Génie logiciel - Projet 2018

Application de gestion de course pédestre

F.Couthouis - V.Garruchet

Encadré par :
M. Pesquet
Mme. Clermont
M. Bertrand

Bordeaux INP
ENSC

Sommaire

Sommaire **2**

Introduction **3**

Spécification générale **4**

Analyse fonctionnelle. 4

Analyse des besoins pour l'interface avec l'utilisateur. 4

Schémas UML associés à vos différentes analyses (diagramme de classes, diagramme de séquences). 5

Description de l'architecture de l'application. 7

Répartition des tâches dans le groupe et planning. 8

Spécifications détaillées **10**

Liste des formulaires composants l'application 10

Justification des choix de conception et de production du code source 10

Résultats et Tests **11**

Listes des exigences métier respectées par le programme 11

Copies d'écran des moments clés. 12

Description des protocoles de tests pour vérifier le bon fonctionnement du programme. 14

Résultats des tests. 17

Bilan et Perspectives **18**

Points positifs et négatifs. 18

Evolutions possibles. 18

I. Introduction

L'objectif de ce projet est la réalisation d'un gestionnaire de course pédestre, capable d'assister les organisateurs pendant les phases d'inscription des participants, de saisie puis d'analyse des résultats.

L'application doit également être conçue pour être aussi ergonomique et conviviale que possible. Enfin, une liste des exigences techniques nous était fournie :

Code	Description
ET_01	L'application est réalisée sous Windows à l'aide de la technologie WinForms
ET_02	Les données persistantes sont stockées dans une base de données relationnelle MySQL
ET_03	L'application est structurée soit selon une architecture en couches (App/DAL/Domain), soit selon une architecture MVP
ET_04	Le lien entre la BD et les objets de l'application est fait à l'aide de l'outil d'ORM NHibernate
ET_05	L'application respecte autant que possible les grands principes de conception étudiés en cours : séparation des responsabilités, limitation de la duplication de code, KISS, YAGNI, etc
ET_06	L'ensemble du code source respecte la convention camelCase
ET_07	Les noms des classes, propriétés, méthodes, paramètres et variables sont choisis avec soin pour refléter leur rôle

II. Spécification générale

A. Analyse fonctionnelle.

Les exigences métier se divisaient en 3 catégories d'importance :

Exigences Prioritaires	Exigences Importantes	Exigences Optionnelles
En tant qu'utilisateur, je peux importer un ou plusieurs participants à partir des informations fournies dans un fichier CSV En tant qu'utilisateur, je peux importer les résultats de la course fournis dans un fichier CSV En tant qu'utilisateur, je peux afficher les informations sur les participants sous forme tabulaire En tant qu'utilisateur, je peux consulter le classement global de la course sous forme tabulaire	En tant qu'utilisateur, je peux accéder au résultat d'un coureur par une recherche à partir de son nom ou de son numéro de dossard L'application gère plusieurs courses. En tant qu'utilisateur, je peux choisir la course gérée	En tant qu'utilisateur, je dois m'authentifier avant toute opération de modification des données de l'application (import, édition, suppression) En tant qu'utilisateur, je peux afficher le classement pour une tranche d'âge de 10 ans choisie

B. Analyse des besoins pour l'interface avec l'utilisateur.

Les utilisateurs finaux sont des organisateurs de course pédestre. L'objectif de notre programme est de les assister pendant les phases d'inscription des participants, de saisie puis d'analyse des résultats.

N'ayant pas accès directement aux utilisateurs, nous nous sommes basés sur les exigences fournies dans le sujet du projet, ainsi que sur notre bon sens pour tenter de fournir un programme répondant au mieux aux besoins des utilisateurs finaux.

Les utilisateurs de notre programme ne sont pas forcément familiers avec les programmes informatiques. L'interface se doit donc d'être assez simple, ergonomique et conviviale, tout en permettant une visualisation de toutes les informations dont l'utilisateur a besoin : informations sur les participants et résultats de chaque course. L'import de fichier CSV doit pouvoir se faire intuitivement, et l'utilisateur doit pouvoir être informé en cas d'erreur.

