

Rocket Fetcher



**ROCKET
LEAGUE**

Projet :

C#
.NET (Xaml/WPF)

Dates :

Du : 30/03/2021
au : 07/06/2021

Professeur :

CHEVALDONNE Marc

Par :

CLERGUE Valentin
GADET Jordan

Sommaire :

Partie 1 :

- Sommaire :		p2
- Contexte :		p3
- Personas & userstories :		p4-6
* Persona 1 :	p4	
* Persona 2 :	p5	
* Persona 3 :	p6	
- Wireframes :		p7-18
* Connexion/Création :	p7-8	
* Master/detail :	p9-13	
* Fonctionnalités supplémentaires :	p14-17	
* Test couleur basique :	p18	
- Diagramme de cas d'utilisation :		p19-23
* Diagramme :	p19	
* Description :	p20-23	
- Considérations ergonomiques :		p24
- Prise en compte de l'accessibilité :		p25

Partie 2 & 3 :

- Diagramme de paquetage + (persistance) :		p26
- Diagramme de classes + (persistance) :		p27
- Diagrammes de séquences :		p28-29
* Mise en œuvre des classes principale :	p28	
* Chargement des données depuis le stub :	p29	
- Description écrite de l'architecture :		p30-40

Vidéo :

- La vidéo de présentation est disponible dans : **[/tags/Video](#)**



Contexte :

Vous en avez marre de passer vos jours et vos nuits à essayer de customiser votre voiture dans *Rocket League* car le système de trie proposé par le jeu n'est pas optimal ? Et bien **RocketFetcher** est là pour résoudre votre problème !
Mais qu'est-ce que le jeu *Rocket League* et l'application **RocketFetcher** ?

Rocket League est un jeu vidéo assez connu dans le monde de l'e-Sport et du milieu compétitif. Il consiste simplement en 2 équipes de 1, 2 ou 3 joueurs s'affrontant dans une arène pendant 5 minutes avec des voitures ayant la capacité de voler avec pour objectif de marquer plus de buts que ses adversaires.
TLDR : Dit avec des mots simples ; c'est un jeu de foot avec des voitures.

Afin de résoudre le problème de la customisation de votre voiture, l'application **RocketFetcher** vous propose une interface simple et épurée vous permettant de facilement créer de multiples sets d'items ainsi que la possibilité de naviguer parmi tous les items existant dans *Rocket League*.

Que vous soyez expert ou débutant, **RocketFetcher** vous permet d'économiser beaucoup de temps ainsi que de partager et recevoir des sets d'items entre amis. Bien qu'il soit nécessaire de connaître les bases de *Rocket League* pour comprendre comment fonctionne la classification des items ainsi que les restrictions qui incombent ; **RocketFetcher** est très facile à prendre en main.

RocketFetcher est une application totalement gratuite que l'on peut utiliser avec ou sans compte. Avec le mode sans compte, les fonctionnalités sont limitées au parcours des items de *Rocket League* et à la création d'un set d'item temporaire (qui disparaît à la fermeture de l'application). Le mode avec compte permet quand à lui de faire ce qui est possible avec le mode sans compte ainsi que de sauvegarder, éditer, importer et exporter des sets d'items en les liants à son compte.

L'application permet donc de naviguer parmi tous les items que possède *Rocket League* grâce à des filtres et de voir les détails de chacun de ces items.



Personas & userstories :

N°1 :

Kevin Bucheron



Intitulé de poste
Directeur Général

Âge
Entre 35 et 44 ans

Niveau d'études
Master ou diplôme équivalent

Réseaux sociaux



Secteur d'activité
Comptabilité

Taille de l'entreprise
51 à 200 salariés

Moyen de communication préféré

- Face à face
- E-mail
- Téléphone

Outils nécessaires au quotidien

- Outil d'e-mailing
- Logiciel de traitement de texte
- Logiciel de planning des salariés
- Applications de stockage et de partage de dossiers en ligne
- Système de gestion et de comptabilité
- Outil de gestion de projet

Responsabilités

Kevin est responsable du bon fonctionnement des différentes équipes et de leur organisation

Indicateurs de performance

38000€/an.

Principaux défis

- Communications et relations avec les clients
- Gestion des projets et organisation

Loisirs

Aime bien jouer à Rocket League et à d'autres jeux vidéos

Objectifs

Avoir un revenu stable lui permettant d'en vivre dans un milieu qui lui plait tout au long de sa vie

En dehors des horaires de travail, Kevin aime bien jouer à *Rocket League*, il avait un bon niveau et progressait rapidement mais était assez triste de sa voiture qui n'est pas trop la voiture de ses rêves, il ne savait pas trop comment s'y prendre et ne trouvait pas d'application facile d'utilisation pour répondre à son besoin...

« Je voulais vraiment personnaliser ma voiture mais j'avais des difficultés à trouver comment faire des tests avec des composants que je n'ai pas, et même si je sais que ce n'est pas nécessaire pour jouer au jeu je voulais quand même bien y arriver !

Ensuite j'ai découvert ***RocketFetcher***, ce qui m'a bien aidé car c'est simple d'utilisation et j'ai pu faire un grand nombre de tests pour enfin avoir la voiture qui me plais ! »

N°2 :

Steve Bami



Intitulé de poste
Etudiant

Âge
Moins de 18 ans

Niveau d'études
Inférieur au baccalauréat

Réseaux sociaux



Moyen de communication préféré

- Réseaux Sociaux
- Sms

Outils nécessaires au quotidien

- Outil d'e-mailing
- Sites Scolaires (ENT...)

Objectifs

En finir avec le Collège et rentrer au Lycée , obtenir son bac et continuer ses études pour devenir développeur Java.

Principaux défis

- Apprendre les bases de programmation
- Prendre un bon train de Travail

Loisirs

Aime Beaucoup jouer aux jeux vidéos avec ses amis du collège, principalement Rocket League car il est membre d'un club de foot depuis 3 ans et qu'il aime bien le concept du jeu.

Quand il n'est pas en cours, Steve aime beaucoup jouer au football en club et jouer aux jeux-vidéos avec ses amis du collège. *Rocket League* lui parle bien car c'est un jeu vidéo qui reprend les bases du football, et avec ses amis ils se sont lancé le défi d'avoir la plus belle voiture du groupe en une semaine, Steve voulait donc les battre à tout prix...

« Avant de découvrir **RocketFetcher** je n'arrivais pas vraiment à trouver un bon design de voiture pour battre celles de mes amis, les tests étaient trop longs et ça ne m'allait pas. En cherchant j'ai donc découvert cette application et elle m'a vraiment beaucoup aidé car grâce à sa vitesse j'ai réussi à rattraper mon retard sur mes amis et à les battre avec ma plus belle création ! »

Steve utilise **RocketFetcher** environ une fois toutes les 2 semaines, au moment où des items sont rajoutés dans *Rocket League*.

N°3 :

Bernadette Claude



Intitulé de poste
Etudiante

Âge
Entre 18 et 24 ans

Niveau d'études
BAC+1

Réseaux sociaux



Secteur d'activité
Psychologie

Moyen de communication préféré

Aime les réseaux sociaux mais préfère le contact humain.

Outils nécessaires au quotidien

- Outil d'e-mailing
- Sites Scolaires (ENT...)
- Outils de traitement de texte

Objectifs

Réussir sa 2nde année en Faculté de Psychologie et continuer ses études. Puis obtenir un travail et vivre avec son petit-ami.

Sources d'information

Experts ou textes vérifiés et approuvés pour les utiliser dans ses études.

Loisirs

Lire des livres comme des Mangas d'un peu tout les genres, regarder des séries et films sur Netflix avec son petit-ami, et depuis peu, jouer à Rocket League et quelques autres jeux en multijoueur principalement. Joueuse de Piano depuis ses 10 ans et toujours pratiquante.

Bernadette a commencé *Rocket League* avec son petit-ami et joue aussi avec quelques amis de temps à autres quand elle n'étudie pas pour ses cours. Elle voulait pouvoir trouver assez facilement deux voitures aux designs différents qui iraient bien ensemble pour son petit-ami et elle-même, selon leurs propres goûts ...

« Je voulais vraiment créer deux voitures qui puissent aller ensemble sans être les mêmes, mais tester tout sur le jeu est très long et me prends trop de temps, j'ai donc décider de chercher quelque chose pour m'aider et j'ai trouvé **RocketFetcher**, qui, de par sa rapidité est simple à utiliser pour atteindre mon but et qui me permet de comparer en même temps sur le jeu et sur l'application des designs pour voir s'ils sont compatibles ! »

Bernadette utilise **RocketFetcher** de manière assez irrégulière. Elle lance l'application de temps à autre quand elle veut changer sa voiture de style.

Wireframes :



Bouton permettant de passer du mode nuit au mode jour et inversement.

RocketFetcher

Champ pour rentrer son nom d'utilisateur.

Champ pour rentrer son mot de passe.



Bouton permettant d'accéder à l'application une fois le login et le password correspondant entré.

☒ Remember me

Case à cocher permettant de sauvegarder le login



Forgot password ?

Bouton permettant de simuler l'envoi d'un mail de récupération du mot de passe.



Create an account

Bouton pour accéder à la page de création de compte.



Guest mode

Bouton pour accéder à l'application sans utiliser de compte

Pour des raisons de simplicité, toutes les fenêtres suivantes ont été faites à la même taille.

Il est cependant prévu qu'elles soient toutes responsives.

Tous les commentaires écrits en bleu n'apparaîtront pas sur l'application, ils sont là pour détailler chacune des fonctionnalités.

Première page lorsque l'on lance l'application. Elle permet de se connecter à son compte ou de passer par le mode sans compte.

Bouton de retour à l'écran de connexion.



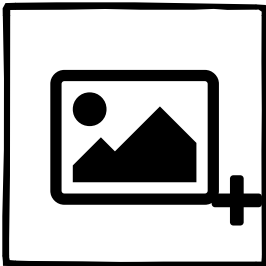
RocketFetcher



Bouton permettant de passer du mode nuit au mode jour et inversement.

