SD^2: Software Design Document

Group 2

Github repo: https://github.com/SaiManikanta1918/snapstore

1. **Project Overview:**

Snapstore is a social media platform designed to facilitate photo sharing and connection among users. In today's digital age, social media has become an integral part of people's lives, allowing them to express themselves, connect with friends and family, and discover new content. However, existing social media platforms often lack certain features or privacy controls, leading to user dissatisfaction and concerns about privacy.

Snapstore aims to address these issues by providing a user-friendly and privacy-focused platform for photo sharing and social interaction. The software targets a diverse range of stakeholders, including:

1. Users: Users are the primary stakeholders of Snapstore. They seek a platform where they can share photos, connect with friends, and discover new content while maintaining control over their privacy and personal information.

2. Administrators: Administrators are responsible for managing the Snapstore platform, ensuring its integrity, safety, and compliance with regulations. They have a stake in maintaining a positive user experience and addressing any issues or concerns raised by users.

3. Developers: Developers are involved in designing, developing, and maintaining the Snapstore software. They aim to create a robust and scalable platform that meets the needs of users and administrators while adhering to best practices in software development.

The problem that Snapstore aims to address is the lack of a privacy-focused and user-friendly social media platform for photo sharing and social interaction. Existing platforms often prioritize advertising revenue over user privacy, leading to concerns about data misuse and privacy violations. Additionally, some platforms lack essential features or customization options, limiting user engagement and satisfaction.

Snapstore will address these issues by providing the following features and solutions:

1. Privacy Controls: Snapstore will offer robust privacy controls, allowing users to customize the visibility of their profile and posts, manage their followers and following list, and control who can interact with their content.

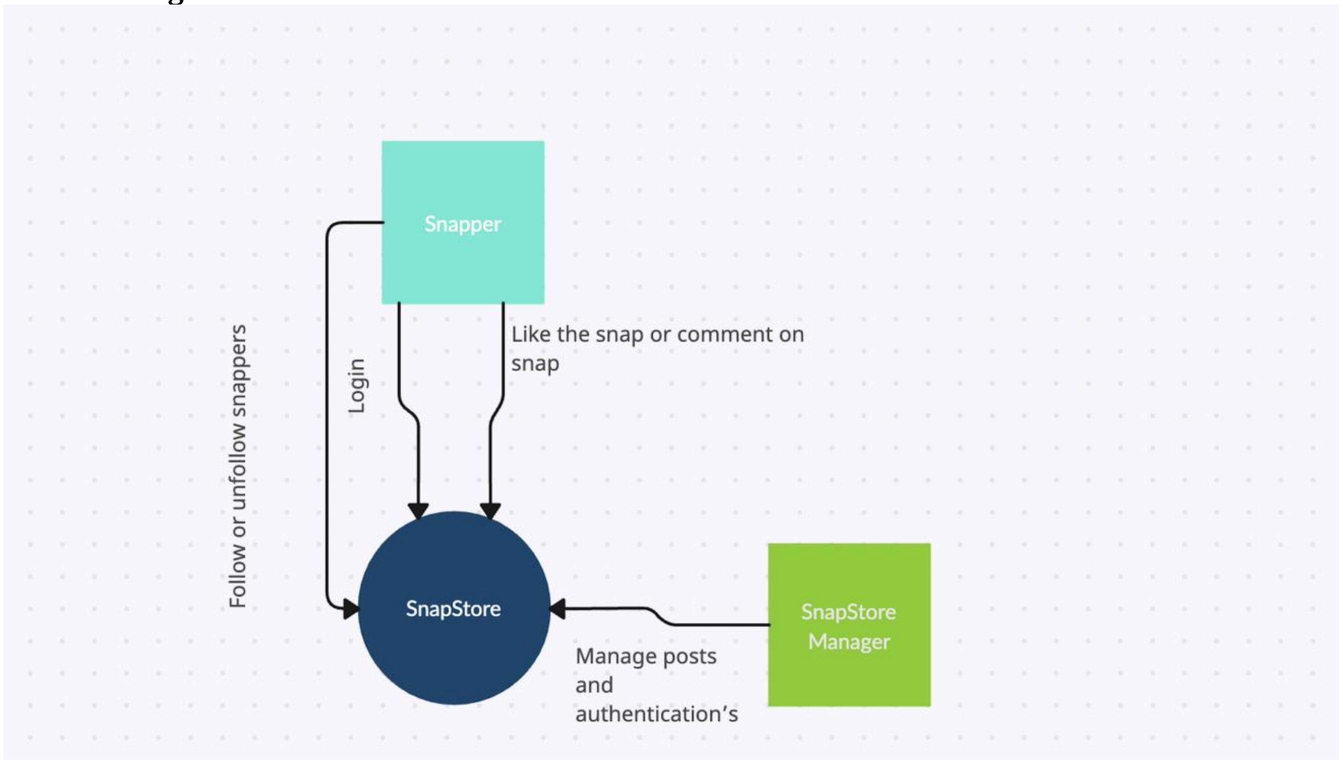
2. User-Centric Design: Snapstore will prioritize user experience and usability, with a clean and intuitive interface that makes photo sharing and social interaction seamless and enjoyable.

3. Real-Time Updates: Snapstore will support real-time updates and notifications, ensuring that users stay informed about new posts, likes, comments, and other activities on the platform.

