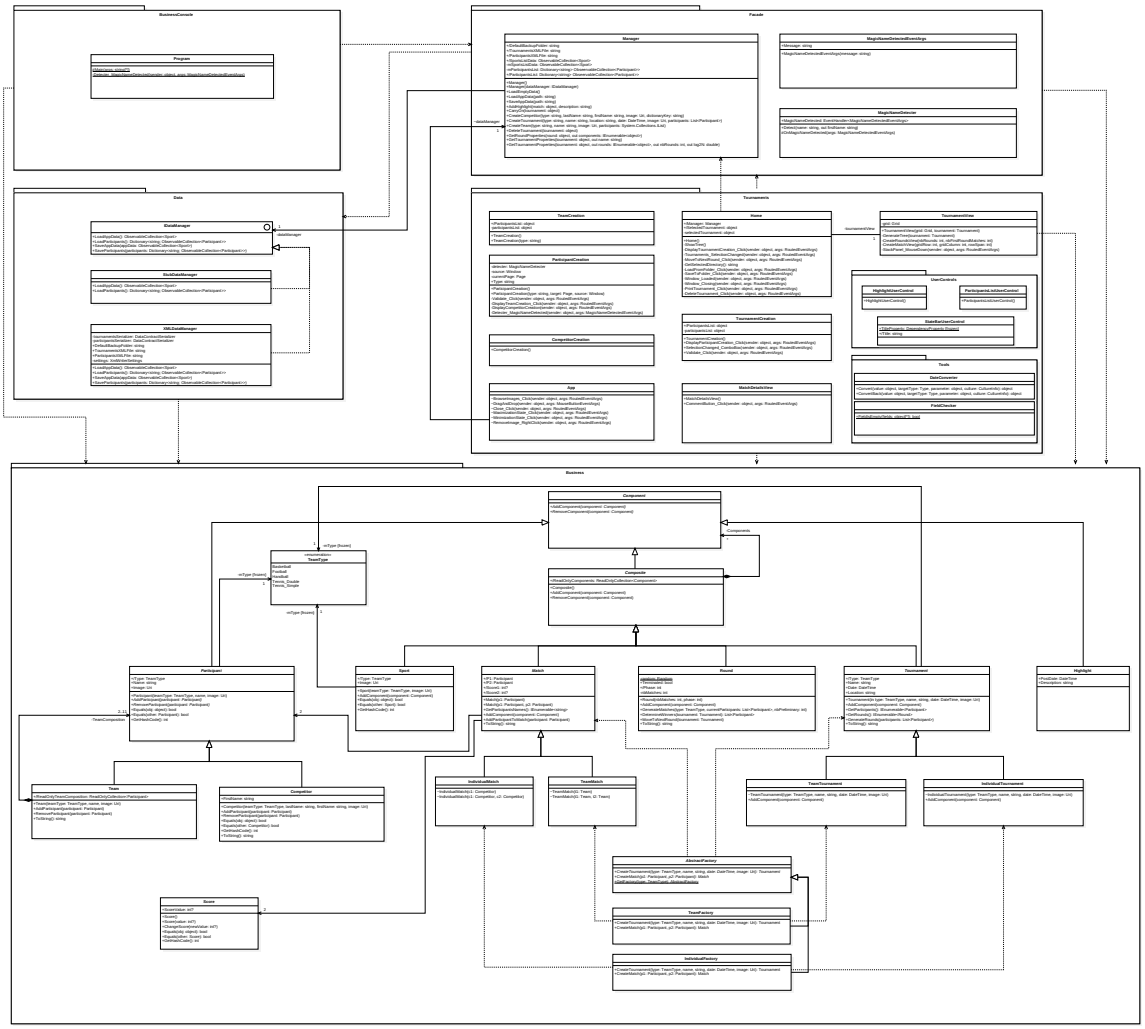
Tournaments

Diagramme de classes



Le diagramme original est aussi disponible en PDF dans le dossier intitulé *Diagrammes UML*.

Description du diagramme de classes

La solution se divise en différents projets qui ont chacun une responsabilité unique. Cette division des responsabilités permet une meilleure maintenabilité de l'application.

Tournaments est notre projet WPF. Il gère la vue et les évènements. Il est également décomposé en dossiers qui regroupent les éléments par finalité (la création d'un participant, la création d'un tournoi...) pour une meilleure visibilité. Le dossier UserControls contient les *contrôles utilisateur* qui permettent de n'avoir qu'un composant et éviter la duplication de code XAML.

Business est une bibliothèque de classes qui représente le métier de notre application. Nous avons implémenté plusieurs patrons de conception, notamment une *fabrique abstraite* qui permet de créer des environnements d'objets : un *IndividualTournament* (resp. *TeamTournament*) ne peut contenir que des *Round* contenant des *IndividualMatch* (resp. *TeamMatch*). Il y a également deux patrons *composite*. Dans l'ensemble, la façade mentionnée plus bas ne connait que les classes abstraites (sauf pour *Participant* et ses filles), ce qui l'oblige à passer par la fabrique abstraite pour créer les tournois et matches. Un sport contient une liste de tournois, qui se composent à la fois de *Round* et de *Participant*. Ces derniers sont nécessaires pour calculer le nombre de tours et de matches que l'arbre va contenir. Un Round se compose de matches et ces derniers contiennent deux participants, ainsi que deux scores. L'implémentation du composite nous a donc permis de n'avoir qu'une seule collection à gérer à travers la classe *Composite*, qui contient une collection de *Component*. L'implémentation du composite sur participant permet d'avoir une possibilité d'évolution de la composition d'une *Team*, qui se compose pour l'instant de *Competitor*. Il y a une couche d'abstraction qui oblige l'entité qui utilise le métier à passer par la fabrique abstraite pour les instancier, les constructeurs des classes filles étant internes. Cela permet de contrôler la fabrication des objets et ne pas créer des tournois individuels avec des matches par équipe. Cette encapsulation n'est pas possible avec *Participant* et ses filles car il n'a pas été possible de les intégrer à la fabrique abstraite, certains sports se composant à la fois de *Competitor* et de *Team*.

