1. Énoncé

Le but de ce projet est de réaliser une application permettant à deux utilisateurs de communiquer l’un avec l’autre par écrit et à distance. Cette application consistera en réalité en deux applications distinctes, une application client et une application serveur. Les clients seront capables de communiquer uniquement avec le serveur qui redirigera les messages aux clients correspondants.

Les informations des utilisateurs seront stockées en base de données relationnelle (mySQL). Quant aux messages des utilisateurs, ils seront conservés dans une base de données non relationnelle (Cassandra).

Afin que les utilisateurs puissent voir les messages hors connexion, ces derniers seront également enregistrés localement dans un fichier. Ce fichier sera synchronisé avec la base de données lors du lancement de l’application.

Les utilisateurs possèderont un profil contenant diverses informations sur eux-mêmes, notamment leur pseudo, leur adresse mail, leur date de naissance, leur description, leurs numéros de gsm et / ou de téléphone fixe ainsi qu’une image de profil. Ces informations seront publiques ou privées selon le choix de l’utilisateur.

Un utilisateur pourra communiquer avec un autre utilisateur lorsque ce dernier est enregistré dans sa liste de contact. Pour ce faire, il doit effectuer une recherche parmi les utilisateurs de l’application via son pseudo ou son adresse mail, le sélectionner dans les résultats et enfin lui envoyer une demande de contact.

Un utilisateur pourra voir le profil des autres utilisateurs.

1. Glossaire

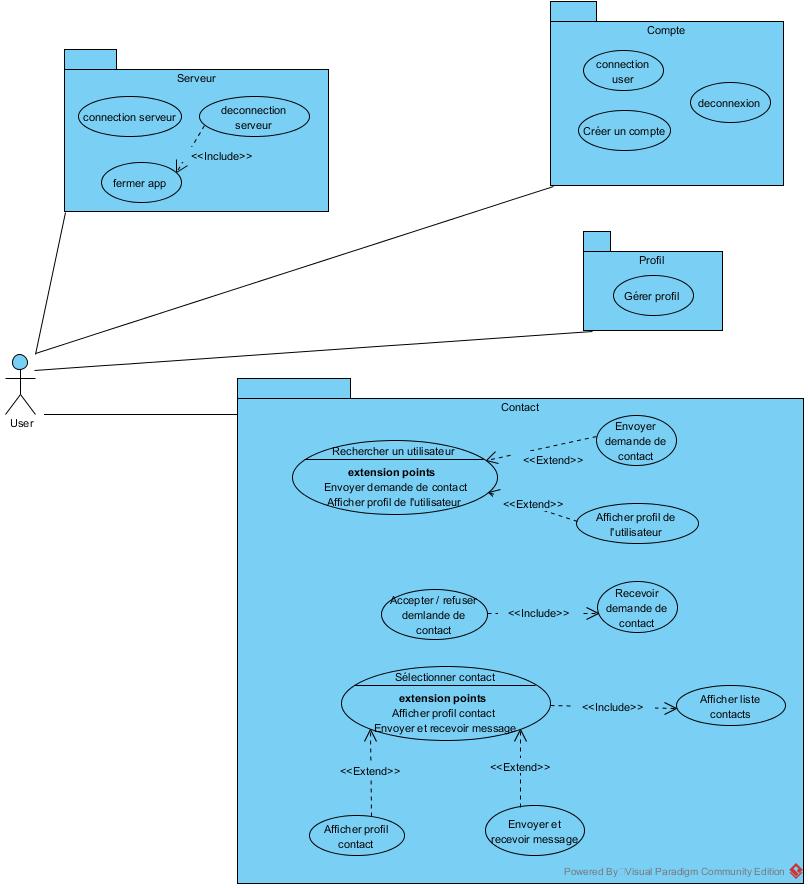
* État d’un utilisateur : indique si l’utilisateur est connecté, déconnecté, occupé, …
* Client : application utilisée par l’utilisateur