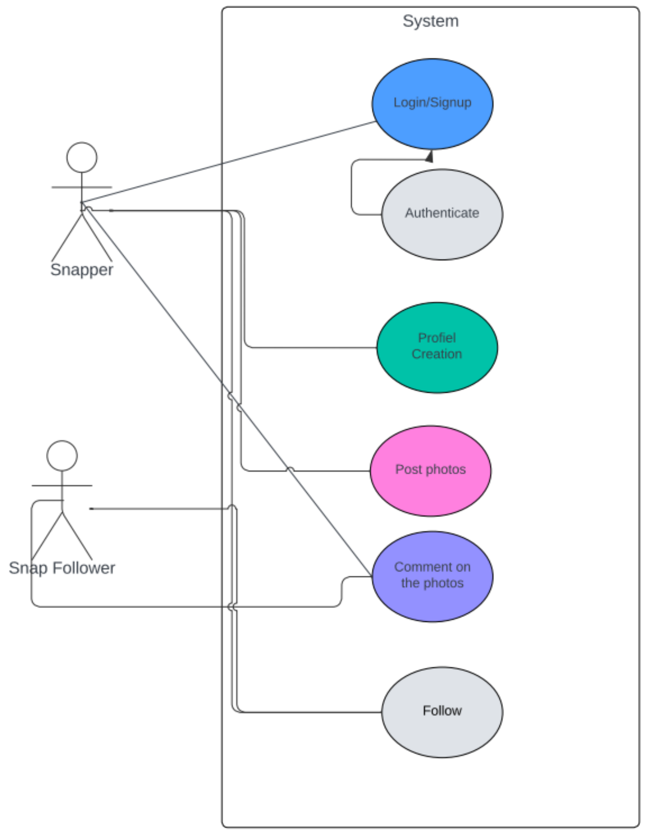
4. Community Guidelines and Moderation: Snapstore will establish community guidelines and implement moderation mechanisms to ensure a safe and respectful environment for all users.

By addressing these issues and providing innovative solutions, Snapstore aims to become the go-to platform for photo sharing and social interaction, trusted by users for its privacy-focused approach and user-friendly design.

Context Diagram:


Snapper
Follow or unfollow snappers
Login
Like the snap or comment on snap
SnapStore
SnapStore Manager
Manage posts and authentication's

Use case Diagram

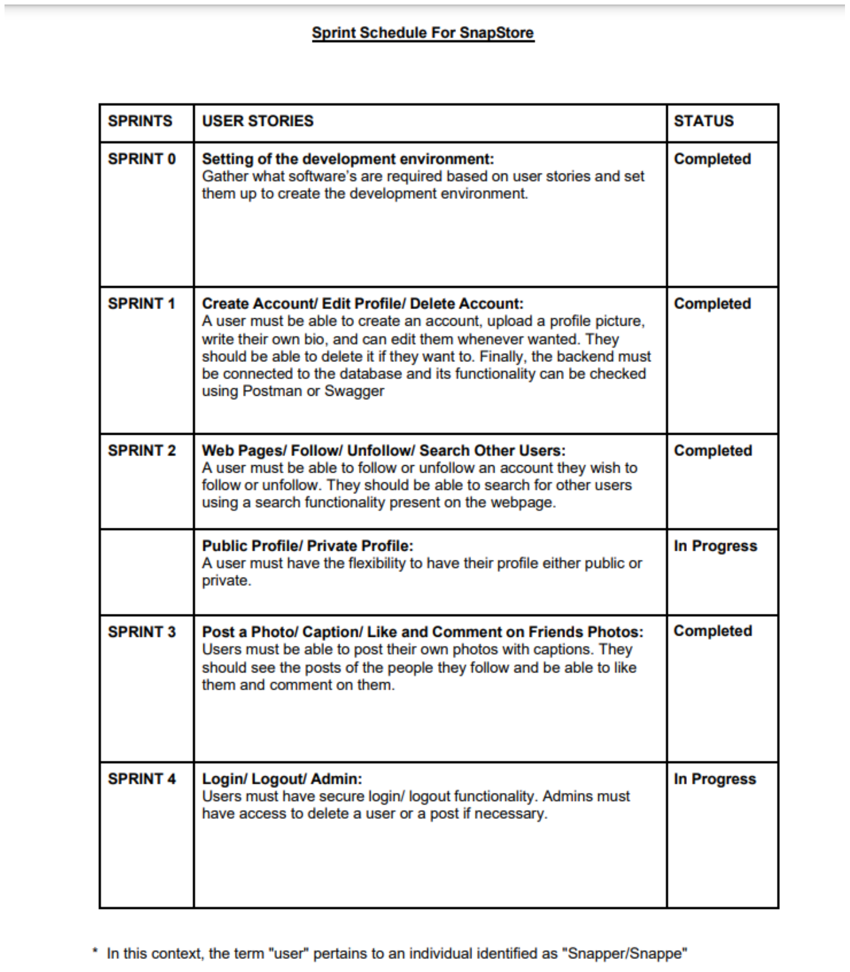


User Stories:

1. As a user, I would like to be able to create an account, so that I can start sharing my photos and connect with others on Snapstore.
2. As a user, I would like to be able to delete my account, so that my personal information and posts are no longer accessible on Snapstore.
3. As a user, I would like to be able to edit my profile, so that I can keep my information up-to-date and accurate.
4. As a user, I would like to be able to follow other users, so that I can stay updated on their posts and activities.
5. As a user, I would like to be able to unfollow users, so that I can control whose content appears on my feed.
6. As a user, I would like to have a public profile/private profile, so that others can easily find and view my posts and information so that I can protect my privacy.
7. As a user, I would like to be able to search for other users, so that I can easily find and connect with friends and other interesting accounts.
8. As a user, I would like to be able to create posts, add captions, and share them with my followers, so that I can express myself and share moments with others.
9. As a user, I would like to be able to like posts from other users, so that I can show my

appreciation for their content.
10. As a user, I would like to be able to comment on posts, so that I can engage with other users and start conversations.
11. As a user, I would like to be able to log in and out of my account, so that I can access Snapstore securely and protect my privacy.
12. As an admin, I would like to have the right to delete user accounts and posts if necessary, so that I can maintain the integrity and safety of the Snapstore platform.

Sprint Backlog:
We have implemented all the features as present in the Sprint Schedule except for the privacy of the profile.