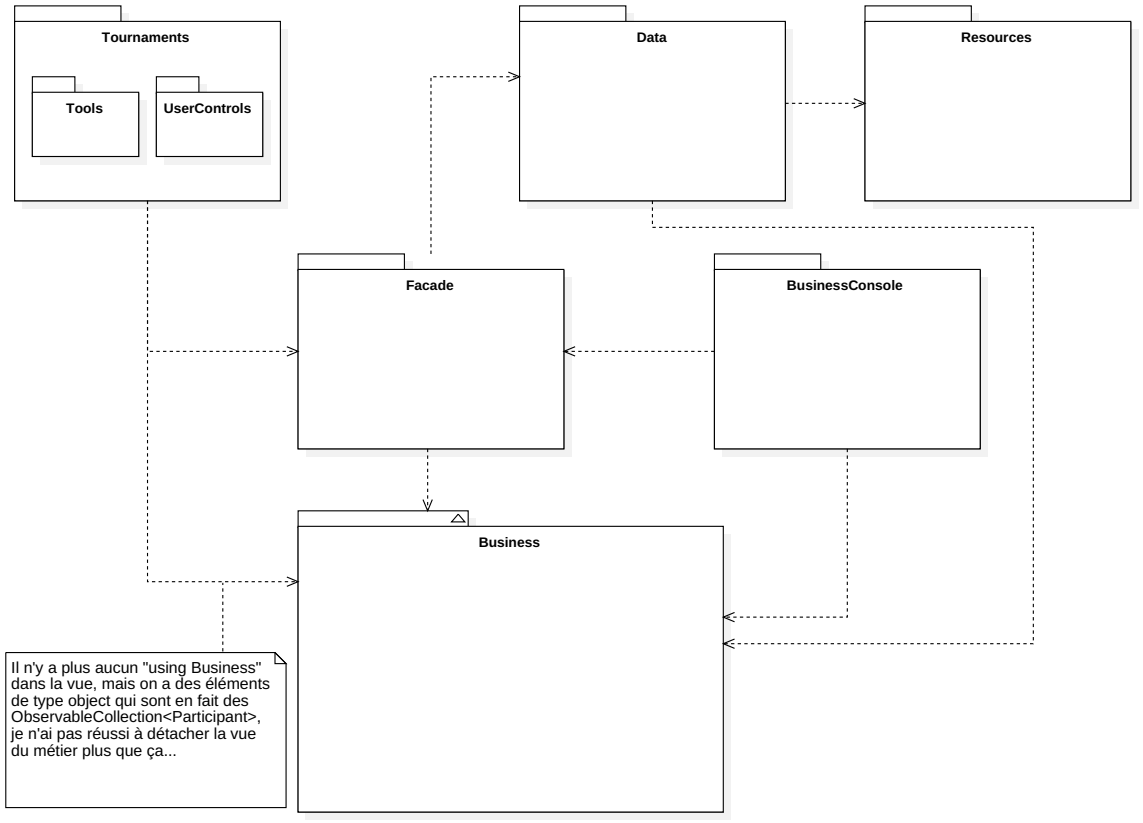
BusinessConsole est une application console qui permet de tester le métier rapidement avant de l'utiliser dans le projet WPF. C'est un gain de temps puisque cela nous évite tous les problèmes liés au binding.

Resources est une bibliothèque de classes qui regroupe les images qui sont ainsi dans une dll à part. Cela est pratique pour la maintenance : si une refonte graphique a lieu, par exemple, il suffit de déployer la dll uniquement.

Data est une bibliothèque de classes qui représente nos données. Elle regroupe principalement nos données de test pour le moment sous la forme d'un *stub*. Plus tard, nous nous en servirons pour charger et sauvegarder les données sérialisées.

Facade est la façade de notre projet. L'utilisateur passe par une façade pour créer ou charger les objets dont il a besoin sans avoir l'implémentation des classes et méthodes, ce qui limite le risque de dysfonctionnements liés à une mauvaise utilisation du métier. C'est le but de la façade, qui permet de donner à l'utilisateur un point de passage unique à un sous-système complexe. Le client passera donc par la façade pour effectuer les cas d'utilisation listés dans notre diagramme de cas d'utilisation.

Diagramme de paquetages



Description du diagramme de paquetages

Pour isoler le métier du reste de l'application, nous avons décidé de mettre en place une façade dans un package propre à lui. La vue ne dépend du métier que pour l'Enum et les listes de Participant, autrement elle passe par la façade pour réaliser les actions métier et récupérer les données qui sont gérées par *Data*. Data dépend du métier pour pouvoir créer les bonnes instances à fournir à l'application et des ressources qui sont les images par défaut de l'application. L'application console dépend, comme la vue, de la façade et du métier pour les mêmes raisons. Il est pour l'instant compliqué de complètement séparer le métier de la vue et de l'application console en raison de l'Enum, qui est indispensable pour qu'un participant pour un sport donné ne se retrouve pas à participer à des tournois d'autres sports.

Preuves des compétences

Conception et Programmation Orientées Objets

Documents

Je sais concevoir un diagramme de classes qui représente mon application [sur 9 points]
Se trouve dans le fichier *Documentation/Diagrammes UML/UML.mdj*, les classes mères sont maintenant au-dessus.

Je sais réaliser un diagramme de paquetages qui illustre bien l'isolation entre les parties de mon application [sur 2 points]
Il y a un début.

Je sais décrire mes deux diagrammes en mettant en valeur et en justifiant les éléments essentiels [sur 9 points]
Se trouve dans le fichier *Documentation/Rapports/C#_Tournaments.pdf*, la description du diagramme de classes est encore à compléter.

Programmation

Je maîtrise les bases de la programmation C# (classes, structures, instances...) [sur 2 points]
Beaucoup d'exemples dans toute la solution.

Je sais utiliser l'abstraction à bon escient (héritage, interfaces, polymorphisme) [sur 3 points]
Competitor et *Team* qui héritent de *Participant* qui est abstraite, utilisation d'une fabrique abstraite et autres exemples. Le polymorphisme est assez bien démontré dans *App.xaml.cs* où nous utilisons les types les plus hauts pour récupérer les sources des événements. **Note à mettre à jour.**

Je sais gérer des collections simples (tableaux, listes...) [sur 2 points]
Nous avons une liste dans *Team*.

Je sais gérer des collections avancées (dictionnaires) [sur 2 points]
Nous en avons une d'instanciée dans *Manager* mais qui est exploitée dans *TournamentCreation.xaml.cs* où il nous est nécessaire de filtrer la liste de participants à intégrer dans un tournoi sur sélection du sport dans la ComboBox.

Je sais contrôler l'encapsulation au sein de mon application [sur 5 points]
Nous avons essayé de garder un maximum d'éléments en *private* ou *internal* (notamment pour que la vue qui passe par la façade n'ait que les classes mères abstraites, les constructeurs de *TeamMatch* et *IndividualMatch* qui héritent de la classe abstraite *Match* sont en *internal*, seule *AbstractFactory* est visible à partir de la façade et c'est elle seule qui se charge d'appeler la bonne fabrique), tout cela pour essayer de garder l'implémentation protégée au maximum. **Note à mettre à jour.**

Je sais tester mon application [sur 4 points]

Nous avons testé tous les cas listés dans le diagramme de cas d'utilisation, à l'exception de l'impression de l'arbre qui ne peut se faire que pour la partie graphique.

Je sais utiliser LINQ [sur 1 point]

Nous avons utilisé LINQ aussi bien pour filtrer des listes dans *Tournament.cs* (région *Filters*, ligne 85) que pour caster des listes dans *TournamentCreation.xaml.cs* (ligne 79).

Je sais gérer les évènements [sur 1 point]

Nous en avons un, les classes sont définies dans l'assembly Facade pour qu'il puisse être à la fois testé par l'application console et utilisé par la vue. Il s'agit d'un événement qui ajoute une petite surprise à la saisie du nom de famille "BOUHOURS" pour un compétiteur.

Interface Homme-Machine

Documents

Je sais décrire le contexte de mon application pour qu'il soit compréhensible par tout le monde [sur 4 points]

Se trouve dans le fichier *Documentation/Rapports/IHM_Tournaments.pdf*.

Je sais dessiner des sketchs pour concevoir les fenêtres de mon application [sur 4 points]

Se trouve dans le fichier *Documentation/Rapports/IHM#_Tournaments.pdf*.