Create an account :

Avatar



Bouton déclanchant l'ouverture de l'explorateur de fichier pour pouvoir choisir une image sur son ordinateur.

Email

mail@service.extension



Pseudo/login

Pseudo/login



Password

Password

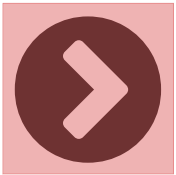


Re-Passwd

Re-Passwd



Champs obligatoires !



Bouton d'enregistrement du compte (après que les informations nécessaires aient été fournies).

Champs de texte servant à renseigner les informations nécessaires à la création d'un compte.

Lastname

Lastname

Firstname

Firstname



Date of birth

02/04/2021






Page de création de compte, elle est accessible depuis celle de connexion ainsi que par le bouton "profil" sur celle du mode sans compte.


RocketFetcher





Informations "bateau" servant à montrer que l'on se trouve sur le mode sans compte.



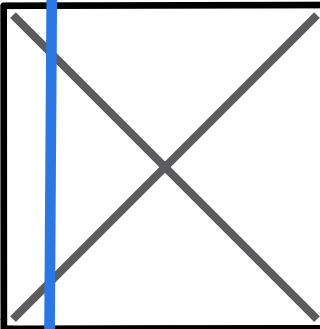
Boutton permettant l'accès à la fenêtre de connexion.

Group by 


Sort by 

 item name

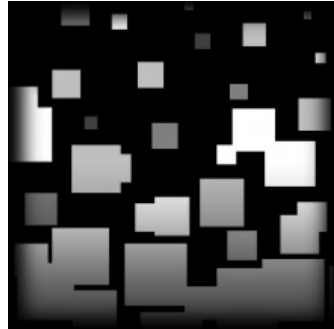
Permet de ne montrer que les items matchant avec l'entrée faite dans ce champ.



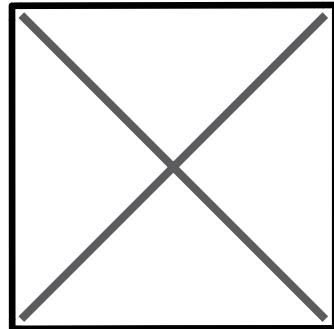
ALL ITEMS



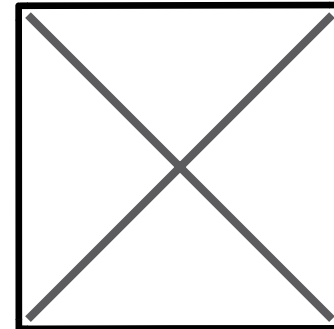
BODIES



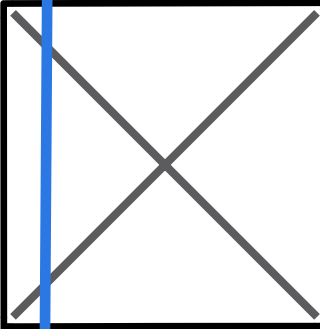
DECALS



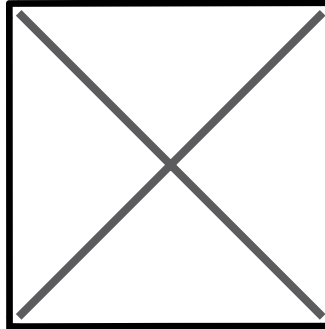
PAINT



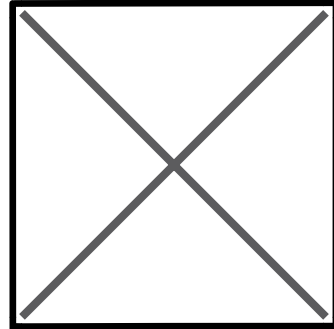
WHEELS



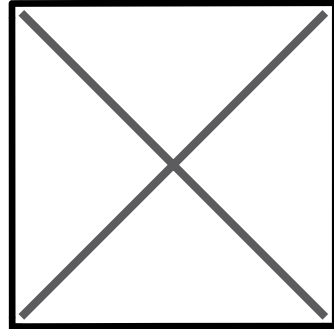
BOOSTS



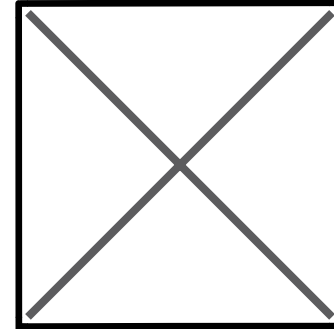
TOPPERS



ANTENNAS




GOAL




TRAILS

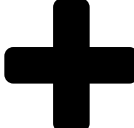
Item set :



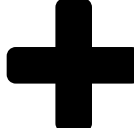
BODY



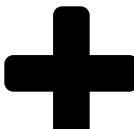
DECAL




WHEELS



BOOST





Clic gauche pour accéder à la page de l'item.
Clic droit pour retirer l'item du set.

Boutons (cases) permettant d'accéder au différentes pages des catégories sus nommées.

Permet de grouper les items par :

- Type (par défaut)
- Rareté
- Couleur
- Collection d'origine

Permet de trier les items par :

- Nom (par défaut)
- Rareté
- Couleur
- Collection d'origine

Page principale du mode sans compte.
Ne sont détaillées que les options intrinsèques à ce mode.

Cette page fonctionne de façon similaire à celle du "mode avec compte" si ce n'est les quelques options manquantes telles que la sauvegarde de set ou encore de l'onglet collection.

Set d'item en cours de création

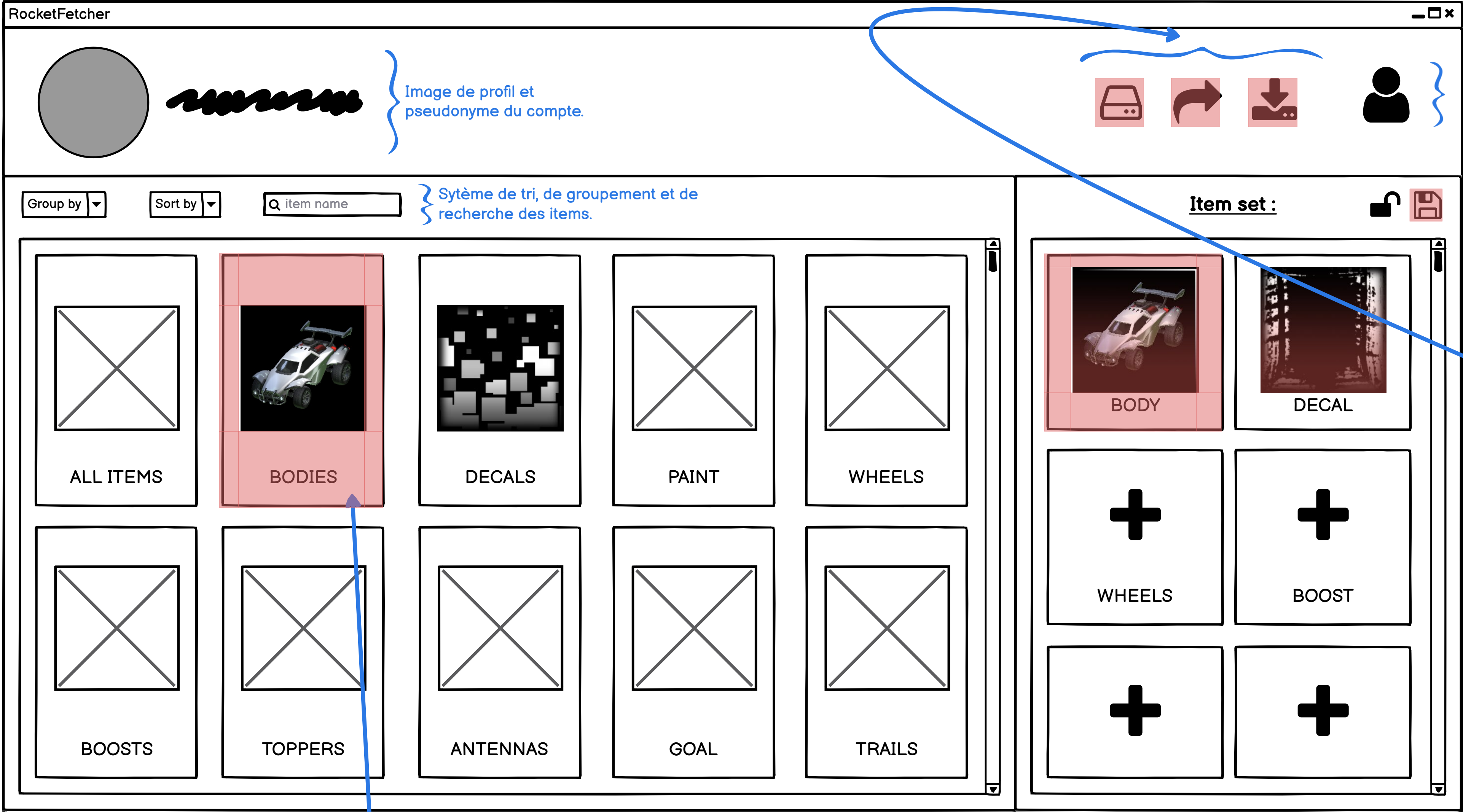


Image de profil et
pseudonyme du compte.

Bouton permettant l'accès à la
fenêtre de gestion du compte.

Système de tri, de groupement et de
recherche des items.

Bouton servant à verrouiller les modifications sur le set actif.
Bouton servant à enregistrer le set actif (redirection vers la
page de sauvegarde).

De gauche à droite :

Bouton permettant
l'accès à la page des
sets enregistrés.

Bouton permettant de
partager le set actif
(l'exporter).

Bouton permettant
d'importer un set
depuis un fichier.

Catégories principales des items
du jeu.
Cliquer sur l'une d'entre elles vous
amène à sa page correspondante.

Nous prendrons par exemple celle
des "voitures" (body).

Nous allons
maintenant
imaginer un clic sur
la catégorie
"bodies".