C. Schémas UML associés à vos différentes analyses (diagramme de classes, diagramme de séquences).

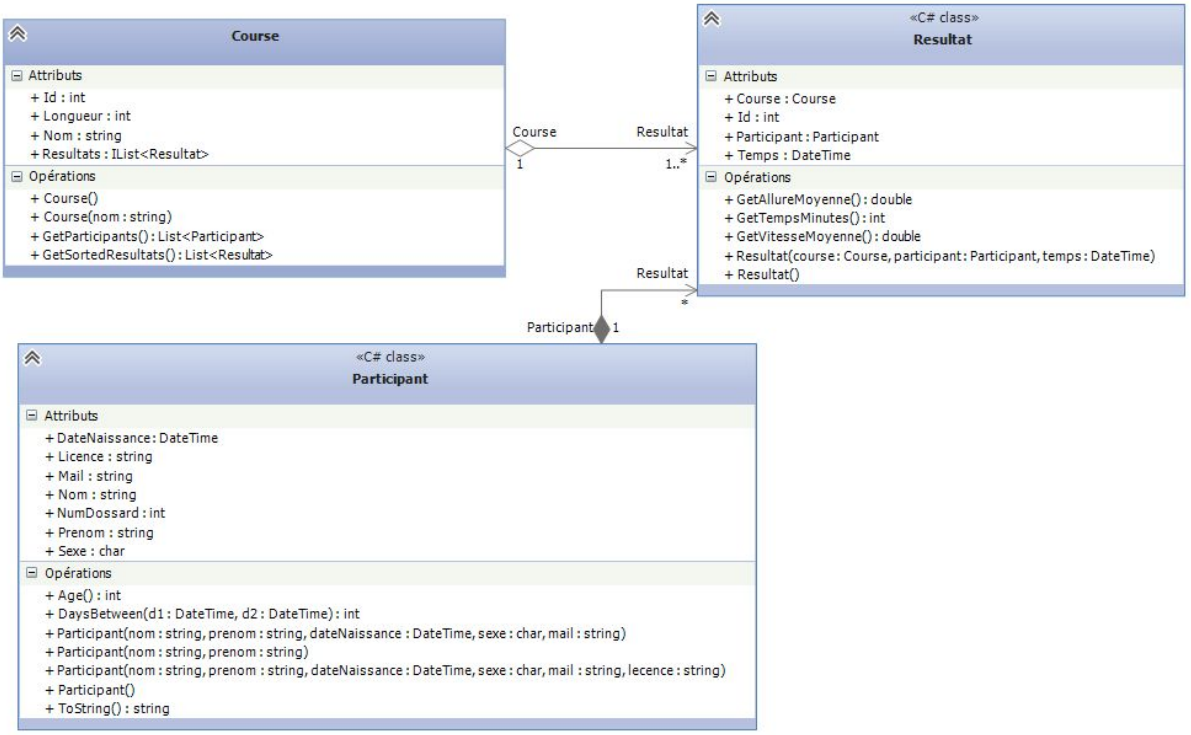


Diagramme des classes

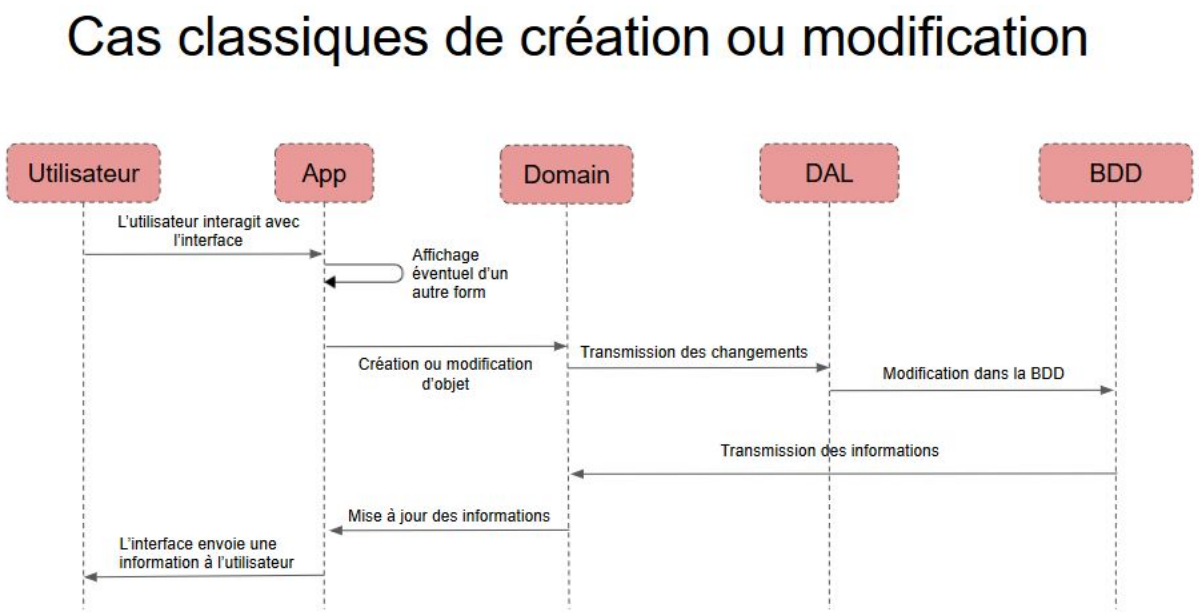
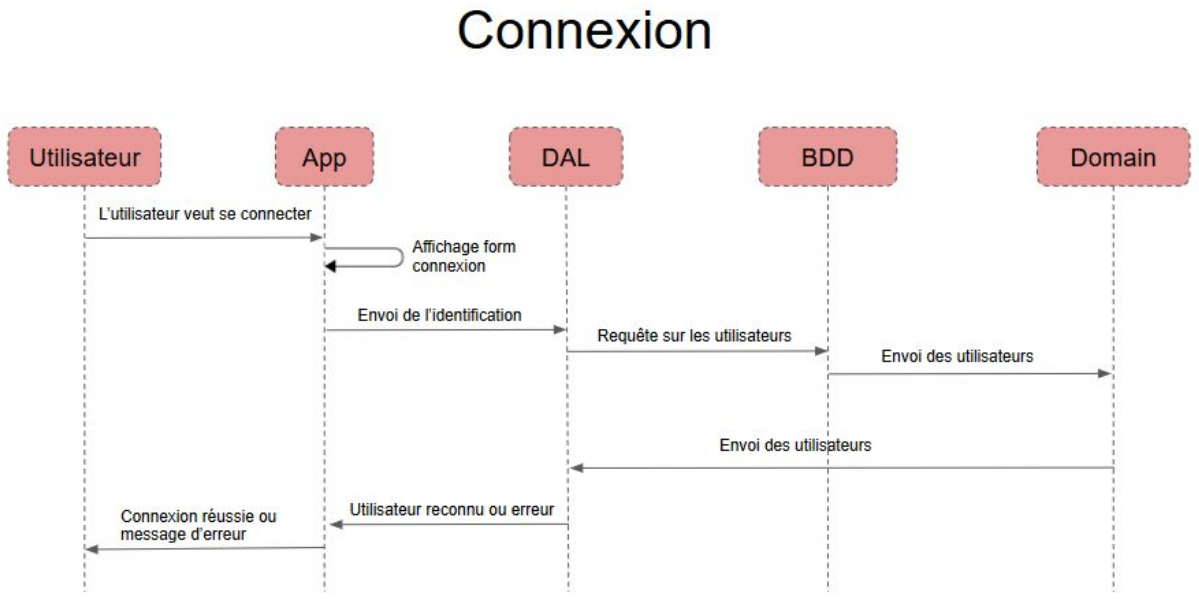