1. Détails et responsabilités des acteurs

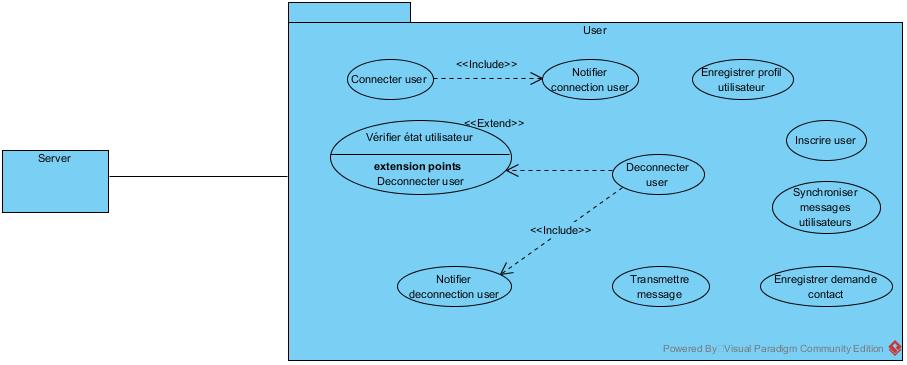
* L’utilisateur :
  + Se connecter
  + Créer un compte
  + Gérer son profil
  + Rechercher un utilisateur
  + Envoyer une demande de contact à un utilisateur
  + Recevoir et accepter / refuser une demande de contact d’un autre utilisateur
  + Sélectionner un contact pour ouvrir la fenêtre de discussion
  + Envoyer un message à un contact grâce à la fenêtre de discussion
  + Recevoir un message
* Serveur :
  + Indique aux contacts d’un utilisateur que ce dernier est connecté
  + Vérifie l’état d’un utilisateur
  + Transmet les messages d’un utilisateur à un autre
  + Connecter l’utilisateur
  + Synchroniser les messages de l’utilisateur
  + Enregistrer les demandes de contact des utilisateurs

1. Diagramme des cas d’utilisation

* Utilisateur



* Serveur



1. Description et énumération des UC

|  |  |  |  |
| --- | --- | --- | --- |
| Numéro de UC | Application | Nom du UC | Description |
| 1 | Client | Connection user | L’utilisateur rentre son identifiant / adresse mail ainsi que son mot de passe. |
| 2 | Client | Créer un compte | L’utilisateur choisira un identifiant unique, un mot de passe et renseignera ses informations personnelles (adresse mail, date de naissance, …). |
| 3 | Client | Gérer profil | L’utilisateur aura la possibilité de gérer son profil en choisissant une image de profil, renseignant une description de lui, son numéro de téléphone, son numéro de gsm, …  Il pourra aussi décider qui (lui uniquement, ses contacts, tout le monde) aura le droit de voir quelle information. |
| 4 | Client | Rechercher un utilisateur | L’utilisateur pourra rechercher un autre utilisateur en renseignant son pseudo. |
| 5 | Client | Afficher profil d’un utilisateur | Ce UC nécessite l’accomplissement de l’UC 4.  L’utilisateur peut consulter le profil d’un utilisateur. |
| 6 | Client | Envoyer demande de contact | Ce UC nécessite l’accomplissement de l’UC 4.  L’utilisateur peut envoyer une demande de contact à un autre utilisateur. |
| 7 | Client | Accepter / refuser une demande de contact | Après avoir reçu une demande de contact, l’utilisateur peut décider de refuser ou d’accepter celle-ci. |
| 8 | Client | Afficher liste de contacts | L’utilisateur peut afficher sa liste de contact et en sélectionner un. |
| 9 | Client | Afficher profil contact | Ce UC nécessite l’accomplissement de l’UC 8.  L’utilisateur peut consulter le profil d’un de ses contacts. |
| 10 | Client | Envoyer et recevoir message | Ce UC nécessite l’accomplissement de l’UC 8.  L’utilisateur peut envoyer et recevoir des messages de ses contacts. |
| 11 | Serveur | Connecter user | Lorsqu’un utilisateur voudra se connecter, le serveur vérifiera les informations de l’utilisateur et le connectera si ces dernières sont correctes.  Les contacts de l’utilisateur seront informés de sa connexion. |
| 12 | Serveur | Inscrire user | Lorsqu’un utilisateur voudra créer un compte, le serveur vérifiera si l’adresse mail ainsi que l’identifiant ne sont pas déjà utilisés et enregistrera les informations de l’utilisateur dans la base de données. |
| 13 | Serveur | Vérifier état user | Régulièrement, le serveur vérifiera l’état des utilisateurs connectés et adaptera le statut de ceux-ci si cela est nécessaire. |
| 14 | Serveur | Transmettre message | Lorsqu’un utilisateur enverra un message à un autre utilisateur, le serveur enregistrera le message, l’heure et la date auxquelles il a été envoyé, le destinataire, l’émetteur ainsi que son statut puis transmettra le message au destinataire. |
| 15 | Serveur | Enregistrer profil utilisateur | Lorsqu’un utilisateur modifie son profil, les modifications sont enregistrées par le serveur |
| 16 | Serveur | Synchroniser messages utilisateur | Si l’utilisateur a utilisé l’application sur un autre appareil, il sera nécessaire de synchroniser les messages stockés sur le serveur avec les messages stockés sur un fichier en local sur l’appareil de l’utilisateur |
| 17 | Serveur | Enregistrer demande contact | Lorsqu’un utilisateur envoie une demande de contact à un autre utilisateur, le serveur enregistre cette demande en base de données afin que la demande puisse être réceptionnée si le destinataire est déconnecté. |
| 18 | Client | Connection serveur | Lors du lancement de l’application client, ce dernier essaie de se connecter à l’application serveur. |
| 19 | Client | Deconnection serveur | Lorsque l’utilisateur ferme l’application, cette dernière se déconnecte d’abord du serveur. |
| 20 | Serveur | Deconnecter user | Lorsqu’un utilisateur se déconnecte, le serveur ferme tous les threads liés à cet utilisateur et notifie ses contacts de la déconnection. |
| 21 | Client | Deconnexion | L’utilisateur se déconnecte de son compte. |

