydeep Mitra. Developed a StackOverflow clone that replicated the website's fundamental features. Employed Javascript, CSS, HTML, React, Express, MongoDB, and Axios to implement functionalities such as browsing, sorting, searching, posting, and filtering by tags. Designed and implemented a RESTful API to handle requests between client and server.

James Leonardi - Took CSE 316 in Spring 2023 with Joydeep Mitra. Used MERN stack to develop a StackOverflow clone that allowed users to post questions, answer questions, comment on answers.

Front-End Component Composition, Front-End Component Design, Back-End API

https://lucid.app/folder/invitations/accept/inv_101345d3-7206-436a-a937-211b24372d92

App

AppBanner

MUI Icon AccountCircle
MUI AppBar
MUI Box
MUI IconButton
MUI TextField
MUI Menu
MUI MenuItem
MUI Toolbar
MUI Typography
MUI Button

LandingWrapper

GuestLanding

MUI Button
MUI Button
MUI Button

Carousel

MapCard

MUI Box
MUI Icon ThumbUp
MUI Icon ThumbDown
MUI Icon Visibility
MUI Box

MUI Icon Button

MemberLanding

ContinueEditSidebar

MapCard

MUI Box
MUI Icon ThumbUp
MUI Icon ThumbDown
MUI Icon Visibility
MUI Box
MUI Comment

MUI Box

LoginScreen

MUI Textfield
MUI Textfield
MUI Button
MUI Button
MUI Button

AccountErrorModal

MUI Box
MUI Button
MUI Modal

RegisterScreen

MUI Textfield
MUI Textfield
MUI Button
MUI Button
MUI Button

AccountErrorModal

MUI Box
MUI Button
MUI Modal

RecoveryScreen

MUI Textfield
MUI Textfield
MUI Textfield
MUI Button
MUI Button
MUI Button

ChangePasswordScreen

MUI Textfield
MUI Button

ProfileScreen

Carousel

MapCard

MUI Box
MUI Icon ThumbUp
MUI Icon ThumbDown
MUI Icon Visibility
MUI Box
MUI Comment

MUI Icon Button

Carousel

DiscussionCard

MUI Box
MUI Icon Comment
MUI Box
MUI Icon ThumbUp
MUI Icon ThumbDown

MUI Icon Button

MUI Avatar
MUI Box
MUI Textfield

AvatarUploadErrorModal

MUI Box
MUI Button
MUI Modal

SearchResults

MapCard

MUI Box
MUI Icon ThumbUp
MUI Icon ThumbDown
MUI Icon Visibility
MUI Box
MUI Icon Comment

MUI Select
MUI Select
MUI Box

PersonalMaps

MapCard

MUI Box
MUI Icon ThumbUp
MUI Icon ThumbDown
MUI Icon Visibility
MUI Box
MUI Icon Comment

MUI Select
MUI Select
MUI Box
MUI Button
MUI Textfield
MUI Icon Search

DiscussionBoard

DiscussionCard

MUI Box
MUI Icon Comment
MUI Box
MUI Icon ThumbUp
MUI Icon ThumbDown

MUI Select
MUI Select
MUI Box
MUI Button
MUI Textfield
MUI Icon Search

DiscussionThread

Comment

MUI Box
MUI Icon ThumbUp
MUI Icon ThumbDown
MUI Avatar
MUI Box

MUI Avatar
MUI Box
MUI Icon ThumbUp
MUI Icon ThumbDown
MUI Icon Comment
MUI Box
MUI Textfield
MUI Icon ArrowRight

DiscussionCreate

MUI Box
MUI Box
MUI Textfield
MUI Textfield
MUI Button

MapView

MapInfo

MUI Box
MUI Icon ThumbUp
MUI Icon ThumbDown
MUI Avatar
MUI Button
MUI Button
MUI Button
MUI Box
MUI ButtonGroup
MUI Icon Add
MUI Icon Delete
Leaftlet tileLayer
Leaftlet GeoJson
MUI Box

MapComments

Comment

MUI Box
MUI Icon ThumbUp
MUI Icon ThumbDown
MUI Avatar
MUI Box

MUI Box
MUI Textfield
MUI Box
MUI Icon ArrowRight

MapEditor

FileToolbar

MUI ButtonGroup
MUI Button x4

MenuToolbar

MUI ButtonGroup
MUI Button x6

TransactionToolbar

MUI ButtonGroup
MUI Icon Undo
MUI Icon Redo
MUI Icon Save
MUI Button

Leaftlet tileLayer
Leaftlet GeoJson
Leaftlet.heat heatLayer

MapInfoSidebar

MUI Box
MUI Checkbox
MUI Textfield
MUI Textfield

TemplateSidebar

MUI Box x5
MUI Box x5

SubdivisionInfoSidebar

MUI Box
MUI Button
MUI Textfield

CustomProperty

MUI Icon ArrowDrop Down
MUI Icon ArrowRight
MUI Icon Delete
MUI Icon Colorize
React-Color Block
MUI Box

PointInfoSidebar

MUI Box
MUI Button
MUI Textfield

CustomProperty

MUI Icon ArrowDrop Down
MUI Icon ArrowRight
MUI Icon Delete
MUI Icon Colorize
React-Color Block
MUI Box

BinInfoSidebar

MUI Box
MUI Textfield

CustomProperty

MUI Icon ArrowDrop Down
MUI Icon ArrowRight
MUI Icon Delete
MUI Icon AddCircle Outline
MUI Icon RemoveCircle Outline
React-Color Block
MUI Box

GradientInfoSidebar

MUI Box
MUI Textfield

CustomProperty

MUI Icon ArrowDrop Down
MUI Icon ArrowRight
MUI Icon Delete
MUI Icon AddCircle Outline
MUI Icon RemoveCircle Outline
React-Color Block
MUI Box

DeleteMapModal

MUI Box
MUI Button
MUI Modal

PublishMapModal

MUI Box
MUI Button
MUI Modal

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);

// The current value in the search box
const [searchQuery, setSearchQuery] = useState('');

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

```
// Response for when a keyboard key is pressed
const handleKeyPress = (event) ...

// 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 Profile menu item
const handleProfile = () => { ...

// 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 Forgot Password menu item
const handleForgot = () => { ...

// Responds to Browse top bar tab
const handleBrowse = () => { ...

// Responds to Create top bar tab
const handleCreate = () => { ...

// Responds to Discuss top bar tab
const handleDiscuss = () => { ...

// Responds to Search box query change
const handleSearchChange = (event) => { ...

// Responds to Search
const handleSearch = () => { ...
```

MapSidebar

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

// Title text
const [title, setTitle] = useState('');

// Description text
const [desc, setDesc] = useState('');

// Handle title text change
const handleTitleChange = (event) => { ...

// Handle desc text change
const handleDescChange = (event) => { ...

// Handle satellite view toggle
const handleSatelliteChange = () => { ...
```

SubdivisionSidebar

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

// Name text
const [name, setName] = useState('');

// Weight value
const [weight, setWeight] = useState(0);

// Subdivision color
const [color, setColor] = useState('');

// Handle weight change
const handleWeightChange = (event) => { ...

// Handle color change
const handleColorChange = (event) => { ...

// Handle new custom property
const handleNewProperty = () => { ...
```

SubdivisionProperty

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

// Name text - will not be editable if not custom
const [name, setName] = useState('');

// Value
const [value, setValue] = useState(0);

// Handle name text change
const handleNameChange = (event) => { ...

// Handle property value change
const handleValueChange = (event) => { ...

// Handle delete property
const handleDeleteProperty = () => { ...
```

BinSidebar

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

// Handle new bin
const handleNewBin = () => { ...
```

BinItem

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

// Bin name
const [name, setName] = useState('');

// Color
const [color, setColor] = useState('');

// Affected areas
const [affectedAreas, setAffectedAreas] = useState([]);

// Add/remove mode - enum
const [editMode, setEditMode] = useState(0);

// Handle name text change
const handleNameChange = (event) => { ...

// Handle color change
const handleColorChange = (event) => { ...

// Handle delete bin
const handleDeleteBin = () => { ...

//  Handle add area button
const handleAddArea = () => { ...

//  Handle remove area button
const handleRemoveArea = () => { ...