Diagramme de séquence classique de l'application

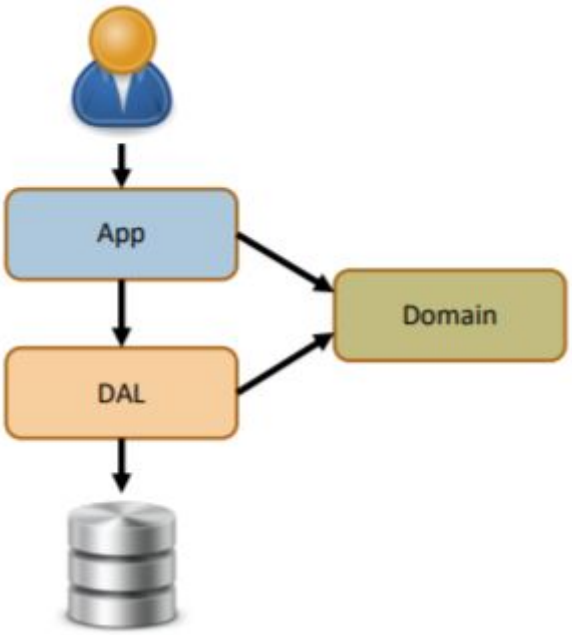


Exemple de diagramme de séquence différent : la connexion

D. Description de l'architecture de l'application.

Notre application a été conçue suivant une architecture en couche, permettant une séparation des responsabilités nécessaire pour un code compréhensible en maintenable, tout en conservant une complexité raisonnable afin de ne pas passer trop de temps à penser l'architecture. Pour un projet de plus grande ampleur, il aurait peut être eût été nécessaire de réfléchir à un autre type d'architecture (comme le modèle MVP par exemple), afin de pouvoir effectuer des tests unitaires sur l'interface.

Détail de notre architecture



Architecture en couche

APP : Interactions avec l'utilisateur (via WinForm)
Domain : Classes métiers (et mapping)
DAL : Gestion de l'accès aux données : repositories pour l'accès à la base de données et lecture de fichiers CSV

E. Répartition des tâches dans le groupe et planning.

Afin d'être efficaces sans se contraindre inutilement, nous avons opté pour une répartition des tâches assez libre. Un planning assez général a été mis en place en début de projet pour convenir de la durée de chaque sprint.

Nous nous sommes premièrement répartis les principales exigences métier de manière équitable afin de créer les bases de notre application : Victor a mis en place les bases de l'interface utilisateur tandis que Fabien s'est chargé des classes métiers et du mapping avec nHibernate.

Nous avons ainsi pu commencer à travailler en méthode agile, en réalisant des sprints d'une durée de une à deux semaines, suivant la charge de travail nécessaire. Il fallait avoir avancer au maximum notre tâche à l'issue des sprints. À la fin de chaque sprint, le code était refactorisé si nécessaire, pour conserver un programme facilement modifiable.

Nous faisions ensuite un point pour que nos développements respectifs restent cohérents, même si nous communiquions constamment afin de prévenir ce problème. Grâce à ces bilans et à cette forte communication, nous pouvions nous entraider si nécessaire et éviter ainsi de rester bloqués tout en partageant nos connaissances et méthodes respectives. Cela permettait également de prendre en compte les potentielles évolutions d'exigences que pouvait apporter le client.

