Tavern Keeper

CSCI 4805 Spring 2025

Table of Contents

Project Name

Tavern Keeper

Team Members

Silas Elder
Kaleb Scott
Evan Primasing
Kenneth Gray

Abstract

Tavern Keeper is a world-building web application designed for game masters and creators to organize and develop their fictional worlds. The platform enables users to create, edit, and manage various aspects of their worlds, including maps, characters, organizations, and locations. The application's user interfaces are built with user-friendliness and easy accessibility in mind and aim to be an introductory tool to an inexperienced user in the realm of world-building.

Tools & Technologies

- VS Code Editor
- Vercel
- React. A javascript framework for the front end.
- Node.js. Javascript to handle backend/business logic.
- PostgreSQL

Revised Requirements

Header

1. **Header**
    1.1. Layout
        1.1.1. The header will be a horizontal layout of various links used to navigate the site. These links will be in the form of icons.
            1.1.1.1. Home icon will take the user to the homepage of Tavern Keeper.
            1.1.1.2. Community icon will take the user to the community board of Tavern Keeper.

1.1.1.3. World icon will take the user to the World Page where all of their worlds are contained.
1.1.1.4. Profile icon will be a selection from preset icons chosen by the user.
1.1.1.4.1. Replaces the sign-in icon after the user signs in. If the user is not signed in, it will be replaced by a sign-in icon.
1.1.1.4.2. When clicked, the profile icon will reveal a dropdown menu where the account link and sign out link will appear.
1.1.1.5. Sign in icon will take the user to the sign in page if they are logged out. If they are logged in, it will be replaced by their profile icon.
1.2. Every page in Tavern Keeper will have this header as a standard.

Grid Display

1. Grid Display

1.1. Pages on the website will use a grid-based display to display lists of selectable data
1.2. The grid will display a grid of interactive, rectangular cells
1.2.1. Each row will contain 3 cells
1.3. Immediately above the grid, there will be a drop-down with options to change the total number of cells displayed at a time
1.3.1. The options will be 9, 18, and 27
1.4. Immediately below the grid, there will be a page bar to change the set of cells displayed
1.4.1. The bar will list the number of the current page
1.4.2. On either side of the page number, there will be single arrows to advance or regress the page by one
1.4.3. On the far sides of the bar, there will be double arrows to advance or regress to the first or last page
1.4.4. When on the first page, both of the left arrows will be grayed out and unusable
1.4.5. When on the last page, both of the right arrows will be grayed out and unusable

Homepage

1. Homepage

1.1. Layout

1.1.1. The Tavern Keeper homepage will contain a large banner introducing the product
1.1.1.1. The banner will be a large image with text overlaid.
1.1.2. The homepage will contain blurbs showcasing Tavern Keeper features and capabilities.
1.1.2.1. A blurb will be a simple section of content containing a circular image along with a paragraph of text.
1.1.2.1.1. There will be 3 blurbs showcased on the homepage
1.1.2.1.2. The first blurb will have the image to the left and the text to the right, with this order alternating in each subsequent blurb.
1.2. The homepage will always be the default page when opening Tavern Keeper.

Sign-In

1. Sign-In

1.1. The sign-in page will use Google API to process logins.
1.2. Upon signing in, the user will be returned to the Tavern Keeper homepage where the Sign In button on the header will be replaced with their profile icon.

World Page

1. World Page

1.1. The world page will have a grid display feature that dominates the page
1.1.1. Each button will represent one of the user's worlds and will display the following information
1.1.1.1. A representative image
1.1.1.1.1. The primary map for the world will be used by default
1.1.1.1.2. If no maps have been added, a default stock image will be used
1.1.1.2. The name of the world
1.1.1.3. View and Edit
1.1.1.3.1. Below each world will be a button that will take the user to the World Preview page
1.1.1.3.2. Below each world will be a button that will take the user to the World Editor
1.1.2. The top-leftmost button will be the "New World" button
1.1.2.1. The image will be a prominent cross, and the name field will read "New World"
1.1.2.2. When clicked, the Create World popup will appear

1.2. Create World Popup
    1.2.1. Initializes world
    1.2.2. Will contain a text field to enter the name of the new world.
    1.2.3. Will contain a text field to enter a short description.
    1.2.4. Will contain a checkbox to toggle whether or not the world will be public or private.
        1.2.4.1. Public vs. Private Worlds
            1.2.4.1.1. Depending on whether or not this is checked is what determines if the user's world is displayed on the community board.
    1.2.5. Will contain a dropdown containing a list of preset genres that may match the user's world that the user will be able to select.
    1.2.6. Upon submitting, this will send the user directly to their newly created world's dashboard.

World Editor

1.1. Main Content
    1.1.1. There will be a label at the top of the page containing the title of the world.
    1.1.2. Next to the name of the world will be an edit icon where the user can change the displayed name of their world.
    1.1.3. The main content will also contain a short user-created description of their world.
    1.1.4. The description of the world will have an edit icon where the user description can be changed.
    1.1.5. Will contain a dropdown to select the genre of world as the user sees fit.
    1.1.6. Will contain checkbox to toggle whether the world is set to public or private.
1.2. Sidebar
    1.2.1. The sidebar will contain various icons linking to different pages for entering different information about a world.
        1.2.1.1. Home icon will link back to the main World Editor page
        1.2.1.2. Map icon will link to the Map Page of the World Editor
        1.2.1.3. Character icon will link to the Character Page of the World Editor
        1.2.1.4. Organization icon will link to the Organization Page of the World Editor
        1.2.1.5. Location icon will link to the Location Page of the World Editor

1.2.1.6. Timeline icon will link to the Timeline Page of the World Editor
1.2.1.7. Misc icon will link to the Misc Page of the World Editor
1.2.1.8. Home icon will link back to the main page of the World Editor
1.2.1.9. Submit button will update the user information and send it to the database

