Subscription Based Sports Website

Requirement Clarification:

1. **User Registration and Account Creation:** The website should provide users with the ability to subscribe and create their accounts, allowing them to personalize their experience and access subscription-based features.

2. **Live Scores and Game Status:** Users should be able to view real-time live scores and stay updated on the status of ongoing games across different sports.

3. **Historical Data and Statistics:** The website should offer a comprehensive collection of historical data and statistics, allowing users to explore and analyze past games and performances.

4. **Scalability and Performance:** The system should be designed to handle a significant number of users and accommodate high levels of concurrent requests, ensuring a smooth and responsive user experience, even during peak periods.

Capacity and Constraints:

1. The system should have robust scalability to handle future growth in user demand and data storage requirements without compromising performance.

2. The system should have robust scalability to handle future growth in user demand and data storage requirements without compromising performance.

3. The system should efficiently process and handle data from various sources, such as live feeds, databases, and external APIs, to ensure accurate and up-to-date information for users.

4. The system should implement efficient caching mechanisms to reduce the load on backend servers and improve response times, especially for frequently accessed data.

5. The system should have adequate bandwidth and network infrastructure to support the transfer of large amounts of data, particularly during peak usage periods.

6. The system should adhere to industry-standard security protocols and measures to protect user data, including encryption, access controls, and secure authentication mechanisms.

7. The system should be designed with fault tolerance and high availability in mind, ensuring that even in the event of hardware failures or network disruptions, the service remains accessible and data integrity is maintained.

8. The system should incorporate effective monitoring and analytics tools to track system performance, identify bottlenecks, and make data-driven optimizations for better scalability and user experience.

System Api

1. **User Management API:** This API is responsible for managing user-related functionalities such as user registration, login, authentication, and profile management.

2. **Sports API:** This API provides access to live scores, game status, schedules, statistics, and historical data for a wide range of sports and events. It enables retrieval of information related to teams, players, matches, and tournaments.

3. **Subscription Management API:** This API handles user subscriptions, allowing users to subscribe to specific sports, teams, or events. It manages subscription plans, payment processing, renewal, and cancellation of subscriptions.

4. **Notification API:** This API facilitates real-time updates and notifications to subscribed users. It sends push notifications, emails, or SMS alerts for various events, such as live score updates, game highlights, match results, and personalized notifications based on user preferences.

5. **Search API:** This API provides search functionality, allowing users to search for specific sports, teams, players, or matches. It offers filters, sorting options, and autocomplete suggestions to enhance the search experience.

6. **Security API:** This API handles security-related functionalities, such as encryption, access control, rate limiting, and protection against common web vulnerabilities like cross-site scripting (XSS) or SQL injection attacks.

7. **Versioning and Deprecation API:** This API manages different versions of APIs to ensure backward compatibility and smooth transitions when introducing new features or deprecating older APIs.

It allows for proper version control and communication with API consumers.

Database Design:

Table: Users

user_id (primary key)
username
password
email
created_at
last_login

Table: Sports

sport_id (primary key)
sport_name

Table: LiveScores

score_id (primary key)
sport_id (foreign key referencing Sports.sport_id)
match_id
home_team
away_team
score
match_status
start_time

Table: HistoricalData

data_id (primary key)

sport_id (foreign key referencing Sports.sport_id)
match_id
home_team
away_team
score
match_date

Table: Subscriptions

subscription_id (primary key)
user_id (foreign key referencing Users.user_id)
sport_id (foreign key referencing Sports.sport_id)
subscription_date
Expiration_date

Table: Security

security_id (primary key)
user_id (foreign key referencing Users.user_id)
security_level
last_activity
login_attempts

Table: APIVersions

version_id (primary key)
api_name
version_number
Release_date

Algorithms Used:

