

Saé302

Développer des applications communicantes

Cuevas - Raynaud - Mirbey



M. SPIES & M. MABED

24/10/2022 - 28/10/2022



R&T
DUT Réseaux &
Télécommunications
IUT Belfort-Montbéliard

Présentation de la Saé 302

Une **SAé** est une **Situation d'Apprentissage et d'évaluation**.

La « **SAé 302 : Développer des applications communicantes** » a débuté le lundi 24 octobre 2022 et a duré jusqu' au 28 octobre 2022 à l'Institut Universitaire de Technologie Réseaux et Télécommunications à Montbéliard.

Le projet consistait en la réalisation d'une **application** client/serveur en utilisant des fonctions de communication pour créer un protocole applicatif au-dessus de la pile de communication TCP/IP.

Les **objectifs** de cette Saé étaient de nous faire utiliser des **méthodes de gestion de projet**, permettant de mener à bien ce travail. Utiliser le langage **Java** étudié dans la ressource R308 : Consolidation Programmation, afin de comprendre et mettre en place une communication entre client et serveur à l'aide du protocole TCP ou UDP.

Ce bilan de projet va vous présenter nos méthodes de gestion de projets ainsi que la démonstration du modèle client/serveur utilisé.



Figure 1 : Travail coopératif au sein de notre groupe.
















Notre organisation

Lors de cette Saé, nous avons utilisé des **méthodes agiles** vues lors de la ressource Gestion de Projet. Ces dernières nous permettent de faire preuve de **flexibilité, réactivité, transparence et anticipation**. Nous avons donc une équipe plus soudée, apte à travailler en collaboration et avançant plus aisément dans un travail.

Les méthodes agiles permettent de prendre plus de recul sur un travail et faire des mises au point, par exemple avec le client pour arriver à un résultat plus précis par rapport à ce qui est souhaité.

Etant donné que notre groupe était nouveau, il nous a fallu un temps d'adaptation, la mise en place des méthodes suivantes semble alors bénéfiques pour entretenir une bonne cohésion.

Tableau Niko-Niko (l'onomatopée japonaise décrivant un sourire) :

	Lundi 16/10	Mardi 17/10	Mercredi 18/10	Jeudi 19/10	Vendredi 20/10
Kyllian					
Thomas R.					
Thomas M.					

Le tableau **Niko-Niko** se présente comme une pratique agile, méthode servant à partager son état d'esprit du jour. Ce qui nous permet de visualiser plus facilement la motivation ou le bien-être de l'équipe, une chose qui est normalement dur à quantifier. Le but n'est pas de rentrer en concurrence entre nous mais de se comprendre, permet d'améliorer la cohésion dans le groupe.

On peut voir que dans l'ensemble, nous étions heureux durant cette Saé, cependant au fur et à mesure de la semaine le stress a commencé à s'installer au vu de la difficulté du projet.

Notre organisation

Mise en place d'un planning Kanban sur Jira :

Le **Kanban** (terme japonais signifiant "panneau") facilite la collaboration entre les membres de l'équipe agile. C'est aussi une méthode efficace pour surveiller l'amélioration continue du produit ou du service à développer beaucoup utilisé dans des projets de programmation où il faut intégrer de nombreuses fonctionnalités.

Ce planning nous a permis d'obtenir une vue d'ensemble du projet et de l'état des différents livrables que l'on a découpé sous forme de petites tâches. Dans notre cas ce planning ne nous permet pas de tester des user story comme dans des entreprises, mais donc de fragmenter un travail.

On peut y voir plusieurs **états** :

- La tâche n'a pas encore été effectuée => À faire ;
- La tâche est en cours de réalisation => En cours ;
- La tâche reste à valider par un professeur ou par le reste du groupe
=> À valider ;
- La tâche est réalisée => Fini.

=> Ce planning Kanban nous apporte également la motivation et la satisfaction de voir les affiches se déplacer au fur et à mesure de notre projet.

On retrouve **5 principaux avantages** lors de l'utilisation de la méthode Kanban :

- Visualiser le flux des travaux (Workflow).
- Limiter le nombre de tâches en cours afin de ne pas lancer trop de tâches en simultanées et s'éparpiller.
- Gérer le déroulement du travail.
- Établir des règles d'organisation.
- Proposer des actions d'améliorations (principalement dans le monde professionnel).

Pour que cette méthode fonctionne bien, chacun avait accès au document commun sous Jira. Celui-ci nous permettait d'entretenir un rôle actif dans le projet. Nous avons donc rempli rapidement, clairement et en toute transparence le document afin de ne pas créer une incompréhension et un retard.

