Spécifications

Dylan LECOMTE & Antoine AUGAY - DII5

Description générale

Contexte de réalisation

Dans le cadre de l'enseignement "Outil formel", nous devons réaliser une application avec des contraintes exigés par les enseignants. Cette application peut être un logiciel, un site internet, une application mobile … c'est au choix des étudiants (travaillant en binôme), mais cette application doit mettre en oeuvre au minimum :

- Un processus de paiement
- De la sécurité
- Des tests

Il ne s'agit pas de réaliser une application incluant énormément de fonctionnalités et d'originalités, mais plutôt de mettre en oeuvre les notions vues en CM, à savoir les tests, les différentes notions autour du paiement … à une application lambda.

Cahier des charges

Pour ce projet, nous avons décidé de développer un jeu d'argent via deux applications :

- Un client (destiné à l'utilisateur voulant jouer)
- Un serveur (gérant les différents utilisateurs)

Le système est simple et fonctionne de la façon suivante : l'utilisateur lance son client et tente de se connecter au serveur avec ses identifiants (il peut créer un compte). Une fois connecté au serveur, en fonction du solde qu'il a sur son compte (de jeu, pas bancaire) il a la possibilité de miser sur une partie (jeu de hasard avec deux résultats possibles : victoire/défaite). En fonction du résultat de la partie, son solde est mis à jour à la hausse ou à la baisse. Le client aura la possibilité de créditer son compte en effectuant une opération de recharge en payant avec une carte de crédit. Il faudra mettre en oeuvre des mécanismes pour vérifier les données entrées et leur cohérence.

Côté serveur, le système doit pouvoir gérer plusieurs clients en même temps, et interagir avec une base de données (SQLite) pour stocker les informations. Il faudra définir un format de trames pour que le client et le serveur puissent interagir et sécuriser ces messages. De même pour la base de données, les données sensibles (mot de passe) devront être sécurisées.
Ce projet, un peu ambitieux, nous fera mettre en oeuvre :

- Un SVN
- Du databinding
- Un environnement multi-clients/serveur
- Interaction avec une base de données
- Gestion de trames sur réseau (définir trame, chiffrement …)
- Hashage de mot de passe en base de données
- Gestion de paiements
- Gestion de threads
- Une multitude de tests

Nous développerons ces applications sur Visual Studio 2015, en C#. Nous avons décidé d'utiliser cet environnement de développement car nous n'avons pas eu l'occasion de travailler dessus précédemment alors qu'il s'agit d'un outil très utilisé dans l'industrie, c'était l'occasion pour nous de le prendre en main.
Pour le SVN, nous utilisons Git, qui est largement répandu et fonctionne très bien avec Visual Studio

Contraintes

Les contraintes imposées par les enseignants sont les suivantes :

- Utilisation d'un gestionnaire de version.
- Mise en place d'un plan de tests, faisant intervenir au minimum :
    - Des tests unitaires, si possible automatisés
    - Des tests fonctionnels, si possible automatisés
- Sécurisation des données sensibles (mots de passe, données bancaires…)
- Un processus de paiement sécurisé

Fonctionnalités

Serveur

L'application serveur devra proposer les fonctionnalités suivantes :

- Stockage, lecture et écriture de données utilisateurs (login, mot de passe et solde) dans une base de données SQLite
- Vérification de l'unicité des logins.
- Vérification des logins/mot de passe lors de la connexion d'un client
- Visualisation des clients connectés et de leur solde en temps réels
- Traitement des demandes clients
    - Mise à jour du solde
    - Demande de solde
    - Ajout d'argent sur le compte
    - ...

Client

L'application client devra proposer les fonctionnalités suivantes :

- Une page d'authentification où l'utilisateur devra renseigner son login et son mot de passe
- Une page de création d'un nouveau compte
- Un moyen de créditer le solde de son compte avec une carte bleu
- La page de jeu où l'utilisateur pourra miser une somme d'argent
- L'affichage du solde en temps réel

Cette application devra s'assurer des points suivants :

- L'utilisateur ne peut pas miser plus que son solde
- L'utilisateur ne peut pas perdre plus que ce qu'il a misé
- L'utilisateur ne peut pas jouer pendant une transaction
- Même si l'utilisateur quitte brusquement l'application, son solde est tout de même déduit de sa mise

