

Présentation du projet « SearchBars »

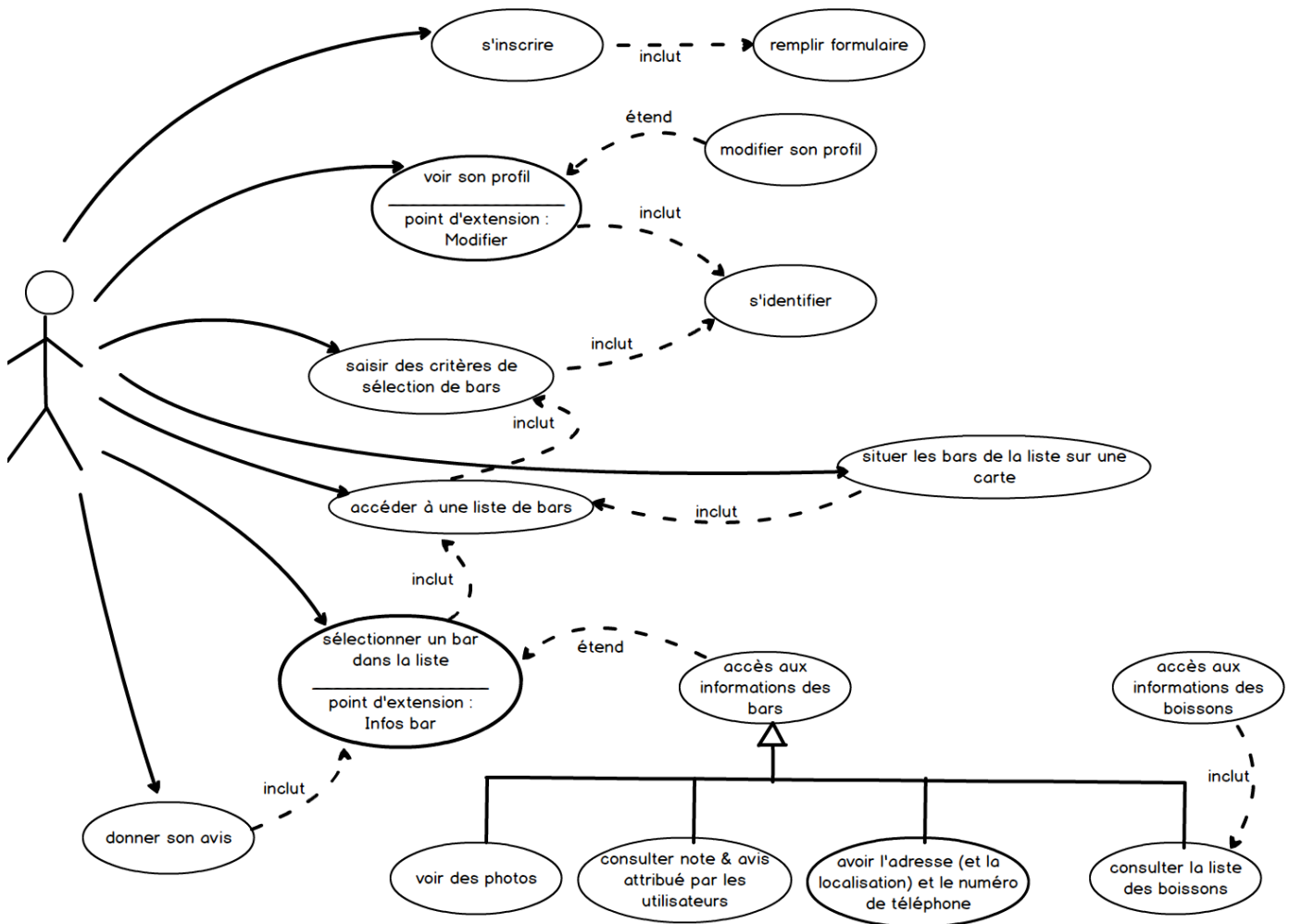
I] Description du contexte

Les amateurs de soirées dans les bars ou encore de la bière sur la terrasse avec leurs amis, n'auront bientôt plus de question à se poser sur quel bar choisir pour leur moment de détente !

Prenons une personne Lambda qui doit se rendre dans une ville qu'elle ne connaît pas, pour siroter une petite bière avec des amis... Le problème c'est que cette personne ne connaît pas les bistrots du coin et se retrouve dans la plus grande incertitude pour trouver dans quel bar emmener ses amis... C'est à cet instant que notre application SearchBars fera son apparition ! Nous l'avons conçue afin que quiconque aimerait aller boire un coup dans un bar puisse le faire en toute rapidité et simplicité. Alors comment nous y sommes pris ? C'est simple notre application propose de remplir un petit formulaire pour que notre dénommé Lambda trouve le bar qu'il souhaite c'est-à-dire ce que celui-ci sert, sa popularité et enfin sa localisation ! Après ce petit formulaire, SearchBars localisera tous les bars qui correspondent à la recherche et vous les localisera sur une carte ! Notre Lambda n'aura plus qu'à choisir son bar avec les petites photos, la carte, la popularité et la localisation... Après sa soirée avec ses amis il pourra, s'il le souhaite, laisser un avis sur le bar afin de pouvoir mieux aiguiller les futurs utilisateurs de notre application SearchBars.

Un autre cas d'utilisation peut être celui d'un chef d'entreprise qui est déplacement ; après une dure journée l'envie lui prend de boire une bière avec ses collègues. Le problème étant qu'ils ne connaissent absolument rien à cette ville. Ainsi l'application SearchBars pourra, en fonction de sa recherche, trouver le bar qui leur convient. Ils pourront rentrer la ville dans laquelle ils se trouvent, puis le type de boisson qu'ils recherchent (ici de la bière) et s'ils veulent que le bar ait une bonne réputation. Il se peut aussi que ces hommes n'aient pas mangé ainsi ils pourront cocher une option pour que le bar fasse aussi de la restauration. Ainsi, ils seront renvoyés sur une liste de bars correspondants à leur recherche accompagnée d'une carte de la ville avec tous les bars recherchés localisés. Cela leur facilitera grandement la vie et leur fera gagner du temps, puisque le temps c'est de l'argent. Ils peuvent aussi s'ils le souhaitent consulter les avis à propos de ce bar. Après s'être rendu dans un bar ils pourront à leur tour, donner leur avis à propos de ce dernier.

II] Diagramme de cas



Description :

Pour pouvoir accéder aux fonctionnalités de l'application, l'utilisateur doit s'identifier.

Pour pouvoir s'identifier, il doit posséder un compte. Ce compte est créé lors de l'inscription de l'utilisateur. Pour réaliser cette inscription, l'utilisateur doit remplir un formulaire comportant différents champs dont certains sont obligatoires et d'autres non.

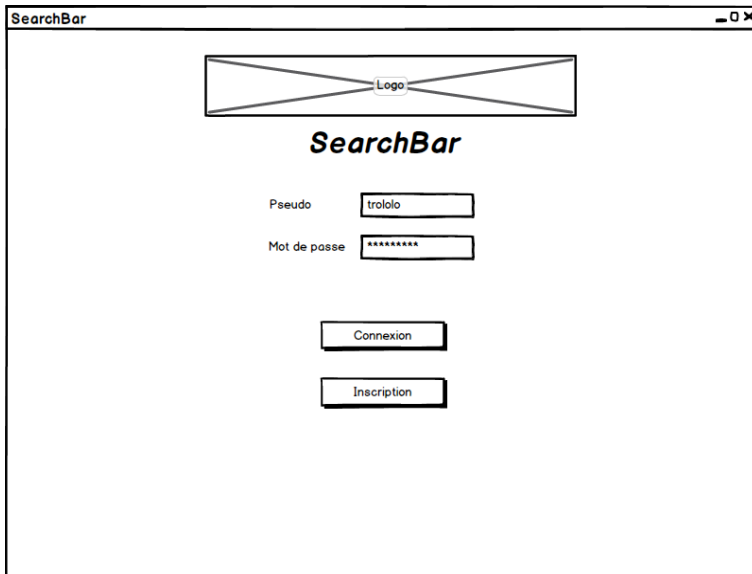
Une fois inscrit, il a donc la possibilité de s'identifier à l'aide de son pseudo et mot de passe choisi précédemment.