// Handle click on region when editing
const handleRegionSelect = (event) => { ...
```

TemplateSidebar

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

// Handle apply Bin template
const handleBinTemplate = () => { ...

// Handle apply Gradient template
const handleGradientTemplate = () => { ...

// Handle apply Heat template
const handleHeatTemplate = () => { ...

// Handle apply Point template
const handlePointTemplate = () => { ...

// Handle apply Satellite template
const handleSatelliteTemplate = () => { ...

// Handle confirm switch to template
const handleConfirmation = () => { ...

// Handle cancel switch to template
const handleCancellation = () => { ...
```

PersonalMaps

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

// User maps
const [maps, setMaps] = useState([]);

// Search query
const [query, setQuery] = useState('');

// Filter setting
const [filterBy, setFilterBy] = useState('');

// Sort setting
const [sortBy, setSortBy] = useState('');
```

```
// Response for when a keyboard key is pressed
const handleKeyPress = (event) ...

// Handle filter change
const handleFilter = () => { ...

// Handle sort change
const handleSort = () => { ...

// Handle search query change
const handleSearchChange = (event) => { ...

// Handle search
const handleSearch = () => { ...
```

EditMap

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

// Loaded map data
const [mapData, setMapData] = useState(null);

// Map preview zoom status
const [zoomLevel, setZoomLevel] = useState(0);

// Which sidebar is open
const [openSidebar, setOpenSidebar] = useState('');

const tps : jsTPS
```

```
// Response for when a keyboard key is pressed
const handleKeyPress = (event) ...

// Handle zoom in on map preview
const handleZoomIn = () => { ...

// Handle zoom out on map preview
const handleZoomOut = () => { ...

// Handle import map button
const handleImport = () => { ...

// Handle export map button
const handleExport = () => { ...

// Handle publish map button
const handlePublish = () => { ...

// Handle delete map button
const handleDelete = () => { ...

////////////////////////////////////////////////////
// Handle undo last action
const handleUndo = () => { ...

// Handle redo last action
const handleRedo = () => { ...

// Tests to see if a redo operation is possible
const canRedo = function() ...

// Tests to see if an undo operation is possible
const canUndo = function() ...

// Handle save map
const handleSave = () => { ...

////////////////////////////////////////////////////
// Handle open Map info sidebar
const handleMapInfo = () => { ...

// Handle open Subdivision info sidebar
const handleSubdivisionInfo = () => { ...

// Handle open Point info sidebar
const handlePointInfo = () => { ...

// Handle open Bin info sidebar
const handleBinInfo = () => { ...

// Handle open Gradient info sidebar
const handleGradientInfo = () => { ...

// Handle open Templates sidebar
const handleTemplates = () => { ...
```

PointSidebar

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

// Name text
const [name, setName] = useState('');

// Size value
const [size, setSize] = useState(1);

// Point color
const [color, setColor] = useState('');

// Point location
const [loc, setLoc] = useState([]);

// Edit active
const [editActive, setEditActive] = useState(false);
```

```
// Handle size change
const handleWeightChange = (event) => { ...

// Handle color change
const handleColorChange = (event) => { ...

// Handle new custom property
const handleNewProperty = () => { ...

// Start Edit
const handleStartEditLocation = (event) { ...

// When an items starts to be dragged
const handleDragStart = (event, targetId) { ...

// When an item is placed into another item
const handleDrop = (event, targetId) { ...
```

PointProperty

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

// Name text - will not be editable if not custom
const [name, setName] = useState('');

// Value
const [value, setValue] = useState(0);
```

```
// Handle name text change
const handleNameChange = (event) => { ...

// Handle property value change
const handleValueChange = (event) => { ...

// Handle delete property
const handleDeleteProperty = () => { ...
```

GradientSidebar

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

// Handle new gradient
const handleNewGradient = () => { ...
```

GradientItem

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

// Gradient name
const [name, setName] = useState('');

// Color
const [color, setColor] = useState('');

// Affected areas
const [affectedAreas, setAffectedAreas] = useState([]);
```

```
// Handle name text change
const handleNameChange = (event) => { ...

// Handle color change
const handleColorChange = (event) => { ...

// Handle delete gradient
const handleDeleteBin = () => { ...

//  Handle add area button
const handleAddArea = () => { ...

//  Handle remove area button
const handleRemoveArea = () => { ...

// Handle click on region when editing
const handleRegionSelect = (event) => { ...
```

GradientSubdivision

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

// Value
const [value, setValue] = useState(0);
```

```
// Handle property value change
const handleValueChange = (event) => { ...
```

DiscussionBoard

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

// List of all posts
const [posts, setPosts] = useState([]);

// Search query
const [query, setQuery] = useState('');

// Filter setting
const [filterBy, setFilterBy] = useState('');

// Sort setting
const [sortBy, setSortBy] = useState('');
```

```
// Response for when a keyboard key is pressed
const handleKeyPress = (event) { ...

// Handle filter change
const handleFilter = () => { ...

// Handle sort change
const handleSort = () => { ...

// Handle search query change
const handleSearchChange = (event) => { ...

// Handle search
const handleSearch = () => { ...

// Handle create new post
const handleCreate = () => { ...
```

CreatePost

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

// Title
const [title, setTitle] = useState('');

// Description
const [desc, setDesc] = useState('');
```

```
// Handle Post
const handlePost = () => { ...
```

jsTPS

jsTPS_Transaction

MapEdit_Transaction

```
store : GlobalStoreContext
oldItemIndex : Number
newItemIndex : Number

constructor(initStore, initOldData, initNewData)
doTransaction()
undoTransaction()
```

SubdivisionEdit_Transaction

```
store : GlobalStoreContext
oldItemIndex : Number
newItemIndex : Number

constructor(initStore, initOldData, initNewData)
doTransaction()
undoTransaction()
```

BinEdit_Transaction

```
store : GlobalStoreContext
oldItemIndex : Number
newItemIndex : Number

constructor(initStore, initOldData, initNewData)
doTransaction()
undoTransaction()
```

GradientEdit_Transaction

```
store : GlobalStoreContext
oldItemIndex : Number
newItemIndex : Number

constructor(initStore, initOldData, initNewData)
doTransaction()
undoTransaction()
```

PointEdit_Transaction

```
store : GlobalStoreContext
oldItemIndex : Number
newItemIndex : Number

constructor(initStore, initOldData, initNewData)
doTransaction()
undoTransaction()
```

GlobalStoreActionType

```
UPDATE_MAP_ID_NAME_PAIRS
UPDATE_DISCUSSION_ID_NAME_PAIRS
LOAD_MAP_ID_NAME_PAIRS
LOAD_DISCUSSION_ID_NAME_PAIRS
SET_CURRENT_MAP
SET_CURRENT_DISCUSSION
CREATE_NEW_MAP
CREATE_NEW_DISCUSSION
HIDE_MODALS
```

GlobalStoreContextProvider

```
const [store, setStore] = useState({
    activePage: 'GuestLandingPage',
    mapIdNamePairs: [],
    discussionIdNamePairs: [],
    userIsdNamePairs: [],
    currentMap: null,
    sortMethod: 'Date',
    filterMethod: 'None',
    currentModal: 'None',
    currentProfile: null,
    currentDiscussionPost: null
});
```

```
//GENERAL ACTIONS

// Change active page
store.changePage = function() ...

//Hide modal
store.hideModal = function () ...

// Reducer function to update store state
const storeReducer = (action) => { ...


//MAP ACTIONS

// Adds a transaction for changing item text in the current list
store.addUpdateItemTransaction = function (index, newText) ...

// Changes the current map name
store.changeMapName = async function (id, newName) ...

// Creates a new map in the database
store.createNewMap = async function () ...

// Starts the process of deleting the current map
store.deleteMapStart = async function () ...

// Completes the process of deleting a map
store.deleteMapConfirm = function () ...

// Starts the process of publishing the current map
store.publishMapStart = async function () ...

// Completes the process of publishing a map
store.publishMapConfirm = function () ...

// Gets all the id/name pairs for the maps
store.loadMapIdNamePairs = async function () ...

// Cancel the delete
store.deleteMapCancel = function () ...

// Updates the map being edited in the database
store.updateCurrentMap = async function () ...

// Like a map
store.likeMap = function () ...

// Dislike a map
store.dislikeMap = function () ...

//Show delete modal
store.showDeleteModal = function () ...

//Show publish modal
store.showPublishModal = function () ...


//DISCUSSION ACTIONS

// Gets all the id/name pairs for the discussions
store.loadDiscussionIdNamePairs = async function () ...

