Project Name: Credit Score Analysis Tool

Client Name: Swedbank (In swedan)

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

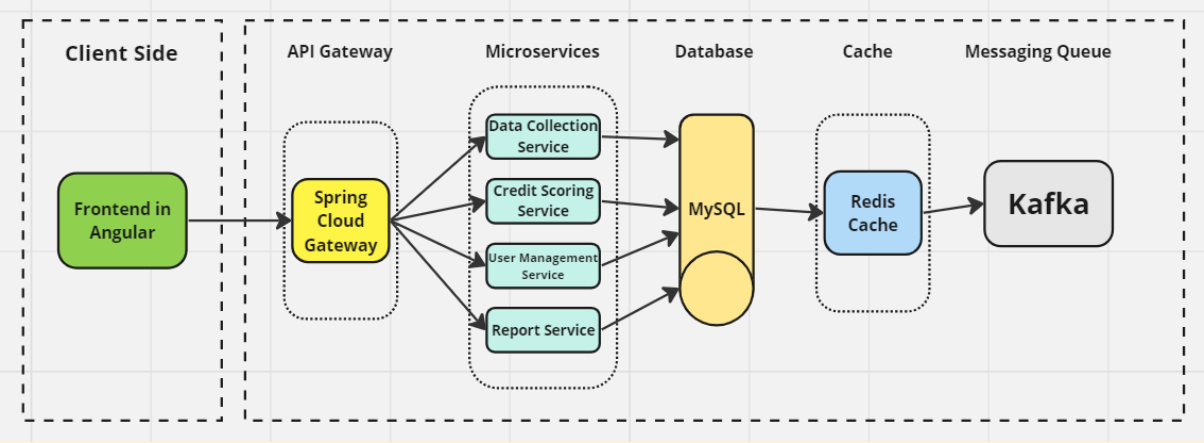
Credit Score Analysis Tool is one of the projects of Swedbank situated in Sweden.

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 both bank staff and customers.

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:

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) 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.

11) What to speak in front of your interviewer?

I recently contributed to the Credit Score Analysis Tool at Swedbank, located in Sweden. This tool streamlines 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 sharpened my skills in developing robust systems and underscores the importance of simplicity in managing credit scores.

12) Project Related All Possible Interview Questions and Answers

Which Java version you are using in this Project?

Currently we are using Java 11

Why did you choose Java 11?

We picked Java 11 for this project because it's reliable and safe for long-term use. It runs smoothly and has some upgrades that make it faster. This helps our application work well and process data efficiently.

Can you tell me some new features that were introduced in Java 11?

HTTP Client, Epsilon Garbage Collector, Z Garbage Collector, Local-Variable Syntax for Lambda Parameters are some of the new features and along with these new features, isBlank(), strip(), stripLeading(), stripTrailing(), and repeat() were also introduced for strings

Which springboot version you are using in this project?

In this project, we are using Spring Boot version 2.5.4. We chose this version because it offers the latest features, improvements, and bug fixes provided by the Spring Boot framework. Additionally, it ensures compatibility with other libraries and tools used in the project ecosystem.

What's the improvements of 2.5.4 version?

Testing enhancements, and dependency management improvements are the major improvements in this version

Can you explain the microservices architecture you used in the Credit Score Analysis Tool? Why did you choose this architecture?

We used a microservices architecture to break down the Credit Score Analysis Tool into smaller, independent services that perform specific tasks. This architecture was chosen because it allows for easier scaling, independent deployment of features, and better fault isolation.

How do microservices communicate in your architecture?

In our architecture, microservices communicate via RESTful APIs over HTTP. They send requests and receive responses in a decoupled manner, ensuring that services can operate independently.

What are the benefits and drawbacks of using an API Gateway?

Benefits: Centralized management of security and authentication, simplified client-side communication, and efficient request routing and load balancing.

Drawbacks: Introduces a single point of failure, potential bottleneck if not properly managed, and can increase complexity in debugging.

How did you ensure data consistency across different services in the Credit Score Analysis Tool project?

We ensured data consistency by using distributed transactions where necessary and implementing compensating transactions to handle failures. Each microservice manages its own database to maintain data integrity.

What databases did you choose for this project and why? How did you handle data modeling?

We chose MySQL for its robustness and support for ACID properties. Data modeling was handled by defining clear schemas based on the requirements of each microservice, ensuring they are optimized for the queries they need to support.

How did you implement security measures in the Credit Score Analysis Tool? Can you detail the authentication and authorization strategies?

