Pieces

Systems Analysis

Executive Summary:

Pieces is a web application developed for artists, level designers, and game designers for the purposes of creating tilesets, and tile maps. It also facilitates connectivity amongst the game community by allowing collaboration on projects and a public marketplace to browse and potentially use others' work. Our goal is to create a web community that not only lets users create and share their maps and tilesets but also promotes interaction between fellow designers with features to offer feedback to publicly shared creations by liking, disliking, and commenting . Some of the competitors who offer similar services are *Tilesetter* and *Tiled*, which are tools that let users make tilesets and maps. However, these tools don't offer an integrated online platform to share and collaborate on their maps. We also aim to make our product more intuitive and satisfying to use than software like *Tiled* and *Tilesetter*, pushing the envelope forward and making it the new optimal choice for artists, level designers, and game designers.

Objectives:

- A tileset uploader.
- A tileset editor. (with option to collaborate).
- A tile map editor (with option to collaborate).
- A map exporter.
- Online accounts for saving & sharing of maps and tilesets.
    - Private share (share with link) and public share (publicly available) options.
- A list of publicly available maps that users can contribute to and offer feedback.
    - A rating system for publicly shared maps (Likes, dislikes, comments).
    - A search functionality to filter and sort maps.
- Friends list system, chat system.
- A public forum/discussion board.
- Report system.

Strategies/Philosophies:

Guiding Principles:

- Intuitive, simple, easy-to-use design elements and design choices. Minimalism.
    - Proper documentation (hover text, a tips page, etc.)
    - Congregated public maps, public users, search functionality.
- Accounts:
    - All account features/settings will be located in account settings, which will be a dropdown on the top right, accessible at all times. We can try to use a modal for less involved user features (website theme, zoom, etc.) to make it look more seamless.

- Promote collaboration:
    - Users can set “collaborator/solo” preference on their accounts.
    - Public maps can be set to either “view only” or “view & edit”.
    - Browse through users and/or maps open for collaborations.

Constraints:

- Maps
    - Exporting and importing maps will be done through only one map type and will not accommodate for different programs. We plan on using *Tiled's* mapping for our import system.
- Tilesets
    - Exporting and importing tilesets will be done through only one tileset type and will not accommodate for other programs. We plan on using *Tilesetter's* tileset format for our import system.

Actors:

- Admins
    - Have control of banning users or deleting maps if deemed inappropriate.
    - Have view access to maps and tilesets.
- Map Creator (Level Designer, Game Designer)
    - Have control of making and contributing to maps.
    - Have export access to maps.
    - Have access to use tilesets.
- Tileset Artist (Artist)
    - Have control of making and contributing to tilesets.
    - Have export access to tilesets.
    - Have view access to public maps.
- All of the Above
    - Have access to the rating/comment system.
    - Have access to user profiles, friends list, chat, and the report system.

Services:

- Create, Login, Logout & Update Account
- Create, Upload, Edit & Delete Tiles
- Create, Edit & Delete Maps
- Undo Redo editing
- Export Maps
- Share Maps
- Search for public maps and “Collaborator” users
- Like, Dislike & Comment on Maps
- Add friends
- Post on discussion forum

Use Case Model

Use Case Listing

Use Case #	UI Context	Use Case Name
1.1	Welcome Screen	Create Account
1.2	Welcome Screen	Cancel Account Creation
1.3	Welcome Screen	Login
1.4	Welcome Screen	Cancel Login
1.5	Any Screen	Logout
2.1	Profile Screen	Update Account
2.2	Profile Screen	Delete Account
3.1	Projects Screen	View My Tilesets
3.2	Projects Screen	View My Maps
3.3	Projects Screen	View My Projects
3.4	Projects Screen	Select project
3.5	Projects Screen & Dashboard	Sort projects
3.6	Projects Screen & Dashboard	Filter projects
3.7	Projects Screen & Dashboard	Search projects
4.1	Map Manager	Create new map
4.2	Map Manager	Delete map
4.3	Map Manager	Edit map properties
4.4	Map Manager	Edit/draw map
4.5	Map Manager	Export map
5.1	Tile Manager	Create new tileset

5.2	Tile Manager	Edit tileset
5.2a	Tile Manager	Upload tile
5.2b	Tile Manager	Create new tile
5.2c	Tile Manager	Edit tile
5.2d	Tile Manager	Delete tile
5.3	Tile Manager	Edit tileset properties
5.4	Tile Manager	Delete tileset
5.5	Tile Manager	Import tileset
5.6	Tile Manager	Export tileset
6.1	Tile & Map Manager	Undo
6.2	Tile & Map Manager	Redo
6.3	Tile & Map Manager	View Collaborators
6.4	Tile & Map Manager	Manage Access
6.5	Tile & Map Manager	Edit roles
7.1	Social Sidebar	Add Friend
7.2	Social Sidebar	Remove Friend
7.3	Social Sidebar	Invite to Collaborate
7.4	Social Sidebar	Check Notifications
7.5	Forum Screen	Add Forum Thread
7.6	Forum Screen	Delete (Own) Forum Thread
7.7	Forum Screen	Add Comment
7.8	Forum Screen	Delete (Own) Comment
8.1	Dashboard (List Mode)	View/select project
8.2	Dashboard (View Mode)	Rate public project
8.3	Dashboard (View Mode)	Comment on project

8.4	Dashboard (View/List Mode)	Favorite public project
8.5	Dashboard (View/List Mode)	Download public project

Use Case Descriptions:
Use Case 1.1: Create Account

Number:	1.1
Name:	Create Account
Story:	User is viewing the Welcome screen and would like to make 2D game maps, so clicks on the “Create Account” button. A modal is displayed where the user then enters their email address, username, password, first and last name in the appropriate text fields. User clicks the Register button. If successful, a user account is created and added to the database. User is forwarded to the Projects Screen, which would not yet have any Tiles and/or maps to view/edit.
Scenario:	User is viewing the Welcome Screen and clicks on the “Create Account” button, which pulls up a modal. User enters john.smith@gmail.com in the email textfield. User enters jSmith123 in the username text field. User enters password123 in the password text field which shows up as **********3. User enters John and Smith in the First and Last name text fields respectively. User clicks the Register button. User is forwarded to the Projects Screen, which would not yet have any Tiles and/or maps to view/edit.

Use Case 1.2: Cancel Account Creation

Number:	1.2
Name:	Cancel Account Creation
Story:	User is viewing the Welcome screen and would like to make 2D game maps, so clicks on the “Create Account” button. The user then starts to enter data in the required text fields in the modal but then decides against making a new account and so presses the “Cancel” button. The modal disappears and the user remains on the Welcome Screen.
Scenario:	User John Smith clicks on the “Create Account” button on the Welcome Screen. This brings up the Create Account modal where he starts entering data in the text fields. John then realizes he already has an account and so simply decides to Cancel this process by clicking on the “Cancel” button.

Use Case 1.3: Login

Number:	1.3
Name:	Login
Story:	User is viewing the Welcome Screen and wants to access the maps they have recently created. User clicks on the “Login” button and is displayed with a modal. User enters their email or their username in the email/username textfield. User enters their password in the password textfield. User clicks the “Login” button. If successful, the user is redirected to their Projects Screen to select a project to work on. If not successful, a warning appears stating that the given information is not valid and the user is prompted to enter the information again.
Scenario:	User is viewing the Welcome Screen and wishes to access the maps they have recently created. User clicks on the “Login” button and a modal pops up. User enters john.smith@gmail.com in the email/username textfield. User enters password123 in the password textfield which shows up as **********3. User clicks the “Login” button. If the user is registered with the same email/username and password, the user is taken to the Projects Screen. If not, a warning appears stating that the given information is not valid and the user is taken back to the Login Screen to try to login again.

Use Case 1.4: Cancel Login

Number:	1.4
Name:	Cancel Login
Story:	User is viewing the Welcome screen and wants to access the maps they have recently created, so clicks on the “Login” button. The user then starts to enter data in the required text fields in the modal but then decides against doing so, perhaps because they have forgotten their account information, and so presses the “Cancel” button.
Scenario:	A user, John Smith clicks on the “Login” button on the Welcome Screen. This brings up the Login modal where he starts entering data in the text fields. John then realizes he does not want to work on editing his maps right now and so simply decides to Cancel this process by clicking on the “Cancel” button.

Use Case 1.5: Logout

Number:	1.5
Name:	Logout
Story:	User is viewing any screen. User clicks on the avatar icon in the top right. A dropdown appears and user clicks ‘Log Out’. If user was currently working on a tile, tileset, or a map, changes are saved and user is logged out. Otherwise, user is just logged out.
Scenario:	User is currently working on a tileset. User clicks on user icon in the top right and clicks Log Out. The current state of the tileset is saved and the user is logged out.

Use Case 2.1: Update Account

Number:	2.1
Name:	Update Account
Story:	User is logged into their account. From any screen, the user clicks on the avatar icon in the top right. A dropdown appears and user clicks “Update Account”. User is forwarded to the Profile Screen where they can see their username, email address (the unique identifier), first name, last name, collaboration status (default ‘collaborator’), profile picture (default empty avatar), bio and password, which is shown as *****s. The user enters updated information for any of these fields and presses the “Update” button. This changes the user’s values in the database and returns the user to the Profile Screen now displaying the updated information.
Scenario:	User John Smith wants to update his collaboration status. John is logged into his account and is currently on the Dashboard screen. He clicks on the avatar icon on the top right. A drop down appears and John clicks on “Update Profile”. He is then taken to the Profile Screen. John clicks on the “Collaboration status” drop down menu and chooses the “Solo” option to specify he does not want to work with other people on projects. John clicks the “Update” button and now sees the collaboration status as ‘Solo’ instead of ‘Collaborator’.

Use Case 2.2: Delete Account

Number:	2.2
Name:	Delete Account
Story:	User wishes to delete their account. From any screen, the user clicks on the avatar icon in the top right. A dropdown appears and user clicks “Update Account”. User is forwarded to the Profile Screen where they can see all their account details including email, username, password etc. User clicks the “Delete Account” button on the bottom and is displayed with a warning modal asking for confirmation. If the user has projects, he is warned that all his data will be lost and access to shared projects will be lost. After the user confirms, he is forwarded to the Welcome screen and his information along with all his work is deleted from the database. In the case that the ‘owner’ of a shared project deletes his account, the project will not be deleted and instead one of the (randomly chosen) collaborators will be made owner.
Scenario:	John Smith wants to delete his account. He is logged into his account and is currently on the Dashboard screen. He clicks on the avatar icon on the top right. A drop down appears and John clicks on “Update Profile”. He is then taken to the Profile Screen. He scrolls down to the bottom and clicks “Delete Account”. He is presented with a warning modal that informs him that his projects will be deleted and access to any shared projects will be lost permanently. John clicks the “Confirm” button and is forwarded to the Welcome Page.

Use Case 3.1: View My Tilesets

Number:	3.1
Name:	View My Tilesets
Story:	The user is currently on the Projects Screen, which by default shows all his projects (maps and tilesets). This is the default “My Projects” tab. The user wants to browse through his tilesets only. User clicks the “My Tilesets” tab/button on the navbar in the left corner. User remains on the Projects Screen but the contents of the page are updated to display only the *tilesets* the user owns, is collaborating on or has favourited.
Scenario:	John Smith is currently viewing all his projects (maps and tiles) in the Projects Screen. He wants to view only his tilesets so he can pick out the best one for the new map he is creating. John clicks the “My Tilesets” tab/button on the navbar in the left corner. He remains on the Projects Screen but the contents of the page are updated to display all his *tilesets* including the ones he created, the ones he is collaborating on and the ones he has favourited.

Use Case 3.2: View My Maps

Number:	3.2
Name:	View My Maps
Story:	The user is currently on the Projects Screen, which by default shows all his projects (maps and tilesets). This is the default “My Projects” tab. The user wants to browse through his maps only. User clicks the “My Maps” tab/button on the navbar in the left corner. User remains on the Projects Screen but the contents of the page are updated to display only the *maps* the user owns, is collaborating on or has favourited.
Scenario:	John Smith is currently viewing all his projects (maps and tiles) in the Projects Screen. He wants to view only his maps so he can invite his friend to collaborate with him in designing the different levels of the game. John clicks the “My Maps” tab/button on the navbar in the left corner. He remains on the Projects Screen but the contents of the page are updated to display all his *maps* including the ones he created, the ones he is collaborating on and the ones he has favourited.

Use Case 3.3: View My Projects

Number:	3.3
Name:	View My Projects
Story:	The user is currently on the Projects Screen, either on the “My Maps” or “My Tilesets” tab. User wishes to view all of his projects; tilesets and maps together. User clicks the “My Projects” tab/button on the navbar in the left corner. User remains on the Projects Screen but the contents of the page are updated to display all *tilesets & maps* the user owns, is collaborating on or has favourited.
Scenario:	User John Smith is currently on the “My Maps” tab on the Projects Screen, where he is presented with a list of all his maps. John wants to view both his maps and tilesets, so he can use a filter and check which projects of his received a rating greater than 3 stars. John clicks the “My Projects” tab/button on the navbar in the left corner. He remains on the Projects Screen but the contents of the page are updated to display all his *projects*.

Use Case 3.4: Select my project

Number:	3.4
Name:	Select my project
Story:	The user is currently on the Projects Screen, on either one of the tabs “My Projects”, “My Maps” or “My Tilesets”. The user is presented with a list of appropriate projects. Each item in the list shows a small preview of the project along with its name. The user wishes to open a project to either view to edit it. User clicks anywhere on the list item. He is forwarded to the Map Manager screen if the item clicked is a map or the Tile Manager if the item clicked is a tile. In the manager screen, the chosen project is displayed and ready to edit.
Scenario:	User John Smith is currently viewing all his projects in the Projects Screen. He scrolls down the list of his projects and finds an incomplete map. John wants to select and open this map, so he can invite his friend to help complete it. John clicks on the small box (list item) which contains a miniature map preview. He is forwarded to the Map Manager screen with the chosen map displayed. Here, he is presented with multiple different options to interact, edit, share and download the given map.

Use Case 3.5: Sort projects

Number:	3.5
Name:	Sort projects
Story:	User is either viewing public projects on the Dashboard or personal projects on the Projects Screen. To increase his productivity and organization, he wants to sort his projects in a meaningful order. By default the Dashboard and Project Screen lists projects most recently modified. User clicks on the “Sort By” button/icon in the top right corner in the navbar. A drop down emerges with options to sort according to project name, creation date, most popular (num downloads), most liked (ratings), size, creator name. User selects the ‘project name’ option. User still remains on the same screen and the drop down disappears but is now displayed a projects list alphabetically sorted on project names.
Scenario:	John Smith is currently viewing all his projects on the Projects Screen. He wishes to sort them according to num downloads since he is planning to improve on his most popular projects. John clicks the “Sort By” button/icon in the top right corner in the navbar. A drop down emerges and he clicks the ‘most popular’ option. The drop down disappears and John is displayed a list of all his projects now sorted according to the number of downloads each has. The ones with the most downloads appear at the top and the ones with the least at the bottom.

Use Case 3.6: Filter projects

Number:	3.6
Name:	Filter projects
Story:	User is either viewing public projects on the Dashboard or personal projects on the Projects Screen. To increase his productivity and organization, he wants to filter projects on a useful property. User clicks on the “Filter By” button/icon in the top right corner in the navbar. A drop down emerges with options to filter by rating, size, tags, owned, shared. User selects the ‘shared’ option. User still remains on the same screen and the drop down disappears but is now displayed a projects list with only projects where he is a collaborator and did not create the project initially.
Scenario:	John Smith is currently viewing all his projects on the Projects Screen. He wishes to filter out projects where he is collaborating with others since he is planning to finish shared projects before he works on his personal stuff. John clicks the “Filter By” button/icon in the top right corner in the navbar. A drop down emerges and he clicks the ‘shared’ option. The drop down disappears and John is displayed a list of all projects where he is a collaborator.

Use Case 3.7: Search projects

Number:	3.7
Name:	Search my projects
Story:	User is either viewing public projects on the Dashboard or personal projects on the Projects Screen. User wishes to search for a particular project he designed earlier, so he can export it to a game engine. User clicks on the search bar in the top right corner of the navbar. He types in a keyword (tag) related to the map he is searching for and hits enter. A search result containing a list of maps with that tag replaces the current contents of the screen.
Scenario:	John Smith is currently viewing public projects on the Dashboard. He wants to play a maze game and so needs to download an appropriate map to upload to his game engine. John clicks the search tab in the top right corner of the navbar. He types in ‘maze’ and clicks the search icon/ or hits enter. John still remains on the dashboard but is now displayed a new list of public projects which contain a tag ‘maze’.

Use Case 4.1: Create New Map

Number:	4.1
Name:	Create New Map
Story:	User is on the Projects Screen and clicks the button named “New”. A dropdown appears under the button with one of the options being “Map". The user clicks on that option and is taken to the Map Editor screen with an empty project.
Scenario:	User John is on the Projects Screen. He clicks on the button labeled “New”. A drop down appears under the button. He clicks on the option labeled “Create New Map". He is redirected to the Map Editor with an empty new project for him to work on.

Use Case 4.2: Delete Map

Number:	4.2
Name:	Delete Map
Story:	User is on the Map Editor Screen and wants to delete their map. The user clicks on the “Settings” button and a modal shows up. Scrolling to the bottom of the modal, there is a button labeled “Delete Map”. The user clicks on that button and the modal is replaced with another modal that asks for confirmation. Once confirmed, the map is deleted from the user’s projects, the user is redirected to the Projects screen, and a banner saying that the project was deleted appears at the top. If the user has shared the project with others and is the owner, a randomly chosen user from the collaborators becomes the new owner.
Scenario:	User John is on the Map Editor Screen and no longer wants to keep the map he has made. He has shared the map with two other collaborators, users Mary and Tim. John clicks on the “Settings” button and a modal pops up. He then scrolls to the bottom of the modal and clicks the “Delete Map” button. The modal is replaced with another modal asking for confirmation. Once John confirms, the map is deleted from John’s Projects and John is redirected to the Projects screen. A banner appears saying that the project has been removed from his Projects Screen. Through random selection, Tim has become the owner.

Use Case 4.3: Edit Map Properties

Number:	4.3
Name:	Edit Map Properties
Story:	User is on the Map Editor Screen and wants to change the settings of the map. User clicks on the “Settings” button and a modal pops up with the various options that the user can change about the map based on their role. Map properties include name, description, size, tags, visibility and more.
Scenario:	User John is on the Map Editor Screen and wants to change the settings of his map. He clicks on the “Settings” button and modal pops up. As he is the owner of the map, he has the ability to change all the options he wants to change. John changes the name of the project so it is easier for other users to find his project.

Use Case 4.4: Edit/Draw Map

Number:	4.4
Name:	Edit/Draw Map
Story:	User is on the Map Editor Screen and wants to edit the map. The user clicks on the “Edit” button and it expands into the various editing options like “Copy” and “Paste”.
Scenario:	User John is on the Map Editor Screen and wants to edit the map. He clicks on the “Edit” button. It expands into the edit options and he can choose how to edit the map.

Use Case 4.5: Export Map

Number:	4.6
Name:	Export Map
Story:	User is on the Map Editor Screen and wants to export the map. The user clicks on the 3 dots button to expand the options and one option is “Export”. The user clicks that and a download of the map’s current state starts.
Scenario:	User John is on the Map Editor Screen and wants to export the map he created to take it to another computer. He clicks on the 3 dots button to expand the options and clicks on the “Export” option. A download of the map’s current state starts.

Use Case 4.7: Save Map

Number:	4.7
Name:	Save Map
Story:	User wants to save the map. User clicks the “Save Map” button and the map will be saved. A notification indicates that the map has been saved. Once saved, the user can log off and the map will still be there when he comes back.
Scenario:	John is making good progress on his map and now wants to save it so he doesn’t lose his progress. He clicks the “Save Map” button and a small pop-up notifies him that his map has been saved. Now he can work on it tomorrow and the map will still be there.

Use Case 5.1: Create New Tileset

Number:	5.1
Name:	Create New Tileset
Story:	User is on the Projects Screen and wants to create a new tileset. The user will click the “New” button and select the “Tileset” option from the dropdown menu. Now the user is taken to a tileset editor screen.
Scenario:	User John wants to create a new tileset for his game. He goes to his Projects page and clicks “New” and then “Tileset” from the drop down to start creating a tileset. This brings John to a tileset editor screen where he begins to make his new tileset.

Use Case 5.2a: Upload Tile

Number:	5.2a
Name:	Upload Tile to Tileset
Story:	User is making a tileset in the tileset editor and already has a tile made that he wants to use. User clicks the “Upload tile” button and it opens the computer’s files. The user selects a tile file of the correct format and after entering the properties of the tile and confirming, the tile is added to the current tileset.
Scenario:	John is in the tileset editor making a tileset and wants to add a tile he’s used in the past to the tileset. He clicks the “Upload tile” button which opens his computer’s files. He selects the existing tile named “grass-tile”. He entered the pixel dimensions of the tile to properly fit it into the tileset and confirmed the action. The tile is now in the tileset editor as a part of his tileset.

Use Case 5.2b: Create New Tile

Number:	5.2b
Name:	Create New Tile
Story:	User is making a tileset in the tileset editor and wishes to add a new tile to the tileset. The user clicks the “add” button and an empty tile is added to the tileset.
Scenario:	John created a new tileset and wants to create his first tile. He clicks the “add” button and it creates an empty tile in the tileset editor. He can now make his tile.

Use Case 5.2c: Edit Tile

Number:	5.2c
Name:	Edit Tile
Story:	Users have the option to either color in pixels, erase pixels, clear pixels, fill in all pixels, and select color to edit the tile.
Scenario:	John creates an empty tile and begins to work on it. He selects the color blue and clicks the bucket icon then the canvas to turn the entire tile blue John then chooses the paintbrush tool and selects the color cyan. John paints individual pixels on the tile to achieve a dotted effect. He erases the mistakes with the eraser tool. After finishing, he decides he doesn't like it and clears the whole tile.

Use Case 5.2d: Remove Tile

Number:	5.2d
Name:	Remove Tile
Story:	User wants to remove a tile from his tileset. User clicks the "delete tile" button which brings up a confirmation modal. Once the user confirms the decision, the tile will then be deleted from the tileset and the user is brought back to the editor to continue working on his tileset.
Scenario:	John is in the middle of making his tile when he decides that he doesn't like it anymore. He decides to completely delete the tile. He clicks the "delete tile" button. John sees a modal pop up asking if he is sure about his decision. He is sure and presses the confirmation button. The tile has now been deleted and John continues to work on his tileset.

Use Case 5.3: Edit Tileset Properties

Number:	5.3
Name:	Edit Tileset Properties
Story:	User wants to edit the properties of his tileset. The user, while on the tileset editor screen, clicks the “settings” button to open the “Tileset Properties” modal. This modal will have settings like name, tags, and description that the user can edit.
Scenario:	John is in the middle of making his tileset when he suddenly wants to change the name. He clicks on the “settings” button to bring up the “Tileset Properties” modal. On this modal, John changes the tileset’s name from “Dungeon Tileset” to “Super Dungeon Tileset”. He does not change the other properties. He closes the modal, and goes back to work.

Use Case 5.4: Delete Tileset

Number:	5.4
Name:	Delete Tileset
Story:	User wants to delete a tileset. The user goes to his projects page and finds the tileset he wants to delete. The user clicks the delete button next to the tileset which brings up a modal asking for confirmation. After clicking “Yes”, the tileset is deleted. Once the tileset has been deleted from the user’s projects, if it was shared with others and the user was the owner, ownership will be passed on to the person appointed by the user.
Scenario:	John decides that his “Super Dungeon Tileset” doesn’t look too super anymore. He wants to delete it. John navigates to his projects page and finds the tileset. He clicks the delete button which brings up a confirmation modal. John clicks yes to confirm. Another modal asks John to appoint the next owner of the tileset after he is gone. He chooses Tim as the next owner. After clicking okay, John has deleted the tileset and Tim is the new owner of the tileset.

Use Case 5.5: Import Tileset

Number:	5.5
Name:	Import Tileset
Story:	User wants to import a tileset into Pieces. On the Projects screen, the user clicks the “Import” button which opens the computer’s files. From the files, the user is able to select a tileset to import into Pieces, as long as the file is of the correct format. After confirming his selection, the tileset is now in the tileset editor and ready to be changed.
Scenario:	John decides it’s best to import a cool tileset he found on the web. To do so, he navigates to the Projects screen and clicks the “Import” button. This brings up his computer’s files, from which he selects a file called “simple-dungeon-tileset”. After confirming his selection, the imported tileset is open in the tileset editor and is ready to be worked on.

Use Case 5.6: Export Tileset

Number:	5.6
Name:	Export Tileset
Story:	User wants to export a tileset. In the tileset editor page, he clicks the “Export” button. User then selects which directory to save the file to. After confirmation, the tileset is saved as a file onto his computer.
Scenario:	John finishes his tileset and wants to export it now. He clicks the “Export” button from his tileset editor screen and it brings up his files application. He decides to name his tileset, “Super Dungeon Tileset 2” and saves it to his “Game Files” folder. The tileset has now been saved to his computer.

Use Case 5.7: Save Tileset

Number:	5.7
Name:	Save Tileset
Story:	User wants to save the tileset. User clicks the “Save Tileset” button and the tileset will be saved. A notification indicates that the tileset has been saved. Once saved, the user can log off and the saved tileset will still be there when he comes back.
Scenario:	John is making good progress on his dungeon tileset and now wants to save it so he doesn’t lose his progress. He clicks the “Save Tileset” button and a small pop-up notifies him that his tileset has been saved. Now he can work on it tomorrow and the tileset will still be there.

Use Case 6.1: Undo

Number:	6.1
Name:	Undo
Story:	User has made some edits to a tile in a tileset or to the tiles in a tile map and has made a mistake. The user wants to undo the most recent action(s) to start from a point in the past. The user clicks the Undo Icon or presses Ctrl + Z (popular undo hotkey) and it undoes the most recent action. Users can repeat this to continue undoing actions up to a limit of 50 undos.
Scenario:	User John has made a mistake in editing his tile map. He accidently moved one of the tiles to the wrong position. He presses Ctrl + Z to undo the most recent action.

Use Case 6.2: Redo

Number:	6.2
Name:	Redo
Story:	User has made some edits to a tile in a tileset or to the tiles in a tile map and has made a mistake. The user has undone the most recent action(s), but changes his mind and wants his most recent changes back.. The user clicks the Redo Icon or presses Ctrl + Y (popular redo hotkey) and it redoes the most recent undone action. Users can repeat this to continue redoing actions up to a limit of 50 redos.
Scenario:	John has made a mistake in editing his tile map. He presses Ctrl + Z to undo the most recent action. After some consideration, he realizes his mistake can be amended instead of undone. He clicks the Redo Icon to bring back his undone work.

Use Case 6.3: View Collaborators

Number:	6.3
Name:	View Collaborators
Story:	The user wants to see the collaborators that have contributed to a map or a tileset. The user presses the Collaborators Icon. A sidebar is displayed with all the users that have contributed with options to add them as a friend.
Scenario:	John is curious as to who has been editing his tile map. He clicks the Collaborators Icon to find out. A sidebar is displayed with all the users that have contributed to his map. He decides he wants to add them as a friend to communicate with in future projects.

Use Case 6.4: Manage Access

Number:	6.4
Name:	Manage Access
Story:	The user wants to make their tileset/map public. They go into the settings of the tileset/map and press the Visibility Icon toggle that manages whether the project is public/private for view permissions. The user also wants others to be able to edit their tileset/map. They press the Lock Icon toggle that manages whether the project can be edited by others.
Scenario:	John wants others to see and edit his tileset. He clicks the Settings Icon on his tileset. A modal is displayed with options and toggles. He clicks the Lock Icon and the Visibility Icon to allow others to edit and see his tileset publicly.

Use Case 6.5: Edit Roles (Sharing and Restricting Access to Specific Users)

Number:	6.5
Name:	Edit Roles (Sharing and Restricting Access to Specific Users)
Story:	User A wants to prevent User B from accessing their tileset/map and also share access to a User C. User A has to click the Share Icon button in their project to access sharing permissions. This will open up a modal with a search box and a list of exception users. They will have to search for User B with their email or name. After clicking on User B, User A will have a dropdown to select whether User B is a viewer, an editor, or none. If User C is on User A's friends list, User A can go into their friends list and click the Invite Icon, which opens a modal. Here, User A can select whether to share view or edit permissions to their project. If User C is not on User A's friends list, User A has to go to sharing permissions in their project and press the Add Icon. They will have to enter the user's email (email will not show, but will search properly) or name in the search and select the proper user.
Scenario:	John wants to add Amy as a collaborator without making his project public. John opens his project and clicks the Share Icon button. John next enters "Amy" in the search box and then selects the proper user account. Amy is added to the list of Exceptions, where John selects "editor" as her permissions.

