Task 1

- First we aggregate the records and calculate total invested amount of each investor (*totalAmountInvested*) and a unique set of syndicates for each investor (*investorsSyndicates*)
- Count the number of syndicates for each investor and store in a Map (*syndicateCountOfInvestor*) where key is count and value is investor id
- Also store all the counts in a list (*syndicateCounts*)
- Sort the list
- Now from the sorted list take the first 5 items to get top 5 investors from *syndicateCountOfInvestor* and their total investment

Task 2

The transaction amount threshold alert was straight forward comparison of the incoming transaction amount and threshold amount .

For alert regarding sudden spike in transaction number was implemented by sliding window. Sliding window is used for time sensitive rate limiter . Sliding window was implemented with data structure called sorted set or ZSET which is provided by Redis . Sorted set / ZSET allows us to store timestamped data and retrieve data within a specific time range .

When processing a transaction for spike alert

1. The syndicate is stored against timestamp in the ZSET
2. Data with timestamps older than Threshold time (1hr / 3600s) is removed
3. Existing number of data are counted on the ZSET
4. Threshold rate is calculated with the count
5. Count and threshold rate are compared to tigger alert

Scalability

We need to put monitoring system in place to decide when to scale

- initially we can go for vertical scaling
- We can then go for horizontal scaling and distribute the workload among multiple instances
- Platform like Apache Kafka can be used to ingest the transactions and distribute the workload among the instances

Data integrity

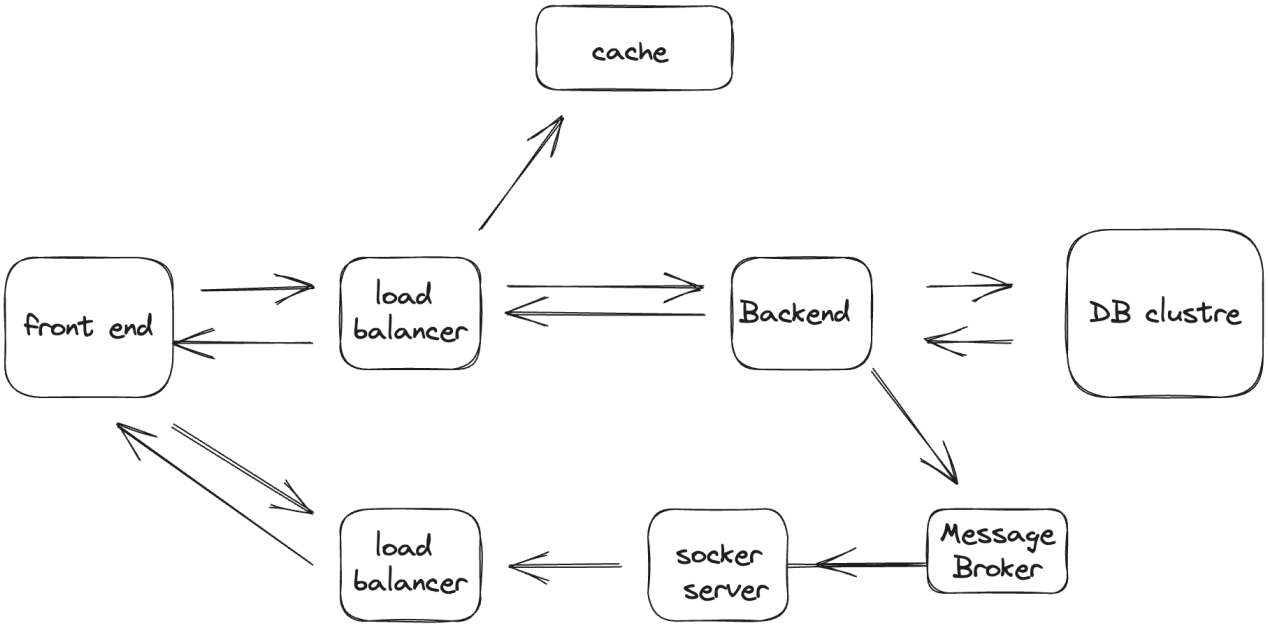
- Data integrity can be ensured by using a ACID compliant database like mysql
- Add proper validation for data

Fault tolerance

- Monitor application for errors
- Using redundant storage system in case of failures
- Use cloud environments fault tolerant features

Task 3

High level Architecture



components

Frontend : the users will interact will the frontend (FE)

Backend : api call from the FE will be passed to the backend (BE)

Socket server : to send live update to the frontend

Message broker : for internal communication among services in this case BE will send a message with necessary payload to message-broker when there is any real time data update , socket server will receive the message and send an update to the FE to notify the use or update the UI

Load balancer : There could be multiple instance of backend or the socket server , FE will communicate not communicate with them directly , instead it will connect with a load balancer (LB) , the LB will route traffic to the services accordingly

DB cluster : A database cluster (preferably managed) for persisting data

Cache : A caching layer for storing frequently accessed data

Tools and Technologies

1. Modern web framework : for backend nestjs as it is nodejs based framework which is great for high concurrent systems and provides structured way for managing project . And nextjs / reactjs or any other modern frontend framework for frontend

2. Github : for version controlling as team collaboration , as it gives built in features for CI/CD

3. JIRA : for project management

4. Slack : for internal communication for team

5. Redis : for caching , it is a widely used tool and has many built in useful data structures

6. Database : An ACID compliant database for storing data , could use managed database for example Amazon RDS , which will eliminated the hassles like scaling the DB manually

7. Kafka / Rabbitmq : as a message broker for communicating among different services , like backend and websocket

8. Web socket / SSE : for sending realtime updates to the frontend

9. Monitoring tools : such as ELK stack or Datadog to monitor the application performance and track errors

10. Cloud platform : AWS or GCP for hosting the infrastructure . Like deploying the backend services on EKS or ECS (GKE or Cloud run in GCP)

11. Docker : for packaging and deploying application

Potential Bottlenecks

Database : try to optimize queries and properly scale database

Websocket : use scaleable websocket system

Concurrency : to ensure concurrent updates and consistency implment database locking , optimistic concurrency control , distributed locks

Implementation and Deployment

1. Define requirements : clearly define requirements for the feature , system and resources
2. Selecting tools : select tools and technologies best fit for the requirements
3. Project management and collaboration tools : break down the task into sprints and add to Jira or other project management software , tools like slack for internal communication . Set up policies for work work , daily standups etc
4. Security : Ensure security best practices for sources code , sensitive information involved in development and access to resources
5. Development : start the development of the feature , have regular code reviews
6. Testing : add proper testing like unit , integration , e2e
7. Documentation : have proper documentation for code and architecture
8. Deployment : Deploy on cloud based infrastructure . Publish feature on a staging environment before making it live
9. Monitoring and alert : monitor application matrices and setup alert for critical cases , and find bottleneck and optimize it
10. Disaster recovery : set it up to ensure data availability in case of disaster