# Software Requirements Analysis
# Communities : A Social Media Platform
# SM02


Harshit Pant
CS21BTECH11021

Satpute Aniket Tukaram
CS21BTECH11056

Mahin Bansal
CS21BTECH11034

Burra Vishal Mathews
CS21BTECH11010


## 1 Overview

### 1.1 System Overview:

The purpose of this software application is to provide a platform where users can create communities of their choice and induldge in communities of similar interests.This product intends to be a social media platform where users can create communities of their choice with different visibilities and induldge in communities of similar interests. Users can also interact with other users through direct messaging.
Inorder to prevent malicious users, the platform also employs a moderation system where users can report posts and comments and moderators can moderate posts and comments in their communities. The product also intends to employ a recommender system to recommend communities and create a user-specific feed for each user. The product also provides a end-to-end encryption for the direct messaging system. The product is a web application and hence is intended to be independent of any underlying operating system.

The principal actors in the system are:

- Guest Users: Users which have not created an account on the platform.
- Registered Users: Users which have created an account on the platform.
- Community Admins: Users which have created a community on the platform.
- Community Moderators: Users which have been appointed as moderators in a community.
- Superuser: Users which have the highest privileges on the platform.
- System: The system is also an actor in the system.

### 1.2 System Context:

The system aims to provide a moderated social media platform where people with similar likings can for communities and organize themselves into groups. The system also focuses on providing a user-specific feed and a direct messaging system which is encrypted and secure and cannot be read/modified by the server. Moderation system is also in place to prevent malicious users from posting inappropriate content.