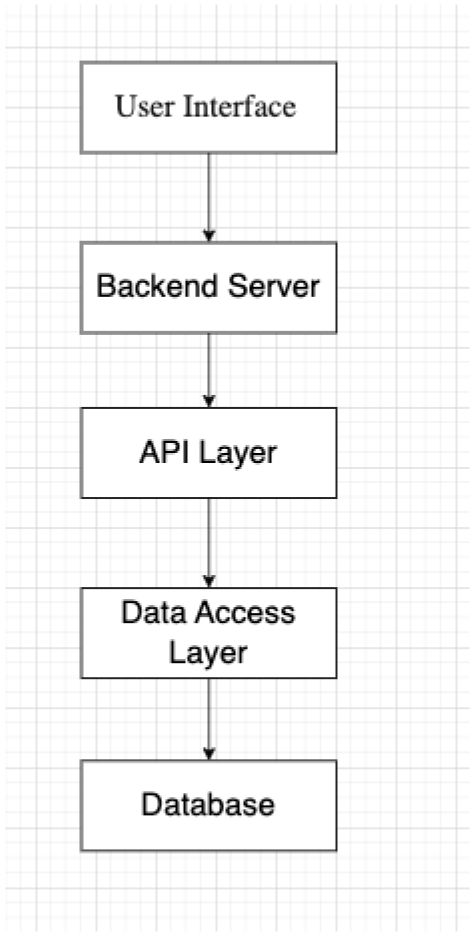
1. Linear Search: Iteratively searches through a list to find a specific value.
2. Binary Search: Efficiently searches a sorted list by repeatedly dividing it in half.
3. Bubble Sort: Iteratively compares adjacent elements and swaps them if they are in the wrong order, gradually moving the larger elements towards the end of the list.
4. Insertion Sort: Builds the final sorted list one element at a time by inserting each new element into its correct position.
5. Selection Sort: Repeatedly selects the minimum element from the unsorted portion of the list and swaps it with the first element.

Data Structures:

1. **Array**: A fixed-size collection of elements stored in contiguous memory locations.
2. **Linked List:** A dynamic data structure where each element (node) holds a reference to the next node in the list.
3. **Stack**: A Last-In-First-Out (LIFO) data structure where elements can be added or removed only from the top.
4. **Queue**: A First-In-First-Out (FIFO) data structure where elements can be added to the rear and removed from the front.
5. **Hash Table**: Stores key-value pairs, providing efficient lookup, insertion, and deletion operations based on the hash function.
6. **Binary Tree**: A hierarchical data structure where each node has at most two children.
7. **Heap**: A complete binary tree where each parent node is either greater than or equal to (in a max heap) or less than or equal to (in a min heap) its children.
8. **Graph**: A collection of nodes (vertices) connected by edges, representing relationships between them.
9. **Trie**: A tree-like data structure used for efficient retrieval of strings and prefix matching.

10**. Linked Hash Map**: Combines the features of a linked list and a hash map, providing both fast lookup and predictable iteration Order.

Design Layer:-



The User Interface will interact with the Backend Server, which will handle user requests and responses. The API Layer will provide the necessary APIs for user management, sports data, subscription management, and notifications. The Data Access Layer will access the database to retrieve and store data. The Database will store user information, sports data, game information, and subscription details.

Twitter Design

Requirement Clarification:

1. Users should have the ability to create and publish tweets, consisting of text, images, and/or videos.

2. The application should provide different timelines for users to view content: Home timeline, displaying tweets from followed users; User timeline, showcasing tweets of a specific user; and Search timeline, allowing users to search for specific topics or hashtags.

3. The system should provide a feature to display trending hashtags and popular topics, enabling users to discover and engage with popular conversations.

4. Users should be able to follow/unfollow other users to customize their Home timeline with relevant content.

5. The application should support user interactions, such as liking, retweeting, and replying to tweets, fostering engagement and conversation.

6. Notifications should be sent to users when their tweets are liked, retweeted, or replied to, ensuring they stay informed about interactions related to their content.

7. The system should allow users to mention and tag other users in their tweets, promoting connections and enabling direct engagement.

8. The application should have a profile page for each user, displaying their tweets, followers, following, and other relevant information.

System API:

1. **User Management API:** This API will facilitate user-related functionalities such as user registration, login, authentication, and account management. It will handle user creation, validation of user credentials, and authorization checks for protected resources.

2. **Tweet API:** This API will handle tweet-related operations, including creating new tweets, retrieving tweets based on various criteria (such as user, hashtag, or timeline), updating or deleting existing tweets, and managing tweet interactions (likes, retweets, replies).

3. **Timeline API:** This API will generate and retrieve different timelines for users. It will include functionalities like generating the home timeline, which shows tweets from followed users, and the user timeline, which displays tweets of a specific user. The API will also handle pagination and filtering options to provide a smooth scrolling experience.