Il peut alors choisir de voir son profil, il pourra voir toutes les informations qu'il a saisies lors de son inscription. Depuis cette page où sont affichées ses informations personnelles, l'utilisateur peut décider de modifier son profil, il pourra alors modifier certains champs de son profil, supprimer des champs optionnels saisis auparavant ou ajouter des informations aux champs qu'il n'avait pas remplis.

Il peut aussi, suite à son identification, saisir différents critères de recherche permettant de sélectionner uniquement les bars répondants aux critères que l'utilisateur saisit. Cela lui permettra d'avoir accès à une liste de tous les bars qu'il pourra situer sur une carte de la ville grâce à des pointeurs. Dans cette liste de bars, il aura la possibilité d'en sélectionner un. A partir de cette sélection, il pourra soit avoir accès à diverses informations sur le bar, telles que voir des photos représentant le bar, consulter les notes et commentaires laissés par d'autres utilisateurs sur ce bar, avoir l'adresse et le numéro de téléphone du bar ainsi que sa localisation sur une carte, mais aussi, consulter la liste des boissons servies par ce bar. A partir de cette dernière, il pourra avoir accès à diverses informations de la boisson sélectionnée telles que son nom, son prix, son volume (si boisson alcoolisée), sa marque, et plus si c'est des boissons telles que les vins.

Enfin l'utilisateur pourra lui aussi donner son avis sur le bar qu'il a sélectionné dans la liste. Mais attention cet avis sera unique, puisqu'un seul avis est permis par utilisateur pour un bar.

III] Sketchs



SearchBar

Logo

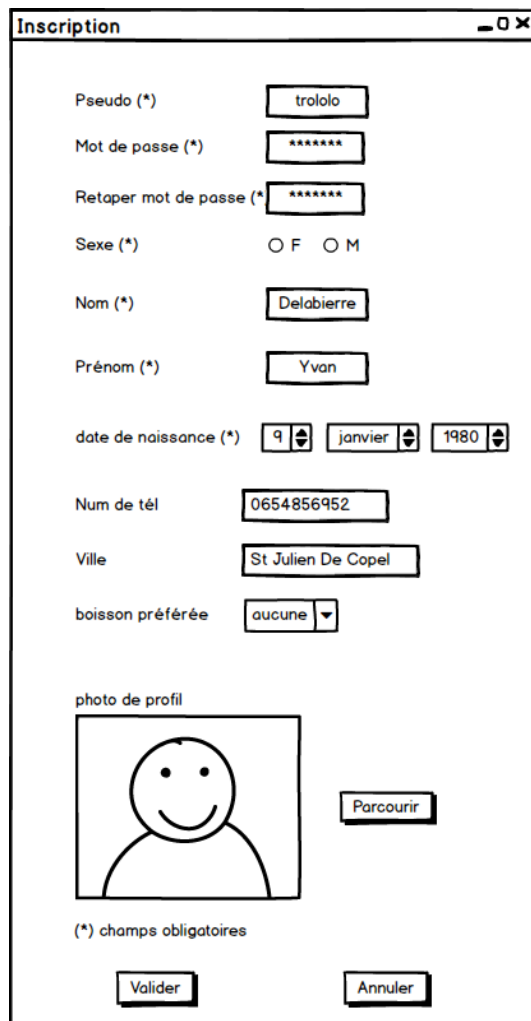
SearchBar

Pseudo

Mot de passe

Sketch 1 :

Page de connexion permettant à un utilisateur de s'authentifier (avec son pseudo et son mot de passe), ou à un nouvel utilisateur de s'inscrire.



Inscription

Pseudo (*)

Mot de passe (*)

Retaper mot de passe (*)

Sexe (*) ☐ F ☐ M

Nom (*)

Prénom (*)

date de naissance (*)

Num de tél

Ville

boisson préférée

photo de profil

(*) champs obligatoires

Sketch 2 :

Formulaire d'inscription à remplir pour un nouvel utilisateur : (champs facultatifs : ville, numéro de téléphone et boisson préférée), une photo de profil sélectionnée par défaut mais l'utilisateur pourra s'il le souhaite sélectionner une photo personnelle dans son ordinateur. Si l'utilisateur n'est pas majeur il ne pourra pas s'inscrire.

Sketch 3 :

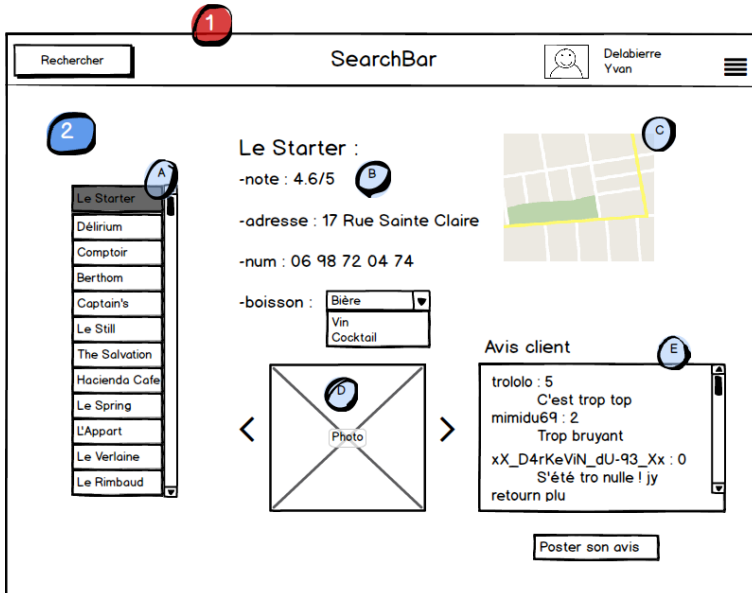
- 1) Entête qui sera commune à toutes les pages après que l'utilisateur soit identifié.
 - A. Bouton « rechercher » qui nous permet d'afficher la page du formulaire de recherche (elle-même pour cette page).
 - B. La photo de profil suivie du nom et du prénom.
 - C. Un bouton « menu » qui aura des fonctions telles que « se déconnecter », « voir son profil »...
- 2) Corps de la page, on y trouve un formulaire à remplir afin de sélectionner un bar avec le plus de précision possible (ville, restauration, boissons servies, note), à savoir que la ville est un critère non facultatif.
 - A. Bouton « + » permettant de rajouter une ListBox pour éventuellement saisir une boisson en plus.

Sketch 4 :

- 1) Entête.
- 2) Résultat d'une recherche après le lancement de la recherche.
 - A. ListBox avec tous les bars remplissant les critères saisis précédemment.
 - B. Nom de la ville sélectionnée suivi d'une carte dynamique (Bing map) où chaque bar présent dans la ListBox sera marqué sur la carte.

