Real-Time Feed Data Processing and Analysis System

**Overview**: The project involves developing a backend system interacting with an external feed API to fetch real-time feed or asset data. The system will process, analyze, and store this data and provide insights through APIs.

### **Objectives**

1. **API Integration**: Successfully integrate with the external feed API to fetch real-time feed data.
2. **Data Processing**: Implement logic to process and normalize the incoming data.
3. **Database Management**: Store processed data in an AWS MySQL database, ensuring data integrity and efficiency.
4. **Real-Time Data Handling**: Develop capabilities to handle and update data in real-time.
5. **API Development**: Create APIs to retrieve and display processed data and insights.
6. **Deployment**: Deploy the system on AWS for scalability and robustness.
7. **Demo**: Demo the project

### 

### **Technologies**

* Java 21 and Spring Boot 3.x for backend development.
* IntelliJ IDEA as the development environment.
* AWS RDS (MySQL) for database services.
* Deployment using ECS
* Explore SQS or another messaging system for real-time data handling.
* Localstack
* Integration Testing and Unit testing using Junit, Mockito.

### 

### **Scope of Project**

* Integration with the external feed API for real-time feed/assets data.
* Developing backend logic to process and store this data.
* Creating RESTful APIs to provide access to the processed data.
* Implementing real-time data update mechanisms.
* Ensuring scalability and security in the cloud environment.

### **Tasks and Deliverables**

* **API Integration**: Code to connect and fetch data from the external feed.
* **Security**: The application needs to be secured using Spring security, etc.
* **Data Processing Module**: System to process and normalize live data.
* **Database Implementation**: Design and create the database schema and integrate it with the backend.
* **API Development**: APIs for data retrieval and analysis.
* **Real-Time Updates**: Implementing a system for handling real-time data updates. Also, feed data will be shown on the client(mobile/web)
* **Documentation**: one-page README.md of the system architecture, code, and deployment strategy.

### 

### **Understanding and Integrating the External Feed API**

* **Objectives**:
  + Familiarise yourself with spring boot and resources which will help for this project development
  + Familiarise yourself with the external feed API's documentation.
  + Set up the development environment using Java, Spring Boot, and IntelliJ IDEA.
  + Create a new module in Hugoserve. Establish a basic connection to the external feed API and perform test calls to understand the response structure and response format.
* **Tasks**:
  + Go through API documentation, focusing on endpoints that provide real-time feed data.
  + Install and configure IntelliJ IDEA, Java, and Spring Boot.
  + Write initial code to connect to the API and fetch data.
  + Analyze the structure of the fetched data (JSON/XML/etc.) and identify key data points (like asset name, price, timestamp).
  + Create a new module to implement the above
  + Get AWS access
* **Deliverables**:
  + A basic application setup capable of fetching data from the external API.

### 

### **Data Processing and Normalisation**

* **Objectives**:
  + Develop logic to process and normalize the fetched data for consistent storage and retrieval.
  + Handle various data inconsistencies and possible errors in API responses.
* **Tasks**:
  + Design data models to represent the feed data in the application.
  + Implement parsing logic to convert API data to these models.
  + Develop error handling for API response anomalies.
  + Test the processing logic with various API responses.
  + Consider using protocol buffers (instead of POJOs) for the serialization and deserialization of JSON, database objects.
* **Deliverables**:
  + A functional data processing module that normalizes and structures the data.

### 

### **Database Design and Integration**

* **Objectives**:
  + Design a MySQL database schema suitable for the structured asset/feed data.
  + Implement database integration within the application.
  + Ensure efficient and secure data storage.
* **Tasks**:
  + ~~Setup localstack on your laptop~~
  + Define database tables and relationships that reflect the data models.
  + Set up AWS RDS with a MySQL.
  + Develop code for database connectivity and CRUD operations.
  + Write unit tests to ensure data integrity and proper database interactions.
* **Deliverables**:
  + A robust and well-structured MySQL database integrated with the application.

