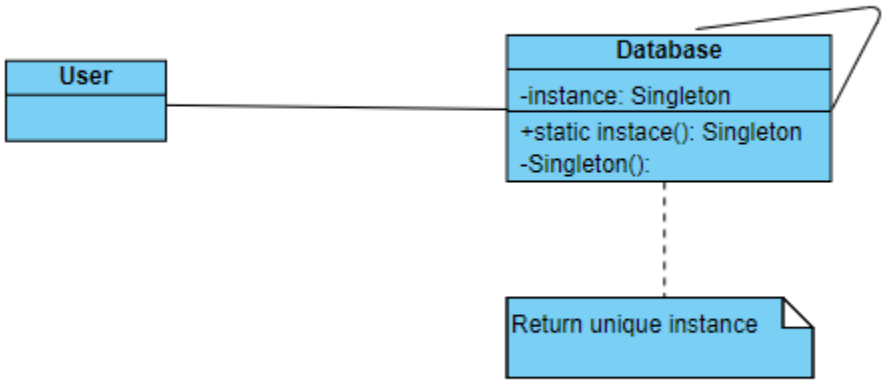
Singleton



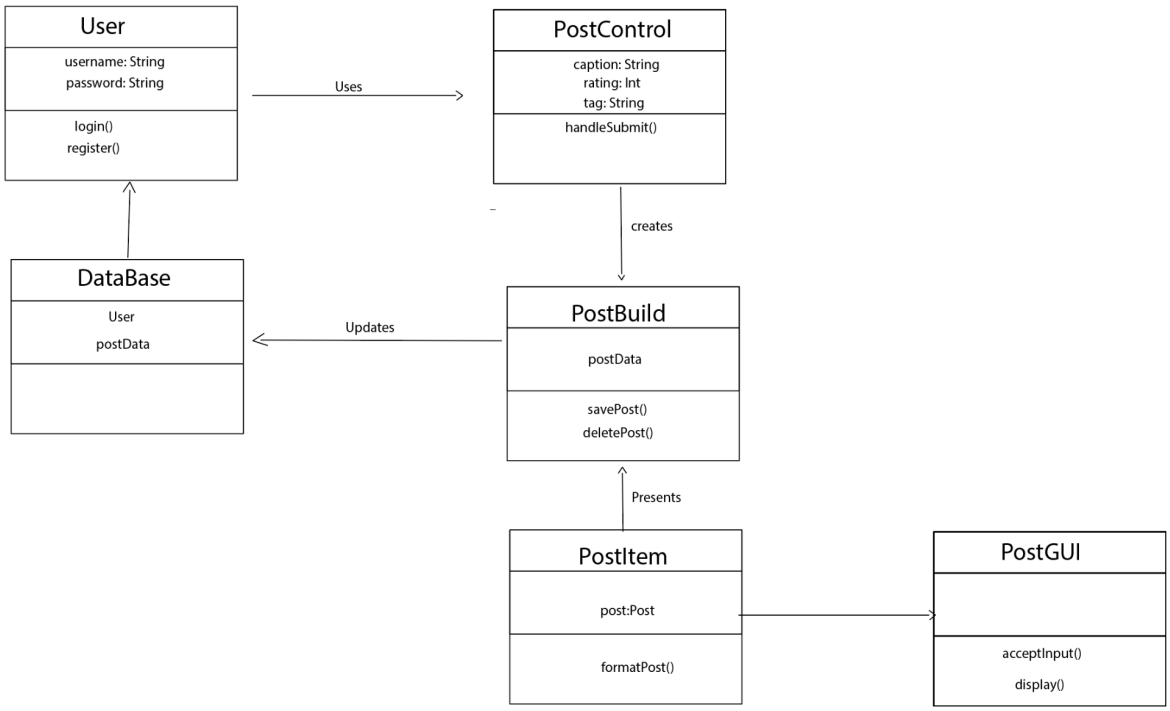
Description:

The database is a Singleton object, so the constructor of the class is private along with the instance variable being private. It has a public static method that returns a reference to this instance of the object. Users can interact with the object by asking for a reference and using the reference.

Justification:

Singleton is a good choice for our database because it causes all the users to be looking at one instance of the database, which keeps everything consistent, and that everything stored in the database is the same for every user. There can only be at most one instance of the database at one time, otherwise, posts created by users might go into a different database and not be viewable by other users.

