26/04/2021

Paola Costa

Sous la supervision de :  
C.Egger, chef de projet  
G.Gruaz, expert 1  
O.Rutz, expert 2

Gestion de cave à vin

Application en C# et .Net

Table des matières

[1 Analyse préliminaire 3](#_Toc73372332)

[1.1 Introduction 3](#_Toc73372333)

[1.2 Objectifs 3](#_Toc73372334)

[1.3 Planification initiale 4](#_Toc73372335)

[2 Analyse / Conception 7](#_Toc73372336)

[2.1 Cadre du projet 7](#_Toc73372337)

[2.2 Concept 7](#_Toc73372338)

[2.2.1 Fonctionnalités 7](#_Toc73372339)

[2.2.2 Modèles de données 8](#_Toc73372340)

[2.2.2.1 Modèle de données conceptuel 8](#_Toc73372341)

[2.2.2.2 Modèle de données logique 9](#_Toc73372342)

[2.3 Maquettes 10](#_Toc73372343)

[2.4 Schéma de navigation 11](#_Toc73372344)

[2.5 Use Cases & Scénarii 12](#_Toc73372345)

[2.5.1 Use Cases 12](#_Toc73372346)

[2.5.2 Scénarios 13](#_Toc73372347)

[2.6 Diagrammes de classe 19](#_Toc73372348)

[2.7 Risques techniques 20](#_Toc73372349)

[2.8 Stratégie de test 20](#_Toc73372350)

[2.9 Infrastructure 21](#_Toc73372351)

[2.9.1 Matériel hardware et système d’exploitation 21](#_Toc73372352)

[2.9.2 Outils logiciels 21](#_Toc73372353)

[2.9.3 Architecture du projet 21](#_Toc73372354)

[*2.10* Planification définitive 22](#_Toc73372355)

[3 Réalisation 25](#_Toc73372356)

[*3.1* Dossier de réalisation 25](#_Toc73372357)

[3.1.1 Répertoires et fichiers du projet 25](#_Toc73372358)

[3.1.1.1 Répartition physique des fichiers 25](#_Toc73372359)

[3.1.1.2 Fichiers et description 25](#_Toc73372360)

[3.1.2 Produit fini 26](#_Toc73372361)

[3.2 Liste des éléments fournis 26](#_Toc73372362)

[3.2.1 Scripts 27](#_Toc73372363)

[3.3 Description des tests effectués 28](#_Toc73372364)

[3.3.1 Tests unitaires 28](#_Toc73372365)

[3.3.2 Tests fonctionnels 28](#_Toc73372366)

[3.3.3 État des tests 29](#_Toc73372367)

[3.4 Problèmes rencontrés et résolution 30](#_Toc73372368)

[3.5 Erreurs restantes 31](#_Toc73372369)

[3.6 Comparaison des délais entre la planification et la réalisation (TODO) 32](#_Toc73372370)

[4 Conclusions 33](#_Toc73372371)

[4.1 Atteinte des objectifs 33](#_Toc73372372)

[4.2 Maintien des délais (TODO) 33](#_Toc73372373)

[4.3 Points positifs et négatifs (TODO) 33](#_Toc73372374)

[4.4 Difficultés particulières 34](#_Toc73372375)

[4.5 Évolutions et améliorations 34](#_Toc73372376)

[5 Annexes - sommaire 35](#_Toc73372377)

Table des illustrations

[Figure 1 : planification initiale partie 1 4](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372425)

[Figure 2 : planification initiale partie 2 5](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372426)

[Figure 3 : planification initiale partie 3 6](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372427)

[Figure 4 : modèle conceptuel de données 8](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372428)

[Figure 5 : modèle logique de données 9](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372429)

[Figure 6 : maquette -> page d'accueil de l'application 10](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372430)

[Figure 7 : maquette -> page d'ajout de bouteille(s) 10](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372431)

[Figure 8 : schéma de navigation 11](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372432)

[Figure 9 : diagramme des Use Case 12](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372433)

[Figure 10 : diagramme de classe 19](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372434)

[Figure 11 : planification définitive partie 1 22](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372435)

[Figure 12 : planification définitive partie 2 23](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372436)

[Figure 13 : planification définitive partie 3 24](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372437)

[Figure 14 : BDD -> génération de table 27](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372438)

[Figure 15 : BDD -> insertion de données 27](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372439)

[Figure 16 : problème rencontré -> référence de projet 30](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372440)

[Figure 17 : erreur -> recherche par mot-clé 31](file:///C:\Users\paola.costa\Documents\GitHub\TPI_gestionCaveAVin\docs_avec_modification\rapport_COSTA.docx#_Toc73372441)

# Analyse préliminaire

## Introduction

Lors d’un repas de famille ou avec des amis, il est toujours plaisant d’avoir une bonne bouteille de vin sortie. Cependant, la nécessité de fouiller pendant plusieurs minutes la cave à vin pour trouver une bouteille adéquate n’est jamais agréable. Cette application a donc pour but de vous laisser choisir la bonne bouteille directement depuis votre ordinateur. Afin de valider mon CFC et dans le cadre de mon TPI, je vais réaliser une application de gestion de cave à vin. Il s’agit d’une application prévue pour un privé. Elle permettra à une personne, même novice en informatique, de gérer des casiers à bouteilles, d’y ajouter ou enlever des bouteilles, d’effectuer une recherche selon des critères particuliers, de consulter l’historique des actions effectuées, ainsi que d’associer une alerte à une bouteille particulière. L’intégralité des données propres à l’application seront stockées dans une base de données.

Ce projet a comme date de début le lundi 03 mai 2021, 08h50. Sa date de rendu finale est le mercredi 02 juin 2021, 10h35. Cela donne un total de 90h pour le réaliser. L’application est réalisée en C#, à l’aide de base de données MySQL.

Enfin, le travail de pré-TPI a été réalisé en amont. Son but a été de revoir les différentes technologies qui seront abordées dans ce projet. Le canevas du rapport et du journal de travail a été récupéré depuis ce précédent travail et adapté. L’intégralité du développement se fera lors du module et du temps mis à disposition.

## Objectifs

Afin de mener à bien ce projet, de nombreux objectifs sont à compléter. La validation de ceux-ci permettra de déterminer le degré de complétion du projet. L’élément principal de ce projet consiste à créer une application « clé en main ». Cela signifie que l’application sera fonctionnelle sans investissements ultérieurs. Ensuite, l’application doit être accessible à des personnes ayant très peu de notions d’informatique. Son fonctionnement sera donc intuitif.

De plus, l’application contiendra plusieurs fonctionnalités. Il s’agit de celles qui sont citées ci-dessous :

* L’application doit permettre d’ajouter des bouteilles à la cave.
* L’application doit permettre de retirer des bouteilles de la cave.
* Il est possible d’effectuer une recherche selon des critères spécifiques.
* Les données de l’application sont stockées dans une base de données.
* L’application doit permettre d’exporter un PDF, contenant une liste de bouteilles.
* L’application doit permettre d’imprimer une liste de bouteilles spécifiques.
* L’application doit permettre d’afficher l’historique des actions effectuées dans l’application.
* L’application doit permettre d’organiser la case en casiers à bouteilles.
* L’application doit permettre d’ajouter des casiers à bouteilles.
* L’application doit permettre de supprimer des casiers à bouteilles.
* L’application doit permettre d’ajouter une alerte spécifique à une/des bouteille(s).

Enfin, afin de faciliter la mise en place de l’application, une procédure d’installation sera également fournie.

## Planification initiale

Comme le CdC fourni début mai est complet et contient l’intégralité des informations nécessaires pour pouvoir mener à terme le projet, celui-ci se déroulera en mode cascade. La répartition des tâches est effectuée dès le début. Comme discuté avec monsieur C.Egger, les différentes planifications seront réalisées sur Excel. Un onglet « avancées » permet de suivre l’état des différentes tâches, afin de savoir en permanence où en est la réalisation du projet.

Sur la figure ci-dessous, on peut voir la répartition des tâches liées à l’analyse et à la conception. L’analyse préliminaire est réalisée entièrement le lundi 03 mai 2021, afin de permettre l’envoi de la planification initiale ce même jour. L’intégralité de ces tâches mènent jusqu’à la planification définitive, qui sera envoyée, comme convenu avec monsieur G.Gruaz, au plus tard jeudi 06 mai 2021. Chaque colonne représente un bloc d’une période scolaire, soit 45 minutes.

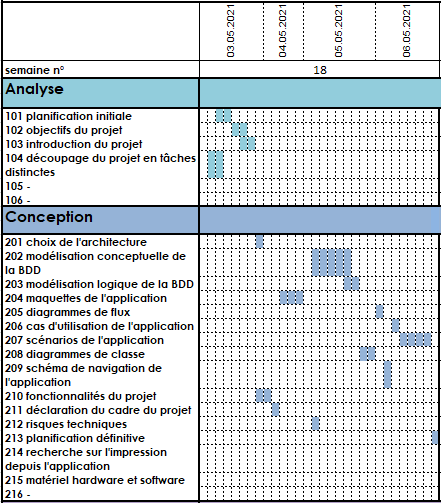


Figure 1 : planification initiale partie 1

Sur la figure ci-contre, il est possible de voir la répartition des tâches concernant l’implémentation.