Use Case 7.1: Add Friend

Number:	7.1
Name:	Add Friend
Story:	User is on the Dashboard Screen and wants to add another as a friend. User clicks on the button to open the social sidebar. Once there, the user clicks on an icon to add a new friend and a textfield appears. User enters the other user's unique username in the textfield. User then clicks "Search" in order to find the user. There, they can click on the name in order to send a friend request which the other user can accept or deny.
Scenario:	User John wants to add user Mary as a friend. John opens the social sidebar and clicks on the "Add Friend" icon. A textfield is shown and John enters Mary's username in the textfield and clicks the "Search" icon. Mary's username is shown in the search results area assuming John did not mistype or that Mary did not tell him a fake username. John clicks on Mary's name to add Mary as a friend. Mary receives a notification that John wishes to be a friend and can accept or deny his request.

Use Case 7.2: Remove Friend

Number:	7.2
Name:	Remove Friend
Story:	User is on the Dashboard Screen. User no longer wishes to be friends with another user. User opens the social sidebar and clicks on that user's username. A contextual menu opens up and an option to unfriend is available. User clicks on the option and a modal pops up asking for confirmation. Once confirmed, the two users are no longer friends and are removed from each other's social sidebar.
Scenario:	User John had a falling out with user Mary and no longer wants to be friends with her. John finds her username in the social sidebar. John clicks on her name and is greeted with a contextual menu that shows an option to unfriend. Once clicked, a modal appears asking him to confirm that he would like to no longer be friends with Mary. John confirms and Mary is removed from John's social sidebar and John is removed from Mary's social sidebar.

Use Case 7.3: Invite to Collaborate

Number:	7.3
Name:	Invite to Collaborate
Story:	User is on the Dashboard Screen. User clicks on the social sidebar to expand the friends list. User then clicks on the friend in order to open the contextual menu, one of the options being “Invite to Collaborate”. This opens a secondary menu that has a list of all the projects in the primary user’s projects menu. User finds the chosen project and clicks on it. User then clicks the “Invite” button to confirm and a notification is sent to the other user that they were added to the project.
Scenario:	User John wants some help from user Mary on a project that he is working on. He knows he has sharing permissions and wants to invite Mary to collaborate. He goes to the dashboard and expands his social sidebar. He clicks on Mary’s username in his friends list and opens the contextual menu. He clicks on the “Invite to Collaborate” option which opens a secondary menu which lets him choose which project. He selects the project he wants and clicks the “Invite” button. Mary is sent a notification that she was added to John’s project and can now view and work on his project.

Use Case 7.4: Check Notifications

Number:	7.4
Name:	Check Notifications
Story:	User is on the Dashboard Screen. User clicks on the social sidebar. User can then click on the notifications button to open a notifications sidebar that replaces the social sidebar and will list all recent notifications for that user.
Scenario:	John wants to check his notifications to see if he was invited to collaborate on any projects. He clicks on the social sidebar and clicks on the notification bell. The notifications sidebar opens and shows that he has no unseen notifications.

Use Case 7.5: Add Forum Thread

Number:	7.4
Name:	Add Forum Thread
Story:	User is on the Forums Screen. User wants to start a new forum thread for their issues/thoughts. User clicks the “New Thread” button and is taken to the Create Thread Screen. The user enters the name of the thread in the name textfield. The user enters any additional description in the description textfield. The user enters any tags that they would like in the tags dropdown. The user clicks on the “Post” button to post their thread to the forum.
Scenario:	User John wants to create a thread to recruit collaborators to his project. He navigates to the Forums screen and clicks on the button “New Thread”. He enters “Request for Collaborators” in the thread name textfield. He enters a description of his project and his requirements in the descriptions textfield. He adds the tag “Help wanted” and “unpaid” from the tags dropdown. He clicks on the “Post” button and his thread is added to the forums to be viewed by other users.

Use Case 7.6: Delete Forum Thread

Number:	7.6
Name:	Delete Forum Thread
Story:	User is on the Forums Screen. User no longer wishes to keep their forum post active. User clicks on the dropdown button of their forum post. An option appears named “Delete Thread” and the user clicks on it. A modal appears asking for confirmation and once given, thread is removed from the forums.
Scenario:	User John no longer needs any more collaborators. He navigates to the forums screen and finds his post asking for collaborators. He clicks on the dropdown button on the thread and clicks “Delete Thread”. A modal appears that asks for confirmation and John confirms. The thread is then removed from the forums.

Use Case 7.7: Add Comment

Number:	7.7
Name:	Add Comment
Story:	User is on the Forums Screen. User clicks on a thread to expand it. A textfield is available for the user to add a comment to the thread if they wish to. User enters some text in the textfield and clicks on the “Comment” button. User’s comment is now linked to the thread and appears under the thread.
Scenario:	User John is on the Forums Screen and sees a request for assistance from another user. John expands the thread and writes in the textfield that he is available to help and what his qualifications are. John clicks on the “Comment” button in order to post his comment under the thread.

Use Case 7.8: Delete Comment

Number:	7.8
Name:	Delete Comment
Story:	User is on the Forums Screen. User no longer wants their comment under a specific thread. User expands the thread and finds their comment. They click the dropdown next to their name and clicks “Delete Comment”. The comment is removed from the thread.
Scenario:	User John no longer wants to assist the other user. He navigates to that specific thread and finds his comment. He clicks on the dropdown next to it and clicks “Delete Comment”. The comment is removed from the thread.

Use Case 8.1: View/select project

Number:	8.1
Name:	View/select project
Story:	The user wants to open a project on the dashboard. The user clicks on the project and the project opens.
Scenario:	John wants to see the tile map “Space” because it is highly rated. He clicks on the box that consists of the preview of the tile map and the name.

Use Case 8.2: Rate public project

Number:	8.2
Name:	Rate public project
Story:	The user wants to help promote a project they found interesting. The user clicks on the Thumbs Up Icon next to the project to increase its like count. The user can also click on the Thumbs Down Icon to decrease the project’s like count. If the user clicks on the Icon when it is already highlighted, it will undo the rating.
Scenario:	John thought the tile map “Space” was really interesting because it really looked like space. He goes back to the dashboard and clicks the Thumbs Up button for “Space” to increase the like count (the rating) of the tile map.

Use Case 8.3: Comment on project

Number:	8.3
Name:	Comment on project
Story:	The user has some feedback they want to share about the project they just saw. The user goes to the dashboard and clicks the Comment Icon for the project. A modal will open where the user can add a comment and reply to other comments. The user can also increase/decrease the rating of the comments that are listed by pressing the corresponding Thumbs Up and Thumbs Down buttons. Comments will be sorted by rating by default, but the user can sort the comments by recent instead. To do this, the user changes the dropdown in the top right of the modal to say “By Recent” instead of “By Helpful” (default).
Scenario:	John thought the tile map “Space” was really interesting because it really looked like space. He wants to leave a comment of appreciation, so he goes back to the dashboard and clicks on the Comment Icon for “Space”. Here, he adds a new comment saying that it was an interesting tile map.

Use Case 8.4: Favorite public project

Number:	8.6
Name:	Favorite public project
Story:	The user wants to save a project to a personalized list. The user presses the Favorite Icon for the project and it is now saved in the user’s personal favorites list.
Scenario:	John thought the tile map “Space” was interesting, so he decided to remember the project by pressing the Favorite Icon for the project. Now, whenever he wants to find “Space”, he can filter by Favorites in the search for the dashboard.

Use Case 8.5: Download public project

Number:	8.7
Name:	Download public project
Story:	The user wants to download a tileset to use for a tile map as well as a tile map to use for his game. The user needs to have view permission of the project to be able to have a download button for it. The user clicks on the Download Icon for the projects in the dashboard. Alternatively, the user can click the Download Icon when viewing the project.
Scenario:	John thought the tile map “Space” was interesting, so he decided to download “Space” to use for his own game. Currently in the map editor for “Space”, John clicks on the Download Icon.

User Interface Model

User Interface View Listing

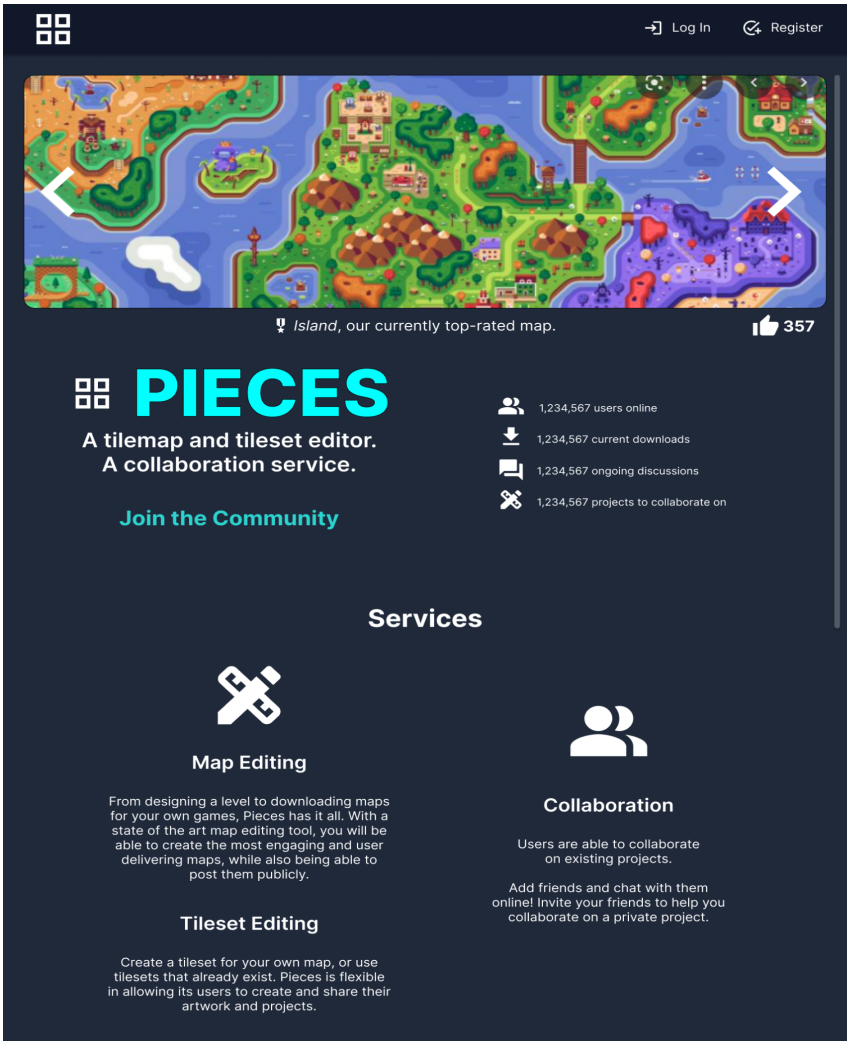
#	Name	Description
1	Welcome Screen	When users are signed out, this screen welcomes them to either create a new account or log in to join the community of map editors, tileset editors, and collaborators/viewers.
2.1	Explore Screen	This screen shows the users all the top trending maps, allowing the user to browse through all public maps, and be able to vote, comment, download, etc.
2.2	Explore Screen (Comments)	This view replaces the body of the explore screen to display comments of the selected tileset/map and be able to add a comment as well as like/dislike.
3.1	Library Screen	This screen lets the user go into the collection of projects they have created as well as projects they may have favorited.
3.2	Library Screen (Add project modal)	This screen lets the user choose the type of project they want to create in their library and pick what default resolution they want for the pixels in the tileset or the number of tiles in the tile map.
4.1	Profile Screen	This screen shows the user all of their account settings and lets the user change their account details as well as delete their account.
4.2	Profile Screen (Edit Mode)	This screen lets the user edit their profile details such as username, email, password etc.
4.3	Profile Screen (Avatar Modal)	This screen is a customization screen for the user to change their avatar on the website to a picture of their choice.
4.4	Profile Screen (Delete Account Modal)	This screen is a confirmation screen for the user to make sure that they want to delete their account.

5.1	Community Screen	This screen shows the user all the threads that are public and shows how popular/positively rated they are.
5.2	Community Screen (Expand thread)	This screen shows the user all the comments of a thread they selected and allows the user to rate the thread, view comments, and to add a comment of their own.
6.1	Map Editor	This screen lets the user create and edit maps by offering features such as putting down tiles, erasing tiles, selecting groups of tiles, moving tiles, managing layers, etc.
6.2	Map Editor (Socials Tab)	This screen helps promote collaboration between team mates with conference chat, suggestions, and permission management.
6.3	Map Editor (Settings Tab)	This screen lets the user import and export maps as well as change tileset properties like title, description, tags, etc.
7.1	Tile Editor	This screen allows the user to create their tileset along with all the involved editing features such as painting, erasing and selecting pixels as well as moving canvas, zooming in, zooming out, etc.
7.2	Tile Editor (Socials Tab)	This screen helps promote collaboration between team mates with conference chat, suggestions, and permission management.
7.3	Tile Editor (Settings Tab)	This screen lets the user import, export tilesets, upload a tile, and change tileset properties like title, description, tags, etc.
8	Tile & Map Editor Modals	Modals on editor screens allow the user to edit permissions and users, and import/export projects.
9	Notifications sidebar	This sidebar lists all notifications the user has received such as new friend requests, like/dislikes on posts etc. It slides out from the right side of any screen when the notification bell icon is clicked.

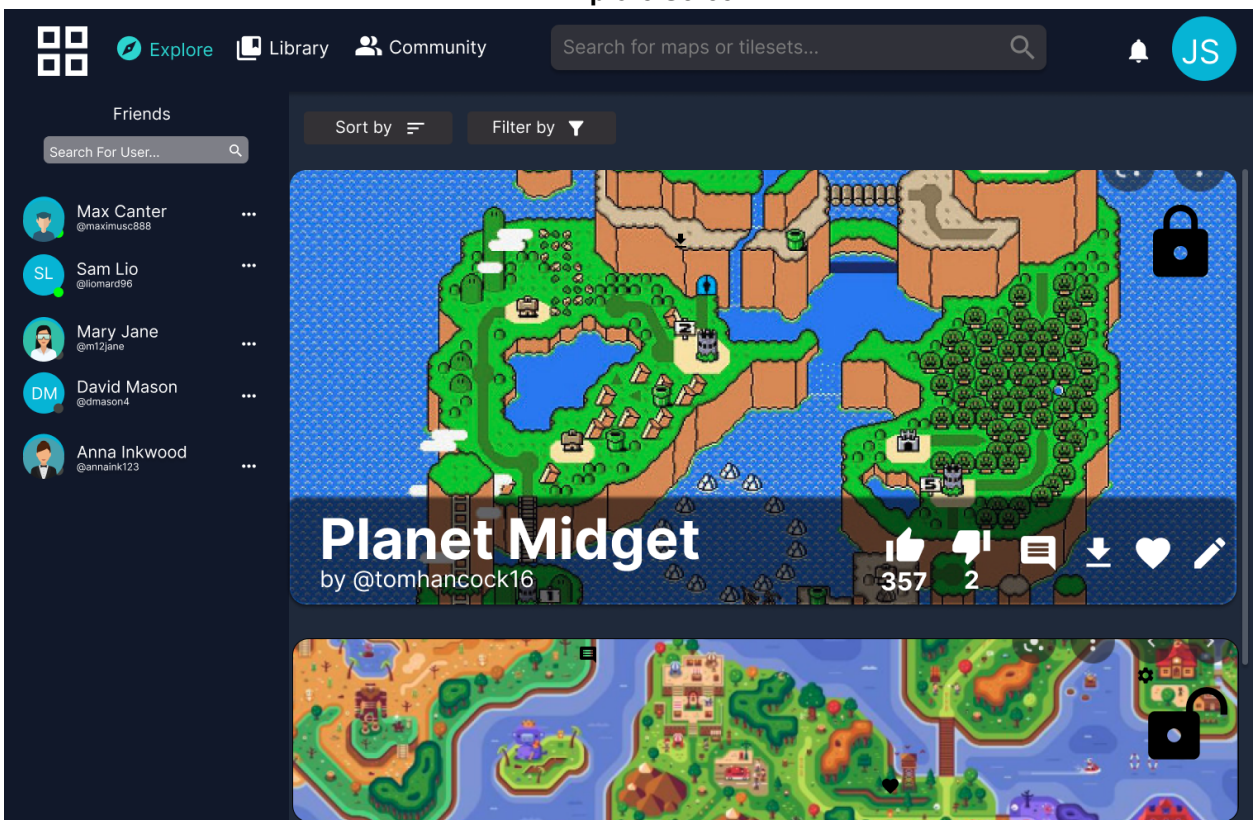
10	Social Sidebar (add friend)	Users can see online status and have an option to start a private chat between themselves and the chosen user. The user can search for another user and click the add friend button on an unfriended user to send a request.
11	User drop down	This screen shows a small dropdown that can be accessed from any screen, allowing the user to go to the profile screen to update account settings or to logout.
12	Register & Login Modals	These modals include important user registration and login information such as email, password, first and last name, and username, popping up when the log in or register buttons are clicked.

User Interface Mockup Diagrams

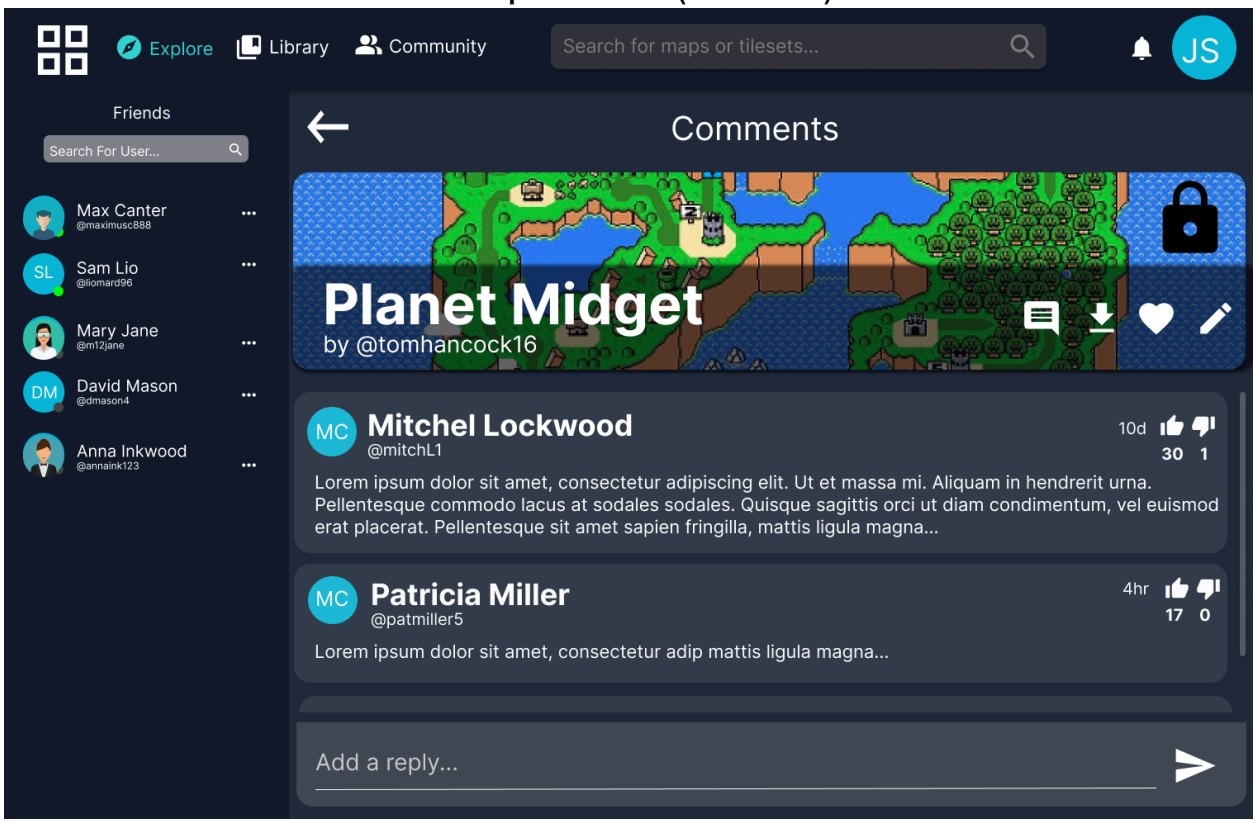
1 - Welcome Screen



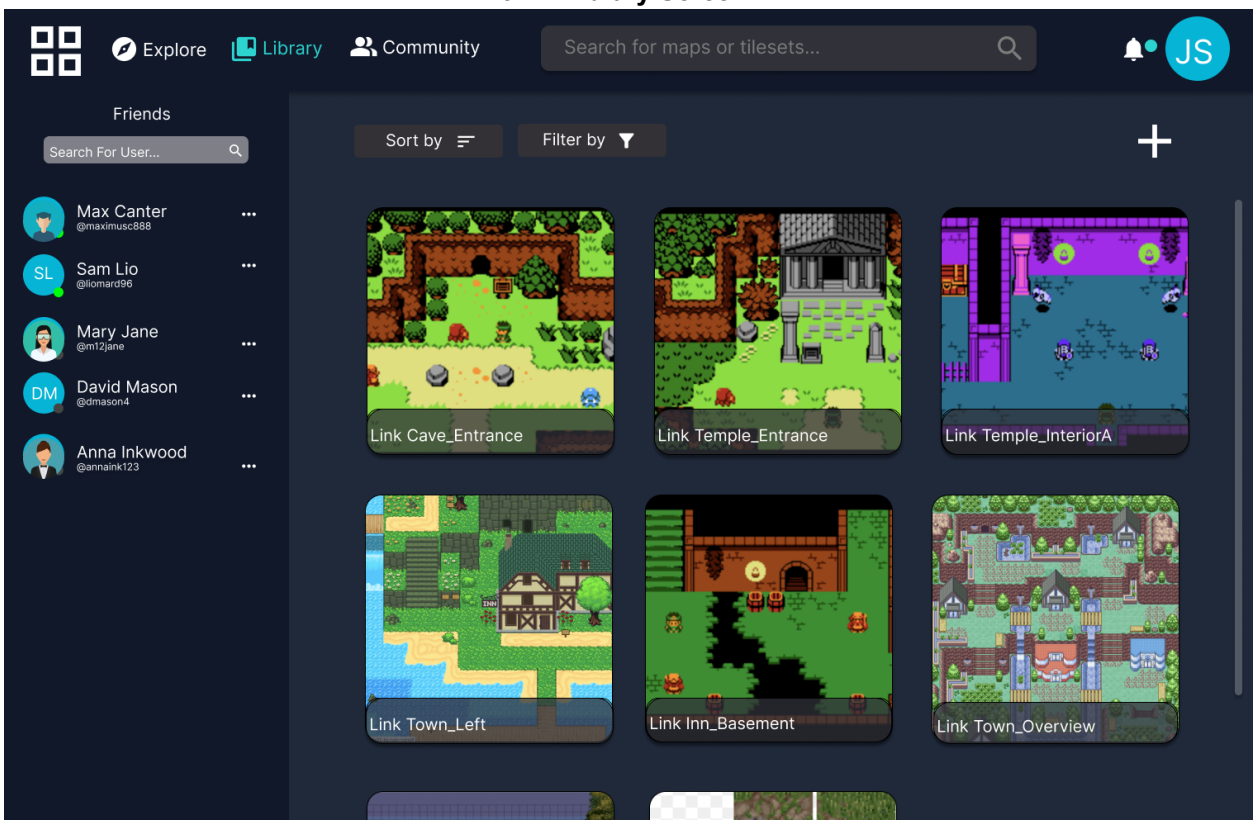
2.1 - Explore Screen



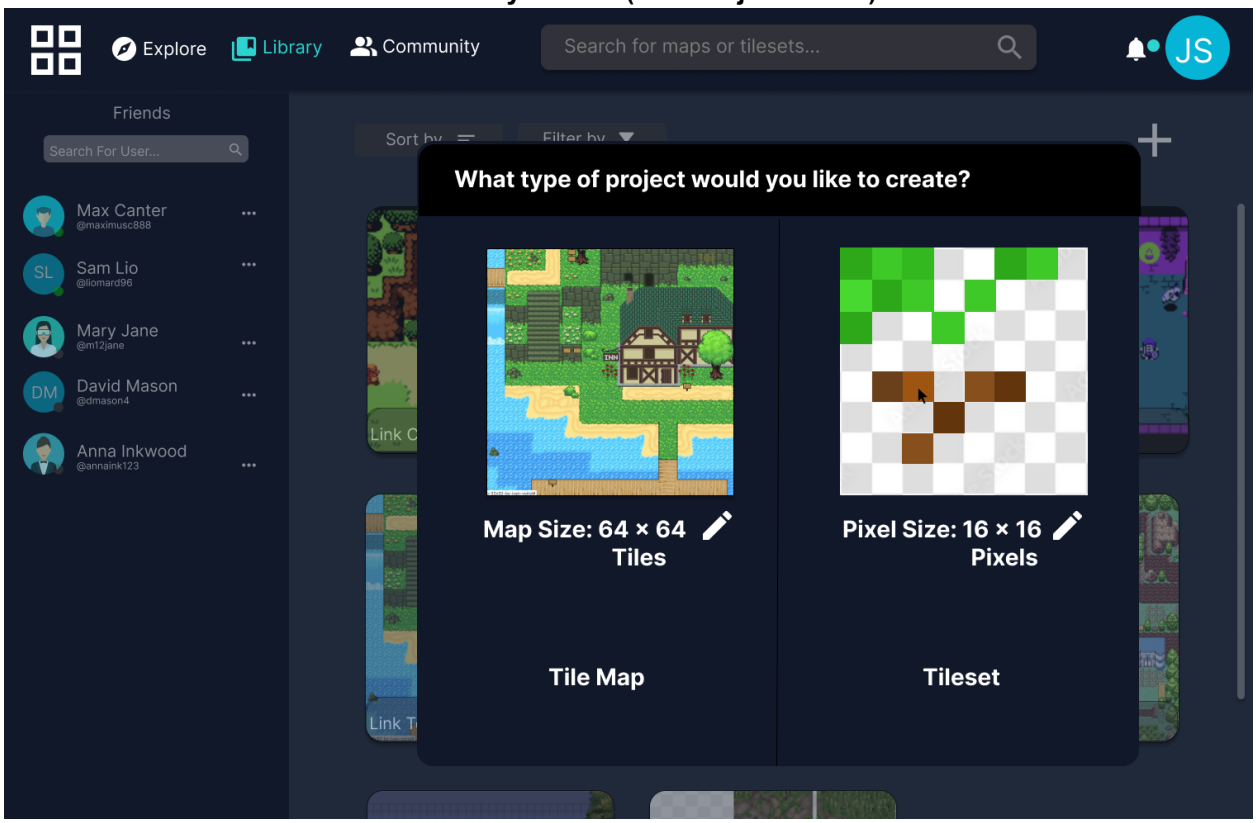
2.2 - Explore Screen (Comments)