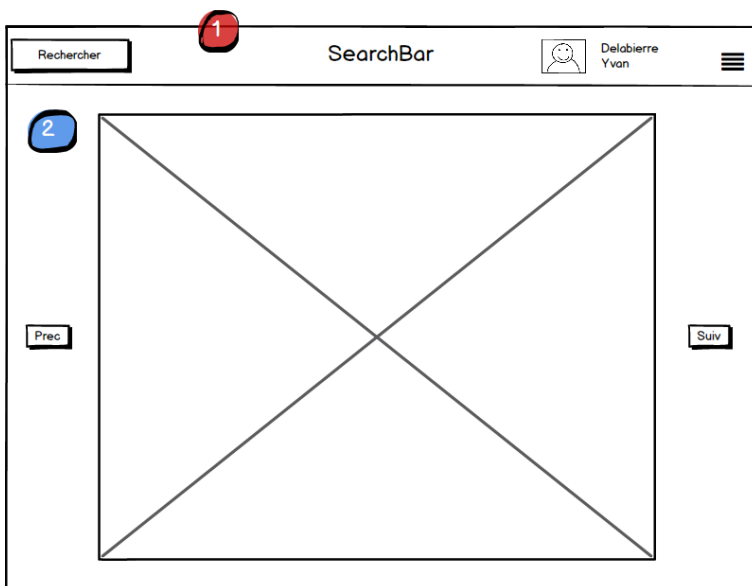
Sketch 5 :

- 1) Entête.
- 2) Page où l'on pourra trouver toutes les informations sur un bar sélectionné précédemment.
 - A. ListBox des bars sélectionnés par la recherche
 - B. Informations de base : nom, note moyenne, adresse, liste des boissons que sert ce bar. A savoir que l'on peut cliquer sur une boisson afin d'en connaître sa composition (exemple pour un cocktail on y trouvera ses ingrédients).
 - C. Carte dynamique (Bing Map), on aura une visualisation du bar sur la carte (avec zoom approprié) afin de le localiser plus facilement.
 - D. Galerie photo du bar, on pourra les faire défiler avec les flèches « prec » et « suiv ». On pourra cliquer sur la photo afin de la voir en plus grand format (Cf. voir sketch suivant).
 - E. Zone où se trouve les avis des utilisateurs, ces avis sont constitués du pseudo de la personne qui laisse l'avis, la note laissée et d'une description facultative. On y trouvera aussi un bouton pour que l'utilisateur puisse saisir son avis (Cf. sketch 7).



Sketch 6 :

- 1) Entête.
- 2) Zone où on trouvera la photo en plus grand format, après avoir cliqué sur la photo (Cf. sketch 5 2) D.). On y retrouve les boutons « suiv » et « prec » pour défiler les photos en un plus grand format.



Poster un avis

Avis

Note ★ ★ ★ ☆ ☆

Commentaires

C'est super sympa mais trop bruyant

Poster Annuler

Sketch 7 :


Fenêtre pop-up qui permet à un utilisateur de saisir un avis, à savoir une note (de 1 à 5) et de joindre une description facultative.

Rechercher 1

SearchBar

Delabierre Yvan

2



Nom : Delabierre
 Prenom : Yvan
 Sexe : Homme
 Age : 36 ans
 DDN : 9/01/1980
 Ville : St Julien de Copel
 Boisson Préférée : aucune

Sketch 8 :

- 1) Entête, on trouvera dans le bouton « menu » des informations administratives différentes comme « modifier son compte ».
- 2) Espace on l'on trouvera toutes les informations saisies lors de l'inscription de l'utilisateur (photo de profil, nom, prénom, pseudo, sexe, âge, date de naissance, ville, boisson préférée).

Vérification du mot de passe

Veuillez saisir votre mot de passe

Valider Annuler

Sketch 9 :

Fenêtre qui redemande de saisir le mot de passe permettant de vérifier que c'est bien le propriétaire qui veut modifier son profil.

Modification du profil

Pseudo (*) trololo

Mot de passe (*) *****

Retaper mot de passe (*) *****

Sexe (*) ☐ F ☐ M

Nom (*) Delabierre

Prénom (*) Yvan

date de naissance (*) 9 janvier 1980

Num de tél 0758965854

Ville St Julien De Copel

boisson préférée bière

photo de profil

Parcourir

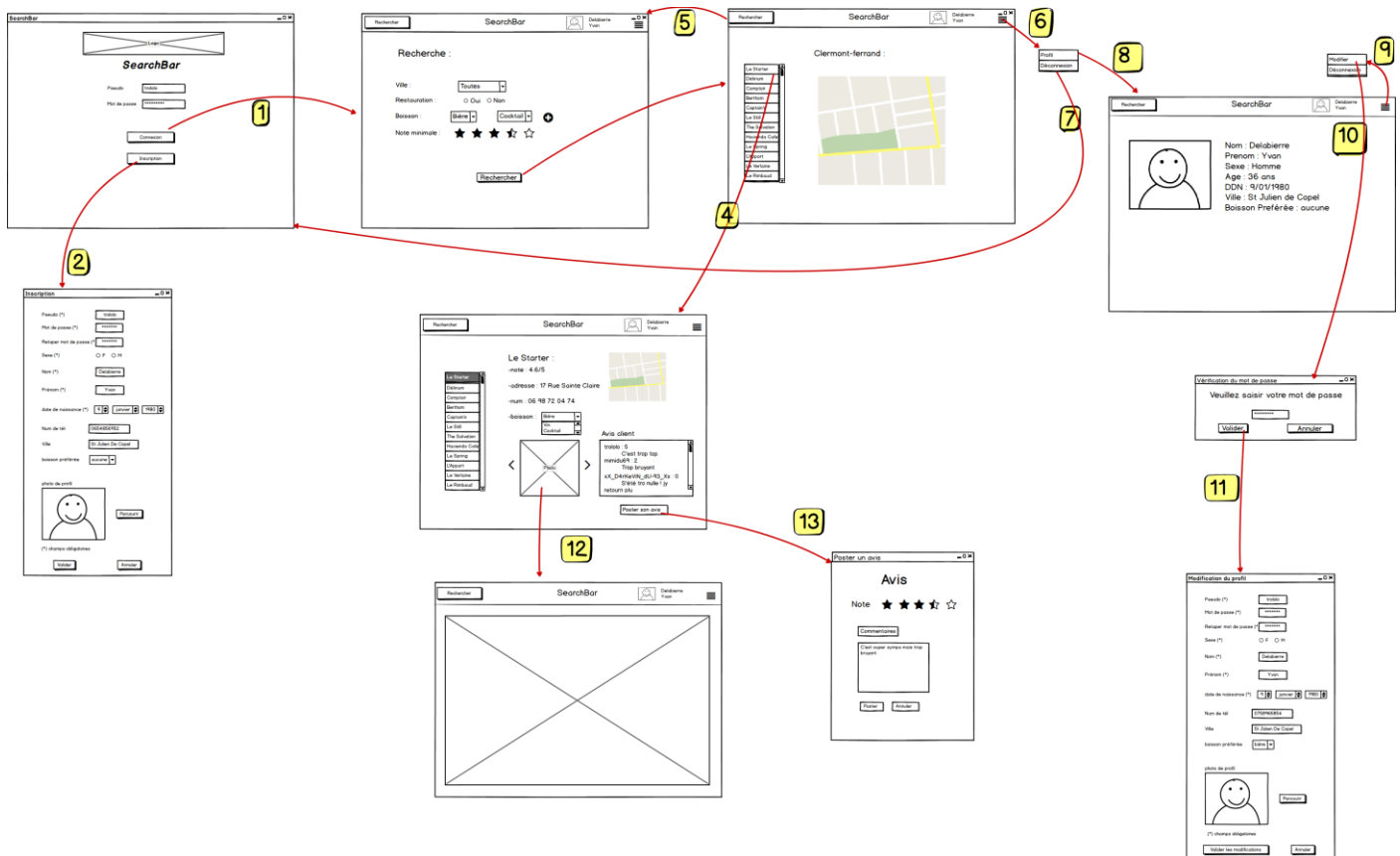
(*) champs obligatoires

Valider les modifications Annuler

Sketch 10 :