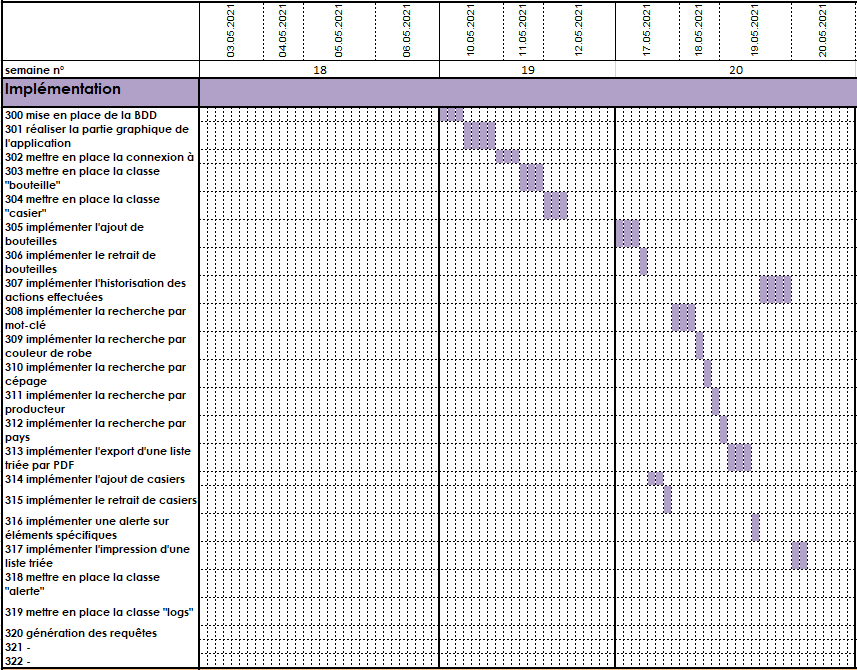


Figure 2 : planification initiale partie 2

L’entièreté du développement devrait pouvoir se réaliser sur deux semaines, afin de laisser suffisamment de temps à la fin pour pouvoir tester. Cela permet également de déborder un peu en cas de souci de développement.

Prévoir l’implémentation avec autant de délai à la fin donne ainsi un petit peu de marge.

Le reste du temps sera consacré à la rédaction du rapport de projet ainsi qu’à la procédure liée à l’installation.

Comme mentionné plus haut, les tests seront réalisés principalement à la fin du développement. Une partie sera malgré tout faite pendant la réalisation, afin de pouvoir s’assurer du bon fonctionnement du programme. Cependant, les tests des limites de l’application seront réalisés après que l’application soit fonctionnelle.

Les lignes avec un fond jaune représentent des tâches récurrentes, qui n’ont pas de durée définie. Il s’agit de tâches qui sont réalisées en continu, selon les besoins. C’est le cas pour le remplissage du journal de travail, de la mise en forme du rapport de projet et des différents tests.

De plus, les lignes concernant les réunions avec le CdP et les experts ne sont pas remplies, car les créneaux associés ne sont pas définis. Il s’agit donc de temps utilisé qui ne peut pas être prévu lors de la planification initiale.

Enfin, une comparaison des variations entre la planification initiale, la planification définitive (après l’analyse/conception) et la planification finale (après la réalisation) sera effectuée à la fin du projet. Le but de cette analyse sera d’étudier les différences et d’en déterminer leurs raisons.

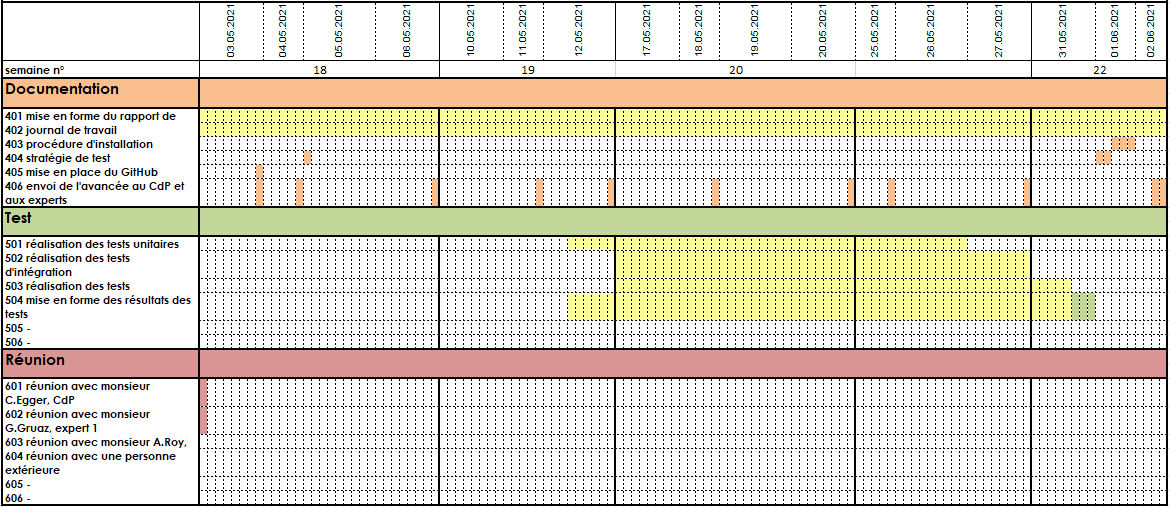


Figure 3 : planification initiale partie 3

# Analyse / Conception

## Cadre du projet

Comme cela a été mentionné précédemment, le temps à disposition pour réaliser le projet est fixe. 90 heures sont allouées pour le faire. Ces 90 heures commencent le lundi 03 mai 2021 à 8h50 et se terminent le mercredi 02 juin 2021 à 10h35. Aucun délai supplémentaire ne sera accordé.

Ensuite, seules les fonctionnalités mentionnées ci-après vont être implémentées. Les éléments d’amélioration ou d’évolution du projet, imaginés pendant la réalisation, seront mentionnées dans la conclusion, au point [Évolutions et améliorations](#_Évolutions_et_améliorations). Aucun élément complémentaire ne sera développé.

Enfin, toutes les technologies utilisées lors de ce projet ont été vues en cours. Même si certaines spécificités techniques ne sont pas connues, le langage en lui-même est un sujet étudié et maîtrisé.

## Concept

### Fonctionnalités

En plus des différents éléments abordés dans le point [Objectifs](#_Objectifs) de l’analyse préliminaire, plusieurs points techniques spécifiques doivent être respectés lors de la réalisation de ce projet. Ils sont explicités ci-dessous :

* Les données internes à l’application sont stockées dans une base de données MySQL.
* Au lancement de l’application, une connexion à la base de données est établie. Cette connexion est fermée lorsque l’application est quittée.
* Il est possible de gérer l’agencement de la cave en casiers à bouteille, dans lesquels les bouteilles sont placés.
* Il est possible d’ajouter des casiers à bouteilles supplémentaires.
* Il est possible de retirer des casiers à bouteilles.
* Il est possible d’ajouter des bouteilles.
* Il est possible de sortir (supprimer) des bouteilles.
* Il est possible d’ajouter une alerte, liée à certaines bouteilles spécifiques. Accompagnée d’un commentaire, cette alerte permet de « réserver » une bouteille pour une occasion spéciale.
* Il est possible d’effectuer une recherche par mot-clé sur l’application.
* Il est possible de trier les bouteilles présentes selon plusieurs critères spécifiques. Ces critères sont les suivants : le cépage, le producteur, le pays ou la robe du vin.
* Il est possible d’exporter, au format PDF, une liste des bouteilles respectant un critère particulier.
* Une historisation des actions effectuées sur l’application est disponible.

Des détails supplémentaires concernant ces fonctionnalités peuvent être trouvés dans le sous-chapitre [Use Cases & Scénarios](#_Use_Cases_&). Le fonctionnement exact de chaque point abordé ci-dessus y est développé.

