Subscription based sports website

Requirement Clarification:

1. The website should allow users to subscribe and create accounts.
2. Users should be able to view live scores and game status for various sports.
3. The website should provide historical data and statistics for previous games.
4. The system should be scalable to handle a large number of users and simultaneous requests.

Capacity and Constraints:

1. The system should be able to handle a large number of concurrent users and provide real-time updates for live scores.
2. The system should store and retrieve historical data for a large number of games and sports.
3. The website should be responsive and provide a seamless user experience even during peak traffic periods.

System API:

1. User Management API: This API will handle user registration, login, and authentication.
2. Sports API: This API will fetch live scores, game status, and historical data for various sports.
3. Subscription Management API: This API will handle user subscriptions and payment processing.
4. Notification API: This API will send real-time updates and notifications to subscribed users.

Database Design:

1. User Table: This table will store user information such as username, email, password, and subscription status.
2. Sports Table: This table will store information about different sports such as sport name, rules, and regulations.
3. Game Table: This table will store information about individual games such as game ID, date, time, teams, and scores.

4. Subscription Table: This table will store information about user subscriptions, including the subscribed sports and subscription expiry date.

Basic Algorithm or Data Structure:

1. To fetch live scores and game status, the system can use APIs provided by external sports data providers.
2. To store and retrieve historical data, a relational database management system (RDBMS) like MySQL can be used.
3. Data structures like arrays, lists, or hash maps can be used to efficiently store and retrieve game information.

High-level Block Diagram:
User Interface -> Backend Server -> API Layer -> Data Access Layer -> Database

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 be able to create and post tweets.
2. The application should support different timelines: Home timeline, User timeline, and Search timeline.
3. Users should be able to view trending hashtags and topics.

Capacity and Constraints:

1. The system should be able to handle a large number of concurrent users and store a vast amount of tweets.
2. The application should provide real-time updates for timelines and trending topics.

3. The system should prioritize security and protect user information.

System API:

1. User Management API: This API will handle user registration, login, and authentication.
2. Tweet API: This API will handle tweet creation, retrieval, and deletion.
3. Timeline API: This API will handle the generation and retrieval of different timelines.
4. Search API: This API will enable users to search for tweets based on keywords, hashtags, and user mentions.
5. Trending API: This API will provide information on trending hashtags and topics.

Database Design:

1. User Table: This table will store user information such as username, email, password, and profile details.
2. Tweet Table: This table will store tweet information, including tweet ID, user ID, content, timestamp, and any associated media.
3. Follower Table: This table will store the follower-followee relationships between users.
4. Hashtag Table: This table will store information about hashtags used in tweets, including the hashtag text and associated tweets.

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.

High-level Block Diagram:
User Interface -> Backend Server -> API Layer -> Data Access Layer -> Database
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.