We implemented security using OAuth for authentication and JSON Web Tokens (JWT) for authorization. OAuth allows secure delegated access, and JWT ensures that only authenticated and authorized users can access resources.

Can you describe how you managed state in the microservices environment of the Credit Score Analysis Tool?

State management was primarily handled by storing state data in databases and caches. Stateless services were designed where possible to improve scalability and resilience.

What challenges did you face while integrating external APIs for credit data retrieval and how did you overcome them?

We faced challenges such as varying data formats and connectivity issues. These were overcome by implementing data normalization processes and using circuit breakers to handle failures gracefully.

How does Kafka fit into your project, and what are its main roles within the project?

Kafka is used for handling real-time data streams and for enabling asynchronous communication between microservices, particularly in operations like real-time credit score updates.

In what way does your project utilize Kafka for real-time data processing?

Kafka processes streams of incoming financial data in real time, allowing for immediate analysis and response, such as updating credit scores instantaneously.

Could you explain a scenario where real-time processing is critical in your application and how Kafka supports this requirement?

Real-time processing is critical when a bank officer requests a credit score update. Kafka facilitates this by immediately processing incoming data from various sources and enabling the Credit Scoring Service to compute and deliver updated scores in real time.

What are some of the best practices you follow while using Kafka in your projects?

Some best practices include monitoring Kafka's performance regularly, ensuring data is partitioned effectively across the Kafka cluster, and using appropriate consumer groups to maximize throughput and fault tolerance.

Can you explain the OAuth flow used in your project?

The OAuth flow involves users being redirected to an authentication provider where they log in. Upon successful authentication, the provider issues an access token that the user uses to make authenticated requests to our services.

What measures did you take to prevent unauthorized access to user data?

We implemented strict access controls, used secure communication channels (HTTPS), and encrypted sensitive data both at rest and in transit to prevent unauthorized access.

What algorithms did you consider for calculating credit scores? (Part of a different team)

This was handled by a different team, but they considered algorithms based on statistical analysis and machine learning models to predict creditworthiness.

What machine learning libraries or tools did you use in calculating the credit score? (part of a different team)

Again, as handled by a different team, common tools likely included Python libraries such as scikit-learn for machine learning models.

How did you test the effectiveness of your credit scoring algorithm?

The effectiveness was tested through back-testing with historical data to compare the predicted outcomes with actual outcomes, ensuring the algorithm's accuracy and reliability.

How do you handle the generation of large reports without impacting system performance?

We use asynchronous processing and batch operations to handle large report generation, ensuring that the system's performance is not impacted during peak usage.

What formats are available for the credit reports, and how are they secured?

Credit reports are available in PDF and HTML formats. They are secured through encryption and are only accessible through secure authenticated sessions.

Could you walk me through the lifecycle of a report request in your system?

A report request starts with a user request through the UI, which is authenticated and routed to the Report Service. The service retrieves data, generates the report, and then delivers it to the user or emails it as necessary.

How do you ensure that reports are only accessible to authorized users?

Reports are secured using OAuth and JWTs, ensuring that only authenticated and authorized users can access them based on their permissions.

What strategies do you use for detecting and responding to security incidents?

We use real-time monitoring tools, including Splunk, to detect anomalies and potential security incidents. Automated alerts and predefined incident response strategies are in place for quick reaction.

How do you handle data encryption and secure data storage?

Data is encrypted using strong encryption algorithms both at rest and in transit. Secure data storage practices are followed, using encrypted databases and secure backup solutions.

Can you describe the data validation rules or preprocessing steps you implemented?

Data validation involves checking for correctness, completeness, and consistency of data. Preprocessing includes normalization, deduplication, and formatting of data before it enters the processing pipelines.

What measures have you implemented to ensure the security of log data sent to Splunk, especially considering the sensitivity of financial data?

Log data is anonymized where possible, encrypted during transmission and storage, and access to Splunk is tightly controlled and monitored.

How did you integrate Splunk in your code?

Splunk integration involves configuring log4j to send logs directly to Splunk via the HTTP Event Collector (HEC), allowing for real-time log monitoring and analysis.

How would you determine which data should be cached in the "Credit Score Analysis Tool" project?

Data frequently accessed and slow to compute, like credit scores and user session information, are ideal candidates for caching to improve performance.

Explain how you would handle cache expiration and invalidation in a distributed system like on this project.

Cache expiration is managed through time-to-live (TTL) settings, and invalidation is handled manually or via event-driven mechanisms when underlying data changes.