### Modèles de données

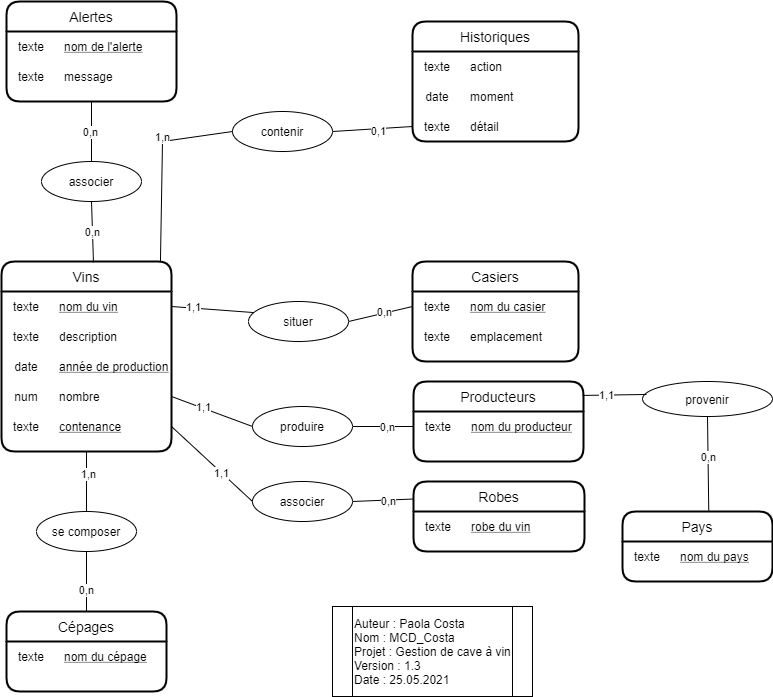
#### Modèle de données conceptuel

Le modèle conceptuel de données (MCD) ci-dessous représente les relations entre l’intégralité des données de l’application.

Parmi les éléments spécifiques à ce modèle, voici les points particuliers :

* L’entité « Robes » représente la couleur du vin. La liste de valeurs possibles sera restreinte aux valeurs les plus courantes : blanc, rosé et rouge.
* L’entité « Producteurs » comportera les différents producteurs à partir duquel provient le vin. Aucune méthode d’ajout de producteur n’est prévue pour le moment.
* L’entité « Casiers » gère des zones de stockage. Le texte « emplacement » est à titre indicatif pour l’utilisateur, il n’a aucun impact sur la réalité.
* L’entité « Historiques » sert à garder un historique des actions. Il comporte la date et l’heure exacte, l’action associée et le détail de ce qu’il s’est passé. Il contient soit une relation avec un vin, soit une relation avec un casier.

Figure 4 : modèle conceptuel de données



#### Modèle de données logique

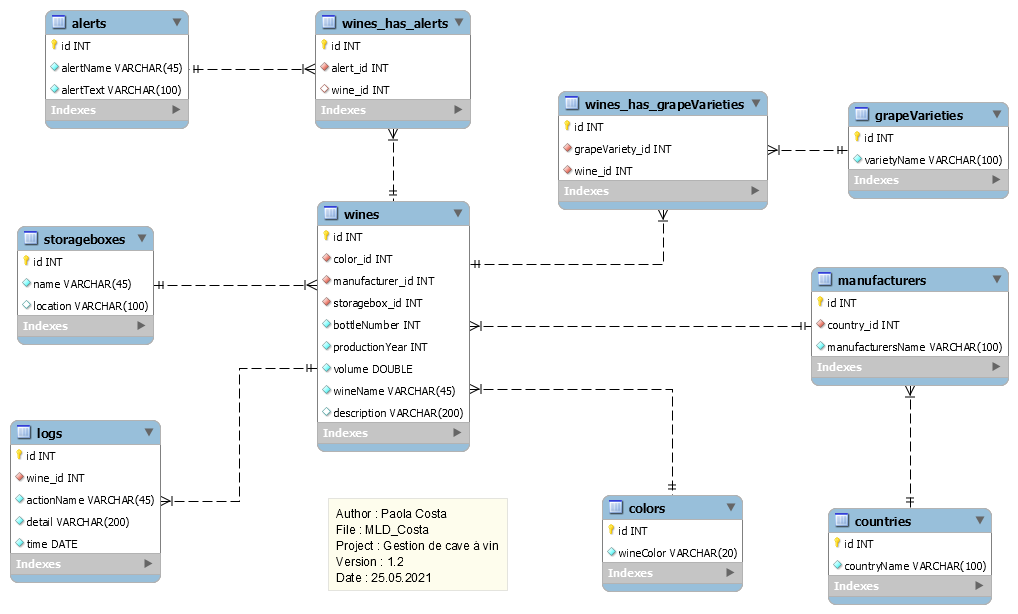


Figure 5 : modèle logique de données

Le modèle logique de données montre l’intégralité des tables présentes pour stocker les données. Ci-dessous se trouve une explication des différentes tables présentes :

* manufacturers : il s’agit du producteur du vin. Chaque producteur est associé à un seul pays.
* countries : il s’agit du pays dans lequel se trouve le producteur.
* colors : il s’agit de la couleur du vin, avec trois valeurs possibles : rouge, rosé et blanc.
* logs : il s’agit de l’historique des actions effectuées sur les bouteilles. Chaque action a un nom, une bouteille associée, une date de réalisation et le détail de ce qui est fait.
* storageboxes : il s’agit des casiers dans lesquels sont stockés les bouteilles.
* alerts : regroupe toutes les alertes. Celles-ci sont associées aux bouteilles.
* wines\_has\_alerts : il s’agit du lien entre les alertes et les bouteilles. La même alerte peut être attribuée à plusieurs vins différents et un vin peut avoir plusieurs alertes.
* grapeVarieties : il s’agit des cépages utilisés pour effectuer les assemblages. Comme le nombre de cépages existants est très important, une liste réduite sera utilisée ici.
* wines\_has\_grapevarieties : il s’agit du lien entre les vins et les cépages. Un vin peut être composé de plusieurs cépages et un cépage peut faire partie de plusieurs vins.
* wines : il s’agit des différents vins présents dans la cave. Un vin est défini par la combinaison nom, année, volume.

## Maquettes

Afin de se représenter un minimum l’application, plusieurs maquettes vont être réalisées dans le cadre de ce TPI. Celles-ci vont être ajoutées ci-dessous et expliquées. Leur but principal est de donner une idée du visuel qui sera obtenu à la fin du projet.

Cette première maquette représente la vision que l’utilisateur a lors de l’ouverture de l’application.

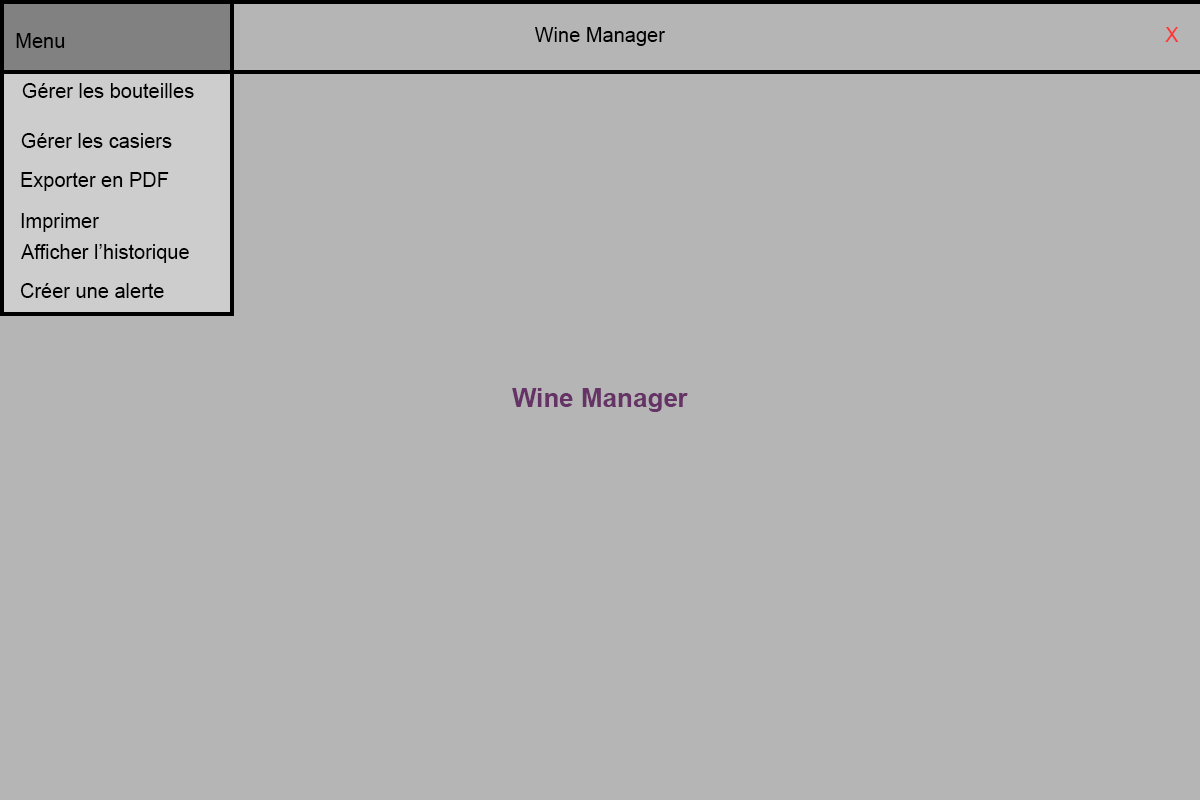


Figure 6 : maquette -> page d'accueil de l'application

Il s’agit d’une page d’accueil, sur laquelle on peut trouver le nom de l’application.

Les diverses fonctionnalités sont utilisables par un menu déroulant, situé en haut à gauche.