Page principale de l'application.
Elle donne accès à presque toutes les
fonctionnalités de l'application.

RocketFetcher

Group by

Sort by

Q item name

Système de tri, de groupement et de recherche des items.

←

BACK

Backfire

Breakout

Dominus

Gizmo

Merc

Octane

Paladin

Road Hog

Venom

Item set :

BODY

DECAL

+

WHEELS

+

BOOST

+

+

Bouton (case) permettant de retourner à la page des catégories.

Nous allons maintenant imaginer un clic sur cette catégorie de véhicule.

Page de recherche des voitures (body). Les pages de recherche des autres items seront similaires.

RocketFetcher

Group by

Sort by

Q item name

Système de tri, de groupement et de recherche des items.

BACK

Paint less

White

Grey

Crimso

Pink

Cobalt

Sky blue

Burn sienna

Saffron

Item set :

BODY

DECAL

WHEELS

BOOST

Bouton (case) permettant de retourner à la page des catégories de véhicules.

Nous allons maintenant imaginer un clic sur ce véhicule.

Page de recherche des octanes (type de voiture). Les pages de recherche des autres items seront similaires.

RocketFetcher

Group by


Sort by

Q item name

Système de tri, de groupement et de recherche des items.

←

BACK



Import : White Octane

Ne sachant quoi mettre dans cet emplacement, nous le laissons libre si une idée nous vient.

PUB : Vidéo Chevaldonné

En dark salmon

Item infos :

Rarity

Import

Type

Car

Series

none

Release date

03/07/2017

Paintable

All (except black)

Certifiable

Yes

Tradeable

No


Has blueprint

No


Price

17.5k - 19.5k

Item set :



BODY



DECAL

+

WHEELS

+

BOOST

+

+

Bouton (case) permettant de retourner à la page de la liste des d'octanes.

Clic gauche pour l'ajouter au set d'item actif (fenêtre de droite)

Page qui détail l'item : white octane
Les pages des autres items seront similaires.

RocketFetcher

Group by

Sort by

Q item name

ALL ITEMS

BODIES

BOOSTS

TOPPERS

Import

File explorer

Disk:/Desktop

Cet explorateur de fichier classique permettra d'aller chercher le fichier du set d'item à travers les différents répertoires windows. Il sera aussi possible de rentrer directement le path du fichier.

Cet explorateur ne ressemble pas du tout à celui que nous utiliserons !

Item set :

BODY

DECAL

BOOST

Pop up d'ajout d'un set déjà existant grâce à un fichier.

RocketFetcher

Group by

Sort by

Q item name

ALL ITEMS

BODIES

BOOSTS

TOPPERS

Share

Item Set 1 : Super voiture de la mort

Item Set 2 : Oulah

Item Set 3 : Full orange

Cliquer sur un des icones de partage
permettra de créer un fichier de
partage dans le dossier mentionné
plus bas.

Go on

Disk:\Desktop\RocketFetcher\Share

Item set :

BODY

DECAL

HEELS

BOOST

Pop up d'exportation d'un set d'item préalablement enregistré.

RocketFetcher

Bouton permettant d'accéder à cette page.

←

Back to fetcher

Item Set 1 : Super voiture de la mort

Item Set 2 : Oulah

Item Set 3 : Full orange

Sets existants

+

New Item Set

Bouton permattant d'enregister le set d'item actif sur un nouvel emplacement (nouveau nom)

Current set

BODY

DECAL

+

Bouton (case) permettant de retourner sur la page des catégories pour continuer l'élaboration du set.

Item set :

BODY

DECAL

+

WHEELS

+

BOOST

+

+

Bouton permettant d'accéder à cette page, sous réserve que le set actif possède au moins 1 item.

Une fois cliqué (fermeture du cadena) cette option empêche de retirer les items déjà présents dans le set. (clic droit inopérant)

Page d'enregistrement d'un nouveau set d'item.

RocketFetcher

Back to fetcher

Item Set 1 : Super voiture de la mort

Item Set 2 : Oulah

Item Set 3 : Full orange

Sets existants

+New Item Set

Current s

BO

New Item Set

Recap

BODY

DECAL

Nom du nouveau set.

Sauvegarder.

Set name

Annuler.

Item set :

BODY


DECAL





HEELS

BOOST

Pop up d'enregistrement d'un nouveau set d'item.
(Vérification)

RocketFetcher



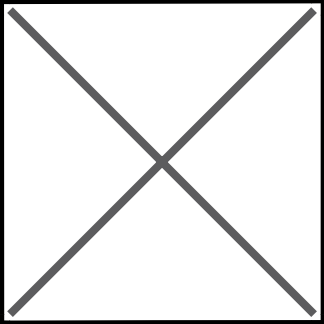


Group by ▼

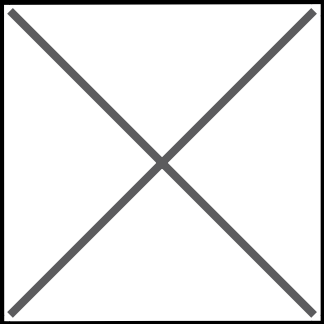
Sort by ▼

Q


item name



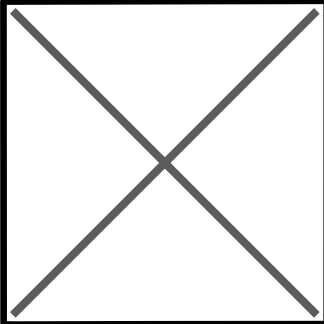
ALL ITEMS



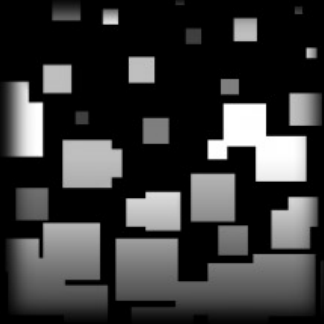
BOOSTS



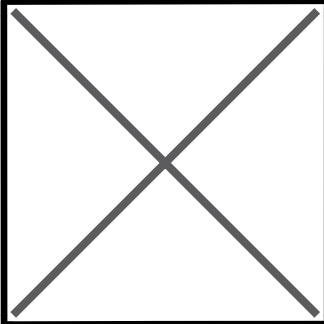
BODIES



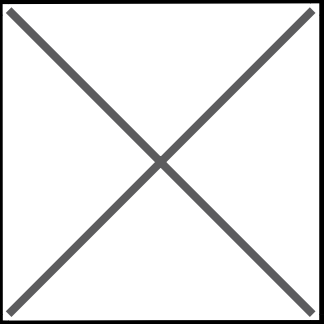
TOPPERS



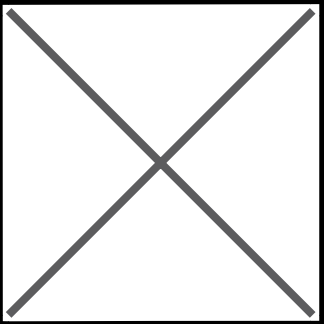
DECALS



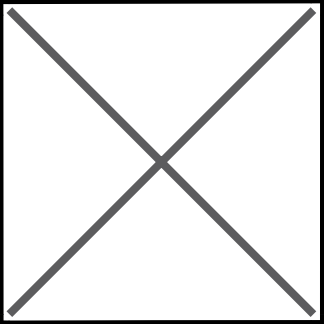
ANTENNAS



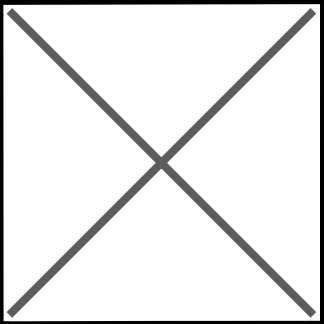
PAINT



GOAL






WHEELS




TRAILS

Item set :







