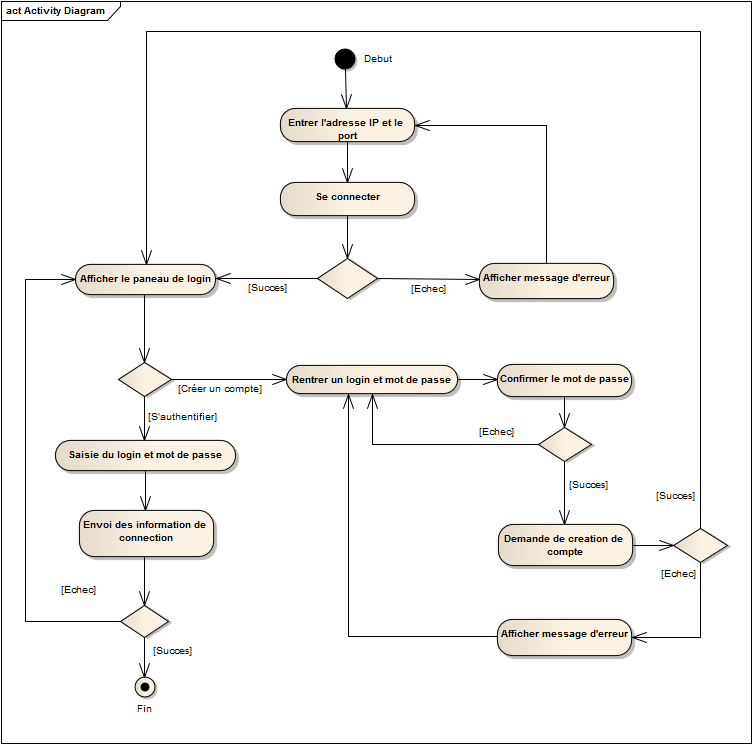
# Diagramme d’activité

Le diagramme d’activité représenter les différents scénarios du lancement du programme jusqu'à être connecté et authentifié (logué) sur le serveur.



# Diagramme de séquence

Le diagramme suivant est un diagramme de séquence. Il représente les différents messages concrets entre le client et le serveur. (cf chapitre sur le diagramme de classe du serveur et du client pour plus d’information sur les classes)

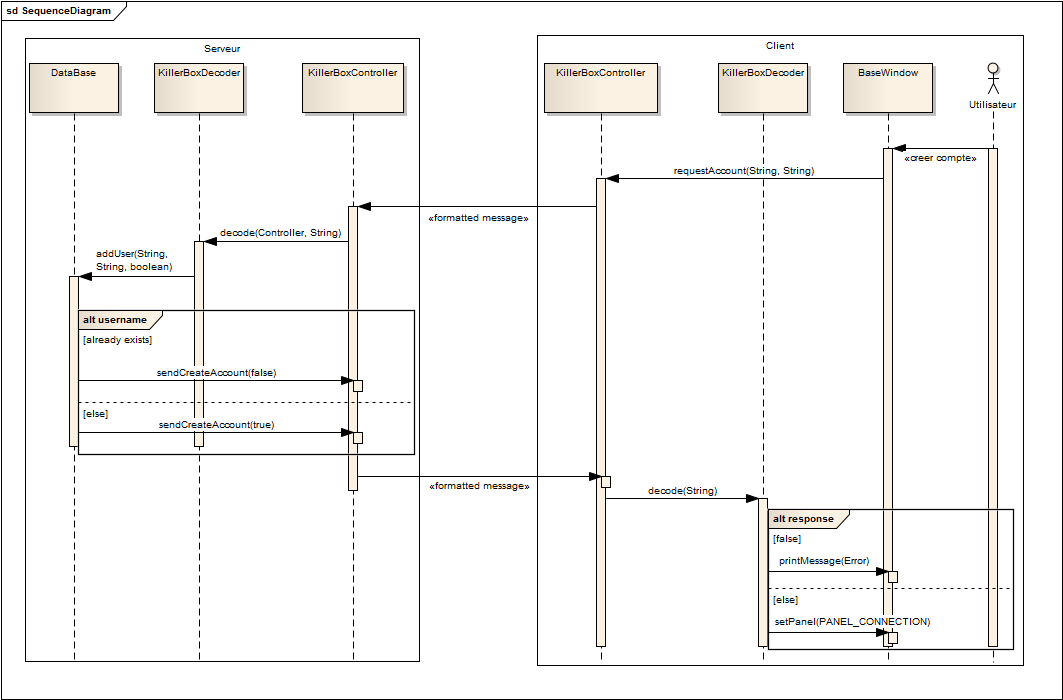
A gauche se trouve le serveur et à droite le client.