Sprint Schedule For SnapStore

SPRINTS	USER STORIES	STATUS
SPRINT 0	**Setting of the development environment:** Gather what software's are required based on user stories and set them up to create the development environment.	Completed
SPRINT 1	**Create Account/ Edit Profile/ Delete Account:** A user must be able to create an account, upload a profile picture, write their own bio, and can edit them whenever wanted. They should be able to delete it if they want to. Finally, the backend must be connected to the database and its functionality can be checked using Postman or Swagger	Completed
SPRINT 2	**Web Pages/ Follow/ Unfollow/ Search Other Users:** A user must be able to follow or unfollow an account they wish to follow or unfollow. They should be able to search for other users using a search functionality present on the webpage.	Completed
	Public Profile/ Private Profile: A user must have the flexibility to have their profile either public or private.	In Progress
SPRINT 3	**Post a Photo/ Caption/ Like and Comment on Friends Photos:** Users must be able to post their own photos with captions. They should see the posts of the people they follow and be able to like them and comment on them.	Completed
SPRINT 4	**Login/ Logout/ Admin:** Users must have secure login/ logout functionality. Admins must have access to delete a user or a post if necessary.	In Progress

* In this context, the term "user" pertains to an individual identified as "Snapper/Snappe"

Based on the requirements gathered for Snapstore, here are the main technical use cases along with their details:

1. User Authentication:
   - Description: Users need to authenticate themselves to access Snapstore and their personalized content.
   - Details:

- Users enter their username/email and password.
- Snapstore verifies the credentials against the user database.
- Upon successful authentication, Snapstore grants access to the user's account and personalized content.

2. Create Account:

- **Description:** Users should be able to create a new account on Snapstore.
- **Details:**
  - Users provide registration details such as username, email, and password.
  - Snapstore validates the provided information (e.g., unique username, valid email format).
  - If validation passes, Snapstore creates a new user account and stores the information securely in the database.

3. Edit Profile:

- **Description:** Users should be able to edit their profile information on Snapstore.
- **Details:**
  - Users navigate to the profile editing section.
  - Users make changes to their profile information such as name, profile picture, and bio.
  - Snapstore validates and updates the modified profile data in the database.

4. Delete Account:

- **Description:** Users should have the option to delete their Snapstore account.
- **Details:**
  - Users navigate to the account deletion section.
  - Users provide authentication to verify their identity.
  - Snapstore permanently removes user data from the database.
  - Snapstore updates associated data (e.g., posts, followers) to reflect account deletion.

5. Create Post:

- **Description:** Users should be able to create and share posts on Snapstore.
- **Details:**
  - Users navigate to the post creation section.
  - Users upload an image and add a caption to the post.
  - Snapstore stores the post information in the database and associates it with the user's account.

6. Like Post:

- **Description:** Users should be able to like posts shared by other users on Snapstore.
- **Details:**
  - Users view a post and choose to like it.
  - Snapstore records the user's like action on the post.
  - Snapstore increments the like count for the respective post.

7. Comment on Post:

- **Description:** Users should be able to comment on posts shared by other users on Snapstore.
- **Details:**
  - Users view a post and enter a comment in the designated comment section.
  - Snapstore records the comment along with user details and post ID.
  - Snapstore displays the comment on the respective post.

8. Follow/Unfollow User:

- **Description**: Users should be able to follow/unfollow other users on Snapstore to stay updated on their activities.
- **Details:**
  - Users navigate to the profile of the user they want to follow/unfollow.
  - Users choose to follow/unfollow the user.
  - Snapstore updates the follower/following lists of both users accordingly.

9. **Set Profile Privacy:**

- **Description:** Users should have the option to set their profile as public or private on Snapstore.
- **Details:**
- Users navigate to the privacy settings section in their profile.
- Users toggle between public and private profile settings.
- If the user selects a public profile:
  Snapstore allows others to easily find and view their posts and information.
- If the user selects a private profile:
  Snapstore restricts access to the user's profile, allowing only approved followers to view content.
- Snapstore updates the profile visibility status accordingly in the database.

These technical use cases provide a detailed understanding of the system's functionality and how users interact with Snapstore's features. They serve as the basis for implementation and testing of the software.

2. **Architectural Overview:**

When planning Snapstore's architecture, we explored different options such as using microservices, going serverless, or sticking with one big system. After careful thought, we decided to go with the big system approach for a few reasons.

<u>Alternative Designs Considered:</u>

1. <u>Microservices Architecture:</u> Microservices architecture breaks the app into small pieces, each doing its own thing. It's good for making the app grow and change easily. But for Snapstore, it would make things too complicated and slow down making the app.

2. Serverless Architecture: Serverless architecture lets developers write code without worrying about servers. It saves money, scales well, and makes things easier to handle. But for Snapstore, it needs control over things like user logins and data storage, which serverless doesn't offer. Using serverless might lock Snapstore into using certain services and limit control over important parts of the app.

Rationale for Choosing Monolithic Architecture:

After careful consideration, we decided to implement Snapstore using a monolithic architecture for the following reasons:

1. Simplicity and Development Speed:
   _Monolithic architecture is simple and easy to work with, which is great for smaller projects like Snapstore. It helps us build, launch, and take care of the app faster, letting us concentrate on adding features and making users happy.

2. Tight Integration and Communication:
   _Snapstore needs its different parts, like user logins, creating posts, and working with the database, to work together smoothly. With a monolithic architecture, it's easier for these parts to communicate well, making development simpler and reducing extra work.

3. Initial Development and Iteration:
   With monolithic architecture, we can rapidly try out and improve new features, making it easy to test and implement changes. This method fits well with our agile development approach, enabling us to deliver updates gradually and collect user feedback early on in the development process.