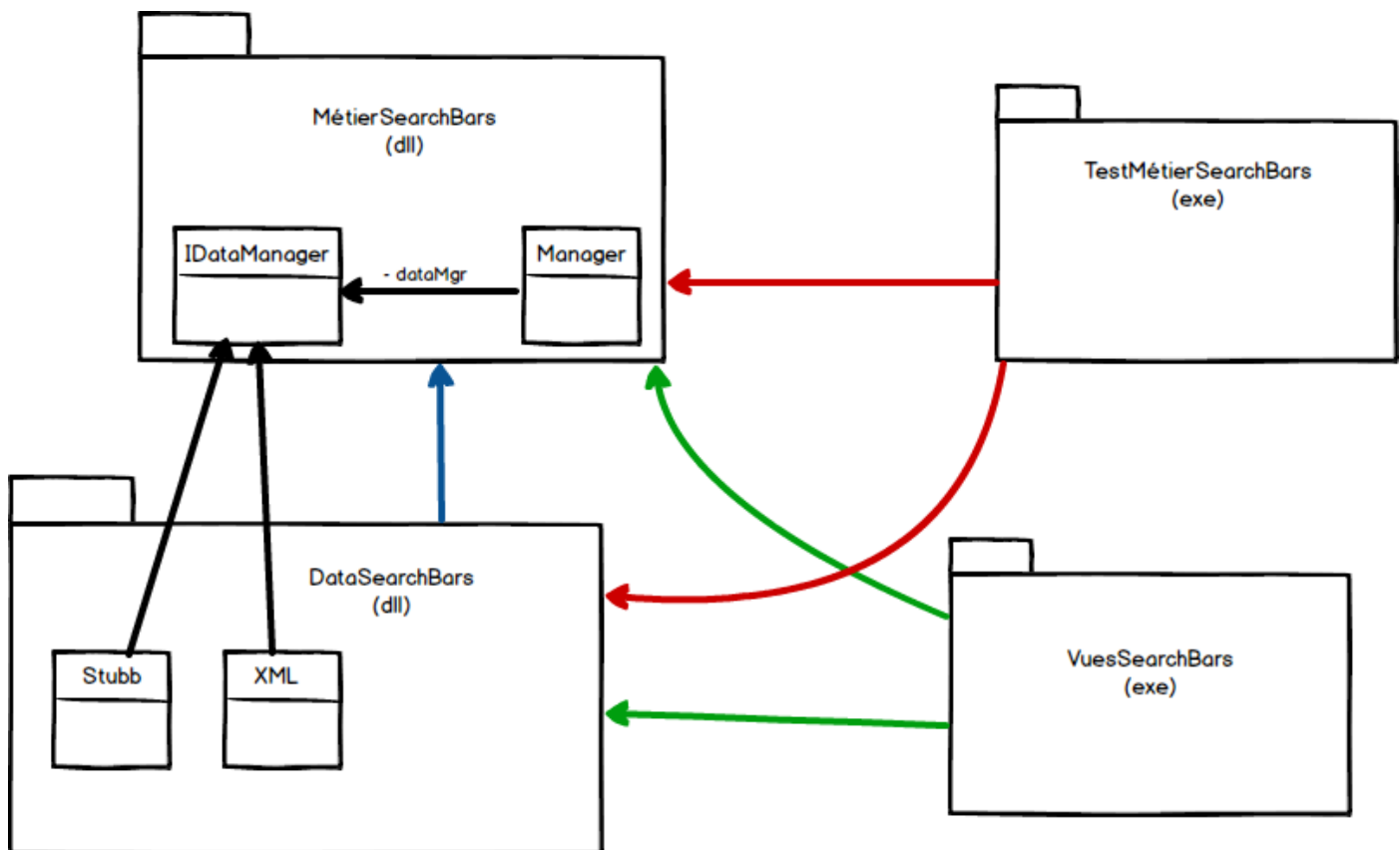
On retrouve le formulaire d'inscription à la différence que tous les champs connus du profil seront pré-remplis. L'utilisateur pourra à sa guise modifier ses informations personnelles à savoir qu'après validation les informations devront être correct (exemple : champs de mot de passe identique, ainsi que pseudo inexistant ou âge toujours supérieur à 18).

IV] StoryBoard



- 1) Appui sur bouton « connexion » : envoi sur la page de recherche (sélection des critères)
- 2) Appui sur bouton « s'inscrire » : envoi sur le formulaire d'inscription (après validation ou annulation, renvoi sur page de connexion)
- 3) Appui sur bouton « rechercher » : valide la recherche et envoi sur la page de la ville et avec les bars répondant aux critères
- 4) Clic sur un bar : envoi sur la page de détail du bar
- 5) Appui sur bouton « rechercher » en haut de page : renvoi sur la page de recherche avec sélection des critères
- 6) Appui sur bouton « menu » : affiche un petit menu avec mon compte ou déconnexion
- 7) Appui sur « déconnexion » dans le menu : renvoi sur la page de connexion
- 8) Appui sur « mon compte » dans le menu : envoi sur la page du compte avec les infos personnelles affichées
- 9) Appui sur bouton « menu » : affiche le menu avec « modifier profil » à la place de « mon compte »
- 10) Appui sur bouton « modifier profil » dans menu : envoi sur une page de vérification du mot de passe pour être sûr que c'est bien la bonne personne qui souhaite modifier les infos
- 11) Appui sur « valider » : envoi sur la page de modification de profil (« annuler » renvoie sur la page « mon compte »)
- 12) Clic sur une photo : affiche la photo en grand format
- 13) Clic sur « poster un avis » : envoi sur la page permettant de laisser un avis

V] Diagramme de paquetage



Description :

- La couche contenant le métier, c'est-à-dire la partie logique de l'application est contenue dans l'assemblage MétierSearchBars (c'est une bibliothèque de classes : .dll)
- La couche contenant les vues, c'est-à-dire les différentes fenêtres de l'application, toute la partie graphique, est contenue dans l'assemblage VuesSearchBars (c'est une application WPF : .exe)
- La couche contenant les tests, c'est-à-dire les classes qui permettent de tester les différentes fonctionnalités du métier dans la console Windows, est contenue dans l'assemblage TestMétierSearchBars (c'est une application console : .exe)
- La couche contenant toute la partie persistance, c'est-à-dire les classes permettant de faire subsister les données dans le temps (par l'intermédiaire de fichiers, de bases de données, etc), est contenue dans l'assemblage DataSearchBars (c'est une bibliothèque de classes : .dll)

Pour pouvoir afficher les données de l'application l'assemblage des vues doit avoir une référence à l'assemblage de persistance. Il doit aussi avoir une référence à l'assemblage métier pour pouvoir utiliser les fonctionnalités logiques du métier.

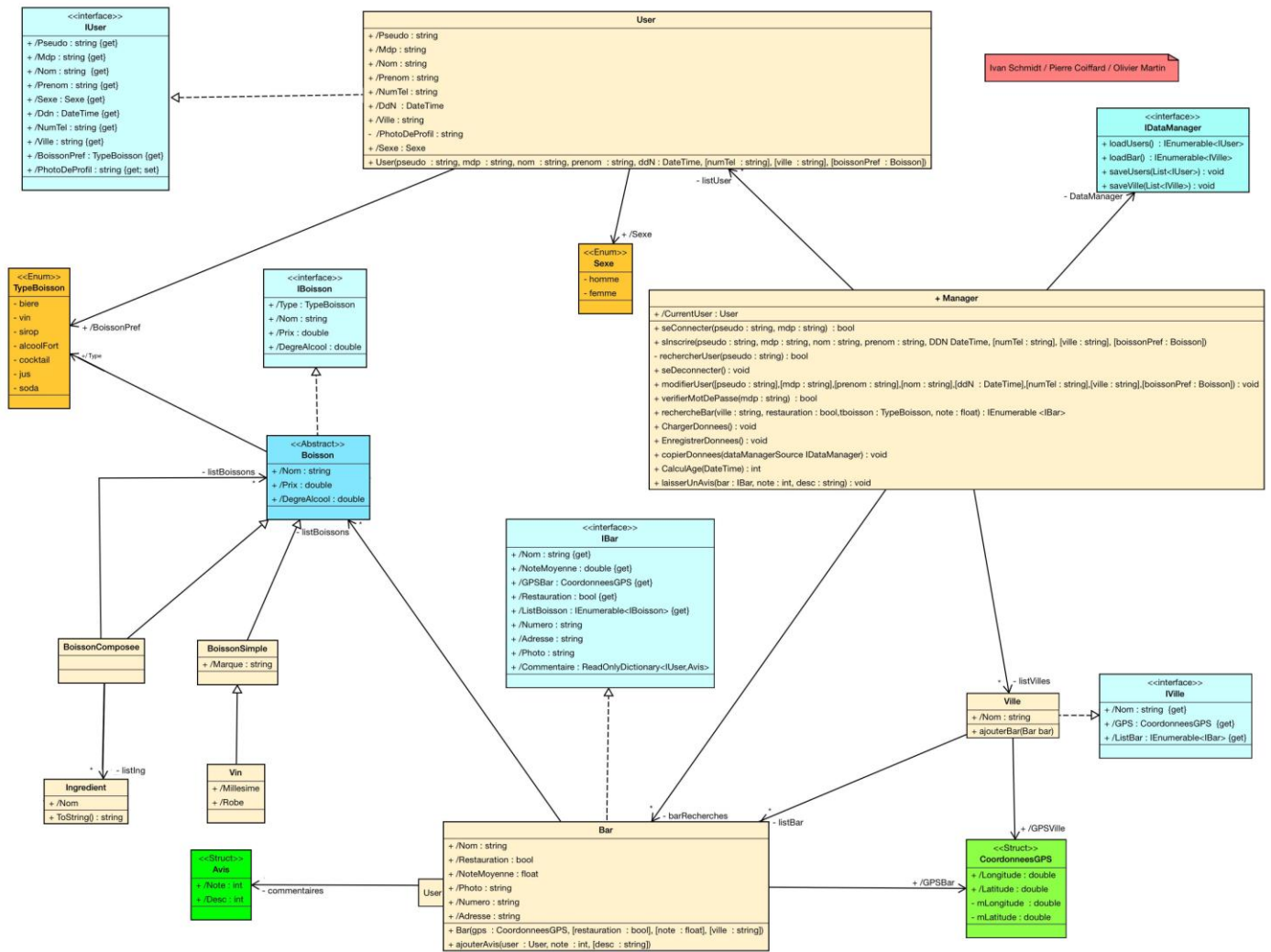
L'assemblage des tests doit lui aussi avoir une référence au métier car son but étant de tester le métier il doit pouvoir utiliser les fonctionnalités du métier. Il doit aussi avoir en référence l'assemblage de persistance pour avoir accès aux données de l'appli et pour pouvoir tester correctement le métier.