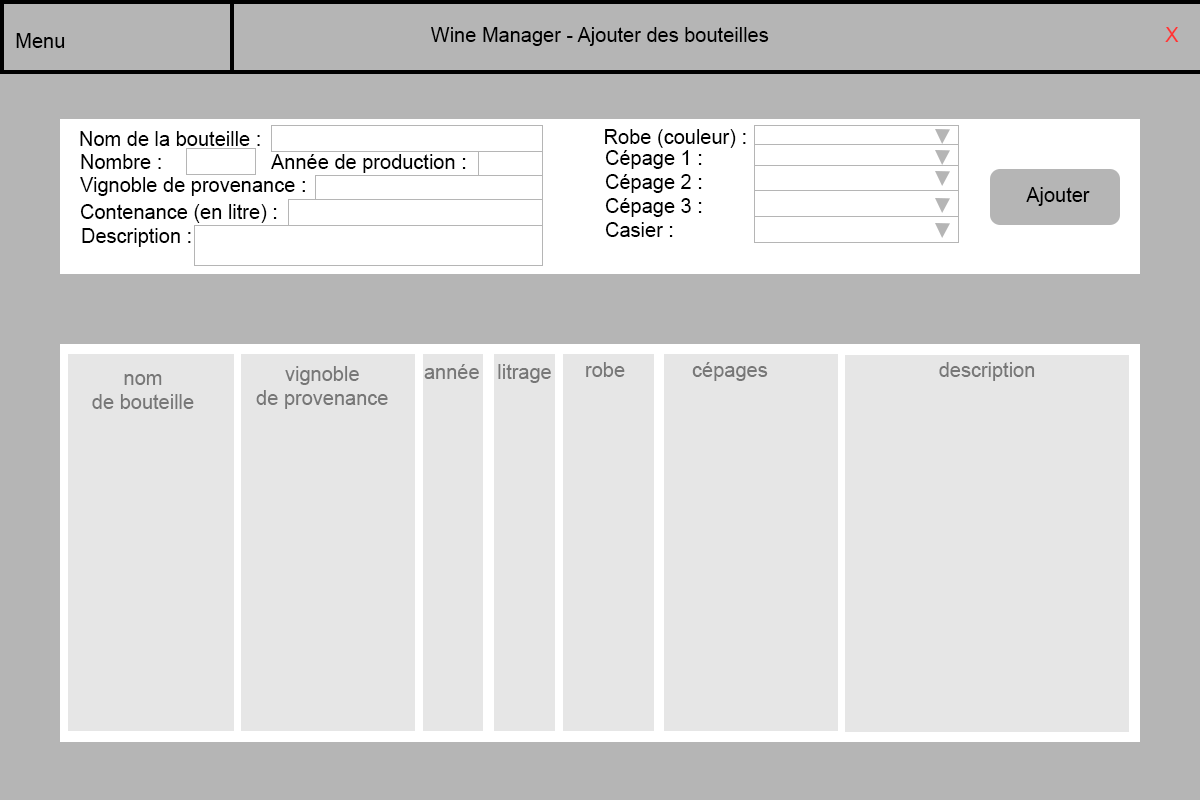


Figure 7 : maquette -> page d'ajout de bouteille(s)

Sur la maquette ci-contre, on peut voir la page sur laquelle il est possible d’ajouter des bouteilles à la cave.

La zone de formulaire en haut de l’écran permet d’entrer les données à ajouter, la zone inférieure permet de voir ce qui se trouve déjà dans la cave.

Le retrait de bouteilles se fait sur une page au visuel semblable à celui de la figure 7. Les modifications sont que la zone de formulaire ne contient pas les mêmes champs. Il contient plusieurs champs de sélection, tels que : nom de bouteille, année, producteur, contenance et nombre.

Pour la partie de l’application permettant de gérer les casiers, le visuel est sur la même base que ci-dessus. Seulement, au lieu d’afficher les bouteilles en-dessous, on trouve l’affichage des différents casiers.

## Schéma de navigation

Le schéma de navigation ci-dessous représente l’organisation de l’application. Chaque losange représente une des options du menu (voir figure 6, dans la partie [Maquettes](#_Maquettes)). Les rectangles sont les différentes pages de l’application. Le texte écrit en dessous de cela apporte des précisions sur le contenu de la page.

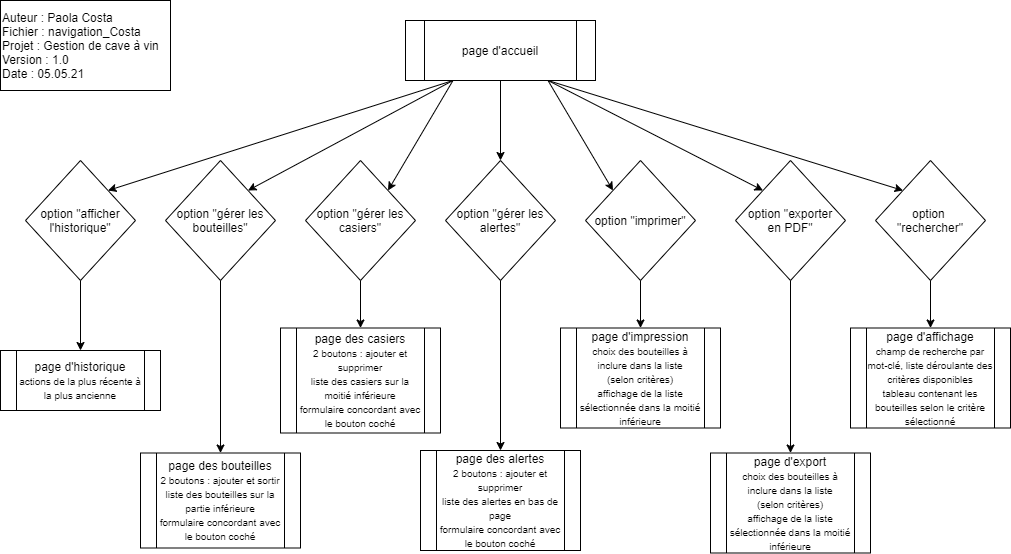


Figure 8 : schéma de navigation

## Use Cases & Scénarii

### Use Cases

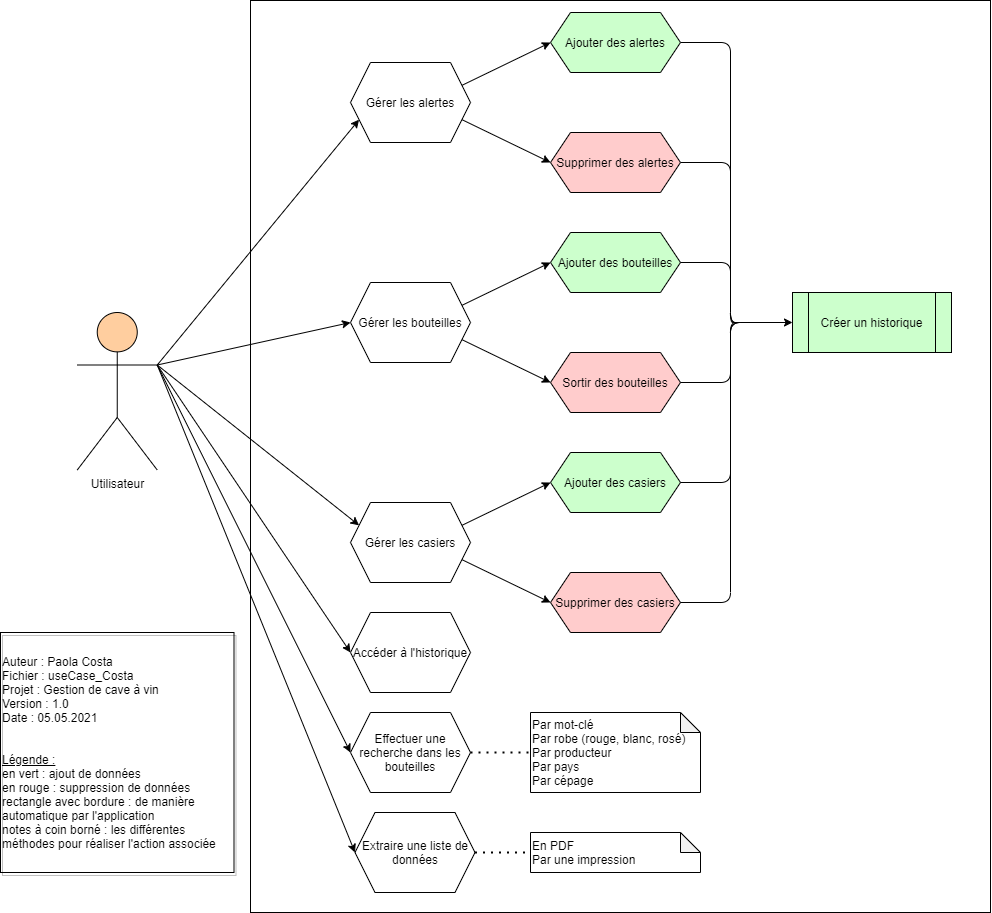


Figure 9 : diagramme des Use Case

Le diagramme ci-dessus représente les cas d’utilisation de l’application. Chaque hexagone au fond blanc représente une action, réalisable par l’utilisateur. Les hexagones verts et rouges représentent des actions plus spécifiques, respectivement des ajouts de données et des suppressions de données.