Describe the process of integrating Redis caching into the codebase of your project?

Redis is integrated as a distributed cache through its client libraries. Services interact with Redis via APIs to store and retrieve cached data, utilizing its performance benefits.

How did you handle testing in a microservices architecture for the Credit Score Analysis Tool project?

Testing involves unit tests for individual components, integration tests to ensure microservices interact correctly, and end-to-end tests to validate the entire workflow.

Which framework do you use for unit testing and why?

In our project, we use Mockito to help test our code. It lets us create fake versions of complex parts of our system so we can test each part separately and make sure it works right on its own. This is really helpful for checking how different parts of our project interact.

What was your approach to logging in this project?

Logging is implemented using log4j2, configured to provide detailed logs for debugging and monitoring. Logs are centralized via Splunk for easy access and analysis.

Can you explain the CI/CD pipeline set up for this project? What tools were involved?

The CI/CD pipeline involves Jenkins for automation, involving stages like build, test, and deploy. GitLab is used for version control, and Docker and Kubernetes manage containerization and deployment.

How did you ensure the performance and scalability of the Credit Score Analysis Tool?

Performance is ensured by load testing, optimizing queries and data models, and using scalable components like Kafka and Kubernetes. Scalability is tested regularly to handle increased loads.

Were there any performance bottlenecks in the Credit Score Analysis Tool? How did you identify and address them?

Performance bottlenecks, primarily in database queries and service response times, were identified using profiling tools. They were addressed by optimizing queries, increasing resources, and sometimes refactoring services.

If a specific microservice in the Credit Score Analysis Tool failed, how would you ensure the rest of the system remains operational?

We use circuit breakers and fallback mechanisms to ensure that if one service fails, others can continue to operate, either by switching to backup services or by degrading functionality gracefully.

Imagine you need to update the scoring algorithm without downtime. How would you proceed?

We would use blue-green deployment techniques to deploy the new version alongside the old, gradually shifting traffic to the new version once it's proven stable, ensuring no downtime.

How would you handle a sudden increase in load, such as during a promotional event?

We would use auto-scaling capabilities in Kubernetes to dynamically allocate more resources based on the load, ensuring the system can handle spikes without degradation of performance.

What was the critical feature you worked on in this project?

One of the most important tasks I worked on was setting up real-time credit scoring using Kafka. This meant that Swedbank could quickly get the latest credit scores for customers. By connecting different parts of the system through Kafka, we made sure that whenever new financial data came in, it was processed right away. This helped the bank make faster decisions and provide better service to its customers.

Which methodology do you use for your project management?

We use the Agile methodology, which allows for flexible planning, progressive development, early deployment, and continuous improvement.

How does the Agile approach help in managing the development and deployment of microservices in your project?

Agile provides a framework that supports the dynamic and complex nature of managing multiple microservices, allowing for regular feedback, iterative development, and rapid adjustments as needed.

Describe how you structure sprints in your project. What is the typical duration of a sprint, and how are tasks prioritized?

Sprints are typically two weeks long. Tasks are prioritized based on their business impact and urgency, guided by the product owner in collaboration with the team.

Which Agile framework (Scrum, Kanban, etc.) do you use in your project, and why was it chosen?

We use Scrum because of its structured approach to managing large projects, its emphasis on regular updates, and its ability to handle complexity through roles like Scrum Master and Product Owner.

How are user stories created and maintained for your project? Who is responsible for writing these stories?

User stories are created by the product owner with input from stakeholders and the development team. They are maintained in a product backlog, which is regularly groomed and prioritized.

How is testing integrated into your Agile process? Are there dedicated sprints for testing, or is it continuous?

Testing is continuous throughout the development process. Each sprint includes development and testing tasks, ensuring that new features are both developed and tested within the same sprint.

How did you send the mails by using java mail service ?

First, I ensure the Spring Boot Starter Mail dependency is in my project's pom.xml. Next in application.properties, I set up my mail server details, like host, port, username, and password. Then I write a service class that uses JavaMailSender to send emails.

In this service, I craft the email content and use the send method to dispatch emails. And finally, I call my mail service from within the registration logic to send the email.

Can you tell me How did you integrate a MySQL in your project?

First we have added the MySQL dependency in our project's POM file. Then, in our application properties, we set up the connection details for MySQL, like the database URL and credentials.
Then we created repository interfaces in our code using Spring Data, which helps in interacting with MySQL.

Finally, we use these repositories in our services to save, retrieve, and manage data in our MySQL database.