4. **Search API:** This API will enable users to search for tweets based on keywords, hashtags, user mentions, or other search criteria. It will perform efficient search operations on tweet data, returning relevant results to users.

5. **Reporting API:** This API will enable users to report inappropriate or abusive content. It will handle user-generated reports, review reported content, and enforce content moderation policies to maintain a safe and positive user environment.

Database Design

Entities:

Users
Tweets
Hashtags
User_Followers (to represent the following/follower relationship between users)
Likes
Retweets
Replies
Notifications

Database Tables:

Users:

user_id (Primary Key)
username
email
password (hashed/salted)
created_at
updated_at

Tweets:

tweet_id (Primary Key)
user_id (Foreign Key referencing Users table)
content
created_at
updated_at

Hashtags:

hashtag_id (Primary Key)
hashtag_text
User_Followers:

follower_id (Foreign Key referencing Users table)
following_id (Foreign Key referencing Users table)

Likes:

like_id (Primary Key)
user_id (Foreign Key referencing Users table)
tweet_id (Foreign Key referencing Tweets table)
created_at

Retweets:

retweet_id (Primary Key)
user_id (Foreign Key referencing Users table)
tweet_id (Foreign Key referencing Tweets table)
created_at

Replies:

reply_id (Primary Key)
user_id (Foreign Key referencing Users table)
tweet_id (Foreign Key referencing Tweets table)
content
created_at

Notifications:

notification_id (Primary Key)
user_id (Foreign Key referencing Users table)
notification_type (e.g., like, retweet, reply)
related_tweet_id (Foreign Key referencing Tweets table)
created_at

Media:

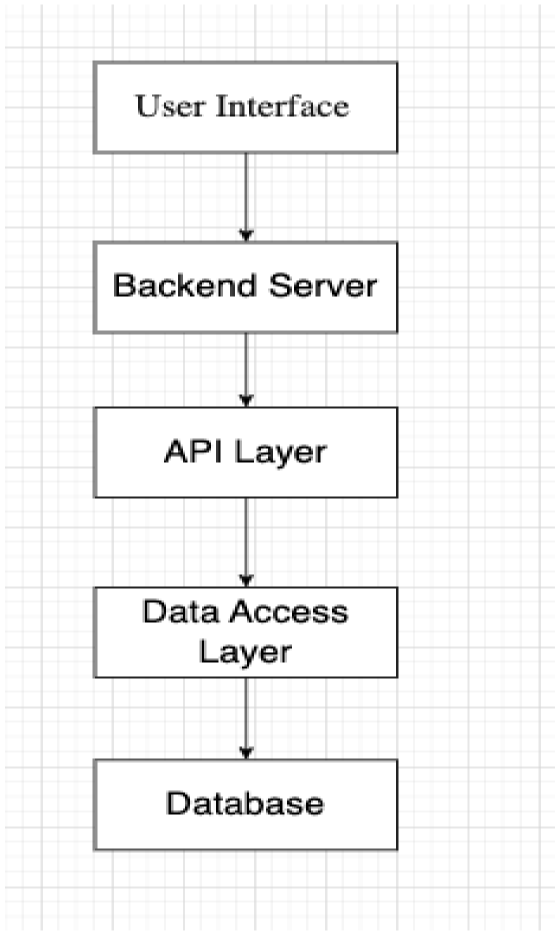
media_id (Primary Key)
tweet_id (Foreign Key referencing Tweets table)
media_url
media_type

Trending:

trending_id (Primary Key)
hashtag_id (Foreign Key referencing Hashtags table)
trending_count
Created_at

Basic Algorithm or Data Structure:

1. To generate timelines, the system can use a combination of user's tweets, tweets from followed users, and relevant tweets based on user interests.
2. To retrieve trending hashtags and topics, the system can use algorithms like counting occurrences of hashtags and tracking tweet popularity.



The User Interface will interact with the Backend Server, which will handle user requests and responses. The API Layer will provide the necessary APIs for user management, tweet creation and retrieval, timeline generation, search functionality, and trending topics. The Data Access Layer will access the database to retrieve and store data. The Database will store user information, tweet information, follower relationships, and hashtag data.