// Creates a new discussion in the database
store.createNewDiscussion = async function () ...

// Updates the current discussion in the database
store.updateCurrentDiscussion = async function () ...

// Like a discussion
store.likeDiscussion = function () ...

// Dislike a discussion
store.dislikeDiscussion = function () ...

//Filter discussion
store.filterDiscussions = function (filterMethod) ...

//Sort discussion
store.sortDiscussions = function (sortMethod) ...


//COMMENT ACTIONS

// Creates a new comment in the database
store.createNewComment = async function () ...

// Updates the current comment in the database
store.updateComment = async function (comment) ...

// Like a comment
store.likeComment = function (comment) ...

// Dislike a comment
store.dislikeComment = function (comment) ...


//MENU BAR ACTIONS

//Change page to default search (browse)
store.changeViewBrowse = async function () ...

//Change page to editor
store.changeViewCreate = async function () ...

//Change page to discussion home
store.changeViewDiscuss = async function () ...

//Search for maps
store.searchMaps = function (keyword) ...


//SEARCH SCREEN ACTIONS

//Filter maps
store.filterMaps = function (filterMethod) ...

//Sort maps
store.sortMaps = function (sortMethod) ...


//USER PROFILE ACTIONS

// Updates the user's personal profile
store.updateProfile = async function () ...

// Updates the user's profile description
store.updateProfileDescription = function () ...

// Updates the user's profile picture
store.updateProfilePicture = function () ...
```

GlobalStoreHttpRequestApi

```
createMap(title, desc, author) {
    method: POST
    route : /api/map
    body: {
        author: String,
        title: String,
        desc: String
    }
}

deleteMapById(id) {
    method: DELETE
    route: /api/map/${id}
    body: {}
}

getMapById(id) {
    method: GET
    route: /api/map/${id}
    body: {}
}

updateMapInfoById(id, map) {
    method: PUT
    route: /api/map/${id}
    body: {
        map : {
            isPublished: boolean
            title: String,
            description: String,
            likes: Number,
            dislikes: Number
        }
    }
}

addCommentToMap(id, comment) {
    method: PUT
    route: /api/map/${id}/comment
    body: {
        comment: {
            author: String,
            content: String
        }
    }
}

updateCommentById(id, comment) {
    method: PUT
    route: /api/comment/${commentId}
    body: {
        comment: {
            content: String,
            likes: Number,
            dislikes: Number
        }
    }
}

deleteCommentById(id, commentId) {
    method: DELETE
    route: /api/comment/${commentId}
    body: {}
}

getCommentById(id) {
    method: GET
    route: /api/commentId/${id}
    body: {}
}

createPost(post) {
    method: POST
    route: /api/post
    body: {
        post: {
            author: String,
            title: String,
            content: String
        }
    }
}

getPostById(id) {
    method: GET
    route: /api/post/${id}
    body: {}
}

addCommentToPost(id, comment) {
    method: PUT
    route: /api/post/${id}/comment
    body: {
        comment: {
            author: String,
            content: String
        }
    }
}

upvotePost(id, userId) {
    method: PUT
    route: /api/post/${id}/upvote
    body: {
        userId: String
    }
}

downvotePost(id, userId) {
    method: PUT
    route: /api/post/${id}/downvote
    body: {
        userId: String
    }
}

upvoteMap(id, userId) {
    method: PUT
    route: /api/map/${id}/upvote
    body: {
        userId: String
    }
}

downvoteMap(id, userId) {
    method: PUT
    route: /api/map/${id}/downvote
    body: {
        userId: String
    }
}

upvoteComment(id, commentId, userId) {
    method: PUT
    route: /api/comment/${commentId}/upvote
    body: {
        userId: String
    }
}

upvoteComment(id, commentId, userId) {
    method: PUT
    route: /api/comment/${commentId}/upvote
    body: {
        userId: String
    }
}

downvoteComment(id, commentId, userId) {
    method: PUT
    route: /api/comment/${commentId}/downvote
    body: {
        userId: String
    }
}

updateMapFile(id, file) {
    method: PUT
    route: /api/map/${id}/file
    body: {
        file: {
            data: String
        }
    }
}

getUserById(id) {
    method: GET
    route: /api/user/${id}
    body: {}
}

updateUserById(id, user) {
    method: PUT
    route: /api/user/${id}
    body: {
        user: {
            username: String,
            email: String,
            pfp: String,
            bio: String,
            passwordHash: String
        }
    }
}
```

DeleteMapModal

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

// Delete map
const handleDeleteMap = () => { ...

// Close modal
const handleCloseModal = () => { ...
```

PublishMapModal

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

// Delete map
const handlePublishMap = () => { ...

// Close modal
const handleCloseModal = () => { ...
```

LandingPage

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

// Popular/recent maps fetched on load
const [popularMaps, setPopularMaps] = useState([]);
const [recentMaps, setRecentMaps] = useState([]);
```

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

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

ProfilePage

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

// Maps and Posts fetched on load
const [userMaps, setUserMaps] = useState([]);
const [userPosts, setUserPosts] = useState([]);
```

```
// Handle profile picture change
const handlePfpChange = () => { ...

// Handle profile description change
const handleDescriptionChange = (event) => { ...
```

Map

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

```
// Handles when the user clicks on the map
const handleClick = () => { ...
```

Post

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

// Comments fetched on load
const [comments, setComments] = useState([]);
```

```
// Handles when the user clicks on the post
const handleClick = () => { ...

// Handles when the user clicks on the poster
const handleUser = () => { ...
```

Comment

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

```
// Handles when the user clicks on the commenter
const handleClick = () => { ...

// Handles upvotes
const handleUpvote = () => { ...

// Handles downvotes
const handleDownvote = () => { ...
```

MapPage

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

// Comments fetched on load
const [comments, setComments] = useState([]);

// Map preview zoom status
const [zoomLevel, setZoomLevel] = useState(0);
```

```
// Response for when a keyboard key is pressed
const handleKeyPress = (event) { ...

// Handle clicking on the map creator
const handleUser = () => { ...

// Handle upvotes - if already voted, will remove it
const handleUpvote = () => { ...

// Handle downvotes - if already voted, will remove it
const handleDownvote = () => { ...

// Handle fork map
const handleFork = () => { ...

// Handle zoom in on map preview
const handleZoomIn = () => { ...

// Handle zoom out on map preview
const handleZoomOut = () => { ...

// Add a comment to the map
const handleAddComment = () => { ...
```

SearchResults

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

// Results
const [results, setResults] = useState([]);

// Filter setting
const [filterBy, setFilterBy] = useState('');

// Sort setting
const [sortBy, setSortBy] = useState('');
```

```
// Handle filter change
const handleFilter = () => { ...

// Handle sort change
const handleSort = () => { ...
```

PostPage

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

// Comments fetched on load
const [comments, setComments] = useState([]);
```

```
// Response for when a keyboard key is pressed
const handleKeyPress = (event) { ...

// Handle clicking on the post creator
const handleUser = () => { ...

// Handle upvotes - if already voted, will remove it
const handleUpvote = () => { ...

// Handle downvotes - if already voted, will remove it
const handleDownvote = () => { ...

