Project Name: Credit Score Analysis Tool

Client Name: Swedbank (In Sweden)

Disclaimer: **This document uses fictional names for the project and client for illustrative and educational use only. Any resemblance to real entities is coincidental.**

1) Introduction

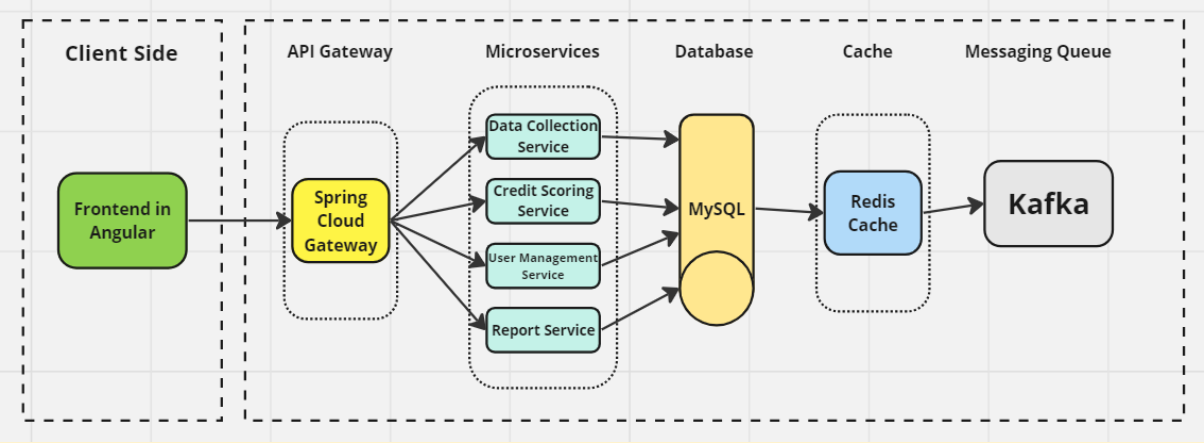
Credit Score Analysis Tool is one of the projects of Swedbank situated in Sweden.

The goal of this tool is to automate the process of credit score evaluation by integrating data from various sources including credit bureaus and banking transaction records.

This tool provided real-time credit scoring, predictive analytics based on customer behaviour, and detailed reporting features for bank staff.

It was designed using a microservices architecture, ensuring scalability and efficient management of different service components.

2) Project Architecture



i) Client Side:

We are using angular 10 in the frontend part of this project and we have a separate frontend developers' team

ii) API Gateway:

It Acts as the single-entry point for all client requests to the backend services. It routes requests to the appropriate microservices and provides common functionalities like authentication and logging.

Spring Cloud Gateway is being used for this purpose, integrating easily with other Spring Boot applications.

Spring Cloud Gateway is a Java-based API gateway for routing and filtering traffic to microservices. It makes it easier and safer for these services to communicate with each other.

iii) Microservices:

The application is split into microservices that can be set up on their own. Each service handles a different part of what the application does. This setup makes the application simpler to manage, easier to fix and update, and allows it to grow more easily.

We have following microservices in this project:

- **Data Collection Service:** It manages the ingestion and preprocessing of financial data from various sources.
- **Credit Scoring Service:** It handles the calculation of credit scores using a predefined algorithm. This service processes input from the Data Collection Service.
- **User Management Service:** It responsible for authentication and user account management.
- **Report Service:** It generates detailed credit reports and analytics based on the credit scores calculated.

iv) Database:

We are using MySQL database for each microservices although we can use different databases for each microservice but as of now we using the same database across all.

v) Cache Mechanism

We are using **redis cache** as an in-memory cache solution for storing frequently accessed data, such as credit score calculations, external data retrieved from APIs, or reference data.

vi) Logging tools:

Using **Log4j2,** it provides powerful logging capabilities for Java applications and integrated **Splunk**, it is a powerful platform for searching, analysing, and visualizing machine-generated data, including logs, in real-time

vii) Messaging system:

Using **Kafka** for operations that require real time updating such as calculating credit score at real time, and also it is used to handle asynchronous communications.

viii) Security:

Implement **OAuth** for secure access control. In Spring Boot, it is a way to let people sign into an app using their accounts from different places.

ix) Deployment and operations

- **Containerization:** Using Docker to containerize the microservices, making them easy to deploy and scale.
- **Orchestration:** Using Kubernetes to manage these containers, handling deployment, scaling, and operation of application containers across clusters of hosts.
- **CI/CD:** Continuous Integration and Continuous Deployment are achieved through Jenkins, automating the build, test, and deployment processes.
- **Git Collaboration Tool**: Using GitLab

3) Project Flow

Step 1: User Interaction and Authentication

- **Action**: A bank officer logs into the system through the User Interface. User can not access this application, only bank officer serve the user and share their details.
- **Process**:
    - The **User Management Service** handles the login request. It authenticates the user credentials against the database.
    - Upon successful authentication, it generates an access token (using OAuth) which the user will use for subsequent requests, ensuring secure sessions.
    - For any request hit (In the form of API) by user, the same access token will be sent to the server for authorization

Step 2: Data Collection and Processing

- **Action**: The bank officer initiates a credit score check for a client.
- **Process**:
    - The **Data Collection Service** gathers financial data from internal databases (like account and loan histories) and external credit bureaus (Third party system).
    - This service validates and preprocesses the data (e.g., normalizing formats, removing duplicates) to prepare it for analysis.