World Editor Sub-pages

1. **World Editor Sub-pages**
    1.1. The World Editor will link to various pages where the user will be prompted to enter and store information about their world.
    1.2. Layout
        1.2.1. Every sub-page will contain a sidebar with links to all the other sub-pages as well as a link back to the main page of the World Editor
        1.2.2. Every sub-page will contain a button at the top of the page to add new entries under that category
        1.2.3. Each sub-page will contain a list of previously created entries under that category
            1.2.3.1. Each previously created entry will be an icon with a name under it
    1.3. Functionality
        1.3.1. Map page
            1.3.1.1. Popup
                1.3.1.1.1. Upon clicking the add button on this page, a popup prompting the user to add a new map will open
                1.3.1.1.2. This popup will contain a text field to enter the name of the map
                1.3.1.1.3. This popup will contain an upload button allowing the user to browse their desktop to upload an image.
                    1.3.1.1.3.1. PNG and JPEG files are supported.
            1.3.1.2. View popup
                1.3.1.2.1. Upon clicking on a previously created map, a detailed view of that map will open. All previously placed pins will be shown.
                1.3.1.2.2. This will contain a zoom-in and zoom-out button.
                    1.3.1.2.2.1. Minimum zoom value: 25%
                    1.3.1.2.2.2. Maximum zoom value: 200%
                    1.3.1.2.2.3. Each click will zoom in or zoom out at a rate of 25%.

1.3.1.2.3. This will contain a pin button where the user can create, place, and remove pins in their world.

1.3.2. Character page

1.3.2.1. Popup

1.3.2.1.1. Upon clicking the add button on this page, a popup prompting the user to add a new character will open.

1.3.2.1.2. Will contain a text field to enter the name of the character.

1.3.2.1.3. Will contain a text field to enter a description of the character.

1.3.2.1.4. Will contain a field where the user will be able to choose preset character portraits.

1.3.2.1.4.1. The user will be able to pick from 20 preset PNG images that span character archetypes from various different genres.

1.3.2.2. View popup

1.3.2.2.1. Upon clicking a previously created character, a popup menu with a detailed view of that character will open.

1.3.2.2.2. The detailed view will contain the character name, description as well as the character portrait.

1.3.2.2.3. The popup will contain an editable field for each of the character's attributes.

1.3.2.2.3.1. Character portraits will be displayed in a circular image at the top of the popup.

1.3.2.2.3.1.1. A pencil icon will be located at the bottom right of the image leading the user to a new popup to select a new character portrait from Tavern Keeper presets.

1.3.2.2.3.2. For character names, a textbox with a label will be located below the character portrait allowing the user to edit the name.

1.3.2.2.3.3. For character description, a textarea with a label will be located below the character name textbox allowing the user to edit the description.

1.3.3. Organization page

1.3.3.1. Popup

1.3.3.1.1. Upon clicking the add button on this page, a popup prompting the user to add a new character will open.

1.3.3.1.2. Will contain a text field to enter the name of the organization.
1.3.3.1.3. Will contain a text field to enter a description of the organization.
1.3.3.2. View popup
1.3.3.2.1. Upon clicking a previously created organization, a detailed view of that organization will open.
1.3.3.2.2. Will contain the name and description of the organization.
1.3.3.2.3. The popup will contain an editable field for each of the organization's attributes.
1.3.3.2.3.1. For organization names, a textbox with a label will be located at the top of the popup.
1.3.3.2.3.2. For the organization description, a textarea with a label will be located below the organization name textbox allowing the user to edit the description.
1.3.4. Location page
1.3.4.1. Popup
1.3.4.1.1. Upon clicking the add button on this page, a popup prompting the user to add a new location will open.
1.3.4.1.2. Will contain a text field to enter the name of the location.
1.3.4.1.3. Will contain a text field to enter a description of the location.
1.3.4.2. View popup
1.3.4.2.1. Upon clicking a previously created location, a detailed view of that location will open.
1.3.4.2.2. Will contain the name and description of the location.
1.3.4.2.3. The popup will contain an editable field for each of the location's attributes.
1.3.4.2.3.1. For location names, a textbox with a label will be located at the top of the popup.
1.3.4.2.3.2. For location description, a textarea with a label will be located below the location name textbox allowing the user to edit the description.
1.3.5. Timeline page
1.3.5.1. Popup

1.3.5.1.1. Upon clicking the add event button on this page, a popup prompting the user to add a new event will open.
1.3.5.1.2. Will contain a text field to enter the name of the event.
1.3.5.1.3. Will contain a text field to enter a description of the event.
1.3.5.1.4. Will contain two combo boxes to enter the date that the event began and the date the event ended.

1.3.5.2. Sorting
1.3.5.2.1. Once events are added, Tavern Keeper will automatically sort each event chronologically by event start date.
1.3.5.2.2. Each event is linked vertically in the Timeline page UI.

1.3.5.3. View popup
1.3.5.3.1. Upon clicking a previously created event, a detailed view of that event will open.
1.3.5.3.2. Detailed view will contain the name, description, and dates for the specific event.
1.3.5.3.3. The popup will contain an editable field for each of the event's attributes.
1.3.5.3.3.1. For event names, a textbox with a label will be located at the top of the popup.
1.3.5.3.3.2. For the event description, a textarea with a label will be located below the event name textbox allowing the user to edit the description.
1.3.5.3.3.3. Below the event description will be a combo box that will allow the user to enter a start date
1.3.5.3.3.4. To the left of the start date combo box will be another combo box to enter the end date of the event
1.3.5.3.4. Upon closing this tab, if a date is changed, the events will be re-sorted.

1.3.6. Miscellaneous Page
1.3.6.1. Popup
1.3.6.1.1. Upon clicking the add button on this page, a popup prompting the user to add a miscellaneous entry will open.
1.3.6.1.2. Will contain a text field to enter the name of the entry.

1.3.6.1.3. Will contain a text field to enter a description of the entry.

1.3.6.2. View popup

1.3.6.2.1. Upon clicking a previously created location, a detailed view of that entry will open.

1.3.6.2.2. Will contain the name and description of the entry.

1.3.6.2.3. The popup will contain an editable field for each of the entry's attributes.

1.3.6.2.3.1. For entry names, a textbox with a label will be located at the top of the popup.

1.3.6.2.3.2. For the entry description, a textarea with a label will be located below the entry name textbox allowing the user to edit the description.