1. Répartition des UC par itération

|  |  |  |  |
| --- | --- | --- | --- |
| Itération | Application | No UC | UC |
| Itération 1 | Client | 18 | Connection serveur |
| 1 | Connection user |
| 21 | Deconnexion |
| 19 | Deconnection serveur |
| 2 | Créer un compte |
| Serveur | 11 | Connecter user |
| 20 | Deconnecter user |
| 12 | Inscrire user |
| 13 | Vérifier état user |
| Itération 2 | Client | 3 | Gérer profil |
| 4 | Rechercher un utilisateur |
| 5 | Afficher profil d’un utilisateur |
| Serveur | 15 | Enregistrer profil utilisateur |
| Itération 3 | Client | 6 | Envoyer demande de contact |
| 7 | Accepter / refuser une demande de contact |
| 8 | Afficher liste de contacts |
| 9 | Afficher profil contact |
| Serveur | 17 | Enregistrer demande contact |
| Itération 4 | Client | 10 | Envoyer et recevoir message |
| Serveur | 14 | Transmettre message |
| 16 | Synchroniser messages utilisateur |

1. Modèle de données

USER

adresse\_mail : varchar(300)

password : varchar(300)

pseudo : varchar (50)

contacts[0-N] : USER

date\_naissance : date

status : char(1)

description : varchar(3000)

no\_gsm : char(20)

no\_fixe : char(20)

DEMANDE\_CONTACT

id : int

date : date

message : varchar(3000)

CONFIDENTIALITE

conf\_adresseMail : char(1)

conf\_dateNaissance : char(1)

conf\_noGsm : char(1)

conf\_noFixe : char(1)