// Add a comment to the map
const handleAddComment = () => { ...
```

HTTP Request

HTTP Response

Server

MapStudioRouter
/api

```
// Handles creating a new map
router.post('/api/map', auth.verify, MapController.createMap)

// Handles deleting a map
router.delete('/api/map/:id', auth.verify, MapController.deleteMapById)

// Handles getting a map by id
router.get('/api/map/:id', auth.verify, MapController.getMapById)

// Handles updating map info
router.put('/api/map/:id', auth.verify, MapController.updateMapInfoById)

// Handles adding a comment to a map
router.put('/api/map/:id/comment', auth.verify, MapController.addCommentToMap)

// Handles updating a comment
router.put('/api/comment/:id', auth.verify, MapController.updateCommentById)

// Handles deleting a comment
router.delete('/api/comment/:id', auth.verify, MapController.deleteCommentById)

// Handles creating a new post
router.post('/api/post', auth.verify, MapController.createPost)

// Handles getting a post by id
router.get('/api/post/:id', auth.verify, MapController.getPost)

// Handles adding comment to a post
router.put('/api/post/:id/comment', auth.verify, MapController.addCommentToPost)

// Handles getting the content of a comment by id
router.get('/api/comment/:id', auth.verify, MapController.getCommentById)

// Handles upvoting a post
router.put('/api/post/:id/upvote', auth.verify, MapController.upvotePost)

// Handles downvoting a post
router.put('/api/post/:id/downvote', auth.verify, MapController.downvotePost)

// Handles upvoting a map
router.put('/api/map/:id/upvote', auth.verify, MapController.upvoteMap)

// Handles downvoting a map
router.put('/api/map/:id/downvote', auth.verify, MapController.downvoteMap)

// Handles upvoting a comment
router.put('/api/comment/:id/upvote', auth.verify, MapController.upvoteComment)

// Handles downvoting a comment
router.put('/api/comment/:id/downvote', auth.verify, MapController.downvoteComment)

// Handles updating map file
router.put('/api/map/:id/file', auth.verify, MapController.updateMapFile)

// Handles getting user by id
router.get('/api/user/:id', auth.verify, MapController.getUserById)

// Handles updating user
router.put('/api/user/:id', auth.verify, MapController.updateUserById)
```

MapController

```
createMap(req, res) {
  method: POST
  route: /api/map
  response: { // Properly formatted legal request
    status: 201 (Created)
    data: {
      id: String,
      map: {
        title: String,
        description: String,
        author: String
      }
    }
  }
  response: { // Improperly formatted request
    status: 400 (Bad Request)
    data: { errorMessage: String }
  }
  response: { // Correct format but incorrect credentials
    status: 401 (Unauthorized)
    data: { errorMessage: String }
  }
}

deleteMapById(req, res) {
  method: DELETE
  route: /api/map/:id
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {}
  }
  // Same error handling as above
}

getMapById(req, res) {
  method: GET
  route: /api/map/:id
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {
      map: {
        author: String
        isPublished: Boolean
        title: String
        description: String
        likes: Number
        dislikes: Number
        comments: [String]
        mapFile: File
        creationDate: Date
        updateDate: Date
      }
    }
  }
  // Same error handling as above
}

updateMapInfoById(req, res) {
  method: PUT
  route: /api/map/:id
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: { id: String }
  }
  // Same error handling as above
}

addCommentToMap(req, res) {
  method: PUT
  route: /api/map/:id/comment
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: { id: String }
  }
  // Same error handling as above
}

updateCommentById(req, res) {
  method: PUT
  route: /api/comment/:id
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: { id: String }
  }
  // Same error handling as above
}

deleteCommentById(req, res) {
  method: DELETE
  route: /api/comment/:id
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {}
  }
  // Same error handling as above
}

getCommentById(req, res) {
  method: GET
  route: /api/comment/:id
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {
      comment: {
        author: String
        content: String
        likes: Number
        dislikes: Number
        publishedDate: Date
      }
    }
  }
  // Same error handling as above
}

createPost(req, res) {
  method: POST
  route: /api/post
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: { id: String }
  }
  // Same error handling as above
}

getPost(req, res) {
  method: GET
  route: /api/map/:id
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {
      post: {
        author: String
        title: String
        content: String
        likes: Number
        dislikes: Number
        comments: [String]
        publishedDate: Date
      }
    }
  }
  // Same error handling as above}

addCommentToPost(req, res) {
  method: PUT
  route: /api/post/:id/comment
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: { id: String }
  }
  // Same error handling as above
}

upvotePost(req, res) {
  method: PUT
  route: /api/post/:id/upvote
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {}
  }
  // Same error handling as above
}

downvotePost(req, res) {
  method: PUT
  route: /api/post/:id/downvote
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {}
  }
  // Same error handling as above
}

upvoteMap(req, res) {
  method: PUT
  route: /api/map/:id/upvote
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {}
  }
  // Same error handling as above
}

downvoteMap(req, res) {
  method: PUT
  route: /api/map/:id/downvote
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {}
  }
  // Same error handling as above
}

upvoteComment(req, res) {
  method: PUT
  route: /api/comment/:id/upvote
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {}
  }
  // Same error handling as above
}

downvoteComment(req, res) {
  method: PUT
  route: /api/comment/:id/downvote
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {}
  }
  // Same error handling as above
}

updateMapFile(req, res) {
  method: PUT
  route: /api/map/:id/file
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {}
  }
  // same error handling
}

getUserById(req, res) {
  method: GET
  route: /api/user/:id
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: {
      user: {
        username: String
        email: String
        pfp: String
        bio: String
        passwordHash: String
        maps: [String]
        posts: [String]
        joinDate: Date
        likedMaps: [String]
        dislikedMaps: [String]
        likedComments: [String]
        dislikedComments: [String]
        likedPosts: [String]
        dislikedPosts: [String]
      }
    }
  }
  // Same error handling as above
}

updateUserById(req, res) {
  method: PUT
  route: /api/user/:id
  response: { // Properly formatted legal request
    status: 200 (OK)
    data: { id: String }
  }
  // Same error handling as above
}
```

Map

```
author: ObjectId [User]
isPublished: Boolean
title: String
description: String
likes: Number
dislikes: Number
comments: ObjectId List [Comment]
mapFile: Buffer
creationDate: Date
updateDate: Date
```

Comment

```
author: ObjectId [User]
content: String
likes: Number
dislikes: Number
publishedDate: Date
```

Post

```
author: ObjectId [User]
title: String
content: String
likes: Number
dislikes: Number
comments: ObjectId List [Comment]
publishedDate: Date
```

User

```
username: String
email: String
pfp: String
bio: String
passwordHash: String
maps: ObjectId List [Map]
posts: ObjectId List [Post]
joinDate: Date
likedMaps: ObjectId List [Map]
dislikedMaps: ObjectId List [Map]
likedComments: ObjectId List [Comment]
dislikedComments: ObjectId List [Comment]
likedPosts: ObjectId List [Post]
dislikedPosts: ObjectId List [Post]
```

Our Express Server

```
databaseManager = require('MongooseManager');
```

GlobalStoreController

```
const databaseManager = require( DatabaseManager ...

createMap(req, res) {
    // ( /* IMPROPERLY FORMATTED REQUEST */)
      // SEND 400 RESPONSE CODE
    // ( /* UNAUTHORIZED REQUEST */)
      // SEND 401 RESPONSE CODE

    // GOOD REQUEST

    let newMap = databaseManager.createMap(...
    if (newMap) {
      // SEND 201 response status CODE AND data
    }
    else {
      // SEND 400 response status CODE
    }
}
...
```

DatabaseManager
<generalized interface>

createMap(name, desc, email):
...
(all subsequent function templates, see MongooseManager for implementations)

MongooseManager
<implements DatabaseManager>

```
createMap(title, desc, author) {
    const newMap = new Maps(title, ...
    newMap.save()
    if (success)
        return newMap
    else
        return null
}

getMapById(id) {
    const map = Maps.findById(...
    if (err)
        return null
    else
        return map
}

updateMapById(id, record) {
    const map = Maps.findAndModify(...
    if (err)
        return null
    else
        return map
}

deleteMapById(id) {
    const map = Maps.deleteOne(...)
    if (err)
        return null
    else
        return map
}
...
(same CRUD operations for each Schema)
```

Maps
<using Mongoose ODM>

```
const MapsSchema = new Schema({
    author: { type: ObjectId, ref: 'User', required: true },
    isPublished: { type: Boolean, required: true },
    title: { type: String, required: true },
    description: { type: String, default: '' },
    likes: { type: Number, required: true },
    dislikes: { type: Number, required: true },
    comments: [{ type: ObjectId, ref: 'Comment' }],
    mapFile: { type: Buffer, required: true },
    creationDate: { type: Date, required: true, default: Date.now },
    updateDate: { type: Date, required: true, default: Date.now }
});
module.exports = mongoose.model('Maps', MapsSchema)
```

Users
<using Mongoose ODM>

```
const UsersSchema = new Schema({
    username: { type: String, required: true },
    email: { type: String, required: true },
    pfp: { type: String }, // file path
    bio: { type: String, default: '' },
    passwordHash: { type: String, required: true },
    maps: [{ type: ObjectId, ref: 'Map' }],
    posts: [{ type: ObjectId, ref: 'Post' }],
    joinDate: { type: Date, default: Date.now },
    likedMaps: [{ type: ObjectId, ref: 'Map' }] ,
    dislikedMaps: [{ type: ObjectId, ref: 'Map' }],
    likedComments: [{ type: ObjectId, ref: 'Comment' }],
    dislikedComments: [{ type: ObjectId, ref: 'Comment' }],
    likedPosts: [{ type: ObjectId, ref: 'Post' }],
    dislikedPosts: [{ type: ObjectId, ref: 'Post' }],
});
module.exports = mongoose.model('Users', UsersSchema)
```

Comments
<using Mongoose ODM>