Je sais enchaîner mes sketchs au sein d'un story-board [sur 4 points]

Se trouve dans le fichier *Documentation/Rapports/IHM#_Tournaments.pdf*.

Je sais concevoir un diagramme de cas d'utilisation qui représente les fonctionnalités de mon application [sur 5 points]

Se trouve dans le fichier *Documentation/Diagrammes UML/UML.mdj*. Il est également expliqué dans */Documentation/Rapports/IHM_Tournaments.pdf*.

Je sais concevoir une application ergonomique [sur 2 points]

Nous avons essayé de faire au plus simple pour l'utilisateur en essayant de lui laisser le plus de choix possible en limitant les risques d'erreurs. Par exemple on lui laisse le choix du fichier de chargement ou de sauvegarde mais la création d'un participant lors de la création d'un tournoi est contrôlée (notamment au niveau du choix du sport). L'application reste intuitive et agréable à utiliser avec un style de fenêtre propre à nous, c'est-à-dire, des formes dans l'ensemble arrondies, que ce soit pour les fenêtres, ou pour les icônes de sports. Les informations essentielles sont visibles sur la page principale et l'utilisateur est guidé le plus possible dans la création des tournois et participants.

Je sais concevoir une application avec une prise en compte de l'accessibilité [sur 1 point]

Nous avons fait en sorte d'ajouter les actions valider à l'appui sur la touche *Entrée* à la création d'un tournoi, d'un participant ou encore l'ajout d'un commentaire, et la fermeture des fenêtres à l'appui sur la touche *Échap*.

Programmation

Je sais choisir mes layouts à bon escient [sur 1 points]

Validé à la première évaluation blanche, nous n'avons pas changé grand chose.

Je sais choisir mes composants à bon escient [sur 1 point]

Validé à la première évaluation blanche, nous n'avons pas changé grand chose.

Je sais créer mon propre composant [sur 2 points]

Il y en a quelques-uns dans le dossier *UserControls* du projet WPF, pour la barre d'état, les temps-forts d'un match, ou encore la liste des participants à un match pour une équipe.

Je sais personnaliser mon application en utilisant des ressources et des styles [sur 2 points]

Il y en a un paquet, que ce soit dans *App.xaml* ou dans les fenêtres directement. Ils se situent entre les balises *Resources*.

Je sais utiliser les DataTemplate (locaux et globaux) [sur 2 points]

Il y a des DataTemplate locaux dans *Home.xaml*, pour la ListView des tournois tout en haut (lignes 83 à 106), ou encore celle des sports tout à gauche (lignes 65 à 76). Nous avons également un DataTemplate global pour la ListView des participants qui se trouve dans *App.xaml* (lignes 25 à 36) car elle est présente dans dexu fenêtres (la création de tournoi ainsi que la création de participants).

Je sais intercepter les évènements de la vue [sur 2 points]

Il y a beaucoup d'exemples dans *App.xaml.cs*.

Je sais notifier la vue depuis des évènements métier [sur 2 points]

Match.cs implémente *INotifyPropertyChanged* car le détail de Match s'actualise même quand on passe les matches au tour suivant et qu'on reste sur la fenêtre de détail.

Je sais gérer le DataBinding sur mon master [sur 2 points]

Nous avons un master qui est la liste des sports à gauche, qui affiche une liste de tournois sur la barre du haut une fois le sport sélectionné.

Je sais gérer le DataBinding sur mon détail [sur 2 points]

La liste de tournois se met à jour à l'ajout d'un tournoi, et va à son tour afficher le tableau de tournoi correspondant au tournoi sélectionné.

Je sais gérer le DataBinding et les Dependency Property sur mes UserControl [sur 2 points]

Nous avons 2 exemples de binding avec les UserControls *ParticipantsListUserControl.xaml* où on bind l'image et le nom d'un participant et *HighlightUserControl.xaml* où il y a un binding sur la date du post et son contenu. Nous avons une DependencyProperty dans *StateBarUserControl.xaml*, qui ajoute un titre à côté de l'icône de l'application dans la barre d'état.

Je sais développer un Master/Detail [sur 2 points]

Oui, nous avons un master à gauche qui représente la liste des sports, qui va à la sélection afficher la bonne liste de tournois associée au sport sélectionné, et la sélection du tournoi affiche les détails du tournoi.

Projet Tuteuré

Documents

Je sais mettre en avant dans mon diagramme de classes la persistance de mon application [sur 4 points]

Se trouve dans le fichier */Documentation/Rapports/PTut_Tournaments.pdf*. Nous avons essayé d'expliquer au mieux pour expliquer le rôle de chaque classe mise en valeur...

Je sais mettre en avant dans mon diagramme de classes ma partie personnelle [sur 4 points]

Se trouve dans le fichier */Documentation/Rapports/PTut_Tournaments.pdf*. Nous avons essayé d'expliquer au mieux pour expliquer le rôle de chaque classe mise en valeur...

Je sais mettre en avant dans mon diagramme de paquetages la persistance de mon application [sur 2 points]

Se trouve dans le fichier */Documentation/Rapports/PTut_Tournaments.pdf*. Ici, ce sont les packages impliqués qui sont mis en valeur, avec une description qui explique quel package a quel rôle.

Je sais réaliser une vidéo de 1 à 3 minutes qui montre la démo de mon application [sur 10 points]

Il y a un raccourci vers une vidéo YouTube dans *Documentation/Video*.

Programmation

Je sais coder la persistance au sein de mon application [sur 4 points]

L'application utilise au premier lancement, les données du stub qui implémente l'interface *IDataManager*. À la fermeture de l'application, l'utilisateur peut choisir de sauvegarder ou non ces données. Cette sauvegarde (et au lancement suivant le chargement), sont gérés par le *XMLDataManager* qui implémente aussi *IDataManager*. La sauvegarde et le chargement des données sont gérés par le *Manager* qui fait appel aux méthodes du XMLDataManager. Les classes sérialisées sont décorées de DataContract, les propriétés à sérialiser sont bien décorées de DataMember, et les classes filles sont spécifiées avec KnownType.

Je sais coder une fonctionnalité qui m'est personnelle [sur 4 points]

L'arbre qui se génère automatiquement avec le binding en code behind avec l'algorithme pensé dans *Documentation/Rapports/PTut_Tournaments.pdf*. On le voit dans *TournamentView.cs* et *Tournament.cs*.

Je sais documenter mon code [sur 3 points]

J'ai commenté à peu près toutes les méthodes en laissant volontairement celles qui sont redéfinies ou qui sont des implémentations d'interface.

Je sais utiliser SVN. [sur 5 points]

La preuve se trouve sur la forge.

Je sais développer une application qui compile [sur 1 point]

Sauf conflit avec les .csproj, il n'y a aucun warning et aucune erreur. Il suffit de revert ses changements avant d'update.

Je sais développer une application fonctionnelle [sur 1 point]

L'application est fonctionnelle, l'ajout d'un tournoi se fait, sauf si un des champs n'est pas rempli, l'utilisateur en est notifié avec une MessageBox. Donc il faut tout renseigner, même une image (une par défaut qui se trouve dans l'assembly *Data*, dans le dossier Images).