Le rectangle vert représente une action réalisée automatiquement par l’application, lorsqu’une des actions connectées est effectuée.

Les rectangles au coin corné représentent les différentes possibilités d’effectuer l’action à laquelle ils sont rattachés.

### Scénarios

Les scénarios représentent les différentes actions réalisables dans l’application. Il s’agit d’une démarche détaillée du fonctionnement d’un élément précis. Ces scénarios vont être utilisés pour réaliser les tests fonctionnels par la suite. Ils sont donc tous associés à un numéro et à un nom, afin de pouvoir retrouver le test correspondant le moment venu.

Situation générique

Personne : utilisateur non authentifié

Emplacement dans l’application : page d’accueil

Scénario 1 : Gérer les bouteilles

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne sélectionne l’option « gérer les bouteilles » dans le menu. |  | La page qui s’affiche est la page des bouteilles de la cave. |

Scénario 1a : Ajouter une bouteille

Situation : l’utilisateur non authentifié est déjà sur la page des bouteilles (voir scénario 1)

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne sélectionne le bouton « ajouter une bouteille » | L’option est déjà sélectionnée | L’affichage ne change pas, le formulaire affiché est celui d’ajout de bouteille(s) |
|  | L’option sélectionnée a changé | Le formulaire d’ajout de bouteille(s) s’affiche |
| L’utilisateur remplit les champs du formulaire (nom, nombre de bouteilles, contenance, année de production) et sélectionne des valeurs dans les listes déroulantes (producteur, robe, cépage) | L’un des champs obligatoires n’est pas rempli | Un message d’erreur s’affiche |
|  | Un champ est rempli avec une valeur non supportée | Un message d’erreur s’affiche |
|  | Tous les champs sont remplis correctement | Retour à la page des bouteilles, avec une actualisation de la liste des bouteilles |

Scénario 1b : Sortir une bouteille

Situation : l’utilisateur non authentifié est déjà sur la page des bouteilles (voir scénario 1)

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne sélectionne le bouton « sortir une bouteille » | L’option est déjà sélectionnée | L’affichage ne change pas, le formulaire affiché est celui de retrait de bouteille(s) |
|  | L’option sélectionnée a changé | Le formulaire de retrait de bouteille(s) s’affiche |
| L’utilisateur sélectionne des valeurs dans les listes déroulantes (nom, année, producteur, contenance) et remplit le champ spécifiant le nombre de bouteilles à sortir | L’un des champs requis n’est pas rempli | Un message d’erreur s’affiche |
|  | Un champ est rempli avec une valeur non supportée | Un message d’erreur s’affiche |
|  | La combinaison des valeurs sélectionnées n’existe pas dans les bouteilles existantes | Un message d’erreur s’affiche |
|  | Les champs nécessaires sont remplis correctement | Retour à la page des bouteilles, avec une actualisation de la liste des bouteilles |

Scénario 2 : Gérer les casiers

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne sélectionne l’option « gérer les casiers » dans le menu. |  | La page qui s’affiche est la page des casiers de la cave. |

Scénario 2a : Ajouter un casier

Situation : l’utilisateur non authentifié est déjà sur la page des casiers (voir scénario 2)

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne sélectionne le bouton « ajouter une casier » | L’option est déjà sélectionnée | L’affichage ne change pas, le formulaire affiché est celui d’ajout de casier |
|  | L’option sélectionnée a changé | Le formulaire d’ajout de casier(s) s’affiche |
| L’utilisateur remplit les champs du formulaire (nom, emplacement) | L’un des champs obligatoires n’est pas rempli | Un message d’erreur s’affiche |
|  | Un champ est rempli avec une valeur non supportée | Un message d’erreur s’affiche |
|  | Les champs nécessaires sont remplis correctement | Retour à la page des casiers, avec une actualisation de la liste des casiers |

Scénario 2b : Supprimer un casier

Situation : l’utilisateur non authentifié est déjà sur la page des casiers (voir scénario 2)

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne sélectionne le bouton « supprimer un casier » | L’option est déjà sélectionnée | L’affichage ne change pas, le formulaire affiché est celui de suppression de casier |
|  | L’option sélectionnée a changé | Le formulaire de suppression de casier s’affiche |
| L’utilisateur sélectionne le casier à supprimer dans la liste déroulante | Le casier contient encore des bouteilles | Un message d’erreur s’affiche |
|  | Le casier est vide | Retour à la page des casiers, avec une actualisation de la liste des casiers |

Scénario 3 : Gérer les alertes

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne sélectionne l’option « gérer les alertes » dans le menu. |  | La page qui s’affiche est la page des alertes. |

Scénario 3a : Ajouter une alerte

Situation : l’utilisateur non authentifié est déjà sur la page des alertes (voir scénario 3)

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne sélectionne le bouton « ajouter une alerte » | L’option est déjà sélectionnée | L’affichage ne change pas, le formulaire affiché est celui d’ajout d’alerte |
|  | L’option sélectionnée a changé | Le formulaire d’ajout d’alerte s’affiche |
| L’utilisateur remplit les champs du formulaire (nom, description) et sélectionne dans la liste déroulante la ou les bouteilles à associer (nom, producteur, année de production, contenance) | L’un des champs obligatoires n’est pas rempli | Un message d’erreur s’affiche |
|  | Un champ est rempli avec une valeur non supportée | Un message d’erreur s’affiche |
|  | Les champs nécessaires sont remplis correctement | Retour à la page des alertes, avec une actualisation de la liste des alertes |

Scénario 3b : Supprimer une alerte

Situation : l’utilisateur non authentifié est déjà sur la page des alertes (voir scénario 3)

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne sélectionne le bouton « supprimer une alerte» | L’option est déjà sélectionnée | L’affichage ne change pas, le formulaire affiché est celui de suppression d’alerte |
|  | L’option sélectionnée a changé | Le formulaire de suppression d’alerte s’affiche |
| L’utilisateur sélectionne l’alerte à supprimer dans la liste déroulante |  |  |
| L’utilisateur clique sur le bouton « supprimer l’alerte » |  | Retour à la page des alertes, avec une actualisation de la liste des alertes |

Scénario 4a : Effectuer une recherche par mot-clé

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| L’utilisateur sélectionne l’option « Rechercher » dans le menu |  | Affichage d’une page avec toutes les bouteilles dans un tableau. Une liste déroulante se trouve à côté d’un champ texte en haut à droite, à côté d’un bouton « Rechercher » |
| L’utilisateur entre du texte dans le champ texte à côté du bouton « Rechercher »  L’utilisateur clique sur le bouton « Rechercher » | Le texte spécifié n’a aucune correspondance dans les noms et descriptions des bouteilles | Le tableau s’actualise et s’affiche vide. |
|  | Correspondance entre le texte entré et le nom et la description d’au moins 1 bouteille | Le tableau s’actualise et affiche les bouteilles correspondant à la recherche. |

Scénario 4b : Filtrer selon un critère

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| L’utilisateur sélectionne l’option « Rechercher » dans le menu |  | Affichage d’une page avec toutes les bouteilles dans un tableau. Des boutons se situent à côté d’un champ texte en haut à droite, à côté d’un bouton « Rechercher » |
| L’utilisateur sélectionne un des filtres présents en tant que bouton | Aucune bouteille ne correspond au critère sélectionné | Le tableau s’actualise et s’affiche vide. |
|  | Le critère sélectionné est le même que la liste déjà affichée | Rien ne se passe. |
|  | Au moins 1 bouteille correspond au critère sélectionné | Le tableau s’actualise et affiche les bouteilles correspondant au critère. |

Scénario 5 : Afficher l’historique

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne sélectionne l’option « Afficher l’historique » dans le menu. |  | La page qui s’affiche est la page de l’historique. |

Scénario 6 : Exporter au format PDF

Situation : l’utilisateur non authentifié est déjà sur la page des recherches.