BODY




DECAL




WHEELS



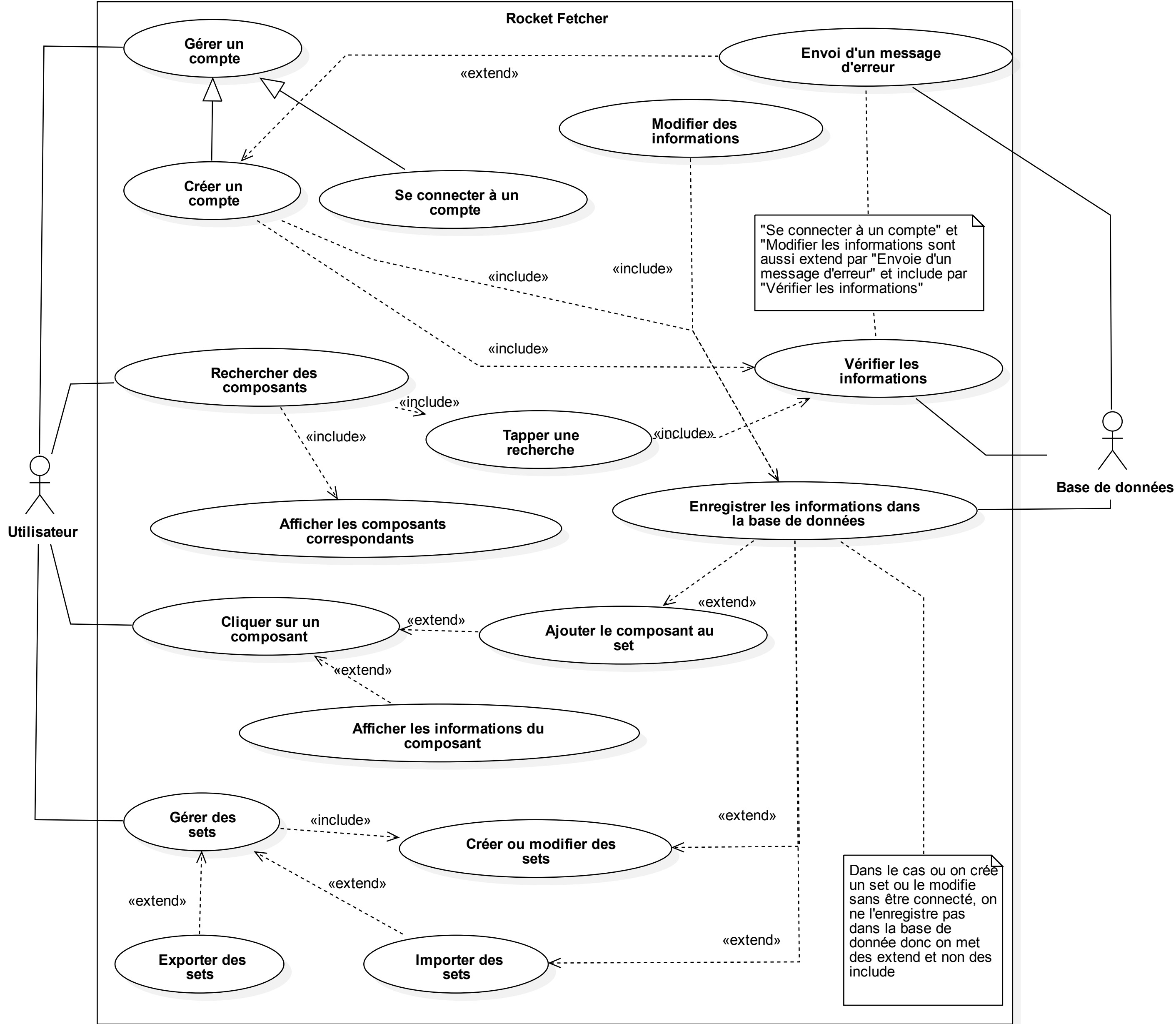
BOOST





Test de coloriage d'une fenêtre basique pour mieux différencier les différentes imbrications des vues.

Diagramme de cas d'utilisation :



Descriptions :

Cas « Créer un compte »

Nom :	Créer un compte
Objectif :	Créer un compte dans l'application
Acteurs principaux :	Utilisateur
Acteurs secondaires :	Base de Données
Conditions Initiales :	-les données du compte a créer ne doivent pas déjà exister dans la base de données.
Scénario d'utilisation :	-l'utilisateur rentre ses données -les données sont examinées par la base de données -Si elles n'existent pas déjà (condition de fin 1) Sinon (condition de fin 2)
Conditions de fin :	1) Le compte est créé et enregistré avec succès 2) Le compte n'est pas créée car les données existent déjà dans la base de données et lancement d'un message d'erreur

Cas « Se connecter à un compte »

Nom :	Se connecter à un compte
Objectif :	Se connecter à un compte dans l'application
Acteurs principaux :	Utilisateur
Acteurs secondaires :	Base de Données
Conditions Initiales :	-les données du compte doivent exister
Scénario d'utilisation :	-l'utilisateur rentre ses données -les données sont examinées par la base de données -Si elles existent déjà (condition de fin 1) sinon (condition de fin 2)
Conditions de fin :	1) Le compte existe, la connexion est effectuée 2) Le compte n'existe pas, connexion impossible et lancement d'un message d'erreur

Cas « Modifier des informations »

Nom :	Modifier des informations
Objectif :	Modifier des informations du compte
Acteurs principaux :	Utilisateur
Acteurs secondaires :	Base de Données
Conditions Initiales :	-les données du compte doivent exister dans la base de données -l'utilisateur doit être connecté
Scénario d'utilisation :	-l'utilisateur modifie des données -les données sont examinées par la base de données -S'il n'y a pas de modifications (condition de fin 1), si les données existent (condition de fin 2), sinon (condition de fin 3)
Conditions de fin :	1) Aucune modification : aucun changement 2) Les modifications sont appliquées et les données sont enregistrées 3) Modification des données existantes : impossible. Envoi d'un message d'erreur.

Cas « Rechercher des composants »

Nom :	Rechercher des composants
Objectif :	Rechercher des composants (item) en fonction de paramètres précis
Acteurs principaux :	Utilisateur
Acteurs secondaires :	Base de Données
Conditions Initiales :	-les composants doivent exister dans la base de données
Scénario d'utilisation :	-l'utilisateur initie le ou les paramètres de recherche -les paramètres sont examinées par la base de données -Si les paramètres correspondent à des composants (condition de fin 1), sinon (condition de fin 2)
Conditions de fin :	1) Affichage des composants correspondants aux paramètres 2) Rien n'est affiché

Cas « Cliquer sur un composant »

Nom :	Cliquer sur un composant
Objectif :	Réaliser une action en fonction du clic
Acteurs principaux :	Utilisateur
Acteurs secondaires :	Base de Données
Conditions Initiales :	-le composant doit exister dans la base de données
Scénario d'utilisation :	-l'utilisateur clique sur le composant -Si le clic est un clic droit (condition de fin 1), sinon (condition de fin 2)
Conditions de fin :	1) Ajout du composant au set actuel 2) Affichage des informations du composant

Cas « Gérer des sets »

Nom :	Gérer des sets
Objectif :	Créer, modifier, importer ou exporter des sets
Acteurs principaux :	Utilisateur
Acteurs secondaires :	Base de Données
Conditions Initiales :	<ul style="list-style-type: none">-le set doit exister dans la base de données pour le modifier ou l'exporter-le set ne doit pas exister pour le créer ou l'importer
Scénario d'utilisation :	<ul style="list-style-type: none">-l'utilisateur interagi avec un set-si le set existe et que l'utilisateur veut le modifier ou l'exporter (condition de fin 1) sinon (condition de fin 2)-si le set n'existe pas et que l'utilisateur veut le créer ou l'importer (condition de fin 3) sinon condition de fin 4)
Conditions de fin :	<ul style="list-style-type: none">1) Modification / Exportation réussie2) Modification / Exportation n'a pas fonctionné et message d'erreur3) Création / Importation réussie4) Création / Importation n'a pas fonctionné et message d'erreur

Considérations ergonomiques :

Mise à l'échelle :

- Taille minimale : 1080x720
- Taille Maximale : 2560x1920

Ces tailles conviennent à la majorité des écrans vendu dans le commerce aujourd'hui.

En effet, l'application a été design pour avoir un rendu agréable tant sur des écran 1080p que 2k mais également sur un écran 1080p vertical. Il est également possible d'adapter la taille de la fenêtre entre les bornes citées plus haut. L'application est aussi très peu consommatrice et est donc adaptée pour une grande partie des appareils (même peu performants) pour le plus grand plaisir des petites configurations.

Disposition du menu :

L'application est constituée d'une bande horizontale sur la partie haute qui regroupe les différents boutons liés à leurs fonctionnalités. Elle est suivie d'une seconde barre servant au tri et à la recherche des différents items. Selon les différents pages, le dessous est différent mais garde une certaine mise en forme.

Toutes les pages du « fetcher » de l'application suivent le même modèle avec les même proportions, ce qui permet à l'utilisateur une navigation simple et rapide dans celle-ci.

Couleurs et Visuel :

Les pages possèdent une même image de fonds, avec des couleurs agréables à la vision, les bandeaux sont assez sobres avec une couleur grisée et permettent un confort des yeux. Les boutons ont des formes à but d'attirer l'œil et possèdent une même forme pour garder une uniformité. L'utilisateur pourra aussi remarquer que certains boutons sont toujours à la même place pour ne pas rendre pénible la navigation. Les diverses possibilités qu'offre notre application ont mis au premier plan le côté visuel et permet une meilleure expérience dans celle-ci.

Prise en compte de l'accessibilité :

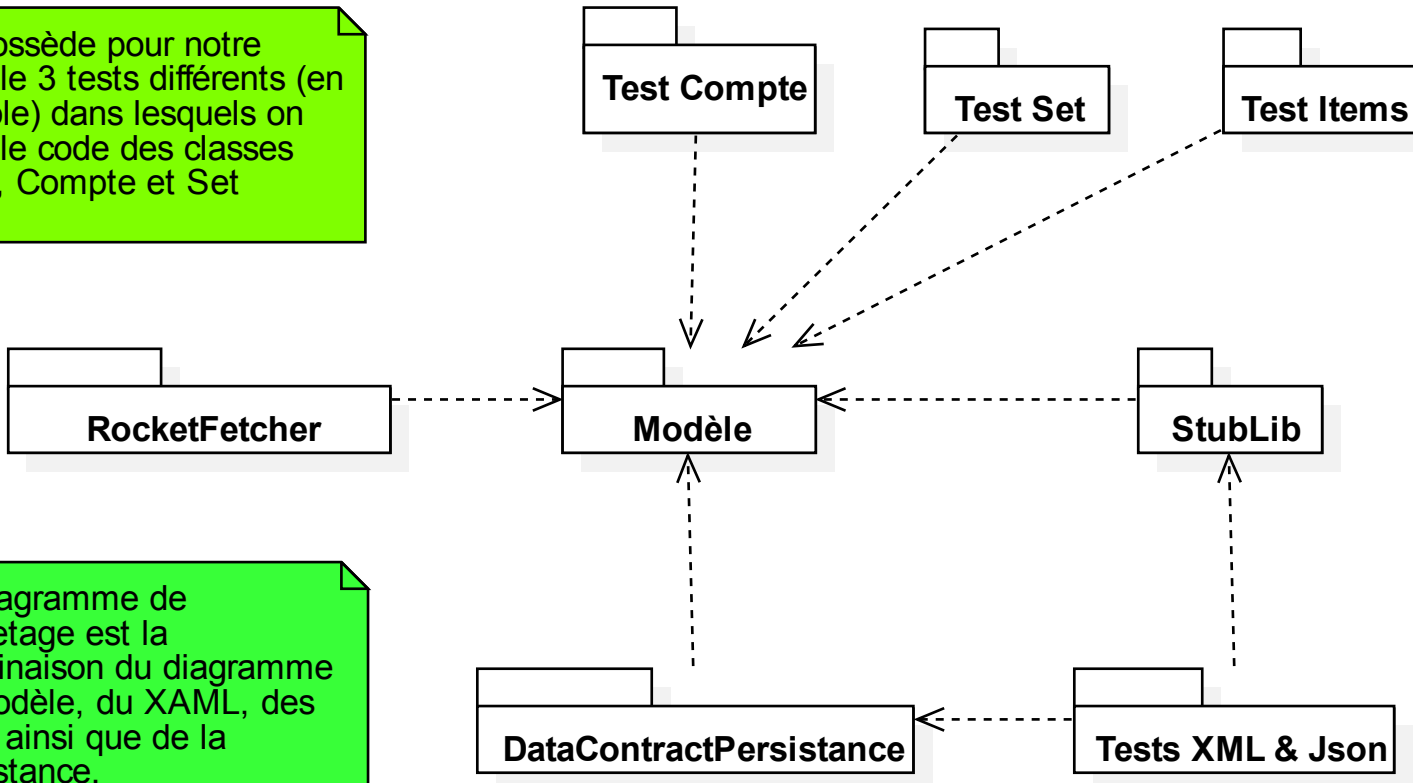
Accessibilité :

