CeX

CeX: Currency Exchange

By: Andres Giovanne Florez Perez

Date: 29/03/2019

Change control

Date	Author	comment
29/03/2019	Andres Giovanne Florez	final delivery

Table of contents

1. Introduction

This document presents the architecture of the CeX system. Currency Exchange is a currency converter developed as a showcase for a client-server application that uses the latest Java technologies and other related technologies. The application has been structured using a three-layer approach following the guidelines of Java Enterprise Edition 7. In addition, it uses Bootstrap 4 (getbootstrap.com), JavaScript, axios to consume and jQuery to provide a user interface compatible with HTML5, responsive and HTML5 compatible Maven 4.0.0 was used to support the automation of the software development life cycle, including automatic compilations and dependency management. Tomcat 7.0 was used as an application server. The exchange rate information is obtained through a RESTful API, with JSON objects, from www.apilayer.net

This document is structured as follows. Section 2 presents the requirements analysis, which shows the functional requirements, quality attribute requirements and restrictions. Section three shows and explains the main design decisions that were made during the first cycle of the project. Section 4 presents the high-level architecture of the system. In that section you will find the main metaphor behind the design. The detailed design is presented in section 5 and the list of known problems is presented in section 6.

2. Requirement analysis

2.1. Original requirement statement

You are an architect in an organization that provides easy to scale and modern solutions, you are required to architect, design and implement a client server application, the client is either a modern web client or a desktop application with a simple user interface, the server is a virtual **currency exchange rate publishing server** and the communication channel is **TCP** or **HTTP**.

The server would expose different services for the client application to consume, list of exchange rates for different currencies, search for the exchange rate for a specific currency and a currency converter to convert a specific amount of money from a currency to another. Use one of the public currency converter API(https://openexchangerates.org/signup/free,https://currencylayer.com/documentatio n, etc...) at server side to get latest conversion rates.

The client application should provide simple and clear interface to the end user to perform different functionalities exposed by the server, listing currencies with their exchange rates, search box for a specific currency, currency exchange box and a refresh button to refresh the rates from the server.

The client application should be **responsive**, any freezes in the client app is not acceptable, the network operations should be **Non blocking** operations, the user should be notified that there is a server call and get notified once response received and rendered without any hangs.

You should use **modern Java technologies** to architect and implement the application with the mentioned specifications.