World Preview

1. **World Preview**
   1.1. Layout
      1.1.1. Will contain a banner
      1.1.2. Will contain a label at the top of the page containing the name of the author
      1.1.3. Will contain a text area containing a short description of the world
      1.1.4. User Data
         1.1.4.1. The page will be separated by headers of each type of data that the user has entered (Maps, characters, locations, organizations, timeline events, and miscellaneous)
         1.1.4.2. Data will be displayed under each header in a grid image format similar to previous sections
   1.2. Functionality
      1.2.1. Upon clicking a widget (map, character, etc), it will operate similarly to the detailed view functionality of the world editor sub-pages (see 7.3).

Profile Page

1. **Profile Page**
   1.1. Upon loading the profile page, users will be prompted to select a profile picture from a set group of pictures.
   1.2. Users will be able to add a short description of themselves.
   1.3. Users will be able to add aliases for themselves.

Community Page

1. **Community Page**
    1.1. Upon loading the community page, users will be able to view other world's created by users.
    1.2. The world will display title, picture, author, genre, and date created.
    1.3. Filter
        1.3.1. Users will be able to filter maps by genre.
    1.4. Search functionality
        1.4.1. Users will be able to search by author name to find a specific world.
        1.4.2. Users will be able to search by world name to find a specific world.
    1.5. View
        1.5.1. Upon clicking on a world, a user will be sent to view a world in detail.
        1.5.2. Here, the user will be enabled to see the world's characters, locations, etc. in detail upon hovering over each component.
        1.5.3. In this view, each world's data will be organized as laid out in World Preview (see 8.1).
        1.5.4. Interactivity
            1.5.4.1. In the view, the user will be able to like the selected world.

Design Description

Overview

Tavern Keeper is a web-based application that combines a variety of components to form the architecture. Tavern Keeper acts as a web-based organizer and encyclopedia for user data pertaining to fictional worlds stored in a simple and aesthetically pleasing user interface. The overall structure of Tavern Keeper is organized into a back-end and front-end format. Generally, the back-end server-side code will be handled through the Vercel infrastructure. The database will also function through the Vercel infrastructure. Meanwhile, the front-end user interface will be written in the Javascript framework React and will communicate with the Vercel server. The React-based front-end will also integrate and utilize the Google Sign-In Authentication API. Lastly, user's emails will also be stored in the Vercel database.

Components & Classes

The login component of Tavern Keeper will be managed by the Google Sign-In API. It will consist of a simple form that will prompt the user for their Gmail and Google account password. Tavern Keeper will not store the password of the user's Google account; however, we will store the Gmail in the database. Authentication will not be handled on the Tavern Keeper infrastructure; we plan to outsource this aspect of the login entirely to the Google API. There are no current plans to implement a sign-up page or functionality, and the assumption is that a Google account already exists for new and returning users. In the case of user names, the default will be a snippet of the user’s Gmail. However, it can be updated to a more fitting alias on the user’s profile page.

A large component of Tavern Keeper will be the user component. The user class will handle managing and updating worlds connected to that user account as well as the updates and management of personalized user data–userName, description, profilePicture. The methods in the user class are mostly simple getter and setter methods used for managing the user fields: userName, description, profilePicture, and email. These methods will be a significant point of focus where information is sent and received from the Vercel database. Furthermore, as mentioned previously, the username for a new user will be a snippet of their email that will be sent to the Vercel database upon login. The other field–worlds–is a list that contains each world object associated with that user. Using two simple methods, addWorld, and removeWorld, the ownership of worlds will be managed by the user class. Ultimately, the user component

is a cornerstone of the Tavern Keeper architecture as it will interact with nearly every other component in some shape or form.

The next component of the Tavern Keeper class architecture is the World component. This component will handle the management and updates for specific data pertaining to that world. The World class will be a large component containing descriptive data about the world, some metadata about the world, and a list of entries associated with that world. One of the fields associated with the World class–genre–will be an enum datatype. At the time of writing, the current fields in the genre enum will be Science Fiction, Fantasy, Romance, Mystery, Horror, Thriller, and Nonfiction. The fields will remain broad to cover a large scope of options. The methods associated with this class take a simple approach tailored to editing, updating, and managing the fields within this class. There are getter methods for the metadata component, getters and setters for the descriptive data, and there are methods for adding entries, removing entries, and receiving the full list of entries. These methods will also send messages back and forth to the Vercel database to quickly update vital world and user information.

Another component of the Tavern Keeper class infrastructure is the Entry component. The entry component is essentially user input in class form. Entries will be used to format, store, and transport user data associated with one world. The structure for a basic entry object is that of a name and a description, along with setters and getters for each. Each method will include calls to the database to react to user input whenever they edit an entry or whenever the user interface needs entry data.

The entry class also has three subclasses that extend from it. Each subclass is essentially an entry with a few extra fields that are specific to that type of data alongside methods to manage these fields and access other components.

The first of the subclasses is the map class. The map class inherits and implements all the fields and methods of an entry along with the following fields: image and pins. The image field is the stored image of the map in the BLOB datatype format. This is whatever is uploaded by the user. The pins field is essentially a list of pin objects placed by the user on that specific map. The pin object is its own component, but it is quite simple. The pin object is a location on the map as specified by its X and Y coordinates used to triangulate its point. It also contains a name and description for the user to outline the pin's purpose. Pins are managed by their respective map object, which includes methods–getPins, addPins, and removePins–to do so. However, the descriptive data about a pin itself is managed within its own class, not by the map class. It also contains the methods–getters and setters–to do so. Which also includes the X and Y coordinates which will be updated each time the user moves a pin. Each of these methods has its own respective field in the Vercel database and will be updated when called upon.

The second of the subclasses is the character class. The character subclass will also not override any fields or methods it inherits from its Entry parent class, except for

the fact that it contains the field: portrait. The character portrait field is responsible for storing the respective character portrait belonging to that character. These portraits will not be user-uploaded but rather will be presets stored in the Vercel database that the user can pick from on Tavern Keeper. The character class will also be responsible for managing the portrait it is associated with using its getter and setter methods. These methods will respond to user input on the Tavern Keeper front end and update the portrait value on the backend by calling the database. By default, the value of the portrait shall be null.