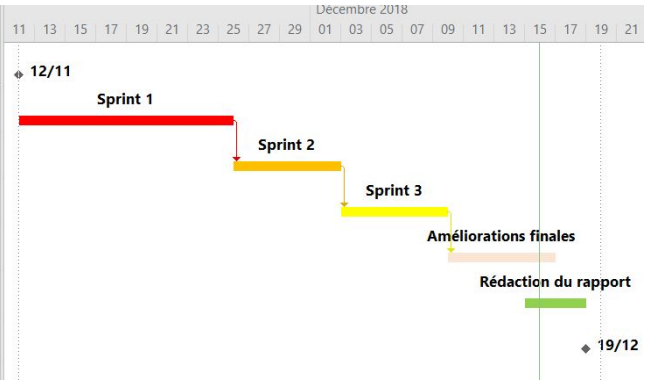
Nous avons également essayé de ne pas travailler sur les mêmes éléments de l'interface simultanément afin d'éviter tout conflit. Nous faisions également évoluer le Trello en y ajoutant des retours sur le travail réalisé, des pense bêtes, en mettant notre travail actuel en test ou en la validant pour rester organisés et d'avancer efficacement.

Ce fonctionnement, bien que nécessitant une confiance mutuelle importante, nous a paru être le meilleur choix pour plusieurs raisons. Premièrement, il était moins contraignant qu'un emploi du temps strict. Ainsi, si l'un de nous souhaitait avancer le projet sur son temps libre, il pouvait le faire quand il le souhaitait. Cela nous a permi d'avancer à notre rythme et de rester motivés tout au long du projet.

Deuxièmement, le Trello évolutif nous servait aussi de support de communication car nous pouvions y inscrire un problème à corriger ou une une nouvelle fonctionnalité à implanter. De plus, lister de nombreuses petites tâches qu'on pouvait achever assez rapidement nous donnait réellement l'impression d'être efficace et d'avancer, ce qui est capital pour conserver l'envie de travailler.

En résumé, notre répartition des tâches fût principalement basée sur le volontariat, la confiance, la communication et l'entraide, ce qui nous a offert un réel confort de travail. Nous n'oublions cependant pas que ce fonctionnement n'a été possible que parce que nous avions tout deux un rythme de travail similaire. En effet, une telle organisation peut être difficile à mettre en place dans un groupe plus important et plus hétérogène.

Nom de la tâche	Durée	Début	Fin	Prédécesseurs
Début du projet	0 jr	Lun 12/11/18	Lun 12/11/18	
Sprint 1	14 jrs	Lun 12/11/18	Dim 25/11/18	
Sprint 2	7 jrs	Lun 26/11/18	Dim 02/12/18	2
Sprint 3	7 jrs	Lun 03/12/18	Dim 09/12/18	3
Améliorations finales	7 jrs	Lun 10/12/18	Dim 16/12/18	4
Rédaction du rapport	4 jrs	Sam 15/12/18	Mar 18/12/18	
Fin du projet	0 jr	Mer 19/12/18	Mer 19/12/18	



Planning général respecté durant le projet

III. Spécifications détaillées

A. Liste des formulaires composants l'application

- **Formulaire principal**

Permet la sélection de course, l'affichage des participants et des résultats. Permet également l'édition des informations présentées après authentification.

- **Formulaire de modification de course**

Permet la modification de la course sélectionnée. Possibilité de modifier son nom ainsi que sa longueur.

- **Formulaire d'authentification**

Permet l'authentification de l'utilisateur pour l'édition de données.
L'identifiant et le mot de passe par défaut sont : "admin" / "admin".

- **Formulaire d'ajout d'utilisateur**

Permet, pour une course donnée, l'ajout d'un utilisateur dans la base de données, avec un résultat vide tant que la course n'a pas eu lieu.

B. Justification des choix de conception et de production du code source

- Nous avons choisi d'associer le numéro de dossard d'un coureur à son identifiant, puisque nous avons supposé que le numéro de dossard était propre à un coureur, et était le même pour toutes les courses. Nous aurions également pu stocker l'identifiant dans une variable à part.

- Nous avons décidé de lier une liste des résultats à une course. Ce choix nécessite d'insérer un résultat vide dans la base de données lors de l'ajout du participant. Cela permet une association simple des résultats liées à une course et permet d'accéder facilement aux résultats d'une course (puisque la liste des résultats d'une course est stockée en temps que propriété de ladite course).