4. Resource Efficiency:
   Using a monolithic architecture for Snapstore helps us use resources efficiently and keeps operational tasks manageable. This approach is more cost-effective in the long term compared to more complicated options like microservices or serverless architecture, given Snapstore's size and needs.

2.1)

Architectural styles are fundamental design principles that dictate the overall structure, organization, and interaction patterns of software systems. They provide a high-level blueprint for organizing the components of a system and defining how they communicate and interact with each other. Architectural styles encapsulate common design patterns, principles, and constraints that guide the development of software systems. Some common architectural styles include:

Client-Server Architecture: In this style, the system is divided into two separate components: a client, which is responsible for presenting information to the user and handling user interactions, and a server, which is responsible for processing requests, executing application logic, and managing data storage. Communication between the client and server typically occurs over a network, such as the internet.

Layered Architecture: Also known as the n-tier architecture, this style organizes the components of a system into multiple layers, with each layer responsible for a specific set of tasks. For example, a typical three-tier architecture consists of a presentation layer (frontend), a business logic layer (backend), and a data access layer (database). Each layer only communicates with adjacent layers, promoting modularity, separation of concerns, and maintainability.

Microservices Architecture: This style decomposes a system into a collection of small, loosely coupled services, each responsible for a specific domain or functionality. Each service operates independently and communicates with other services through lightweight protocols such as HTTP or messaging queues. Microservices architecture promotes scalability, flexibility, and resilience by allowing services to be developed, deployed, and scaled independently.

Client-Server Architecture:

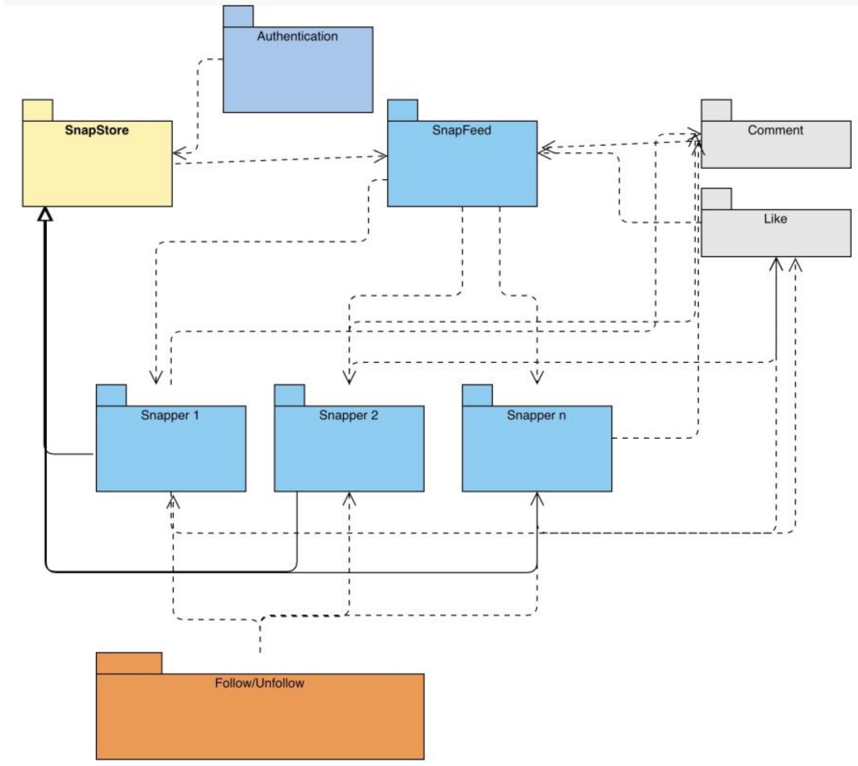
Description: Snapstore employs a client-server architecture where client devices (browsers) interact with a centralized server to request and receive data.

Rationale: This architectural style facilitates the separation of concerns between the client-side presentation logic (frontend) and the server-side business logic (backend). It enables easier maintenance, scalability, and flexibility in the system. Additionally, client-server architecture supports the real-time synchronization of data between clients and the server, which is essential for a social media platform like Snapstore.

RESTful Architecture:

Description: Snapstore's backend follows a Representational State Transfer (REST) architectural style, where resources are identified by URIs, and operations (GET, POST, PUT, DELETE) are performed using HTTP methods.

Rationale: RESTful architecture provides a standardized approach to designing web services, making it easier to develop, maintain, and scale the backend of Snapstore. By adhering to REST principles, Snapstore ensures interoperability, modifiability, and scalability of its API, enabling seamless integration with various client applications and future expansion of features.



2.2)

The key elements of our system, including the frontend, backend, and database, are distributed across different processors. The frontend operates within client browsers, while the backend and database function on a server. These components communicate over the internet, with the frontend executing in client browsers and the backend logic and database hosted on Firebase-managed servers. Communication between the frontend and backend/database occurs via the HTTP protocol.

<u>Communication Protocol:</u>

Snapstore utilizes the HTTP protocol for communication between the client browser (frontend) and Snapstore server (backend). HTTP is a widely accepted protocol for web communication, providing a standardized method for exchanging data between clients and servers. It is preferred due to its simplicity, ease of implementation, and compatibility with various platforms and technologies. Moreover, HTTP's stateless nature aligns with the request-response model of web applications like Snapstore.

