

**Detailed Workflow:**

1. **User Login**:
   * The user logs into the platform through the front-end (web/mobile app).
   * The front-end sends an authentication request to the **User Management Service**, which verifies credentials and returns an authentication token (JWT).
2. **Portfolio Data Retrieval**:
   * The front-end fetches the user’s portfolio data by calling the **Asset Management Service** through RESTful APIs.
   * This service retrieves asset details from the database (SQL/NoSQL).
3. **Price Updates**:
   * The **Pricing Service** fetches real-time asset prices every 10 minutes from external data providers (stock prices, mutual funds).
   * The service pushes updated prices to the **Asset Management Service**.
4. **Portfolio Recalculation**:
   * The **Portfolio Calculation Service** receives updated prices and calculates the current value of each user’s portfolio.
   * This service uses the latest asset prices and user data to update the portfolio value.
5. **Real-Time Updates**:
   * The updated portfolio value is pushed to a message queue (Kafka/RabbitMQ).
   * The **Real-Time Updates** service consumes the queue and sends updated portfolio data to the user’s front-end through WebSockets.
6. **Database Management**:
   * User data, asset data, and portfolio values are stored in the database.
   * The database must support horizontal scaling and efficient retrieval for fast updates and queries.

**Reliability and Scalability:**

* **Horizontal Scaling**: Each service (pricing service, portfolio calculation service) can scale horizontally to handle more users and data.
* **Data Consistency**: Ensuring atomic updates (e.g., using transactions or ensuring consistency in the message queue) is key for reliable portfolio calculations.
* **Message Queue**: Kafka or RabbitMQ ensures that real-time updates are reliably delivered, even during temporary service downtime.
* **Cache**: Redis or similar caching mechanisms can be used to cache frequently accessed data (like user portfolios) for faster retrieval.