Je sais mettre à disposition un outil pour déployer mon application [sur 2 points]

L'installeur est dans le dossier *Installeur*, où se trouvent notamment un dossier intitulé *Application Files* qui contient les dll déployées et l'installeur nommé *setup.exe*.

Je sais ajouter un Easter-Egg à mon application [points bonus]

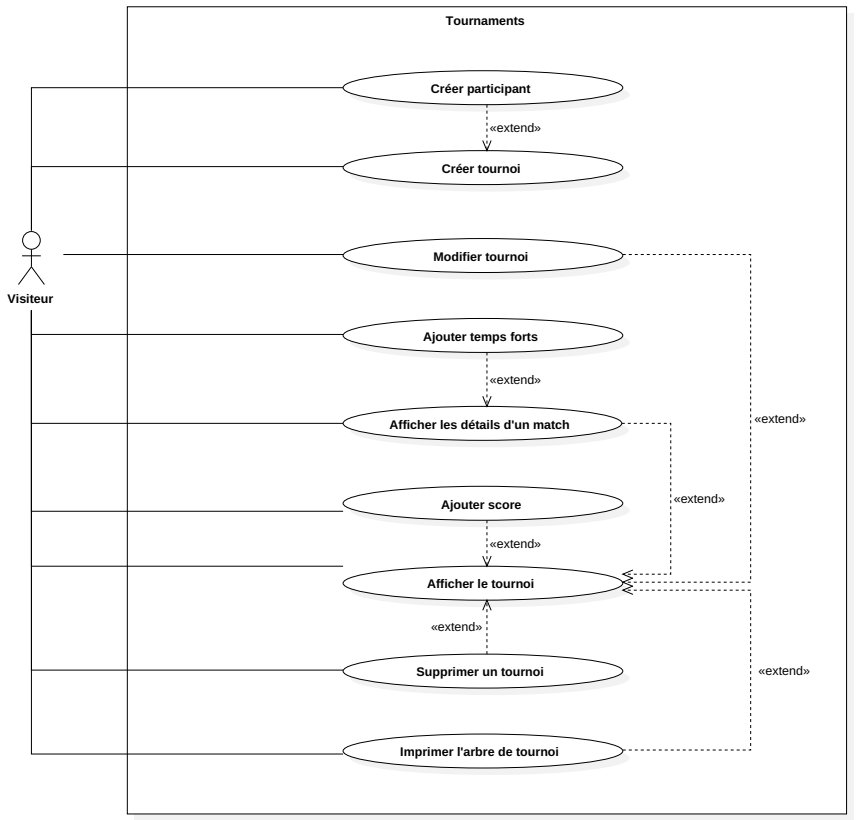
Il s'agit de notre événement personnel, qui à la saisie du nom "BOUHOURS" définit lui-même le prénom pour un compétiteur. Cela ne marche donc pas pour les équipes. Il faut soit avoir sélectionné *TennisSimple* dans la ComboBox de la fenêtre de création de tournoi, soit créer un nouveau membre pour une équipe pour que cela fonctionne.

Tournaments

Contexte

Cette application est un gestionnaire de tournoi à élimination directe multi-sports destinée aux organisateurs de tournois. Cette application permet donc à des écoles ou même des amateurs d'organiser des rencontres amicales numériquement, ce qui limite l'utilisation excessive de papier. Il est possible de créer des tournois, de saisir des compétiteurs qui seront placés aléatoirement dans le tableau qui servira à suivre l'évolution de la compétition (scores des différentes rencontres, évolution des compétiteurs dans la compétition...). Les différentes rencontres opposeront soit des équipes, soit des individus. Chaque rencontre possède un détail qui regroupe un espace temps forts relatifs à la rencontre, les informations sur les compétiteurs pour les tournois individuels, la composition des équipes pour les tournois en équipe.
Il est possible de créer un tournoi s'il n'existe pas déjà. Il faut alors spécifier le nom du tournoi, le lieu, la date, une image, le sport. La sélection de ce dernier mettra à jour la liste des participants possibles pour le tournoi.
Pour accéder à un tournoi, il est nécessaire de sélectionner un sport dans le menu de gauche, puis de sélectionner le tournoi correspondant dans la liste.

Diagramme de cas d'utilisation



Description des cas d'utilisation

- Créer un tournoi (nécessite d'avoir des participants)

  1. L'utilisateur clique sur le bouton "Ajouter" pour accéder à la fenêtre de création d'un tournoi.
  2. L'utilisateur renseigne le nom du tournoi.
  3. L'utilisateur renseigne le lieu du tournoi.
  4. L'utilisateur renseigne la date de début du tournoi.
  5. L'utilisateur renseigne la date de fin du tournoi.
  6. L'utilisateur renseigne le sport.
  7. L'utilisateur choisit une image pour le tournoi.
  8. L'utilisateur renseigne le nombre de participants qui vont s'affronter.
  9. L'utilisateur définit ses participants depuis les listes générées selon le nombre de participants.
  10. L'utilisateur
      (a) valide la création du tournoi.
      (b) annule la création du tournoi.
  11. Le système affiche
      (a) si la création a été effectuée avec succès, un message d'erreur sinon.
      (b) la page précédant la tentative de création du tournoi.

- Créer un participant

  1. L'utilisateur clique sur le bouton "Ajouter" pour accéder à la fenêtre de création d'un participant.
  2. L'utilisateur choisit le sport pratiqué par le participant.
  3. L'utilisateur choisit une image qui représente le participant.
  4. L'utilisateur indique qu'il s'agit
     (a) d'une équipe.
     (b) d'un compétiteur individuel.
  5. L'utilisateur renseigne le nom
     (a) de équipe.
     (b) du compétiteur.
  6. L'utilisateur renseigne
     (a) la composition de l'équipe parmi la liste proposée.
     (b) le prénom du compétiteur.
  7. L'utilisateur
     (a) valide la création du participant.
     (b) annule la création du participant.
  8. Le système affiche
     (a) si la création a été effectuée avec succès, un message d'erreur sinon.
     (b) la page précédant la tentative de création du participant.

- Afficher les détails d'un match (nécessite d'avoir affiché le tournoi)

  1. L'utilisateur clique sur une rencontre dans l'arbre du tournoi pour accéder au détail.
  2. Le système affiche les informations relatives à la rencontre sélectionnée (score, composition des équipes, temps forts).

- Modifier un tournoi (nécessite d'avoir affiché le tournoi)

  1. L'utilisateur remplace directement le nom ou le lieu du tournoi.
  2. Le système met à jour le tournoi automatiquement.

- Ajouter un temps fort d'un match (nécessite d'avoir affiché le détail d'une rencontre)

  1. L'utilisateur clique sur une rencontre dans l'arbre du tournoi pour accéder au détail.
  2. L'utilisateur se rend en bas de la page pour y voir les temps forts du match sélectionné.
  3. L'utilisateur clique sur le bouton ajouter, en-dessous des temps forts.
  4. L'utilisateur renseigne le commentaire lié au temps fort.
  5. L'utilisateur
     (a) valide l'ajout.
     (b) annule l'ajout.
  6. Le système revient à la page de détail du match en affichant
     (a) la liste des temps forts mis à jour.
     (b) la même chose qu'avant.