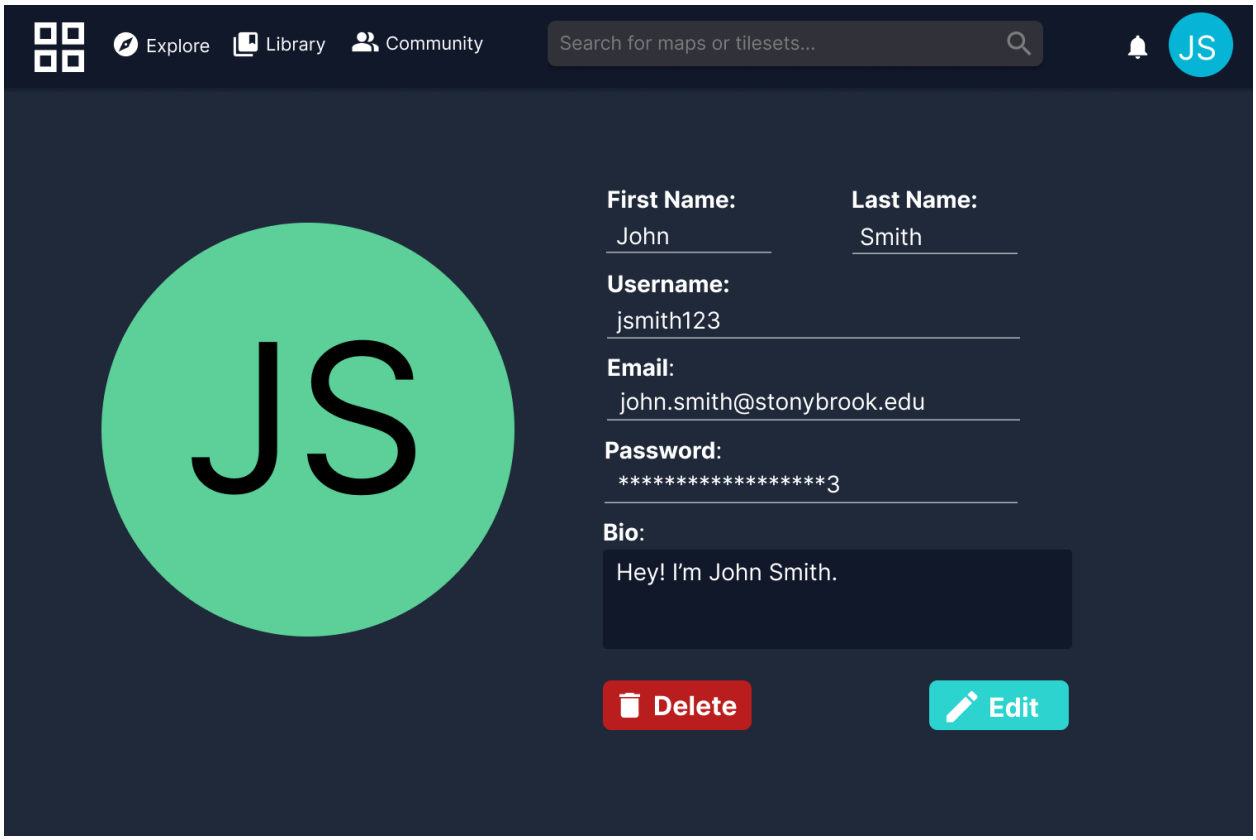
3.1 - Library Screen



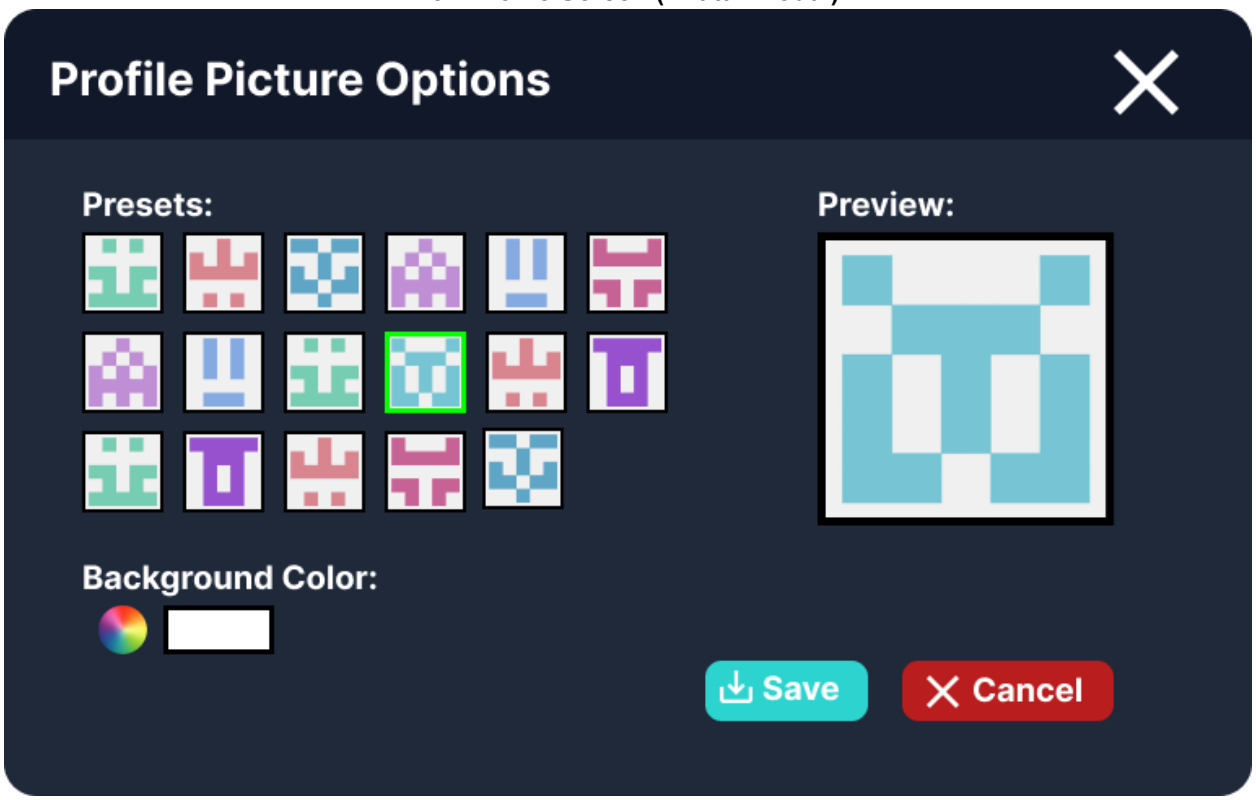
3.2 - Library Screen (Add Project Modal)



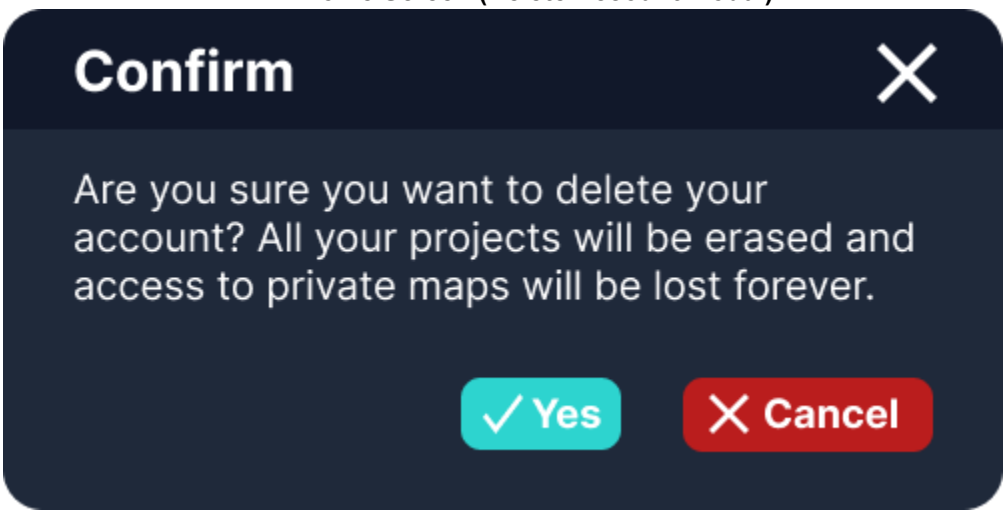
4.1 - Profile Screen



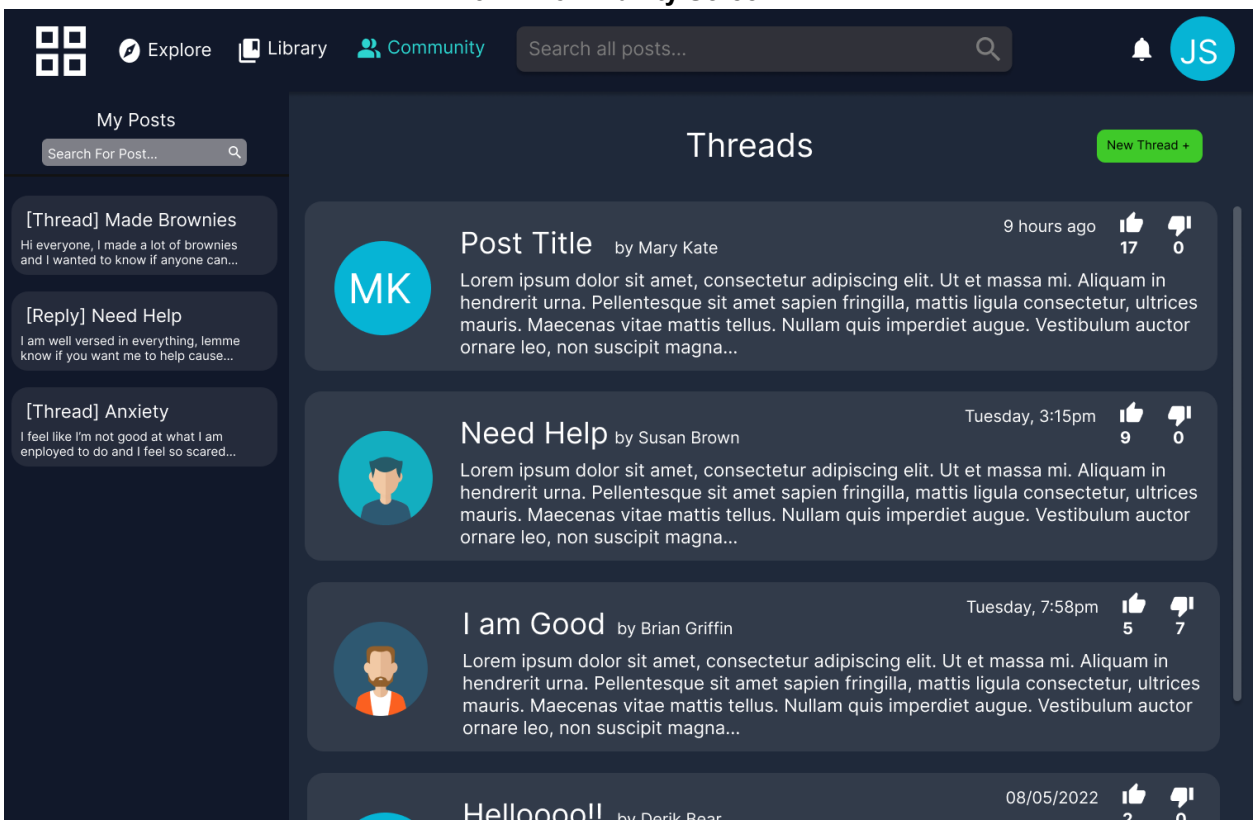
4.3 - Profile Screen (Avatar Modal)



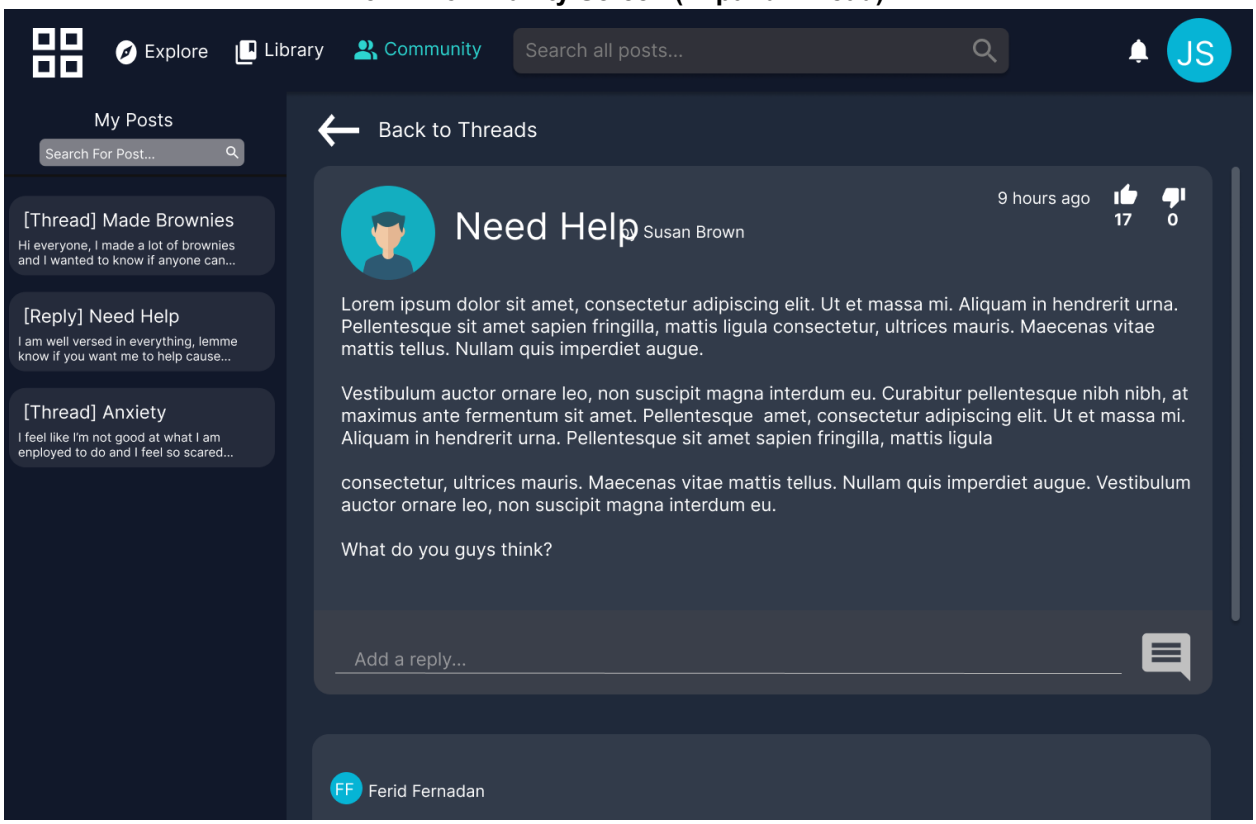
4.4 - Profile Screen (Delete Account Modal)



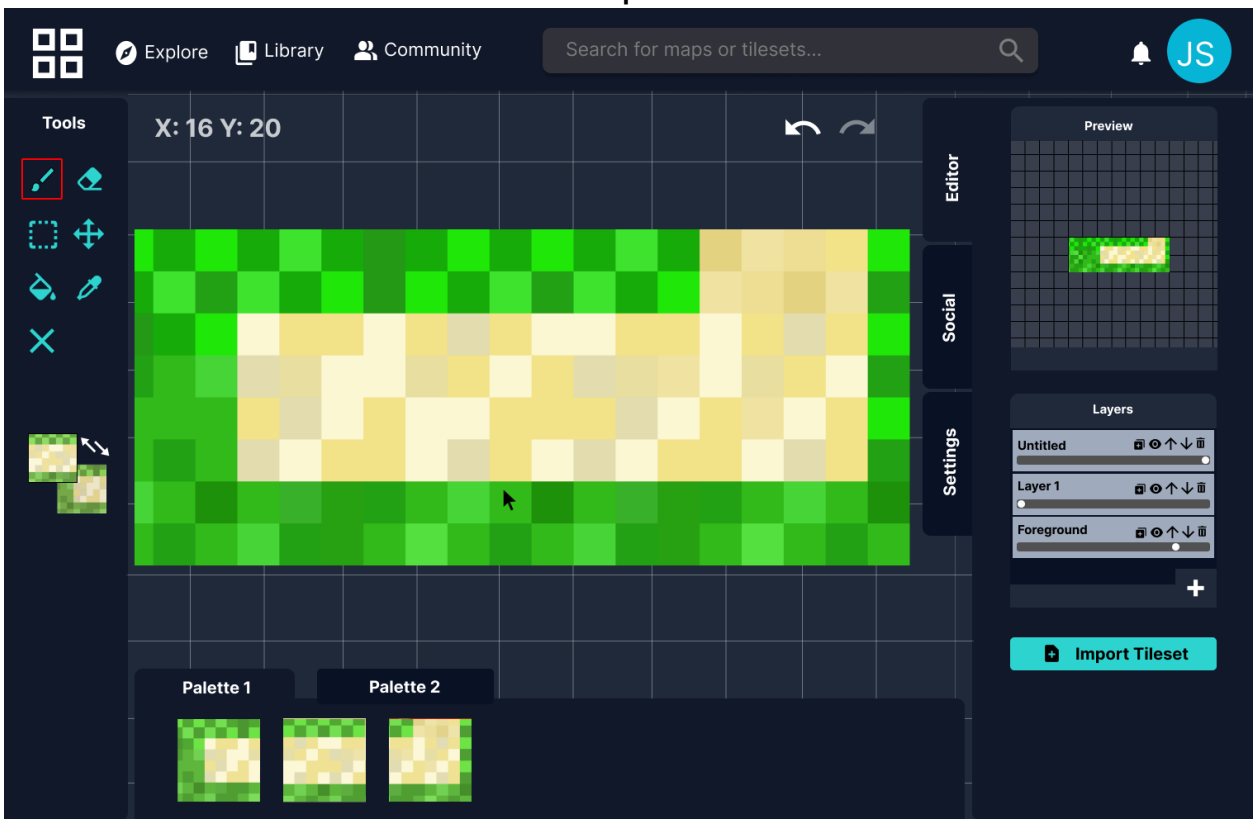
5.1 - Community Screen



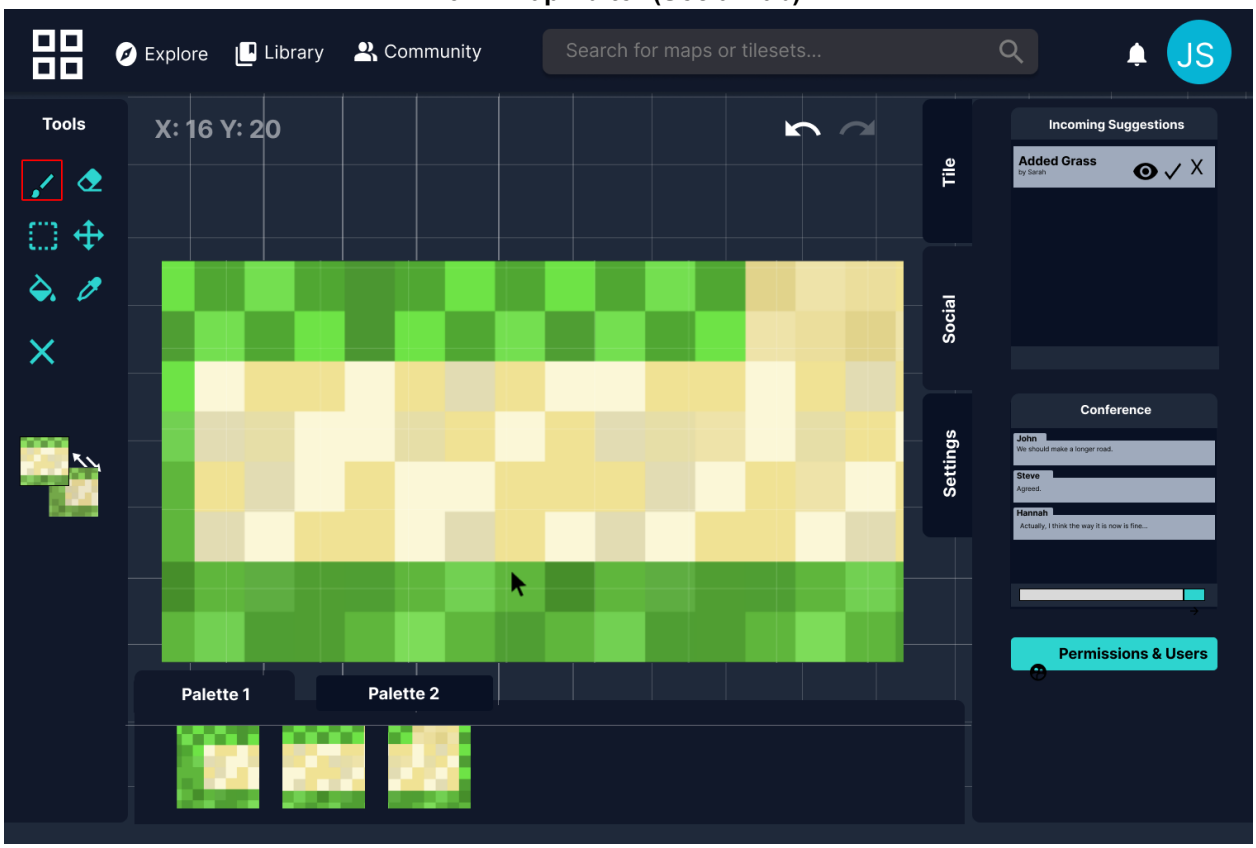
5.2 - Community Screen (Expand Thread)



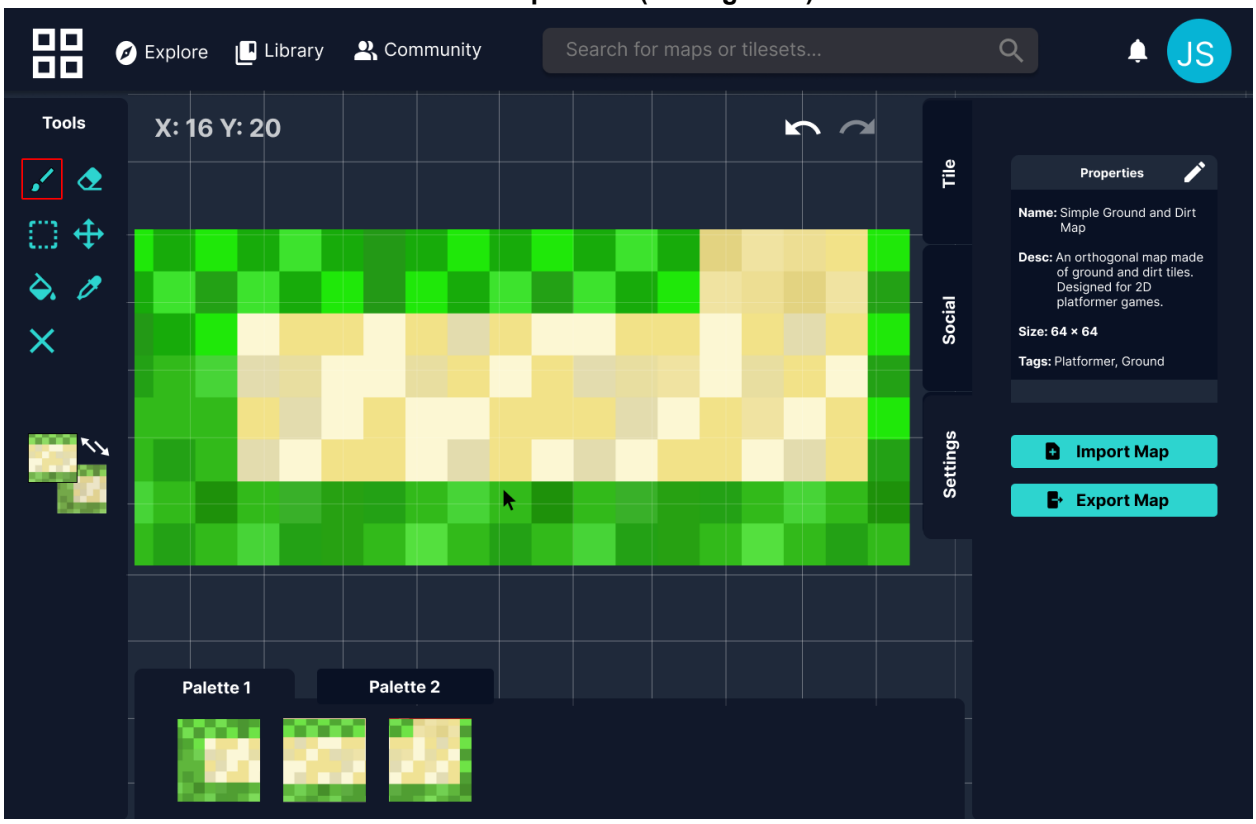
6.1 - Map Editor



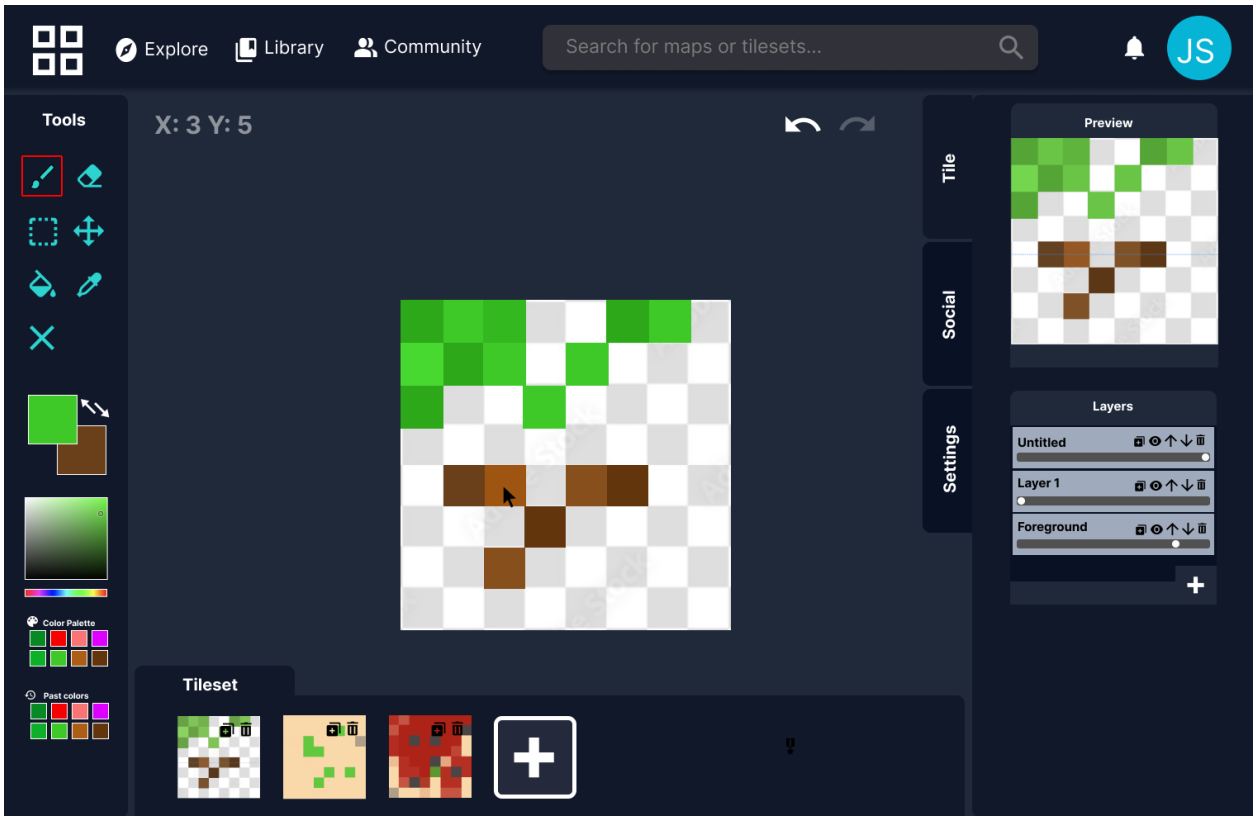
6.2 - Map Editor (Social Tab)



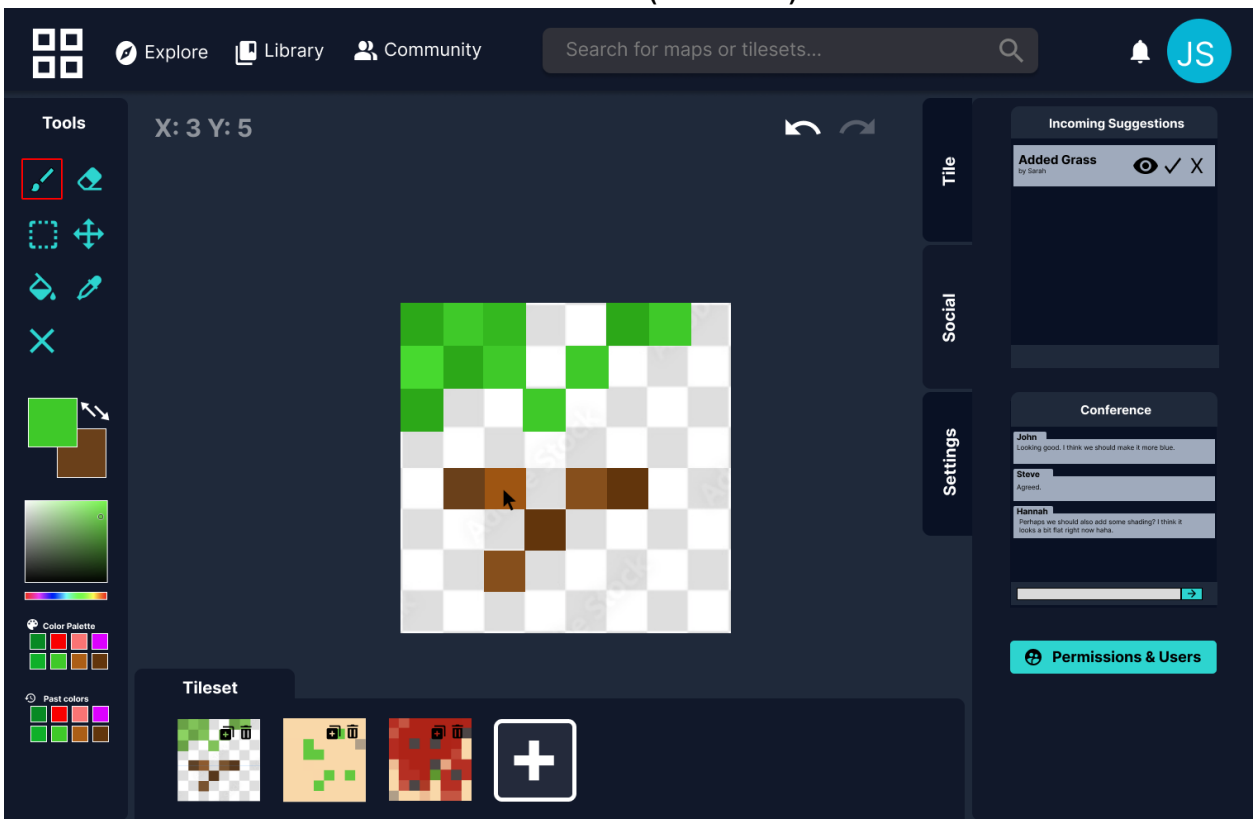
6.3 - Map Editor (Settings Tab)