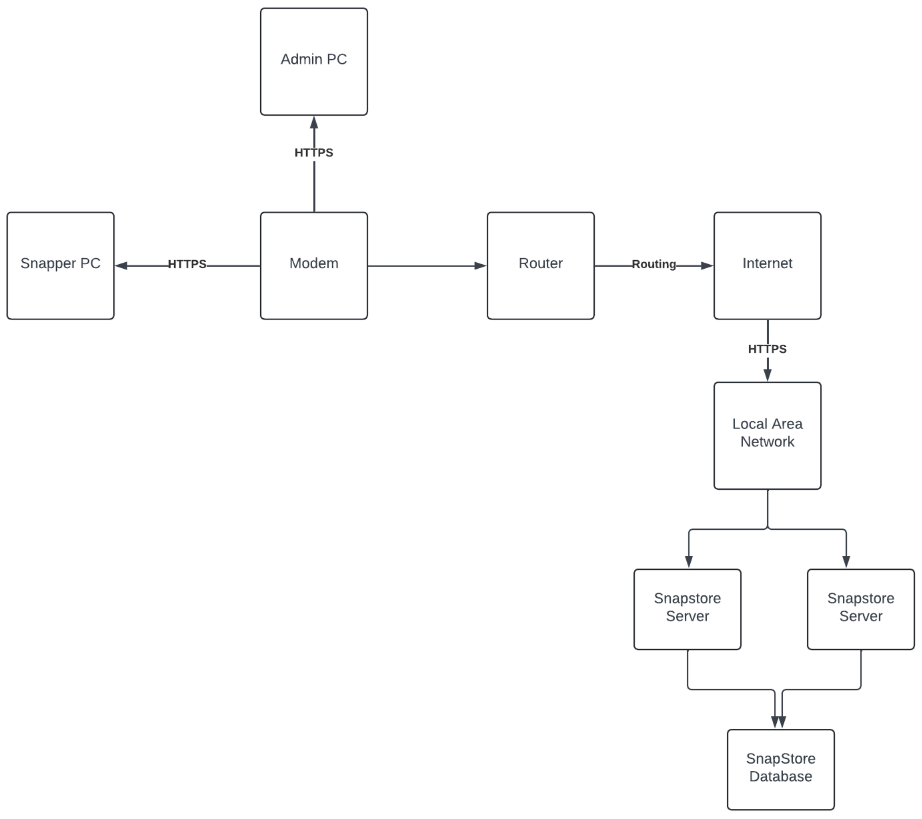
Reasoning for Choosing HTTP Protocol:

1. Compatibility: HTTP is universally supported by modern web browsers and servers, making it suitable for facilitating communication between Snapstore's client and server components.

2. Simplicity: HTTP offers a clear and easily understandable communication framework, simplifying the development, debugging, and upkeep of the Snapstore application.

3. Statelessness: HTTP operates in a stateless manner, meaning each client requests the server to be independent and doesn't rely on past interactions. This characteristic harmonizes with the stateless nature of web applications, aiding in scalability and load distribution.

4. Security: While HTTP alone lacks inherent security measures, it can be paired with HTTPS (HTTP Secure) to provide encryption and ensure secure data transmission between the client and server, safeguarding user information's confidentiality and integrity over the network.



2.3)

For Snapstore application, we utilize Firebase as our database solution for persistent data storage. Firebase provides a NoSQL cloud database that allows for real-time synchronization and offline data persistence, making it suitable for our application's requirements. Here's an overview of the data that must be stored and the database schema used in Firebase:

Data to be Stored:

User Data:

User information such as username, email, profile picture, bio.

Authentication credentials (handled by Firebase Authentication).

User preferences/settings.

Post Data:

Posts shared by users, including captions, images, timestamps, and other metadata.

Likes and comments associated with each post.

Follower/Following Relationships:

Information about users following other users and vice versa.

Approach:

We store data in Firebase's Realtime Database, which is a NoSQL cloud database that allows for flexible data structures and real-time synchronization. Firebase provides SDKs for various platforms (including JavaScript for web applications), allowing easy integration with our React.js frontend.

Database Schema (Firebase Realtime Database):

Based on the schema you've provided, it appears that you're using a NoSQL database structure, likely Firebase Realtime Database or a similar document-oriented database. Here's a breakdown of the schema and how the data is organized:

User Collection:

Attributes:

userId: Unique identifier for each user.

bio: User's biography.

createdAt: Timestamp indicating when the user account was created.

email: User's email address.

followers: List of user IDs representing followers.

following: List of user IDs representing users followed by the user.

fullName: User's full name.

posts: List of post IDs created by the user.

profilePicUrl: URL to the user's profile picture.

username: User's username.

<u>Post Collection:</u>

<u>Attributes:</u>

postId: Unique identifier for each post.

createdAt: Timestamp indicating when the post was created.

caption: Caption text accompanying the post.

imageUrl: URL to the image associated with the post.

comments: List of comment IDs associated with the post.

tags: List of tags associated with the post.

likes: List of user IDs representing users who liked the post.

saves: List of user IDs representing users who saved the post.

createdBy: User ID of the user who created the post.

key: Additional key attribute.

<u>Comment Collection:</u>

<u>Attributes:</u>

commentId: Unique identifier for each comment.

createdAt: Timestamp indicating when the comment was created.

createdBy: User ID of the user who created the comment.

postId: ID of the post to which the comment belongs.

Database Structure:

The database follows a document-oriented structure, with collections (user, post, comment) containing individual documents representing entities (users, posts, comments).

Relationships between entities are established using references (e.g., userId, postId) stored within each document.

Collections are indexed by their respective IDs (userId, postId, commentId) for efficient retrieval and querying.

Overall, this schema efficiently organizes user data, posts, and comments within a NoSQL database, facilitating real-time updates and flexible querying for Snapstore's functionalities. The denormalized structure and use of document-oriented storage align well with the requirements of a social media application like Snapstore, where fast retrieval and scalability are crucial.