- Imprimer l'arbre de tournoi (nécessite d'avoir affiché un tournoi)

  1. L'utilisateur clique sur le bouton *Imprimer* sur la page principale.
  2. Le système affiche une boîte de dialogue présentant les réglages de l'imprimante.
  3. L'utilisateur
     (a) valide l'impression.
     (b) annule l'impression.
  4. Le système
     (a) lance l'impression sur l'imprimante choisie et ferme la boîte de dialogue.
     (b) ferme la boîte de dialogue.

- Supprimer un tournoi (nécessite d'avoir affiché un tournoi)

  1. L'utilisateur clique sur le bouton "Supprimer ce tournoi".
  2. Le système affiche une boite de dialogue pour confirmer la suppression.
  3. L'utilisateur
     (a) valide la suppression du tournoi.
     (b) annule la suppression du tournoi.
  4. Le système ferme la boite de dialogue et
     (a) affiche une page blanche, en attendant que l'utilisateur sélectionne un autre tournoi.
     (b) ne fait rien.

- Afficher un tournoi

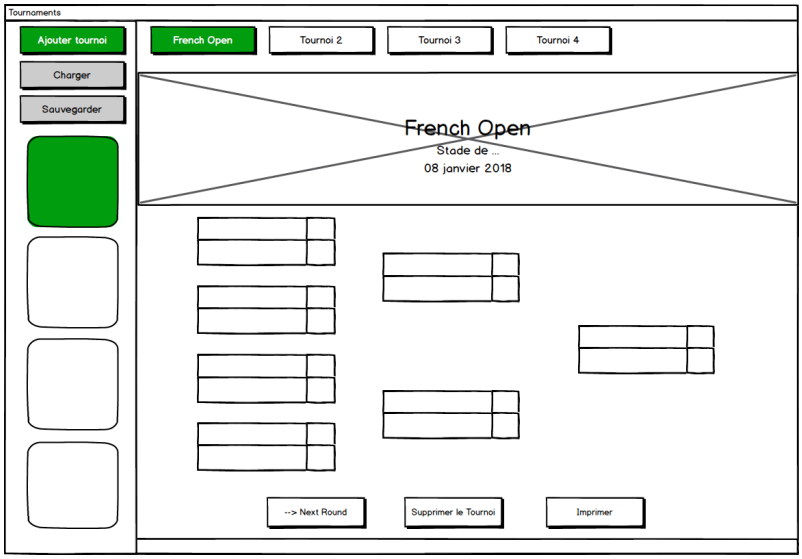
  1. L'utilisateur sélectionne un sport.
  2. L'utilisateur sélectionne une année.
  3. L'utilisateur sélectionne un tournoi parmi la liste proposée.
  4. Le système affiche les informations relatives au tournoi (nom, lieu, date...), l'arbre du tournoi, ainsi que deux tableaux correspondant aux matches à venir et au classement.

- Mettre à jour le score d'un match (nécessite d'avoir affiché un tournoi)

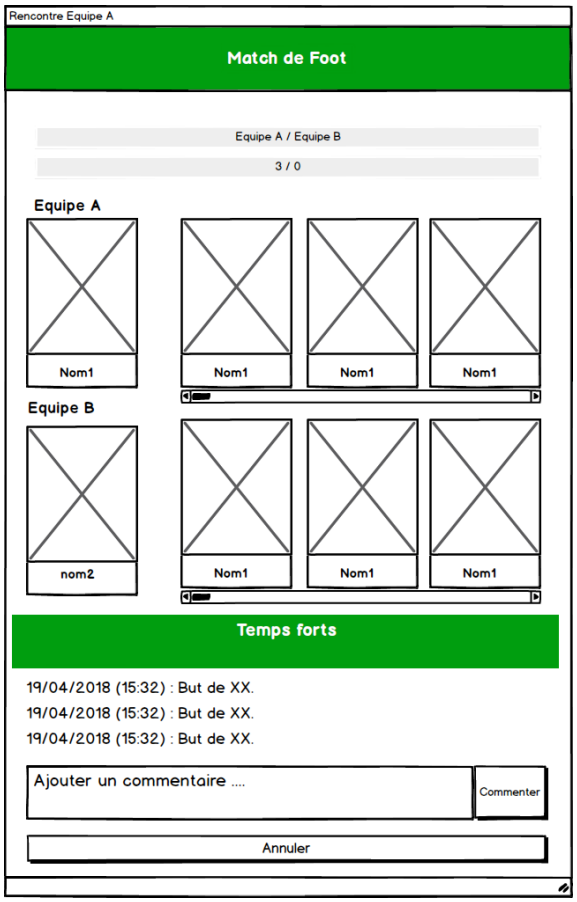
  1. L'utilisateur sélectionne une rencontre dans l'arbre.
  2. L'utilisateur saisit le score dans le champ prévu à cet effet.
  3. Le système met à jour le score automatiquement (binding bi-directionnel).