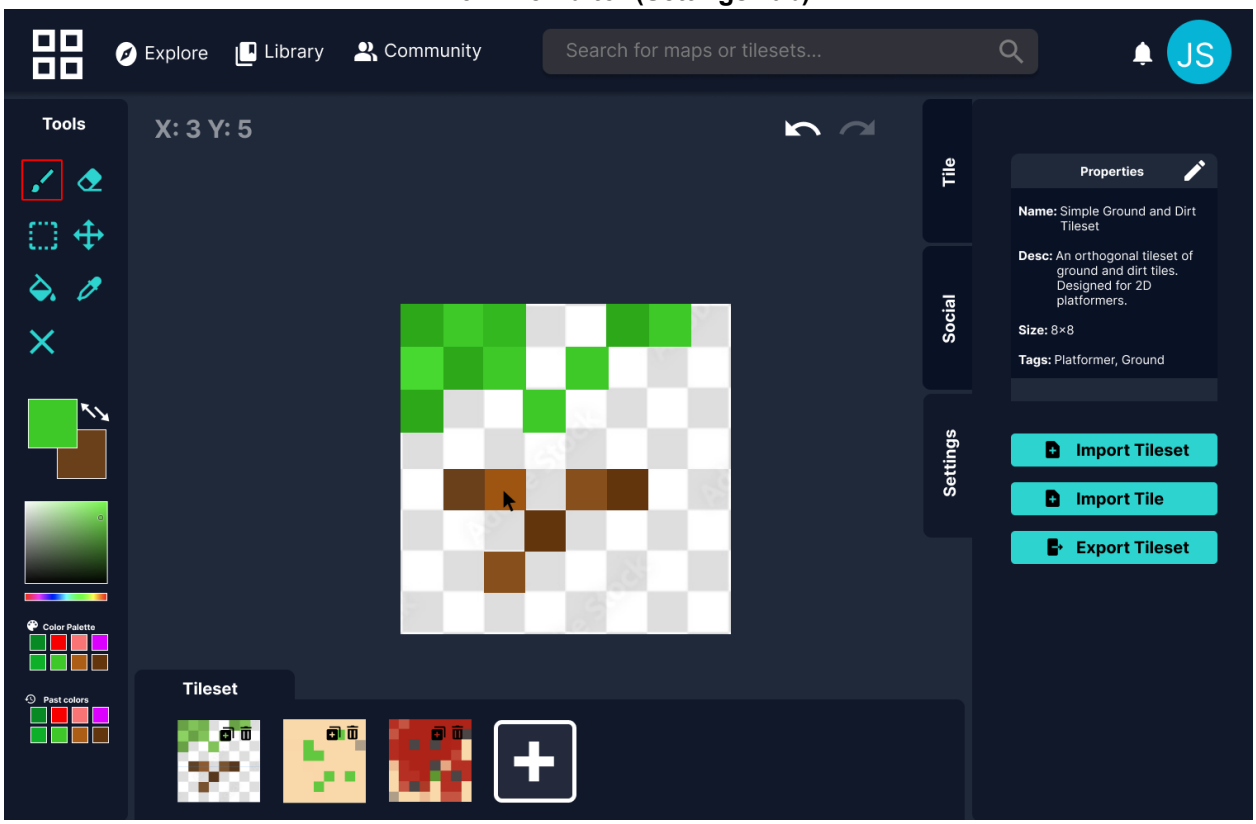
7.1 - Tile Editor



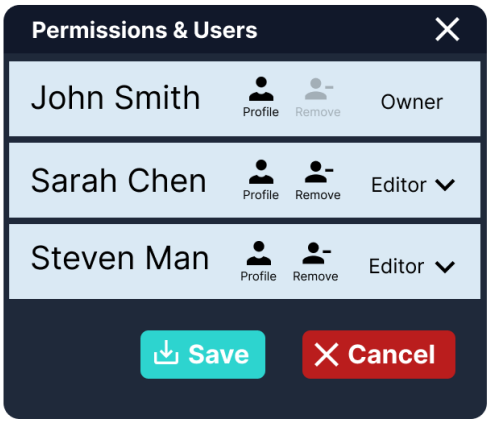
7.2 - Tile Editor (Social Tab)

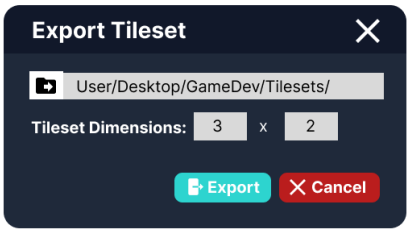


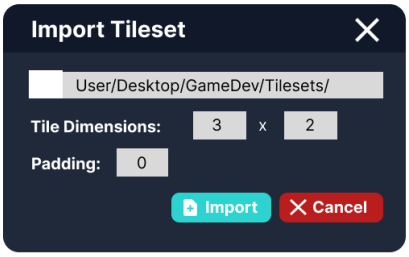
7.3 - Tile Editor (Settings Tab)

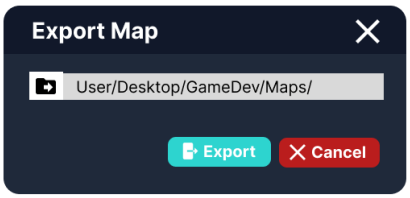


8 - Tile & Map Editor Modals

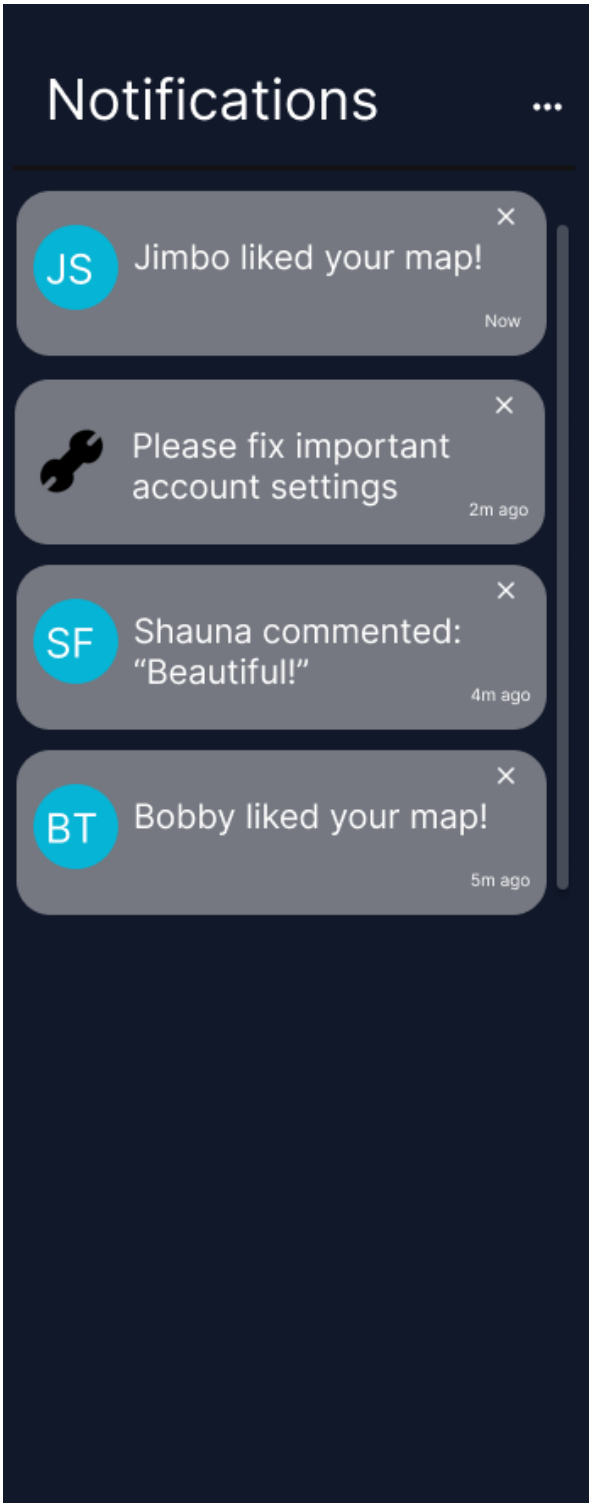




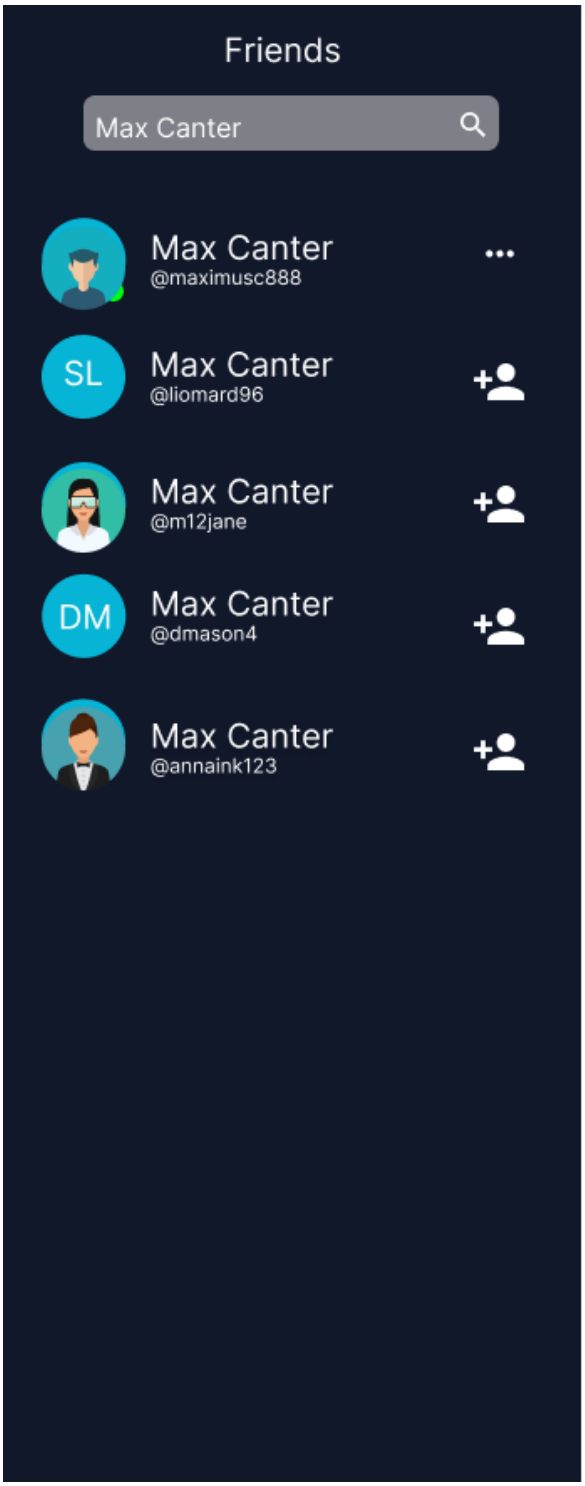




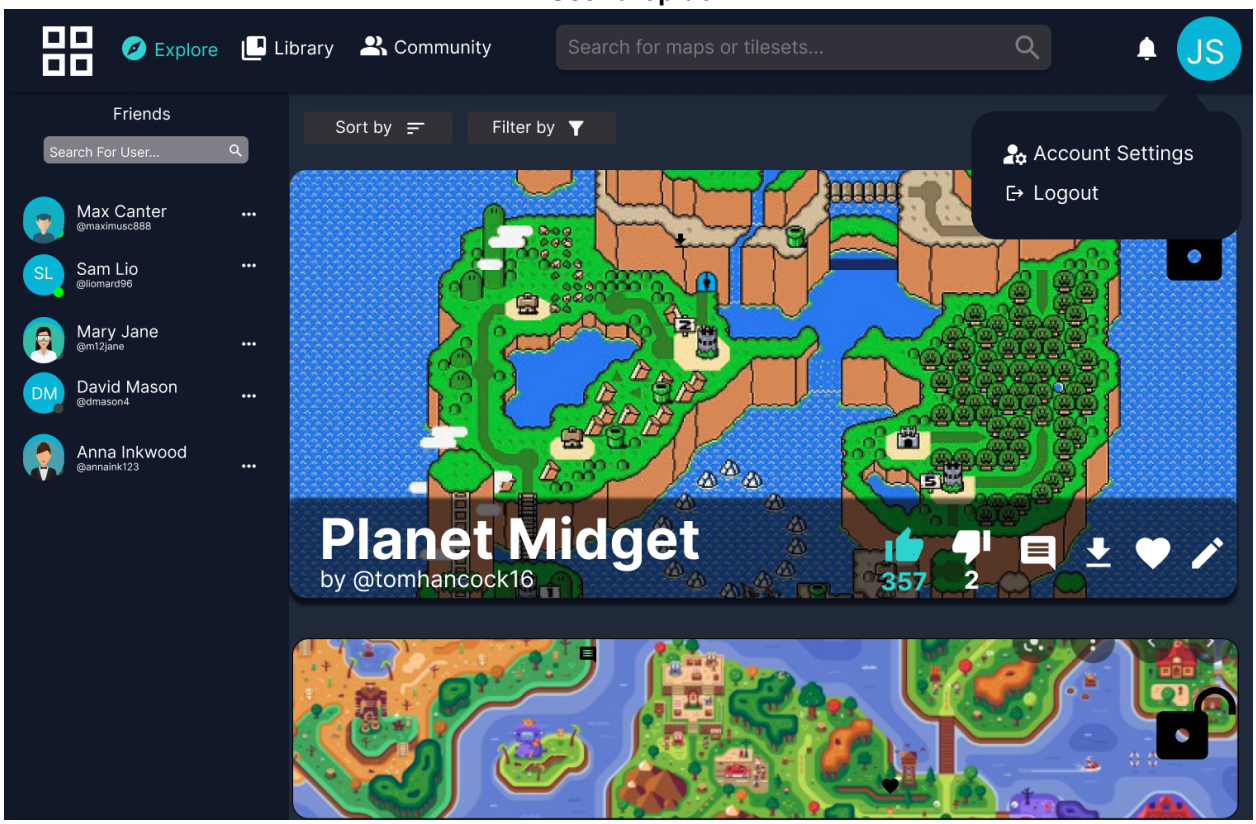
9 - Notifications Sidebar



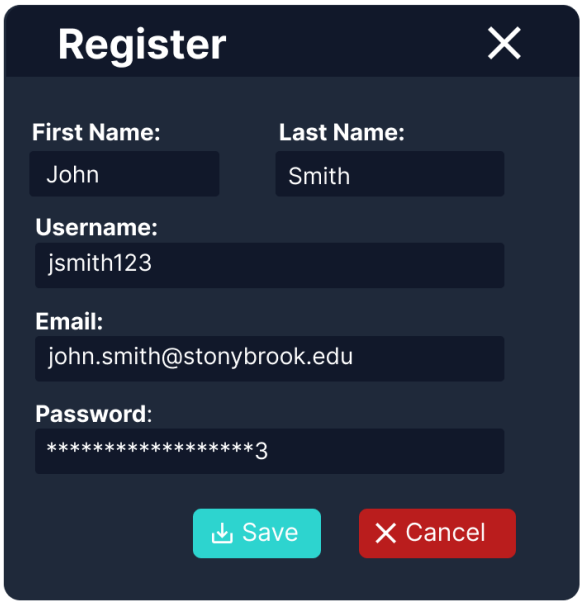
10 - Social Sidebar (add friend)

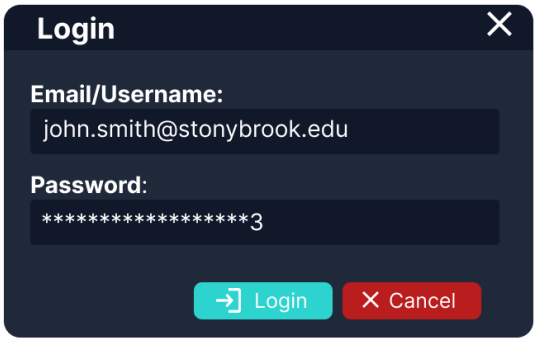


11 - User drop down



12 - Register & Login Modals





Data Model

Persistent Data Design

User Model

```
const userSchema = new Schema(
    {
        _id: {
            type: ObjectId,
            required: true
        },
        firstName: {
            type: String,
            required: true
        },
        lastName: {
            type: String,
            required: true
        },
        email: {
            type: String,
            required: true
        },
        username: {
            type: String,
            required: true
        },
        password: {
            type: String,
            required: true
        },
        notifications: {
            type: [Notification],
            required: true
        },
        profilePic: {
            type: Image,
            required: false
        },
        bio: {
            type: String,
            required: false
        },
        friends: {
            type: [String],
            required: true
        },
        chats: {
            type: [String],
            required: true
        }
    }
);
```

Notification Model

```
const notificationSchema = new
Schema({
    _id: {
        type: ObjectId,
        required: true
    },
    senderId: {
        type: String,
        required: true
    },
    seen: {
        type: Boolean,
        required: true
    },
    notificationMsg: {
        type: String,
        required: true
    },
    sentAt: {
        type: String,
        required: true
    }
});
```

Image Model

```
const imageSchema = new Schema({
    _id: {
        type: ObjectId,
        required: true,
    },
    publicId: {
        type: String,
        required: false,
    },
    url: {
        type: String,
        required: false,
    }
});
```

Tileset Model

```
const tilesetSchema = new Schema({
    _id: {
        type: ObjectId,
        required: true
    },
    tilesetName: {
        type: String,
        required: true
    },
    tilesetDescription: {
        type: String,
        required: false
    },
    tilesetTags: {
        type: [String],
        required: false
    },
    tilesetBackgroundColor: {
        type: String,
        required: false
    },
    imagePixelWidth: {
        type: Number,
        required: true
    },
    imagePixelHeight: {
        type: Number,
        required: true
    },
    tileHeight: {
        type: Number,
        required: true
    },
   tileWidth: {
        type: Number,
        required: true
    },
    tiles: {
        type: [Tile],
        required: true
    },
    padding: {
        type: Number,
        required: false
    },
    source: {
        type: String,
        required: false
    },
    ownerId: {
      type: String,
      required: true
    },
    collaboratorIds: {
       type: [String],
       required: false
    },
    isPublic: {
      type: Boolean,
      required: true
    },
    isLocked: {
      type: Boolean,
      required: true
    }
});
```

Tile Model

```
const tileSchema = new Schema({
    _id: {
        type: ObjectId,
        required: true
    },
    tilesetId: {
        type: ObjectId,
        required: true
    },
   height : {
     type: Number,
     required: true
   },
   width: {
     type: Number,
     required: true
   }
});
```

Map Model

```
const mapSchema = new Schema({
    _id: {
        type: ObjectId,
        required: true
    },
    mapName: {
        type: String,
        required: true
    },
    mapDescription: {
        type: String,
        required: false
    },
    mapBackgroundColor: {
        type: String,
        required: true
    },
    mapHeight: {
        type: Number,
        required: true
    },
    mapWidth: {
        type: Number,
        required: true
    },
    tilesets: {
       type: [Tileset],
       required: false
    },
    tileHeight: {
       type: Number,
       required: true
    },
    tileWidth: {
       type: Number,
       required: false
    },
    ownerId: {
        type: String,
        required: true
      },
      collaboratorIds: {
         type: [String],
         required: false
      },
      isPublic: {
        type: Boolean,
        required: true
      },
      isLocked: {
        type: Boolean,
        required: true
      },
      layers: {
        type: [Layer],
        required: true
      }
});
```

Layer Model

```
const layerSchema = new Schema({
    _id: {
        type: ObjectId,
        required: true
    },
    layerName: {
        type: String,
        required: true
    },
    visible: {
        type: Boolean,
        required: true
    },
    opacity: {
       type: Number,
       required: true
    }
});
```

Thread Model

```
const threadSchema = new Schema({
    _id: {
        type: ObjectId,
        required: true
    },
    senderId: {
        type: String,
        required: true
    },
    threadText: {
        type: String,
        required: true
    },
    sentAt: {
        type: String,
        required: true
    },
    threadName: {
        type: String,
        required: true
    },
    replies: {
        type: [replies],
        required: false
   }
});
```

Replies Model

```
const repliesSchema = new Schema({
    _id: {
        type: ObjectId,
        required: true
    },
    senderId: {
        type: String,
        required: true
    },
    replyMsg: {
        type: String,
        required: true
    },
    sentAt: {
        type: String,
        required: true
    },
    replyingTo: {
        type: ObjectId,
        required: true
    },
    replies: {
        type: [replies],
        required: false
   }
});
```

Data Dictionary

Name	Description
bio	A short description of the user. May be left blank if the user so chooses.
chats	An array of chat ids which the current user is a part of.
collaboratorIds	An array of user ids that have edit access to a project.
email	The email address of the user.
firstName	The first name of the user.
friends	An array of user ids that the current user has added as friend.
height	The height of the tile.
id	Unique field that acts as the primary key.
imagePixelHeight	The height, in pixels, of the whole object.
imagePixelWidth	The width, in pixels, of the whole object.
isLocked	Determines if a project is editable by others. True if not editable, false if editable.
isPublic	Determines if a project is visible to others. True if visible, false if not.
lastName	The last name of the user.
layerName	The name of the layer.
layers	An array of Layers that make up a Map.
mapBackground Color	The default color of a certain tile coordinate. A color can be specified, or left unspecified/transparent.
mapDescription	The description of the map.

mapHeight	The height of the map given in tiles.
mapName	The name of the map.
mapWidth	The width of the map given in tiles.
mapSource	The external source file containing the map.
notifications	An array of notification objects (which represent a single notification) that the current user has received.
notificationMsg	The text/content of the notification.
opacity	The opacity of a layer in any project.
ownerId	The user ID associated with any project.
padding	The amount of space between each tile on a tileset.
password	The password of the user, used to login.
profilePic	The current profile picture of the user.
publicId	The public ID of the image that the users can see.
receiverId	The ID of the receiver of the chat message.
recieverSeen	Whether or not the receiver has seen the chat message.
replies	A list of replies that can form from replying to one thread.
replyingTo	The user ID associated with the message being replied to.
replyMsg	The text/content of the reply.
seen	True or false depending on if the user has interacted with the notification.
senderId	The ID of the chat message sender.
sentAt	A time string for when a message, thread post, or thread reply has been sent.

threadName	The title of the thread.
threadText	The description of the thread.
tileHeight	The height of the tileset given in pixels.
tileHeight	The height of the tileset given in pixels.
tiles	The array of tiles that are used in the tileset.
tilesetBackgroundColor	The default color of a tileset. If a pixel on a tile is not colored in, then this color is used. If unspecified, then the pixel is transparent.
tilesetDescription	A string description of a tileset.
tilesetId	The unique identifier number of a tileset.
tilesetName	The name of a tileset.
tilesets	An array of Tileset objects that are used in a Map.
tilesetSource	The external source file containing the tileset.
tilesetTags	An array of strings that categorize a tileset.
tileWidth	The width of the tileset given in pixels.
url	The url to the image.
username	The string that a user sets to log in with. Also acts as the user's display name.
visible	Whether or not the layer is visible .
width	The width of the tile.
x	The x position of the tile.
y	The y position of the tile.

Domain Analysis Model

1. Similar Problems/Projects

Wireframer :
Full stack ***MERN*** web application for ***creating*** and ***editing*** user interface mockups. Similar philosophy of ***collaboration*** with multiple people working on the same project. ***Sharing*** work with others and ***saving*** it for later use while also ***exporting*** projects to different formats. Wireframer also supports ***notifications*** as well as profile pictures for user accounts, which are stored in the ***cloudinary*** database. Another notable feature is ***dragging & dropping*** components in the editor.

Wireframer and Pieces has many overlapping goals namely to edit & create projects, collaborate and share with others, import and export work, user authentication, persistent data etc. Pieces will take a lot of inspiration from Wireframer, while also using an almost identical tech stack.

Top5Lister :
Another full stack MERN web application. Top5lister, similar to Pieces, supports ***user account management*** while facilitating ***creation*** and ***editing*** of projects with features like ***saving*** data for later use, ***undo & redo, sorting, filtering*** etc. Another commonality between both applications is having a tab with ***public*** work (projects/lists) that users can interact with such as ***viewing, liking, disliking, commenting*** etc.

Pieces intends to utilize a lot of the Top5Lister structure/philosophy, since both share almost identical big picture objectives. This project acts as a great base to build on top of.

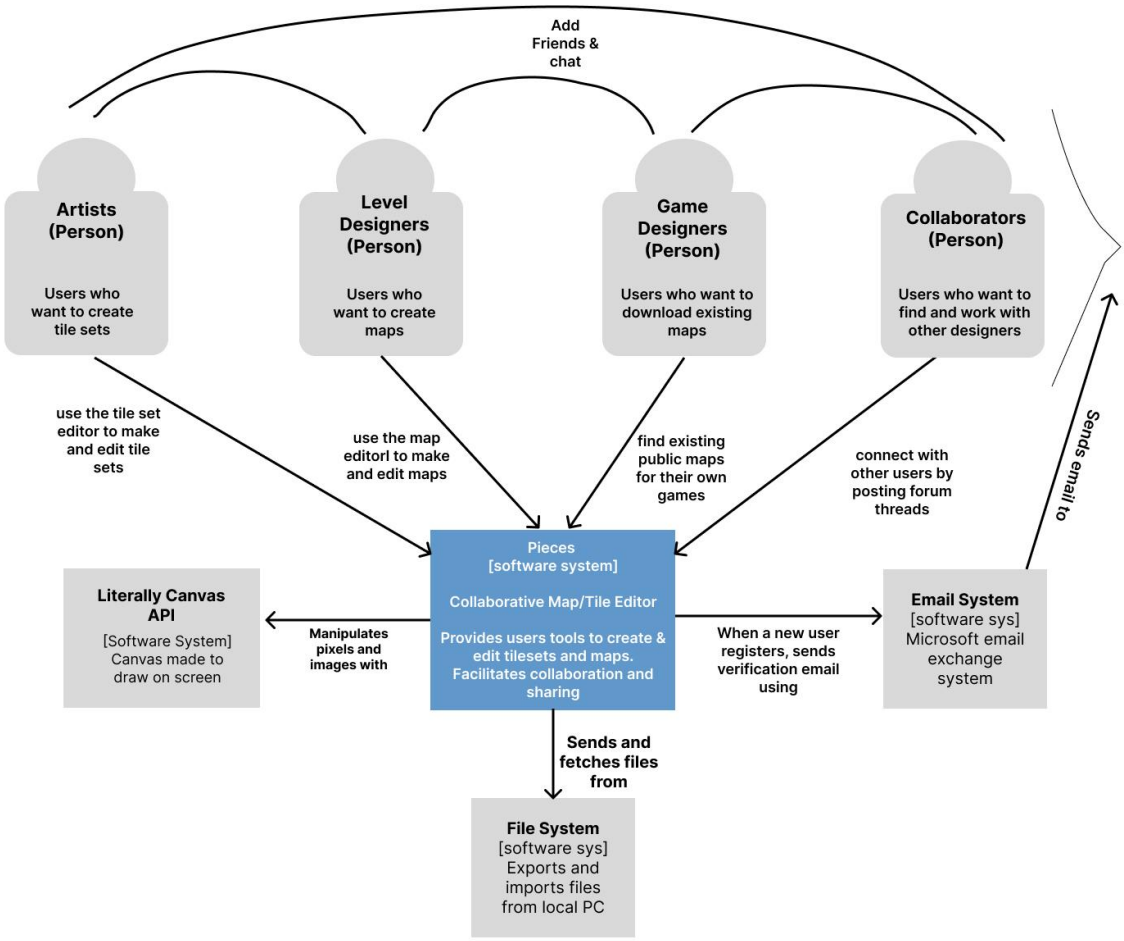
Tiled:
Tiled is a free application used for creating maps intended for 2D games. While Pieces offers much more than a **map editor**, a big part of our goal is to offer an easy-to-use yet effective map editor. While Tiled has its good points, we believe we can learn from our own experiences using Tiled to create a better map editor. We will still offer key tools that Tiled has like a **brush, bucket, eraser, dropper, and more**. We will also keep the ability to separate a project into different **layers** for users that may want to separate different aspects of their map. While Tiled doesn't offer a tileset editor, much of the key points taken from their map editor can also be applied to our own tileset editor.