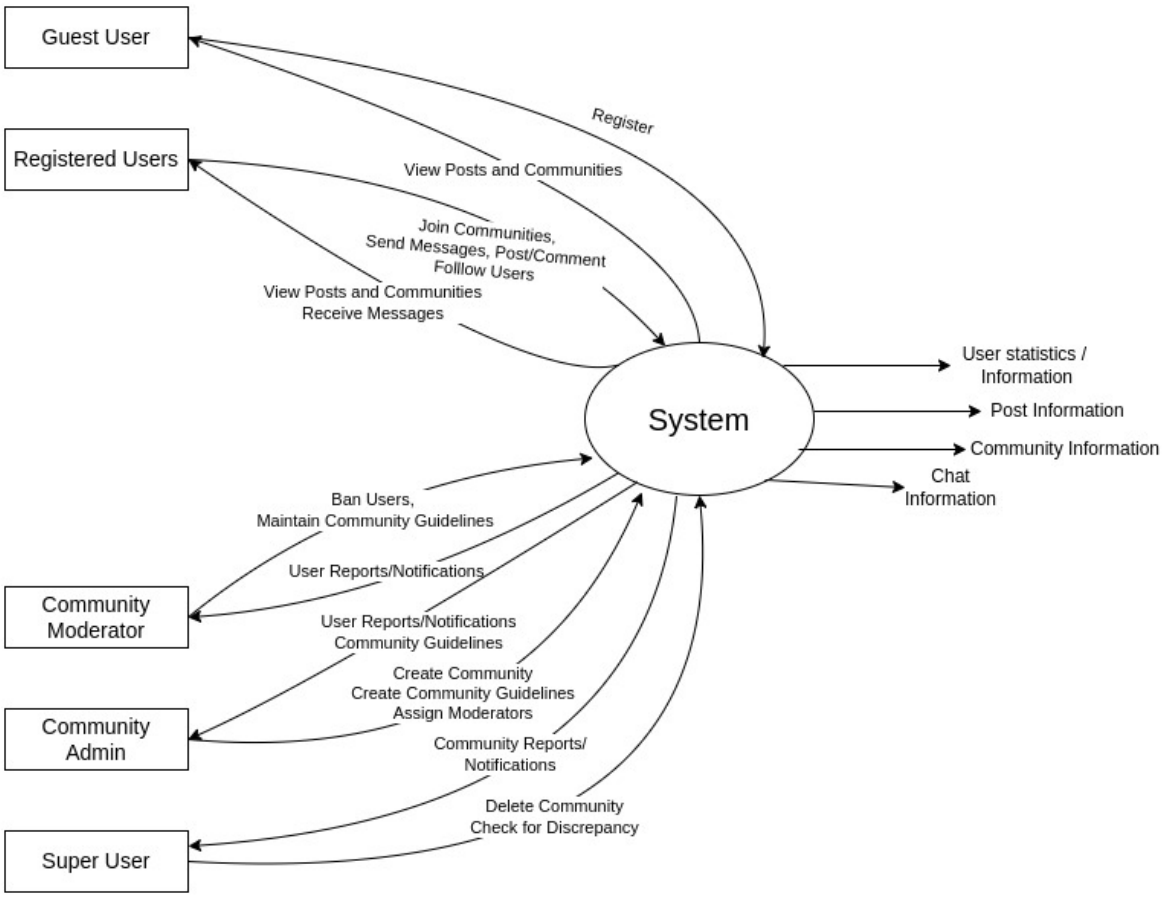


Figure 1: System Context Diagram

### 1.3 Stakeholders of Communities:

The main stakeholders of the system are:

- **Guest Users**
  - Guest Users can view public communities, posts and other registered users with public profiles.
  - Guest Users can view the Trending page.
- **Registered users**
  - Registered can create communities with different visibilities and can join communities.
  - Users can create posts in communities and perform other CRUD operations. They can comment on posts.
  - Users can upvote and downvote posts and comments.
  - Users can report posts and comments.
  - Users can search for communities, users and posts.
  - Users can follow other users.
  - Users can view their user-specific-feed and their notifications.
  - Users can use a direct messaging system to communicate with other users

- **Community moderators**
  - Moderators can moderate posts and comments in their communities.
  - Community moderators can ban users from their communities.
  - Community moderators can issue warnings to members of their communities.
- **Community admins**
  - Community admins can add and remove moderators from their communities.
  - Community admins can delete their communities.
  - Community admins can revoke user-bans.
- **Platform admins/superusers** can delete users and communities.

### 1.4 Scope of this Document:

In this document we describe architecture for our software. For the purpose of this document, we will be considering the component-connector diagrams to analyze the proposed architecture of the system.

### 1.5 Definitions and Acronyms:

- **CRUD** : Create, Read, Update, Delete
- **API** : Application Programming Interface
- **BLOB** : Binary Large Object
- **Microservices** : A software development technique—a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.
- **Metadata** : Data that provides information about other data.
- **HTTP** : HyperText Transfer Protocol
- **End-to-End Encryption** : A system of communication where only the communicating users can read the messages.
- **Recommender System** : A subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item.
- **Moderation System** : A system to prevent malicious users from posting inappropriate content.
- **Direct Messaging** : A form of communication between two people that is private and secure.
- **Community** : A group of people with a common interest.
- **Cache**: A software component that stores data so that future requests for that data can be served faster.
- **Guest User**: A user which has not created an account on the platform.
- **Registered User**: A user which has created an account on the platform.

- **Community Admin**: A user which has created a community on the platform.
- **Community Moderator**: A user which has been appointed as a moderator in a community.
- **Superuser**: A user which has the highest privileges on the platform.
- **ATAM**: Architecture Tradeoff Analysis Method

# 2 Architecture Design

## 2.1 Architecture 1: The Repository Model

- This architecture revolves around the data repository, which contains communities data, user data, conversation data, roles data for communities, posts/comments data for communities and authorization data
- All other components are connected to data repository through different connectors

