Rapport de projet Outils Formels

Ce document explique ce que nous avons réalisé pour notre projet d'outil formel. Ce rapport contient 3 parties :

- Une présentation des livrables
- Une présentation des fonctionnalités développées
- Une présentation des tests réalisés

Avant de lire ce document, il est important d'avoir lu les spécifications du projets disponibles à la racine de l'archive.

Nous avons rédigé une doc d'installation du projet pour pouvoir :

- Utiliser l'application
- Ouvrir le projet sous Visual Studio

Cette doc est aussi disponible à la racine de l'archive

Table des matières

1. Livrables

Le livrable du projet est rendu sous forme d'archive est contient les éléments suivants :

- Rapport.pdf : il s'agit de ce document, détaillant ce qui a été fait
- Spécification.pdf : un cahier des spécifications décrivant notre projet, ce que nous avions prévu de réaliser
- Installation.pdf : un document expliquant comment utiliser l'application et le projet
- Client : répertoire contenant l'application du client. L'exécutable se trouve à cet emplacement : Client\bin\Debug\Client.exe
- Serveur : répertoire contenant l'application du serveur. L'exécutable se trouve à cet emplacement : Server\bin\Debug\Server.exe
- Projet : répertoire contenant l'archive de notre projet. Pour ouvrir le projet, le fichier se trouve à l'emplacement : Projet\ClientServerOF\ClientServerOF.sln

2. Fonctionnalités

2.1 Gestion utilisateurs

2.1.1 Création d'un nouveau compte

La création d'un nouveau compte se fait dans la fenêtre de login. Il faut sélectionner "Nouvel utilisateur", renseigner login + mot de passe. Les contraintes développées sont les suivantes :

- Le login ne doit pas exister en base de données. Si l'utilisateur a renseigné un login existant pour la création, le serveur renverra un message pour avertir que le login est déjà pris.
- Le mot de passe doit contenir en 8 et 15 caractères, avoir au moins une majuscule et un chiffre

Lorsqu'un utilisateur crée un compte, il se connecte automatiquement

2.1.2 Connexion

Pour se connecter, il faut renseigner Login + mot de passe sur la fenêtre de login. Si l'authentification se passe bien, la fenêtre de jeu se lance. Sinon :

- Le login est inconnu (message alertant l'utilisateur)
- Le mot de passe est incorrect (message alertant l'utilisateur)

Si c'est la connexion au serveur (serveur pas démarré) qui échoue et non les informations de connexion, on informe aussi l'utilisateur avec un message

2.1.3 Stockage des informations en base de données

Pour notre base de données, nous avons décidé d'utiliser une base SQLite. Celle-ci est légère, portable, parfait pour notre application.

Notre base de données est simple puisqu'elle ne contient qu'une seule table : User. Cette table à 4 colonnes :

- id : identifiant automatique de l'utilisateur
- username : login de l'utilisateur
- password : mot de passe de l'utilisateur
- balance : montant disponible pour cet utilisateur

Nous mémorisons ces informations pour chaque utilisateur de la base de données. Le fichier de base de données est stocké à la racine de l'exécutable du serveur (fichier database.db).

2.1.4 Hashage du mot de passe

Les mots de passe ne sont pas stocké en clair dans la base de données. Nous utilisons une fonction de dérivation de clé, PBKDF2. Cette méthode met en œuvre plusieurs mécanismes. Le premier est le salage, celui-ci consiste à ajouter une information à la donnée à stockée afin d'empêcher que deux informations identiques conduisent à la même empreinte. Le second est la répétition du chiffrement un certain nombre de fois (dans notre cas 10000 fois). Ces deux points réduisent les possibilités de casser le mot de passe à la force brute.

2.2 Gestion de la communication client-serveur

Les messages entre le client et le serveur transitent sur le réseau. Afin d'éviter que la récupération de données avec un analyseur de trames soit possible, nous avons chiffré l'ensemble des trames. Nous utilisons l'algorithme de chiffrement symétrique par bloc TripleDES, la clé est hashée avec l'algorithme MD5. Ces algorithmes sont disponibles dans le package de chiffrement de Visual Studio (System.Security.Cryptography).

2.3 Ajout d'agent sur le solde

Afin d'ajouter de l'argent sur son compte, l'utilisateur doit "payer" avec une carte de crédit virtuelle.

Nous contrôlons les informations rentrées par l'utilisateur à deux niveaux :

- Structure de la données (taille de la chaîne, vérification si bien numérique etc)
- La cohérence de l'information (pour numéro de carte et date d'expiration) afin de savoir si la date et le numéro de carte bleu sont valides.

Pour payer, l'utilisateur doit renseigner 4 champs :

- Numéro de carte de crédit
- Date d'expiration
- Cryptogramme
- Montant

Pour la carte de crédit, nous vérifions la structure de la données en testant s'il s'agit bien d'une suite de caractères numériques de 16 caractères. Pour vérifier la cohérence du numéro de carte renseigné, nous avons utilisé l'algorithme de Luhn (mod10). Il s'agit d'un standard de sécurité PCI (Payment Card Industry)

Pour la date d'expiration, nous vérifions la structure de la données en vérifiant que le format : MM/AAAA est respectée. Pour vérifier la cohérence de la date d'expiration, nous avons vérifié que le mois est bien compris entre 1 et 12 et que la date n'est pas ultérieure à la date du jour et pas supérieure à la date du jour + 2 ans (car une carte de crédit n'est valable que deux ans).