Notre organisation

Voici le document permettant de finaliser l'interface graphique :

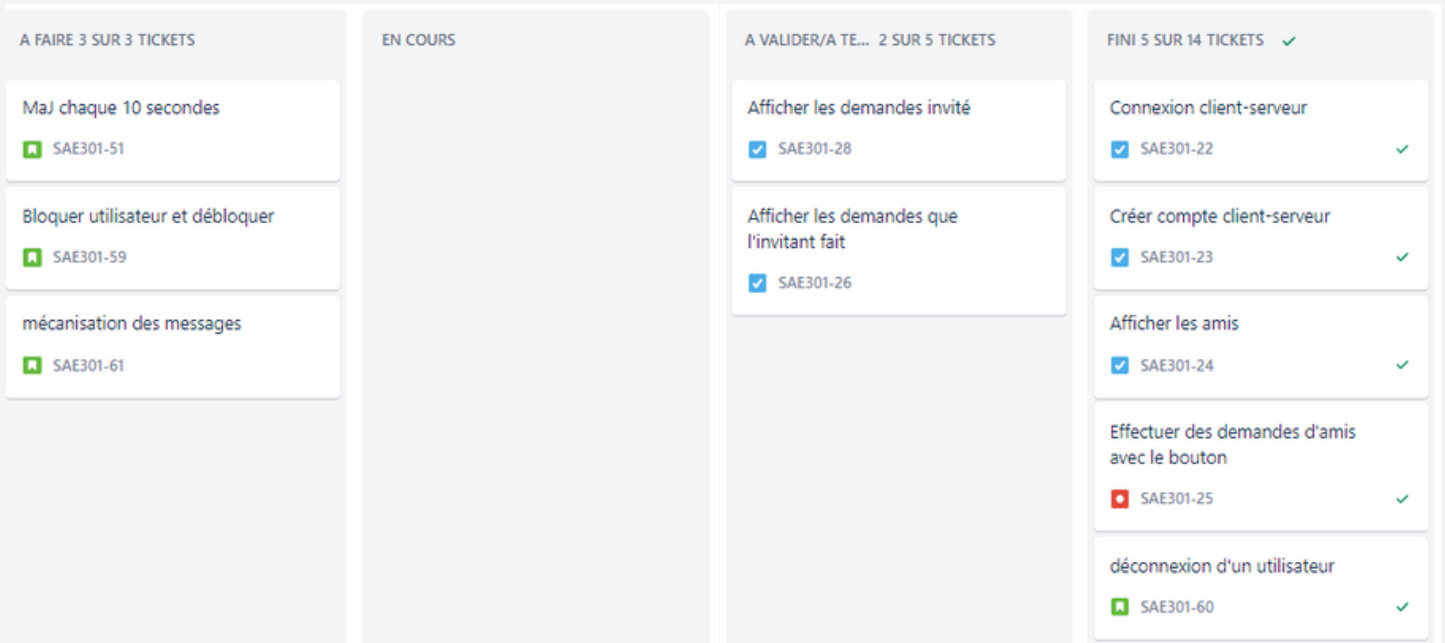


Figure 2 : Sprint, découpage de tâches

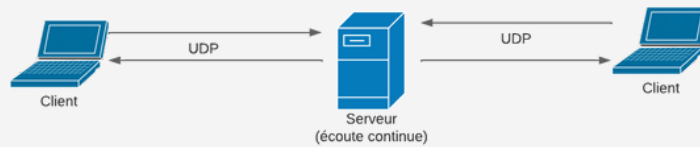
Lors de cette Saé nous avons essayé une répartition des tâches dite "rotative". Notre rôle change au sein d'une même journée.

Un premier s'occupait du serveur (programmation et traitement), un autre du client (graphique et terminal) et le dernier prenait des notes, aidait une personne du groupe, ce qui permettait une meilleure prise de recul notamment dans le domaine de l'algorithmie.

Cette méthode nous est parue logique, sachant que les principaux facteurs faisant échouer des projets sont le fait de n'avoir que des profils en "I" (une seule grande compétence) et d'avoir une mauvaise communication au sein du groupe. Notre but était alors d'être coopératif et polyvalent dans les tâches effectuées.

Le modèle utilisé

Lors de cette Saé, nous avons pu mettre en place un modèle client-serveur pour notre application codée en Java.