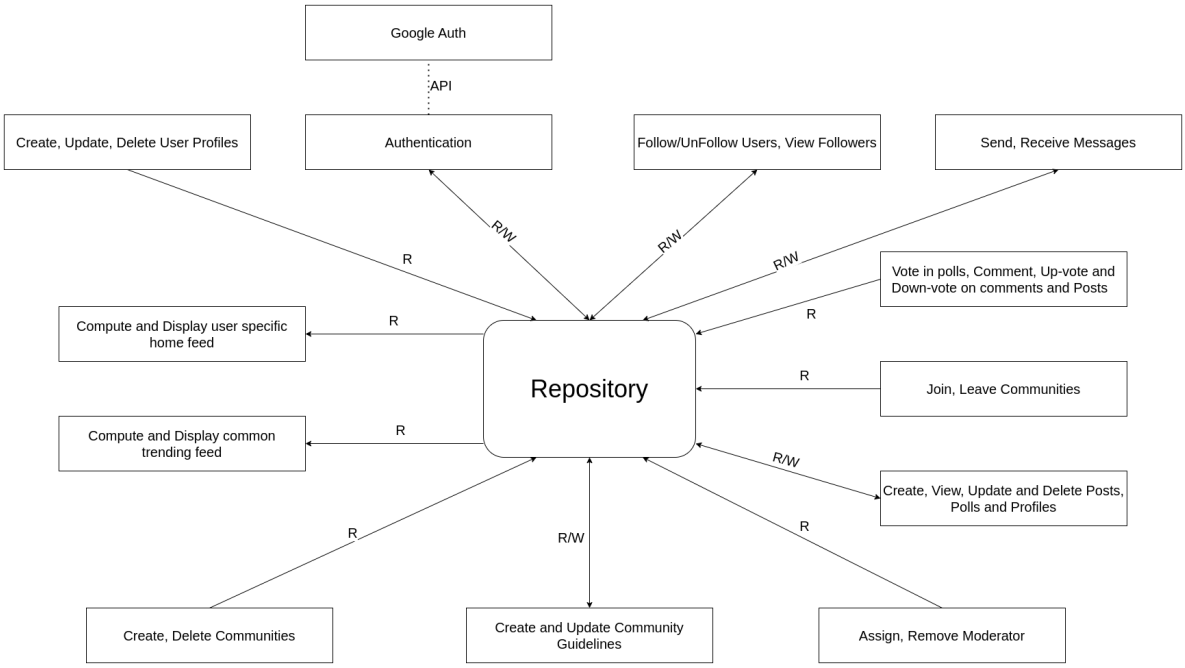


Figure 2: Architecture 1: The Repository Model

Table 1: Component Description

| Component | Component Type | Description |
|---|---|---|
| **Data Repository** | Database | This module contains communities data, user data, conversation data, roles data for communities, post/comments data for communities, authorization data for users |
| **Create, Update, Delete User Profiles** | Database Insertion | This module creates, deletes and updates user profiles for which it deletes data from repository or insert different parameters in data repository |
| **Follow/UnFollow Users, View Followers** | Database Insertion and Database Access | This module allows user to view followers of other users and follow/unFollow other users in system |
| **Send, Receive Messages** | Database Access | This module provides a messaging interface to users to send and receive messages from different users |
| **Compute and Display user specific home feed** | Processing(database access) | This modules creates home feed specific to each user by using the people and communities they follows. |
| **Compute and Display common trending feed** | Processing(database access) | This modules creates trending feed common for all users by using the recent activity in communities. |
| **Vote in polls, Comment, Up-vote and Down-vote on comments and Posts** | Database Access | This module enables user to vote in polls created by other users, Comment on different posts, up-vote or down-vote on comments and posts created by other users |
| **Join, Leave Communities** | Database Access | This module enables users to send join requests or join directly to communities and leave communities joined before |

Continued on next page

Table 1: Component Description (Continued)

| Component | Component Type | Description |
|---|---|---|
| **Create, View, Update and Delete Posts, Polls and Profiles** | Database Access | This module enables user too to create posts for communities, create polls, delete and update these posts |
| **Assign, Remove Moderator** | Database Access | This module enables community admin to assign new moderators and remove existing moderators |
| **Create and Update Community Guidelines** | Database Access | This module enables community admin to create and update respective community guidelines |
| **Create, Delete Communities** | Database Access | This module enables users to create communities and community admins to delete communities |
| **BLOB Storage** | Database | This module stores images and other media files |
| **Chat DB** | Database | This module stores the encrypted messages sent by users |
| **E2E Encryption** | Processing | This module encrypts and decrypts the messages sent by users |
| **Cache** | Database | This module stores frequently accessed data to improve the performance of the system |
| **Cache Manager** | Processing | This module manages the cache and updates it when the data changes |
| **Recommender System** | Processing | This module recommends communities to users based on their interests and activities |
| **Job Queues** | Processing | This module manages the background jobs such as sending notifications,managing comments, updating the cache, etc. |