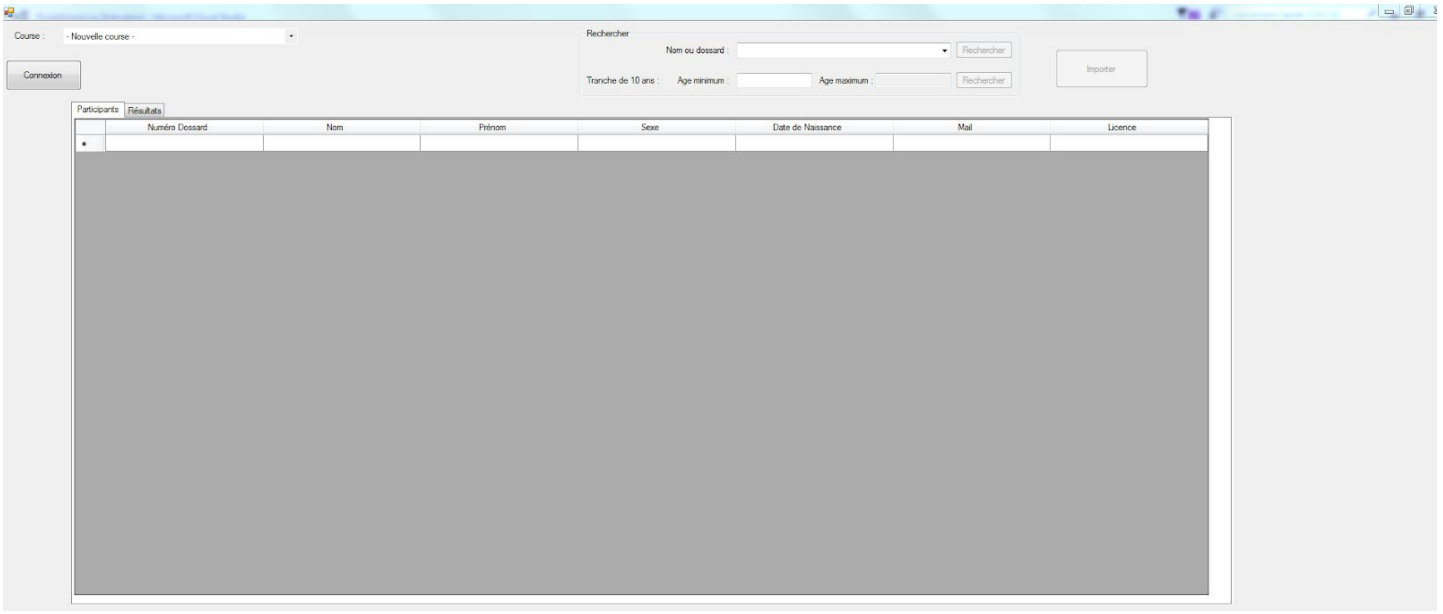
IV. Résultats et Tests

A. Listes des exigences métier respectées par le programme

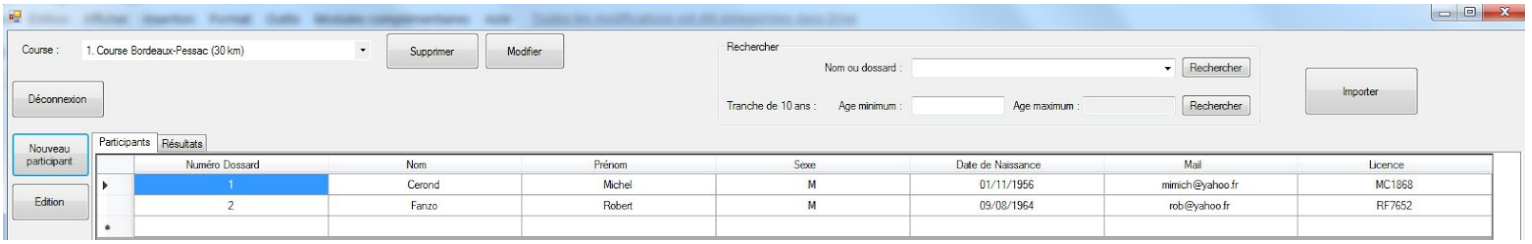
Nous avons fait en sorte de couvrir toutes les exigences métier demandées par le client.

Exigences Prioritaires	Exigences Importantes	Exigences Optionnelles
✓ En tant qu'utilisateur, je peux importer un ou plusieurs participants à partir des informations fournies dans un fichier CSV ✓ En tant qu'utilisateur, je peux importer les résultats de la course fournis dans un fichier CSV ✓ En tant qu'utilisateur, je peux afficher les informations sur les participants sous forme tabulaire ✓ En tant qu'utilisateur, je peux consulter le classement global de la course sous forme tabulaire	✓ En tant qu'utilisateur, je peux accéder au résultat d'un coureur par une recherche à partir de son nom ou de son numéro de dossard ✓ L'application gère plusieurs courses. En tant qu'utilisateur, je peux choisir la course gérée	✓ En tant qu'utilisateur, je dois m'authentifier avant toute opération de modification des données de l'application (import, édition, suppression) ✓ En tant qu'utilisateur, je peux afficher le classement pour une tranche d'âge de 10 ans choisie

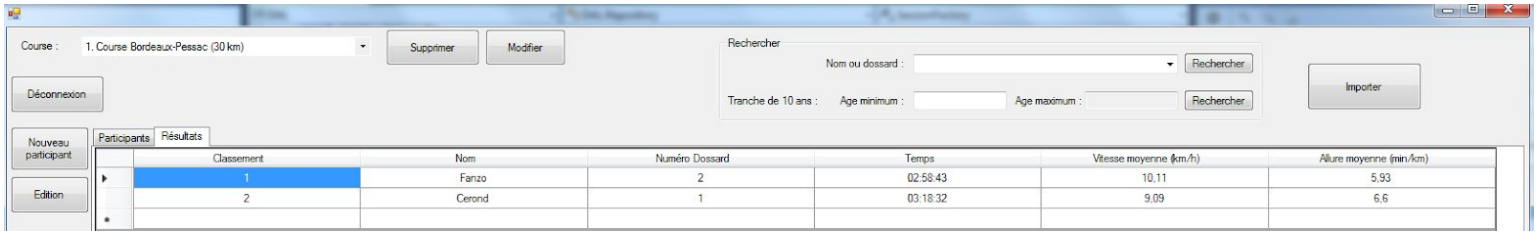
B. Copies d'écran des moments clés.