### 

### **API Development and Real-Time Data Handling**

* **Objectives**:
  + Create APIs within the application to serve processed feed/asset data.
  + Implement mechanisms to handle real-time data updates.
* **Tasks**:
  + Design RESTful endpoints for data retrieval
    - 1. Real-time API to fetch feed
    - 2. Historical price feed with pagination
  + Write logic to update the database with new data from the feed API at regular intervals.
  + Basic authentication for APIs
  + Implement caching strategies for performance optimization.
  + Test APIs for functionality and reliability.
* **Deliverables**:
  + Functional and documented APIs that provide access to up-to-date feed/asset data.

### 

### **Testing, Deployment, and Demo**

* **Objectives**:
  + Thoroughly test the entire application for bugs and performance issues.
  + Deploy the application on AWS.
  + Create documentation for the system.
* **Tasks**:
  + Conducted integration unit testing ( including code coverage) and stress testing to ensure system robustness.
  + Prepare the AWS environment for deployment (Elastic Beanstalk or EC2 setup).
  + Deploy the application and monitor its performance.
  + Write documentation for the system with a list of features that will be part of the demo.
* **Demo**:
* Live feed rates
* Historical Price feed with pagination
* Basic Authentication/Authorization
* Analysis of the feed rates → percentage increase/decrease in the given time period, etc.
* Buy / Sell an asset + Show end customer balance + hold / withdraw
* Optional → Returns calculator based on previous analysis of data (for a particular duration)
* Make some changes and deployment
* Anything else you want to demo
* Please decide party venue + bill after demo - Very Important

### APIs

Real-time price feed for any metal -

<https://goldbroker.com/api/spot-price?metal=XAG&currency=PKR&weight_unit=g>

Historical data API -

<https://goldbroker.com/api/spot-prices?metal=XAG&currency=PKR&weight_unit=g>

Bonus Full history

<https://goldbroker.com/api/historical-spot-prices?metal=XAG&currency=PKR&weight_unit=g>

### Learning Resources

**Spring boot + Intellij**:

IntelliJ IDEA is a versatile IDE for Java and other languages, known for its code completion and refactoring tools. Please follow the below links to set up IntelliJ with a sample spring boot project.

<https://www.youtube.com/watch?v=35EQXmHKZYs>

<https://www.youtube.com/watch?v=Fkyuqq0gmCo>

**Infrastructure**

Follow AWS and Kubernetes to understand our infra and deployment setup.

AWS:

<https://www.youtube.com/watch?v=k1RI5locZE4>

Kubernetes

[https://www.youtubxe.com/watch?v=d6WC5n9G\_sM](https://www.youtube.com/watch?v=d6WC5n9G_sM)

Git:

Git is a version control system used in software development to track and manage code changes. It supports collaboration, branching, and merging. Follow the below links for practice and practical examples

<https://www.youtube.com/watch?v=RGOj5yH7evk>

<https://www.youtube.com/watch?v=tRZGeaHPoaw>

Builder design pattern

### **Coding guidelines**

<https://docs.google.com/document/d/1i57m2D3_BIyTBoTnHJLeLegigSCy6yD9DWDwQiiHcRk/edit#heading=h.rk11072dddjr>

**Task - 1**

1. Go through the backend code of the company.
2. Setup localstack for running the code locally.
3. Get familiar with the coding style.

**Task - 2**

Paginate the Quest fetching API.

**Task - 3**

Create a new DTO for giving responses to frontend of SgLocate.

**Task - 4**

Today the Daily Task Scheduler does not consider expiration details while updating valid until ts. But from now, if a quest has expiration details, it should first update valid until till the expiration date and after that it should expire the quest

**Task - 5**

Migration from SQL i.e. AWS RDS to NoSQL i.e. AWS DynamoDB.

Documentation Link :

https://hugoserve.atlassian.net/wiki/spaces/HugoAPP/pages/74612794/Migration+from+SQL+to+NoSQL+for+Reports