Nous avons pensé à rendre l'application disponible avec ou sans compte pour les personnes qui comptent l'utiliser beaucoup ou non. L'application est totalement accessible et gratuite pour tout le monde qui peut posséder un appareil électronique (le seul problème). Mis à part ça, nous n'avons pas vraiment mis en place de mesures pour les non-voyants ou mal-voyant car ceux-là ne peuvent vraiment jouer au jeu du fait que ce jeu est accès exclusivement sur le visuel. Notre application est néanmoins accessible à une majorité des personnes qui seraient intéressées pour l'utiliser les textes sont par ailleurs assez lisibles.

Partie 2 & 3 :

Diagramme de paquetage + (persistance) :

On possède pour notre modèle 3 tests différents (en console) dans lesquels on teste le code des classes Item , Compte et Set



Ce diagramme de paquetage est la combinaison du diagramme du Modèle, du XAML, des Tests ainsi que de la Persistance.

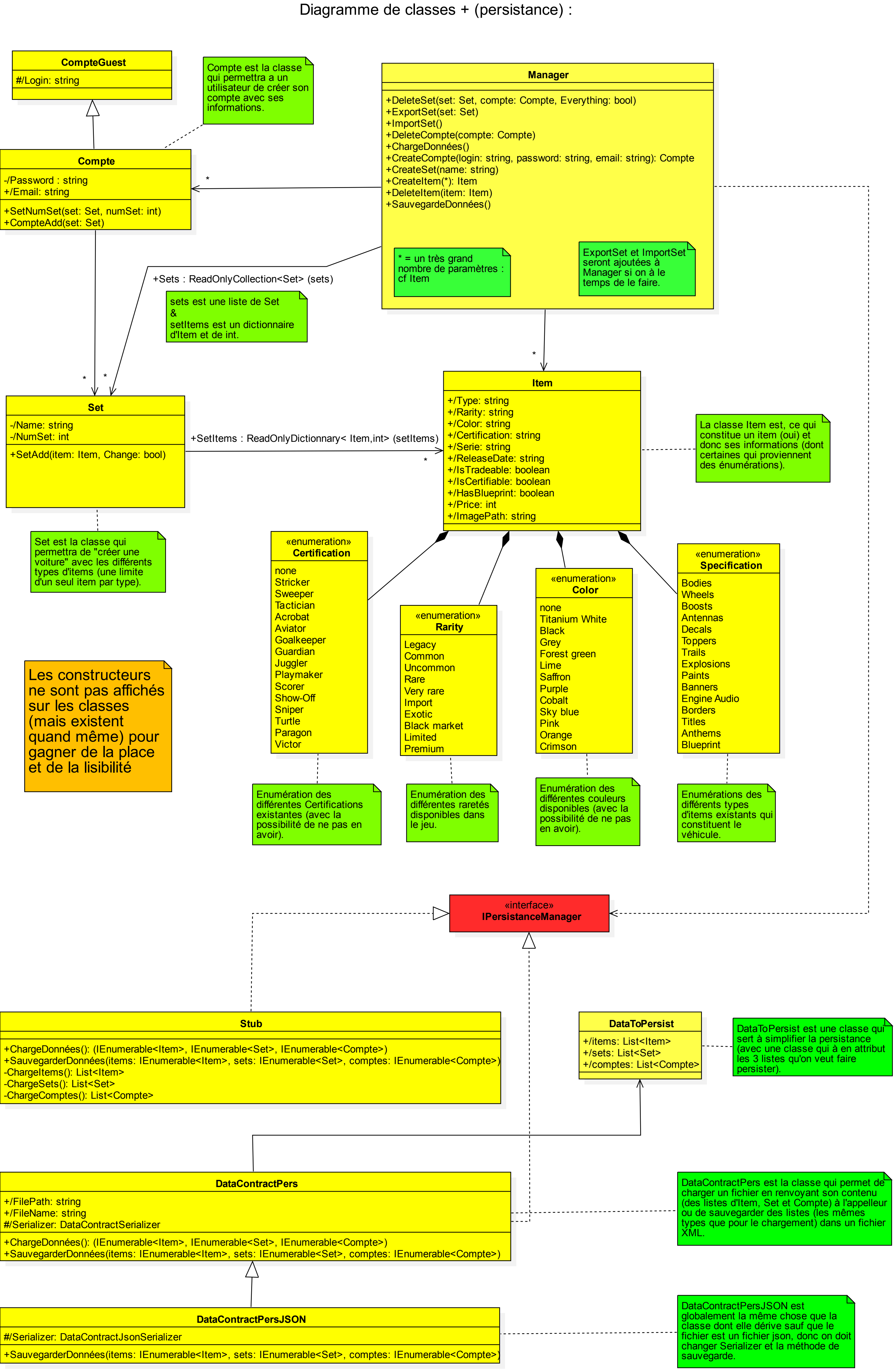


Diagramme de séquence n°1 :

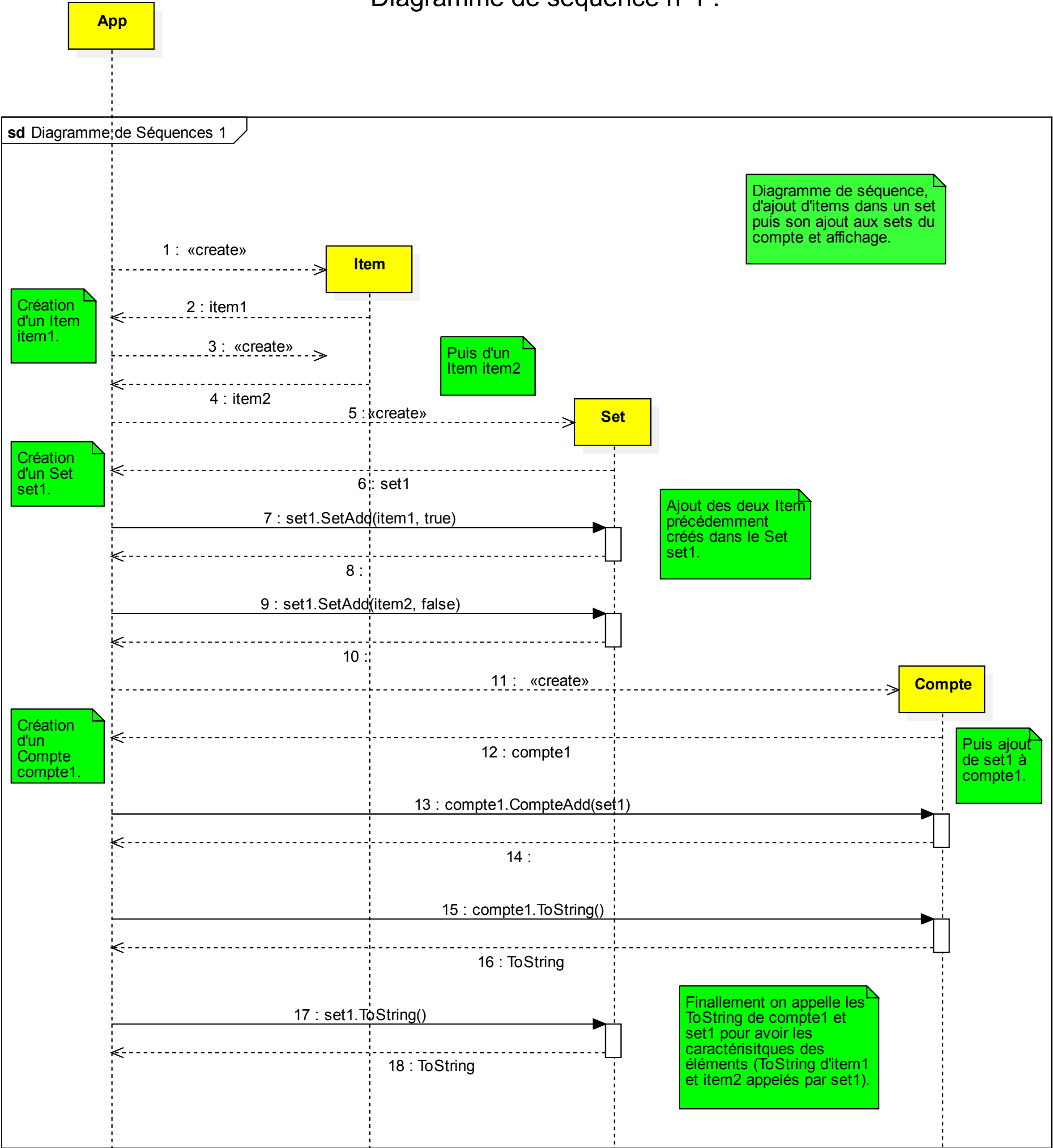
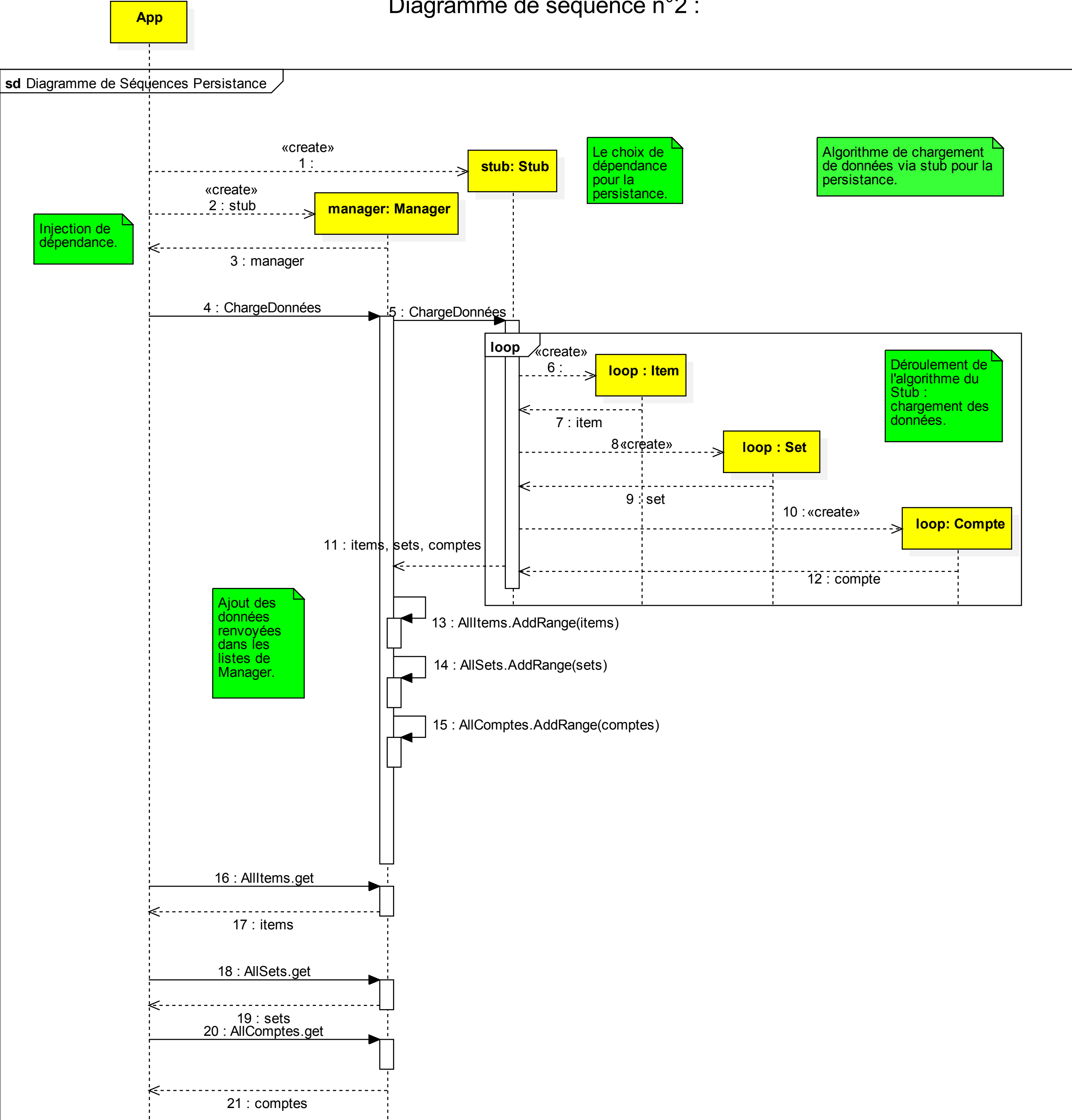