2. Complete Technology Set

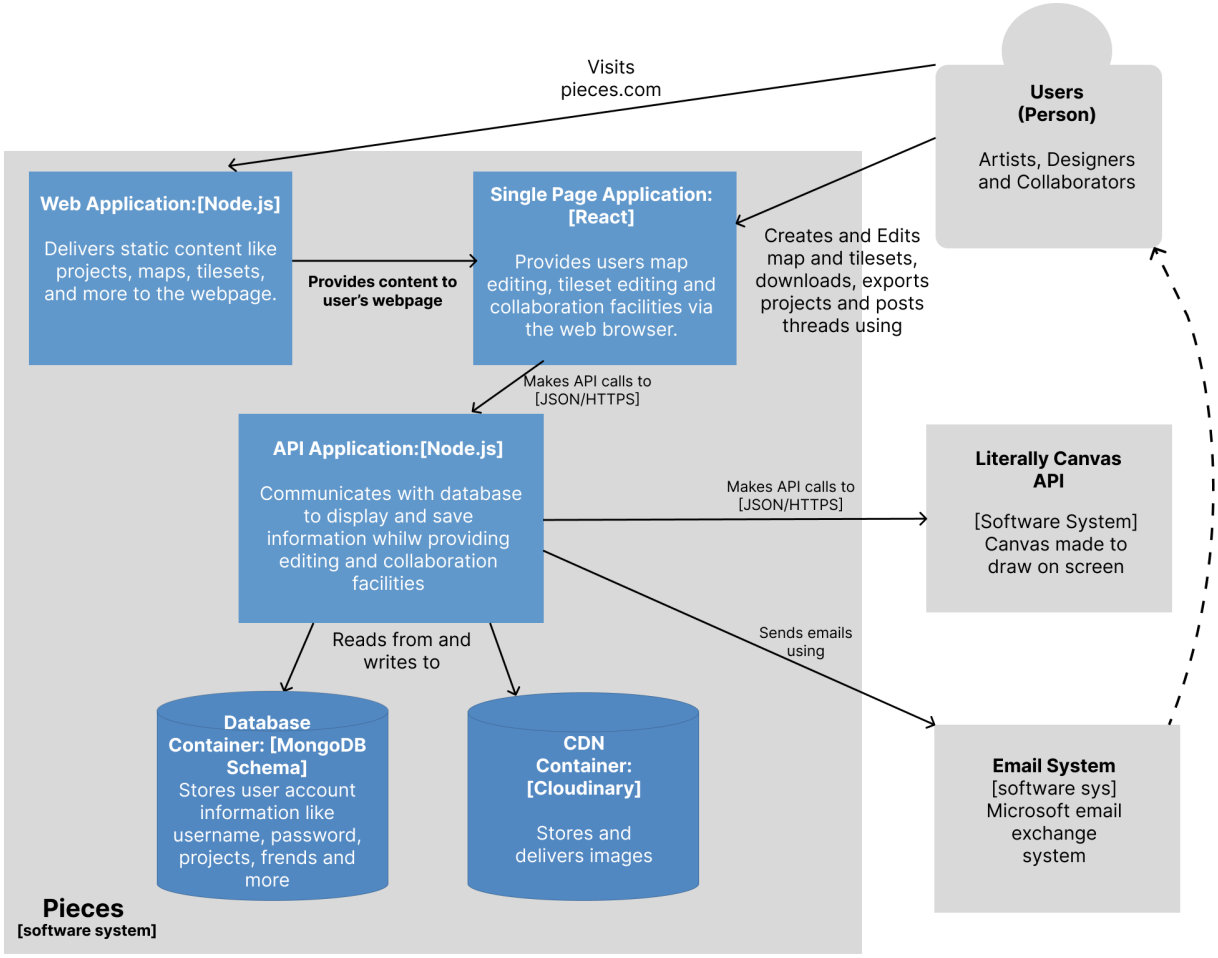
- **MongoDB Atlas** - NoSQL database to save/store data about users and projects
- **Node.js** - backend API services
- **Express** - Middleware with node; helps create web applications and manage services and routes.
- **React** - front end components
- **Material UI** - icons and customizable components
- **JavaScript** - client side to dynamically update ui, and server side to provide services.
- **HTML/CSS** - UI content and styling
- **React-rnd** - to drag and drop react components
- **Node-mailer** - to send emails to verify accounts
- **Bcryptjs** - to hash passwords
- **Literally Canvas API** - to create and edit a canvas for tileset editing and map editing
- **Cloudinary** - database to save images for user profile pictures.

3. Architecture

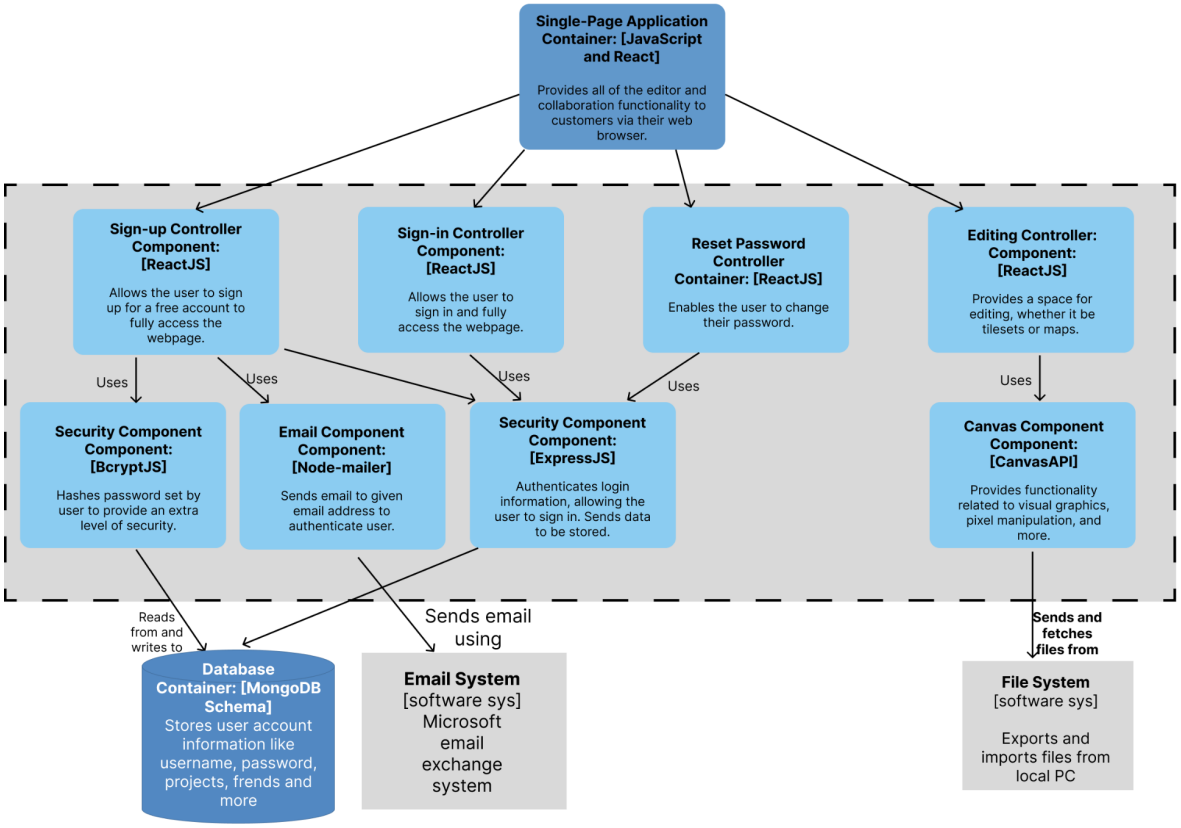
System Context Diagram



Container Diagram



Component Diagram



4. Training Verification

Ahnaf Hasan

- **ToDoTracker**
  - **Time:** Spring 2021
  - **Desc:** An application to create todo lists.
  - **Technologies:** MongoDB, React, Node.js, JavaScript, HTML, CSS

Iman Ali

- **Wireframer**
  - **Time:** Aug 2021 - present
  - **Desc:** A simple yet powerful wireframing application to create & edit UI mockups.
  - **Technologies:** React, Node, Express, MongoDB Atlas, JavaScript, HTML/CSS, Material UI, Cloudinary, React-rnd, Node-Mailer, GraphQL, Apollo.

- **WolfieTools FrontEnd API**
  - **Time:** Aug 2021 - present
  - **Desc:** A frontend API for customizable components and to stylize beautiful, interactive, and responsive user interfaces.
  - **Technologies:** Node.js, React, Dart Sass, JavaScript, HTML/CSS, gulp.js
- **ToDoTracker**
  - **Time:** Aug 2021 - present
  - **Desc:** An application for creating todo lists with features such as undo, redo, sorting, editing and reordering.
  - **Technologies:** Mongo Atlas, Ex press, React, Node, JavaScript, HTML/CSS, Apollo and GraphQL

Tommy Lin

- **Puzzle Knight**
  - **Time:** Spring 2021
  - **Desc:** Final project for CSE380. While working on this project, I gained a lot of experience using map editors and tileset editors that I can apply to Pieces.
  - **Technologies:** Tiled, Typescript, gulp.js, Pixilart
- **ToDoTracker (Github: tommylin121314)**

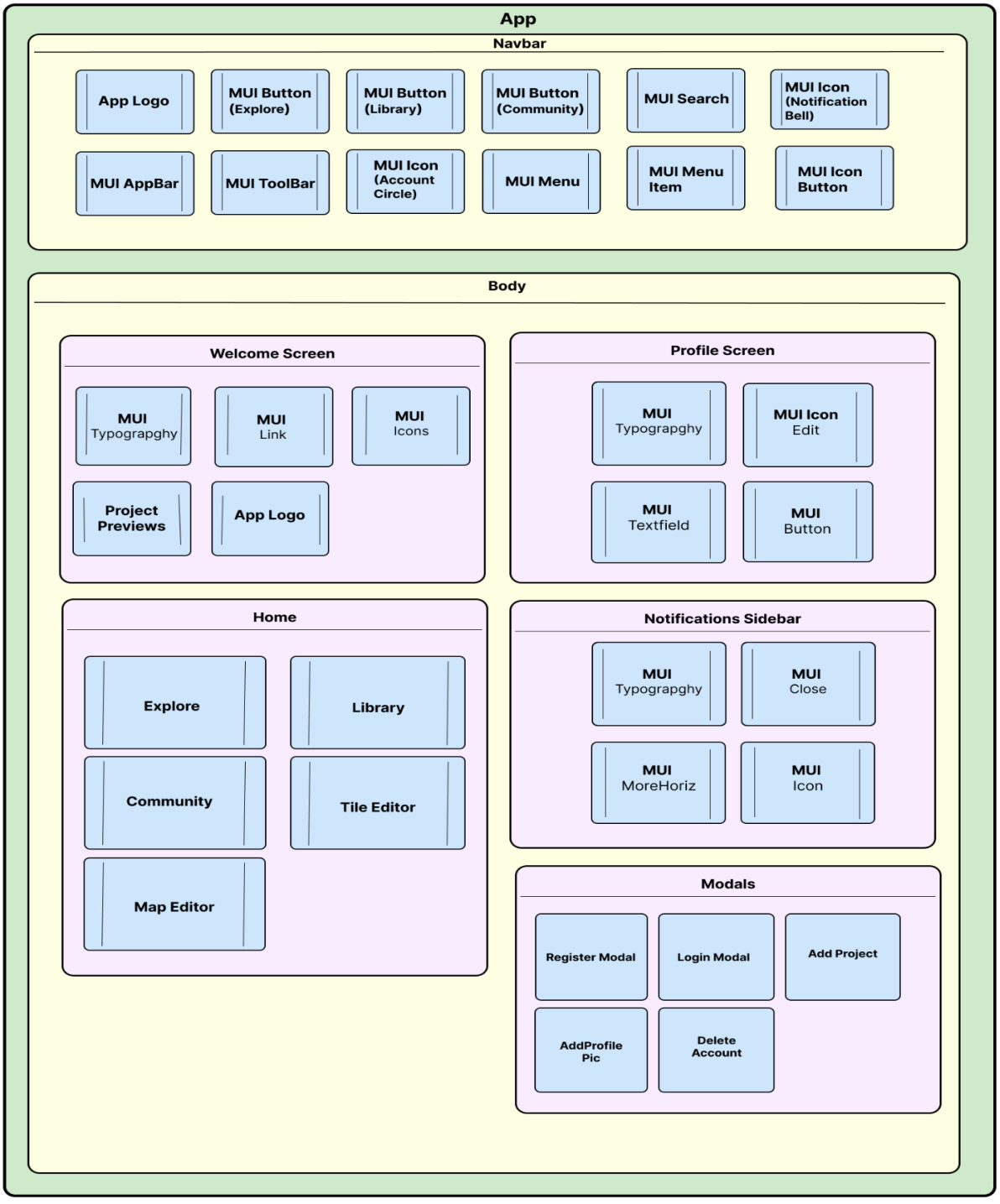
    - **Time:** Spring 2021
    - **Desc:** To Do List web application
    - **Technologies:** MongoDB, React, Node.js, Javascript, HTML, CSS

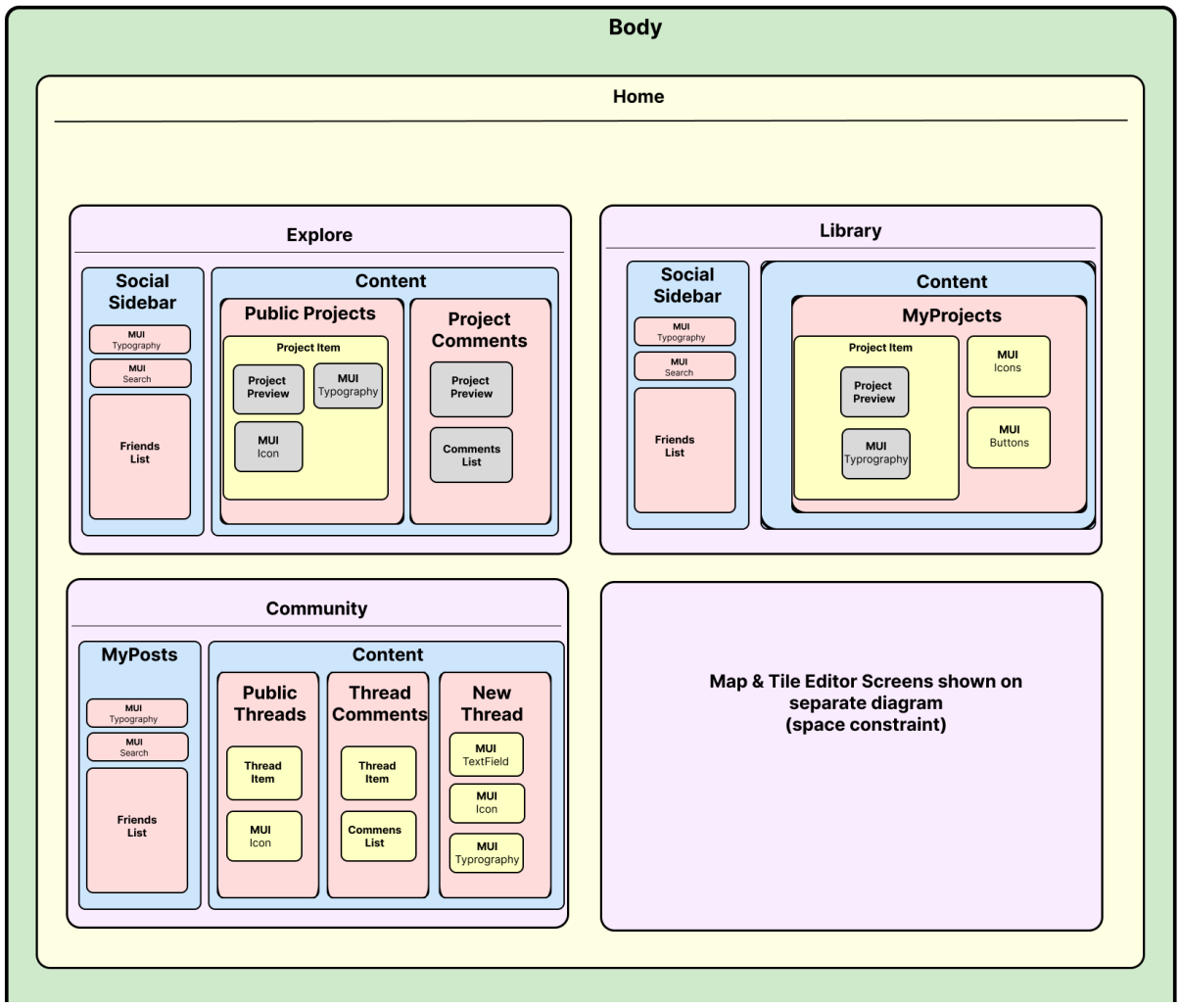
Vincent Chi

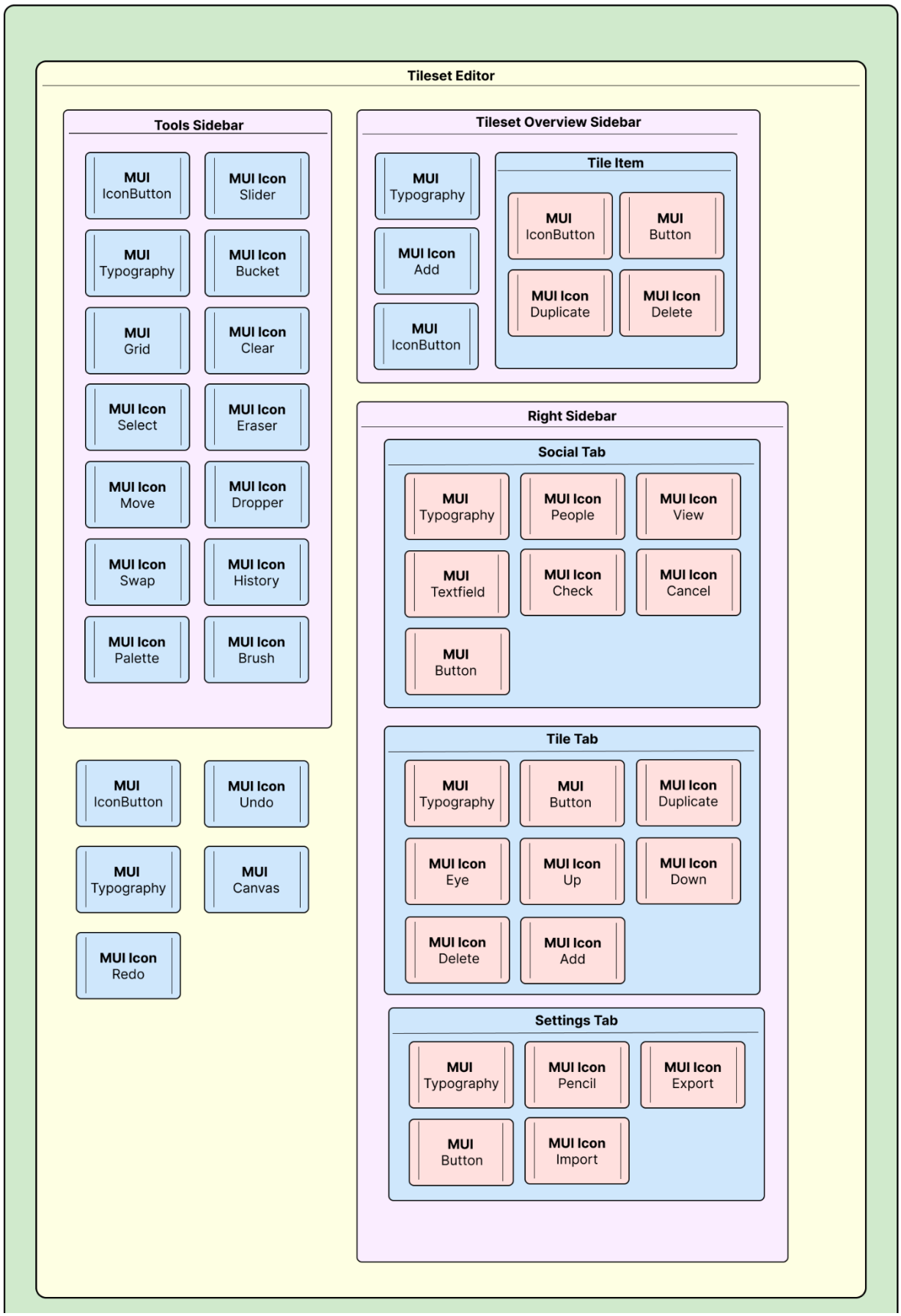
- Training Completed: CSE 316
    - **ToDoTracker**
        - **Time:** Spring 2021
        - **Desc:** An application for creating todo lists and items. Has undo, redo, and sort functionality.
        - **Technologies:** MongoDB, Express, React, Node.js, Javascript, HTML, CSS

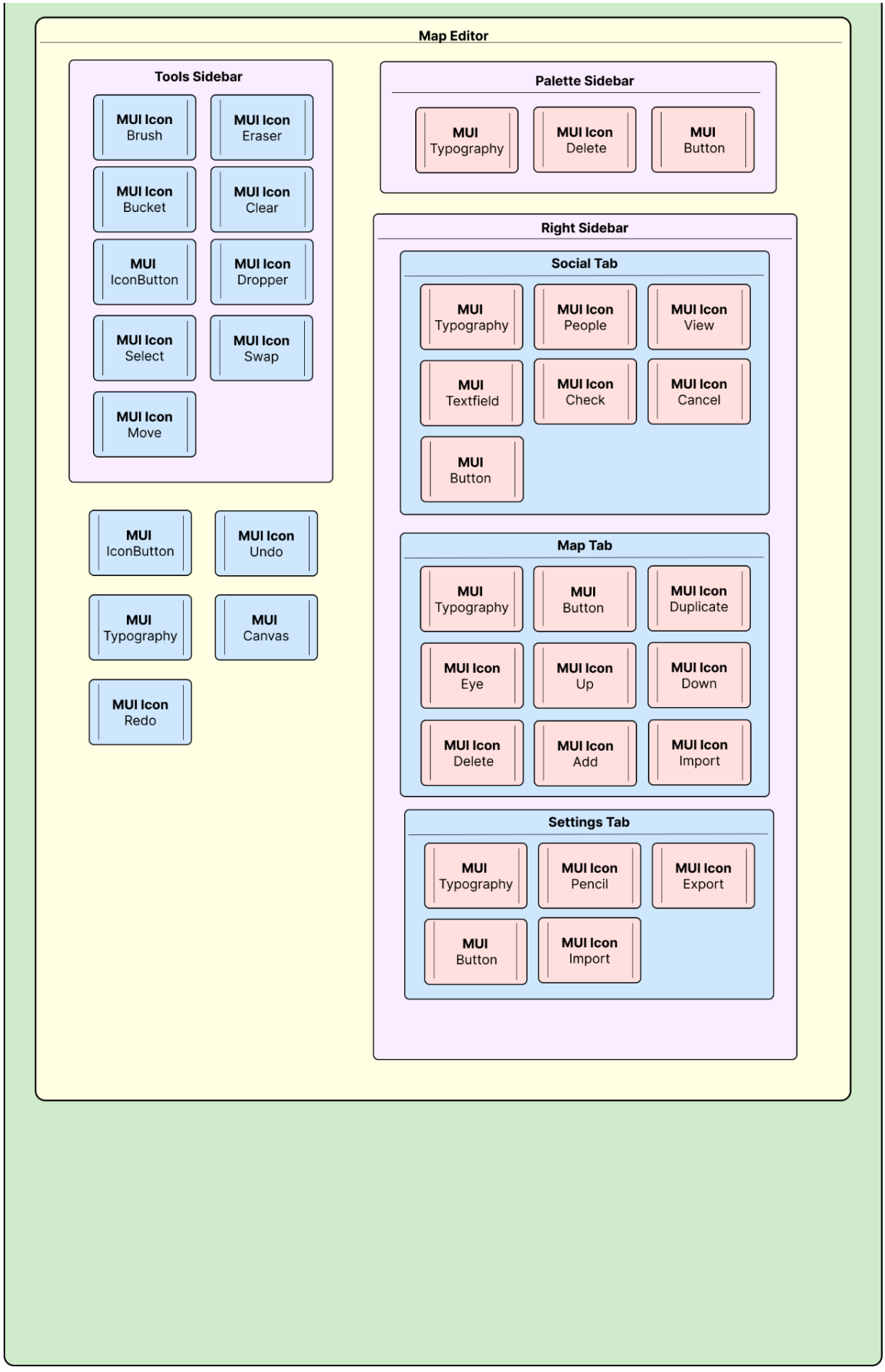
Software Model

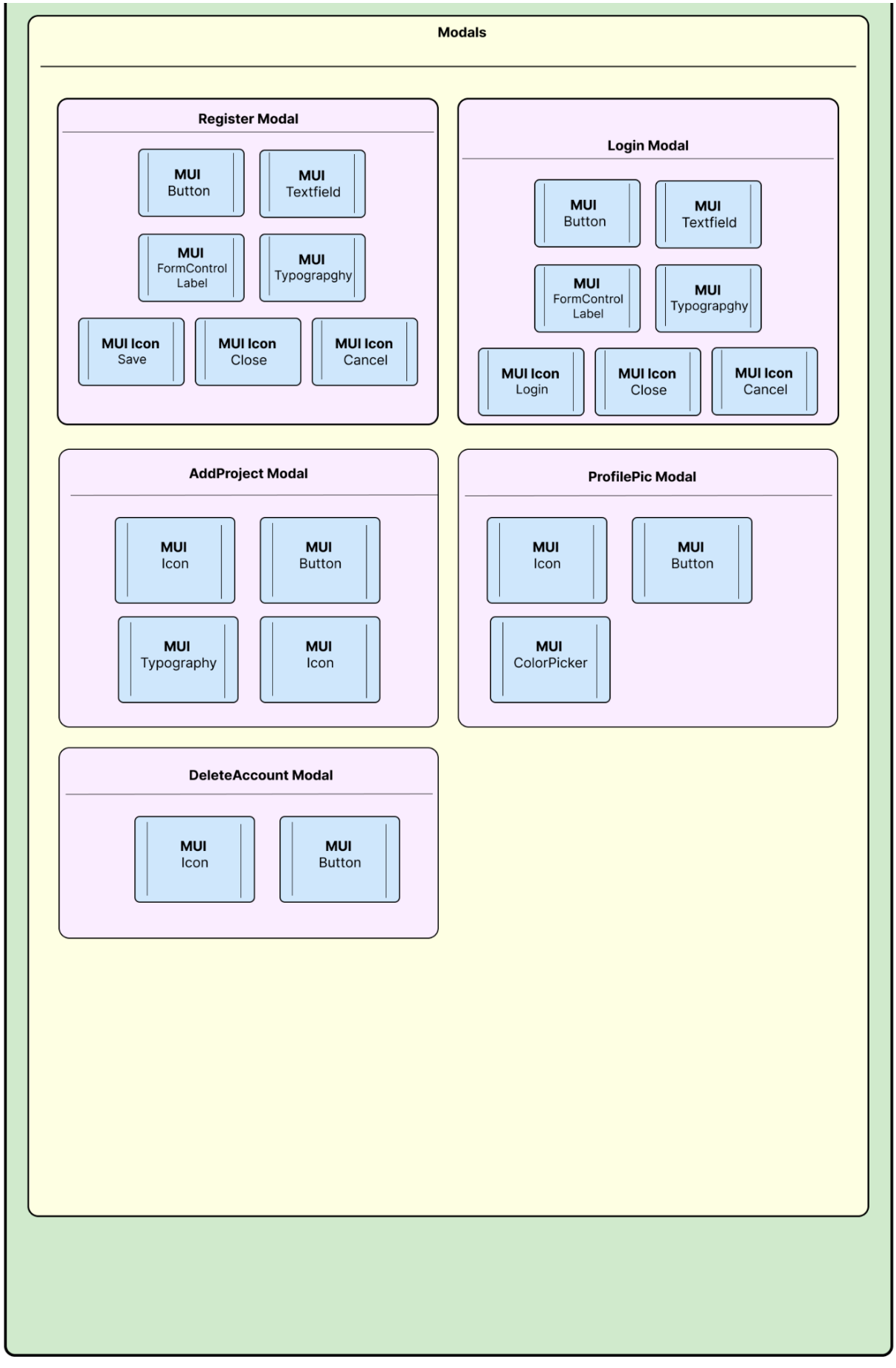
Front-End Component Composition




Body
Home
Explore
Social Sidebar
MUI Typography
MUI Search
Friends List
Content
Public Projects
Project Item
Project Preview
MUI Typography
MUI Icon
Project Comments
Project Preview
Comments List
Library
Social Sidebar
MUI Typography
MUI Search
Friends List
Content
MyProjects
Project Item
Project Preview
MUI Typography
MUI Icons
MUI Buttons
Community
MyPosts
MUI Typography
MUI Search
Friends List
Content
Public Threads
Thread Item
MUI Icon
Thread Comments
Thread Item
Commens List
New Thread
MUI TextField
MUI Icon
MUI Typrography
Map & Tile Editor Screens shown on separate diagram (space constraint)