L'assemblage métier possède une classe **Manager** qui est la façade (utilisation du patron de conception « Facade ») du métier. Cette classe possède un attribut privé de type **IDatamanager**. Cette interface impose un contrat permettant la sauvegarde et la lecture des données. Cette interface doit être implémentée par les classes de persistance de l'assemblage **DataSearchBars**. On utilise ici le patron de conception « Strategy »

Pour pouvoir effectuer cela l'assemblage de persistance doit posséder une référence au métier : pour que ces classes (**Stub** et **XML**) puissent implémenter l'interface **IDatamanager**.

VII] Diagrammes de classes

- Ensemble du Métier

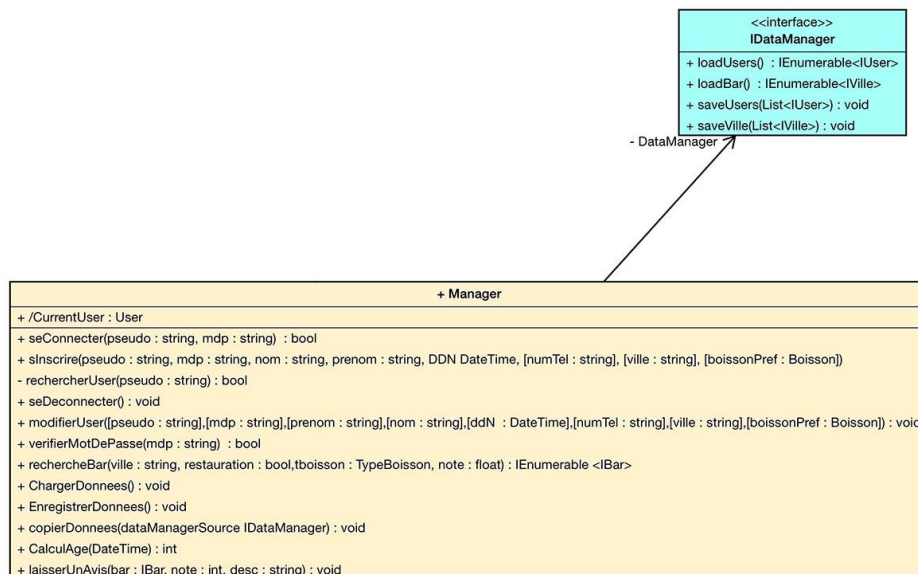


Ce diagramme de classe représente l'ensemble du métier. Comme nous pouvons le voir il est constitué de classes (roses) dont la majorité sont internes à l'exception d'une : le Manager.

Il y a aussi plusieurs structures (CoordonneesGPS, Avis et Ingredient (oublié sur le diagramme)). On a choisi de faire des structures plutôt que des classes puisqu'elle n'ont seulement un ou 2 types primitifs et n'ont pas une utilité qui nécessite d'utiliser une classe.

Cette classe est l'unique porte d'entrée du Métier par un assemblage externe. C'est ce que l'on appelle une façade : on utilise donc ici le patron de conception « Facade ». Nous allons étudier plus précisément ce Manager et un de ses attributs important le DataManager.

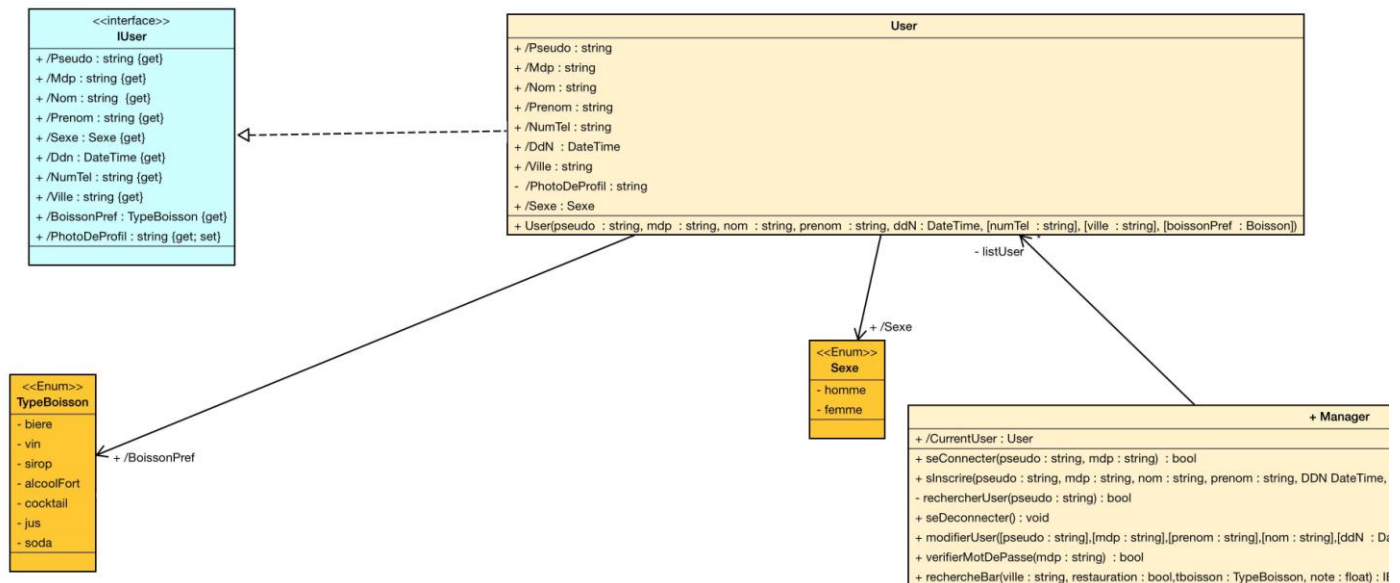
Classe Manager :



Le Manager est donc la classe principale, le point d'entrée du Métier. Comme nous pouvons le voir il possède une propriété privée de type `IDataManager`. Cette propriété et cette interface sont utiles pour la persistance (cf. diagramme de classe de persistance). Comme nous pouvons le voir le Manager contient une propriété `CurrentUser` qui va représenter l'utilisateur (User) actuellement connecté à l'application. Nous pouvons aussi voir qu'il contient des méthodes utiles à la connexion et à la déconnexion. Une méthode `s'inscrire` permet à un utilisateur de s'inscrire dans l'application, cette inscription est validée si elle correspond aux critères définis, tel que par exemple on ne peut s'inscrire s'il y a déjà un utilisateur avec le même pseudo ; cela explique la présence de la méthode `rechercherUser`. Il y a aussi une méthode `modifierUser` qui permet à un utilisateur de modifier son profil. La méthode `verifierMotDePasse` a été créée pour vérifier si un mot de passe saisi correspond au mot de passe de l'utilisateur courant. Un utilisateur peut rechercher des bars, pour cela il passera par la méthode `rechercheBars`. Il a aussi la possibilité de laisser un avis sur un bar, d'où la présence de la méthode `laisserAvis`.

