The Financial Trading System architecture is presented in a UML diagram. It contains 6 components (5 components + 1 database) and 1 actor.

Actors: Administrators, Financial consultants. The end-users of this system are the staff working with the internal system of the bank.

Web interface component

End-users use the interface (accessible through a web browser) to allow them to interact with the system. They can log in within the system (API to authentication component), give input for calculations (API to computational-intensive calculation component) or get a notification (when sent by Notification Service). It is built as an Application Layered Architecture style, and together with Authentication component, works as a centralized system architecture.

Authentication component

It starts when end-user logs in, and the component receives from web interface an username and password. It has 2 methods: getUser(), that gets the data (userID) of the end-user from a database, and authorizeCAStoken(), that will send the userID to authorization service, service that will generate a token, which will be sent back to authentication. The end-user will receive permission to use the other components and the system will not ask them to introduce their credentials until they log out (by using the token generated for authorization). It is built as an Application Layered Architecture style.

Customer Database

It contains the data, history, processes, and any actions associated with customers. It is built as a Resource-Based Architecture (ROA). It communicates with Authentication (and indirectly with Web Interface) via RESTful APIs.

Notification Service

It pushes notifications to end-users via web interface, with the possibility of actors to choose if they want e-mail and/or push-up notification. It notifies them about a finished calculation, for example. It is based on Service-Oriented Architecture style. Communicates with the other components via event triggers.

Computational-Intensive Calculation

It uses input data from end-users and follows a request for calculation (stocks, credits, etc.). It has the capability to execute the hard-calculation and also make a time series simulation over a certain set period. It can work also offline, and it stored and retrieves the input and output data, and when finished it assigns it to the customer’s entry in database. Its architecture style is Object-based.

Calculation Storage

Stores the input and output data for Computational-Intensive Calculation. Uses Resource-based Architecture. It communicates with Computational-Intensive Calculation component via RESTful APIs.

The system is build as a hybrid architecture, where the web interface and authentication/authorization components work as a centralized part, and the rest of the components are decentralized (Peer-to-Peer). The system follows the requirements described in the assignment, showing transparency regarding the actors interacting with the system, and how components communicate with each other.