```
const CommentsSchema = new Schema({
    author: { type: ObjectId, ref: 'User', required: true },
    content: { type: String, required: true },
    likes: { type: Number, required: true },
    dislikes: { type: Number, required: true },
    publishedDate: { type: Date, required: true }
});
module.exports = mongoose.model('Comments', CommentsSchema)
```

Posts
<using Mongoose ODM>

```
const PostsSchema = new Schema({
    author: { type: ObjectId, ref: 'User', required: true },
    title: { type: String, required: true },
    content: { type: String, required: true },
    likes: { type: Number, required: true },
    dislikes: { type: Number, required: true },
    comments: [{ type: ObjectId, ref: 'Comment' }],
    publishedDate: { type: Date, required: true }
});
module.exports = mongoose.model('Posts', PostsSchema)
```

MapStudio - Auth Management Design Diagram

Text

MapStudio Team | October, 2023

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** = () => { ...

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

// Responds to Browse top bar tab
const **handleBrowse** = () => { ...

// Responds to Create top bar tab
const **handleCreate** = () => { ...

// Responds to Discuss top bar tab
const **handleDiscuss** = () => { ...

// Responds to Search box query change
const **handleSearchChange** = (event) => { ...

// Responds to Search
const **handleSearch** = () => { ...

LoginScreen

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

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

// Responds to Forgot Password menu item
const **handleForgot** = () => { ...

RecoveryScreen

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

// Responds to button click to submit forget account's email and username
const **handleSubmit** = (event) => { ...

RegisterScreen

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

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

ChangePasswordScreen

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

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

AuthActionType

GET_LOGGED_IN
LOGIN_USER
LOGOUT_USER
REGISTER_USER
FORGOT_ACCOUNT
CHANGE_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) => ...

// Recover the forgot account
auth.**forgotAccount** = async(userData) =>...

// Change the password
auth.**changePassword** = async (userData) => {

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

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

forgotAccount(email) {
    method: 'POST',
    route: **/auth/forgetAccount**
    data: {
        email: String,
    }
}

changePassword(currentPassword,
newPassword) {
    method: 'POST',
    route: **/auth/changePassword**
    data: {
        currentPassword: String,
        newPassword: 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 account recovery requests
router.post('/**forgetAccount**',AuthController.recoverAccount)

// Handles user password change requests
router.post('/**changePassword**',
AuthController.changePassword)

AuthController

loggedIn(req, res) {
    method : GET
    route : /auth/loggedIn
    response : {
        // properly formatted request
        status: **200** (Ok)
        data : {
            loggedIn: (true or false)
            user: {
                username: 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: {
                username: String,
                email : String
            }
        }
    }
    response {
        // improperly formatted request
        status: **400** (Bad Request)
        data : { errorMessage : String}
    }
    response {
        // properly formatted but incorrect credentials
        status: **401** (Unauthorized)
        data : { errorMessage : String }
    }
}

recoverAccount(req, res) {
    method: 'POST'
    route: '/auth/forgetAccount'
    response: {
        // Account recovery email sent successfully
        status: **200** (Ok)
        data: {
            message: "Recovery email sent, please check your inbox."
        }
    }
    response: {
        // User email not found or other server error
        status: **400** (Bad Request)
        data: { errorMessage: "Error sending recovery email, please try again." }
    }
}
changePassword(req, res) {
    method: 'POST'
    route: '/auth/changePassword'
    response: {
        // Password changed successfully
        status: **200** (Ok)
        data: {
            message: "Password changed successfully."
        }
    }
    response: {
        // Old password did not match or bad request data
        status: **400** (Bad Request)
        data: { errorMessage: "Old password does not match or bad request data." }
    }
    response: {
        // Properly formatted but unauthorized to change password
        status: **401** (Unauthorized)
        data: { errorMessage: "Unauthorized to change password." }
    }
}

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: {
                username: String,
                email : String
            }
        }
    }
    response : {
        // improperly formatted request or bad data
        status: **400** (Bad Request)
        data : { errorMessage: String }
    }
}

UserSchema

username : String
email : String
passwordHash : String
maps : [ObjectId]
pfp: String
bio: String
posts: [ObjectId]
joinDate: Date
likedMaps: [ObjectId]
dislikedMaps: [ObjectId]
likedComments: [ObjectId]
dislikedComments: [ObjectId]
likedPosts: [ObjectId]
dislikedPosts: [ObjectId]

AuthManager

// Signs a token for authentication using user details
signToken(userId) => { ...
    returns: {token: String}
}

// Verifies the incoming request for valid authentication token
verifyRequest(req, res, next) => { ...

}

// Verifies if the request is made by an authenticated user
verifyUser(req) => { ...

}

Progress Reviews

Progress Review 1 (11/7/23 4:00-4:30 PM)
Attendees: All

- Showcased our deployed barebones CRUD application which involved adding a person and their age to a database. Through the front-end, anyone was able to add, modify, and delete individual people in the database. Showed the code for the barebones application and explained it.
- Presented the portals for Firebase, where our front-end was being hosted, and Azure, where our back-end was being hosted, to show that our deployment was up and running.
- Displayed the GitHub repository that we are using for version control, and added TA Prakash Upadhyay to the repository.