Continued on next page

Table 1: Component Description (Continued)

| Component | Component Type | Description |
|---|---|---|
| **API Gateway** | Processing | This module acts as a single entry point for all the API requests and routes the requests to the appropriate microservice |
| **Microservices** | Processing | This module contains the business logic of the application and is divided into multiple microservices such as user service, community service, post service, etc. |
| **Authenticaiton Service** | Processing | This module handles the authentication of the users and generates the JWT tokens |
| **Cookies** | Processing | This module stores the session information of the users |
| **Scoring Service** | Processing | This module calculates the score of the posts and comments based on the number of upvotes, downvotes, and the time of posting |
| **Connections Service** | Processing | This module manages the connections between the users such as followers, following, etc. |
| **Feed Service** | Processing | This module generates the user-specific feed and the trending feed |
| **Comments Service** | Processing | This module manages the comments trees on posts. |
| **Posts Service** | Processing | This module manages the posts and polls in the communities |
| **Listing Service** | Processing | This module manages the listing of the search results and layout of the communities and posts. |

Continued on next page

Table 1: Component Description (Continued)

| Component | Component Type | Description |
|---|---|---|
| **Report Service** | Processing | This module manages the reporting of the posts and comments by the users |
| **Votes Service** | Processing | This module manages the voting on the posts and comments by the users |
| **Chat Service** | Processing | This module manages the direct messaging system |

Table 2: Connector Description

| Connector | Component Type | Description |
|---|---|---|
| **R** | Database Access | Only reading the the data from repository |
| **R/W** | Database Access and Modification | Both reading and writing data from repository |
| **API** | HTTP | External authentication through google api and communication protocol |
| **Pipe** | Pipeline | Data flow from one component to another |
| **Control** | RPC | Calling a function on the server |

