Université du Québec à Rimouski

Campus de Lévis

Travail pratique #1

Par

Goulet Bastien et Larrivée Cédrick

Travail présenté Martin Arsenault.

Dans le cadre du cours sécurité informatique.

INF36207-MS

01–2023

Table des matières

[Algorithme 3](#_Toc125906336)

[Algorithme en detail 3](#_Toc125906337)

[Générer le jeton sur le client 3](#_Toc125906338)

[Validation du jeton sur le serveur 4](#_Toc125906339)

[Application client 4](#_Toc125906340)

[Application serveur 5](#_Toc125906341)

[Résultats 5](#_Toc125906342)

[Conclusion 5](#_Toc125906343)

[Annexes 6](#_Toc125906344)

# Algorithme

## Algorithme en détail

Pour éviter d’écrire du code redondant, nous avons décidé d’implémenter nos deux applications avec l’architecture clean. Cela nous a permis d’utiliser le même noyau pour nos deux applications. La génération des jetons est donc la même sur les deux applications. Pour créer le jeton, nous avons décidé de nous baser sur le temps écoulé depuis le premier janvier 1970. L’utilisation de la date précise et de la réutilisation du noyau de nos applications assure la synchronisation du serveur et du client. Pour générer le jeton, nous récupérons le temps écoulé, pour ensuite le hacher avec une clé secrète. Finalement, la fonction ComputeOpt va construire un entier à partir de notre hash. L’opérateur << est utilisé pour garantir que seulement 8 bits seront décalés pour ajouter à notre variable qui sera notre jeton.

## Générer le jeton sur le client

Pour générer un jeton pseudo-aléatoire sur notre application client, nous devons commencer par initialiser les divers services requis avec l’aide d’interface. Les services counter, hash et otp nous seront utiles. Quand ce dernier sera initialisé, la méthode ComputeNextOtp sera appelée, le hash génèrera notre tableau d’octets qui sera passé à ComputeOpt. Ce dernier créera notre jeton aléatoire.

## Validation du jeton sur le serveur

Pour valider le jeton sur le serveur ce dernier doit être généré de la même manière que sur notre application client puisque les deux applications utilisent le même noyau. Encore une fois durant l’initialisation de la fenêtre, la méthode ComputeNextOtp sera appelée, le hash que nous avons passé au constructeur génèrera notre tableau d’octets qui sera passé à la fonction ComputeOpt qui comme décrit plus haut créera notre jeton aléatoire. Pour valider le jeton du client, ce dernier devra l’entrer dans le champ prévu à cet effet et ensuite appuyer sur le bouton Login. Cela va enclencher la validation du jeton qui va comme suit : le champ vérifie la longueur du jeton. S’il n’est pas égal à six caractères, l’accès sera refusé. Nous allons tenter de convertir la chaine de caractère qui représente le jeton en un entier encore une fois, si cela ne fonctionne pas l’accès sera refusé. Finalement, une fonction du service otp sera appelée. Elle fera la validation du jeton de façon très simple. Pour ce faire, elle va accéder au jeton généré localement pour le comparer au jeton qui lui est passé en paramètre. La fonction retourne vraie si les deux jetons sont identiques. L’accès sera alors confirmé. Elle retournera faux s’ils ne le sont pas. Dans quel cas, l’accès sera refusé.

# Application client

L’application client est relativement simple d’utilisation. Il suffit de l’ouvrir et vous pourrez alors voir le temps restant avant la génération du prochain jeton. Toutes les 60 secondes, le compteur sera réinitialisé et un nouveau jeton sera généré. Sous le compteur se trouve le jeton actuel. Pour voir l’interface allez à l’annexe image 1.

# Application serveur

Tout comme l’application client, notre application serveur est aussi très simple d’utilisation. Un champ est présent pour entrer votre jeton qui doit être de 6 chiffres. Si le nombre de chiffres entré est plus long ou plus court, ou si des caractères autres que des chiffres sont présents alors l’accès sera refusé lors de l’utilisation du bouton Login qui permet de valider le jeton entré dans le champ. Finalement, le dernier jeton valide sera affiché sous le bouton de connexion.