- Showed the code for the front-end and back-end tests written for the barebones application.
- Reran GitHub actions to show that CI/CD and tests were working properly.
- Presented and shared the Monday build schedule with TA Prakash Upadhyay.

Progress Review 2 (11/14/23 4:00-4:20 PM)
Attendees: All

- Explained the purpose and goals of the MapStudio web application.
- Showcased all of our front-end views with hard-coded data.
- Explained our code to demonstrate how we were implementing screen switching and how the hierarchy of React components was laid out.
- Reran GitHub actions to show that CI/CD and tests were working properly.
- Shared our project documents with TA Prakash Upadhyay for reference.

Progress Review 3 (11/21/23 4:00-4:15 PM)
Attendees: All

- Showcased account creation and login, as well as password recovery with an emailed temporary token. Also displayed screens that were different depending on whether the user was logged in or not.
- Showed profile biography editing and profile picture changing, and noted there were a few bugs that we would fix by the next progress review.
- Showed and explained the backend code written for CRUD operations on a map schema, including routes and necessary controller functions.
- Reran GitHub actions to show that CI/CD and tests were working properly.

Progress Review 4 (11/28/23 4:00-4:15 PM)
Attendees: All

- Showcased the entirety of the community section of the application. This included community post creation, viewing, commenting, liking/disliking, searching, and sharing.
- Showcased the map view screen, which now allows community actions like liking/disliking and commenting, as well as loads the relevant map metadata like title, author, and description.
- Presented additions to the profile screen which include a user's most recently created maps and authored discussion posts
- Reran GitHub actions to show that CI/CD and tests were working properly.

Progress Review 5 (12/7/23 4:00-4:15 PM) - Originally scheduled for 12/5/23 -Rescheduled by TA Prakash Upadhyay
Attendees: All

- Showcased the uploading (including both the map geometry and the map schema), deleting, and publishing of maps from the map edit screen
- Walked through the process of forking a map from the map view screen. Also showed the generation and downloading of JPG/PNG files of the map and the exporting of the map geometry and map schema from the map view screen.
- Performed subdivision editing, which involved changing the color, weight, and properties of the subdivision and updating the schema of the map to reflect these changes in real-time. Also showed that the map metadata (name and description) were editable as well.
- Showed the satellite view toggle functionality was working
- Reran GitHub actions to show that CI/CD and tests were working properly.

Post Mortem

Challenges and Issues Faced

Many of the most challenging issues that the team faced involved the use of technologies that were new to us. In particular, this was the first time any team member had hosted a public, full-stack web application on the cloud. This created an immense learning curve at the beginning of project development in which we had to do a great amount of research into which services fit our needs the best. After we chose to host the front-end on Firebase and the back-end on Azure, we also had to figure out how to configure the services to fit our requirements for testing and CI/CD. There were also times in which issues in application development were related to these services, for example when certain settings in Azure were causing CORS errors in the application. In the later stages of development, we encountered many challenges regarding map editing, particularly getting the map graphic to render correctly.

What Was Done Well

The build schedule was laid out in a clear, concise, and logical manner such that each team member knew their responsibilities and goals for the week. Each general action item was accompanied by their corresponding use case numbers, as outlined in the Use Case Modeling section of the document, leading to a well-defined, finished build for each progress meeting. The parallelized nature of the build schedule helped reduce the lag time in which team members had to wait for others to finish their tasks for the week. The structure of the application was well-planned and did not produce any major problems in implementation.

What Could Have Been Improved

During the planning phase of the application, the schedule was set up such that the development of the core functionalities of the application, that being the editing of map graphics, was pushed to the end of the timeline. This made it such that we were unable to dedicate as much time as we would have liked to add additional features, quality-of-life improvements, and bug fixes to what is the most important part of the application.

As the project progressed, the need for functional code started to outweigh the need for clean, organized code. The time constraints of the project caused us to write messy/hacky, but functional code rather than code that was both clean and organized as well as functional. As this project was relatively short and straightforward, with the actual build lasting a total of 7 weeks, the technical debt accumulated from their poor practices did not boil over into something greater. However, given more time to work on the project, many parts of the code would need to be refactored to restore organization and good programming practices.

Communication between members could have been improved. Although communication was solid between team members, there were some points in which bug fixing and feature development were hindered by a lack of communication and thus collaboration. Most of the work on the application by team members was done remotely from each other and most communication was done via text communication. It would have been greatly beneficial to dedicate more time to in-person development sessions or communication via voice chat instead of via text.

Lessons for Future Software Projects

The major takeaways from the development of the MapStudio web application were that:

- Core parts of the application should have been started earlier.
- More frequent meetings would have helped reduce conflicts.
- More detailed initial planning of implementation could have reduced nonoptimal code.

Appendix A: All Meeting Minutes

Meeting 1 (08/31/23 2:30 pm)
Attendees: All