Sécurité

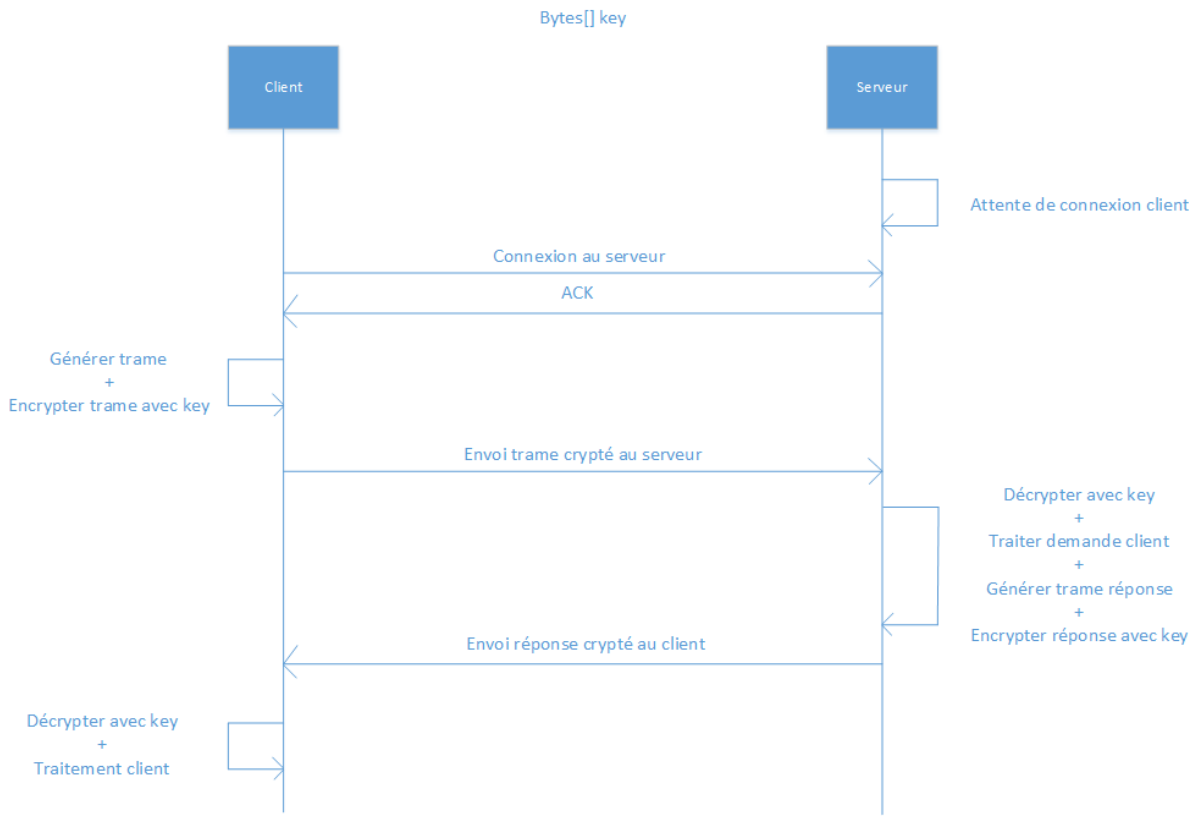
Mot de passe en base de données

Les mots de passe ne devront pas être stockés en base de données, nous utiliserons donc un hash du mot de passe, c'est-à-dire transformer, via des procédés mathématiques complexes, une suite de caractères sans qu'il soit possible de faire l'opération inverse. Cette méthode est très efficace, car l'attaquant ne trouvera dans notre base de données que des infos transformées.
Pour cela, nous utiliserons l'algorithme MD5, même si aujourd'hui il n'est plus considéré comme fiable, le but ici est de mettre en place un système sécurisé et non d'avoir le système le plus sécurisé possible.

Sécuriser les trames réseau

Nous devons développer deux applications dont la communication se fait en TCP. Nous devons donc définir un format de trame pour que nos applications puissent interagir, mais aussi mettre en place un chiffrement pour ces trames. En effet, si une personne mal intentionnée tente d'intercepter les trames réseau, il ne doit pas pouvoir distinguer en clair des informations dans les trames.

Voici un schéma résumant un échange de trames cryptées client-serveur pour nos applications :


Bytes[] key
Client
Serveur
Attente de connexion client
Connexion au serveur
ACK
Générer trame
+
Encrypter trame avec key
Envoi trame crypté au serveur
Décrypter avec key
+
Traiter demande client
+
Générer trame réponse
+
Encrypter réponse avec key
Envoi réponse crypté au client
Décrypter avec key
+
Traitement client

Tests

Plan de test

Pour nos tests, nous avons décidé de nous limiter aux fonctionnalités offertes par Visual Studio. Après avoir regardé ce que propose la solution, nous avons déduit qu'il suffisait largement à ce que nous souhaitions faire. Nous allons donc ajouter à nos projets, deux autres projets :

- Un projet de test unitaire
- Un projet de test fonctionnel

Une fois ajoutés et paramétrés, nous réaliserons nos tests de la façon suivante :

- A chaque fonction sera associée à un test unitaire
- Chaque interface utilisateur sera associée un test fonctionnel
- Chaque nouvelle version sur la branche master de notre SVN devra satisfaire l'ensemble des tests citées ci-dessus

Unitaires

Nous allons utiliser l'outil de test de Visual Studio. Il faut ajouter à notre solution un projet de test unitaire, tous les tests unitaires se trouveront dans celui-ci. Il est possible d'exécuter l'ensemble des tests unitaires et d'avoir l'ensemble des résultats.
Nous utiliserons également la librairies Moq disponible pour visual studio afin de tester certaines de nos méthodes en les isolant du reste du système. Cela nous sera utile notamment pour simuler la base de données.

Fonctionnels

Visual Studio propose un système de tests d'interface utilisateur automatisé. Pour cela nous allons ajouter un autre projet avec Visual Studio : un projet de test d'interface utilisateur. A l'aide d'un recorder, on enregistre une série d'actions sur l'interface de l'application que l'on pourra rejouer. Ainsi, à chaque nouvelle version logicielle à générer, on applique des tests fonctionnels répondants à des scénarios fixés lors du paramétrage des tests.