### 2.2 Architecture 2: Layered Architecture

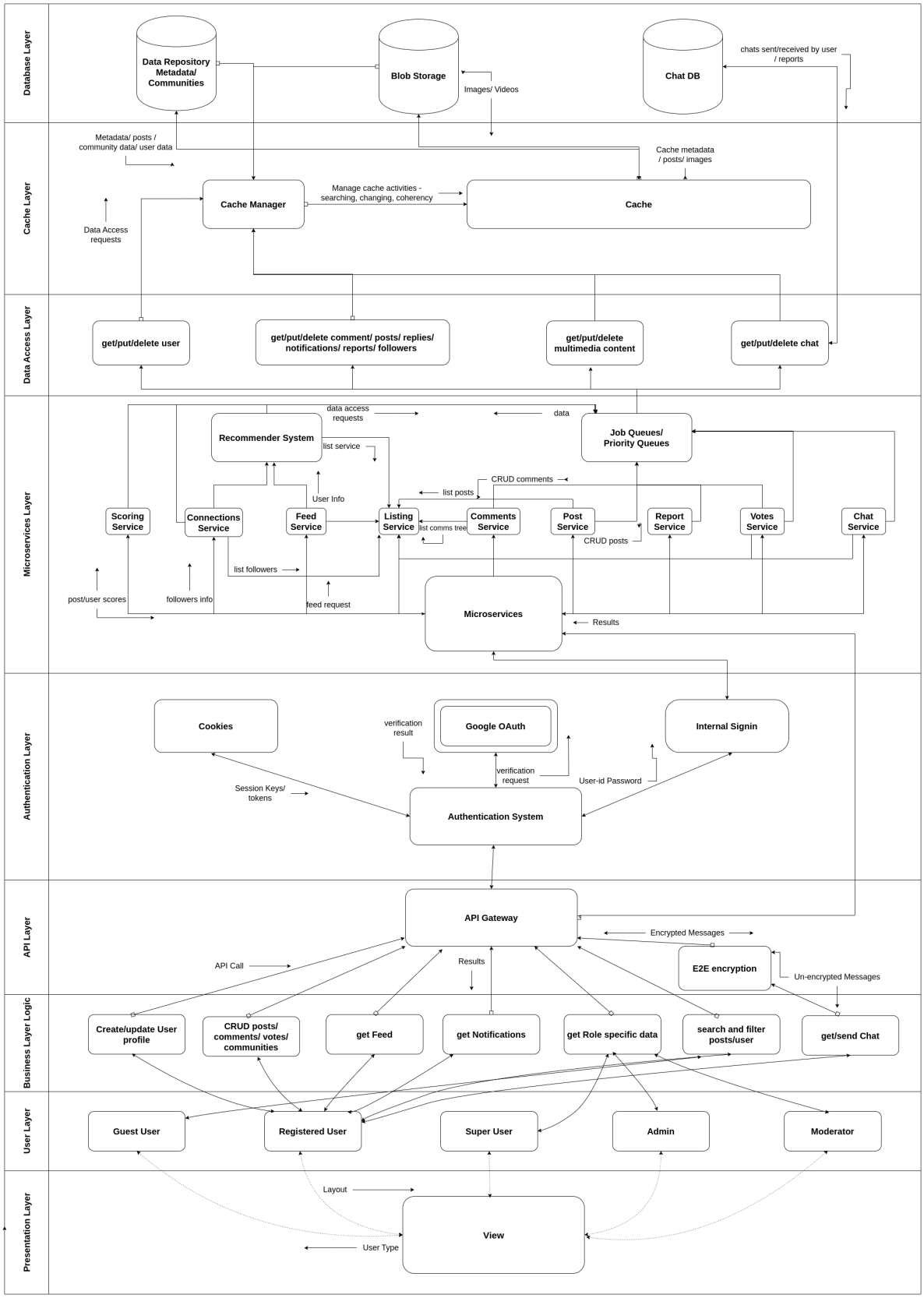


Figure 3: Architecture 2: Layered Architecture

# 3 ATAM : Architecture Tradeoff Analysis Method

**Here are the reasons why a layered architecture would be more suitable than a repository architecture for a this application:**

- **Clear Separation of Concerns:**
  This application involves multiple functionalities such as user interactions, content presentation, and data storage. Layered architecture provides a clear separation of these concerns into distinct layers, making the system easier to understand and maintain.

- **Scalability of User Interactions:**
  In this app, the user interaction layer is likely to experience varying levels of demand, such as spikes in posting, commenting, or voting activities. Layered architecture allows for the independent scaling of individual layers. Thus, if there's a surge in user interactions, the corresponding layer can be scaled horizontally without impacting other layers.

- **Flexibility for Future Changes:**
  This platform may evolve over time, introducing new features or altering existing ones. Layered architecture facilitates this evolution by allowing changes to one layer without affecting others. For example, if there's a need to revamp the user interface or introduce new functionalities, modifications can be made to the respective layer without disrupting the entire system. Also the microservices layer in the architecture can be used to add new functionalities without affecting the existing ones.

- **Simplicity in Managing User Interactions:**
  This applications primarily revolve around managing user interactions, such as posting, commenting, and voting. Layered architecture simplifies the management of these interactions by providing dedicated layers for handling user input, processing business logic, and presenting data. This clarity in design aids in efficient development and maintenance of the application.

- **Performance:**
  The cache layer at the top of the data layer can be used to store frequently accessed data, which can improve the performance of the system. This can be used to store user-specific feeds and trending feeds.

- **Security:**
  The E2E encryption component can be placed in the messaging layer to ensure that the messages are secure and cannot be read by the server.

- **Scalability:**
  The layered architecture can be easily scaled horizontally by adding more instances of the same layer. This can be useful in case of a surge in user interactions or data storage requirements.

- **Latency**:
  The layered architecture can be a little slower than the repository architecture as there are multiple layers involved in processing the request. However, the layered architecture can be optimized to reduce latency by using caching and load balancing.

- **Debugging**:
  A layered architecture can be easier to debug as the concerns are separated into distinct layers. This can help in identifying the source of issues and fixing them quickly.

Hence a layered architecture is preferred for this application due to its ability to provide clear separation of concerns, scalability for user interactions, flexibility for future changes, adherence to common web application patterns, and simplicity in managing user interactions.

Below is a table comparing the two architectures based on the above points:

Table 3: ATAM Analysis

| **Criteria** | **Repository Model** | **Layered Model** |
|---|---|---|
| Data Security | Less secure due to direct access from all modules to data | More secure as the data access only happens through data access layer |
| Performance | Less | More due to caching |
| Error Handling | Will have to do lot of work to handle errors for different modules | Layer wise error handling will be more efficient |
| Adding Features | Difficult | Easy as we can add an additional microservice |
| Code Generation Process | Difficult as have to handle different modules completely as seperate | Easier to work layer wise |
| Simultaneous user activity | Difficult to handle and Discrepancy may arrive | Can be done by using buffering and cache service |
| Database access | Simultaneous Database access and modifications | Less database access due to cache service |