Diagramme de séquence n°2 :



Description écrite de l'architecture :

Diagramme de Classes :

Architecture :

Manager:

Manager est un peu le point central dans notre code : Il permet de centraliser les fonctions les plus importantes et utiles et surtout il nous permet de faire un lien entre le code en c# et le XAML (dans notre cas).

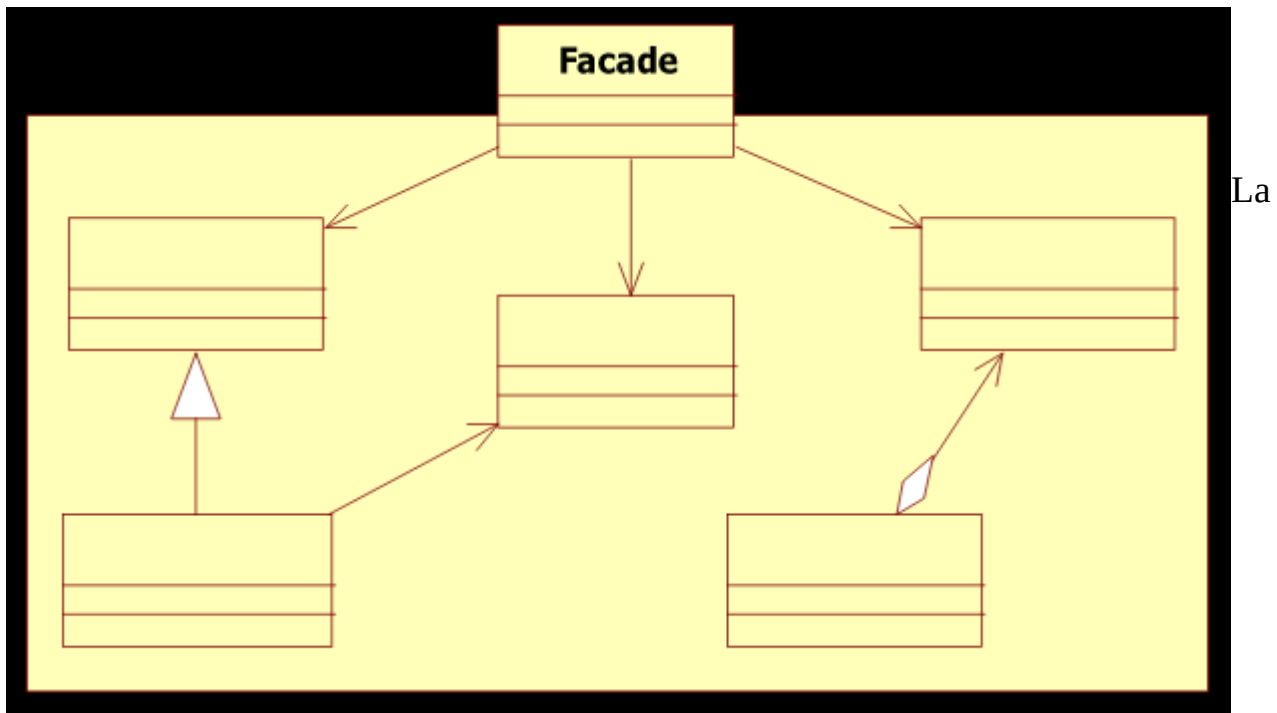
Pourquoi ?

Manager est ce qu'on appelle une "Façade"; une Façade est un intermédiaire entre un élément externe et un sous-système de classes qui propose une vue simple de ce sous-système car elle en cache la complexité pour seulement en montrer la façade (d'où son nom). Ainsi, les fonctionnalités sont limitées et donc recentrées sur les besoins du système externe et seulement sur le nécessaire. Depuis la façade on peut réaliser tout ce qui est normalement faisable par les classes du sous système. La Façade permet donc une indépendance du sous système en permettant au sous-système d'évoluer sans qu'elle ne change (à moins de changements majeurs bien sûr (ex : ajout d'une nouvelle bibliothèque de classes)). La façade est le point d'accès principal au sous système, ainsi, l'indépendance de celui-ci est permise car il ne dépend en rien du manager.

Dans notre cas, Manager permet une simplification du sous-système du côté de l'utilisateur, mais surtout un lien entre eux qui permet des interactions dans les deux sens. On garde ainsi l'indépendance du code vis à vis de la persistance car sans ça, on doit changer dans chaque classes plusieurs choses pour adapter le code (pour peu qu'on en ai énormément cela devient très lourd et long). Notre sous-système perdrait alors en ré-utilisabilité car il ne serait adapté qu'à ce changement et non à des possibles autres solutions.

Manager permet donc de lancer les processus de chargement de données et de sauvegarde, en gardant un impact très minime sur le reste du code (ajout de [DataContract] et [DataMember]). C'est lui qui dit à la Stratégie ce qu'on lui demande pour qu'elle puisse ensuite prendre le relais et appliquer la demande (on appelle cela une délégation).

Par exemple sur l'image d'en dessous on voit que la façade est en dehors du sous-système. Ce qui montre que la façade est le point d'entrée de celui-ci mais qu'elle reste quand même en dehors du sous-système.



Stratégie :

Qu'est-ce qu'une Stratégie ? Dit simplement, c'est une manière de faire des choses différemment en obtenant des résultats similaires/homogènes. En d'autres termes ; si on prend une classe avec un comportement spécifique et qui le traite de différentes façons, on peut alors la décomposer en plusieurs classes séparées ; l'ensemble est ce que l'on appelle une stratégie. Ainsi, la classe originale (qu'on appelle contexte) doit garder une référence vers la classe abstraite de la Stratégie (dans notre cas une injection de dépendance par constructeur ayant pour paramètre un élément de type de l'interface IPersistenceManager). Pour pouvoir utiliser des stratégies concrètes et interchangeable qui implémentent la classe abstraite de la stratégie, le contexte délègue alors à la stratégie la réalisation de la fonction de la classe concrète choisie.

Dans notre cas : la Stratégie provient de l'interface IPersistenceManager (c'est le contrat entre la stratégie et la façade) et des classes qui l'implémentent.