On voit donc que la majorité des cas du diagramme de cas sont mis en place ici par le biais de ces méthodes. Ainsi le Manager est bel et bien la façade du Métier puisque pour mettre en œuvre un cas on passe par le Manager. On voit aussi la présence de méthodes `chargerDonnees`, `enregistrerDonnees` et `copierDonnees` utilisées pour la sauvegarde et le chargement des données.

Classes autour de User :



Nous allons maintenant nous intéresser à la partie autour de la classe User. Comme nous pouvons le voir le Manager est aussi constitué d'une liste de User, privée pour permettre de l'encapsuler correctement. Cette liste permet au Manager de connaître tous les utilisateurs de l'application. Lors d'une inscription il ajoute le nouvel User à cette liste. Il la consulte aussi pour la connexion.

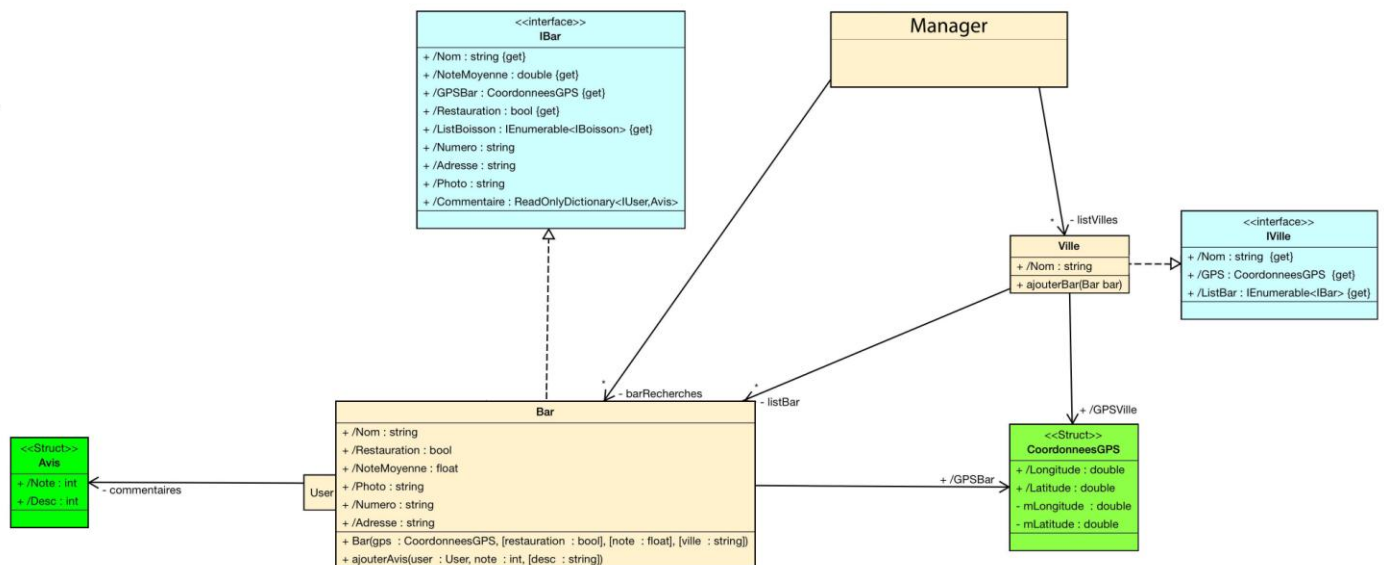
La classe User est constituée de plusieurs propriétés simples telles que le nom, le pseudo, le mot de passe, ...

Elle est aussi constituée d'une propriété pour le sexe de l'utilisateur qui est une énumération « Sexe » qui peut prendre comme valeur Homme ou Femme. De plus, il y a aussi une propriété portant sur le type de boisson qu'il préfère, qui est lui aussi une énumération avec plusieurs valeurs telles que la bière, les sodas, les jus, ...

Nous avons choisi de faire cette propriété pour pouvoir éventuellement trier les bars qu'un utilisateur va rechercher selon ceux qui servent le type de boisson qu'il préfère.

Nous pouvons aussi voir qu'une interface IUser est implémentée par User. Cela permet d'avoir une façade immuable de User et cette interface est publique à l'inverse de User qui est interne. Ainsi on dehors du Métier on manipule des IUser et non des User cela empêche de pouvoir modifier les propriétés de User et cette façade wrappe la classe mutable User en immuable.

Classes autour de Bar et Ville :



Nous avons ici simplifié le Manager pour s'intéresser plus particulièrement aux classes Ville et Bar.

Tout d'abord nous pouvons voir que le Manager possède la aussi une liste privée de Ville pour avoir accès à toutes les villes de l'application. Il possède aussi une liste privée de Bar pour avoir la liste des bars recherchés, qui est mise à jour lorsque l'on effectue une recherche depuis le Manager (méthode `rechercherBar`). Ces 2 listes sont encapsulées par des `IEnumerable` publics, ce qui permet d'éviter toutes les actions qui modifie la liste. Elles sont donc publiques en lecture seule uniquement. De plus les `IEnumerable` ont comme types les façades immuables de Bar et Ville, respectivement `IBar` et `IVille`. Ces façades immuables fonctionnent de la même manière que la façade immuable de User.

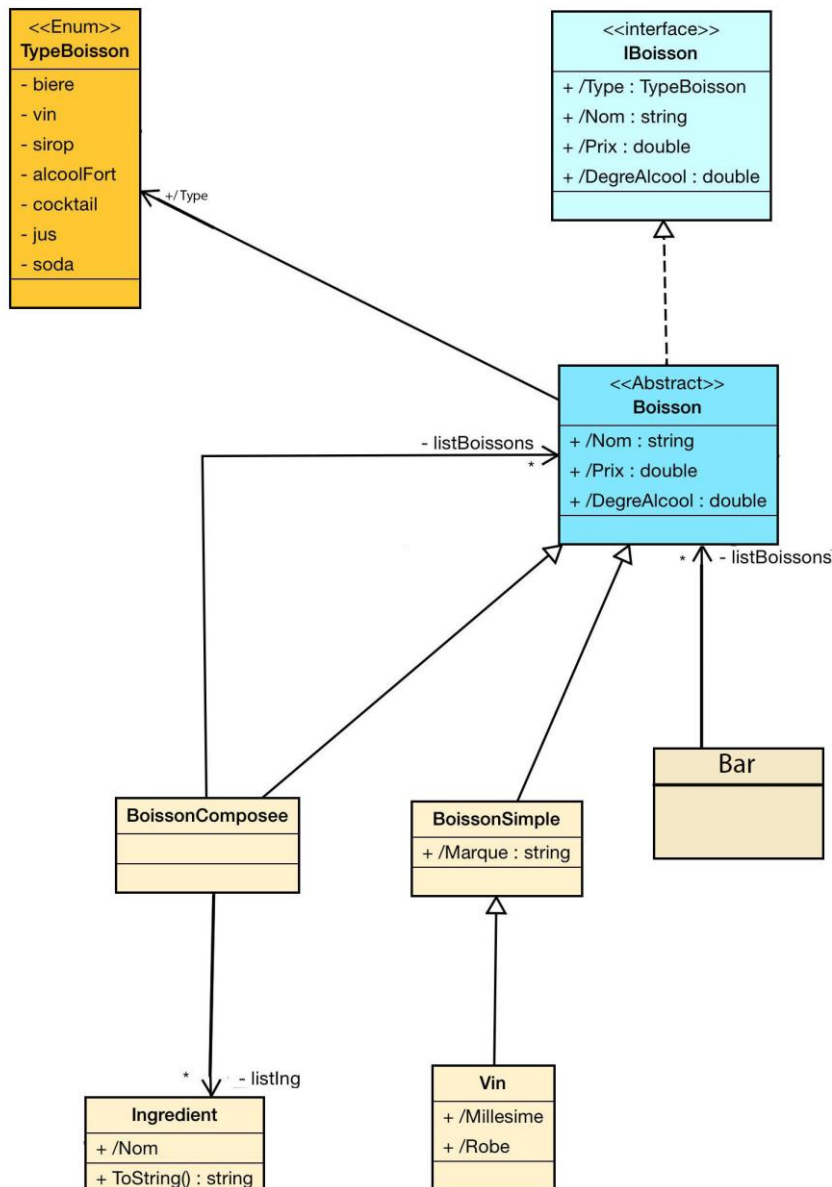
Une Ville est uniquement représentée par un nom et des CoordonneesGPS.