Step 3: Credit Scoring Calculation

- **Action**: Processed data is forwarded to the Credit Scoring Service.
- **Process**:

- The **Credit Scoring Service** receives the cleansed data and applies a scoring model. This model might include statistical algorithms and machine learning to evaluate credit risk based on historical data. (You don't have to go deep inside these algorithms, you can say a different team was working on these algorithms, we are just consuming their services by RESTfull APIs)
- The resulting credit score, along with detailed analytics, sent directly to the **Report Service.**

Step 4: Report Generation and Delivery

- **Action**: Generation of a credit report based on the calculated score by **credit scoring service.**
- **Process**:
  - The **Report Service** fetches the score and analysis from the Credit Scoring Service.
  - It compiles a detailed credit report, which may include recommendations or flags for high-risk factors.
  - The report is then made available to the bank officer via the UI and can also be sent to other stakeholders (like credit managers) via email (we use java email service to send the mail)

Common activities across all the steps:

- Throughout all steps, all data transactions and user actions are logged and can be seen on **Splunk (Logging Tool)**
- All interactions between the client (browser) and the microservices go through the **Spring Cloud Gateway (An API gateway tool),** which routes requests, handles load balancing, and provides an additional layer of security.
- For all the operations that require real time data updating, such as credit score calculation, **Kafka** is being used.

4) Use of Kafka in this Project:

It is used in **Real-Time Credit Scoring Updates** and the topic name is **"`credit-score-updates`"** in this project**:**

- **Description**: Kafka streams are used to trigger credit scoring calculations as soon as new data is available.
- **Implementation**: Whenever new financial data is collected, it is published to a specific Kafka topic. The Credit Scoring Service listens to this topic, and upon receiving data, it calculates or updates the credit scores in real-time.

5) Use Log4j2 in the Project.

Log4j2 provides powerful logging capabilities for Java applications, offering various features such as asynchronous logging, structured logging, and dynamic configuration.

- First, we excluded the default logging dependency (**Spring Boot Starter Logging**) that comes with Spring Boot, which includes **Logback**, and then we added dependencies for **Log4j2** in our POM file.
- Then, we configured the **Yml file** in the resources

6) Integration with Splunk

Splunk is a powerful platform for searching, analyzing, and visualizing machine-generated data, including logs, in real-time.

Sending Logs to Splunk:

- **Log4j2** is configured to send logs directly to Splunk using Splunk Universal Forwarder.
- Configured Log4j2 to include additional metadata in log messages, such as host information, to provide context for log analysis in Splunk.

7) Use of Redis cache:

Redis can serve as an in-memory cache solution for storing frequently accessed data, such as credit score calculations, external data retrieved from APIs, or reference data.

Where are we using Redis Cache:

- **Credit Score Calculation**: Store the results of credit score calculations in Redis cache to avoid recalculating scores for the same customer repeatedly.
- **External Data Retrieval**: Cache data retrieved from external sources, such as credit bureau data or financial transaction history, to reduce latency and API call overhead.

8) Use of CI/CD Pipeline in this project:

I. **Version Control System**: We used Git, which is a system that helps developers manage and keep track of changes in their code. It's hosted on websites like GitHub or GitLab. This system allows developers to save different versions of their work (called

"commits") and use branches to work on new features or fix bugs without affecting the main code.

II. **Continuous Integration Tools**: We chose Jenkins, an automation tool, to help manage the process of automatically checking and merging the code developers submit. Jenkins uses a special file called a Jenkinsfile to manage these tasks, which makes it easier to update and maintain the process.

III. **Build Automation**: Whenever someone submits new code (a commit), Jenkins starts a series of actions: it compiles the code, runs tests to check if the code works properly, and analyzes the code for any potential errors. We use tools like Maven or Gradle, which are specifically designed for Java applications, to handle these tasks and manage any software libraries the code depends on.

IV. **Artifact Repository**: After Jenkins successfully builds the code, it produces files (like JAR files for Java applications). These files are stored in a special storage area called an artifact repository (like Nexus or Artifactory). This repository keeps all the different versions of the software, which helps if we need to go back to a previous version.

V. **Continuous Deployment**:
    a. **Containerization**: We use Docker to package the application with everything it needs (like libraries and other dependencies) into a container. This ensures the application works the same way in different computing environments.
    b. **Orchestration**: Kubernetes is used to manage these containers across different servers. It helps with balancing the load, scaling up or down as needed, and managing any failures.
    c. **Deployment Strategies**: To update the application with minimal disruption, we use techniques like blue-green deployments or canary releases. These methods allow us to test new versions in the real environment without affecting the existing system and switch back easily if something goes wrong.