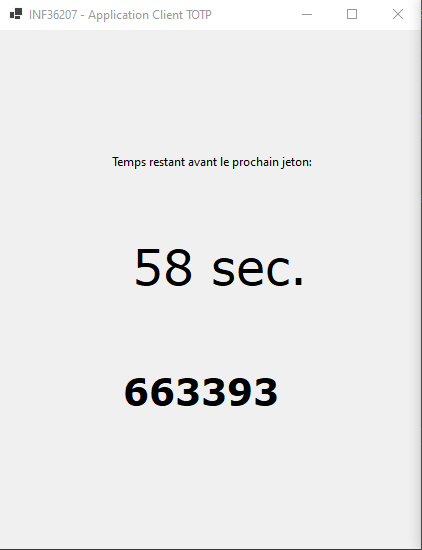
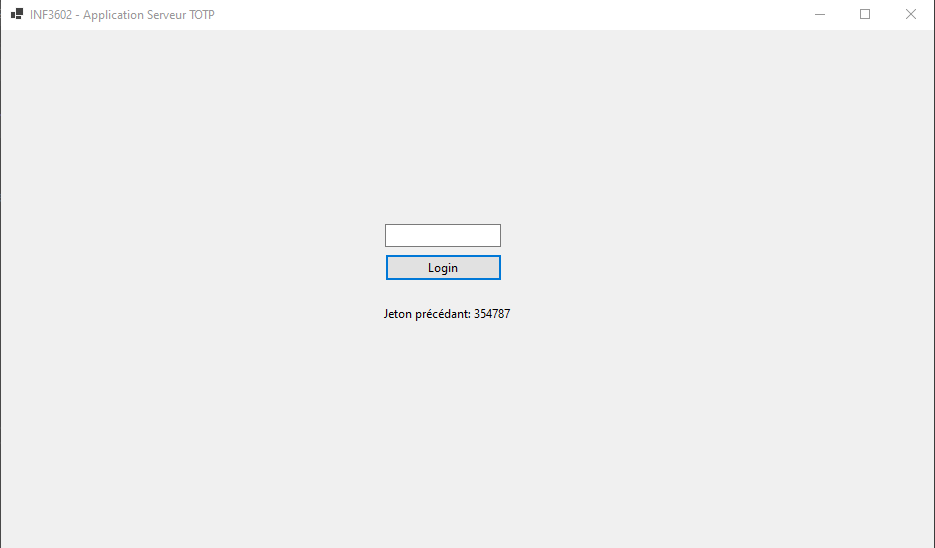
# Résultats

L’utilisation de l’architecture clean était un bon choix. Il a grandement réduit le temps de développement, car il nous a permis de réutiliser notre code dans les deux applications tout en facilité l’intégration avec l’interface et ce dû à l’inversion des dépendances. Durant les tests, aucun bogue n’a été trouvé et les systèmes remplissaient bien leurs fonctions.

# Conclusion

En conclusion, nos applications, bien que simplistes, répondent bien aux divers critères qui étaient exigés. Notre application client génère un jeton pseudo-unique toutes les 60 secondes et l’afficher à l’écran avec le temps de vie reste. Notre serveur génère le même jeton pour pouvoir faire la validation. Les deux applications créent les mêmes jetons malgré le fait qu’ils ne communiquent pas ensemble. De plus, aucun des deux n’a besoin d’une connexion à internet pour fonctionner. En somme, elles sont totalement indépendantes l’une de l’autre.

# Annexes



bizzehdeebizzehdee 19.9k99 gold badges4545 silver badges7676 bronze badges. “How to Get the Unix Timestamp in C#.” *Stack Overflow*, 1 July 1960, https://stackoverflow.com/questions/17632584/how-to-get-the-unix-timestamp-in-c-sharp.

freeCodeCamp.org. “How Time-Based One-Time Passwords Work and Why You Should Use Them in Your App.” *FreeCodeCamp.org*, FreeCodeCamp.org, 5 June 2018, https://www.freecodecamp.org/news/how-time-based-one-time-passwords-work-and-why-you-should-use-them-in-your-app-fdd2b9ed43c3/.

OgglasOgglas 56.9k3232 gold badges309309 silver badges387387 bronze badges, and NSKNSK 18,611 gold badge33 silver badges77 bronze badges. “Google Authenticator One-Time Password Algorithm in C#.” *Stack Overflow*, 1 Sept. 1964, https://stackoverflow.com/questions/46442076/google-authenticator-one-time-password-algorithm-in-c-sharp.

Tdykstra. “C# Console App Template Changes in .NET 6+—.NET.” *C# Console App Template Changes in .NET 6+—.NET | Microsoft Learn*, https://aka.ms/new-console-template.