Ces dernières sont une structure qui regroupe 2 propriétés publiques, la latitude et la longitude. Cette structure sert donc à rassembler une latitude et une longitude sous forme d'un unique type. Ces coordonnées GPS serviront pour les Maps dans la vue (cf. diagramme de partie personnelle).

Une Ville possède aussi une liste privée de Bar, regroupant tous les bars présents dans cette ville. Cette liste est elle aussi encapsulée par un `IEnumerable<IBar>` comme on peut le voir dans la façade `IVille`.

Un Bar est quand à lui représenté par un nom et des CoordonneesGPS. Il possède aussi des propriétés telles que le numéro de téléphone, l'adresse et une photo. La propriété `restauration` indique si le bar sert aussi à manger ou non. Nous pouvons aussi voir qu'un Bar possède un dictionnaire ayant en clé un User et en valeur un Avis. Un Avis est une structure possédant une note (entier pouvant aller de 0 à 5 compris) et d'une description. Ce dictionnaire sert donc à conserver les différents avis laissés par différents User sur ce Bar. Nous avons décidé qu'un utilisateur ne pourrait laisser qu'un seul avis par Bar. Ce dictionnaire est lui aussi encapsulé pour empêcher qu'il puisse être modifier mais pour qu'il soit uniquement en lecture seule. Il est encapsulé par un `ReadOnlyDictionary<IUser,Avis>` pour qu'on ne puisse rien modifier que ce soit le dictionnaire ou la clé ou la valeur puisque IUser est immuable et qu'une structure est immuable aussi. Enfin, même si elle n'apparaît pas sur ce zoom du diagramme, un Bar possède une liste de Boisson privée qui elle aussi est encapsulée grâce à un `IEnumerable<IBoisson>` comme nous pouvons le voir dans la façade `IBar`.

Classes autour de Boisson :



Nous pouvons donc voir qu'un Bar (simplifié) possède bien une liste privée de Boisson. Boisson est classe abstraite qui va être dérivée en 2 types de boissons : les boissons composées, qui sont comme leur nom l'indique composées d'autres boissons et d'ingrédients, et les boissons simples.

Boisson est abstraite pour qu'on ne puisse pas créer uniquement une boisson sans qu'elle ne soit composée ou simple.

On a utilisé ici le patron de conception Composite puisque qu'une classe fille (BoissonComposee) possède une liste de sa classe mère (Boisson).

Une Boisson est représentée de façon générale et commune à toutes les boissons d'un nom, d'un prix, d'un degré d'alcool et d'un type (élément de l'énumération TypeBoisson présentée lors de User).

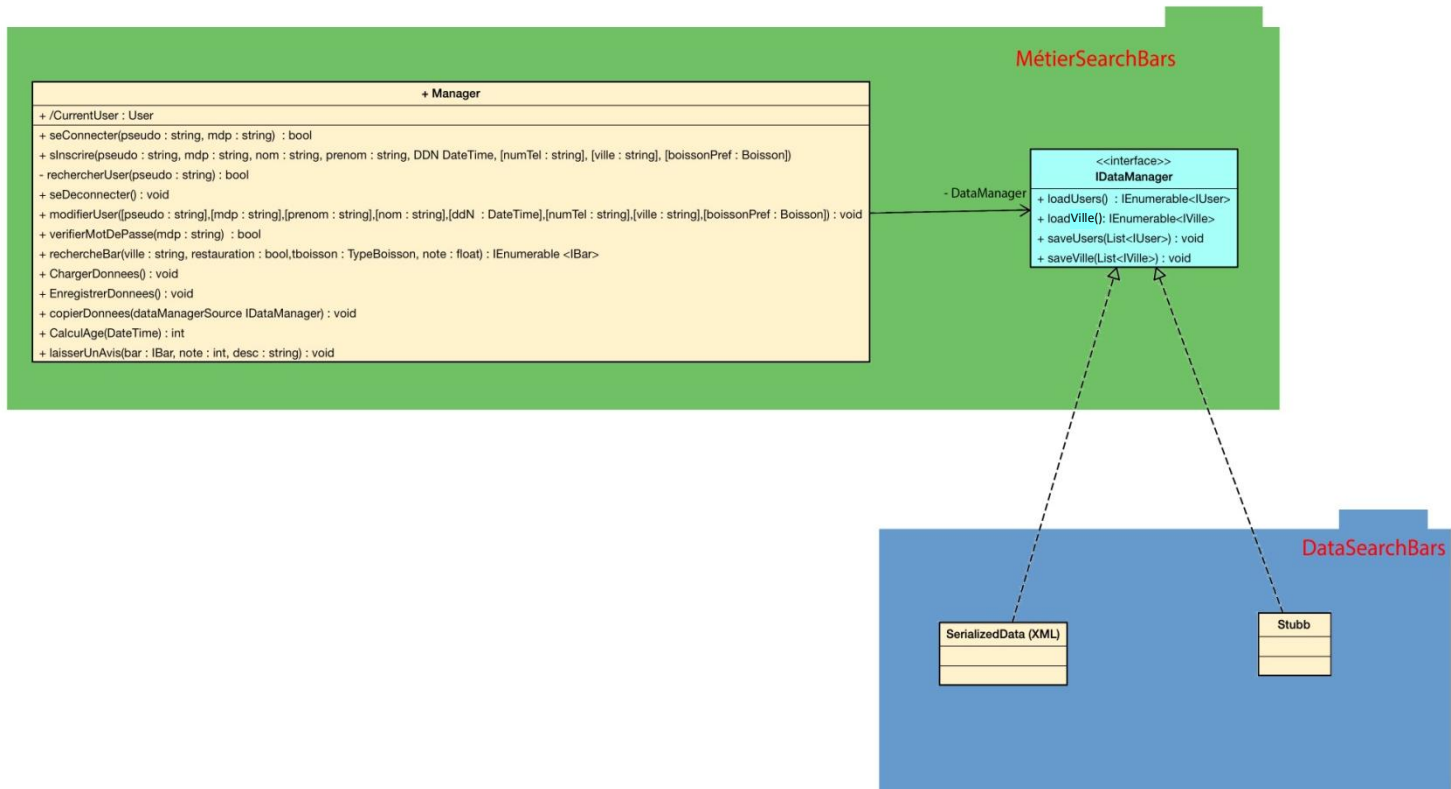
De plus les boissons simples ajoutent une marque permettant de différencier 2 vodkas par exemple.

Les boissons composées sont donc constituées d'une liste de boisson et d'une liste d'ingrédients, tels que de la menthe, du citron, ... Ici Ingredient est indiqué comme étant une classe mais en réalité c'est une structure ayant uniquement un nom permettant d'identifier un ingrédient.

La classe Vin dérive de BoissonSimple puisqu'un vin n'est pas composé d'autres boissons. Elle rajoute un millésime (année du vin) et une robe (rouge, rosé, ...).

Pour encapsuler les boissons et les rendre immuables on utilise ici aussi une façade IBoisson pour wrapper les Boissons.