2.4)

Here are the assumptions about the control flow of your system:

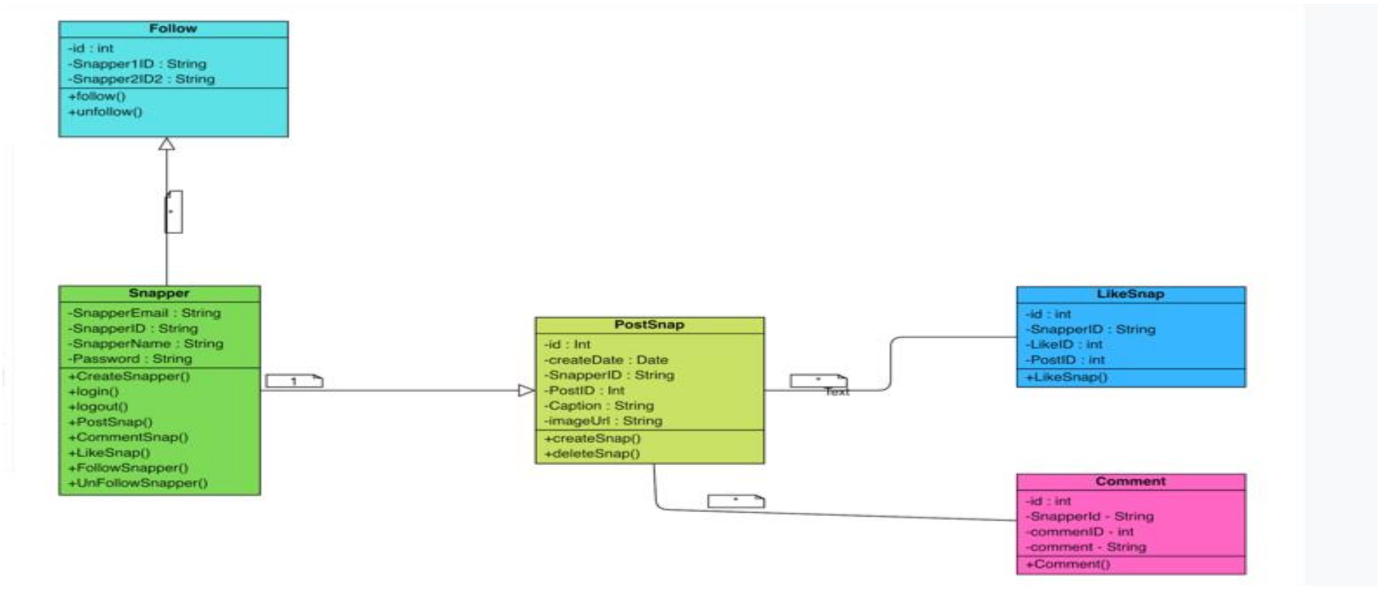
1. Procedural or event-driven: Our system appears to be primarily event-driven. Each user interacts with the system by generating various actions such as creating an account, posting, commenting, etc. These actions trigger events in the system which are then processed accordingly. Users can generate these actions in different orders based on their interactions with the platform.

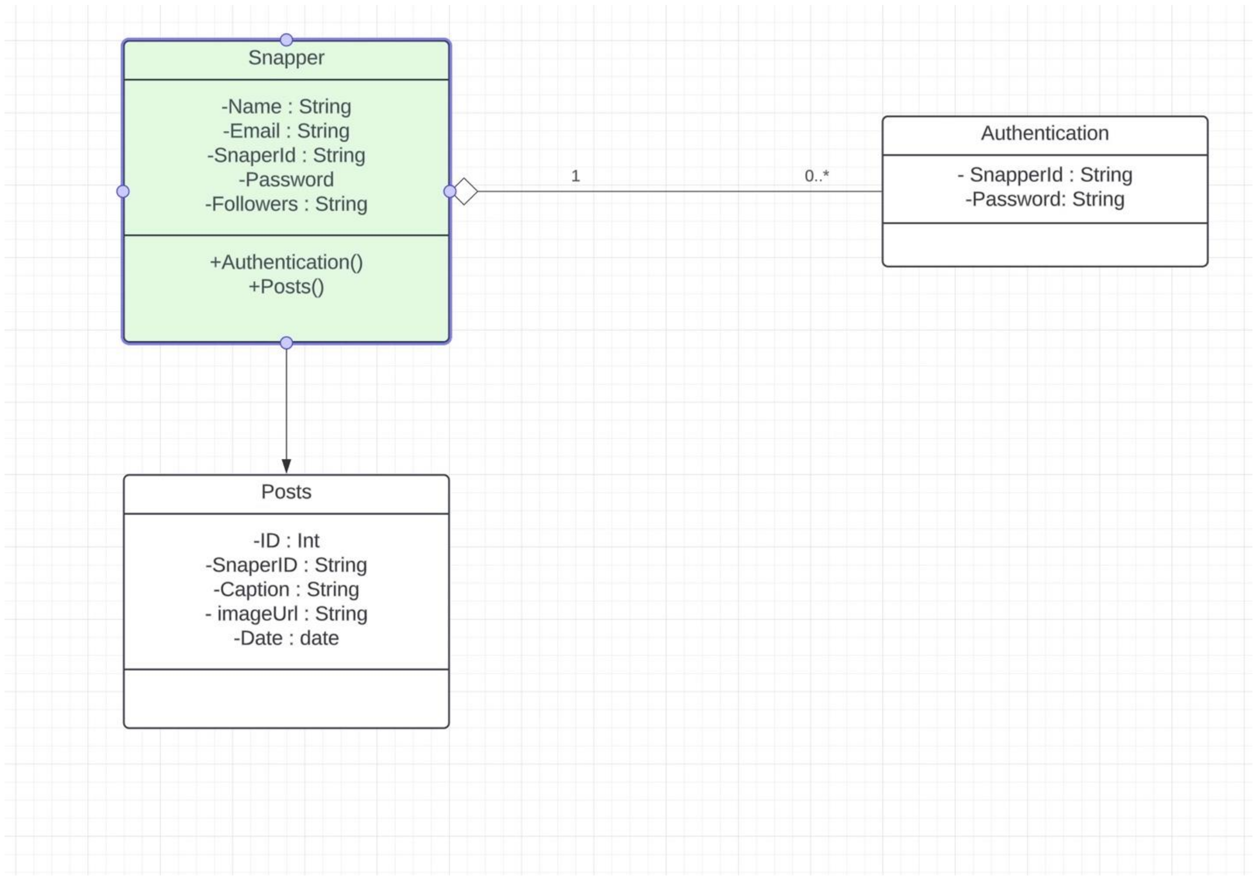
2. Time dependency: Our system is event-driven, with no strict time dependency. While there may be timestamping of posts or tracking of user activities, the system doesn't seem to have any specific timer-controlled actions or real-time constraints. Users can interact with the platform at their convenience, and there's no periodic real-time processing required.