Soit :

(StratégiesConcrètes)

-Stub

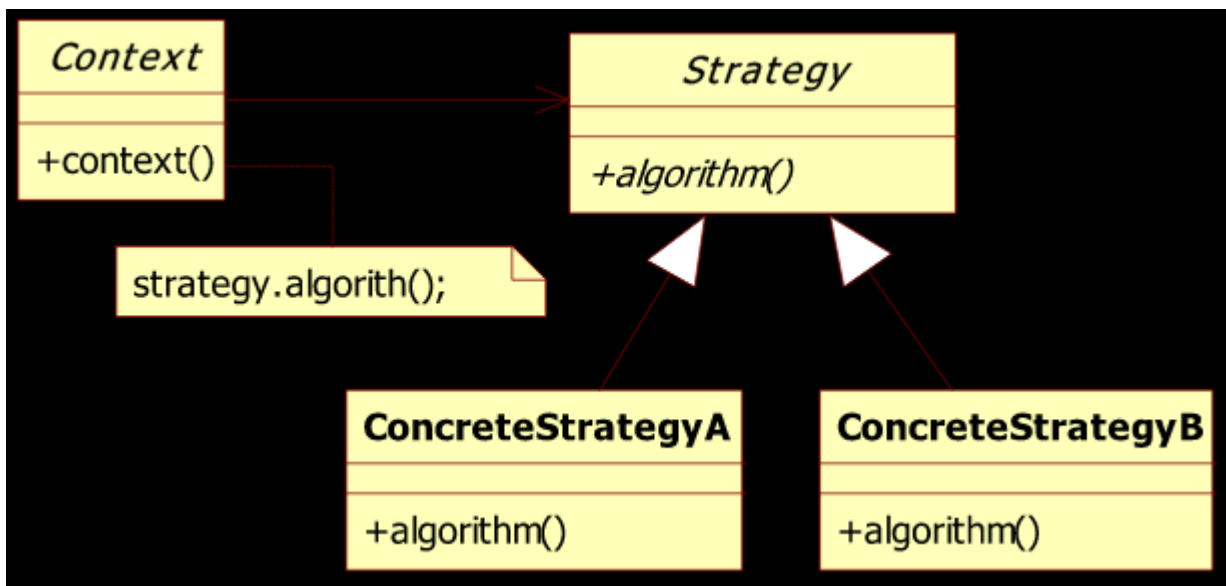
-DataContractPers

-DataContractPersJSON

On obtient ainsi une extensibilité de nos stratégies concrètes ; on peut en ajouter sans qu'aucuns problèmes n'apparaissent (tant qu'elles fonctionnent).

L'objectif de cette Stratégie est de nous permettre d'implémenter de la persistance de plusieurs manières différentes (cf au dessus) tout en gardant l'indépendance de notre Manager. Celui-ci n'est directement lié à aucune des classes concrètes de la stratégie et n'a pas besoin de changer pour chaque stratégie concrète (ce qui s'explique par l'interchangeabilité des classes concrètes de notre stratégie). La stratégie de persistance nous permet donc de sauvegarder dans un fichier ou charger des données depuis un fichier.

Le reste des classes n'a pas de forme particulièrement spécifique à un patron de conception donc nous en parlerons dans la partie suivante.



On voit bien sur cette image la représentation de ce qu'on a dit plus tôt : Un contexte qui possède un algorithme particulier. Tandis que cet algorithme appelle dans la stratégie, la méthode qu'il souhaite parmi celles disponibles pour réaliser ce qu'il veut accomplir.

Description des choix inter-classes :

Classe Item :

On possède dans notre modèle plusieurs énumérations (4 au total : Certification, Rarity, Color, et Specification).

On précise qu'on utilise des énumérations pour simplifier le code et le raccourcir mais surtout qu'on le fait car les données sont connues et finies (imposées par le jeu).

- Certification liste les certifications qui sont disponibles dans Rocket League.
- Rarity permet lui de lister les raretés des différents items (exemple de Common qui veut dire Commun).
- Color est la liste des couleurs disponibles (Malheureusement pas de **DarkSalmon**).
- Specification permet de savoir ce qu'est exactement l'Item (exemple des antennes (Antennas) ou des roues (Wheels)...).

Ces 4 énumérations sont utilisées pour créer des Item qui prennent en paramètres :

- Type (un élément de Spécification converti avec ToString pour avoir sa valeur string et non sa valeur short) On l'utilise car il y a plusieurs types d'items différents, ainsi on doit pouvoir les différencier par ce critère. / On veut pouvoir trier par Type.
- Rarity (un élément de Rarity converti avec ToString pour avoir sa valeur string et non sa valeur short) Idem que pour type, il existe plusieurs raretés. / On veut pouvoir trier par rareté.
- Color (un élément de Color converti avec ToString pour avoir sa valeur string et non sa valeur short). On l'utilise car un Item peut avoir une couleur et qu'il y a une quantité limitée disponible de couleurs à celles dans l'énumération Color. / On veut pouvoir trier par couleur.
- Certification (un élément de Certification converti avec ToString pour avoir sa valeur string et non sa valeur short). On l'utilise car un Item peut avoir une certification (ou non).
- Serie (un string correspondant au groupement de l'Item (c'est à dire des Items liés)). / On veut pouvoir trier par Serie.
- RealeaseDate (un DateTime correspondant à la date de sortie de l'Item).
- IsTradable (un bool qui permet de savoir si jamais l'Item est échangeable (certains ne le sont pas)).
- IsCertifiable (un bool qui permet de savoir si un Item peut avoir une Certification ou non (Il peut y avoir des Items non Certifiables et sans Certification MAIS il peut aussi y avoir des Items Certifiables sans Certification)).
- HasBluePrint (un bool qui permet de savoir si un Item possède un BluePrint ou non).
- Price (un int qui donne le prix de l'Item).

Item est notre classe la plus "basse" parmi les classes que l'on a créé car on avait besoin d'une classe pour rassembler les différentes caractéristiques des objets de Rocket League (cf ci-dessus).

Classe Set :

On possède aussi la classe Set qui possède :

- NumSet (un int qui représente son Numéro (pour le différencier des autres Set)). Au cas ou des Sets soient identiques.

- Name (un string qui a le nom du Set) pour que le set soit différenciable des autres.

- ReadOnlyDictionary<Item, int> SetItems (un dictionnaire en lecture seule (pour éviter que l'utilisateur casse tout avec des bêtises) que l'on restreint à l'appel des fonctions autorisées). On le crée à partir de setItems (un Dictionary<Item,int>) qu'on utilise dans le constructeur sous la forme suivante :

“SetItems = new ReadOnlyDictionary<Item, int>(setItems);”

L'encapsulation permet une sécurité quant au fonctionnement à toute épreuve (ou presque) du code).

Encapsuler le dictionnaire empêche l'utilisateur de l'utiliser, car celui-ci est en read only. « add » pourrait par exemple casser les Set en ajoutant plus d'un Item de chaque type (Or, clear, add ... ne sont pas implémentés par cette classe. On n'a pas de problème avec ça et l'utilisateur est donc limité dans ses actions sur le système). On utilise un dictionnaire pour avoir en valeur une correspondance avec l'énumération Spécification (un set ne peut pas avoir deux fois le même type d'item).

Set possède des méthodes bien à lui :

- SetAdd est une méthode particulière qu'on peut apparenter à la méthode Add des collections mais en bien plus poussée dans le cas présent :

Elle prend en paramètres :

- Un Item item (soit l'item à ajouter).

- Un bool Change (permettant de savoir, si un Item avec le même type existe déjà, si on veut écraser ou non).

Elle vérifie si le dictionnaire vaut null, au quel cas on stoppe la fonction. Si le nombre d'éléments vaut TAILLEMAX (variable statique qui vaut 14 (soit le nombre de Spécifications)) et que Change vaut false alors on ne peut rien faire et on stoppe la fonction.

Sinon, en fonction des différents cas elle agit différemment :

- Si le dictionnaire SetItems ne possède pas d'Item du type de celui qu'on veut ajouter, alors on ajoute forcément l'Item.

- Si il y a un Item avec le même type et que Change == false, alors on ne fait rien et on stoppe la fonction (dans le cas ou l'utilisateur fait une action non voulue).

- Enfin, si il y a un Item qui existe déjà avec le type de l'Item voulu et que Change == true, alors on cherche avec un « foreach » quel est l'item, puis on donne à un int « j » sa Value. On retire l'ancien Item et on ajoute le nouveau avec pour Value notre « j » (soit la Value de l'ancien Item) ; de manière grossière, on le remplace.

On utilise cette méthode pour gérer les ajouts, dans le cas où un ajout conduirait à un doublon de type, pour que un set ne contiennent jamais plus que un type d'Item à la fois, elle permet de gérer automatiquement selon ce que met l'utilisateur, les différents cas de figures énoncés plus haut et on a donc pensé qu'elle était adaptée à la situation.

-SetNumSet permet de définir le numéro d'un Set (après l'avoir recherché grâce à LINQ) en fonction de si celui-ci est déjà attribué à un autre Set (que l'on trouve dans une liste). On trie ensuite la liste. On utilise cette méthode pour éviter un doublon lors de la modification du numéro d'un set.

On veut éviter le cas du doublon pour pouvoir se retrouver dans les Set et avoir une unicité certaine.

-DeleteItemFromSet (prend en paramètre un Item et l'enlève du set si il y est). La méthode est utilisée pour retirer un Item d'un set (elle est utilisée par le manager dans sa fonction DeleteItem).

Classe CompteGuest et Compte:

CompteGuest est une classe très simple faite spécialement pour le mode « guest » de l'application. Il possède un Login (un string pour le pseudo de l'utilisateur) donné par le constructeur ("Guest") puis l'on crée aussi un Set SetActif (avec pour nom SetActif). Son ToString renvoie que le compte disparaîtra car CompteGuest n'est pas fait pour durer (ce qui est la raison de pourquoi cette classe est si courte (et porte ce nom)).

On a décidé de créer cette classe car pour un Guest (un invité) sur une application ou même un site web, il n'y a jamais trop de fonctionnalités.
Pourquoi ?

Car un CompteGuest n'est pas enregistré dans la base de données, il est détruit une fois qu'on ne l'utilise plus à l'instar des Compte classiques qu'on enregistre. C'est donc car il est éphémère que CompteGuest est plutôt vide ...