Sketches

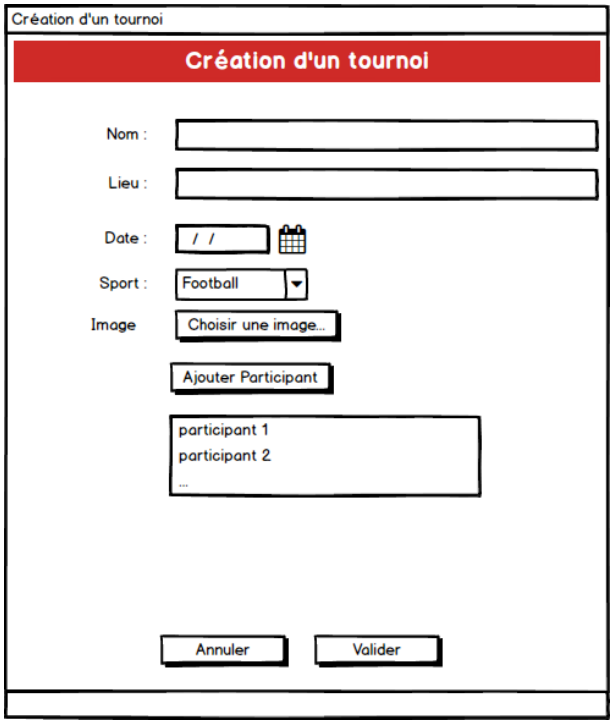
La fenêtre principale



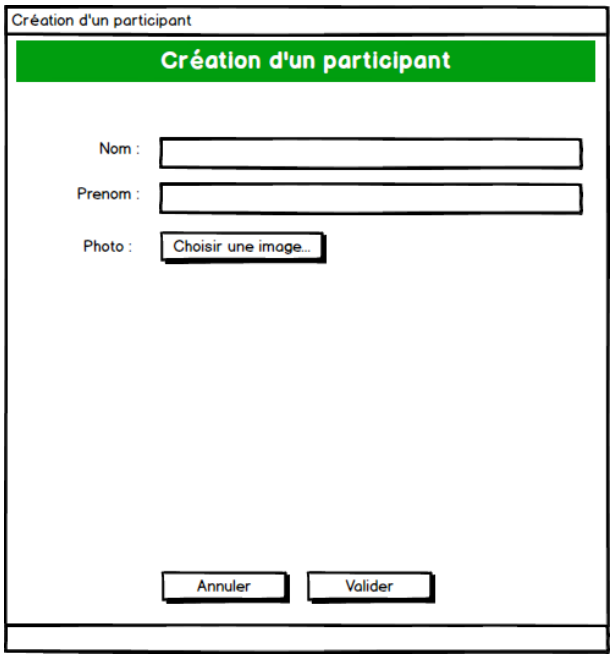
La fenêtre de détail d'un match



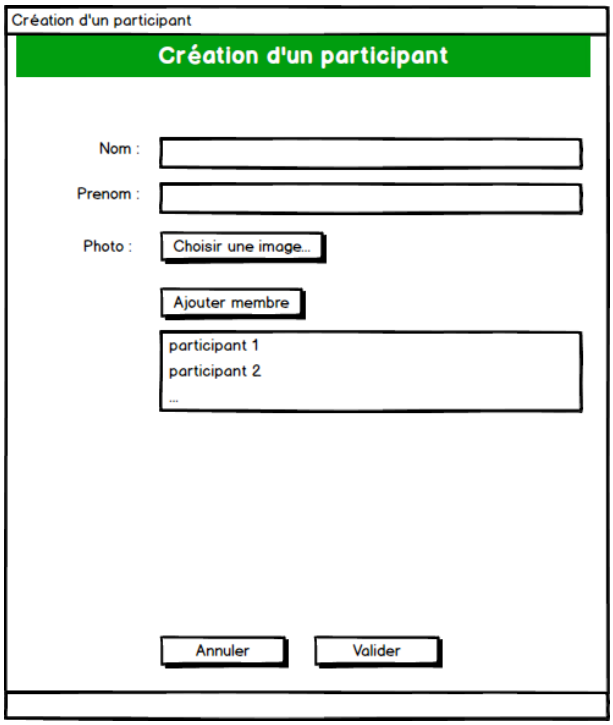
La fenêtre de création d'un tournoi



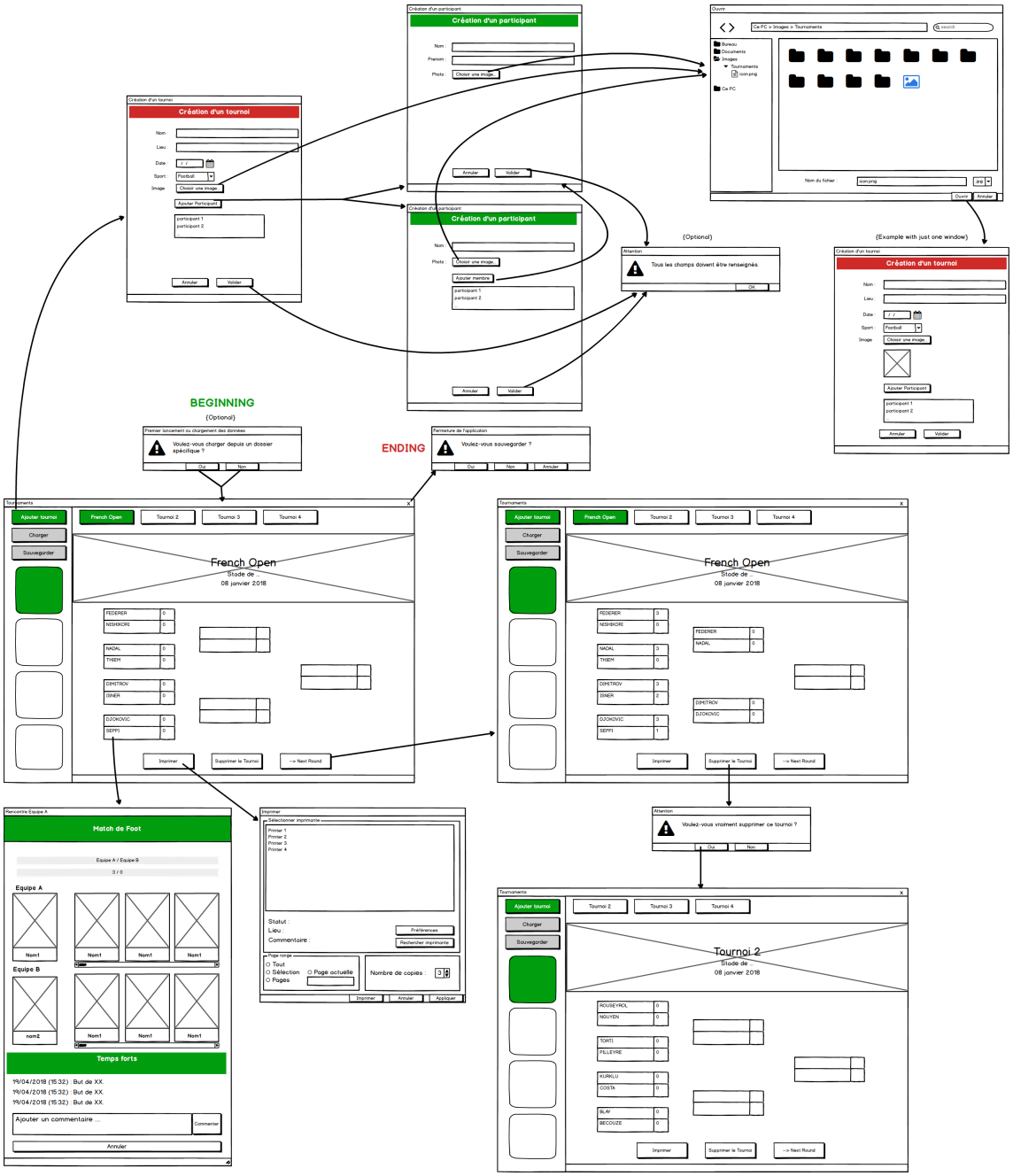
La fenêtre de création d'un participant (ici un compétiteur)



La fenêtre de création d'un participant (ici une équipe)



Storyboard


Création d'un participant
Création d'un tournoi
Ouvrir
(Optional)
(Example with just one window)
BEGINNING
(Optional)
ENDING
French Open
Tournoi 2
Tournoi 3
Tournoi 4
Ajouter tournoi
Charger
Sauvegarder
Imprimer
Supprimer le Tournoi
--> Next Round
Match de Foot
Temps forts
Tournoi 2

Tournaments

Ajouts personnels

Notre ajout personnel est l'arbre de tournoi en lui-même, qui est généré à partir du nombre de participants pour un tournoi donné. Le raisonnement est le suivant[1].

Soit N le nombre de participants pour un tournoi donné. On l'exprime en fonction de la puissance de 2 inférieure la plus proche et du reste que l'on note a.