Lastly, the third of the entry subclasses is the event class. Without change, the event subclass will implement each method and field from the entry parent class. What differentiates the event class from the entry class are the fields of startDate and endDate. These fields specify dates for fictional events in the user timeline, which, in turn, is used to sort the events chronologically. The event class has getter and setter methods that will be used to respond to user input on Tavern Keeper and then update the database efficiently.

User Interface Storyboard & Pages

Upon entering the Tavern Keeper website, the user will first be fronted by the homepage. The homepage is the introduction of Tavern Keeper mostly tailored to new users wondering what Tavern Keeper is about. The user will look upon a large background picture and the header. Below will be alternating blurbs containing an image and a short paragraph introducing Tavern Keeper.

A key component of the Tavern Keeper user interface is the dynamic header. The header will be the navigation hub used by the users to get around to the main pages of the website and will be visible on all pages. There are four main buttons located on the Tavern Keeper header; there is the home button, the world button, the community button, and–depending on whether or not the user is signed in or not–the account or the sign in button.

Upon clicking the community icon on the dynamic header, the user will be sent to the Tavern Keeper community page. The community page is a simple board where users looking to display their world or receive feedback can post links to their world. The page layout is quite simple. First, at the top of the page, is a simple search bar that will look for worlds based upon two criteria: world name and author name. Next, to the right of the search bar, a filter option will help the user narrow down their search query. In this filter, users will be able to filter their searches by genre, likes, or the date created. Now, below both the search bar and the filter icon is the main content of the community page. The main content is where all of the user uploaded worlds will be contained. Each world icon will be a small preview of the world and will include a few attributes pertaining to that world. Each icon will contain the world name, world description, world author–userName–, genre, date created, and likes. Each icon will be contained within

an adjustable grid display layout, with the default being three world icons in a row. Lastly, the page scroller will be at the bottom of the page, used to navigate between various numbered pages on the community board.

If the user is not signed into Tavern Keeper using their google account, there will be a sign-in icon on the dynamic header. Upon clicking this button, the user will be directed to the Tavern Keeper sign-in page. The sign-in page is a simple form that will be powered by the Google sign-in API asking them for their Gmail and password.

After signing in, the sign-in button will disappear and be replaced by the account icon. Upon clicking the account icon, the user will be sent to the account page. The account page is a simple page used to edit simple points of user descriptive data. Here, there will be a textbox at the top of the main content to edit the username. Below that textbox there will be another textbox to edit or add a user description. Lastly, to the left of the username textbox will be a small round profile picture portrait. This will be an interactable button that can be used to add or edit a user's current profile picture. The profile pictures will be a few generic, preset PNGs.

The last button on the dynamic header is the world icon. Upon clicking the world icon, the user will be sent to the world page. At the top will be a Create World button. When clicked, this button will open the WorldPagePopup. The WorldPagePopup form contains all the fields needed to initialize a new world in Tavern Keeper. First, it will have a label and a textbox prompting the user to enter their world name. Next, it will have another label and textbox for the user to enter the world description. Next, it will have a label and a button that extends into a dropdown menu allowing the user to choose their preset genre for their new world. Lastly, it will have a label and a checkbox that will give the user the option to select whether their world is private or public–with public worlds being displayed on the community page automatically. Below the create world button is the world page's main content. The world page's main content layout is somewhat similar to the grid layout in the community page. Like the community page, the main content of the world page is populated by world icons used to represent a world that belongs to a user. The default layout is three worlds to a row, but this aspect is adjustable. If the number of worlds becomes too large to fit on the page, a page scroller, similar to that of the community page, will appear. When hovering over a world icon, the user will be presented with two options, view or edit.

After creating and initializing a world, a world icon will populate their world page, as mentioned previously. Upon clicking the edit button on the world icon, the user will be sent to the world overview page. This is the hub where users can add, edit, and remove data about their world. There will be a sidebar on the main world overview page and every subpage. This sidebar will be the main form of navigation throughout the data entry process. It will be populated with buttons linking to every world overview page. It will also contain a submit button to be used when users are finished entering data, and calls can be sent to the database.

The initial and main world overview page is used to toggle simple and general aspects of a world. Furthermore, the layout is simple to reflect this. At the very top of the page, above the sidebar, there will be a simple label welcoming the user. A series of fields will be below that label and parallel to the sidebar. The first field is a label and an edit icon allowing the user to edit the name of their world. Upon clicking the edit icon, the label turns into a textbox giving way for changes. The next field is a label and an edit icon prompting the user to edit the description of their world. Functionality works the same as the previous. The next field is a label and another edit icon allowing for genre editing. Upon clicking the edit icon, the label becomes a dropdown menu listing all the preset Tavern Keeper genres. The last field is a label and a checkbox that prompts users to change their world's publicity status.

The last group of pages are the world overview subpages. In terms of format and functionality, these pages are nearly identical. At the top of the page, there is an add button. This button will open a popup that prompts users to fill in fields to create a new entry under that subpage's respective category. Below the add button is the grid layout of previous entries under that category. Its format and function are identical to the community page and the world page. Upon clicking the previous entry icons, a popup will open, allowing the user to edit each attribute for that datatype, mostly using text boxes.

Entity Relationship Diagram

As mentioned previously, we will use the Vercel infrastructure for data storage. Vercel offers a free PostgreSQL option for its clients, and that is the option that is being built upon. Storage is a vital component of Tavern Keeper, and its infrastructure relies on it. Tavern Keeper's database design will consist of nine tables: User, Worlds, Profile Pictures, Characters, Locations, Organizations, Events, Maps, and Pins. Also, each table contains an ID as its primary key.

The first table is the User table–arguably the most vital table. The User table data fields–all of which have been mentioned–are username as a string, description as a text, profile picture, which is an integer and a foreign key, and lastly, their email, which is a string. Most of these fields are initialized during the user's first sign-in with Tavern Keeper. Furthermore, all of these fields but the email are editable on the Tavern Keeper website. The next table is the Profile pictures table–a simple table that stores the preset PNGs that will act as the user profile pictures. This table is linked with the user table is toggled, and accessed by ID.

The next table is the Worlds table–the largest table. The Worlds table contains a userID as its foreign key. The rest of its fields are title as a string, genre as a string, Public as a boolean–used to indicate the world's publicity status, description as a text, DateCreated as a date, and Likes as an int. All of these fields effectively describe a world and coincide with its respective class.