Les messages échangés représentent la demande de création d’un compte sur le serveur, par un utilisateur quelconque. Les stéréotypes « *formatted  message* » indiquent un protocole d’échange bien défini entre les deux applications.

L’utilisateur (via l’interface) graphique, décide de créer un compte sur le serveur. L’action sur le bouton « créer compte » fait appel au contrôleur qui envoie les informations pour créer le nouveau compte. Le message formaté circule via le réseau pour afin arriver sur le contrôleur du coté serveur. Le contrôleur va ensuite transférer l’intégralité du message au décodeur (protocole). En lui passant une référence sur le contrôleur de la connexion qui a générée le message.

Le décodeur va décomposer le message et reconnaître sa signification. Celui-ci va comprendre qu’il faut ajouter un utilisateur (*user*) avec le mot-de-passe (*pass*) dans la base de données à l’aide de la classe *DataBase*. De là, deux situations peuvent se produire : L’utilisateur existe déjà dans la base de données, dans l’autre cas, l’ajout du compte à réussi. Dans ces deux cas, le contrôleur envoi un message au client associé à ce contrôleur. Un booléen est utilisé pour indiquer le succès ou l’échec de l’opération.

Retour côté client. Le contrôleur reçoit le message et le passe au décodeur. En cas d’une réponse négative, le décodeur indique à l’interface d’afficher un message d’erreur, sinon l’utilisateur est redirigé vers la page de création de compte.



# Diagramme de classe du serveur

La partie serveur est décomposée en 2 packages.

* « network »
* « killerbox »

## Package network

Le package network permet de fournir une structure de classe la plus réutilisable possible. Elle est décomposée des classes suivantes : *Server, Controller, Decoder, AbstractServerStatus, MessageServerStatus, EnumServerStatus*.

### Server

Cette classe permet représenter un serveur basique. Elle permet à des clients de se connecter. Un thread principal est en attente de connexion. A chaque nouvelle demande, un thread est crée ainsi qu’un nouveau contrôleur pour cette connexion. La gestion complète du client est confiée au thread.

Les différentes connexions sont indexées à l’aide d’un ID unique. Afin de faire le lien entre le contrôleur et l’ID de connexion, une table de hachage est utilisée. L’ID est utilisé comme clé et l’instance du contrôleur comme donnée.

La classe est considérée comme un « modèle ». Ce modèle est susceptible d’émettre des messages de type « AbstractServerStatus ».

### Controller

C’est le contrôleur pour chaque connexion. Celui-ci (par le biais d’un thread) permet de récupérer les messages du client. Cette classe fournit également une méthode permettant d’envoyer de simples chaînes de caractère au client.

### Decoder

Est une classe abstraite permettant de décoder un message. Le protocole doit être défini par les classes utilisatrices. Concrètement, la méthode *decode(string)* doit être implémentée.

### AbstractServerStatus

Il est possible d’utiliser cette classe afin d’ajouter différents messages du serveur. (En cas d’ajout de fonctionnalités par exemple). Celui-ci est composé d’un *EnumServerStatus* afin d’indiquer le type de message. Cette classe permet en outre d’implémenter une architecture MVC sur le serveur. C'est-à-dire afficher les messages sur une console, un panneau graphique, etc.