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne clique sur « exporter en PDF |  | Un fichier PDF se crée sur le disque C: |

Scénario 7 : Imprimer

Situation : l’utilisateur non authentifié est déjà sur la page des recherches.

|  |  |  |
| --- | --- | --- |
| Action | Condition | Réaction |
| La personne sélectionner « imprimer » |  | L’impression du fichier se lance |

## Diagrammes de classe

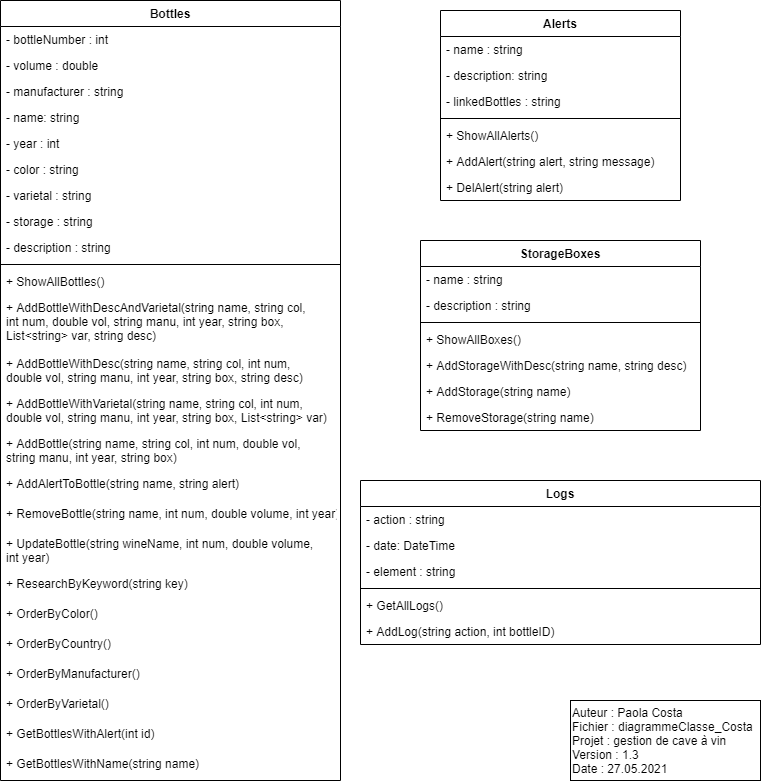


Figure 10 : diagramme de classe

Le diagramme de classe reprend les différentes classes nécessaires pour la partie graphique de l’application. Afin de pouvoir afficher des données dans l’application, il est nécessaire de stocker temporairement les données récupérées depuis la base de données, afin de pouvoir les afficher par la suite.

Les différentes classes ci-dessus servent exactement à ça :

* La classe « Bottles » permet de stocker les données liées à l’affichage des bouteilles.
* La classe « Alerts » permettra d’afficher les données liées aux alertes.
* La classe « StorageBoxes » permettra d’afficher les données liées aux casiers.
* La classe « Logs » permettra d’afficher l’historique des actions de l’application.

## Risques techniques

Même si idéalement, il faudrait qu’aucun risque ne soit présent, il n’en est rien. De nombreux problèmes peuvent se poser lors du déroulement du projet.

Ainsi, un des éléments qui peut venir impacter l’avancée est un souci technique. Il est possible que, lors de la réalisation, un point spécifique du développement embête. Cela entraînerait des retards. La conséquence finale serait que le projet ne pourrait pas être complété dans les délais, laissant l’implémentation inachevée.

Afin d’éviter ce problème, le rythme d’avancée du projet proposée est soutenu. Cela laisse de la marge à la fin, au cas où un retard arriverait, afin que la réalisation puisse quand même être complétée. La demande d’aide à un professionnel, avant que cela ne s’éternise trop, est également une solution.

Un autre élément qui peut être problématique sont les absences. En effet, particulièrement avec le COVID-19 actuellement, il n’est pas inenvisageable qu’une quarantaine ou un confinement soit nécessaire alors que le développement devrait se faire. Une absence plus courte, due à la maladie, est également possible.

Si la situation d’une quarantaine devait se présenter, une prise de contact avec le CdP et les experts aurait lieu dès que possible. Cette discussion aurait pour but de déterminer des actions à suivre. Les deux solutions possibles sont : repousser la date de rendu, selon le temps d’inactivité ou autoriser l’avancée du projet depuis un lieu autre que le CPNV.

Enfin, il est possible que la motivation ne soit pas là 100% du temps. Cela aurait pour conséquence, encore une fois, d’entraîner des retards. Afin d’éviter ces baisses de motivation, la mise en place d’un fichier d’avancée des tâches, qui permet de voir où le projet en est, a été mis en place. Le fait de voir des tâches passer de « en cours » à « terminé » chaque jour motive et permet d’encourager la personne à continuer de travailler sur le projet.

## Stratégie de test

Afin de limiter au maximum les problèmes techniques, des tests réguliers sont prévus, une fois la réalisation commencée. Ces tests ont pour but de détecter les erreurs avant que le développement n’ait trop avancé, pour que la perte de temps soit minimale. Ceux-ci ne seront pas validés, il s’agit juste de repérer les erreurs de compilation du programme.

Plusieurs types de tests vont donc être réalisés lors de ce projet. Il s’agit de tests unitaires et de tests fonctionnels. Les tests unitaires servent à tester une unité, un bloc de code ayant une fonction bien spécifique, alors que les tests fonctionnels permettent de vérifier que l’entièreté des unités interagissent correctement et retourne le résultat attendu.

Pour que le test unitaire ait un sens, il faut qu’un élément logique entre en compte dans l’application. Ainsi, seules les fonctions incluant de la logique seront testées avec des tests unitaires. Ces tests seront réalisés grâce aux classes de tests en C#.

Par rapport aux tests fonctionnels, leur réalisation se fait sur la base des [Scénarios](#_Scénarios). Chaque scénario représente la démarche d’un test spécifique. Si le résultat obtenu correspond au résultat attendu dans le scénario, le test est considéré réussi.

Les tests liés aux scripts de la base de données seront uniquement fonctionnels. Lors de la génération de la base de données avec MySQL Workbench, si aucune erreur ne s’affiche dans la console, le test des scripts sera considéré comme validé.

## Infrastructure

### Matériel hardware et système d’exploitation

L’infrastructure hardware utilisée pour réaliser ce projet consiste uniquement en un ordinateur du CPNV, installé avec un système d’exploitation Windows10 x64. Aucun matériel supplémentaire n’est nécessaire.

### Outils logiciels

Afin de mener à bien ce projet, divers outils logiciels vont être nécessaires. Ces logiciels sont listés ci-dessous. L’utilisation d’autres versions des logiciels peut potentiellement entraîner une incompatibilité et rendre le projet inaccessible, ou avec des erreurs qui ne sont pas présentes autrement.

Développement en C# : Microsoft Visual Studio Enterprise 2019, Version 16.8.5

Paquet : MySql.Data, version 8.0.24 (intégration du MySQL)

Paquet : iText7, version 7.1.15 (création de PDF)

Paquet : FreeSpire.PDF, version 7.2.0 (impression d’un PDF)

Base de données : MySQLWorkbench, version 6.3.6 build 511 CE (64bits)

Maquettes : Adobe Photoshop 2019, version 10.0.6

Diagramme de classe, MCD, schéma de navigation : draw.io, version 13.9.9

Rapport de projet : Microsoft Word 2016 MSO, version 16.0.4266.1001

Planifications, journal de travail, avancée des tâches : Microsoft Excel 2016 MSO, version 16.0.4266.1001

Historisation des données : L’intégralité du pré-TPI est disponible sur GitHub. Les données sont présentes à l’adresse suivante :

<https://github.com/AhVen98/TPI_gestionCaveAVin>

GitHub est la méthode de versioning utilisée dans ce projet. Il s’agit du moyen mis en place pour pouvoir récupérer les anciennes versions du projet en cas de soucis/besoins.

### Architecture du projet

L’architecture qui sera utilisée dans ce projet sera une architecture MVVM. Il s’agit d’une architecture qui effectue une liaison entre les vues et les modèles, et ce dans les deux sens. Cela signifie que les vues interagissent avec les modèles et que les modèles interagissent avec les vues.

L’architecture MVC (modèle, vue, contrôleur) est une architecture qui aurait potentiellement pu être utilisé. Cependant, lors du pré-TPI, j’ai réalisé un projet de gestion en C# en essayant d’intégrer le MVC. Il m’a été nécessaire, moins d’une semaine avant la reddition du projet, de faire marche arrière et d’enlever ce contrôleur, car une fonctionnalité de base n’était pas compatible avec sa présence et bloquait toute la réalisation. Ce choix a donc été abandonné et ne sera pas réitéré lors du TPI.

## Planification définitive

On peut voir ci-après la planification définitive, c’est-à-dire la planification après l’analyse et la conception. L’illustration qui suit représente la partie analyse et conception, qui est donc terminé et ne sera plus modifiée.

Les variations entre la planification préliminaire et la planification définitive de ces deux parties proviennent tout d’abord d’un ordre de réalisation des tâches différents. Ensuite, le temps nécessaire pour créer le MCD fut moindre, grâce à l’expérience engrangée pendant le pré-TPI. De plus, aucun diagramme de flux n’a été réalisé, car cela était inutile ici. Enfin, la rédaction des scénarios a pris plus de temps que prévu. Au final, le temps nécessaire à réaliser l’analyse/conception dans son entier était bien évalué, la répartition était juste différente.