The next group of tables are all world data because they are all pieces of user input that are associated with a specific world. Consequently, each of these has a WorldID as a foreign key, linking them to a certain world. The first of which is the Characters table. The characters table is a standard entry–name and description–with a character portrait field that is of the BLOB datatype. This will be a simple preset image stored on Tavern Keeper.

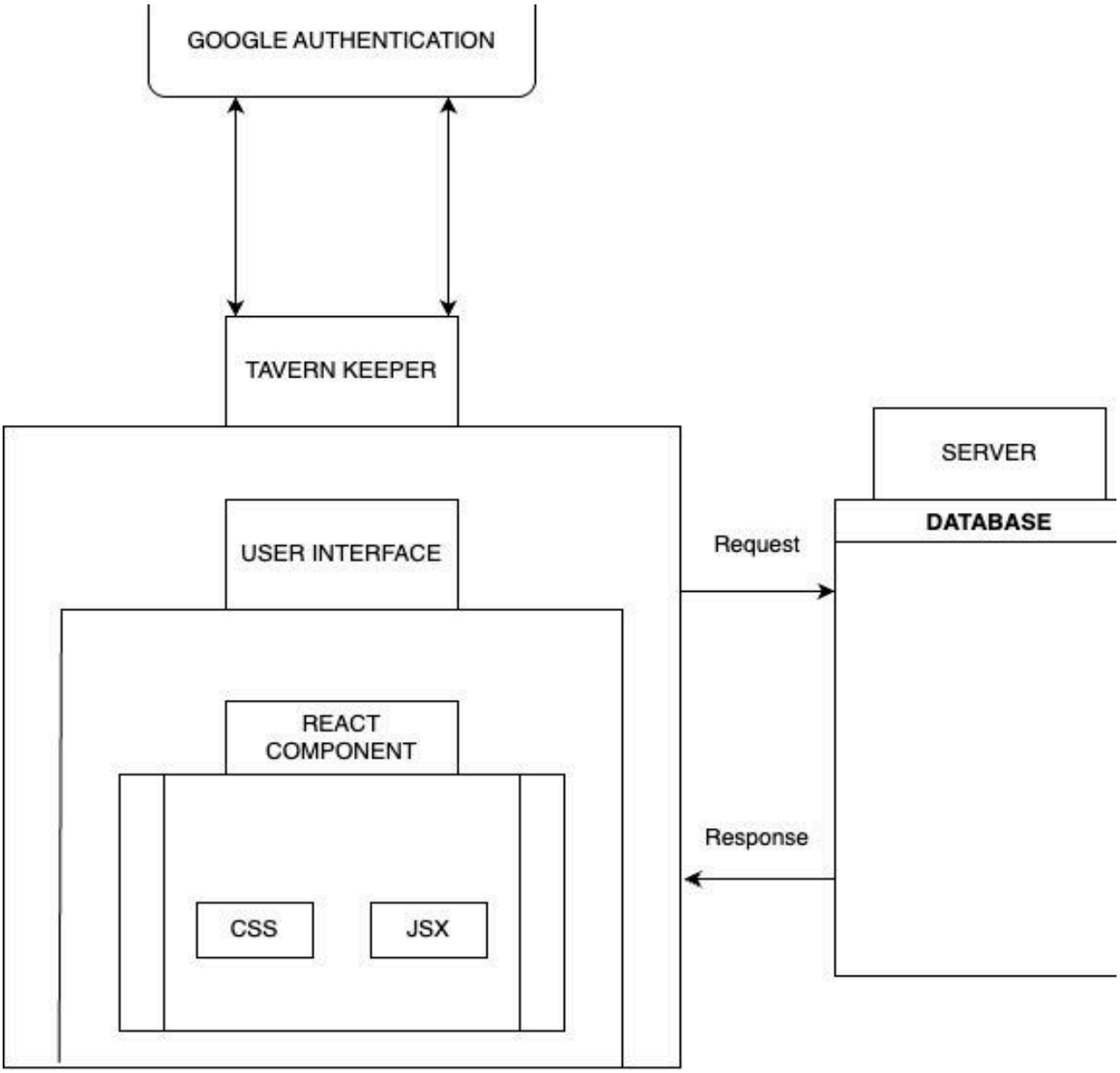
The next world data tables are the Locations and the Organizations table. These are grouped together because they have the same fields–name and description. They have the structure of a basic entry, but the only difference is that they describe different things. Next is the Events table. This table has the structure of a basic entry, but it has a start date and an end date added on to specify chronology. After that is the Maps table. The Maps table contains a name and a description, but it also has a picture stored as a BLOB datatype. This will be user-uploaded. Outside of the realm of world data is the Pins table. This table relates to the Maps table as it has a mapID as its foreign key. It contains a name and description and X and Y coordinates to store where it points on a specific map.

