HIGH LEVEL DESIGN

Banking Application (Console based application)

EXECUTIVE SUMMARY

. The application automates the process of collecting mortgage data from several banks, taking into account that the data can be accessed by executing Semantic Web Services from different banks. Then it provides this aggregated information to users, according to the data that they have filled-in in appropriate query forms. Following the DIP standards, we present an extensive description of the application, the requirements (functional, non-functional and project constraints) and a sketch of the desired user interface.

Monolithic Architecture

The traditional way to implement bank management systems is with a monolithic architecture, where different tasks are managed with a single unified unit. Java handles a variety of jobs that come from the branch office, from electronic card processing and from the requests to API open to other banks and clients.

The load balancer distributes evenly jobs between application servers that run multiple copies of the application, and on the backside, the application manages database requests. The load balancer also plays the role of the inner firewall, in addition to the outer firewall that provides first-level protection from network attacks.

The architecture is simple and thus easy to develop and deploy, scaling is also relatively easy with the load balancers - just add another application/database server when needed.

The monolithic architecture is the easiest to implement, but it is hard to maintain in the long run, and it is very hard to add new features or update the old ones.

Moreover, with a monolithic architecture, it is hard to introduce new tools and frameworks to the developing stack, as there are no points inside the monolithic structure to connect new technologies.

Event-Driven Architecture

Event-driven architecture is one of the alternatives to the traditional Monolithic architecture that centers around "events" rather than entities.

For example, events when an office teller submits a new client, when a customer presses an ATM button to withdraw money, or when the account balance is getting lower than the threshold. All incoming events are registered within the API layer and added to the Kafka stream.

The Apache Kafka, an open-source distributed event streaming platform, distributes events between the consumers: Application Jobs, Data storage, Statistics collectors, Notification routines, and the special Fraud Check engine. Fraud Check is written in Python to automatically detect and block suspicious transactions to prevent money laundering and other violations.

With the event-driven architecture, implementing new services is easier, as there is no need to constantly synchronize states between them, for example, to synchronize the current balance state with the upcoming transactions from credit cards and open API.

Instead, the program can introduce an even that controls and fixes the balance changes, so the transaction service only needs information about balance changes and does not care about other transactions and processes.

As event-driven architecture requires a careful design in the initial stages and thus needs more time and resources spend on development, this is a good option for large financial institutions that aim for a large client auditory and provide a row of different services.