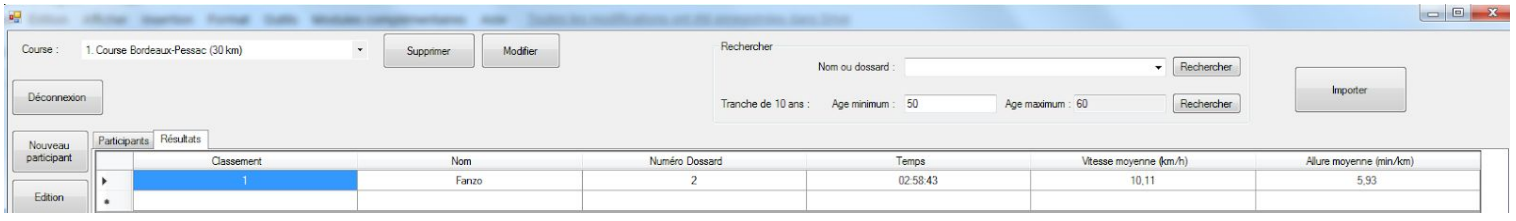
Écran d'accueil



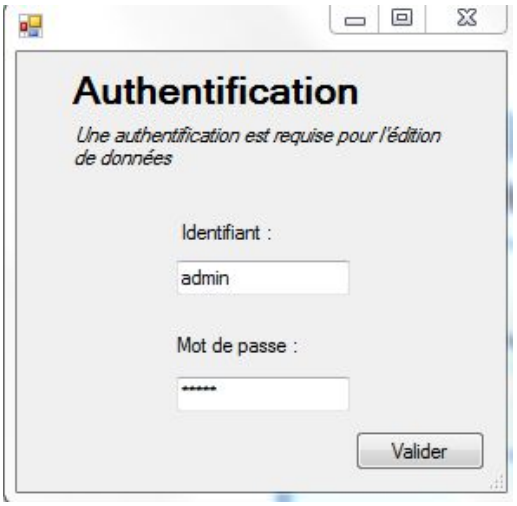
Informations sur les participants



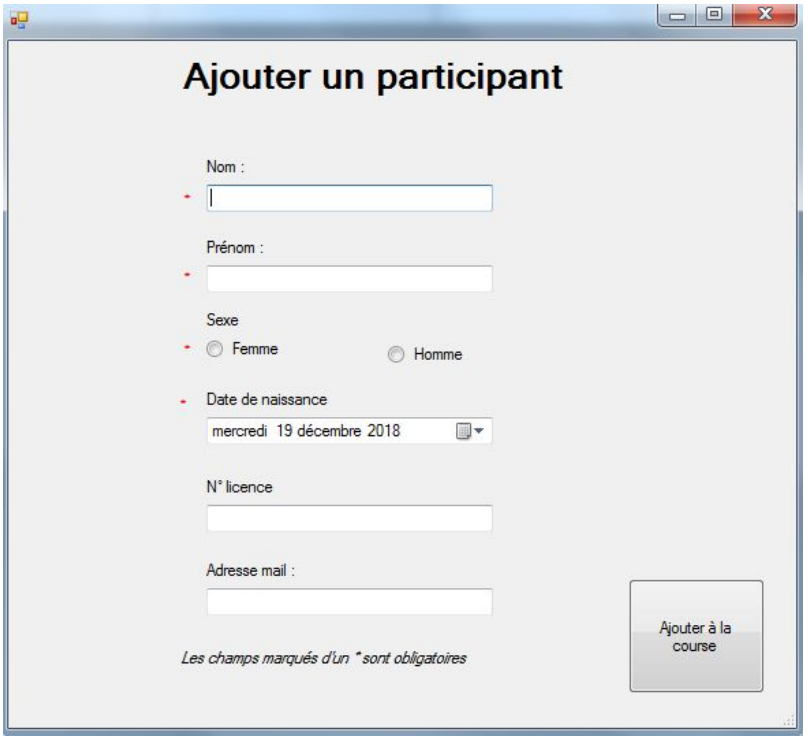
Résultats



Tri des résultats par tranche d'âge



Formulaire d'authentification



Formulaire d'ajout de participant

C. *Description des protocoles de tests pour vérifier le bon fonctionnement du programme.*

❑ Tests unitaires pendant la réalisation du code

Certaines fonctionnalités jugées "à risque" - *comme l'import de fichier CSV, ou encore le tri des résultats par ordre d'arrivé* - ont été développées suivant une approche TDD afin de gagner du temps sur le débogage. Le test prévu sur l'import du fichier CSV nécessitant l'utilisation de la méthode permettant l'obtention d'une course suivant son identifiant dans la BDD, nous avons également réalisé un test pour cette méthode.