Servers & Messaging

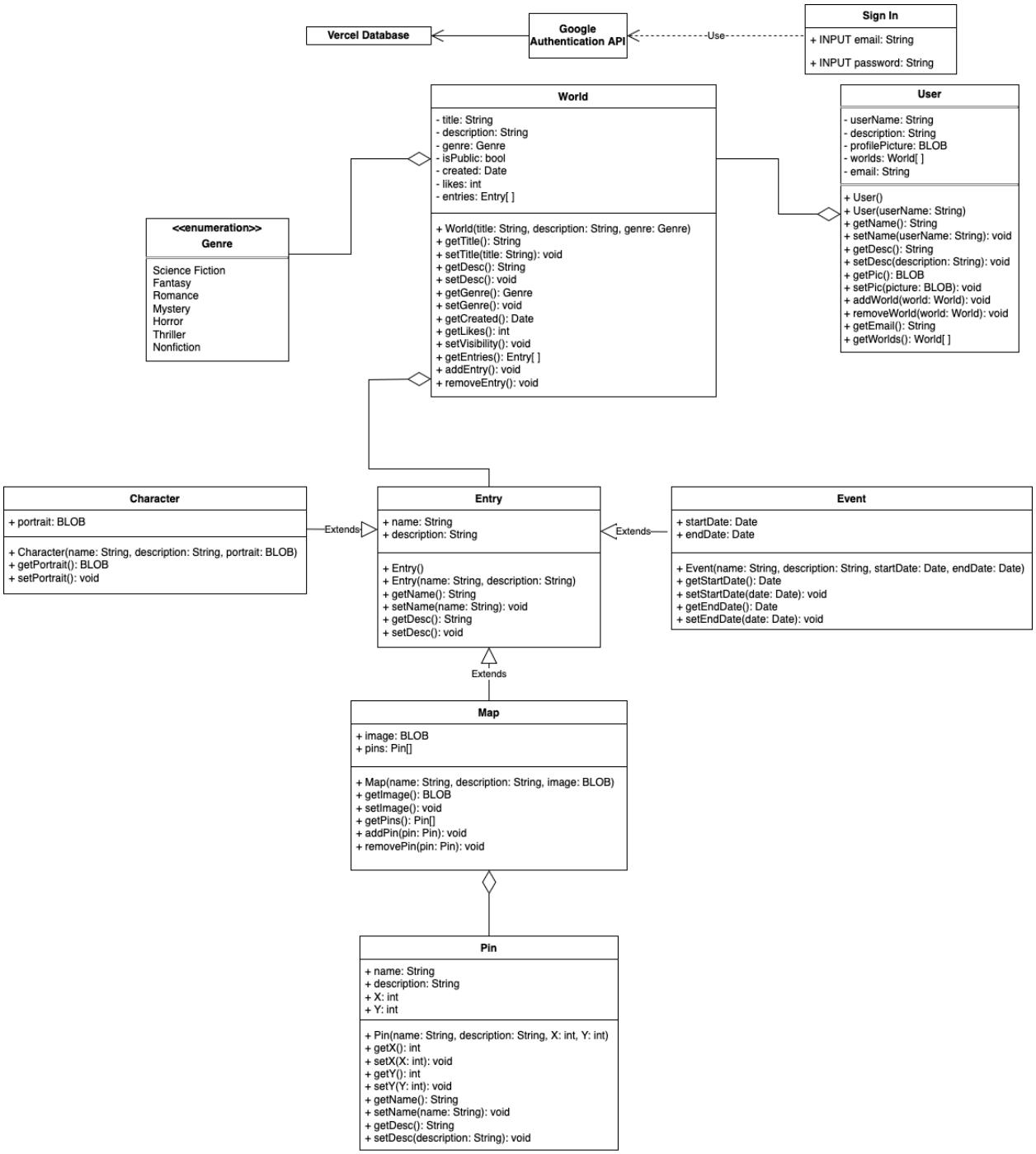
As mentioned previously, the server infrastructure that will be handling messaging between the front end and back end of our architecture is Vercel. The data format that will be used to send data back and forth between the front end and the database is JSON. The inherent structure of JSON fits the Tavern Keeper architecture seamlessly. Data will be sent back and forth using HTTP GET and HTTP POST methods. Various examples of the Tavern Keeper messaging structure are provided in the fourth appendix.

Appendices

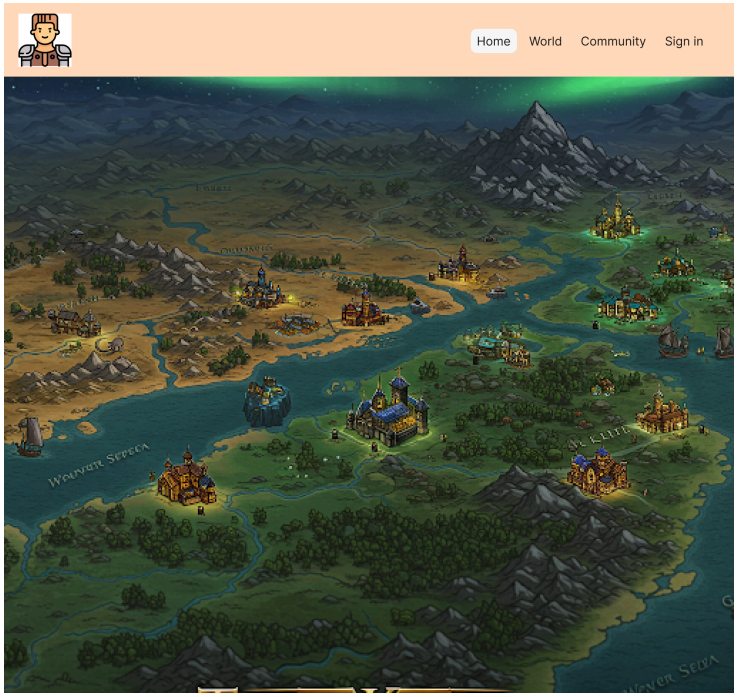
Block Diagram



Component Diagram



User Interface Storyboard



Home World Community Sign in

Choose an account

Account

Get started with Google

Use another account

Home World Community Account

Enter username

Profile Picture

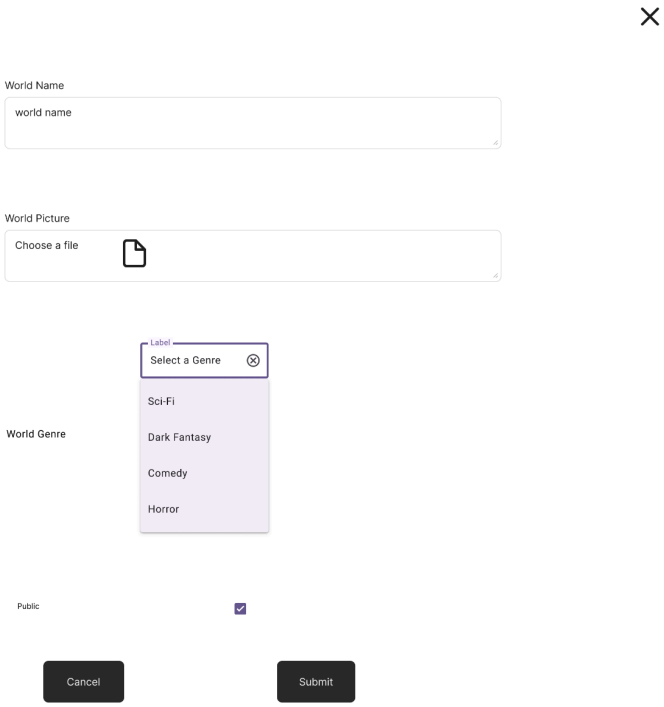
Description of themselves and their world