Contraintes d’intégrité

status : c = connecté

o = occupé

n = ne pas déranger

i = invisible

d = déconnecté

conf\_adresseMail : v = visible

c = contacts uniquement

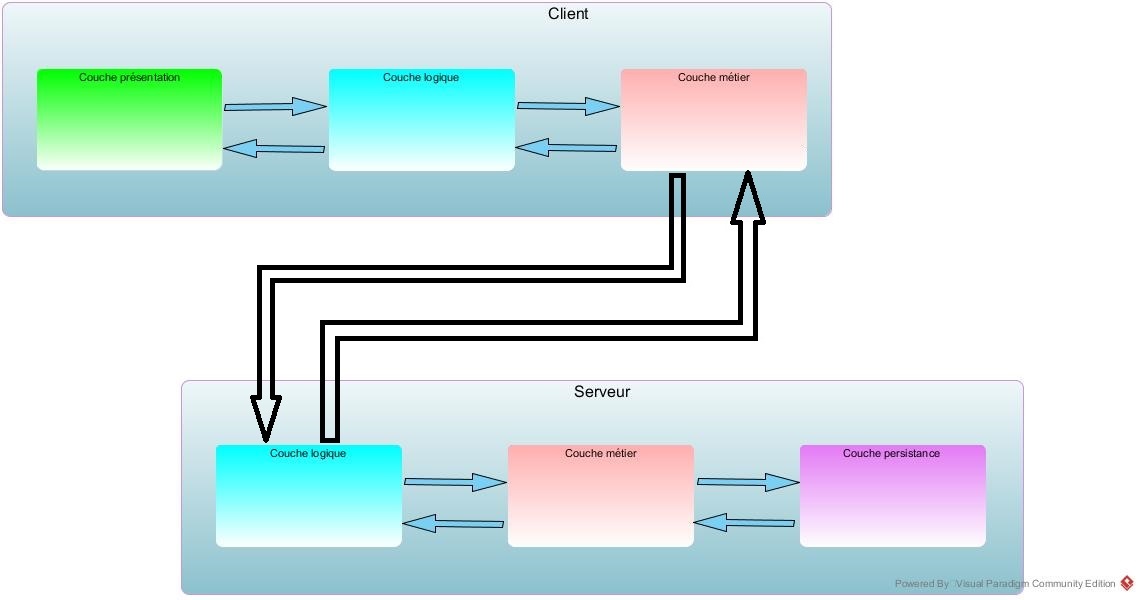
p = privé

conf\_dateNaissance : voir conf\_adresseMail

conf\_noGsm : voir conf\_adresseMail

conf\_noFixe : voir conf\_adresseMail

1. Architecture de l’application



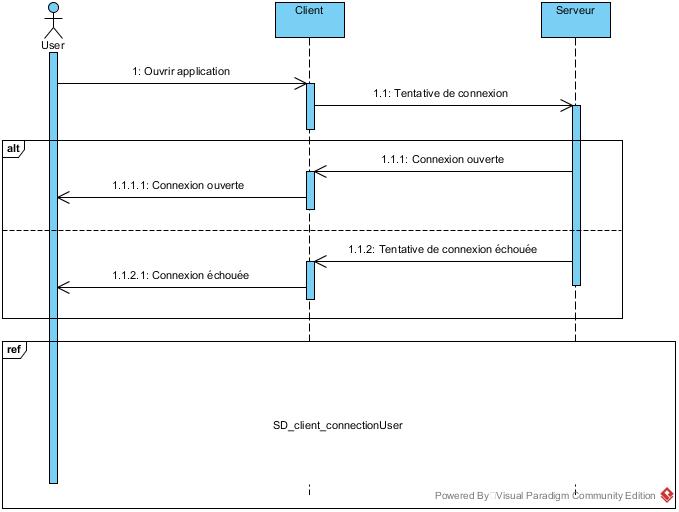
1. Itération 1
   1. Description textuelle des cas d’utilisation

|  |
| --- |
| **18 Connection serveur**  **Acteur principal**: l’utilisateur.  **Objectifs**: permettre à l’application client de se connecter au serveur |
| **Préconditions**: néant.  **Postconditions**: le UC 1 « connection user » démarre  **Scénario nominal**   1. Lors du lancement de l’application, cette dernière tente d’ouvrir une connexion vers le serveur. 2. Une fois la connexion ouverte, la fenêtre de l’application s’ouvre.   **Alternatives**  1.a. L’application n’a pas réussi à ouvrir la connexion vers le serveur.   1. Un message d’erreur « Connexion échouée » s’affiche. 2. Après un temps défini, retour vers l’étape 1 du scénario nominal.   **Exigences supplémentaires**  Une fenêtre indiquant que l’application tente de se connecter au serveur doit s’afficher durant la tentative de connexion. |
| **1 Connection user**  **Acteur principal**: l’utilisateur.  **Objectifs**: permettre à l’utilisateur de se connecter à son compte |
| **Préconditions**: l’application possède une connexion avec le serveur.  **Postconditions**: l’utilisateur est connecté à son compte et a accès aux fonctionnalités de l’application.  **Scénario nominal**   1. L’utilisateur rentre ses données (email et mot de passe) dans la fenêtre et clique sur le bouton « Connexion ». 2. Les données sont envoyées au serveur. 3. Si le serveur envoie une réponse affirmative, l’utilisateur aura accès aux fonctionnalités de l’application.   **Alternatives**  3.a. Le serveur envoie une réponse négative   1. Un message d’erreur « Email ou mot de passe incorrect » s’affiche. 2. Retour à l’étape 1 du scénario nominal.   **Exigences supplémentaires**  Néant. |