❑ Protocole de test pour l'obtention d'une course suivant son identifiant dans la BDD (méthode CourseRepository.FindById(int id))

- Création d'une course de test et ajout dans la BDD
- Comparaison entre l'Id obtenu via la méthode FindById (myResult) et le véritable id de la course (expectedResult)

```
[TestMethod()]
0 références | 0 modification | 0 auteur, 0 modification
public void FindByIdTest()
{
    //Création d'une course de test et ajout dans la BDD
    CourseRepository courseRepository = new CourseRepository();
    Course c = new Course("courseTest");
    courseRepository.Save(c);

    //Comparaison entre l'Id obtenu via la méthode FindById (myResult)
    //et le véritable id de la course (expectedResult)
    int expectedResult = c.Id;
    int myResult = courseRepository.FindById(c.Id).Id;

    Debug.WriteLine("expectedResult : " + expectedResult, "; myResult : " + myResult);

    Assert.AreEqual(myResult, expectedResult);
}
```

❑ Protocole de test pour l'import de CSV (méthode CSV.Import())

- Création d'une course et on l'ajoute à la BDD
- Import du CSV "CSV\\inscrits.csv"
- On tente de retrouver la course importée dans la BDD
- Vérification : on regarde si le nom du premier participant de la course dans la BDD correspond à celui du CSV importé

```
[TestMethod()]
0 références | Fabien Couthouis, Il y a 23 heures | 1 auteur, 2 modifications
public void ImportTest()
{
    //Création des repositories
    ParticipantRepository participantRepository = new ParticipantRepository();
    ResultatRepository resultatRepository = new ResultatRepository();
    CourseRepository courseRepository  = new CourseRepository();

    //On crée une course et on l'ajoute à la BDD
    Course course = new Course("Test");
    courseRepository.Save(course);

    //On importe le CSV "CSV\\inscrits.csv"
    CSV csv = new CSV("..\\..\\..\\CSV\\inscrits.csv", course);
    csv.Import(participantRepository, resultatRepository, courseRepository);

    //On tente de retrouver la course importée dans la BDD
    Course courseBDD = courseRepository.FindById(course.Id);

    //Et on regarde si le nom du premier participant de la course dans la BDD correspond à celui du CSV importé
    Participant myResult = course.Resultats[0].Participant;
    Participant expectedResult = new Participant("Vignon", "Alexandre", new DateTime(1978, 11, 19), 'M', "alexv@mail.fr");

    Assert.AreEqual(expectedResult.Nom, myResult.Nom);
}
```

- ❏ Protocole de test pour le tri des résultats (méthode Course.GetSortedResultats()

- Création d'une course comprenant 3 résultats non triés
- Création d'une liste comprenant ces 3 résultats triés par ordre d'arrivée
- Comparaison entre la liste triée et la sortie de la méthode testés

```
[TestMethod()]
0 références | 0 modification | 0 auteur, 0 modification
public void GetSortedResultatsTest()
{
    //Création d'une course avec 3 résultats non triés
    Course course = new Course();
    Resultat r1 = new Resultat(course, new Participant(), new DateTime(2018,6,12, 2,36,2));
    Resultat r2 = new Resultat(course, new Participant(), new DateTime(2018,6,12, 1, 23, 26));
    Resultat r3 = new Resultat(course, new Participant(), new DateTime(2018,6,12, 2, 02, 42));

    IList<Resultat> lres = new List<Resultat> { r1, r2, r3 };
    course.Resultats = lres;

    //Liste attendue (triée)
    List<Resultat> expectedList = new List<Resultat> { r2, r3, r1 };
    //Résulat de GetSortedResultats()
    List<Resultat> myList = course.GetSortedResultats();

    Debug.WriteLine("myList 1 : " + myList[1].Temps );
    Debug.WriteLine("expectedList 1 : " + expectedList[1].Temps);

    CollectionAssert.AreEqual(expectedList, myList);

}
```

❑ **Test d'interaction des méthodes en testant des fonctionnalités**

Dans la même optique, que précédemment, nous avons décidé d'automatiser les tests pour certaines fonctionnalités posant problème lors du développement - *comme la suppression d'une course suivant son identifiant dans la BDD*- .