The Container Presentation Design Pattern:



Description:

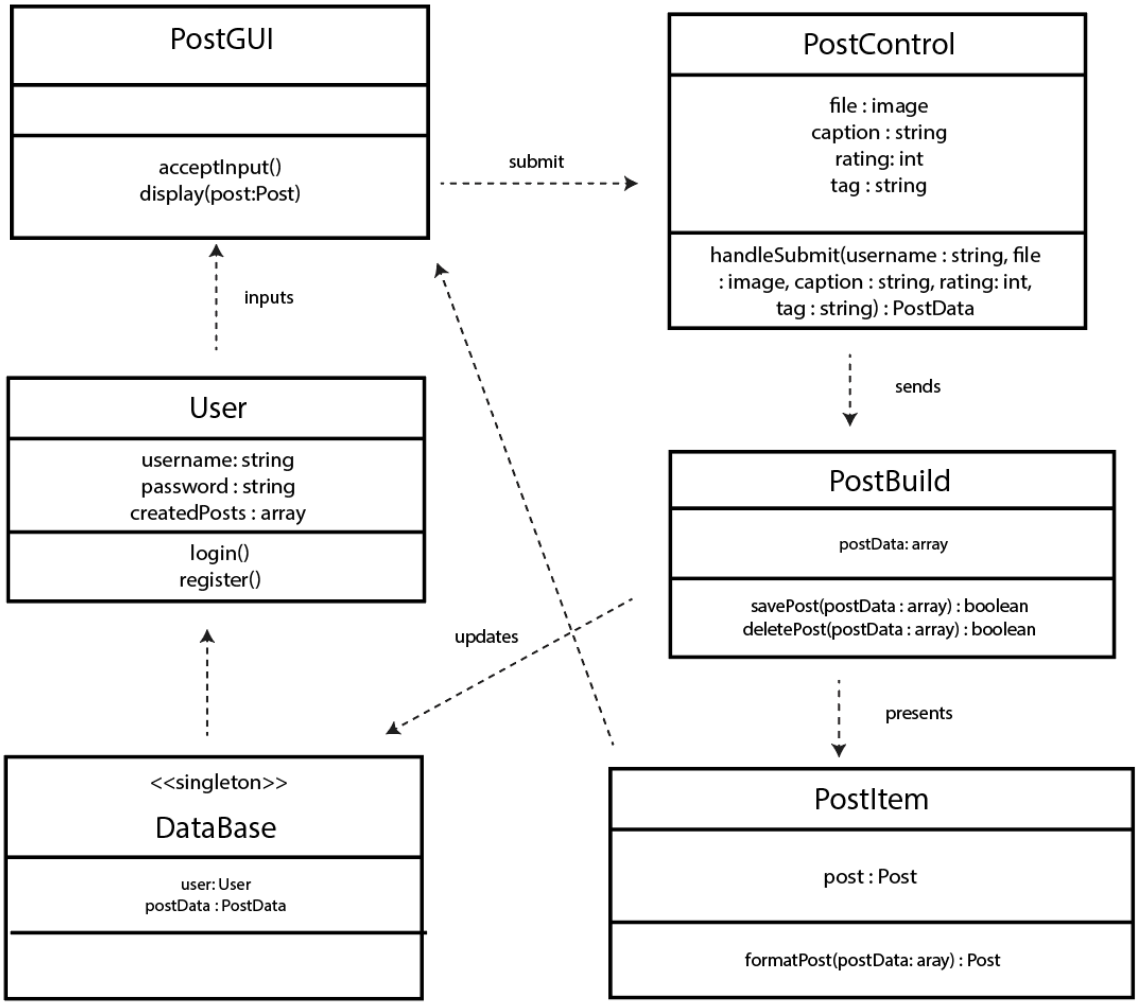
This design pattern organizes user interactions and content posting in a structured way. It starts with the User, which handles user login and registration. The PostControl manages post actions like captioning, rating, and tagging, as well as handling post submissions. The PostBuild interacts with the Database to update User and Post data. The PostItem presents posts by communicating with both the PostBuild and PostGUI components to process and display the post data.

Justification:

The Container Presentation design pattern is suitable for our website called Poppin because our code has Container classes for handling post logic and Presentation classes for

handling how a post is viewed on our website. The User part handles login and registration. Our diagram has two Container Parts: the PostControl part manages logical post actions like handleSubmit(), and the PostBuild part updates user and post data in the database via savePost() and deletePost(). These are Container components because they handle the logic of creating a post and saving it to the database. Our code has one Presentation class: the PostItem part shows individual posts with formatPost(), working to visualize the results of PostBuild and PostGUI. This structured approach makes the program easier to understand and maintain, and it allows us to individually edit the logical and visual aspects of posts. This helps us add or change features more easily.

2.Design Class Diagram:



Our DCD shows the process of a user creating a post. First, the user must have registered and logged in to access the post page. On the post page, they input their post information through the post GUI (image, caption, rating, tag). PostControl.js receives the inputted post items from the text boxes and combines them into a postData array. It then sends this array to Post.js (called PostBuild here to avoid confusion), which updates the database to include this post. Post.js also sends this postData to PostItem.js, which formats the information into a Post that can be displayed by PostUI.

Class	**Attributes**	**Functions**

User	username: string password : string createdPosts : array	login() register()
PoppinGUI		acceptInput() display(post:Post)
Database	user: User postData : PostData	
PostBuild	postData : array	savePost(postData : array) : boolean deletePost(postData : array) : boolean
PostItem	post: Post	formatPost(postData: aray) : Post
PostControl	file : image caption : string rating: int tag : string	handleSubmit(username : string, file : image, caption : string, rating: int, tag : string) : PostData