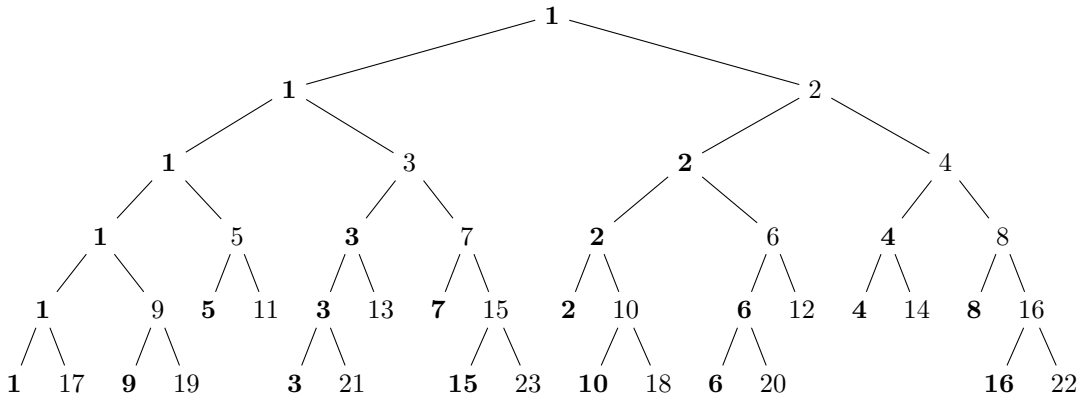
$$N = 2^n + a$$

On fait rentrer $2a$ joueurs au premier tour, avant de faire rentrer les $N - 2a$ autres joueurs au deuxième tour.
Le nombre de tours (rounds), noté R, équivaut à la partie entière par excès du logarithme en base 2 du nombre de participants N.

$$R = \lceil \log_2(N) \rceil$$

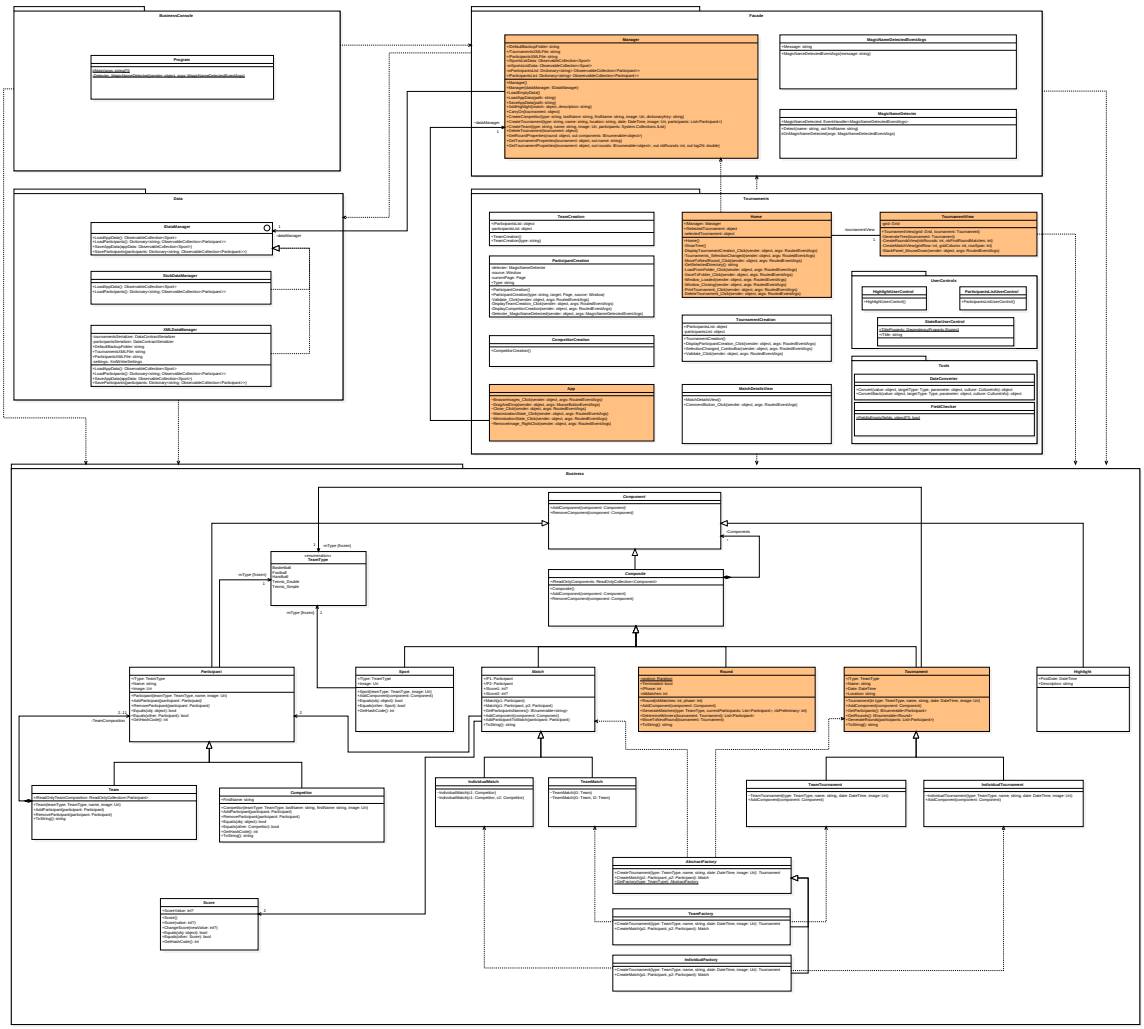
Exemple
$N = 23$, donc $N = 2^4 + 7$, soit $a = 7$.
On fait rentrer $2a = 14$ joueurs au premier tour, il reste donc $N - 2a = 23 - 14 = 9$ joueurs qui rentreront au deuxième tour.
À l'issue du premier tour, il reste $a = 7$ joueurs. Au deuxième tour, 9+7 = 16 joueurs s'affronteront.
Le nombre de tours équivaut à $R = \lceil \log_2(23) \rceil = 5$ (car $\log_2(23) \approx 4.52$).

Illustration de l'exemple
Le premier nœud ne compte pas, il faut 2 participants pour former un match et un tour est formé d'un (ou de) match(es).



