1. Overview

The Data Trading System facilitates the streaming and viewing of large datasets using **Node.js**, **WebSocket**, and **Angular**. This system is designed to prevent server overload by breaking down large datasets into manageable chunks that are streamed to users based on their role. The system ensures efficient data management and role-based access control (RBAC), where **Admin** users have full access, and **Regular** users can access a limited subset of the data.

2. User Authentication & Authorization

Login Flow:

- **Authentication:**
  - Both **Admin** and **Regular User** authenticate using an API. Upon successful login, the system returns a **JWT (JSON Web Token)** that contains the user's role and other necessary credentials.
  - JWT is securely stored in the browser's **localStorage**. This allows the system to persist user sessions across page refreshes, so users don't need to re-authenticate each time they access the application.
- **Role-based Access Control (RBAC):**
  - The user role (Admin or Regular User) is embedded in the **JWT token**, and the token is passed with each request to verify access.
  - **Admin** users have access to the entire dataset, while **Regular** users have limited access to only a subset of the data, ensuring appropriate segregation of duties and data privacy.

Logout Flow:

- When a user logs out, the JWT stored in **localStorage** is removed, effectively ending the session and ensuring that the user cannot access any further resources until they log in again.

3. Data Loading and Streaming

Load Component (Data Fetching Trigger):

- Users can initiate data loading by clicking on the **"Load Component"** button. Once clicked, the system establishes a **WebSocket** connection to the server, initiating the data streaming process.

WebSocket Communication:

- **WebSocket Connection:**
  - A **persistent WebSocket connection** is established between the client (Angular) and the server (Node.js). Once established, the WebSocket connection remains

    open for the entire session unless the user logs out or navigates away from the page.
  - WebSocket ensures real-time, two-way communication between the client and the server, allowing data to be streamed efficiently without repeatedly opening and closing connections.
- **Data Chunking:**
  - The system **streams data in chunks** to avoid overloading the client or server. Each chunk contains **100 items** per second.
  - The chunking process helps manage large datasets by breaking them down into smaller, more manageable pieces, which are sent progressively to the client.

Role-Based Data Access:

- **Admin Role:**
  - Admin users are permitted to access the **entire dataset**. They will receive all chunks of data as they are streamed.
- **Regular User Role:**
  - Regular users are restricted to receiving only the **first 10 chunks** of the dataset. This is typically equivalent to 1,000 items, with each chunk containing 100 items. Regular users can view the first few pieces of the dataset but do not have access to the entire dataset.

4. Component Change & WebSocket Management

Handling Component Navigation:

- When the user navigates to different components or pages within the application, it is essential to **stop** unnecessary data loading to prevent excessive resource consumption.
- The **WebSocket communication** is **closed** when the user switches components or navigates away from the data streaming component, ensuring that data is only loaded when the user is actively viewing the relevant data.
  - This also prevents the server from continuing to stream data unnecessarily, reducing bandwidth usage and preventing overload.

WebSocket Connection Maintenance:

- Once the WebSocket connection is established, it **remains open** and active as long as the user remains on the page that requires data streaming. The connection is automatically closed if the user navigates away or closes the browser tab.
- This persistent connection reduces the need to establish a new WebSocket connection every time a chunk of data is required, thus improving performance and reducing latency.

5. MongoDB Integration

User Management with MongoDB:

- MongoDB is used to manage user login and signup processes, storing users' credentials, roles (Admin or Regular User), and session data.
- When a user logs in, the server queries MongoDB to authenticate the user and retrieve the associated role. The role is then embedded within the JWT token sent to the client.

Role-based Data Filtering with MongoDB Aggregation:

- The dataset in MongoDB is stored in a collection. Data is filtered and aggregated based on the user's role before being streamed to the client.
- For Admin users, the full dataset is returned from the database.
- For Regular users, an aggregation pipeline is applied to filter and limit the data, ensuring that only a subset of the dataset (e.g., first 10 chunks) is returned.
- MongoDB Aggregation Pipeline:
- The server uses MongoDB aggregation queries to manage data filtering and pagination for regular users. This allows for optimized queries, ensuring only relevant data is sent to the client and minimizing unnecessary processing.

6. Logger Integration

System Monitoring & Error Tracking:

- A logging system is integrated across the application to monitor and debug the system's activity.
- The logger tracks key events, such as WebSocket connections, data requests, successful or failed logins, data streaming progress, and errors.
- Logging helps to:
- Monitor the system's health and identify issues in real time.
- Track system activities such as user interactions, data loading, and component changes.
- Record error messages for easier debugging and faster resolution of issues.
- Log Types:
- Info Logs: Track routine operations like user logins, successful data fetching, and WebSocket connections.
- Error Logs: Capture errors that occur, such as failed data fetches, WebSocket disconnections, and user authentication failures.
- Log Formats:
- Logs are structured in JSON or a simple text format, allowing for easy integration with monitoring tools and external log management services.

7. Security and Data Integrity

Token-Based Authentication (JWT):

- Authentication is handled via JWT tokens, which are passed with every API and WebSocket request. This ensures that each request is securely validated and authorized, preventing unauthorized access to sensitive data.
- The JWT includes the user's role and other necessary data, ensuring that the server can validate the user's identity and role without querying the database for every request.

Data Privacy & Integrity:

- Role-Based Access Control ensures that each user only sees the data they are authorized to view. Admins can view all data, while Regular users are limited to the first 10 chunks of data.
- The aggregation pipeline in MongoDB ensures that the data sent to users is properly filtered before reaching the client, reducing the risk of data leaks.

Secure WebSocket Communication:

- WebSocket communication is secured using TLS (Transport Layer Security), ensuring that data transmitted between the client and server is encrypted and protected from interception.

8. Performance Optimization

Data Chunking:

- The dataset is divided into chunks of 100 items each, which allows for better resource management. Streaming data in smaller chunks helps to reduce memory and bandwidth usage, ensuring that the client doesn't become overwhelmed by large datasets.

Efficient Data Fetching with MongoDB Aggregation:

- MongoDB's aggregation pipeline is used to filter data before it is sent to the client. This ensures that only the necessary data is processed and streamed, reducing server load and improving overall system performance.

Persistent WebSocket Connection:

- WebSocket is kept open throughout the session, which eliminates the overhead of repeatedly establishing new connections for each data request. This improves real-time data delivery speed and reduces latency.