2.2. Functional requirements

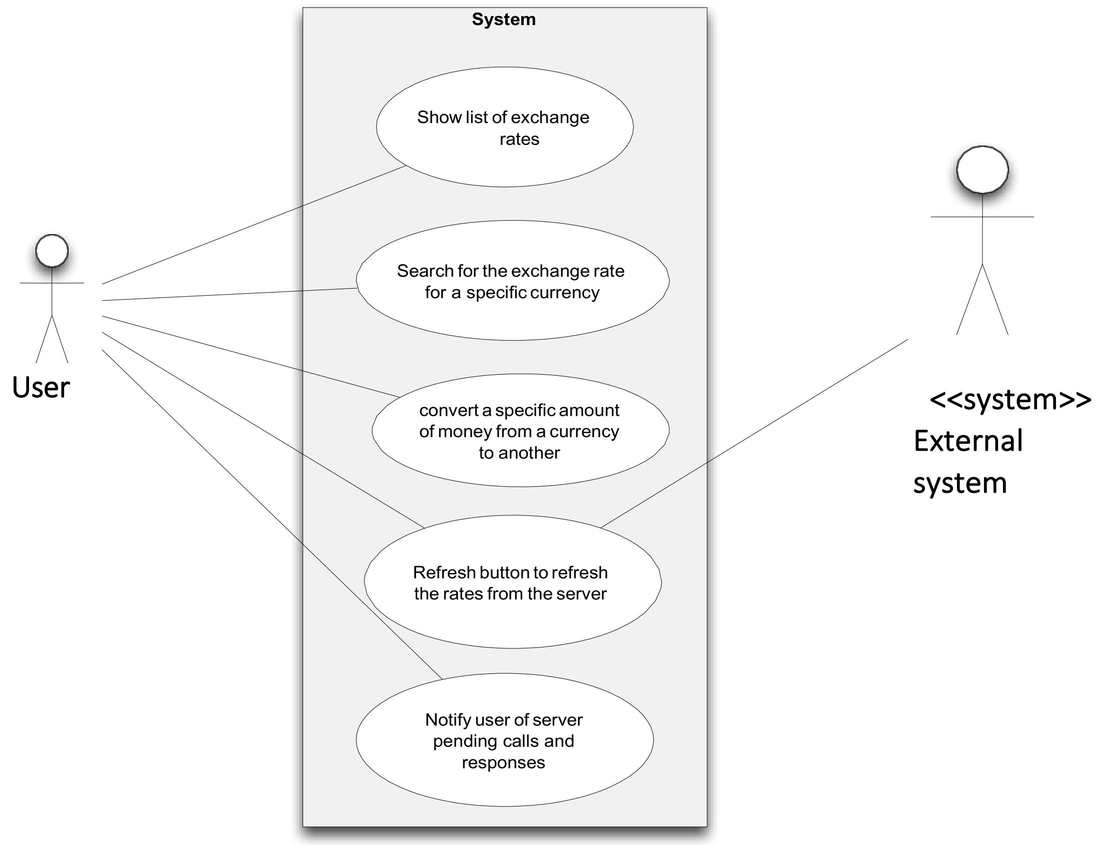

System
Show list of exchange rates
Search for the exchange rate for a specific currency
convert a specific amount of money from a currency to another
Refresh button to refresh the rates from the server
Notify user of server pending calls and responses
User
<<system>> External system

Figura 2.1 Uses cases for the system

Figure 2.1, shows a use cases diagram depicting the functional requirements that where identified:

1. List of exchange rates for different currencies
2. Search for the exchange rate for a specific currency
3. A currency converter to convert a specific amount of money from a currency to another.
4. Refresh button to refresh the rates from the server.
5. The network operations should be **Non blocking** operations, the user should be notified that there is a server call and get notified once response received and rendered without any hangs.

2.3. Quality attributes

The following quality attributes requirements were identified:

1 **Responsive:** The client application should be **responsive**, any freezes in the client app is not acceptable,
2 **Non blocking:** The network operations should be **Non blocking** operations.
3 **Usability:** The client application should provide simple and clear interface to the end user
4 **Scalability:** The application should be scalable.

2.4. Constrains

The following is a list of the constrains for the development:

1. The application should use modern Java technologies.
2. the communication channel is **TCP** or **HTTP**
3. Use Maven as a dependency management tool.

