**Paiement / Abo / Utilisateur**

* **Authentification SSO:**
* Token user

Pour la connexion à notre jeu, on va utiliser [OAuth](https://auth0.com/intro-to-iam/what-is-oauth-2) qui va nous permettre de générer un [token d’authentification](https://www.okta.com/identity-101/what-is-token-based-authentication/) pour ensuite pouvoir s’authentifier au jeu et accéder à sa progression.

* Enregistrement du token en SQL

On enregistre le token d’authentification qu’on a reçu via OAuth en SQL, pour ensuite enregistrer le token en base de données, ainsi que l’id/uuid de l’utilisateur afin de le différencier des autres utilisateurs. On enregistre aussi un refresh token dans la session du serveur ainsi qu’une date de validité, ce qui va nous permettre d’actualiser le nouveau token de l’utilisateur et ainsi pouvoir mettre à jour sa date de validité du token, sans pour autant avoir besoin de se reconnecter.

* Gestion de session

On vérifie que le token existe et que sa date de validité. Si jamais la date de validité du token est dépassée, on vérifie que le refresh token existe, S’il existe on l’utilise pour générer un nouveau token et une nouvelle date de validité. Dans le cas où le refresh token n’est pas valide, l’utilisateur est déconnecté et on lui demande de se reconnecter.

* Connection avec google, blizzard, microsoft

La première connexion se fait via trois services : Google, Blizzard ou bien Microsoft. Lors de la première connexion, l’utilisateur peut alors se connecter avec l’un de ses trois services. Une fois connecté, l’utilisateur est authentifié à la base de données par le compte qu’il a utilisé. Pour cela, l’API va lui créer un profil, avec les différentes informations de connexions pour l'identifier. Cela inclut la date de connexion ainsi que le service utilisé. Si le service utilisé est différent entre deux connexions, un nouveau profil va être créé.

* **Paiement**
* Stockage des informations de l’utilisateur

Pour le paiement de l’utilisateur, on utilise Stripe, qui est le plus simple, le plus populaire, moins onéreux que Paypal pour les taxes. D’autant plus que Stripe nous permet de faire des paiements avec des cartes bleues, ainsi que d’autres services de paiement tel que Paypal, Google Pay, Apple Pay, Visa, MasterCard… Ainsi que des centaines de devises différentes. Stripe permet aussi de faire des factures pour le paiement.

* Interaction avec des services externes

Notre API interagit avec Stripe, qui s’occupe du paiement pour nous. Cela permet de faciliter les transactions et de ne pas avoir besoin de faire l'interaction avec les différents services de de paiement soit même, qui demande des autorisations bien spécifiques propre à chaque service.

* Enregistrement des paiements effectués par l’utilisateur

On enregistre les informations de paiement (pas la carte bleue) sur la base de données SQL. Cela permet d’avoir une trace du paiement et d’avoir les logs des paiements effectués.

* Évaluer les abonnements (quel type)

En fonction du paiement choisi, on va enregistrer quel est le service utilisé pour l’utilisateur. Cela nous permettra ensuite de proposer directement la dernière méthode de paiement utilisée.

* Vérification du paiement (carte pas expiré)

Pour vérifier la carte, on va d’abord faire un virement de 0,01 euro pour voir si la carte n’est pas expirée ou bien qu’elle a de l’argent dessus. Cela permet de faire d’abord la vérification de la date d’expiration de la carte bleue. Une fois la vérification faite, on procède au paiement officiel (le vrai payement). Si c’est un abonnement, on propose à l’utilisateur de faire un renouvellement automatique. Pour cela, on enregistre en plus de la date du paiement, la date du prochain renouvellement et on enregistre les informations du paiement. Dans le cas contraire, on ne va pas enregistrer les informations du paiement (c'est-à-dire le numéro de carte bleue etc…) et on procède juste au paiement.

* Type de paiement

Tout se fait via Stripe pour gérer les paiements.

* Évaluer la devise du paiement

Stripe va convertir lui-même le montant de à devoir. Sachant que la somme et la devise est mise en dur dans le code. De ce fait, l’utilisateur va toujours payer le même prix dans la devise choisie par le développeur, et ne peut pas changer de devise pour payer moins chère.

* Calculer les taxes des micro-transactions

Le calcul des taxes se fait directement dans le code, en fonction des règles à appliquer selon cette dernière. Etant donné que nous utilisons Stripe, nous aurons [des frais](https://stripe.com/fr-us/pricing/local-payment-methods#:~:text=Nous%20proposons%20une%20tarification%20int%C3%A9gr%C3%A9e,cas%20de%20conversion%20de%20devises.) liés à la conversion de devise et aux transactions. Pour cela, on va tout mettre dans le code pour prendre en compte aussi la somme liée à tout cela (qui est déjà défini par Stripe à l’avance.)

* Securité

Pour des raisons de sécurité, on ne gère pas les paiements d’abonnement car il s’agit d’une responsabilité critique et trop grosse pour le gérer nous-même. C’est pour cela que nous utiliserons des services externes en termes de paiement.

* Issue de secoure

Si jamais Stripe n’est pas disponible, on peut partir sur son concurrent direct Paypal. Ce dernier a les mêmes fonctionnalités que Stripe, à la différence qu’il est disponible dans plus de pays mais à [des frais plus chers](https://www.paypal.com/fr/webapps/mpp/paypal-fees#no-fee). En plus, Paypal prend des commissions en fonction de la devise convertie, ce qui va compliquer l’implémentation de ce service, pour avoir la somme exacte avec les frais inclus.

* **Abonnement**
* Durée de l’abonnement

La durée de l’abonnement est définie déjà à l’avance dans le code et en base de données. De plus, la date d’expiration de l’abonnement est enregistrée dans la base de données. On vérifie aussi que l’utilisateur a un renouvellement sur son abonnement. On regarde aussi que la carte (si elle est enregistrée) n’a pas expiré, sinon on peut prévenir que la carte n’est plus valide. Si les droits ne sont plus valides, on va mettre à jour le compte de l’utilisateur et lui enlever les avantages de son abonnement.

* Validité de l’abonnement

La validité de l’abonnement se vérifie en base de données. On va regarder la date de validité. Si jamais l’abonnement fini et non renouvelé, on décide d’enlever les avantages à l’utilisateur et de lui demander de se reconnecter pour mettre à jour son profil.

* Attribution

Lors de l’achat in-app, on attribue un skin ou bien un abonnement dans la base de données. Cela se fait par l’ajout de référence des images dans la base de données (de l’id de l'image dans un tableau par exemple), et d’un boolean pour l’abonnement.

* **Utilisateur**
* Sauvegarde des données de l’utilisateur

Une fois connecté, le profil de l’utilisateur est créé dans notre base de données Cassandra. Le pseudo du joueur, les préférences (langage, option), abonnement en cours, skins, token authentication, niveau du joueur, nombres de parties jouées, de parties gagnées, perdues, historique des parties sont stockés dedans.

* Sauvegarde périodique

Lors de l’achat in-game, ou bien d’une fin de partie ou encore lors des changements dans les paramètres du jeu, les données de l'utilisateur sont sauvegardées dans la base de données Cassandra. Cette dernière étant en haute disponibilités, les données seront accessibles tout le temps et en sécurités. Cela permet d’éviter la perte de la progression du joueur.

|  |
| --- |
| Schéma d’action du joueur |