9) Data Layer / Schemas:

We are using MySQL for the microservices as of now and are planning to migrate intro MongoDB in few of services

Below are the major tables and their major fields (Its impossible to cover every table and every fields here but you can tell these major things to the interviewers if they ask and this is more than sufficient)

I. **Users Table**
    a. **user_id** (Primary Key)
    b. **username**
    c. **password_hash**
    d. **email**
    e. **role**
    f. **created_at**

g. **updated_at**
h. **last_login**
i. **status** (e.g., active, inactive)

II. **Credit Scores Table**
a. **credit_score_id** (Primary Key)
b. **user_id** (Foreign Key)
c. **score**
d. **date**
e. **score_type** (e.g., FICO, VantageScore)
f. **score_history** (JSON or array to track historical scores)
g. **algorithm_used** (details about the scoring algorithm)

III. **Financial Records Table**
a. **record_id** (Primary Key)
b. **user_id** (Foreign Key)
c. **transaction_date**
d. **amount**
e. **transaction_type** (e.g., credit, debit)
f. **transaction_description**
g. **category** (e.g., utilities, groceries, entertainment)

IV. **Reports Table**
a. **report_id** (Primary Key)
b. **user_id** (Foreign Key)
c. **credit_score_id** (Foreign Key)
d. **generated_date**
e. **report_data** (possibly stored as JSON or XML)
f. **report_type** (e.g., standard, detailed)

V. **Audit Logs Table**
a. **log_id** (Primary Key)
b. **user_action**
c. **timestamp**
d. **user_id** (Foreign Key)
e. **details** (description of the action take

10) REST APIs:

1. Data Collection Service

1. GET /data/{userId} - Retrieves all financial data for a specific user.
2. GET /data/{userId}/transactions - Retrieves transaction history for a user.
3. POST /data - Submits new financial data.
4. PUT /data/{userId} - Updates existing financial data for a user.
5. DELETE /data/{userId} - Removes financial data for a user.
6. GET /data/{userId}/loans - Fetches loan data for a user.
7. POST /data/batch - Submits financial data for multiple users.

8. GET /data/stats - Provides statistics about collected data.

2. Credit Scoring Service

1. GET /score/{userId} - Fetches the credit score for a user.
2. POST /score/calculate - Calculates the credit score based on provided data.
3. PUT /score/{userId} - Updates the credit score for a user.
4. DELETE /score/{userId} - Deletes the credit score data for a user.
5. GET /score/history/{userId} - Retrieves the score history of a user.
6. POST /score/batch - Calculates scores for multiple users.
7. GET /score/average - Provides the average score across all users.
8. PUT /score/refresh - Refreshes and recalculates scores based on latest data.

3. User Management Service

1. GET /users/{userId} - Retrieves user details.
2. POST /users - Registers a new user.
3. PUT /users/{userId} - Updates user details.
4. DELETE /users/{userId} - Deletes a user.
5. GET /users/all - Retrieves all users.
6. POST /users/authenticate - Authenticates user login.
7. PUT /users/{userId}/password - Updates a user's password.
8. GET /users/{userId}/status - Checks the status of a user's account.

4. Report Service

1. GET /reports/{userId} - Fetches all credit reports for a user.
2. POST /reports/generate - Generates a new credit report.
3. PUT /reports/{reportId} - Updates a specific report.
4. DELETE /reports/{reportId} - Deletes a specific report.
5. GET /reports/{reportId} - Retrieves a specific report.
6. POST /reports/batch - Generates reports for multiple users.
7. GET /reports/stats - Provides statistics on generated reports.
8. PUT /reports/{reportId}/send - Sends a report to a specified recipient.

11) What to add in your resume/CV?

Credit Score Analysis Tool, Swedbank (Sweden)

Description: Automated credit score evaluation platform integrating data from multiple sources to provide real-time scoring and predictive analytics. Utilized microservices architecture for scalability and efficiency.

Responsibilities:

- Worked on the implementation of real-time credit scoring using Kafka, enhancing decision-making speed.
- Collaborated with the team to develop REST APIs and implement business features for seamless functionality.
- Integrated OAuth for secure user authentication, ensuring robust system security.
- Utilized Redis cache for efficient data caching, optimizing application performance and reducing database load.
- Contributed to the design and development of the microservices architecture, enabling easy deployment and maintenance.

12) What to speak in front of your interviewer?

I recently worked on Credit Score Analysis Tool by Swedbank, basically it's a Sweden based project. This tool does the credit evaluation process by providing real-time updates and predictive analytics.

Using Spring Boot and Java, I implemented features such as instant credit score updates for clients' quick assessments. Additionally, I ensured seamless communication among system components with REST APIs. Security was important, so I established secure logins to safeguard user data. To enhance performance, I integrated Redis cache, optimizing data storage for faster processing. This project helped me in boosting my skills also learned a lot of team work.