3. Design decisions

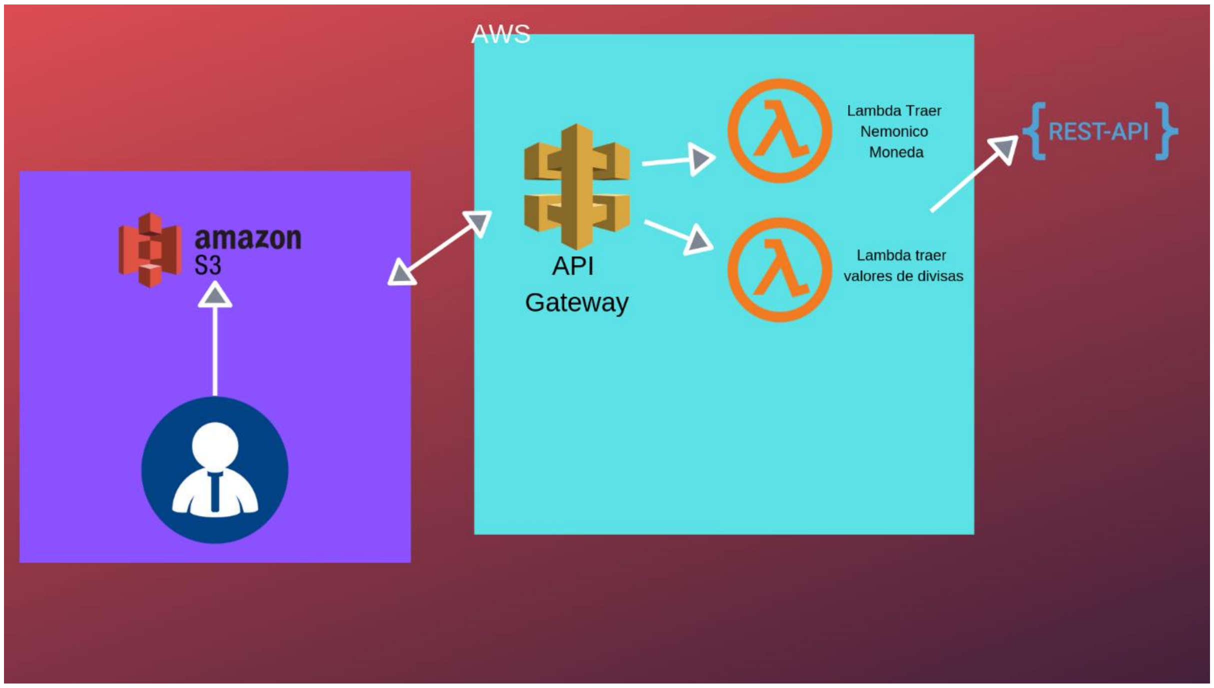
3.1. Architecture Style

Regarding the architectural style, we consider two alternatives. First, we could use Java frameworks, such as those proposed by sparkjava.com. These frameworks use the POJO Java development and the latest Java 8 features (for example, lambda functions) to provide web support, taking full advantage of the power and simplicity of Java and the JVM. Even difficult, this approach is excellent for the implementation of microservices and the rapid creation of prototypes, could pose problems if the evolution of the application includes more complex requirements of security, transactions and scalability. On the other hand, we also consider the full stack of Java Enterprise Edition 7 technologies. This technology provides a mature set of standards and a recommended three-layer approach to structuring applications. It also provides transparent support for complex security, transaction, high availability and other requirements. However, the style of declarative programming and the required infrastructure could create difficulties for maintainability. Programmers must have advanced knowledge in the JEE platform to address maintenance and evolution.

Taking into account the previous alternative, we decided to use JEE 7 in combination with modern web technologies such as HTML5, JavaScript, jQuery and others (see section 3.5, Stack of

technologies). The decision is motivated by several aspects. First, the programming style for JEE 7 has been drastically simplified. The standard is quite mature and well maintained. Added support for new technologies in each version, for example, JEE 7 provides native support for web services, web services RESTful, JSON, JavaScript and others. The open source community and the business community provide invaluable resources and support on the Internet. The platform allows to evolve and adopt complex security, transaction and grouping requirements without changing the implementation. The search for efficient and efficient application servers is ongoing.

In summary, we are going to use a three-layer approach. The first layer will be the presentation layer with a web client. The second layer will support the business logic. The third layer is the external provider from which we obtain information on exchange rates, this layer serves as a layer of delegated persistence. The web client will use the faces of the Java server, HTML5, JavaScript, jQuery and others. The server will use POJOS to implement the business logic.



3.2. Communication technologies

The requirement raises two restrictions with respect to communications, the communication channel must be HTTP or TCP, and network operations must not be blocked. Considering these requirements, we chose to use a RESTful API through HTTP to retrieve the information of the exchange rates of an external provider (in this case www.apilayer.net). We also decided to implement a web client that will use HTTP to communicate with the server.

The application will run on an S3 server which is the part of the front then it communicates with the API gateway who has lambda functionalities, and this consumes directly to the rest API by HTTP.

3.3. Responsive interface

The requirement establishes that the interface must be receptive, emphasizing the non-blocking aspect of the graphical user interface. To address this requirement, we decided to implement our client using modern web technologies. In particular, we will use HTML5 CSS and JavaScript and jQuery frameworks and axios in addition to bootstrap4 to keep the page responsive and take advantage of non-blocking aspects of our interface and thus be able to make any device adapt to the page

3.4. Performance and Caching policies

With respect to the list of supported currencies, this information is also retrieved from the server, but it is only recovered once when the applications are loaded. This information is considered static, if the situation changes (that is, more coins are added) the policy may vary.

3.5. Stack of technologies

This section summarizes the discussions in this chapter by presenting a list of the technologies used in this project and a short sentence to remember the justification:

1. Java 8, use of modern java technologies.
2. JEE 7, a robust and mature platform, supports complex security, scalability, transaction and communication requirements.
3. NetBeans 8, eclipse, visual studio code.
4. S3 runs with HTML5, AJAX and JavaScript.
5. Lambda functions made in java with maven.
6. JavaScript, jQuery, axios, to support the non-blocking behavior in the client.
7. Bootstrap 4.0, to address browser portability, aesthetics and compatibility with mobile devices and desktops (Responsive interface).
8. RESTful webservices, to communicate with the external provider.

4. Known limitations and future work

The problem of the cache could not be solved and there is not a good way to use the lambda properly since it is bringing all the json and not something specific.

4.1. Web interface fine tuning

The interface has been tested, however for a production deployment more tests and fine tuning must be done. For example, more options may be provided to reset and clear the dashboard.

4.2. Error management

The implementation is not done to control errors against the user

4.3. Deployment tests

No deployments tests were executed.

4.4. Clustering tests

The design seems simple and robust enough to support default clustering by the application server. However, no clustering tests have been performed.

4.5. Extensive user tests

The application has very few user tests.

4.6. Xtreme refactoring where needed

Refactoring is always needed, in impossible to have a perfect design in the first attempt. Refactoring! Refactoring! Refactoring!

4.7. Persistence management

We have not implemented persistence. However, more complex requirements may force for a persistence implementation. The system may use JPA for such endeavor.

4.8. Cross browser tests

The application hasn't been tested in Mac browsers.

4.9. Lambda functions

The lambda functions worked perfect as POJOS

4.10. Api Gateway

There were some problems when using the lambda function in the Gateway API since it was necessary to activate the CORS and add some headers