Compte hérite de CompteGuest donc de sa propriété Login (modifiable dans Compte et non prédéfini) mais ajoute aussi :

Pour éviter de créer 2 classes distinctes alors qu'elle ont toutes deux des points communs, on a implémenter un héritage qui permet de raccourcir le code et d'éviter l'effet "déjà vu" sur la classe Compte.

-Password (un string qui contient le mot de passe du compte). Nécessaire pour s'identifier ensuite (tout comme le Login de Compte).

-Email (un string qui contient l'adresse Email du compte). Si on a besoin de contacter l'utilisateur en cas de problème ou suite à d'autres actions.

-ReadOnlyCollection<Set> Sets (Même idée que pour SetItems de Set mais ici pour ranger des sets et encapsuler la collection (qui est une liste définie par Sets = new ReadOnlyCollection<Set>(sets); (sets qui est de type : List<Set> sets))). Similaire à notre ReadOnlyDictionnaire, on utilise ici une ReadOnlyCollection pour encapsuler une collection

(ici une liste comme précisé) avec toujours pour but d'empêcher l'utilisateur de casser le code en faisant des choses qu'on ne voudrait pas (clear, add ... ne sont pas implémentée par cette classe donc on n'a pas de problème ici aussi). Le but comme énoncé est de limiter l'impact de l'utilisateur sur le code.

Compte ajoute :

La méthode `CompteAdd` qui prend un `Set` et qui si celui-ci n'est pas dans la liste des sets du compte est ajouté. Sinon on renvoie la fonction. On veut que chaque `Set` n'existe qu'une fois dans la liste de `Compte`.

-La méthode `CompteSetNumSet` permet de faire appel à `SetNumSet` dans `Compte` en utilisant `Sets` comme liste de `Set`. On utilise l'appel à la fonction de `SetNumSet` pour éviter d'avoir un `NumSet` avec un setter en public et donc la possibilité que l'utilisateur fasse des siennes.

-`DeleteSetFromCompte` (prend en paramètre un `Compte` et un `Set` puis enlève ce dernier si celui-ci existe déjà dans le `Compte`). La méthode est utilisée pour retirer un `Set` d'un compte (elle est utilisée par le manager pour sa fonction `DeleteSet`).

`Compte` représente l'utilisateur, c'est lui qui rassemble ses informations personnelles ainsi que ses créations (ses sets qui sont eux mêmes remplis d'Items), on l'utilise donc pour permettre la création d'un compte et d'effectuer des modifications ainsi que de supprimer ce dernier.

Manager :

Manager possède en attributs :

-`AllItems` (une `ReadOnlyObservableCollection<Item>` pour pouvoir stocker des données qui vont agir directement sur le XAML car la modification ou suppression de l'une de ces données sera apparente sur le XAML). Elle possède une `ObservableCollection<Item>` pour qu'on puisse interagir avec elle qui stocke tous les items créés (utilisés ou non).

-`AllSets` (une `ReadOnlyObservableCollection<Set>` pour pouvoir stocker des données qui vont agir directement sur le XAML car la modification ou suppression de l'une de ces données sera apparente sur le XAML). Elle possède une `ObservableCollection<Item>` pour qu'on puisse interagir avec elle qui stocke tous les sets créés.

-`AllComptes` (une `ReadOnlyObservableCollection<Compte>` pour pouvoir stocker des données qui vont agir directement sur le XAML car la modification ou suppression de l'une de ces données sera apparente sur le XAML). Elle possède une `ObservableCollection<Item>` pour qu'on puisse interagir avec elle qui stocke tous les comptes existants

-`Persistence` (de type `IPersistenceManager`, cet attribut est celui qui permet la sauvegarde et le chargement des données)

Manager possède certaines fonctions comme :

- DeleteCompte (prend un Compte et détruit les références qu'il possède).
- DeleteSet (prend un Set, un Compte, un bool Everything et détruit les références que le Set possède (si Everything vaut true, on le détruit partout, sinon on le détruit dans le compte donné en paramètre)).
- DeleteItem (prend un Item et détruit les références que l'Item possède).

Ces méthodes servent à "supprimer" ce qu'ils prennent car en supprimant les références qu'ils ont, le garbage collector viendra les détruire). On veut les utiliser si on ne veut plus de tel ou tel chose par exemple.

-CreateCompte / CreateSet / CreateItem (On comprends ce qu'elles font.) Elles sont des fonctions simples qui servent juste à simplifier la vie pour le côté utile.

-ChargeDonnées (charge les données grâce à l'attribut Persistance de type IPersistenceManager depuis la stratégie).

-SauvegardeDonnées (sauvegarde les données grâce à l'attribut Persistance de type IPersistenceManager depuis la stratégie).

Mais on à aussi des fonctions (pas encore implémentée ni écrites) :

- ExportSet qui permettra de partager ses créations à d'autres utilisateurs
- ImportSet qui permettra de recevoir une création de quelqu'un d'autre

(Qu'on implémentera si on à le temps de le faire)

-(Peut-être d'autres)

Le Manager possédera aussi des fonctions pour la passerelle entre les classes et le XAML .

Stratégie :

Notre stratégie contient plusieurs classes dans nos bibliothèques de classes StubLib et DataContractPersistance.

Dans StubLib :

On trouve la classe Stub (une classe de persistance un peu en carton) car :

Elle possède des méthodes qu'elle tient de l'interface IPersistenceManager (SauvegarderDonnées et ChargeDonnées).

Et d'autres méthodes propres à elle :

- ChargeItems
- ChargeSets
- ChargeComptes

Qui, respectivement :

Crée des Item, les mets dans une liste et la renvoie.

Crée des Item, des Set, met des Item dans les Set et les Set dans une liste de Set et la renvoie.

Crée des Item, des Set, des Compte, met des Item dans les Set et des Set dans les Compte, met les Compte dans la liste de Compte et la renvoie.

Ces méthodes sont appelées par ChargeDonnées qui va renvoyer ces listes à l'appelant.

Pourquoi le Stub est une classe « en carton » ?

- Elle ne charge que les mêmes données, ne sauvegarde rien.

Dans DataContractPersistance :

On trouve plusieurs classes : DataContractPers, DataContractPersJSON et DataToPersist.

DataToPersist : Une classe qui possède en attribut une liste d'Item, une liste de Set et une liste de Compte. Elle sert à éviter de compliquer le code pour la persistance car on l'utilise pour persister nos 3 listes.

DataContractPers : Elle possède 3 attributs normaux et 2 attributs calculés.

- RelativePath qui est le chemin relatif du fichier.
- FileName qui est le nom du fichier.
- Serializer qui nous permet de sérialiser le fichier en gardant les références aux objets (éviter les doublons) au format XML (de type DataContractSerializer).
- FilePath qui est une propriété calculée permettant de remonter au dossier voulu depuis le dossier actuel.
- PersFile qui permet d'avoir l'emplacement et le Nom du Fichier pour pouvoir l'éditer ou le lire. Elle implémente les méthodes de l'interface IPersistanceManager soit :

- ChargeDonnées vérifie que le dossier existe, sinon elle le crée. On crée un DataToPersist data auquel on donnera les valeurs de ce qui est lu avec Serializer après avoir ouvert un chemin dans le fichier. Puis on retourne les listes de data.

-SauvegarderDonnées prend en argument un IEnumerable d'Item, de Set et de Compte. Vérifie que le dossier existe, sinon elle le crée. On crée un DataToPersist data auquel on donne les valeurs des IEnumerable passés en arguments. On crée des settings d'indentation. Puis on utilise un TextWriter dans le fichier grâce à PersFile qui nous permet d'utiliser un XmlWriter writer pour pouvoir modifier/créer un fichier XML grâce à writer et data.

DataContractPersJSON : dérive de DataContractPers et possède donc ses attributs et méthodes. Elle modifie les attributs non-calculés pour les adapter au format JSON dans son constructeur (Serializer devient un DataContractJsonSerialier pour palier à ça).

Elle modifie la méthode de sauvegarde pour qu'elle ne prenne plus le format XML et qu'elle lui soit ainsi adapté : on à toujours data notre DataToPersist. Mais on utilise un simple canal (writer) qui crée un fichier en fonction de PersFile tandis que Serializer permet l'écriture en fonction de writer et data.

On utilise ces fonctions et leur contenu pour pouvoir rendre nos Item, Set et Compte persistants afin de les réutiliser.

Diagramme de Paquetages :

Dans notre Appli, on possède une bibliothèque de Classes (où l'on trouve nos classes) dont dépendent nos 5 Tests :

- Test_Items nous permet de vérifier si un Item se forme comme on le souhaite.
- Test_Set permet de vérifier si un Set se forme correctement, puis si il peut contenir des Item et si ses méthodes fonctionnent correctement.
- Test_Compte permet de vérifier ce qu'il se passe pour un Compte et un CompteGuest et si les méthodes de Compte fonctionnent correctement.
- Test_PersXML vérifie qu'on sauvegarde/charge correctement des données dans un fichier XML.
- Test_PersJSON vérifie qu'on sauvegarde/charge correctement des données dans un fichier JSON.

Diagrammes de Séquences :

Le premier diagramme de séquence représente la création de 2 items ; item1 et item2 (on pourrait en rajouter), leur ajout, la création d'un Set set1, dans set1 avec la fonction SetAdd (expliquée plus haut). Ensuite on crée un Compte compte1 puis on ajoute set1 dans le Sets de compte1 avec CompteAdd (expliquée plus haut). Et enfin on appelle le ToString de compte1 et de set1 qui donnent respectivement les informations du compte ainsi que celles du set qui contiennent celles des items.

Le second diagramme de séquence représente le chargement depuis un stub appelé par le Manager. On appelle, après la création du Manager, la fonction ChargeDonnée de celui-ci qui appelle la fonction éponyme du stub (dépendance choisie). Elle crée des Item, des Set et des Comptes puis renvoie une liste de ceux-ci. On récupère le contenu et l'ajoute aux listes du Manager.