Dans une **relation client-serveur**, le client établit une communication avec le serveur sur un réseau (ici local). Le serveur répond ensuite aux requêtes des clients via un service ou des ressources. Le serveur se trouve alors tout le temps en état **d'écoute** alors que le client reste temporairement en écoute (seulement après une requête).

Dans notre cas, les communications entre les clients et le serveur étaient assurées par le protocole **UDP** (User Datagram Protocol), mais nous aurions pu choisir le protocole TCP.

Nous avons préféré le protocole UDP en raison de la **simplicité** de la syntaxe et à la **rapidité** de l'envoi et de la réception des sockets. Nous avons également choisi UDP plutôt que TCP en raison de la manière dont les protocoles communiquent.

UDP offre une transmission de données **sans connexion**, ce qui facilite la gestion du programme et évite les erreurs liées à l'ouverture, au maintien et à la fermeture des connexions.

En début de semaine, afin de bien cerner le sujet principal qui traite des communications, nous avons réalisé, au tableau des **schémas présentant les liens** qui se font entre clients et serveur en fonction du type de message transmis. On pouvait alors avoir les idées au clair pour écrire du code en harmonie avec notre groupe.

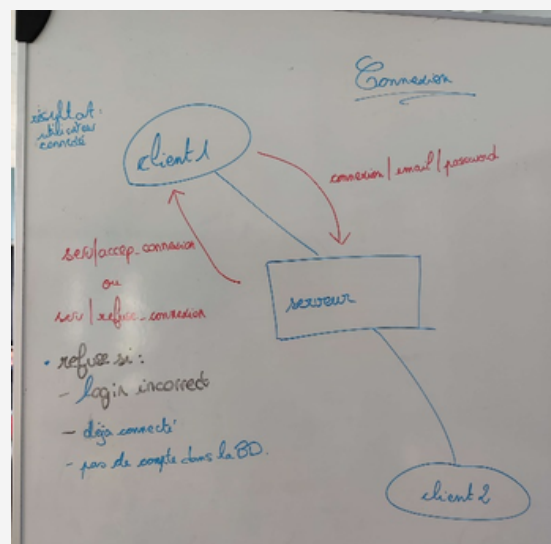
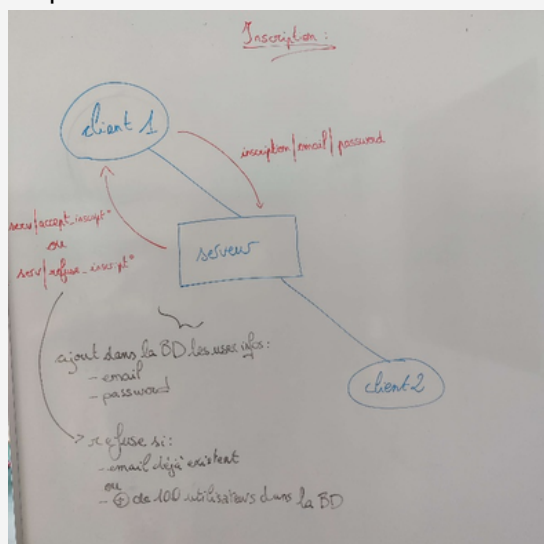


Figure 3 & 4 : Relations client serveur > inscription (gauche) et connexion (droite)

Le modèle utilisé

Pour réaliser notre application, nous avons réalisé **deux classes** qui ont été utilisées par le programme serveur : la classe **Utilisateur** et la classe **Message**.

La classe utilisateur prend en compte un certain nombre d'informations :

- Un nom d'utilisateur unique et un mot de passe permettant à l'utilisateur de s'identifier sur l'application.
- Une liste pour les amis, une pour les demandes d'amis reçues et une pour les utilisateurs bloqués.

Nous avons choisi de **ne pas implémenter** de classe "ami" comme suggérée, dans la mesure où un ami est déjà un utilisateur.

Cependant grâce à la classe utilisateur, nous avons quand même la possibilité de vérifier si notre utilisateur est ami avec un autre, cela sera utile dans le cadre de l'envoi des messages et la gestion de ces derniers.

Un utilisateur peut dialoguer avec ses amis, pour cela, nous avons créé des méthodes permettant d'ajouter ou de lire une liste de messages.

La classe message est **étroitement liée** à la classe utilisateur. Celle-ci leur permet de pouvoir échanger. Pour créer un message, on doit donner deux utilisateurs, un utilisateur source et un utilisateur destinataire, un motif, un objet (que nous avons décidé de ne pas utiliser comme dans les **réseaux sociaux modernes**) et les données. Nous pouvons récupérer les différentes informations d'un message comme son destinataire ou son contenu en utilisant les méthodes que nous avons défini.