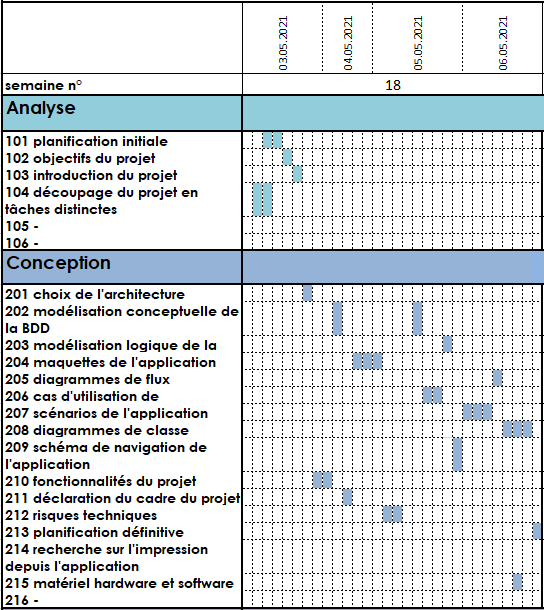


Figure 11 : planification définitive partie 1

Concernant la planification définitive de la partie « implémentation » du projet, très peu de modifications ont été effectuées sur les durées et délais. La seule modification majeure se trouve au niveau des tâches à réaliser.

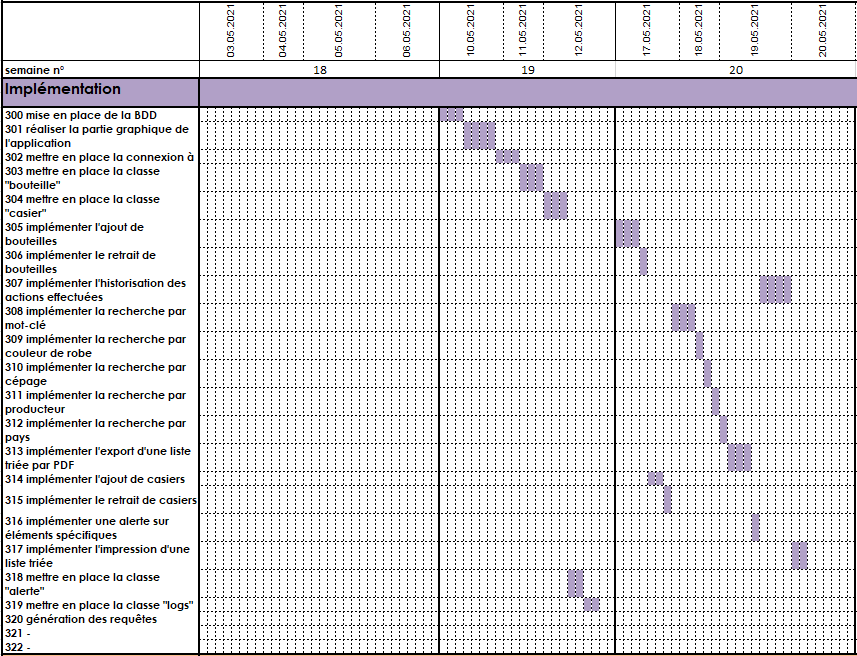


Figure 12 : planification définitive partie 2

En effet, les tâches concernant la mise en place des alertes et de l’historique avaient été oubliées lors de la planification initiale.

Celles-ci ont été ajoutées à la fin de la première semaine de réalisation, lors d’heures auparavant laissé libres pour la réalisation de tests et la rédaction de la documentation.

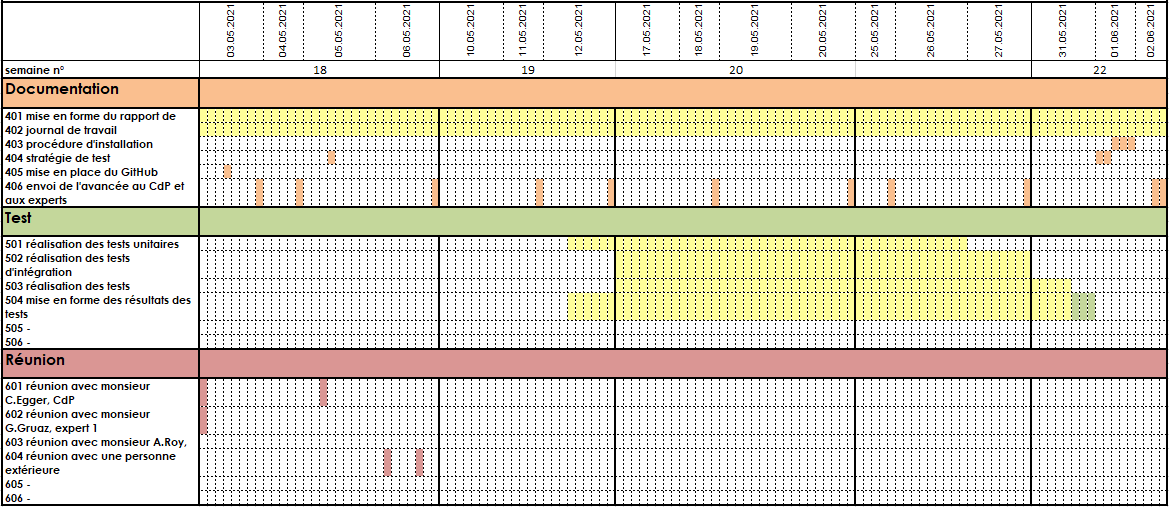


Figure 13 : planification définitive partie 3

Dans cette dernière partie de la planification définitive, il est possible de voir apparaître les réunions ponctuelles avec différentes personnes. Ces réunions/discussions ne sont pas planifiées à l’avance, car il s’agit de rencontres en fonction des besoins du projet. Il est donc impossible de les prévoir à l’avance.

# Réalisation

## Dossier de réalisation

### Répertoires et fichiers du projet

#### Répartition physique des fichiers

Le projet C# consiste en une solution globale, « wineManager ». Celle-ci comporte 3 parties, appelés projets.

* La première, « wineManager\_Library », contient l’intégralité des classes d’objets ainsi que les méthodes utilisées. Il s’agit du côté « données » de l’application.
* La seconde, « wineManager\_View », regroupe l’intégralité des WindowsForms. C’est cette partie qui contient tout ce qui a trait au côté visuel de l’application.
* La troisième et dernière partie est « wineManager\_Tests ». Il s’agit du projet qui contient tous les tests unitaires effectués lors du projet, afin de s’assurer du bon fonctionnement des différentes méthodes.

#### Fichiers et description

Partie 1 : wineManager\_Library

* Alerts.cs : il s’agit d’une classe qui gère toutes les méthodes propres à l’affichage, l’ajout et la suppression d’alertes.
* Bottles.cs : il s’agit d’une classe qui gère toutes les méthodes permettant l’affichage, l’ajout et le retrait de bouteilles. L’affichage inclut les méthodes de tri des bouteilles selon différents critères.
* DBConnection.cs : il s’agit du fichier qui met en place la connexion à la base de données de l’application. Sans lui, il n’est pas possible d’établir la connexion.
* DBRequest.cs : il s’agit du fichier qui contient l’intégralité des requêtes SQL du projet. Toutes les classes qui ont besoin d’interagir avec la base de données passent par ce fichier pour le faire.
* Logs.cs : il s’agit de la classe qui gère l’historique des actions effectuées sur les bouteilles. Cette classe gère les méthodes d’affichage et d’ajout de logs. Il n’y a aucun moyen pour l’utilisateur d’appeler directement ces méthodes, elles sont appelées automatiquement sur des actions bien précises de l’utilisateur.
* StorageBoxes.cs : il s’agit de la classe qui gère les différents espaces de stockage de bouteilles. Les méthodes présentes gèrent l’affichage, l’ajout et la suppression de casiers.

Partie 2 : wineManager\_View

Chaque fichier cité ci-dessous contient en réalité deux fichiers. Le premier, « .Designer.cs », comprend la partie graphique du fichier sous forme de code. Le second, « .cs », comprend les méthodes liées à cette partie graphique.

* AlertsManagement.cs : il s’agit du formulaire qui permet l’affichage des alertes. Il contient un récapitulatif des alertes présentes, ainsi qu’un espace de formulaire pour l’ajout ou la suppression d’alerte en haut de la page.
* BottlesManagement.cs : il s’agit du formulaire qui permet l’affichage des bouteilles. Il contient un récapitulatif des bouteilles présentes, ainsi qu’un espace de formulaire pour l’ajout ou le retrait de bouteilles en haut de la page.
* Logs.cs : il s’agit du formulaire qui permet l’affichage de l’historique des actions. Il contient uniquement un tableau qui affiche les différentes actions effectuées.
* MainPage.cs : il s’agit de la page d’accueil de l’application. Un menu en haut à gauche donne accès aux différentes pages.
* Research.cs : : il s’agit du formulaire qui permet d’effectuer des recherches dans les bouteilles. Les bouteilles présentes dans la base de données sont affichées et différentes options de tri sont disponibles. Il s’agit également de la page par laquelle l’impression et l’export au format PDF sont possibles.
* StorageBoxesManagement.cs : il s’agit du formulaire qui permet l’affichage des casiers. Il contient un récapitulatif des casiers déjà présents, ainsi qu’un espace de formulaire pour l’ajout ou la suppression de casiers en haut de la page.