3. Concurrency: Since our system doesn't use multiple threads, the execution is single-threaded. This means that the system processes one request or event at a time, sequentially. There are no separate threads of control for different components, and therefore no need for synchronization between threads.

3 Detailed System Design

3.1 Static view

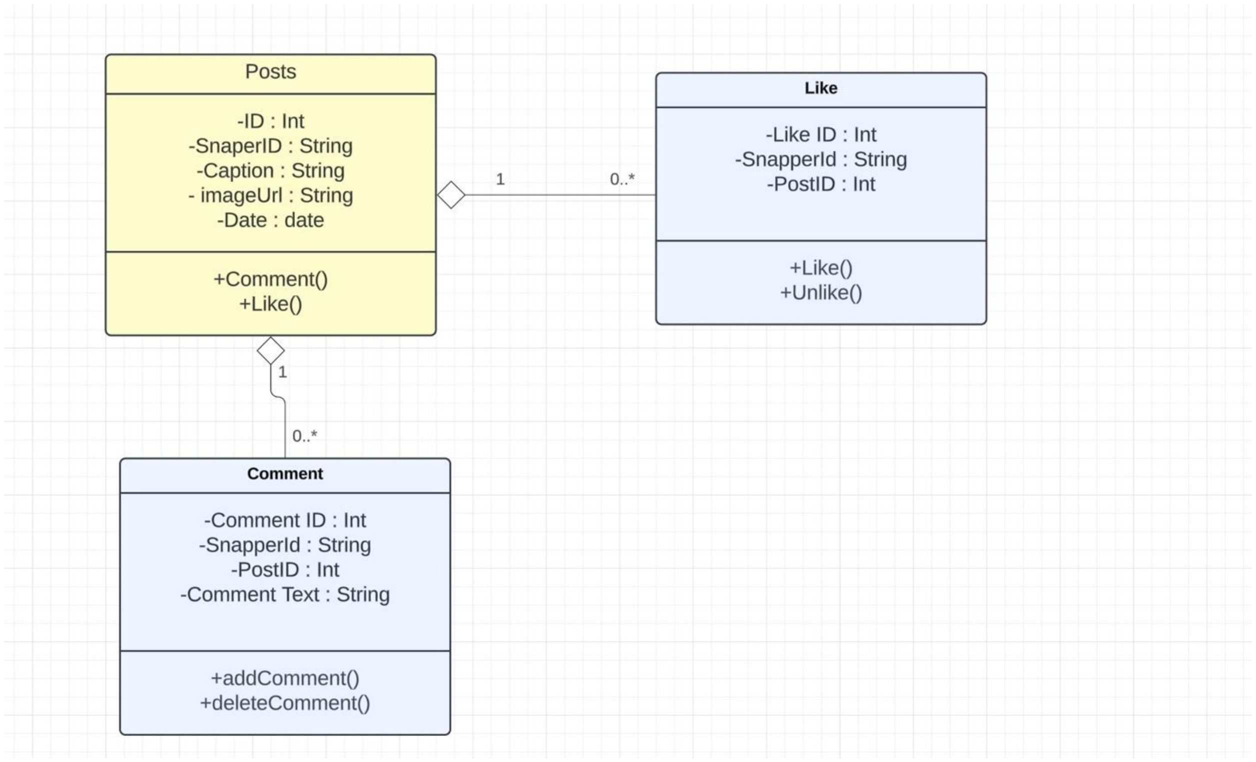

Follow
-id : int
-Snapper1ID : String
-Snapper2ID2 : String
+follow()
+unfollow()
*
Snapper
-SnapperEmail : String
-SnapperID : String
-SnapperName : String
-Password : String
+CreateSnapper()
+login()
+logout()
+PostSnap()
+CommentSnap()
+LikeSnap()
+FollowSnapper()
+UnFollowSnapper()
1
PostSnap
-id : Int
-createDate : Date
-SnapperID : String
-PostID : Int
-Caption : String
-imageUrl : String
+createSnap()
+deleteSnap()
*
Text
LikeSnap
-id : int
-SnapperID : String
-LikeID : int
-PostID : int
+LikeSnap()
*
Comment
-id : int
-SnapperId - String
-commenID - int
-comment - String
+Comment()


Snapper
-Name : String
-Email : String
-SnaperId : String
-Password
-Followers : String
+Authentication()
+Posts()
1
0..*
Authentication
- SnapperId : String
-Password: String
Posts
-ID : Int
-SnaperID : String
-Caption : String
- imageUrl : String
-Date : date

Snapper Class: Represents a user of the system. It encapsulates user attributes such as id, username, email, and password.