- Partie persistance



On peut voir ici que le Manager possède une propriété privée, le DataManager de type IDataManager. Cette propriété permet de savoir quel type de persistance est choisi pour l'instance du Manager en cours. En effet lors de l'instanciation du Manager, on lui passe un paramètre de type IDataManager.

IDataManager est une interface appartenant à l'assemblage Métier. Cette interface impose un contrat permettant le chargement et la sauvegarde des données. Les 2 méthodes loadUsers et loadVille permettent de charger la liste de User et la liste de Ville du Manager à partir du type de persistance choisi. Elles retournent respectivement un IEnumerable<IUser> et un IEnumerable<IVille> car IEnumerable permet d'encapsuler les List et IUser et IVille sont les façades immuables de User et Ville. Les 2 méthodes saveUsers et saveVille permettent d'enregistrer les 2 listes du Manager à l'aide du type de persistance choisie.

Chaque classe de l'assemblage DataSearchBars qui souhaite effectuer la persistance du métier doit implémenter l'interface publique IDataManager.

Quand une classe implémente IDataManager, elle doit redéfinir les 4 méthodes de chargement et de sauvegardes décrites précédemment. Elle redéfinit selon le mode de persistance choisi. Ici, deux classes implémentent IDataManager : le Stubb et la persistance en XML.

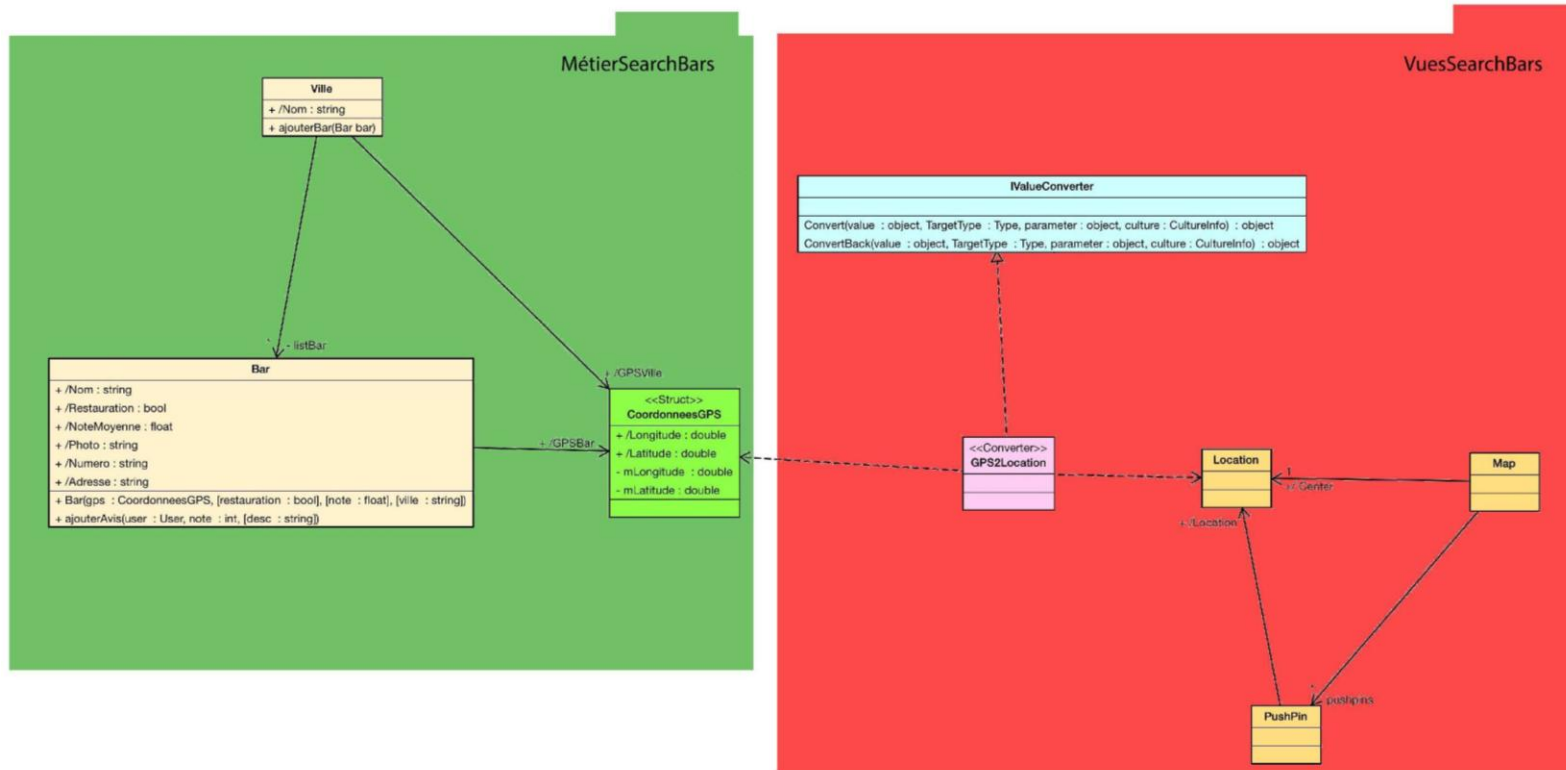
Le Stubb permet de créer des données en brut, cela permet d'effectuer des tests rapidement sans avoir codé une persistance effective. Malgré qu'il implémente les méthodes de sauvegarde, il ne permet pas de sauvegarder quoi que ce soit puisque qu'il est exécuté à chaque lancement de l'application.

La persistance en XML sérialise la liste de User et la liste de Ville en XML à l'aide de DataContract présent au sein de chaque classes qui va être sérialisées. L'idéal aurait été ici de séparer les DataContract qui sont dans les classes du Métier dans un autre assemblage (comme par exemple DataSearchBarsà) par le biais d'interfaces.

Lors de l'instanciation du Manager, on choisit donc quel type de persistance on utilise en lui passant dans le constructeur une instance d'une classe implémentant IDataManager. Cette façon de pouvoir interchanger les classes sans pour autant modifier le code du Manager s'appuie sur le patron de conception Strategy. Il permet donc ici de pouvoir créer d'autres types de persistance et de pouvoir les utiliser sans toucher à l'assemblage Métier.

De plus, lors de l'ouverture de l'application (construction du Manager), la méthode chargerDonnees va être appelée pour lire les données (par le biais de loadUser et loadVille). Le processus inverse est appelé dans la déconnexion pour enregistrer les données à chaque déconnexion.

- Partie personnelle (ajout de BingMaps)



Nous avons décidé d'ajouter à notre projet des cartes permettant de situer les bars et les villes sur une carte du monde plus ou moins zoomée selon le besoin. Pour cela nous avons pensé à créer une structure regroupant une latitude et une longitude en un seul type : `CoordonneesGPS`. Comme nous pouvons le voir dans la partie Metier (vert), les villes et les bars possèdent chacun une propriété de type `CoordonneesGPS` pour pouvoir les localiser.

Dans la partie des Vues, nous utilisons la bibliothèque permettant d'utiliser les BingMaps (Microsoft.Maps.MapControl.WPFdll).

Nous utilisons cette bibliothèque qui nous permet d'avoir un objet `Map` qui est créé à chaque fois qu'on souhaite afficher une carte, comme par exemple la vue de la ville avec tous les bars recherchés pointés sur la carte ; ou encore sur le détail d'un bar ou on affiche la carte centrée sur le bar avec un pointeur sur le bar sélectionné uniquement.

Pour afficher correctement ces cartes, nous avons besoin de les centrer soit sur la ville soit sur le bar sélectionné. Pour cela on utilise la propriété `Center` de `Map`, qui est de type `Location`. La classe `Map` et `Location` sont comprises dans la bibliothèque de BingMaps, nous n'avons donc pas eu besoin de les créer, nous nous contentons de les utiliser.

Pour transformer les `CoordonneesGPS` des villes et des bars en `Location`, nous avons dû utiliser un `Converter` qui convertie des `CoordonneesGPS` en `Location` utilisable par la propriété `Center` de la `Map`. Ce `Converter` implémente l'interface `IValueConverter` comme tous les `Converters` le doivent.