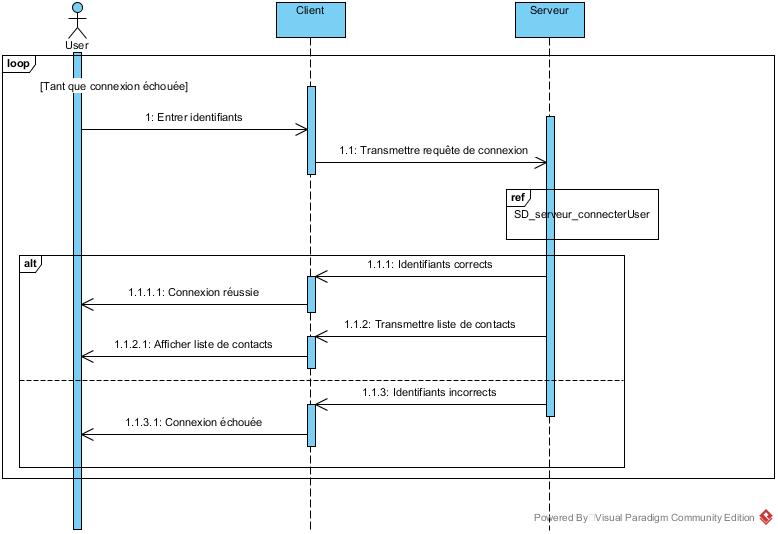
|  |
| --- |
| **19 Deconnection serveur**  **Acteur principal**: l’utilisateur.  **Objectifs**: fermer proprement l’application client |
| **Préconditions**: l’application possède une connexion avec le serveur.  **Postconditions**: la connexion au serveur a été fermée et l’application est fermée  **Scénario nominal**   1. L’utilisateur clique sur le bouton « Quitter ». 2. L’application signale sa fermeture au serveur et se ferme.   **Alternatives**  Néant.  **Exigences supplémentaires**  Néant. |
| **21 Deconnexion**  **Acteur principal**: l’utilisateur.  **Objectifs**: se déconnecter de son compte pour revenir sur la fenêtre de connexion |
| **Préconditions**: l’utilisateur est connecté à son compte.  **Postconditions**: l’utilisateur s’est déconnecté de son compte sur l’appareil  **Scénario nominal**   1. L’utilisateur clique sur le bouton « Se déconnecter ». 2. L’application envoie la demande de déconnexion au serveur. 3. L’application ferme les fenêtres en cours et affiche la fenêtre de connexion.   **Alternatives**  Néant.  **Exigences supplémentaires**  Néant. |
| **2 Créer un compte**  **Acteur principal**: l’utilisateur.  **Objectifs**: créer un compte |
| **Préconditions**: l’application possède une connexion avec le serveur et la fenêtre de connexion est ouverte.  **Postconditions**: l’utilisateur a créé son compte avec succès  **Scénario nominal**   1. L’utilisateur clique sur le bouton « créer un compte ». 2. L’utilisateur rentre les informations demandées et clique sur le bouton « S’inscrire ». 3. L’application envoie la demande au serveur. 4. Si le serveur indique le succès de l’inscription, l’application affiche un message « Inscription effectuée avec succès ».     **Alternatives**  4.a. L’inscription a échoué   1. L’application affiche un message d’erreur indiquant le problème. 2. Retour à l’étape 2 du scénario nominal.   **Exigences supplémentaires**  Néant. |
| **11 Connecter user**  **Acteur principal**: le serveur.  **Objectifs**: connecter l’utilisateur, lui donner accès à sa liste de contacts et notifier ses contacts de sa connexion. |
| **Préconditions**: néant.  **Postconditions**: la base de données sait que l’utilisateur est connecté, de même que ses contacts et le serveur fournis au client sa liste de contacts.  **Scénario nominal**   1. Lorsque le serveur reçoit la demande de connexion, il vérifie si les identifiants sont corrects. 2. S’ils sont corrects, le serveur recherche la liste de contacts de l’utilisateur en base de données. 3. Le serveur envoie la liste de contacts au client. 4. Le serveur ouvre des threads qui seront à l’écoute du client afin de lui fournir les discussions avec les contacts de l’utilisateurs lorsque le client les demandera.     **Alternatives**  2.a. Les identifiants sont incorrects.   1. Le serveur envoie une réponse négative au client.   **Exigences supplémentaires**  Néant. |
| **20 Déconnecter user**  **Acteur principal**: le serveur.  **Objectifs**: fermer les threads utilisés par le client |
| **Préconditions**: l’utilisateur est déjà connecté.  **Postconditions**: les threads utilisés par le client sont fermés.  **Scénario nominal**   1. Le serveur reçoit une demande de déconnexion de la part d’un client. 2. Le serveur ferme tous les threads utilisés par le client. 3. Le serveur vérifie si l’utilisateur du client est connecté via d’autres clients. 4. Si ce n’est pas le cas, le serveur indique à la base de données et aux contacts de l’utilisateur que ce dernier s’est déconnecté.     **Alternatives**  4.a. L’utilisateur est connecté via un autre client.   1. Le serveur arrête le traitement de la déconnexion.   **Exigences supplémentaires**  Néant. |
|  |
| **12 Inscrire user**  **Acteur principal**: le serveur.  **Objectifs**: Enregistrer un utilisateur auprès de la base de données. |
| **Préconditions**: néant.  **Postconditions**: l’utilisateur est enregistré dans la base de données et peut se connecter.  **Scénario nominal**   1. Le serveur reçoit la demande avec les données nécessaires. 2. Le serveur vérifie les données. 3. Le serveur enregistre les données en base de données. 4. Le serveur notifie le client de la réussite de l’opération.   **Alternatives**  4.a. L’utilisateur existe déjà.   1. Le serveur notifie le client de l’échec de l’opération.   **Exigences supplémentaires**  Néant. |
| **13 Vérifier état user**  **Acteur principal**: le serveur.  **Objectifs**: vérifier si l’utilisateur est toujours connecté. |
| **Préconditions**: néant.  **Postconditions**: néant.  **Scénario nominal**   1. Régulièrement, le serveur effectuera un ping vers les clients afin de vérifier si la connexion est toujours disponible. 2. Si la connexion est toujours disponible, le serveur n’effectuera aucune action.   **Alternatives**  2.a. La connexion n’est plus disponible.   1. Le serveur fermera les threads utilisés par le client concerné. 2. Si l’utilisateur du client n’est connecté via aucun autre client, notifier la base de données et les contacts de l’utilisateur de la déconnexion de ce dernier.   **Exigences supplémentaires**  Néant. |