Après avoir réalisé les différentes classes demandées, nous avons mis en place les premières parties de nos codes serveur et client.

Le serveur gère une liste d'utilisateurs, et est **identifié** par un **port** et une **IP**.

Nous avons défini que le serveur serait constamment en **écoute** et se contenterait de répondre aux clients, c'est sur ce principe que repose le modèle client-serveur en UDP.

En fonction du message reçu, le serveur va **découper** chaque partie afin de réaliser le traitement adéquat. C'est dans cette partie que le serveur vérifie également si les différentes contraintes sont respectées ou non, par exemple si la taille de la liste d'amis d'un utilisateur ne dépasse pas la taille maximale ou si une inscription d'utilisateur n'a pas déjà été faite. Nous avons utilisé une norme de nommage des requêtes **très précise**. Voici deux exemples :

```
envoi ( msg: "inscription@hakim@mdp2" );  
System.out.println ( x: recu ( ) );  
  
envoi ( msg: "connexion@francois@mdp1" );  
System.out.println ( x: recu ( ) );
```

```
envoi ("update@" + LoginPageGui.myName ( ) );  
//exemple de resultat : update@thomas
```

Figure 5 & 6 : Scénario de test avec inscription et connexion d'utilisateur (gauche) et requête de mise à jour des listes (droite)

Le modèle utilisé

Pour le **client**, nous avons défini qu'une fois un message envoyé, le client attendrait une réponse. Le client demande également des **actualisations** régulières (chaque 10 secondes) au serveur pour obtenir les dernières versions de sa liste de ses messages ou la liste de ses amis notamment. Nous avons réalisé **deux versions principales** du client. Un client en mode text-only et un client graphique.

- Le client **text-only** fonctionne à l'aide d'une boucle qui demande à l'utilisateur de rentrer des informations ou des numéros permettant d'identifier au mieux ce qu'il souhaite faire en format terminal.
- Il existe aussi le client sous forme **d'interface graphique** avec 3 types de pages : la page de connexion, le menu et la page d'envoi de message. Dans notre cas un scénario (qui réalise la connexion d'utilisateur, demandes d'amis ainsi que l'acceptation, l'envoi de messages) a lieu. Ce qui nous permet de cibler plus rapidement d'éventuels **problèmes** lors de tests.

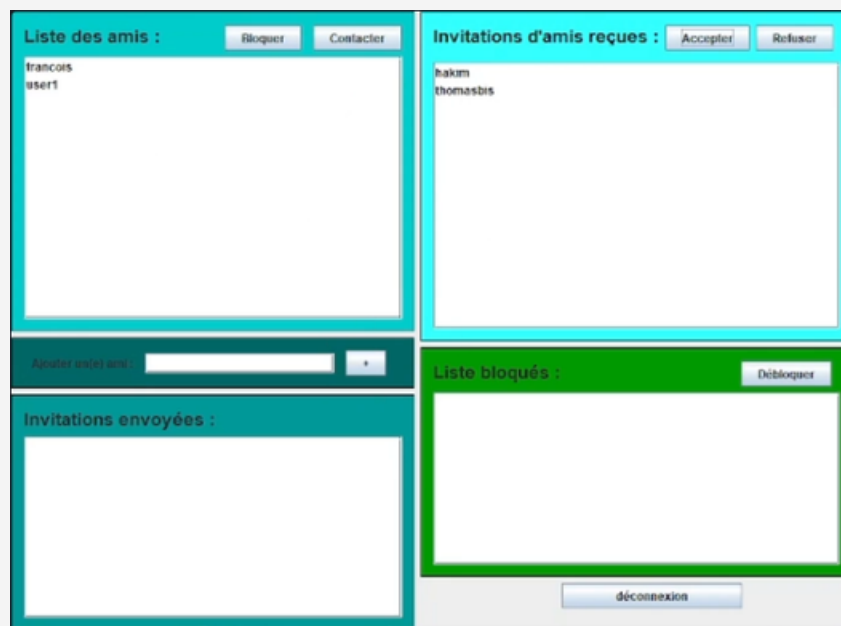


Figure 7 : Menu de l'application

Nous avons pensé à plusieurs **pistes d'améliorations** que nous aurions aimé mettre en place si nous avions eu davantage de temps. Nous aurions aimé réaliser un **tableau associatif** contenant pour chaque utilisateur les messages reçus par utilisateurs, mais nous n'avons pas pu le mettre en place dans le temps imparti.

```
Liste = {User1: [{User2:"Message1", "Message2"}, {User3:"Message1"}], User2: [{User1:"Message1", "Message2"}]}
```

La liste ci-dessus est utilisée à titre d'exemple mais ne respecte ni la syntaxe **Java** (Syntaxe du langage du projet) ni la syntaxe de **Python** (Syntaxe du langage de l'exemple), elle sert à illustrer à quoi ressemblerait notre liste. La gestion des données était plus complexe que prévu, pour nous simplifier la tâche nous avons décidé de ne pas utiliser un stockage de ce type.