Topic:

- Introductions
- Discuss schedules
- Choose meeting times

Takeaways:

- All members are available during Tues/Thurs after class

– No major meetings in between –

Meeting 2 (10/31/23 3:30 pm)
Attendees: All

Topic:

- Set up workflow
- Version control
- Map out build schedule (Monday)
- Frontend services to use (Heroku, **Firebase**, Netlify, Glitch)
- Backend services to use (**MongoDB**, **Azure**, Vercel, AWS EC2)

Takeaways:

- GitHub set up, all members have access
- GitHub organization plan established (1 branch/member, merge to main)
- Monday set up with the build schedule
- Tasks distributed on Monday

Meeting 3 (11/07/23 3:30 pm)
Attendees: All

Topic:

- Discuss previous week
- Layout of our application
- MaterialUI

Takeaways:

- Application layout established
- Need to hard code all rendering
- Will use MaterialUI for rendering
- Will use Firebase for frontend hosting

Meeting 4 (11/14/23 3:30 pm)
Attendees: All

Topic:

- Discuss previous week
- Presentation next week

Takeaways:

- Create Google Slides for presentation
- Outline presentation topics
- Decide who is presenting what
- Using MongoDB + Azure for backend services

Meeting 5 (11/21/23 3:30 pm)
Attendees: All

Topic:

- Discuss previous week
- Discussion boards
- Community actions
- How will maps be shown to the user

Takeaways:

- Details of discussion board & posts laid out

Meeting 6 (11/28/23 3:30 pm)
Attendees: All

Topic:

- Discuss previous week
- Map uploading and data storage
- How are we storing map data?

- How are we storing edit data?
- Exporting maps as images
- Exporting maps as JSON

Takeaways:

- Separate geometry and edit data
- Store maps in Azure, link to Azure files in MongoDB
-

Meeting 7 (12/05/23 4:00 pm)
Attendees: All

Topic:

- Discuss previous week
- How to resolve conflicts in editing
- How each type of editing will work

Takeaways:

- Take the most recent edit for conflicts
- Finalized plans for remaining editing types

Meeting 8 (12/15/23)
Attendees: All

Topic:

- Discuss previous week
- Discuss remaining work
- Discuss presentation
- Existing bugs

Takeaways:

- Need to add a legend
- List of all remaining bugs to be fixed and remaining small features to implement, sorted by priority
- Plans for final presentation

Appendix B: Requirements Presentation Slides

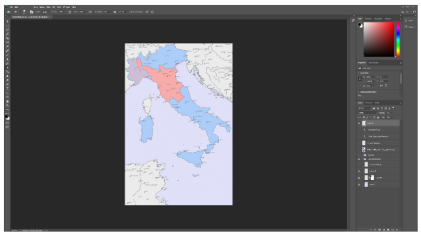
MapStudi

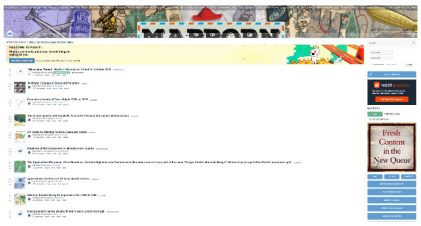
By Razzmatazz

Preston Chang, James Leonardi, Wei Qi, Kevin Tao

The Problem

- Map creation and the map community are two separate worlds
- Map creation software (GIMP, Photoshop, ArcGis, Canva, MapChart)
- Map community spaces (forums, Reddit)
- No easy bridge between



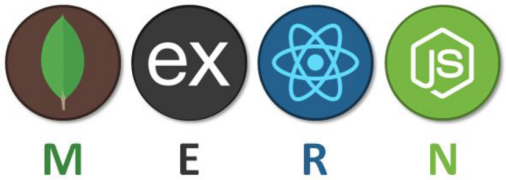


Our Solution

- A MERN stack web application that aims to combine both creation and community

- Easily create, edit, and share maps

- Comment on and discuss maps and cartography topics



MapStudio

Objectives - Map Creation

- Create easily shareable custom map graphics

- Support upload of Shapefiles, GeoJSON, KML formats

- Naming regions, attaching custom data properties to regions, lots of freedom in designing map graphics

Objectives - Community Interactions

- Find, share, fork, and edit maps created by others

- Export maps as custom JSONs, PNGs, or JPGs

- Easily search and filter published maps

- Comment on maps and discuss cartography topics on a forum

Objectives - Design Philosophy

- UI/UX - easy to navigate, quick, responsive, consistent and united design

- Simplicity - application should be simple as to reduce user confusion and choice overload

- Security - secure accounts and members will be able to recover forgotten passwords

Guests vs Members: Guests

- Guests can only view maps
- No write, only read
- Other actions will prompt a login
- Able to log in, register, recover account
- No community privileges
- Guests can become members by registering and logging in

Guests vs Members: Members

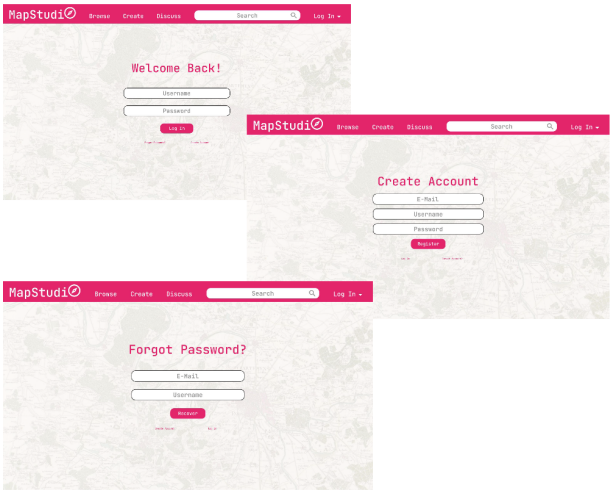
- Full access to view, edit, save, publish maps
- Community privileges such as liking/disliking, commenting, forking other members' projects
- Able to participate in forum discussions

Guest Landing Page

- Large greeting to web application
- Prompt to view the most popular maps
- Suggests the guest to register account
- Existing members can easily click the “Log In” button
- Create/Discuss buttons will redirect to login for guests

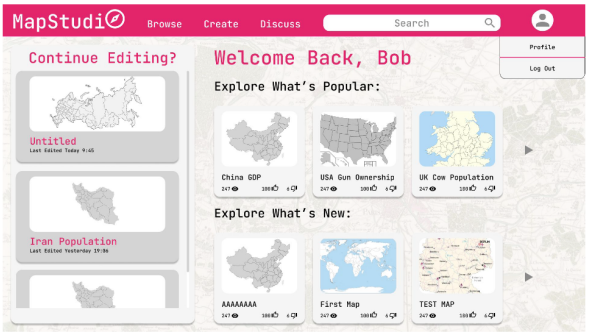
Log In/Create Account/Forgot Password

- Recover account by providing email and username used to register
- Will be emailed a code to change account password
- Able to easily switch screens easily



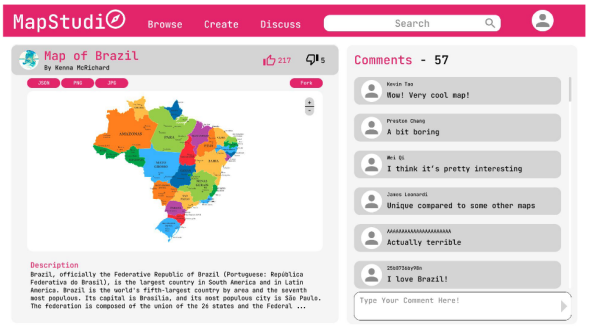
Member Landing Page

- Different from the guest landing page
- “Continue Editing?” sidebar allows easy access to most recently worked on maps
- Still suggested popular and new maps to view
- Create/Discuss buttons will work



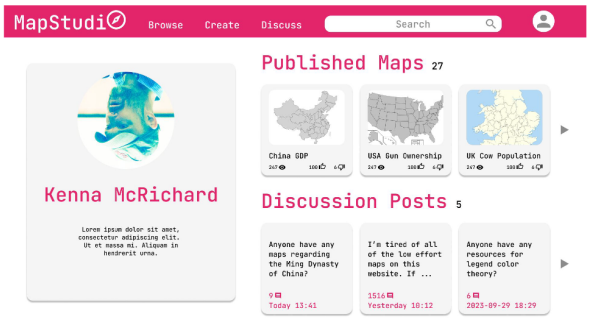
Map View

- Able to view and zoom in/out map
- Like/dislike, download as different file formats, and fork map for editing
- Comment on map to share thoughts and opinions
- Click on username to see profile



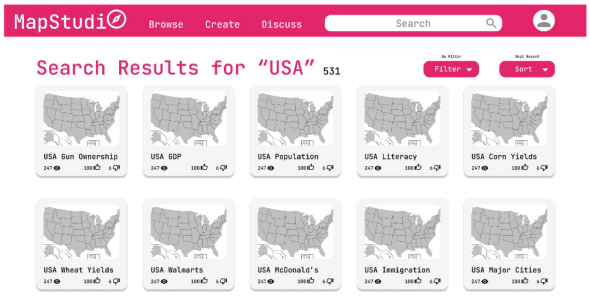
Profile Screen

- Username, profile picture, profile descriptions
- On a member's own profile, edit by clicking on picture or profile description
- Can see and navigate to all maps and discussion posts that a user has created



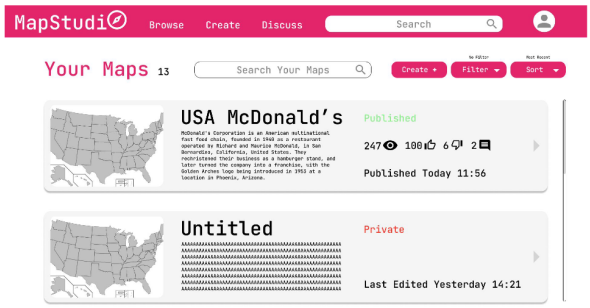
Map Search

- Type a query into the search bar in the menu bar
- Results will appear than can be sorted (ex. By newest) and filtered (ex. By map type)
- Clicking "browse" will result in this screen will default search parameters



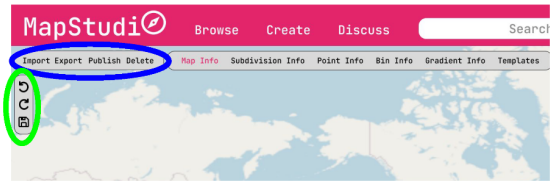
Personal Maps

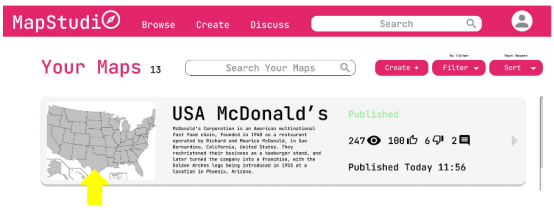
- Able to view one's own maps
- Can search, sort, and filter like regular map search
- Can edit private maps and can view published maps



Map Editing- Common Operations

- Import Shapefile, GeoJSON, or KML files as a starting point
- Export maps as custom JSON files or as PNG/JPG
- Publish map to make public
- Delete map
- Zoom in/zoom out
- Undo/redo
- Save draft





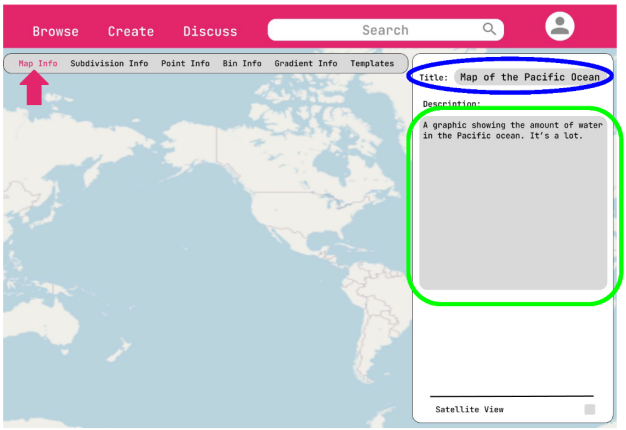
Click map cards into edit screen

Map Editing - Edit Map Info

- Click “Map Info” to open info edit sidebar

- Rename map by typing new name into title box

- Write map description in the description text box



