**Paiement / abonnement / utilisateur**

Load balancer :

* Pour éviter la surcharge des serveurs, ou simplement l’inaccessibilité du serveur applicatif par les utilisateurs en cas de crash, on les redirige vers d’autres serveurs en marche

Message broker :

* Pour éviter un nombre important de requêtes d’un coup qui surchargerait le serveur, le message broker va permettre de dépiler / traiter chaque requête au rythme du serveur

Architecture = micro-service :

* Car on ne veut pas qu’un service n’ayant pas de rapport avec un autre ait une influence sur son fonctionnement / état. Par exemple le service de boutique qui casserait le serveur sur une architecture monolithique casserait l’entièreté du jeu y compris les parties en ligne.

Acteurs :

* Utilisateur / joueur :
  + Paiement pour s’abonner au Battlepass
    - Depuis le menu principal, va sur la page Battlepass
    - Clique sur le bouton « s’abonner »
    - Va sur la page de paiement
  + Paiement pour acheter des armes et cosmétiques sur le catalogue
    - Depuis le menu principal, va sur le catalogue
    - Ajoute les objets dans son panier
    - Va sur le panier
    - Exécute le paiement de son panier
* Intermédiaire de paiement / API (Stripe, …)
* Banque (autorisation de paiement)

Besoins :

* Besoin d’une page / formulaire de paiement
* Besoin d’un intermédiaire / facilitateur de paiement (stripe, paypal, …)
  + Communication par le biais d’une API REST
* Besoin d’une modale de confirmation de paiement en 2FA par la banque de l’utilisateur.
  + Communication par le biais d’une API REST

Interactions entre acteurs et besoin :

* Utilisateur rempli et envoi le formulaire de paiement Stripe
  + Ceci envoi une requête POST à Stripe
* Stripe reçoit la requête et paie l’entreprise à qui appartient le jeu
* Le web service de paiement reçoit la réponse

Diagramme de cas d’utilisation :

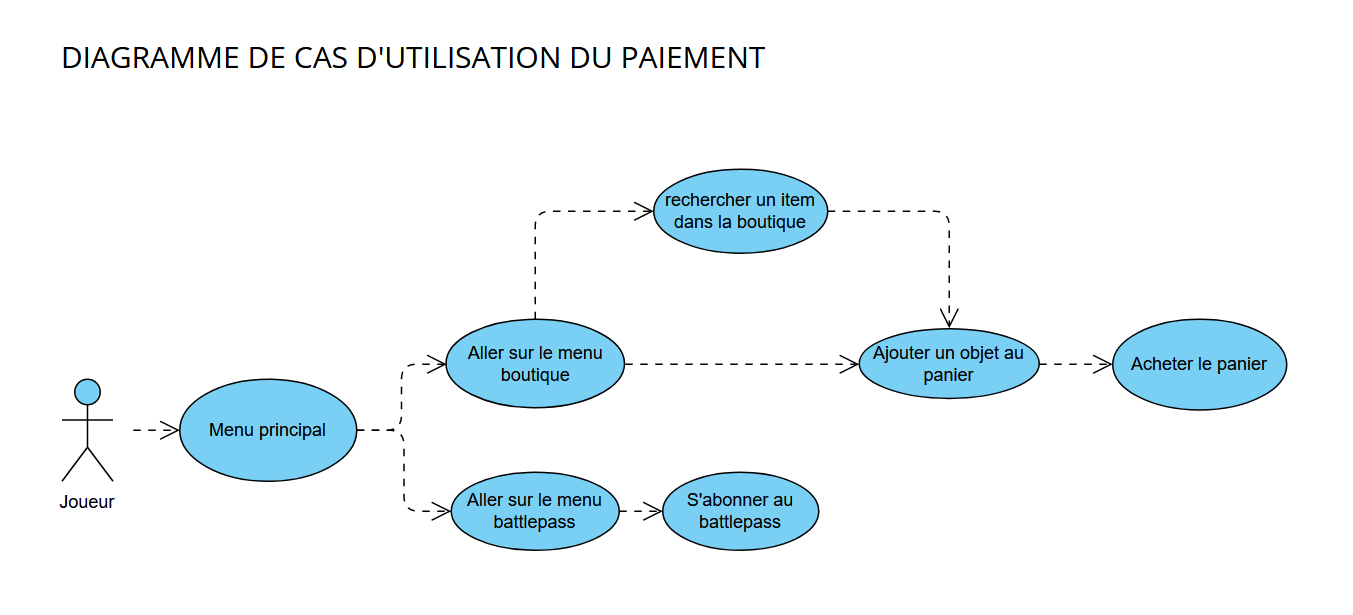
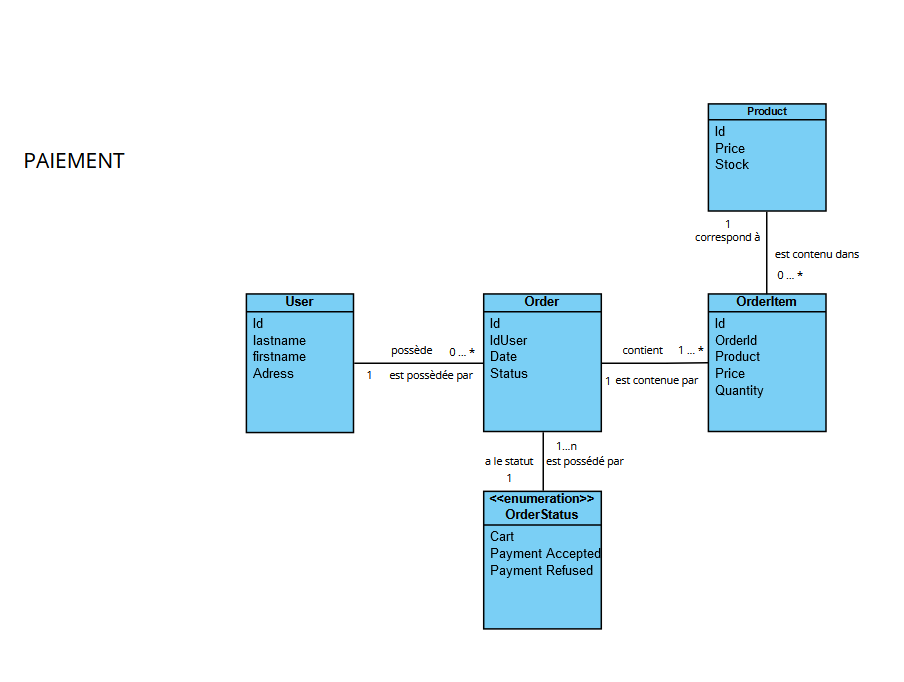


Diagramme de classe :



Modele C4 :