* 1. Diagrammes de séquence

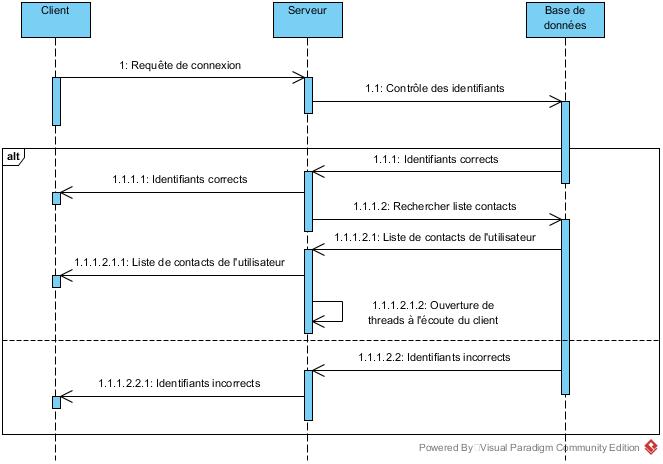
1. Connection Serveur (client)



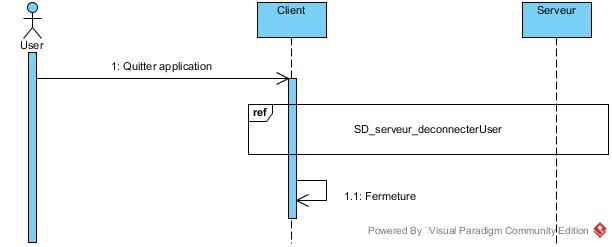
1. Connection user (client)