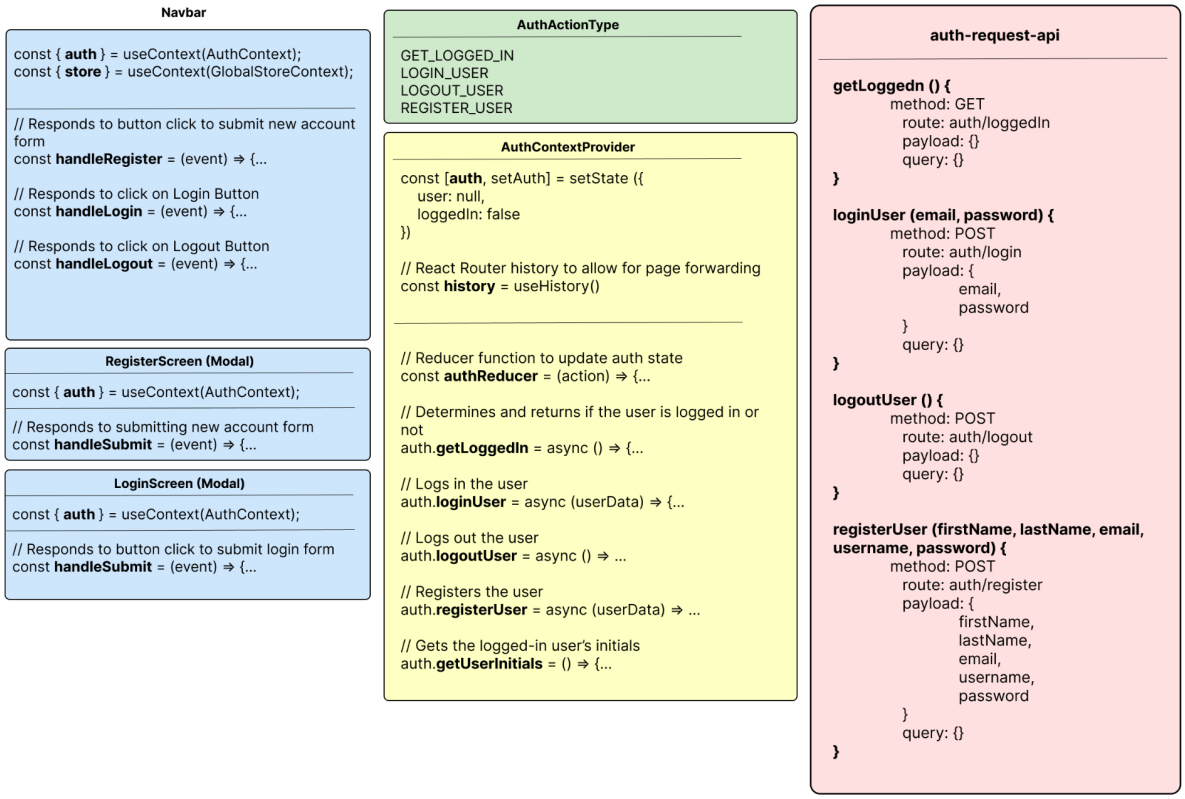

Tileset Editor
Tools Sidebar
MUI IconButton
MUI Icon Slider
MUI Typography
MUI Icon Bucket
MUI Grid
MUI Icon Clear
MUI Icon Select
MUI Icon Eraser
MUI Icon Move
MUI Icon Dropper
MUI Icon Swap
MUI Icon History
MUI Icon Palette
MUI Icon Brush
MUI IconButton
MUI Icon Undo
MUI Typography
MUI Canvas
MUI Icon Redo
Tileset Overview Sidebar
MUI Typography
MUI Icon Add
MUI IconButton
Tile Item
MUI IconButton
MUI Button
MUI Icon Duplicate
MUI Icon Delete
Right Sidebar
Social Tab
MUI Typography
MUI Icon People
MUI Icon View
MUI Textfield
MUI Icon Check
MUI Icon Cancel
MUI Button
Tile Tab
MUI Typography
MUI Button
MUI Icon Duplicate
MUI Icon Eye
MUI Icon Up
MUI Icon Down
MUI Icon Delete
MUI Icon Add
Settings Tab
MUI Typography
MUI Icon Pencil
MUI Icon Export
MUI Button
MUI Icon Import


Map Editor
Tools Sidebar
MUI Icon Brush
MUI Icon Eraser
MUI Icon Bucket
MUI Icon Clear
MUI IconButton
MUI Icon Dropper
MUI Icon Select
MUI Icon Swap
MUI Icon Move
MUI IconButton
MUI Icon Undo
MUI Typography
MUI Canvas
MUI Icon Redo
Palette Sidebar
MUI Typography
MUI Icon Delete
MUI Button
Right Sidebar
Social Tab
MUI Typography
MUI Icon People
MUI Icon View
MUI Textfield
MUI Icon Check
MUI Icon Cancel
MUI Button
Map Tab
MUI Typography
MUI Button
MUI Icon Duplicate
MUI Icon Eye
MUI Icon Up
MUI Icon Down
MUI Icon Delete
MUI Icon Add
MUI Icon Import
Settings Tab
MUI Typography
MUI Icon Pencil
MUI Icon Export
MUI Button
MUI Icon Import


Modals
Register Modal
MUI Button
MUI Textfield
MUI FormControl Label
MUI Typograpghy
MUI Icon Save
MUI Icon Close
MUI Icon Cancel
Login Modal
MUI Button
MUI Textfield
MUI FormControl Label
MUI Typograpghy
MUI Icon Login
MUI Icon Close
MUI Icon Cancel
AddProject Modal
MUI Icon
MUI Button
MUI Typography
MUI Icon
ProfilePic Modal
MUI Icon
MUI Button
MUI ColorPicker
DeleteAccount Modal
MUI Icon
MUI Button

Front-End Component Design & Back-End API

Auth Client



Auth Server

AuthRouter /auth

```
// Handles request if user logged in or not
router.get( '/loggedin' , AuthController.loggedIn)

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

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

// Handles new user register requests
router.post( '/register' , AuthController.register)
```

AuthController

```
loggedIn (req, res) {
        method: GET
        route: auth/loggedIn
        status 200 (OK) response : {
            payload: {
                loggedIn (true/false)
                user ({firstName..}/ null)
            }
        }
        status 400 (BAD REQUEST) response : {
            payload: { errorMsg }
        }
}


login (req, res) {
        method: POST
        route: auth/register
        status 200 (OK) response : {
            cookie: set token
            payload: {
                loggedIn (true/false)
                user ({firstName..}/ null)
            }
        }
        status 400 (BAD REQUEST) response : {
            payload: { errorMsg }
        }
        status 401 (Unathourized) response : {
            payload: { errorMsg }
        }
}


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

Register (req, res) {
        method: GET
        route: auth/register
        status 200 (OK) response : {
            payload: {
                user ({firstName..}/ null)
            }
        }

        status 400 (BAD REQUEST) response : {
            payload: { errorMsg }
        }
}
```

User Schema

```
id: ObjectId
firstName: String
lastName: String
email: String
username: String
password: String
notifications: [Notification]
profilePic: Image
bio: String
friends: [String]
chats: [String]
```

AuthManager

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

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

// for verifying a user
verifyUser = (req) => { ...
```

Store Client

GlobalStoreContextProvider

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

}

const tps : jsTPS

//Checks if an action can be redone
store.canRedo = function()...

//Checks if an action can be undone
store.canUndo = function()...

//Redos the last redo-able action
store.redo = function()...

//Undos the undo-able last action
store.undo = function()...

//Places a color pixel onto the canvas
store.placeColor = function()...

//Erases a color pixel from the canvas
store.eraseColor = function()...

//Fills all empty pixels in contact with cursor position
store.fillColor = function()...

//Switches the primary color with the secondary color
store.swapColor = function()...

//Sets the primary color
store.setColor = function()...

//Places a tile onto the canvas
store.placeTile = function()...

//Erases a tile from the canvas
store.eraseTile = function()...

//Fills all empty tiles in contact with cursor position
store.fillTile = function()...

//Switches the primary tile with the secondary tile
store.swapTile = function()...

//Sets the primary tile
store.setTile = function()...

//Clears the canvas
store.clearCanvas = function()...
```

ProjectsList

```
const [hoverProject, setHoverProject] = useState{}
const [activeProject, setActiveProject] = useState()
let projects = [];
   const { data, refetch } = useQuery(GET_DB_PROJECTS);
   if (data) {
      projects = data.getAllProjects;
   }

//Responds to action on-click event based on current tool
const handleClick = (event) {}

//Responds to start of hover action
const handleHoverStart = (event, projectId) {}

//Repsponds to the end of a hover action
const handleHoverEnd = (event, projectId) {}
```

ProjectsListItem

```
const {data} = useState{}
const [editing, setEditing] = useState(false)

//Edits the information of the project
const editProjectData = (event, projectId) {}

//Responds to the clicking of a project
const handleClick = (event, projectId) {}
```

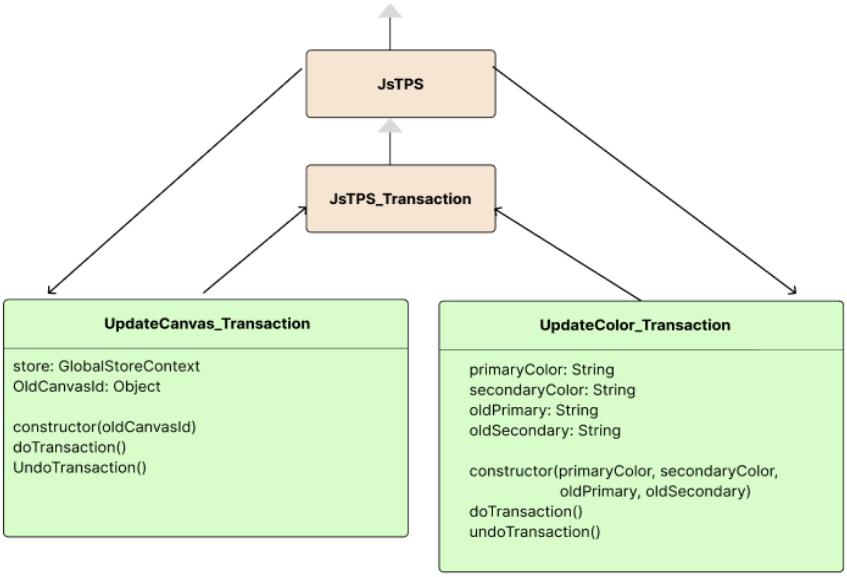
FriendsList

```
const [currentFriends, setFriends] = useState()
let friends = [];
   const { data, refetch } = useQuery(GET_DB_FRIENDS);
   if (data) {
      friends = data.getAllFriends;
   }

// Responds to action on-click event based on current tool
const handleClick = (event) {}

// Responds to on-click event of the add friend icon
const handleAddFriend = (event, userId) {}

// Responds to on-click event of the remove friend icon
const handleRemoveFriend = (event, userId) {}
```



Canvas

```
const {store} = useContext(GlobalStoreContext);
const [currentTool, setTool] = useState(0);
const [primary, setPrimary] = useState(0);
const [secondary, setSecondary] = useState(0);
```

```
//Responds to action on-click event based on current tool
const handleClick = (event) {}

//Responds to action key-press event
const handleKeyPress = (event) {}

//Responds to the start of a drag event
const handleDragStart = (event) {}

//Responds to the end of a drag event
const handleDragEnd = (event) {}

//Places a pixel/tile at cursor position onto the canvas
const place = (event) {}

//Erases a pixel/tile at cursor position from the canvas
const erase = (event) {}

//Clears the whole canvas
const clear = (event) {}
```

RepliesList

```
let replies = [];
   const { data, refetch } = useQuery(GET_DB_REPLIES);
   if (data) {
     replies = data.getAllReplies;
   }
```

```
//Responds to action on-click event based on current tool
const handleClick = (event) {}

//Edits reply textfield
const editReply = (event) {}
```

ThreadsList

```
const [activeThread, setActiveThread] = useState{}
let threads = [];
   const { data, refetch } = useQuery(GET_DB_THREADS);
   if (data) {
      threads = data.getAllThreads;
   }
```

```
//Responds to action on-click event based on current tool
const handleClick = (event) {}

//Adds new thread
const createThread = (event, userId) {}

//Opens specific thread
const openThread = (event) {}

//Upvotes or downvotes a thread
const voteThread = (event) {}
```

ThreadsListItem

```
const {data} = useState{}
const [editing, setEditing] = useState(false)
```

```
//Edits the information of the project
const editProjectData = (event, projectId) {}

//Responds to the clicking of a project
const handleClick = (event, projectId) {}
```

RepliesListItem

```
const {data} = useState{}
const [editing, setEditing] = useState(false)
```

```
//Edits the information of the project
const editProjectData = (event, projectId) {}

//Responds to the clicking of a project
const handleClick = (event, projectId) {}
```

Database Design

Our Express Server

```
databaseManager = require("MongoDBManager");
```

GlobalStoreController

```
databaseManager = require("MongoDBManager");

createTileset(req, res) {
   if( IMPROPERLY FORMATTED REQUEST ) {
       // SEND 400 RESPONSE CODE
       payload: { errorMessage }
   }
   let newTileset = databaseManager.createTileset(...);
   if (newTile) {
       // SEND 201 RESPONSE CODE
      payload: {
         Tile( {id, ... } )
      }
   } else {
      // SEND 400 RESPONSE CODE
      payload: { errorMessage }
   }
}

deleteTileset(req, res) {
   method: DELETE
   route: /store/deletetileset
   status 200 (OK) response: {
    // PROPERLY FORMATTED LEGAL REQUEST
    payload: { }
   }
   status 400 (BAD REQUEST) response: {
    // IMPROPERLY FORMATTED REQUEST
    payload: { errorMessage }
   }
   status 403 (FORBIDDEN) response: {
    // PROPERLY FORMATTED BUT NOT OWNED BY USER
    payload: { errorMessage }
   }
   status 404 (NOT FOUND) response: {
    // PROPERLY FORMATTED LEGAL BUT NOT FOUND
    payload: { errorMessage }
   }
}

saveTileset(req, res) {
    method: POST
    route: /store/save
    status 200 (OK) response: {
     // PROPERLY FORMATTED LEGAL REQUEST
     payload: { Tileset(currentState) }
    }
    status 400 (BAD REQUEST) response: {
     // IMPROPERLY FORMATTED REQUEST
     payload: { errorMessage }
    }
    status 403 (FORBIDDEN) response: {
    // PROPERLY FORMATTED BUT USER NOT ALLOWED
    payload: { errorMessage }
    }
    status 404 (NOT FOUND) response: {
    // PROPERLY FORMATTED LEGAL BUT NOT FOUND
    payload: { errorMessage }
    }
}

getTileset(req, res) {
   method: GET
   route: /store/findtileset
   status 200 (OK) response: {
    // PROPERLY FORMATTED REQUEST
    payload: { Tileset }
   }
   status 400 (BAD REQUEST) response: {
    // IMPROPERLY FORMATTED REQUEST
    payload: { errorMessage }
   }
  status 404 (NOT FOUND) response: {
   // PROPERLY FORMATTED LEGAL BUT NOT FOUND
   payload: { errorMessage }
  }
}

postThread(req, res) {
    method: PUT
    route: /store/post
    status 201 (CREATED) response: {
     // PROPERLY FORMATTED LEGAL REQUEST
     payload: { Thread(threadId) }
    }
    status 400 (BAD REQUEST) response: {
     // IMPROPERLY FORMATTED REQUEST
     payload: { errorMessage }
    }
    status 401 (UNAUTHORIZED) response: {
     // USER NOT LOGGED IN
     payload: { errorMessage }
    }
}

deleteThread(req, res) {
    method: DELETE
    route: /store/removethread
    status 200 (OK) response: {
     // PROPERLY FORMATTED REQUEST
     payload: { }
    }
    status 400 (BAD REQUEST) response: {
     // IMPROPERLY FORMATTED REQUEST
     payload: { errorMessage }
    }
    status 401 (UNAUTHORIZED) response: {
     // USER NOT LOGGED IN
     payload: { errorMessage }
    }
    status 403 (FORBIDDEN) response: {
    // PROPERLY FORMATTED BUT USER NOT ALLOWED
    payload: { errorMessage }
    }
    status 404 (NOT FOUND) response: {
    // PROPERLY FORMATTED LEGAL BUT NOT FOUND
    payload: { errorMessage }
    }
}
```

MongoDBManager
<specialized Implementation>

```
createTileset(ownerId) {
   let newTileset = new Tileset(ownerId);
   newTileset.save();
   ... // GIVE TILESET AN ID
   if (success) {
     return newTileset;
   }
}

deleteTileset(ownerId, tilesetId) {
   let foundTileset = Tileset.findTileset(tilesetId);
   if (!foundTileset) return 404; // TILESET NOT FOUND
   if (foundTileset.ownerId == ownerId) {
      ... // HOUSE KEEPING
      Tileset.delete(foundTileset);
      return 200;
   } else {
      // UNAUTHORIZED
      return 403;
   }
}

saveTileset(userId, tilesetId) {
  let foundTileset = Tileset.findTileset(tilesetId);
  if (!foundTileset) return 404; // TILESET NOT FOUND
  if (userId not in foundTileset.collaboratorIds) {
    // USER NO LONGER HAS PERMISSION TO EDIT
    return 403;
  }
  foundTileset.save();
  return foundTileset;
}

getTileset(tilesetId) {
  let foundTileset = null;
  ... // SEARCH DATABASE FOR TILESET
  return foundTileset;
}

postThread(authorId, threadTitle, threadContent) {
 let newThread = new Thread(threadTitle, threadContent, authorId);
 newThread.save();
 ... // GIVE THREAD AN ID
 return newThread;
}

deleteThread(userId, threadId) {
 let foundThread = Thread.findThread(threadId);
 if (!foundThread) return 404; // THREAD NOT FOUND
 if (foundThread.ownerId != userId) {
   // USER NOT THREAD'S OWNER
   return 403;
 }
 ... // HOUSE KEEPING
 Thread.delete(foundThread);
 return 200;
}
```

Tile
<using Mongoose ODM>

```
const TileSchema = new Schema(
   {
      id: { type: ObjectId, required: true },
      tilesetId: { type: ObjectId, required: true },
      height: { type: Number, required: true },
      width: { type: Number, required: true }
   },
   { timestamps: true }
);

module.exports = mongoose.model(Tile, TileSchema);
```

Tileset
<using Mongoose ODM>

```
const TilesetSchema = new Schema(
   {
      id: { type: ObjectId, required: true },
      tilesetName: { type: String, required: false },
      tilesetDescription: { type: String, required: false },
      tilesetTags: { type: [String], required: false },
      tilesetBackgroundColor: { type: String, required: false },
      imagePixelWidth: { type: Number, required: true },
      imagePixelHeight: { type: Number, required: true },
      tileHeight: { type: Number, required: true },
      tileWidth: { type: Number, required: true },
      tiles: { type: [Tile], required: false },
      padding: { type: [Number], required: false },
      source: { type: String, required: true },
      ownerId: { type: String, required: true },
      collaboratorIds: { type: [String], required: false },
      isPublic: { type: Boolean, required: true },
      isLocked: { type: Boolean, required: true }
   },
   { timestamps: true }
);

module.exports = mongoose.model(Tileset, TilesetSchema);
```

Thread
<using Mongoose ODM>

```
const ThreadSchema = new Schema(
   {
      id: { type: ObjectId, required: true },
      senderId: { type: String, required: true },
      threadText: { type: String, required: false },
      sentAt: { type: String, required: true },
      threadName: { type: String, required: true },
      replies: { type: [Replies], required: false }
   },
   { timestamps: true }
);

module.exports = mongoose.model(Thread, ThreadSchema);
```

Chat
<using Mongoose ODM>

```
const ChatSchema = new Schema(
   {
      id: { type: ObjectId, required: true },
      senderId: { type: String, required: true },
      receiverId: { type: String, required: true },
      receiverSeen: { type: String, required: true },
      message: { type: String, required: true },
      sentAt: { type: String, required: true }
   },
   { timestamps: true }
);

module.exports = mongoose.model(Chat, ChatSchema);
```

Layer
<using Mongoose ODM>

```
const LayerSchema = new Schema(
   {
      id: { type: ObjectId, required: true },
      layerName: { type: String, required: false },
      visible: { type: Boolean, required: true },
      opacity: { type: Number, required: true }
   },
   { timestamps: true }
);

module.exports = mongoose.model(Layer, LayerSchema);
```

Replies
<using Mongoose ODM>

```
const RepliesSchema = new Schema(
    {
        id: { type: ObjectId, required: true },
        senderId: { type: String, required: true },
        replyMessage: { type: String, required: true },
        sentAt: { type: String, required: true },
        replyingTo: { type: String, required: false },
        replies: { type: [Replies], required: false }
    },
    { timestamps: true }
);

module.exports = mongoose.model(Replies, RepliesSchema);
```

Map
<using Mongoose ODM>

```
const MapSchema = new Schema(
    {
        id: { type: ObjectId, required: true },
        mapName: { type: String, required: false },
        mapDescription: { type: String, required: false },
        mapTags: { type: [String], required: false },
        mapBackgroundColor: { type: String, required: false },
        mapWidth: { type: Number, required: true },
        mapHeight: { type: Number, required: true },
        tileHeight: { type: Number, required: true },
        tileWidth: { type: Number, required: true },
        tilesets: { type: [Tileset], required: false },
        layers: { type: [Layer], required: true },
        ownerId: { type: String, required: true },
        collaboratorIds: { type: [String], required: false },
        isPublic: { type: Boolean, required: true },
        isLocked: { type: Boolean, required: true }
    },
    { timestamps: true }
);

module.exports = mongoose.model(Map, MapSchema);
```

Image
<using Mongoose ODM>

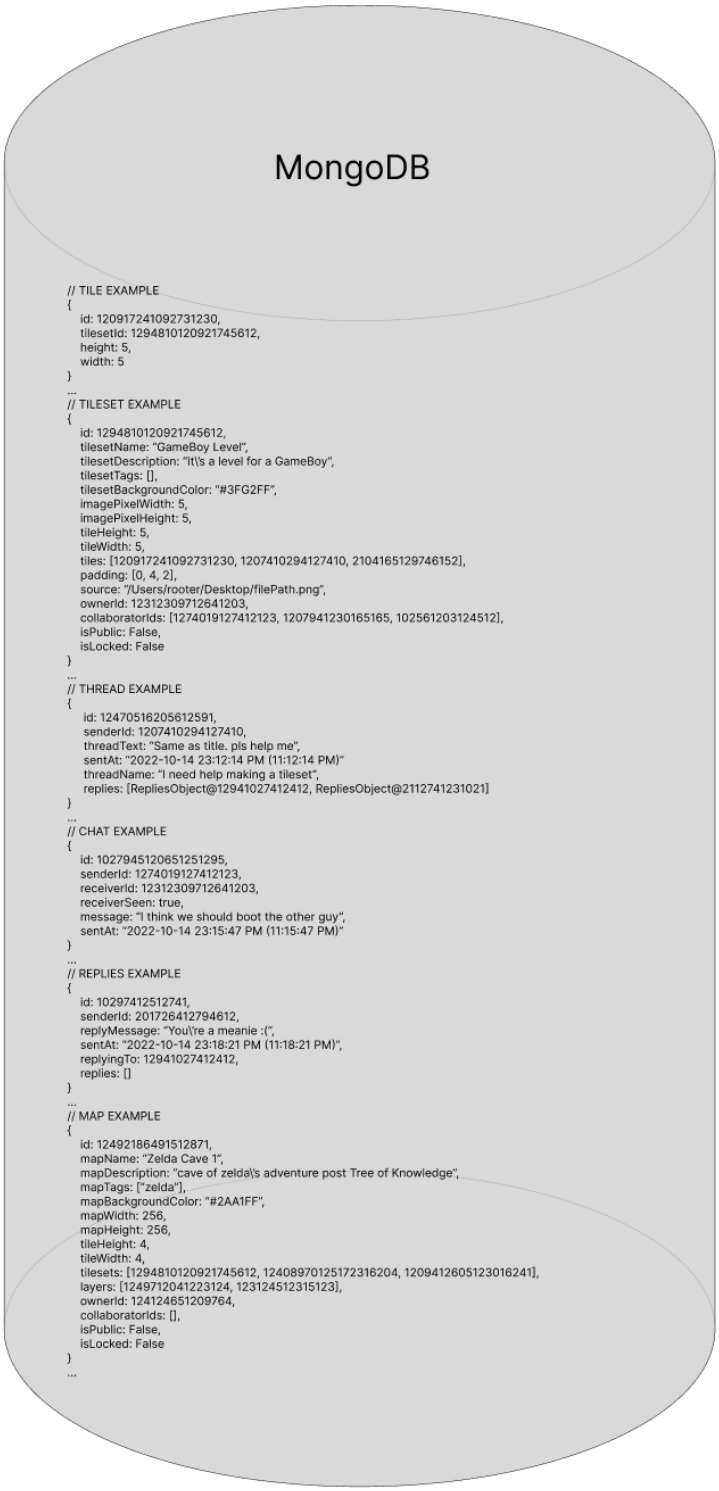
```
const ImageSchema = new Schema(
    {
        id: { type: ObjectId, required: true },
        publicId: { type: ObjectId, required: true },
        url: { type: String, required: true }
    },
    { timestamps: true }
);

module.exports = mongoose.model(Image, ImageSchema);
```

Notification
<using Mongoose ODM>

```
const NotificationSchema = new Schema(
    {
        id: { type: ObjectId, required: true },
        senderId: { type: ObjectId, required: true },
        seen: { type: String, required: true },
        notificationMsg: { type: String, required: true },
        sentAt: { type: String, required: true }
    },
    { timestamps: true }
);

module.exports = mongoose.model(Notification,
NotificationSchema);
```



Progress Reviews

Build 1:

For build #1, we focused on getting the back-end and deployment to work. This meant creating the app, deploying to heroku, setting up MongoDB Atlas, and writing any necessary code to connect the front-end to the back-end like controller functions and schema types. We were able to complete build #1 without much complication. The biggest challenge was figuring out how to deploy to Heroku, but that was eventually figured out.

Build 2:

In build #2, we decided to get the main screens of our application working. These screens included the Welcome screen, the Explore screen, and the Library screen, and the Editors. We added statistics and welcome messages on the Welcome screen where users initially land on. The Explore and Library screens display projects and include features like sorting, filtering, and pagination. We also created the front-end for the tileset and map editors, and added functionality in later builds.

Build 3:

In build #3, we wanted to start working on one of the more important features of our application, the tileset editor. We implemented basic tools in this time, like the brush, eraser, bucket, dropper, and clear. We also used react-color to set the colors of our primary and secondary brushes. Apart from the editing aspects of the editor, we also worked on navigating to the tileset editor by clicking on a tileset project in the library, switching between tiles in a tileset, deleting tilesets, and creating new tilesets. All of these changes would also be reflected in the back-end when appropriate.

Build 4:

In build #4, we felt that after build #3, the next step would be to make the map editor. The map editor shares a lot of features with the tileset editor, so those parts of this build were simple to implement. A new feature we worked on this build was converting arrays of color values into usable images. We then used these images to display tiles in the palette section as well as the primary and secondary tile. This is also when we decided to save tile images to the back-end as well as their tile data (which is an array of colors). This change was key in making features like importing and exporting as well as displaying tiles easier.

Build 5:

In build #5, we worked on giving users the ability to collaborate with each other on a project. This was made possible using web sockets to transmit data from collaborator to collaborator. Any changes made by one user were sent to and received by other collaborators on the project. While collaborating was the main focus on this build, we also worked on fixing bugs and possible latency issues to make the site more user-friendly.

Build 6:

In the final build, we worked on getting our site integrated with other tileset and map editors as well as game engines. This meant being able to import and export our projects in meaningful formats. For importing tilesets, we decided that importing a .PNG was the most intuitive option. We would take an image and the user given dimensions and break up the image into tiles that make up a tileset. We also export our own tilesets as images. The format we exported maps in was an altered version of the .JSON format that Tiled uses. We made it usable with Phaser as well as other engines like Wolfie2D. Apart from importing and exporting, we also did notifications and the social sidebar, which really added to the social aspect of the site.

Post Mortem

For the most part, developing the pieces application was a smooth process. Design decisions made earlier on in the planning phases helped immensely in ensuring a productive and punctual build procedure. However, now that we have completed the assignment, just like any team of developers, we too have some things we would have tackled differently.

The biggest change we would have liked to make was using an API for our editing canvas. Currently we have implemented a stack of React Grid components to mimic the behavior of a canvas. This did in fact cause some latency problems. We were able to mitigate the slow response of the edits in the canvas by implementing a clever way of viewing only part of the map. This ensures that the user, at any given time, is only viewing a max 16x16 sized map. Displaying only a limited number of pixels/tiles on the screen at any given moment in turn establishes good edit speeds. In hindsight though, it would have been easier to have used an API for this. We did try to make the switch over to the API, but since a lot of our code was dependent on the grid structure, we were not able to do so before the due date.

Another part of our application that we would have done differently is the branding of our site. During the limited time given to us, we focused on key features like editing, collaboration, and more. As a result, the CSS could be better in certain areas and we could have shown a greater presence on the site as a brand. A possible solution would have been to specify a certain color theme to use throughout the application. Though we had a few colors we wanted to use, it wasn’t enough to cover all the different parts of the site. So, although the site has the necessary features like creating tilesets, creating maps, importing, exporting, and more, we would have liked for it to have a more professional look and feel.