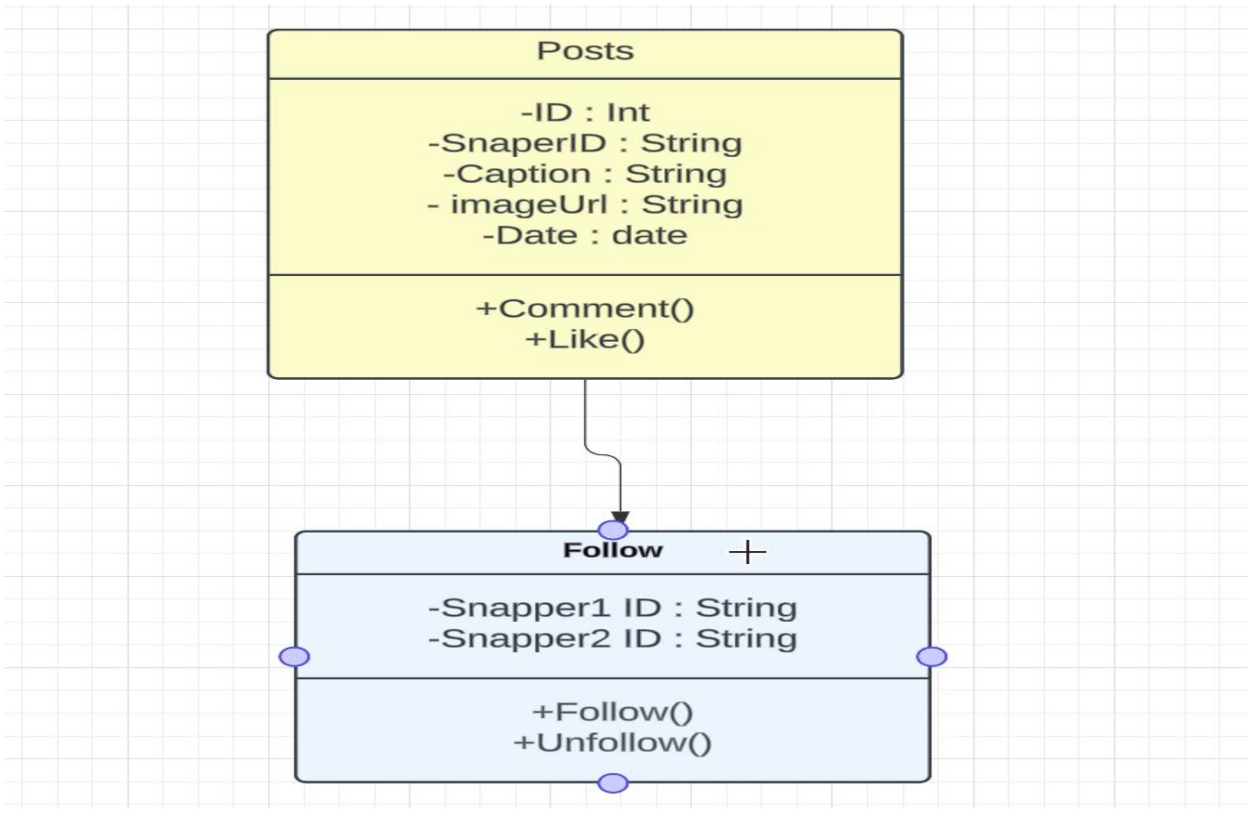
Authentication Class: Manages user-related operations like registration, authentication.

Post Class: Represents a post made by a Snapper. It contains attributes such as id, user_id , caption, image_url, and created_at.



Comment Class: Represents a comment made by a snapper on a post. It contains attributes such as id, Snapper_id, post_id, content.

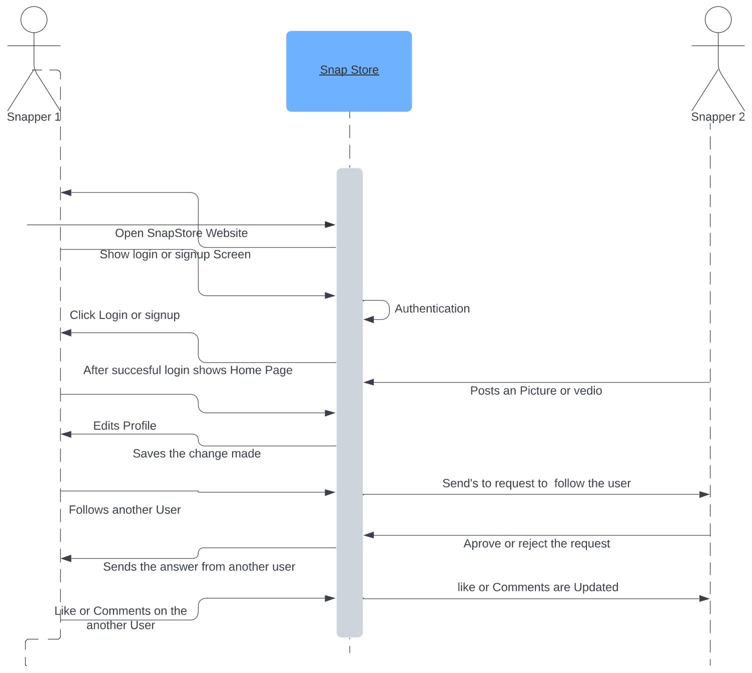
Like Class: Represents a like made by a snapper on a post. It contains attributes such as id, Snapper_id, post_id, like_id.



Follow Class: Represents the relationship between users, indicating who follows whom. It contains attributes like Snapper1_id, Snapper2_id.

This design follows a simple and straightforward approach, allowing for easy maintenance and scalability.

3.2 Dynamic view


Snap Store
Snapper 1
Snapper 2
Open SnapStore Website
Show login or signup Screen
Authentication
Click Login or signup
After succesful login shows Home Page
Posts an Picture or vedio
Edits Profile
Saves the change made
Send's to request to follow the user
Follows another User
Aprove or reject the request
Sends the answer from another user
like or Comments are Updated
Like or Comments on the another User

- When an snapper open the application it asked to login or signup.
- After successful authentication they are directed to the Feed page where they see the snaps of other snappers
- They can go search bar and search for another snapper posts or snappers
- When the profile changes are made they are updated in the database.
- Snapper can interact with feature like following other Snappers
- They also can Like and comment the posts of other snappers