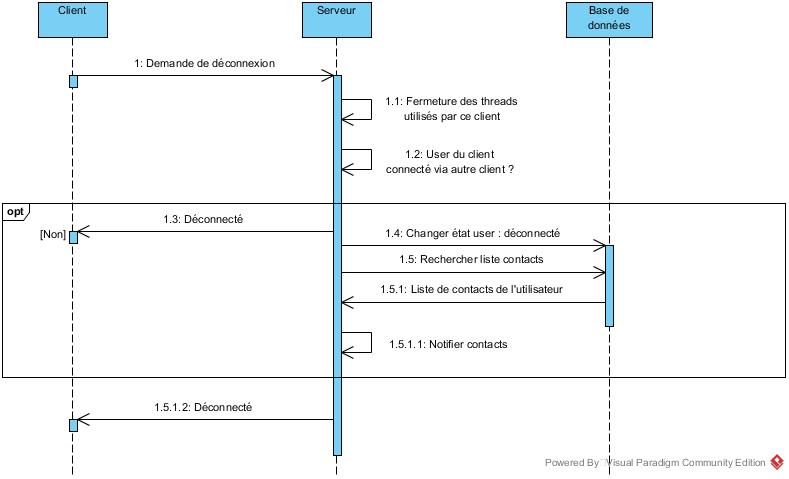
1. Connecter user (serveur)



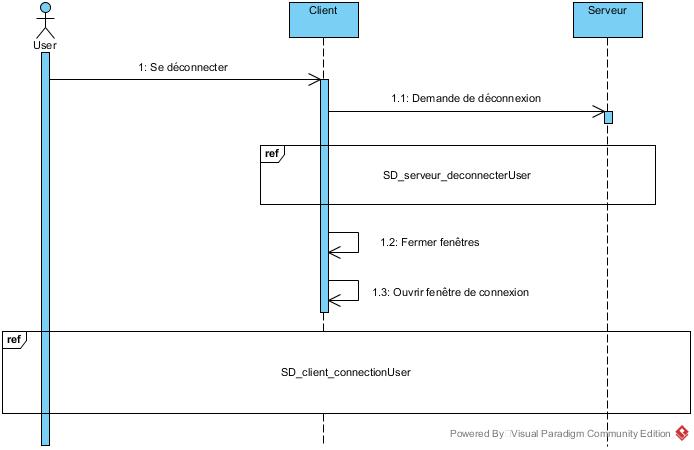
1. Deconnection serveur (client)



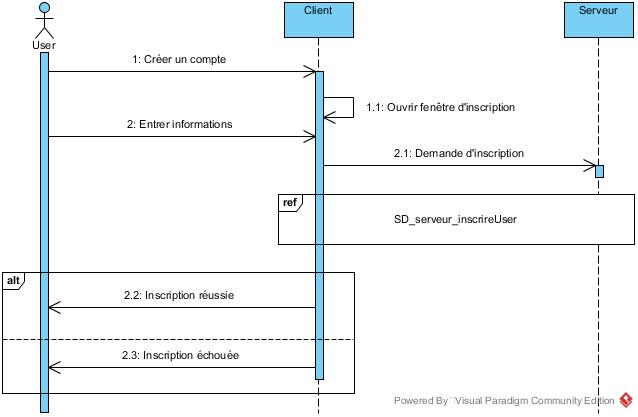
1. Déconnecter user (serveur)



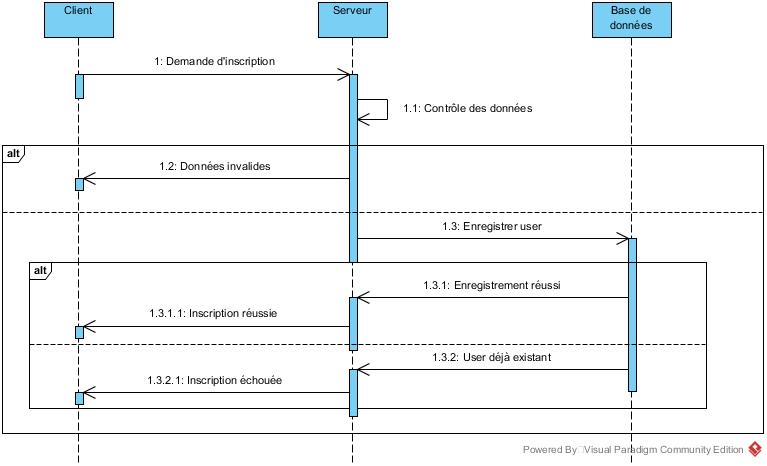
1. Deconnexion (client)