Le modèle utilisé

Nous avons aussi voulu mettre en place un **hash** pour chaque paramètre des sockets envoyés par le serveur et le client, permettant de sécuriser l'entièreté des échanges en masquant les identifiants et les types de requêtes utilisés. Nous aurions alors reçu des requêtes de ce type :

```
Recu du client : $2y$10$kRn3xf5Ku9Dho.iAaTL@$2y$10$oa3kX/XA1rZj.Jf1BP@$2y$10$07j4ukE/LMoN  
// = connexion@thomas@test
```

Figure 8 : Hash d'une requête reçue par le serveur

On retrouve bien les @ permettant de **séparer** chaque partie de la requête. On voit également que chaque partie est hashée ne permettant pas à un attaquant de trouver l'action demandée par un utilisateur. Cela permet également de protéger les informations sensibles comme les identifiants ou les mots de passe des utilisateurs. Au lieu de comparer du texte, le serveur comparerait alors des hashes.

Finalement, nous voulions également mettre en place une **interface sur Android Studio** en parallèle des deux interfaces déjà existantes. Nous aurions alors fait un **portage** de la version Netbeans sur Android Studio



Version mises en place à la fin de la Saé



Version espérées initialement à la fin de la Saé

Au vu du travail à réaliser, nous avons décidé de ne pas **prioriser** ces propositions mais nous n'avons pas pour autant eu la possibilité de les mettre en place dans le temps imparti.

Nous avons néanmoins atteint la **majorité** des objectifs que nous avions priorisés.

Conclusion

Lors de cette SAé, nous avons dû utiliser nos **connaissances** en programmation Java, gestion de projet, communications réseaux et Anglais (lecture de documentation).

Nous avons **rencontré certains problèmes** tels que :

- La mise en place de l'envoi des données par le client, nous avons alors utilisé des fonctions comme dans le code qui nous était présenté.
- La réception et traitement d'un message au niveau du serveur. Il nous a fallu mettre en place l'écoute en continue avec une condition "while(true)", et mettre un algorithme de traitement sélectionnant le premier mot de la requête.
- L'affichage des messages dans l'interface graphique qui a été difficile à mettre en place. Pour y remédier il a fallu moduler la réponse du serveur afin de réaliser un traitement spécifique au niveau de la page du client graphique à l'aide de différents caractères de coupure "@" puis "/".
- L'actualisation de la page toutes les 10 secondes à laquelle nous n'avions pas trouvé de solution en autonomie, malgré la recherche sur des forums et de la documentation. Il nous a fallu récupérer le code déposé sur Github pour comprendre la fonction que nous devions utiliser.

Lors de cette SAé, nous avons pu aider d'autres groupes ce qui a favorisé **l'entraide** et nous a permis d'avoir un œil nouveau sur un problème rencontré. Pour améliorer le fonctionnement au sein de notre groupe nous pensons qu'**un peu d'expérience ferait l'affaire**, car c'était la première fois que nous étions amenés à travailler tous les trois ensemble mais la cohésion et le travail était présent. Cette SAé nous aura permis de travailler efficacement en équipe.

Au fil de la semaine sortir de notre code était de plus en plus compliqué, nous n'avons pas su prendre la prise de recul nécessaire pour pouvoir comprendre plus facilement ce qui avait déjà été fait. Nous avons aussi essayé un nouveau **"format"** pour nous. Travailler plus en **autonomie** qu'à notre habitude et favoriser nos recherches et connaissances acquises. Cela nous a permis de davantage **partager nos connaissances et nos expériences** en programmation, ce qui a été bénéfique à tous les membres du groupe pour s'améliorer.

C'est également ce qui nous a permis de **contourner des problèmes** en cherchant des informations par nous-même. Ceci nous permet de capitaliser des connaissances qui pourront être utilisées dans les projets à venir ainsi que dans notre vie professionnelle.