Create new world

Existing world, if already created

Existing world, if already created

Existing world, if already created


World Name
world name
World Picture
Choose a file
World Genre
Label
Select a Genre
Sci-Fi
Dark Fantasy
Comedy
Horror
Public
Cancel
Submit

Home World Community Sign in

Welcome Username

Maps

Chars

Orgs

Locations

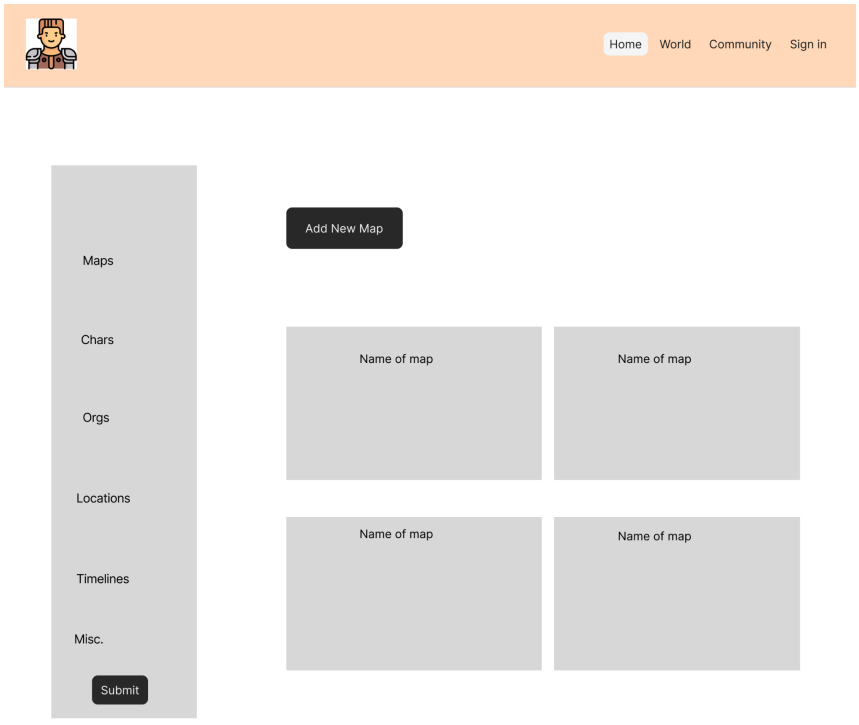
Timelines

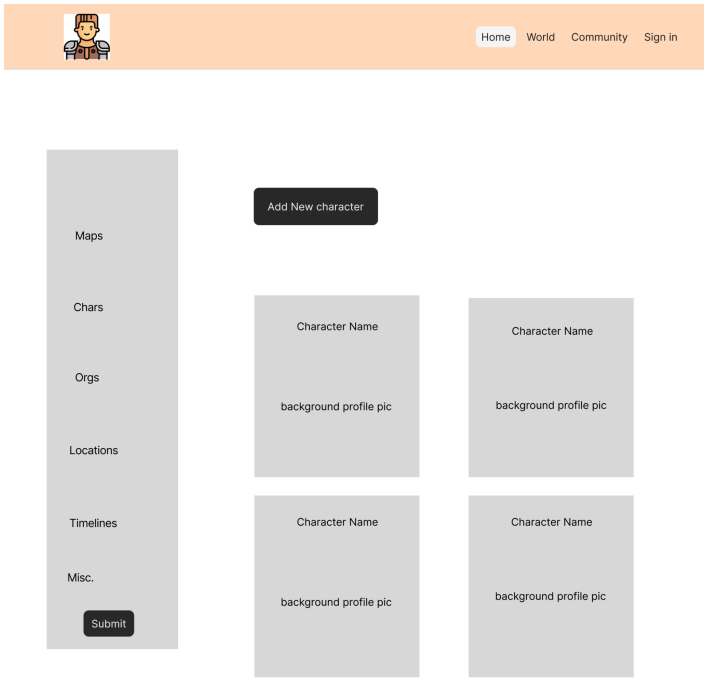
Misc.