❑ Protocole de test de suppression d'une course

- On crée une course, un participant et un résultat
- Ajout du résultat à la course puis sauvegarde de la course dans la BDD
- On vérifie que la course est- bien présente dans la bdd après son ajout
- On supprime la course
- On vérifie qu'elle est bien supprimée

```
0 références | Fabien Couthouis, il y a 23 heures | 1 auteur, 2 modifications
public void DeleteTest()
{
    //On créé une course, un participant et un résultat
    CourseRepository courseRepository = new CourseRepository();
    ResultatRepository resultatRepository = new ResultatRepository();
    ParticipantRepository participantRepository = new ParticipantRepository();

    Course c = new Course("courseTest");
    courseRepository.Save(c);
    Participant p = new Participant("test", "test");
    participantRepository.Save(p);

    Resultat r = new Resultat(c,p,new DateTime());
    resultatRepository.Save(r);

    //Ajout du résultat à la course
    c.Resultats.Add(r);
    courseRepository.Save(c);

    int id = c.Id;

    //La course est-elle bien présente dans la bdd après son ajout ?
    if (courseRepository.FindById(c.Id).Id != id )
        Assert.Fail();

    //Suppression de la course
    courseRepository.Delete(c);

    Course result = courseRepository.FindById(id);
    Debug.WriteLine("Resultat GetById : " + result);

    //La course a-t-elle été supprimée ? Est-est elle bien absente dans la bdd ?
    Assert.AreEqual(result, null);
}
```

❑ **Test d'interface avec des scenarii d'utilisation**

❑ Protocole de test d'import de participants

Vous êtes un administrateur et vous souhaitez afficher les résultats d'une nouvelle course qui est actuellement stockée dans un fichier CSV.

Premièrement, vous devez vous connecter avec l'identifiant "admin" et le mot de passe "admin". Une fois connecté, récupérez le fichier CSV qui se trouve dans le dossier "H:\2A\Génie Log\Projet Génie Log\projet-2018-projetgl_couthouis_garruchet\CSV" des participants de la course puis celui de leurs performances.

- ❏ Protocole de test d'ajout de participant à une course

Vous êtes un administrateur et vous souhaitez ajouter un participant à la course 1.
Premièrement, vous devez vous connecter avec l'identifiant "admin" et le mot de passe "admin". Une fois connecté, ajoutez Joe Carry, un jeune homme né le 23 janvier 1987 à la course 1.

- ❏ Protocole de test d'édition de participant à une course

Vous êtes un administrateur et vous venez de voir qu'une coquille s'est glissée dans les résultats de la course 1. Vous souhaitez donc la corriger.
Premièrement, vous devez vous connecter avec l'identifiant "admin" et le mot de passe "admin". Une fois connecté, veuillez corriger le nom de Michel Cerond qui s'écrit "Ceront" et sa performance, il a en réalité mis 3 heures, 22 minutes et 34 secondes à accomplir la course.

- ❏ Protocole de test de renommage d'une course

Vous êtes un administrateur et vous venez de voir qu'une coquille s'est glissée dans les résultats de la course 1. Vous souhaitez donc la corriger.
Premièrement, vous devez vous connecter avec l'identifiant "admin" et le mot de passe "admin". Une fois connecté, renommez la course 1 en "Course Pessac-Mérignac".

D. Résultats des tests.

Les tests nous ont permis de pouvoir déboguer rapidement en cas de soucis. En fin de projet, tous les tests étaient évidemment validés.

V. Bilan et Perspectives

A. Points positifs et négatifs.

Notre solution a l'avantage de répondre à toutes les exigences formulées par le client. Nous avons également essayé de créer une interface simple et intuitive afin d'en faciliter la prise en main. Cette volonté de sobriété peut cependant nuire légèrement au design qui est peu évolué. Pour autant, l'application a été conçue principalement dans l'optique d'être efficace et sans fioriture. Nous avons en effet privilégié un fonctionnement instinctif et rapide pour que les futurs utilisateurs puissent rapidement prendre en main l'outil et y entrer ou en extraire les informations nécessaires.
En résumé, mis à part un design relativement léger, notre solution est efficace, répond à toutes les exigences évoquées par le client et a pour objectif d'être adapté aux futurs utilisateurs qui ne sont pas là pour se prendre la tête, mais pour encadrer un évènement sportif.

B. Evolutions possibles.

Comme nous l'avons souligné plus tôt, une piste d'amélioration serait l'évolution du design. En effet, il est possible tout en restant sobre, de créer une interface un peu moins austère. Deuxièmement, il n'est pas possible de supprimer un participant. Nous avons effectivement pensé qu'un participant était supposé participé à la course et qu'en cas d'absence, il suffirait simplement de mettre sa performance en "non défini". Ainsi, il est possible de garder une trace des personnes n'ayant pas pu venir. Cependant, nous pouvons imaginer que les utilisateurs puissent avoir un autre point de vu et c'est pourquoi ajouter cette fonctionnalité pourrait être bien. Enfin, une dernière petite chose pourrait être amélioré mais sa non réalisation est dûe à l'outil WinForms. Lorsqu'on veut éditer directement dans le dataGridView, il faut absolument cliquer au moins une fois sur l'écriture à l'intérieur et non pas seulement sur le blanc autour auquel cas la case ne serait pas "sélectionnée" selon WinForms bien qu'on puisse écrire dedans. Ce problème de sélection empêche notre fonction de mise à jour de la BDD de fonctionner correctement dans ce cas, ce qui peut être problématique.
Pour conclure, un projet pareil peut toujours être amélioré, que ce soit par l'ajout de nouvelles fonctionalités ou par une amélioration de l'interface, ce qui pourrait être fait après les premiers essais en cas réel et tout au long de la vie du projet.