EagleLibrary - System Design

La progettazione del sistema EagleLibrary inizia dalla stilatura dei requisiti funzionali e non. Per poter individuare quelli che, secondo noi, erano i requisiti che caratterizzavano l'applicazione da implementare abbiamo in prima istanza riportato tutti quei requisiti che si evincevano facilmente dalle specifiche del progetto, in più, ci siamo immedesimati nell'utente finale dell'applicazione per cercare di formalizzare altri requisiti che rendessero il sistema quanto più semplice ed user friendly possibile.

Dunque i documenti che abbiamo prodotto come output di tale procedura sono disponibili al seguente indirizzo :
https://github.com/giodag/eagleLibrary/blob/develop/doc/Requirement%20and%20Assumptions/Requisiti.docx

Il secondo step è stato quello di partire dai requisiti e dalle specifiche di progetto per tirare fuori
dei casi d'uso che iniziassero a farci capire ad alto livello quale sarebbe state le interazioni tra :

- user & system;
- subSystem & subSystem.

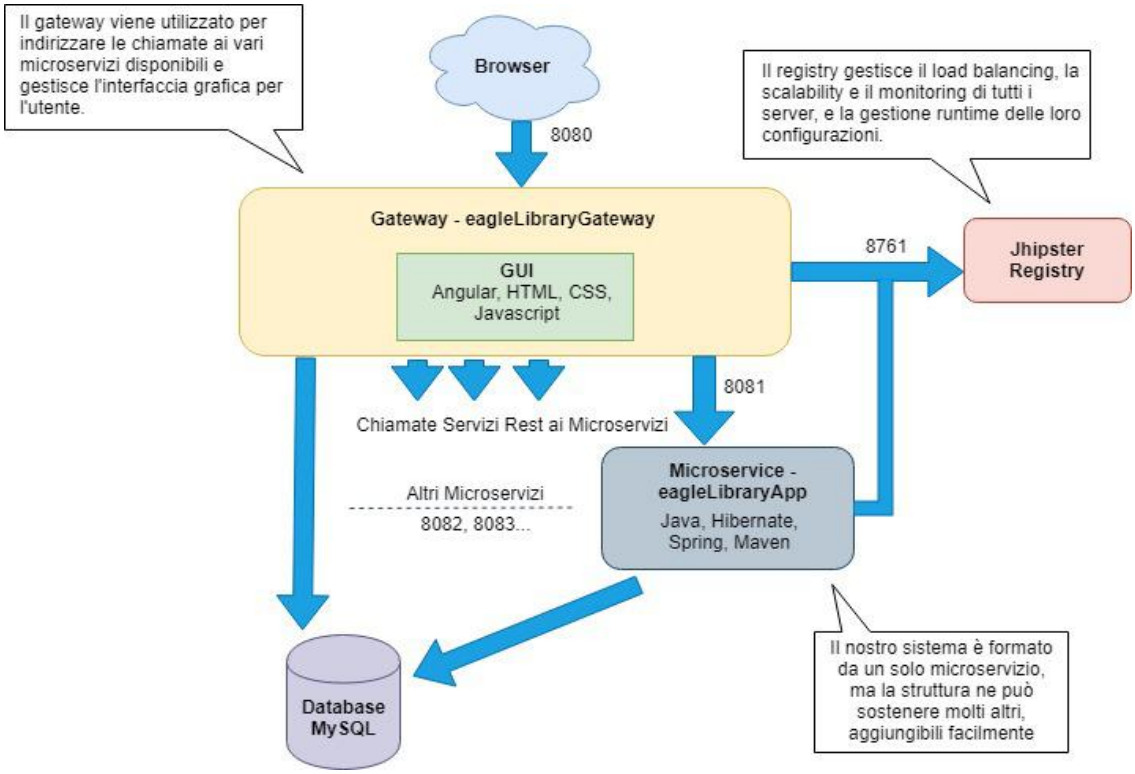
Ricavando dunque i vari use case implementati nel file disponibile all'indirizzo che segue :
https://github.com/giodag/eagleLibrary/blob/develop/doc/Models/EagleLibrary.mdzip

A questo punto ci siamo concetrati sull'estrazione delle entità che caratterizzano il sistema e abbiamo formalizzato tutto in un class diagram. In questa fase abbiamo dovuto fare delle assunzioni che ci hanno permesso di semplificare alcuni comportamenti del sistema e degli attori che vi interagiscono.