A third aspect of the application we would have approached differently is the deployment. Currently we are using Heroku. They do provide a decent amount of their resources in the free trial, just not enough for our needs building the pieces application. CSE416 requires our app to be deployed for all of the 6 builds up until the final presentation. Since we ran out of credits we have been paying for the deployment services, which looking back now I don't think is necessary at all. With the huge array of options available these days we could definitely have found a better deployment tool for free.

Appendix A: All Meeting Minutes

	Example	Meeting 1	Meeting 2	Meeting 3	Meeting 4	Meeting 5	Meeting 6	Meeting 7
Date ->	01-01-2022	09-06-2022	09-13-2022	09-15-2022	09-22-2022	09-23-2022	10-03-2022	10-10-2022
Time ->	12:00PM-1:15PM	10:45AM-11:00AM	11:00AM-1:00PM	10:40PM-11:10PM	6:30PM-8:15PM	4PM-12AM	4PM-12AM	4PM-12AM
Iman Ali	1.25	0	2	0.5	1.75	8	8	8
Vincent Chi	1.25	0.25	0	0.5	1.75	8	8	8
Ahnaf Hasan	1.25	0	2	0.5	1.75	4.5	8	8
Tommy Lin	0.75	0.25	2	0	1.75	8	8	8

Meeting 9	Meeting 10	Meeting 11	Meeting 12	Meeting 13	Meeting 14	Meeting 15	Meeting 16	Meeting 9
10-24-2022	10-28-2022	11-04-2022	11-11-2022	11-18-2022	11-25-2022	12-02-2022	12-09-2022	10-24-2022
4PM-12AM	4PM-12AM	4PM-12AM	4PM-12AM	4PM-12AM	4PM-12AM	4PM-12AM	4PM-12AM	4PM-12AM
7	8	4.5	5	8	8	7.5	8	7
8	5	8	5	8	8	7.5	6.5	8
8	5	8	8	6	7	8	7.5	8
6	8	3	8	7	7	8	8	6

Total Meeting Hours:	
Iman Ali	92.25
Vincent Chi	90.5
Ahnaf Hasan	90.25
Tommy Lin	91

Appendix B: Requirements Presentation Slides

Pieces

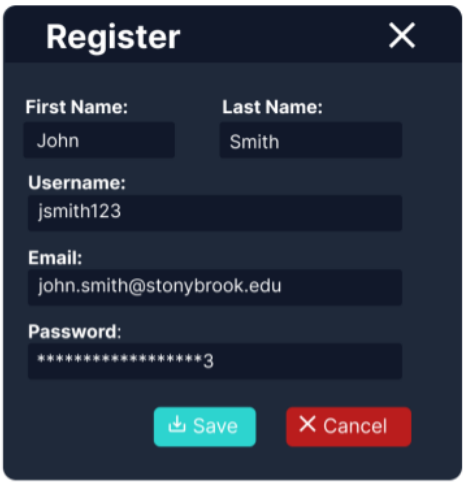
Team Cyan
Ahnaf Hasan, Iman Ali, Vincent Chi, Tommy Lin

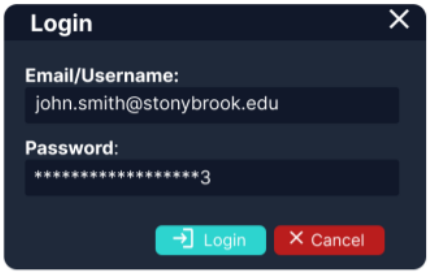
Welcome Screen

Pieces has a unique welcome screen that invites the user to join the community and the conversations regarding tile maps and tilesets.

We use the Welcome Screen as a brief introduction to the services the application provides.

In the middle of the screen is a call to action button to register for our application. On the upper-right corner are the login and register buttons that are common among web applications.





The register and login modals appear from the user requesting to login or register.

In the modals, the user can type their information and proceed to register them into the database or attempt to login with those credentials.

Explore Page

List of

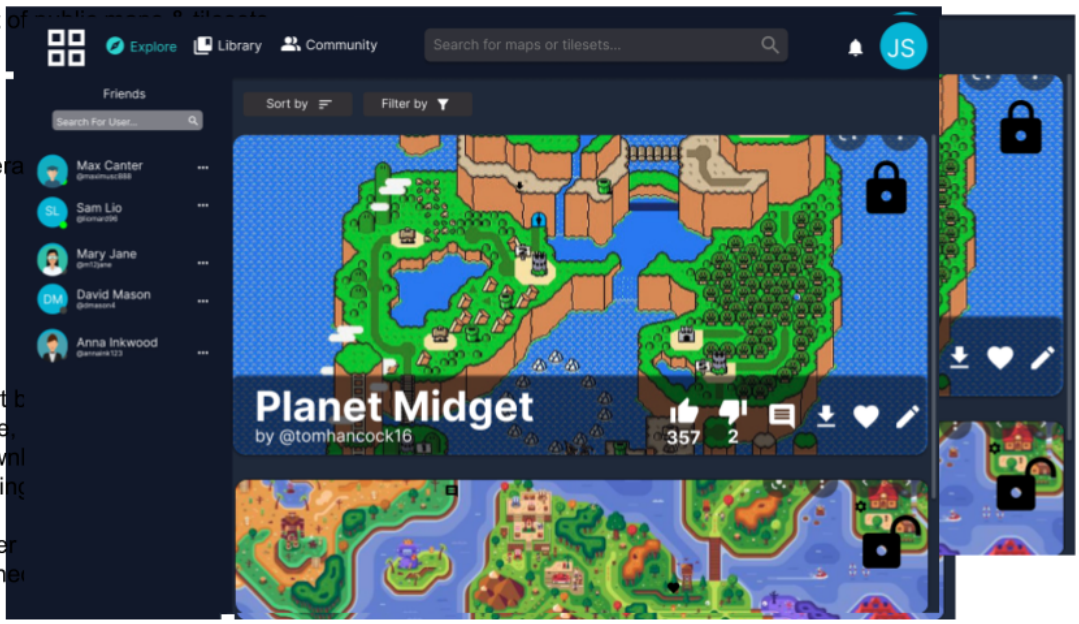
-
-

Intera

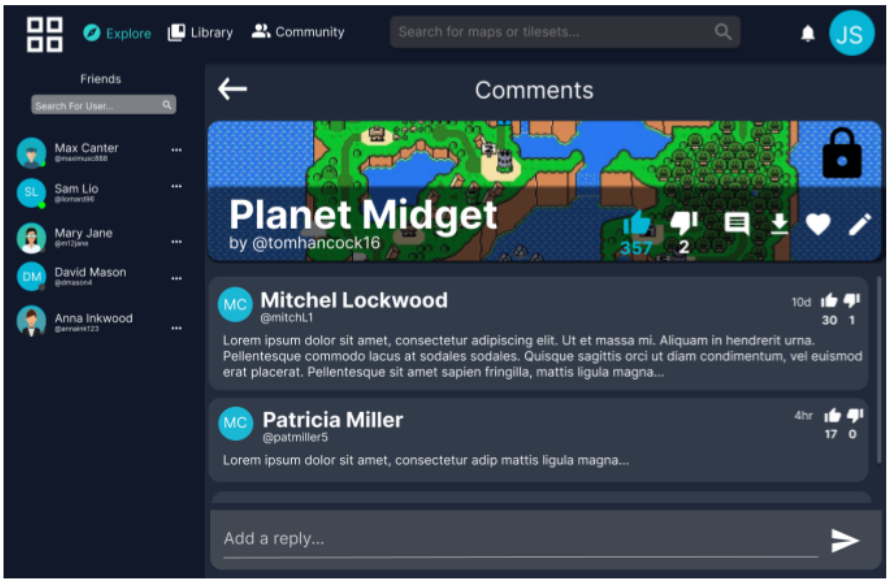
-
-
-
-
-

Sort b
date,
downl
(ratin

Filter
owne



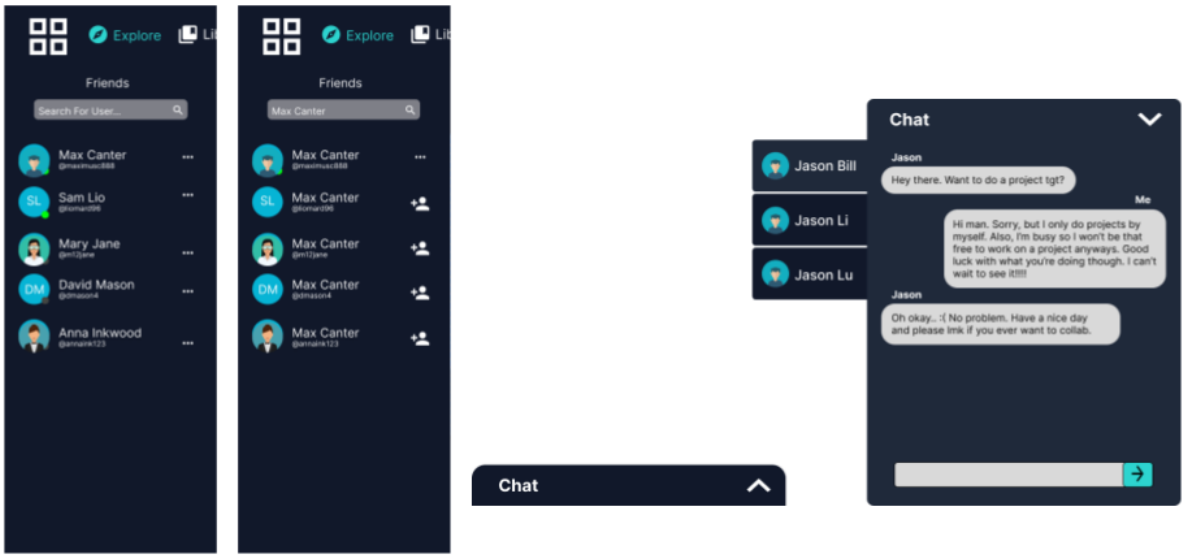
Explore Page (Comment)



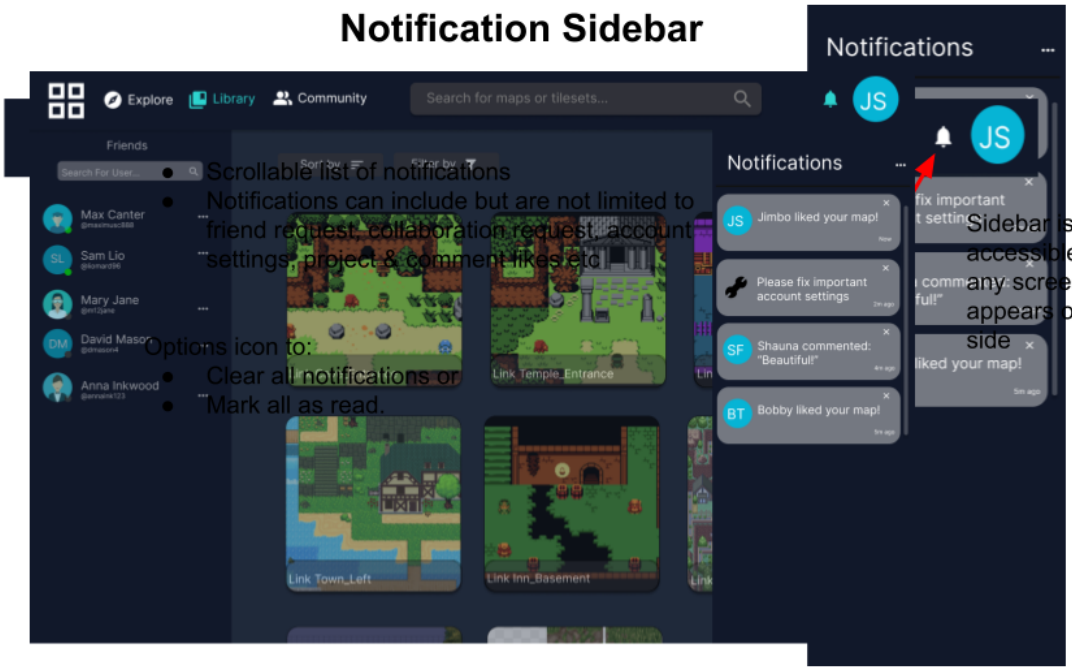
Sidebars

Can be accessed from any tab

Social Sidebar



Notification Sidebar



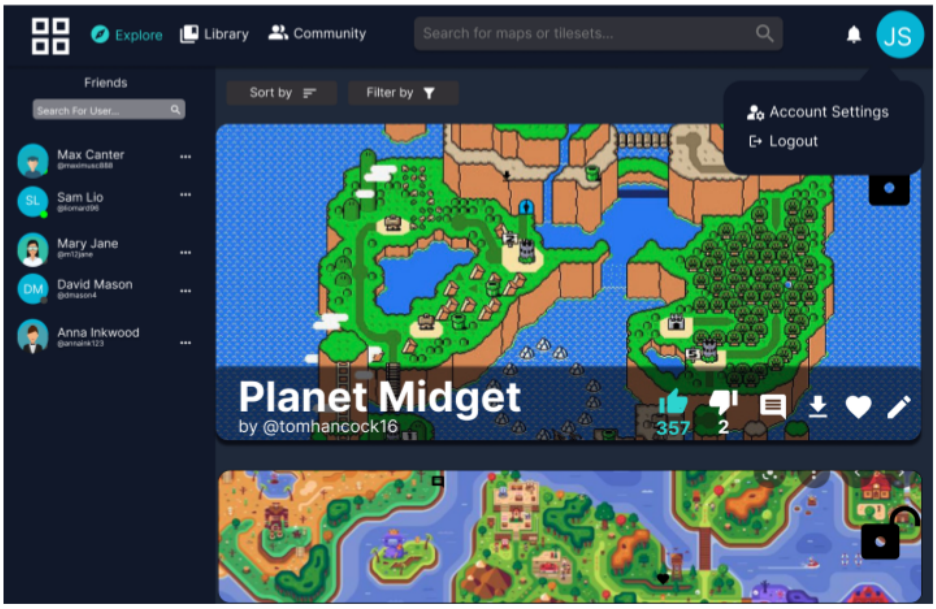
- Scrollable list of notifications
- Notifications can include but are not limited to friend request, collaboration request, account settings, project & comment likes etc

Options icon to:

- Clear all notifications or
- Mark all as read.

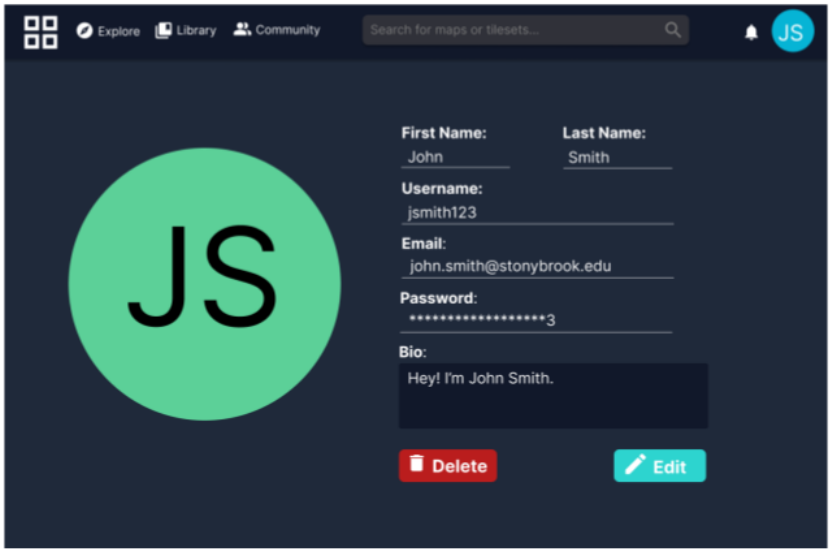
Sidebar is accessible from any screen and appears on right side

User Icon



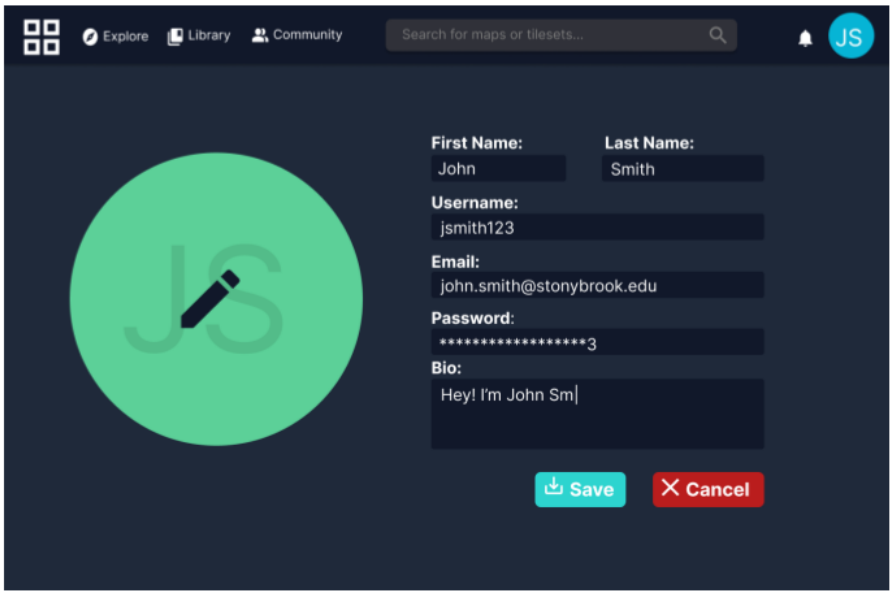
Profile Screen

- Accessed from the user icon
- Allows user to change their personal information

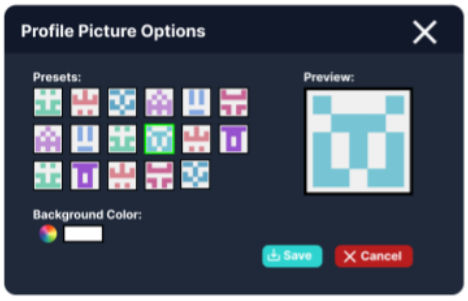


Profile Screen (Edit)

- If navigated away from the page, changes aren't saved

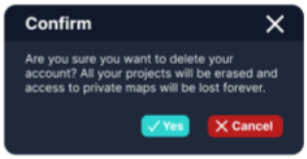


Profile Screen (Modals)



- Choose from a preset amount of profile pictures and change your background color to personalize

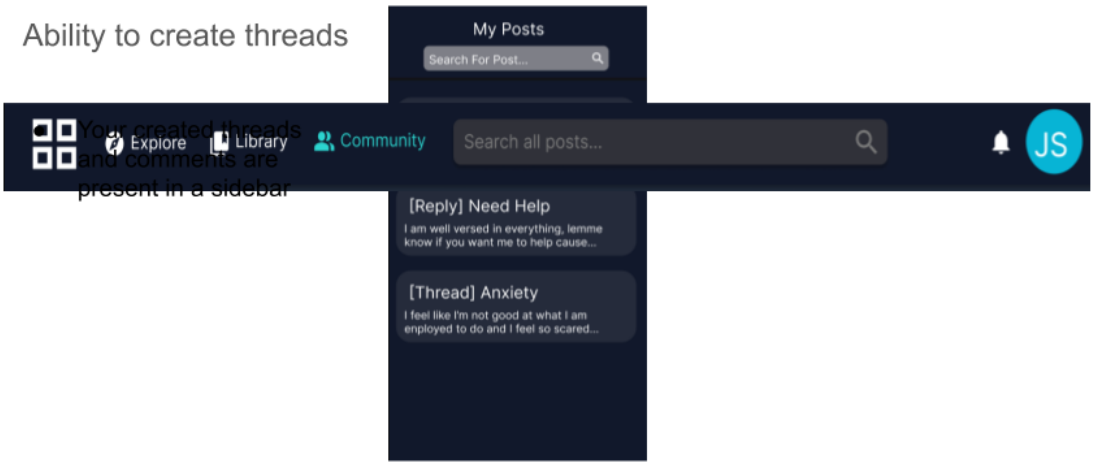
- Delete your account easily



Community Screen

The community screen is a way to interact with all of Pieces' users.

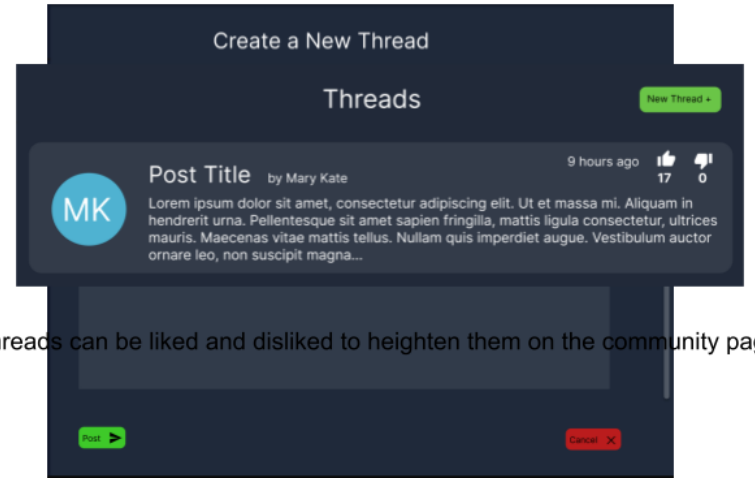
- Ability to create threads



Your created threads and comments are present in a sidebar

Threads

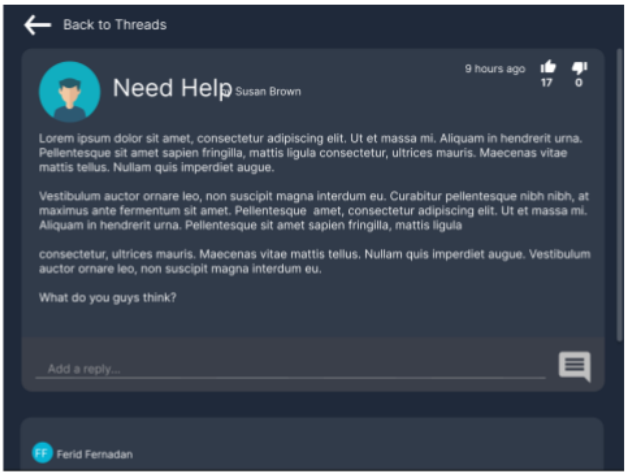
Threads can be created easily with the "new thread" button



Threads can be liked and disliked to heighten them on the community page

Comments

Comments are added to threads to interact with other users

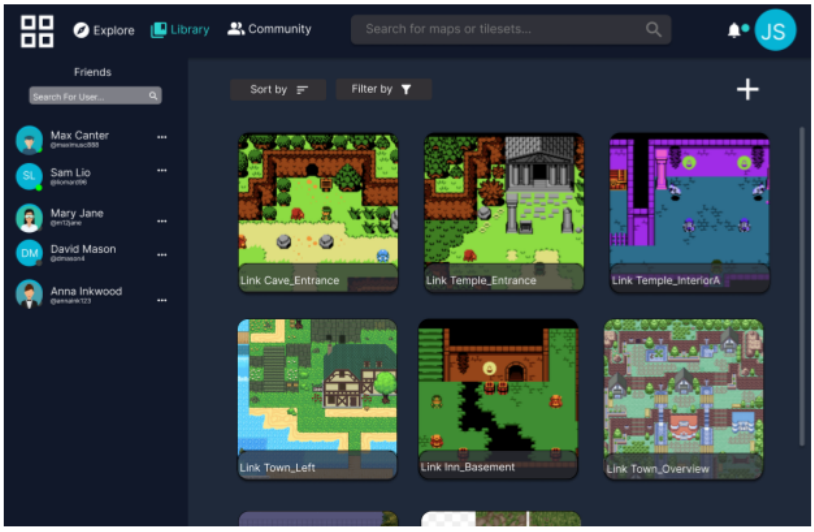


Library Screen

The Library Screen appears when the user clicks the Library button on the navigation bar.

This screen displays all the projects the user has made, contributed to, or favorited. It also allows the user to sort or filter by these categories along with others, such as filter by creator, sort by creator name, sort by project name, etc.

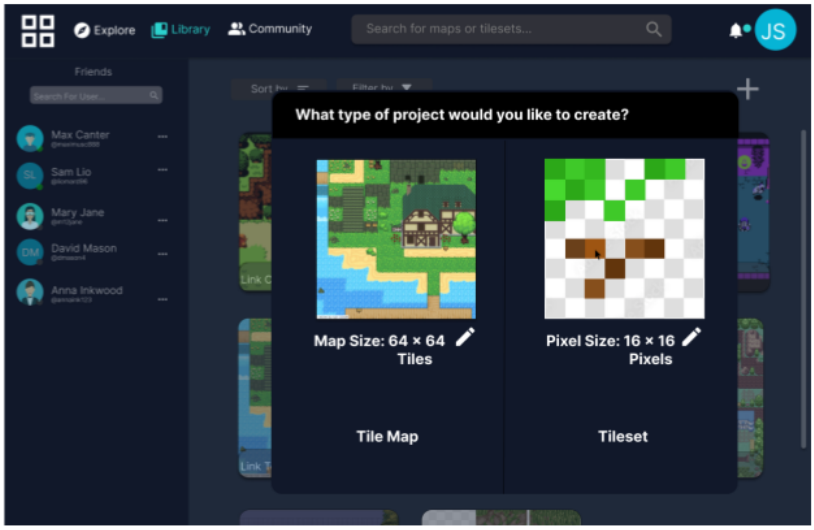
The user may click on a project to open it in their respective editor.



Library Screen (Add project modal)

The add project modal appears from the user requesting to add a project into their library.

In the modal, the user can choose between a tile map or a tileset project as well as adjust the initial size of the map/tile.