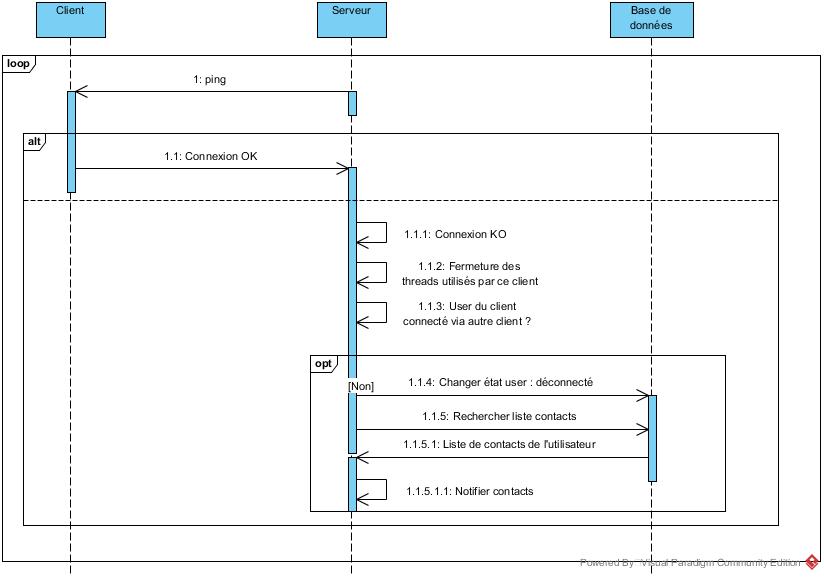
1. Créer un compte (client)



1. Inscrire user (serveur)



1. Vérifier état user (serveur)



* 1. Modèle logique de la base de données relationnelle



* 1. Code SQL de la base de données relationnelle

create database messagerie;

use messagerie;

create table CONFIDENTIALITE (

adresse\_mail varchar(300) not null,

conf\_adresseMail char(1) not null,

conf\_dateNaissance char(1) not null,

conf\_noGsm char(1) not null,

conf\_noFixe char(1) not null,

constraint FKverrouiller\_ID primary key (adresse\_mail));

create table DEMANDE\_CONTACT (

id int not null,

date date not null,

message varchar(3000) not null,

adresse\_mail varchar(300) not null,

constraint ID\_DEMANDE\_CONTACT\_ID primary key (id));

create table USER (

adresse\_mail varchar(300) not null,

password varchar(300) not null,

pseudo char(1) not null,

image char(1) not null,

date\_naissance date not null,

status char(1) not null default 'd',

description varchar(500) not null,

no\_gsm varchar(20) not null,

no\_fixe varchar(20) not null,

id int,

USER\_1\_ varchar(300),

constraint ID\_USER primary key (adresse\_mail));

alter table CONFIDENTIALITE add constraint FKverrouiller\_FK

foreign key (adresse\_mail)

references USER (adresse\_mail);

alter table DEMANDE\_CONTACT add constraint ID\_DEMANDE\_CONTACT\_CHK

check(exists(select \* from USER

where USER.id = id));

alter table DEMANDE\_CONTACT add constraint FKenvoyer

foreign key (adresse\_mail)

references USER (adresse\_mail);

alter table USER add constraint FKrecoit

foreign key (id)

references DEMANDE\_CONTACT (id);

alter table USER add constraint FKcontact

foreign key (USER\_1\_)

references USER (adresse\_mail);

alter table USER

add constraint checkStatus

check (status = 'c' OR status = 'o' OR status = 'n' OR status = 'i' OR status = 'd');

alter table CONFIDENTIALITE

add constraint checkConfAdMail

check (conf\_adresseMail = 'v' OR conf\_adresseMail = 'c' OR conf\_adresseMail = 'p');

alter table CONFIDENTIALITE

add constraint checkDateNaissance

check (conf\_dateNaissance = 'v' OR conf\_dateNaissance = 'c' OR conf\_dateNaissance = 'p');

alter table CONFIDENTIALITE

add constraint checkNoGsm

check (conf\_noGsm = 'v' OR conf\_noGsm = 'c' OR conf\_noGsm = 'p');

alter table CONFIDENTIALITE

add constraint checkNoFixe

check (conf\_noFixe = 'v' OR conf\_noFixe = 'c' OR conf\_noFixe = 'p');

* 1. Code de la base de données Cassandra

create keyspace messagerie with replication = {'class':'SimpleStrategy','replication\_factor':1};

create table messages (

sender text,

receiver text,

message text,

date\_message timeuuid,

primary key((sender),date\_message))

with clustering order by (date\_message desc);

create index ind\_receiver on messagerie.messages(receiver);