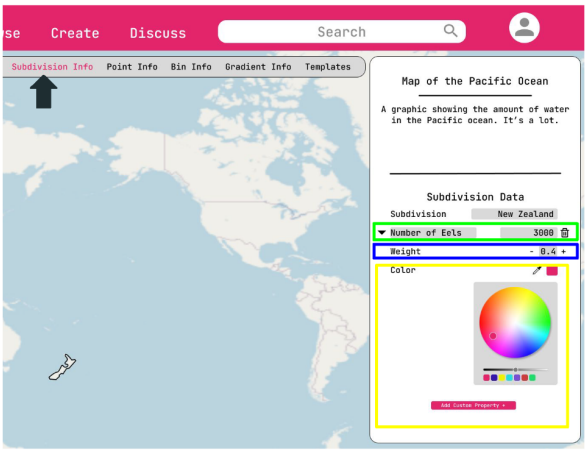
Map Editing - Edit Subdivision Info

- Click “Subdivision Info” to open the info edit sidebar

- Add custom properties

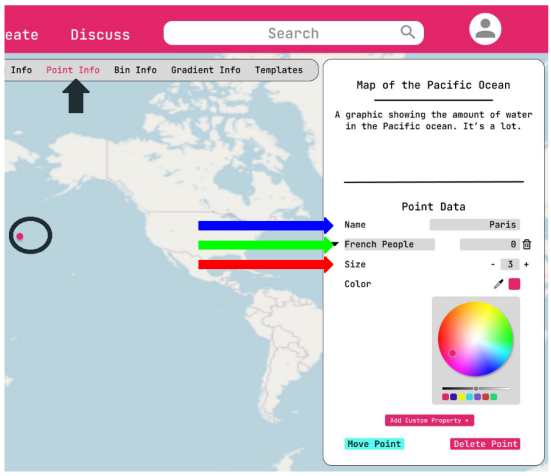
- Edit property weight by clicking “-”/“+” or just entering a value

- Change color by using color picker or entering color codes like RGB or Hex



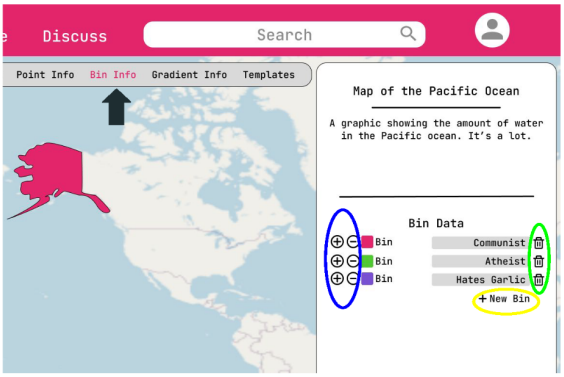
Map Editing - Edit Point Info

- While the point edit sidebar is open, click anywhere to add a point
- Click a point to open the edit sidebar
- Edit name by typing in name text box (blue arrow)
- Edit data by selecting data category and typing in (green arrow)
- Edit Weight/size by clicking “-”/“+” or just entering (red arrow)
- Change color by using color picker or entering color codes like RGB or Hex



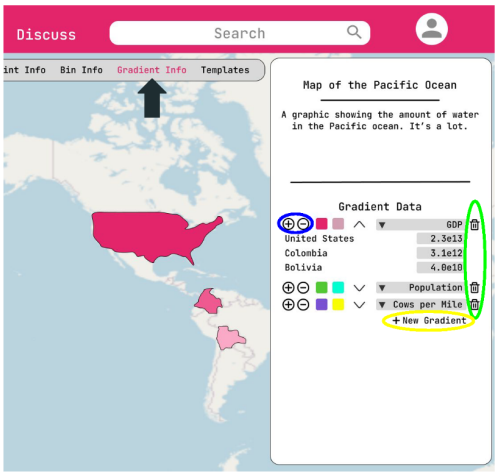
Map Editing - Edit Bin Info

- Click “Bin Info” to open edit sidebar
- Add new bin by click “+New Bin”
- Delete bin by click “trash can” icon
- Edit name just typing in the name text box
- Edit color by click color square to open color picker or entering color codes like, RGB or Hex
- Add/Remove an area to bin by selecting area then click ”+”/”-” buttons of corresponding bin to finish



Map Editing - Edit Gradient

- Click “Gradient Info”to open edit sidebar
- Add new gradient by click “+New Gradient”
- Delete gradient by click “trash can” icon
- Edit name selecting from a dropdown of the available properties in each subdivision.
- Edit color by click color square to open color picker or entering color codes like, RGB or Hex
- Add/Remove an area to gradient by click ”+”/”-” buttons first then click the target areas to finish

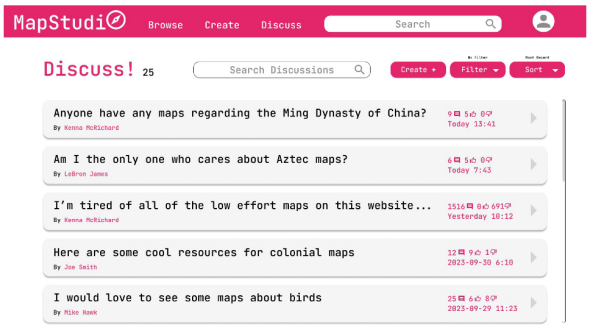


Map Editing - Edit Template

- Click “Template” to open template sidebar
- Choose a template as a starting point for the desired map

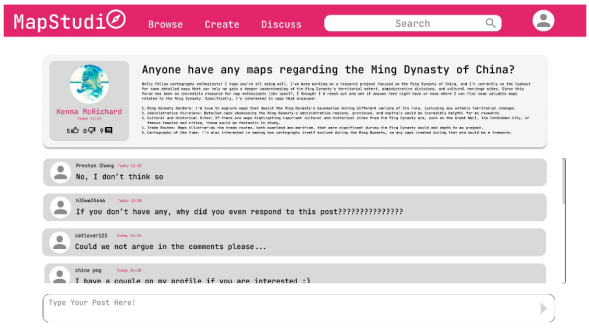
Discussion Home

- Click on “Discuss”
- Able to search, sort, and filter like map search
- Default sorted by most recent
- See number of comments, likes/dislikes, publish date



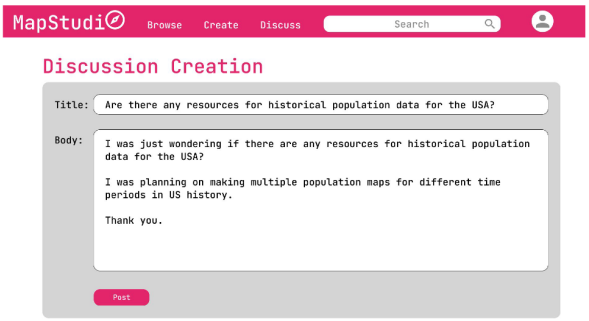
View Discussion

- Able to read post
- Like/dislike
- Comment on post to share thoughts and opinions
- Click on username to see profile



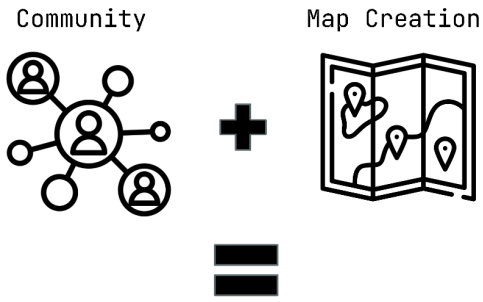
Create Thread

- Fill out title and body fields
- Useful for discussing topics not tied to any one map
- Foster a sense of community and constructive discussion



Conclusion

- MapStudio bundles both map creation and community interactions into a single web application
- Easy-to-use, but powerful map graphics editor to appeal to beginners and experts
- Robust community features to build a sense of collaboration and camaraderie within user base



MapStudi

Appendix C: Final Presentation Slides

N/A

Appendix D: Detailed Descriptions of Team Member Contributions

Preston Chang - Developed the entirety of the front-end and back-end of the Week 1 bare-bones application. Designed the UI for most of the front-end views, excluding the map editing screen. Implemented all of the front-end views, excluding the map editing screen. Implemented the authentication features of registering for an account and logging into an account. Fixed issues involving CORS errors during our front-end, hosted on Firebase, communicating with our back-end, hosted on Azure. Implemented the entirety of the community section which involves creating community posts, reading community posts, commenting on community posts, liking/disliking community posts, and searching/sorting community posts. Implemented the loading of community post cards on a user's profile. Implemented the publishing of maps. Implemented the loading of map results on the landing page and search screen as well as the sorting and filtering of those maps. Implemented the forking of already published maps. Expanded the profile functionality so that users can view the profiles of others. Implemented the map thumbnails and worked on map view rendering. Created the list views for point editing and subdivision editing in the map editing screen. Continuously fixed small bugs throughout all parts of the application.

James Leonardi - Set up the Monday schedule. Set up the GitHub repo. Worked on many back-end routes for map operations. Implemented front-end for creating maps, listing a user's maps (including sorting and filtering), and the same to the user profile. Implemented map deletion. Added confirmation modals for deletion and reset through templates. Added snackbars to provide feedback for various errors and warnings. Did extensive bugfixes, refactoring, and other miscellaneous features, including favicon, create tab prompting user login, text box hints, hiding passwords, etc. Fixed problematic tests.

Kevin Tao - Helped set up the Monday. Set up all GitHub Actions workflows for automated testing and deployment. Set up the majority of tests for both front-end and back-end. Set up deployments on Firebase, Azure, and MongoDB Atlas. Laid out build schedule. Set up the forgotten password system and the profile screen. Set up Azure Blob Storage for images and GeoJSON data. Added comments to the map view. Did work on importing JSON schemas and uploading map geometry to cloud storage. Implemented map editing view, all map rendering, map legend, map information editing (map title, map description), satellite view, subdivision editing (adding/editing/deleting properties, weight, and color), point editing (adding, moving, deleting, renaming points, size, and color), bin editing (creating/renaming/deleting bins, changing color, adding/removing subdivisions), gradient editing (creating/deleting/refreshing gradients, changing colors, changing which property it affects, adding/removing subdivisions, and generating gradient color scale), and templates (resetting the geometry/schema to presets). Implemented the transaction system for undo/redo/save with keyboard shortcuts. Modified the subdivision list to allow editing of every property. Added snackbars for tooltips. Other miscellaneous bug fixes.

Wei Qi - Implemented some back-end functions for maps. Implemented importing map geometry files in GeoJSON/KML/Shapefile formats. Wrote test cases for map editing for importing, exporting, publishing, deleting, and editing info. Added basic template front-end. Implemented heat maps, including rendering/parsing imported csv files and customizing heat radius/blur.