Le assunzioni di cui sopra sono disponibili al seguente link :
https://github.com/giodag/eagleLibrary/blob/develop/doc/Requirement%20and%20Assumptions/Assunzioni.docx

Una volta che sono diventati chiari quali dovessero essere le entità e come avrebbero dovuto interagire tra loro ad alto livello abbiamo iniziato a pensare al tipo di architettura da utilizzare. Fin da subito abbiamo voluto scegliere qualcosa che divergesse dal classico monolite così siamo andati verso una soluzione più modulare che non richiedesse aver presente già tutta l'architettura del sistema. Senza precluderci nulla quindi, abbiamo optato per un'architettura a microservizi. Nella fattispecie di implementare (almeno per la prima versione del programma) 3 microservizi deployabili indipendentemente con Spring Boot. I tre microservizi sono i seguenti :

- Registry - modulo che agevola il monitoring, server balance, application's performance e così via.
- Gateway - modulo che gestisce la parte grafica e che comunica con il microservizio che verrà presentato al punto successivo per mezzo di chiamate rest.
- Application - modulo che gestisce tutta la parte back end della nostra applicazione.



La piattaforma che ci ha aiutato a realizzare l'architettura del nostro progetto è stata Jhipster.
Come prerequisito all'utilizzo di Jhipster abbiamo dovuto installare sui nostri terminali :

- Java 8;
- NodeJs 8.11.3 LTS (le versioni non LTS non sono supportate);
- Yeoman 2.0.5;
- Yarn 1.13.0;

Jhipster ci ha aiutato molto per la generazione automatica della DTS e di conseguenza di tutta la struttura del progetto. Inoltre abbiamo utilizzato la piattaforma anche per generare una prima versione delle JPA, dei POJO e dei CRUD per le entità principali dell'applicazione.

Lo start up dei vari moduli con l'ausilio di Jhipster ha incluso nel progetto features che potremmo decidere di non tenere e/o customizzare a seconda delle esigenze del cliente e del team di sviluppo.

Tra i framework che abbiamo utilizzato ci sono anche i seguenti :

- Maven per la gestione del progetto, build automation e la gestione delle dipendenze;
- Spring, nella fattispecie l'utilizzo di @Annotation per lo scanning e il detecting dei package nel progetto. In più lo utilizziamo per l'injection.
- Hibernate per la gestione del DB lato backend.

La scelta sul RDBMS da utilizzare è ricaduta su MySQL, un database open source molto semplice da utilizzare, gestibile con i vari tool messi a disposizione da Oracle, con il quale abbiamo già avuto esperienza in passato.

Attraverso questi passaggi abbiamo quindi consolidato la parte architetturale della nostra applicazione, a questo punto abbiamo fatto un passo indietro e siamo tornati sulle specifiche del progetto per guardare i requisiti da un punto di vista differente, invece di elencare cosa potesse/oppure no fare un subSystem [mi riferisco quindi a Viewer, Transcriber, etc...] abbiamo implementato uno schema che descrivesse tutte e sole le operazioni che vengono effettuate dai subSystem sulle entità che abbiamo individuato.

Tale schema è consultabile al link :
https://github.com/giodag/eagleLibrary/blob/master/doc/Models/Entity%20-%20Sub%20System%20Relations/Entity%20-%20Sub%20System%20Relations.txt

Questa nuova prospettiva ci ha offerto spunti per iniziare a buttare giù i primi high level design (che da qui denoterò con HLD) intesi come activity diagram (almeno in questa prima fase, solo activity) che appunto spiegano come interagiscono le entità con i vari subSystem.

Tali design sono disponibili al seguente link :
https://github.com/giodag/eagleLibrary/tree/master/doc/Models/Activity%20Diagrams