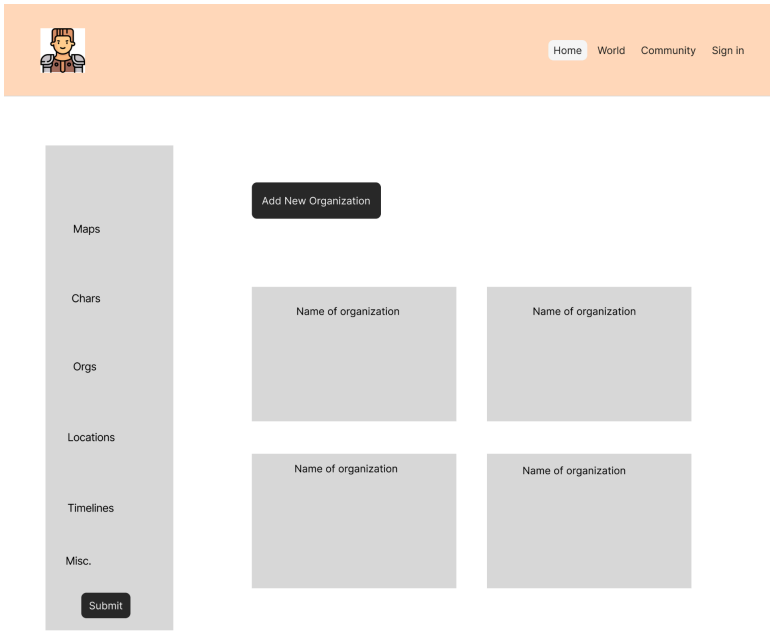
Submit

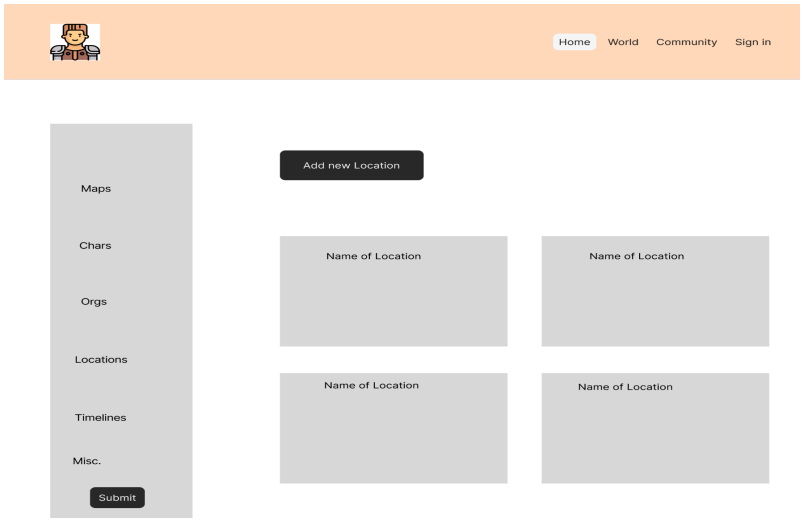
World Name

World Description


Home
World
Community
Sign in
Maps
Chars
Orgs
Locations
Timelines
Misc.
Submit
Add New Map
Name of map
Name of map
Name of map
Name of map


Home
World
Community
Sign in
Maps
Chars
Orgs
Locations
Timelines
Misc.
Submit
Add New character
Character Name
background profile pic
Character Name
background profile pic
Character Name
background profile pic
Character Name
background profile pic


Home
World
Community
Sign in
Maps
Chars
Orgs
Locations
Timelines
Misc.
Submit
Add New Organization
Name of organization
Name of organization
Name of organization
Name of organization


Home
World
Community
Sign in
Maps
Chars
Orgs
Locations
Timelines
Misc.
Submit
Add new Location
Name of Location
Name of Location
Name of Location
Name of Location

Home World Community Sign in

Maps

Chars

Orgs

Locations

Timelines

Misc.

Submit

Add new Timeline

Name of Timeline

Name of Timeline

Name of Timeline

Name of Timeline

Home World Community Sign in

Maps

Chars

Orgs

Locations

Timelines

Misc.

Submit

Add Miscellaneous

Name of Miscellaneous

Name of Miscellaneous

Name of Miscellaneous

Name of Miscellaneous

Home World Community Sign in

Maps

Map Name
Map picture in background

Map Name
Map picture in background

Map Name
Map picture in background

Characters

Character Name
character pic in background

Character Name
character pic in background

Character Name
character pic in background

Organizations

Organization Name

Organization Name

Organization Name

Locations

Location Name

Location Name

Location Name

Timelines

Timeline Name

Timeline Name

Timeline Name

Miscellaneous

Miscellaneous Name

Miscellaneous Name

Miscellaneous Name

Home World Community Sign in

World Name

Description of world

Author Genre

Date Created

Like

World Name

Description of world

Author Genre

Date Created

Like

World Name

Description of world

Author Genre

Date Created

Like

World Name

Description of world

Author Genre

Date Created

Like

Previous 1 2 3 ... 67 68 Next

Message Documentation

Adding a world

```
{
        "title": "string",
         "description": "string",
        "genre": "string",
        "isPublic": "boolean",
        "created": "Date",
        "likes": "int",
        "entries": [“Entries”]
 }
```

Example

```
{
        "title": "Mystic Realms",
         "description": A world filled with magic and mystery",
        "genre": "Fantasy",
        "isPublic": true,
        "created": "3/3/25",
        "likes": 0,
        "Entries": []
 }
```

Response

```
{
        "success": true,
         "message": "World successfully added",
        "worldID": w123,
 }
```

Adding a Character

```
{
        "portrait": "string (URL/Base64)",
         "name": "string",
        "description": "string"
}
```

Example

```
{
        "portrait": "https://example.com/images/char456.png",
```

```
    "name": "Bob",
    "description": "A mage"
}
```

Response

```
{
    "success": true,
    "message": "Character successfully added",
    "characterID": c123
}
```

Adding a User

```
{
    "username": "string",
    "description": "string",
    "profilePicture": "string (URL/Base64)",
    "worlds": ["Worlds"],
    "email": "string"
}
```

Example

```
{
    "username": "Random1",
    "description": "Enjoys map making",
    "profilePicture": "https://example.com/images/profile123.png",
    "worlds": ["w123"],
    "email": "adventurer@example.com"
}
```

Response

```
{
    "success": true,
    "message": "User successfully registered",
    "userID": u123
}
```

Adding an Entry

```
{
    "name": "string",
    "description": "string"
}
```

Example

```
{
        "name": "The Dark Forest",
        "description": "A dense and eerie forest full of unknown creatures."
}
```

Response

```
{
        "success": true,
        "message": "Entry successfully added.",
        "entryId": "e567"
}
```

Adding an Event

```
{
        "name": "string",
        "description": "string",
        "startDate": "string (ISO 8601)",
        "endDate": "string (ISO 8601)"
}
```

Example

```
{
        "name": "The Great Battle",
        "description": "A legendary battle between the elves and orcs.",
        "startDate": "2025-06-10T08:00:00Z",
        "endDate": "2025-06-11T18:00:00Z"
 }
```

Response

```
{
        "success": true,
        "message": "Event successfully added.",
        "eventId": "ev678"
}
```

Adding a Map

```
{
        "image": "string (URL/Base64)",
```

```
        "pins": [Pins],
         "name": "string",
         "description": "string"
}
```

Example

```
{
        "image": "https://example.com/images/map123.png",
         "pins": ["p001", "p002"],
        "name": "Kingdom of Eldoria",
         "description": "A vast kingdom with many hidden treasures."
}
```

Response

```
{
        "success": true,
         "message": "Map successfully added.",
         "mapId": "m999"
}
```

Adding a pin to a map

```
{
        "name": "string",
        "description": "string",
        "x": "int",
        "y": "int"
}
```

Example

```
{
         "name": "Ancient Ruins",
        "description": "A forgotten place of power.",
         "x": 450,
        "y": 320
}
```

Response

```
{
        "success": true,
         "message": "Pin successfully added.",
        "pinId": "p123"
```

```
}
```

Storage Documentation