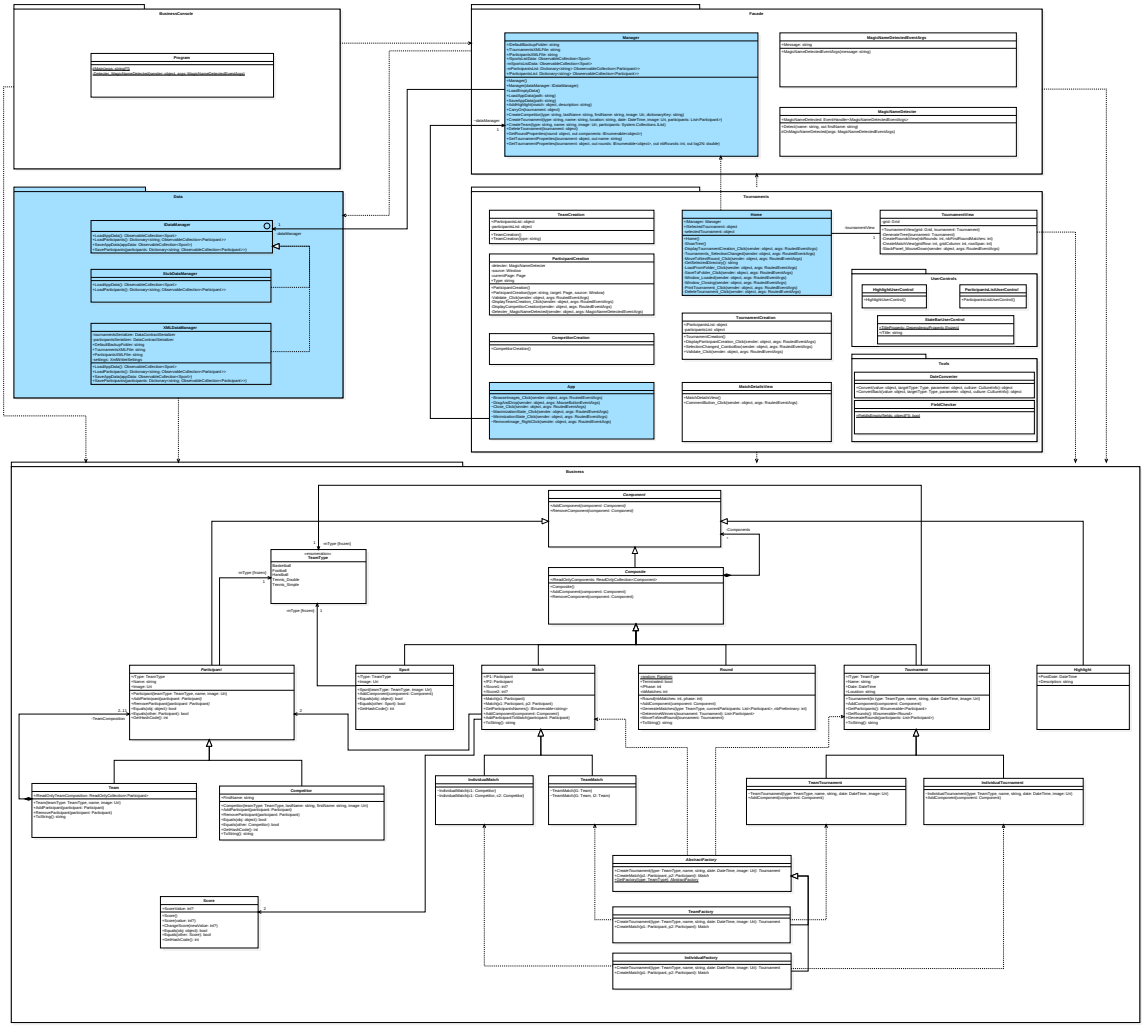
Avec une répartition quelconque, on voit qu'on arrive bien à 5 tours, avec 7 matches au tout premier tour.

[1] *Merci à M. Wohrer pour l'aide qu'il nous a apportée pour trouver l'algorithme le plus égalitaire.*



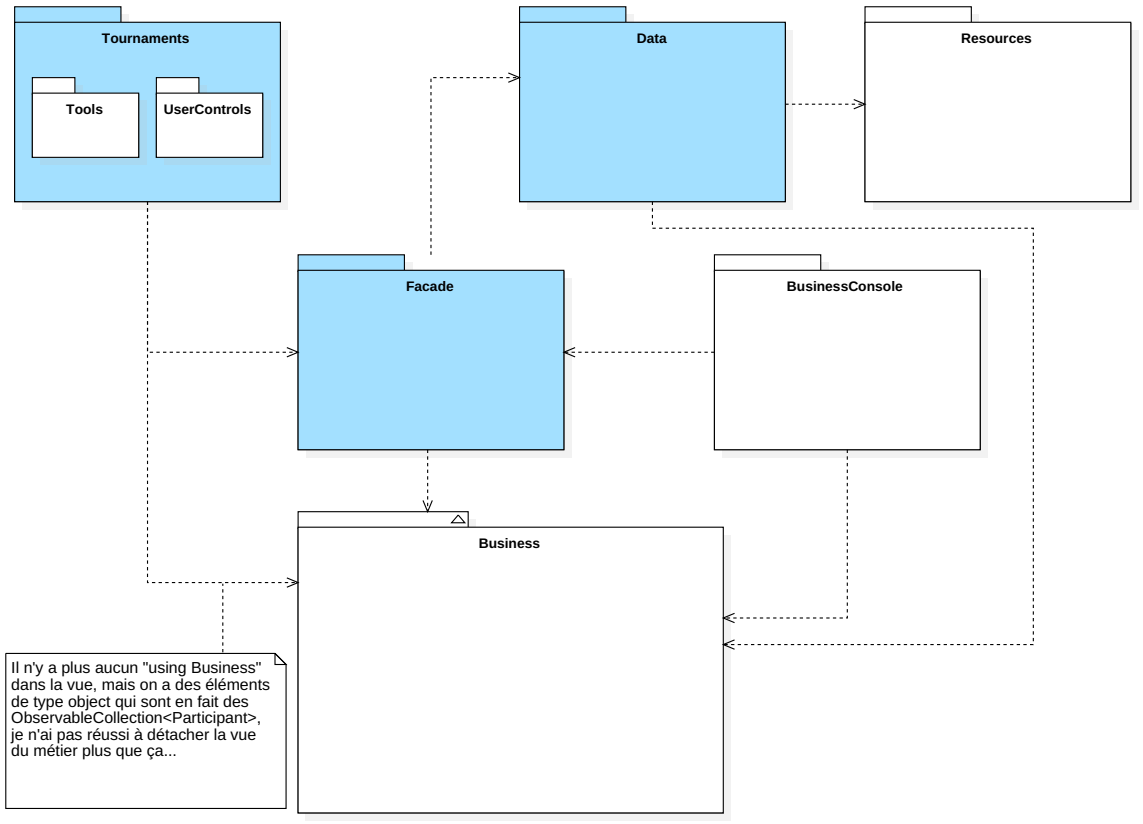
Home fait appel, au moment de la sélection d'un tournoi, au constructeur de *TournamentView* en effaçant le tableau déjà affiché s'il y a besoin. L'implémentation de cet algorithme se fait à la fois dans *TournamentView*, et dans *Tournament*. En réalité, le *Manager* (dont l'instance est stockée dans *App*) va se servir de *Tournament* et les passer à *TournamentView* pour qu'il puisse générer le bon nombre de lignes et colonnes de la grille passée en paramètre de son constructeur, l'arbre en lui-même et gérer l'alignement des matches par rapport au tour précédent (sauf pour les matches préliminaires). *Round* est utilisé, quant à lui, pour instancier le bon nombre de matches par tour. On a ainsi un arbre généré dynamiquement, ce qui nous a obligés à faire du binding en C#.

Persistence au sein de l'application



Nous avons en bleu les classes auxquelles le client a accès pour charger ou sauvegarder ses données. Toutes les classes métier à l'exception des fabriques sont sérialisées.

Le client a le choix de lancer le chargement ou la sauvegarde depuis un dossier s'il n'a pas choisi de le faire au lancement avec la boîte de dialogue ou à la fermeture de l'application. À ce moment il se situe dans *Home*. Cette classe va récupérer le dossier sélectionner et le passer au *Manager*, qui est stocké dans *App* qui va s'occuper de définir le répertoire courant avant de lancer le chargement ou la sauvegarde des tournois ainsi que des participants avec le *IDataManager* approprié, dans notre cas, le *XMLDataManager*.



On voit donc maintenant que la vue fait appel à la façade qui délègue ensuite le chargement ou la sauvegarde à *Data*, qui est responsable des données de l'application.