Pour le cryptogramme, nous vérifions la structure de la données en testant s'il s'agit bien d'une suite de caractères numériques de 3 caractères.

Pour le montant, nous vérifions la structure de la données en testant s'il s'agit bien d'une suite de caractères numériques. Nous vérifions la cohérence en regardant si le montant n'est pas supérieur à 1000 euros, somme maximum fixé par notre application.

2.4 Affichage des clients connectés sur le serveur

L'interface du serveur affiche en temps réel les clients connectés et leurs soldes. Celle permet d'avoir un vu d'ensemble sur les utilisateurs en cours. Cette affichage mise à jour grâce à des message entre les threads côté serveur. Les threads de la gestion des clients envoi l'évolution de l'état du client à chaque changement.

2.5 Jeu d'argent

2.5.1 Vérification des montants joués

Pour jouer, l'utilisateur doit renseigner un montant. Ce montant est contrôlé pour s'assurer qu'il ne puisse pas miser une somme absurde. En effet, nous vérifions que l'utilisateur a bien l'argent qu'il mise dans sa cagnotte (celle-ci est affichée sur l'IHM) pour pouvoir parier.

Une méthode est appelée périodiquement pour vérifier si cette contrainte est respectée et grise ou non le bouton pour parier en conséquence.

2.5.2 Sécurisation du jeu

Nous avons implémenté notre solution afin que l'utilisateur ne puisse pas quitter brusquement l'application s'il voit qu'il a perdu une partie, et donc espérer ne pas être débité. Pour cela, nous envoyons une première trame au serveur lors du pari pour débiter l'utilisateur de la somme pariée. Et seulement lorsque la partie sera terminée côté client, nous renvoyons une trame au serveur pour mettre à jour l'argent de l'utilisateur s'il a gagné (pas la peine de renvoyer de trame s'il a perdu car il a déjà été débité).

3. Tests

3.1 Analyseur de code

3.1.1 SonarLint

SonarLint est un plugin officiel de SonarSource permettant l'analyse syntaxique du code source directement dans l'IDE en utilisant les règles d'analyse de SonarQube. Il nous a été présenté lors des cours de test, nous avons donc décidé de l'utiliser.

A la manière de CheckStyle, SonarLint affiche les problèmes directement dans le code en les surlignant/soulignant et en les listant dans un panel dédié. L'installation de SonarLint se fait très rapide en passant par les gestionnaires de plugins de Visual Studio et ne nécessite aucune configuration particulière pour fonctionner.

Nous l'avons utilisé dès le début de notre projet et cela nous a permis de relever des incohérences et donc d'optimiser notre code.

3.1.2 Analyseur de code VS

Nous avons aussi utilisé l'analyseur de code intégré en natif de Visual Studio. Il fonctionne exactement de la même façon que SonarLint, mais permet de relever corrections supplémentaires. Nous n'avons pas pris en compte toutes les erreurs relevées par l'analyseur de code car il y'en a beaucoup et elles sont parfois superflues.

3.2 Tests unitaires

Les tests unitaires ont été mis en place seulement du côté du serveur pour des raisons de temps. Ces tests sont disponibles dans le projet ServerTests de la solution. Nous avons utilisé les outils que propose Visual Studio pour effectuer ces tests. Ces tests couvrent la gestion de la base de données, le chiffrement et la gestion des trames.

Dans chaque test unitaire un pattern à était conservé, il est le suivant :

- Préparation du test : Initialisation des variables, instanciation de la classe testés, définition du résultat attendu.
- Action : exécution de la fonction à tester
- Vérification : comparaison des résultats
- Remise en état (facultatif) : annulation de l'effet de la fonction testée

Cette structure permet d'être plus efficace dans la mise en place de ces tests.

Ces tests nous ont permis de relever quelques failles et sont utiles lors de la mise à jour de l'application, il n'y a pas besoin de retester toutes les fonctions mais simplement de rejouer les tests unitaires (sur Visual Studio, onglet Test -> Exécuter -> Exécuter tous les tests). Nous n'avons pas créé

de test unitaire pour toutes nos fonctions car cela aurait été beaucoup trop long (nous avons beaucoup de code dans le projet). De plus, le but ici était d'apprendre à mettre en place des tests unitaires et non de tout tester.

3.3 Tests fonctionnels

Pour les tests fonctionnels, nous avons utilisé l'automatisation de l'interface utilisateur de Visual Studio. Il s'agit d'un outil disponible uniquement dans la version Entreprise de Visual Studio et permet de créer des tests automatisés qui vérifient l'interface utilisateur d'une application. Ces tests sont disponibles dans le projet TestsIHM de la solution, dans la classe CodeUITest1, méthode CodedUITestMethod1.

Pour tester l'outil, nous avons créé 3 scénarios de test :

- Un de connexion du client au serveur (login + mdp corrects)
- Un d'une partie jouée
- Un d'ajout d'argent

Ces scénarios ont été très facile à mettre en place et nous ont permis de gagner beaucoup de temps lorsque nous voulions tester les fonctionnalités. Par contre, le scénario ne fonctionne pas pour la connexion. Lorsque nous avons créé le scénario, nous avons rentré login + mdp, nous nous sommes connecté au serveur et cela a marché. Or lorsque l'on rejoue le scénario, le serveur renvoie que le mot de passe n'est pas correct. Nous pensons que cela est dû à l'utilisation d'un champ PasswordBox, qui est un champ assez particulier. Lors du développement, nous avons eu du mal à récupérer le mot de passe de ce champ, nous avons dû mettre en œuvre des interactions (EventTrigger) pour y arriver.