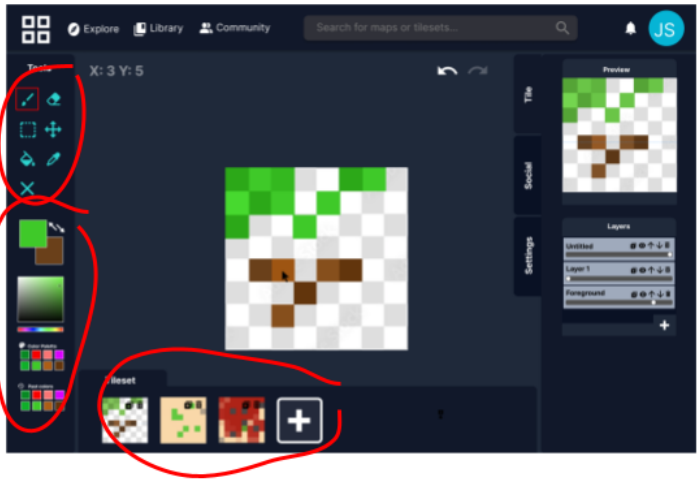
Tileset Editor Screen

Where the creativity happens…

Supplies tileset designers with the necessary tools to succeed

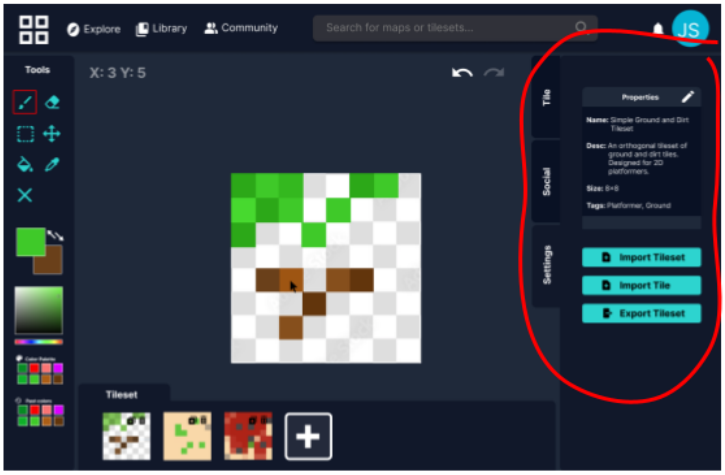
- Includes a paintbrush, eraser, color dropper, and more

Boasts an intuitive and minimalistic design so users can skip the learning curve and dive right in



Tileset Editor Screen

Continued…



- Tile tab features a preview window and access to layers
- Social tab enables team collaboration
- Settings tab displays the tileset's properties as well as import and export functionality

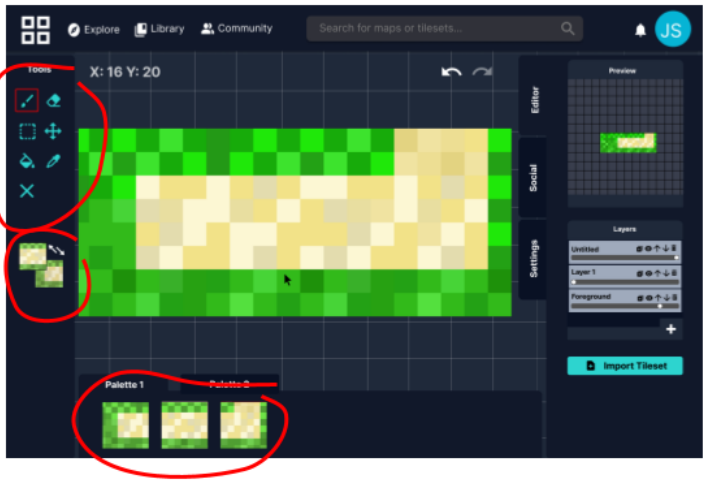
Map Editor Screen

Where the creativity also happens…

Provides map designers the essential tools in a easily digestible layout

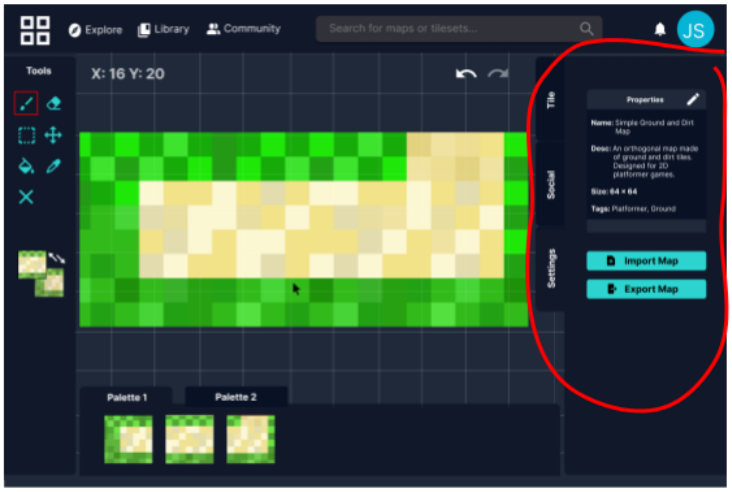
Functionalities include but not limited to:

- Placing a tile
- Erasing a tile
- Filling a map with a tile
- Importing tilesets
- Importing and exporting maps
- And much more…



Map Editor Screen

Continued…

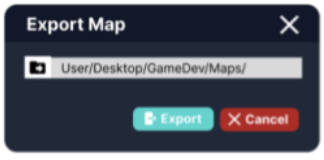


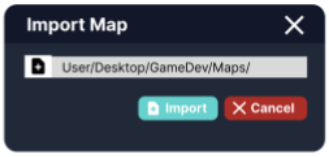
- Tile tab displays preview of the map as well as giving access to the layers
- Social tab enables team collaboration
- Settings tab allow a user to edit and view the properties and import/export maps

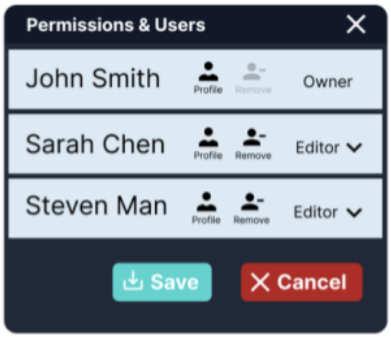
Editor Screen Modals

Here are the modals we will be creating for our editor screens:

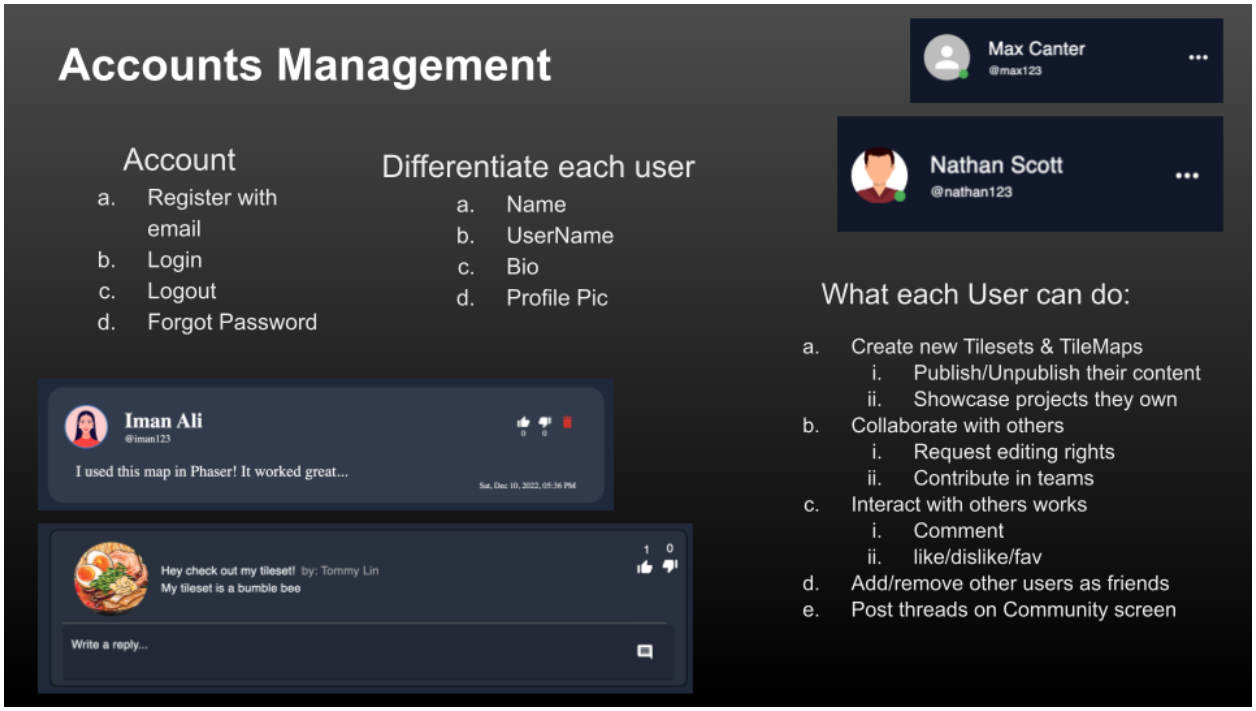
- Export tileset
- Import tileset
- Export map
- Import map
- Permissions & Users







Appendix C: Final Presentation Slides



Tilesets

Create New

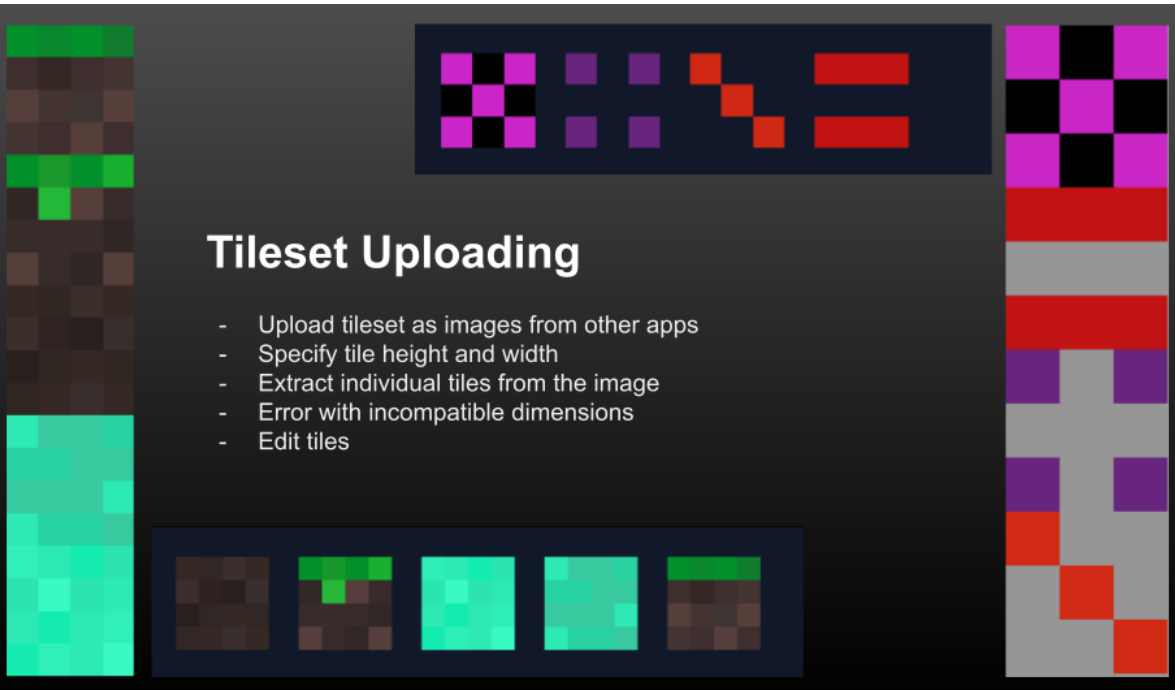
Specify:
- Tileset Name
- Tile height & width

- Get empty default tile

Upload Tileset

Specify:
- Tileset Name
- Tile height & width
- Input Image

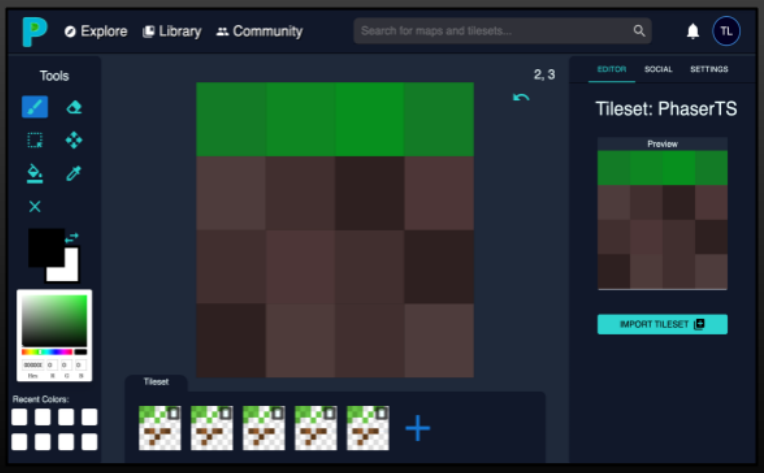
- Split image into tiles



Tileset Uploading

- Upload tileset as images from other apps
- Specify tile height and width
- Extract individual tiles from the image
- Error with incompatible dimensions
- Edit tiles

Tileset Editing



From left to right:

- The tool bar features important tools and color selection
- The canvas area displays the current tile you're working on
- The Tileset tab shows the number of tiles in this tileset
- Undo/redo and current index
- The preview shows the tile as a whole

Map Editing

From left to right:

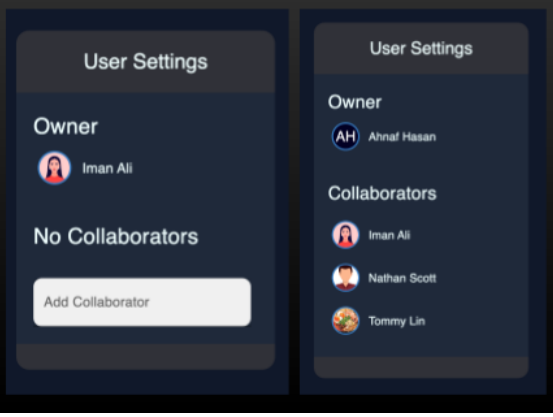
- The tool bar features important tools shows primary/secondary tile
- The canvas area displays the current tile you're working on
- The tabs are your palettes to choose tiles to place
- Undo/redo and current index

Collaborative Map Editing

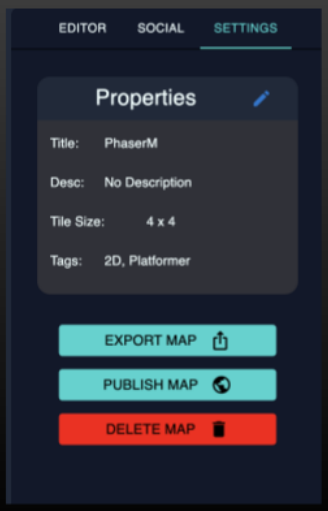
Web Sockets

- Allows real time collaboration
- Emits from one user to another
- Other user catches the emit and responds according
- User ➡ server ➡ user

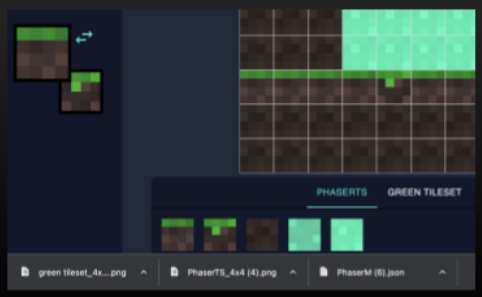
Editors Social Tab
- Owner
- Add/Remove Collaborators
- Request Edit access



Map Exporting



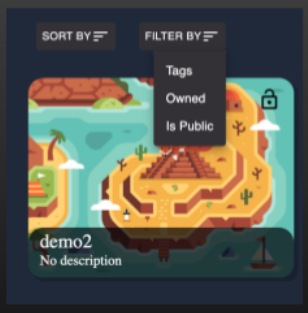
- Exported in .JSON format
- Comes with .PNG files of tilesets used by map
- Can be used in game engines like Phaser

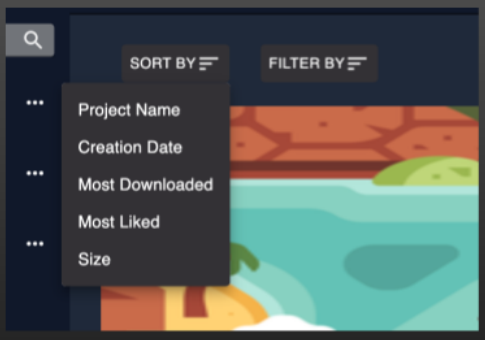


Map Classification and Search

Sort

- Project Name
- Creation Date
- Most downloaded
- Most liked
- Size

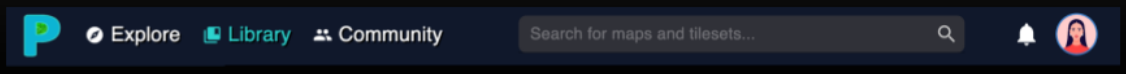




Filter

- Tags
- Owned
- isPublic

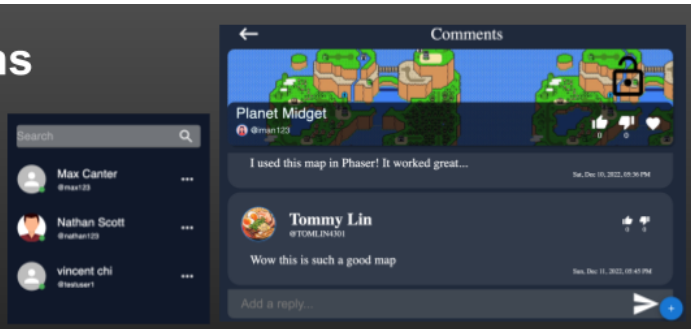
Search for public maps in explore

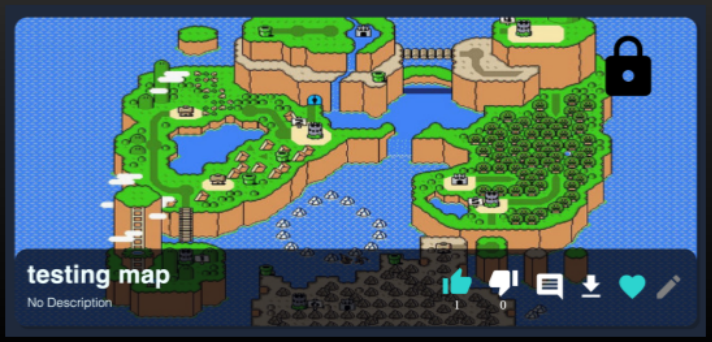


Community Interactions

Public Projects

1. Like,
2. Dislike,
3. Fav
4. Comment
5. Request collaboration

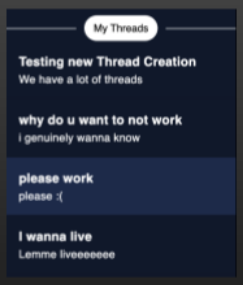




Friends

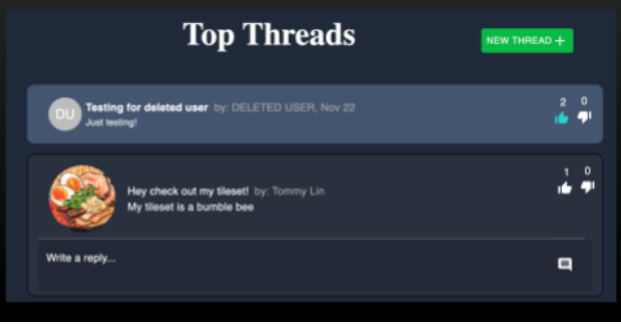
1. Add/Remove Friends
2. View Profiles

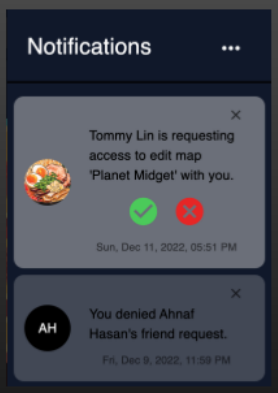
Community Interactions



Community Screen

1. Post threads
2. Like, dislike,
3. Comment on threads
4. Like, dislike replies

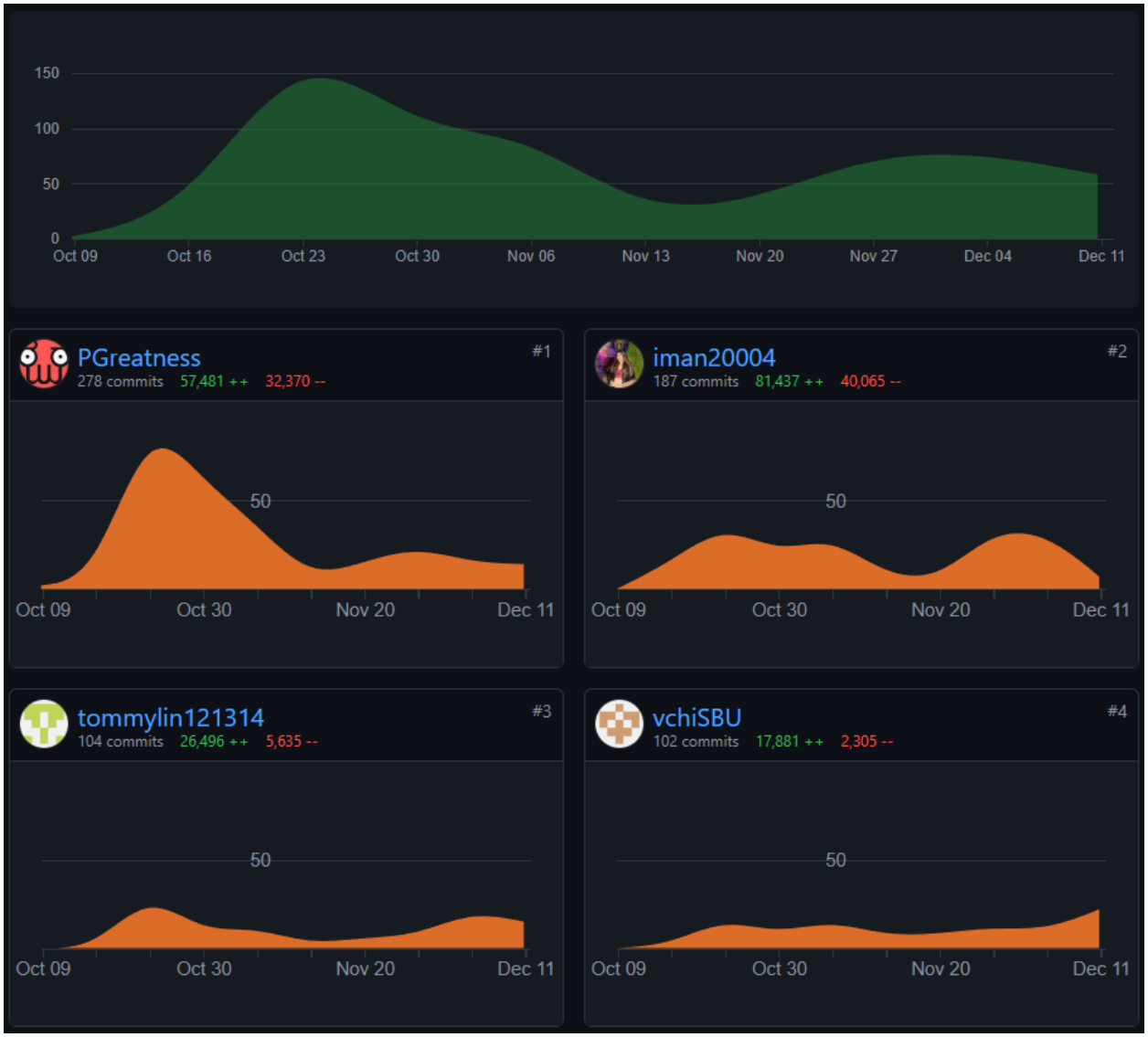




Notifications

1. Collaboration requests
2. Friend requests
3. Approve, deny

Appendix D: Detailed Descriptions of Team Member Contributions



Ahnaf Hasan - PGreatness
Iman Ali - iman20004
Tommy Lin - tommylin121314
Vincent Chi - vchiSBU

Ahnaf Hasan:

1. Heroku deployment (with Iman)
    a. Heroku deployment bug fixes
    b. Heroku payment
2. Web sockets
    a. Collaboration
    b. Notifications
    c. Social sidebar
3. Community
    a. Threads
    b. Creating new threads
    c. Deleting threads
    d. Liking threads
    e. Disliking threads
4. User Favs (for importing within app)
5. Pagination
    a. Show next page
    b. Show result size
6. Collaboration in Tileset and Map Editor
    a. Searching for users
    b. Adding user to project
7. Map Editor Viewport
    a. Moving viewport
    b. Linking viewport to backend
    c. Creating viewport
8. Import Tileset
    a. Show compatible tilesets
    b. Select compatible tilesets
9. Import Tile
    a. Show compatible tiles
    b. Select compatible tiles
10. Community store
11. Notification Sidebar
    a. Notification
    b. Notification transitions and empty notification case
12. Navbar (front end)
13. Create new Tileset/Create new Map Floating Button
14. Social Sidebar Skeleton

15. Tile Schema
16. Helped making final presentation

Iman Ali :

1. User controller functions
    a. Register
    b. Login
    c. Logout
    d. loggedIn (check if user already loggedin)
    e. Tokens, cookies and verification
    f. Forgot password (with email verification)
    g. Change password
    h. Reset password
    i. Update user name, username, email, bio
    j. User profile pic upload (cloudinary)
    k. User profile pic delete (cloudinary)
    l. Delete user
    m. getUserById, getUserByUsername
2. Explore Screen
    a. Front end react
    b. like/dislike/fav/locked/unlocked -> front end
    c. Edit request to locked projects
    d. Navigate to view screens for unlocked projects
    e. Navigation to explore comments page
    f. Sorting
3. Explore Comments Screen
    a. Front end react
    b. View single projects comments
    c. Delete comments
    d. Like/dislike (front end)
    e. Back button
4. View Tileset Screen (everything front and backend)
    a. Display project
    b. Owner and collaborators display
    c. Properties display
5. Library Screen
    a. Partial front end
    b. Helped with rendering correct projects to show (backend)
6. Create Button
    a. Map modal (front end)

    b. Tileset Modal (front end)
7. Import Tileset
    a. Import image tilesets
    b. Split image into tiles according to user specified height and width
    c. Modals
    d. Do not let import tilesets with empty tiles (from within app)
8. Export tileset as image (everything)
9. Export Map
    a. Partially split with Tommy
    b. Download json file and tileset images
    c. Modals
10. Editors (Map & Tileset)
    a. Adding/removing collaborators (front and backend)
    b. Displaying owner and collaborators
    c. Editing properties (name, description, tags)
    d. Delete project (w Modals)
    e. Publish/unpublish projects (everything)
    f. Export tileset as png (w modals)
    g. Deleting tilesets in map editor
    h. Cannot delete tilesets in use from map editor (w warning modal)
11. Profile Screen (everything)
    a. Front end
    b. Edit mode to change name, username, email, bio
    c. Change password (w modal)
    d. Change profile pic
12. Navbar
    a. Helped with design (frontend)
    b. Navigation between explore, library, community, profile and editors with correct tab highlighted
13. User Avatar
14. Drop down (logout, account settings)
15. Notifications
    a. Clear all
    b. Delete one
    c. Seen/unseen diff colors
    d. Notifications icon red (w socket)
    e. Friend requests
    f. Edit requests
    g. Approve/deny
    h. Inform others
    i. Inform user itself with new notification

16. Social Sidebar
    a. Add friend (backend)
17. Reset Password Screen (everything front and backend)
18. Global store and global auth set up
19. Heroku deployment (with Ahnaf)
20. Helped making final presentation

Tommy Lin:

1. Map Controller Methods
    a. createMap
    b. deleteMap
    c. getMapById
    d. getAllUserMaps
    e. getCollabMaps
    f. updateMap
    g. publishMap
2. Thread Controller Methods
    a. Create thread
    b. Delete thread
3. Schemas
    a. Map schema
    b. Thread schema
4. Library Screen
    a. Partial front-end
    b. Sorting functionality
    c. Filtering functionality
5. Git Actions / Jest
    a. Set up automatic testing with Github (with Ahnaf)
    b. Created map test cases
6. Tileset Editor
    a. Front-end of TilesetToolbar
    b. Front-end of TilesetCanvas
    c. Partial Front-end of TilesetRightbar (with Ahnaf and Iman)
    d. Brush tool
    e. Eraser tool
    f. Bucket tool
    g. Dropper tool

    h. Color selecting
    i. Saving recently used colors
    j. Switching between primary and secondary colors
    k. Undo
    l. Redo
    m. Current coordinate indicator
    n. Displaying a tile
    o. Displaying all tiles of tileset
    p. Switching between tiles
    q. Deleting tiles
    r. “Cannot delete last tile in tileset” functionality
    s. Adding new tile
    t. Updating tileset properties (with Iman)
    u. Preview
    v. Saving changes to back-end
7. Map Editor
    a. Front-end of MapToolbar
    b. Front-end of MapCanvas (with Ahnaf who did viewports)
    c. Partial Front-end of TilesetRightbar (with Ahnaf and Iman)
    d. Brush tool
    e. Eraser tool
    f. Bucket tool
    g. Dropper tool
    h. Tile selection from palette
    i. Switching between primary and secondary colors
    j. Undo
    k. Redo
    l. Current coordinate indicator
    m. Displaying all tiles of tileset
    n. Updating map properties (with Iman)
    o. Saving changes to back-end
8. Exporting Map
    a. Exporting map in .JSON format (with Iman)
    b. Exporting tileset .PNG files used in map (with iman)
    c. Export from MapRightBar (with Iman)
    d. Export from Explore (with Iman)
9. Import Tileset
    a. Converting image to data (with Iman & Ahnaf)
    b. Invalid dimensions error message
10. Helped making final presentation

Vincent Chi:

1. UI Design Lead
    a. Led the design process and decisions for the entire application.
    b. Designed the application Logo.
2. Controller Functions
    a. Project Comment Controller
        i. getAllProjectComments, getCommentbyId, updateComment
    b. Tileset Controller
        i. createTileset, deleteTileset, updateTileset, getAllUserTilesets, getTilesetbyId, publishTileset
    c. User Controller
        i. getUsersbyUsername, removeFriend
    d. Thread Controller
        i. Get all replies, delete reply
    e. Map Controller
        i. Get map by id
3. Schemas
    a. Tileset schema
    b. Reply schema
4. Explore Screen Comments (Needs Refresh)
    a. Front end tweaks and improvements
    b. Load all project comments (backend)
        i. Fixed bug where comments were incorrectly listed based on likes and dislikes
        ii. Unique to each project but uses get all project comments.
    c. Post a comment (backend)
    d. Like comment (backend)
    e. Dislike comment (backend)
5. Welcome Screen (everything)
    a. Front end react
    b. Call to action register button
    c. Statistics (backend)
6. Library Screen
    a. Partial front end
7. Explore Screen
    a. Front end tweaks and improvements
8. Profile Screen

    a. Initial front end (later changed by Iman)
9. Community Screen (Thread Replies) (Needs Refresh)
    a. Add reply
    b. Delete reply
    c. Select reply to reply to
    d. View all replies
        i. Unique to each thread but uses get all replies.
10. Social Sidebar
    a. Partial front end
        i. Font, profile picture, spacing.
    b. Load friends
        i. With options dropdown (front end)
    c. Search for users (by username)
    d. Add friend
    e. Remove friend
    f. View friend profile
        i. Friend profile frontend modal
        ii. Friend information backend
11. Global Store
    a. Corresponding functions to a lot of the API/controller calls mentioned above, such as loadAllPublicProjectComments
12. Community Store
    a. Corresponding functions to some of the API/controller calls mentioned above, such as load thread replies
13. Helped making final presentation