Partie 3 : wineManager\_Tests

Ce projet contient l’intégralité des classes de tests. Celles-ci permettent d’exécuter automatiquement les tests unitaires liées aux différentes méthodes du projet. Chaque fichier « .cs » du projet Test reprend la classe associée du projet Library.

### Produit fini

Ci-dessous, vous pourrez trouver les spécifications techniques de l’application. Afin de pouvoir installer le système, toutes les informations nécessaires sont disponibles sur l’annexe A9 (manuel d’installation). La liste des erreurs encore présente dans l’application est présente au point [Erreurs restantes](#_Erreurs_restantes_(TODO)), plus loin dans ce même document.

|  |  |
| --- | --- |
| **Nom de l’application** | wineManager |
| **Version du produit** | 1.0 |
| **Système d’exploitation nécessaire** | Windows10 |
| **Logiciels annexes nécessaires** | MySQL Workbench |

## Liste des éléments fournis

Lors du rendu du projet le 02 juin 2021, plusieurs éléments seront fournis au client. Ci-dessous se trouve un tableau récapitulatif des fichiers fournis, de leurs formats ainsi que leurs emplacements :

|  |  |  |
| --- | --- | --- |
| **Fichier** | **Format** | **Emplacement** |
| Rapport de projet | PDF | Dossier « Release » sur GitHub |
| Code source | .zip | Dossier « Release » sur GitHub |
| Scripts de base de données | .sql | Dans le .zip du code source |

### Scripts

Pour pouvoir travailler avec des données stockées dans une base de données, il est nécessaire de construire cette base et d’y insérer des données pertinentes. Ci-dessous, il est possible de trouver les informations sur la méthodologie et la structure des scripts liés à la base de données.

Génération de la base de données

Afin de générer la base de données, un script .sql existe. La génération des tables se fait systématiquement sur le même modèle, illustré ci-contre.

Tout d’abord, une vérification a lieu, afin de savoir si la table existe déjà. Si ce n’est pas le cas, celle-ci est créée.

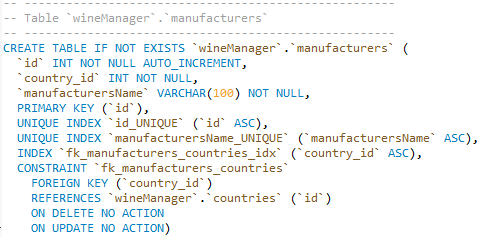


Figure 14 : BDD -> génération de table

Ensuite, les différentes colonnes, avec leurs paramètres respectifs, sont ajoutées à la table. Une fois cet ajout fait, l’indexation est réalisée.

Enfin, les contraintes sont appliquées aux différentes colonnes, afin que la base de données sache comment réagir en fonction des interactions.

Insertion des données dans la base

Afin de réaliser l’insertion des données sans heurts, il est nécessaire que celles-ci soient ajoutées selon un ordre défini. En effet, certaines tables nécessitent des données d’autres tables. Ces tables dites de référence doivent donc être remplies en premier.

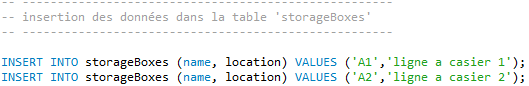


Figure 15 : BDD -> insertion de données

Sur l’illustration à droite, il est possible de voir la syntaxe utilisée pour réaliser les insertions de données :

Le « insert into … » représente la table dans laquelle les données doivent être ajoutées. La parenthèse qui suit indique les noms des différentes colonnes où les données doivent être insérées, séparés par des virgules. La parenthèse derrière « values » contient les données à insérer dans la table. Chaque chaîne de caractère doit être contenu dans des guillemets, sinon une erreur de syntaxe aura lieu et l’insertion ne se fera pas.

La génération des données a pris un temps assez conséquent au final. Elles ont été générées de la manière suivante, table par table :

1. Génération des données sur Excel, sous forme de tableau à plusieurs colonnes
2. Ecriture des morceaux récurrents des requêtes dans des cases au-dessus du tableau
3. Construction des requêtes complètes en utilisant les numéros de cases et les morceaux de requêtes à l’aide de la fonction CONCATENER()
4. Copie de la colonne contenant les requêtes complètes dans un fichier .sql
5. Exécution du .sql pour vérifier que toutes les requêtes fonctionnent
6. Ajout du contenu au fichier insertData.sql global

## Description des tests effectués

### Tests unitaires

Les tests unitaires sont des tests automatiques qui sont réalisables dans Visual Studio 2019 (projet wineManager\_Tests de la solution wineManager). Afin de les lancer, il est nécessaire de changer la chaîne de connexion à la base de données (fichier DBConnection.cs).

Pour ce faire, il faut commenter la ligne habituelle et enlever le commentaire de la ligne contenant la base de test. Il est également nécessaire de régénérer la base de données à chaque lancement des tests automatiques, sinon certains tests ne se valideront pas.

En regardant le tableau des tests unitaires à la [page suivante](#_État_des_tests), il est possible de voir que la majorité des tests unitaires échouent. Il ne s’agit pas d’un problème de développement des méthodes de l’application, comme l’atteste les tests fonctionnels réussis, mais d’un souci au niveau de l’implémentation des méthodes de tests.

La correction de ces tests pour les rendre fonctionnels est réalisable, mais cela nécessiterait un investissement en temps trop conséquent pour pouvoir l’inclure dans les délais restants.

### Tests fonctionnels

Les tests fonctionnels sont des tests exécutés manuellement qui permettent de vérifier le bon fonctionnement de l’application. Ils sont réalisés sur la base des [Scénarios](#_Scénarios). Si tout fonctionne comme annoncé dans le scénario, le test est considéré comme validé. Si ce n’est pas le cas, le test est considéré comme échoué.

Test 1 : Insertion de données dans la base de données

*Méthodologie de test :* vérifier qu’aucune erreur ne se produit lors de l’exécution du script d’insertion de données

*Contrôle :* la fenêtre de logs, présente dans MySQL Workbench, permet de s’assurer que toutes les requêtes ont été exécutées correctement.

Test 2 : Génération de la base de données

*Méthodologie de test :* vérifier qu’aucune erreur ne se produit lors de l’exécution du script d’insertion de données

*Contrôle :* la fenêtre de logs, présente dans MySQL Workbench, permet de s’assurer que toutes les requêtes ont été exécutées correctement.

Scénario 4a : Recherche par mot-clé

La recherche par mot-clé ne fonctionne pour le moment pas. Il reste une erreur au niveau de la requête SQL, qui rend l’utilisation de la recherche impossible et fait planter le programme.

### État des tests

L’état des tests est un tableau qui reprend l’ensemble des tests qui ont été effectués, ainsi que leur état (validé / non validé). Cela permet de voir facilement qu’est-ce qui est fonctionnel et qu’est-ce qui ne l’est pas. Le détail des démarches de tests se trouvent dans [Scénarios](#_Scénarios) et [Tests](#_Tests_unitaires).

Tests unitaires :

|  |  |
| --- | --- |
| Nom du test | État |
| AddAlerts\_AllParams\_OK | OK |
| DelAlerts\_AllParams\_OK | OK |
| ShowAllAlerts\_NoSorting\_OK | NO OK |
| AddAlertToBottle\_Allparams\_OK | OK |
| AddBottleWithDescAndVarietal\_OK | NO OK |
| GetBottlesWithAlert\_NoSorting\_OK | NO OK |
| GetBottlesWithName\_NoSorting\_OK | NO OK |
| OrderByColor\_alphabetical\_OK | NO OK |
| OrderByCountry\_alphabetical\_OK | NO OK |
| OrderByManufacturer\_alphabetial\_OK | NO OK |
| RemoveBottle\_Allparams\_OK | OK |
| ResearchByKeyWord\_existingword\_OK | NO OK |
| ShowAllBottles\_NoSorting\_OK | NO OK |
| UpdateBottle\_Allparams\_OK | OK |
| AddLog\_AllParams\_OK | NO OK |
| GetAllLogs\_NoSorting\_OK | NO OK |
| AddStorageWDesc\_AllParams\_OK | OK |
| RemoveStorage\_\_OK | OK |

Tests fonctionnels :

|  |  |
| --- | --- |
| Nom du test | État |
| Scénario 1a : ajout de bouteilles | OK |
| Scénario 1b : retrait de bouteilles | OK |
| Scénario 2a : ajout de casiers | OK |
| Scénario 2b : suppression de casiers | OK |
| Scénario 3a : ajout d’alerte | OK |
| Scénario 3b : suppression d’alerte | OK |
| Scénario 4a : recherche par mot-clé | NO OK |
| Scénario 4b : filtre par critère | OK |
| Scénario 5 : afficher l’historique | OK |
| Scénario 6 : imprimer | OK |
| Scénario 7 : exporter en PDF | OK |
| Test 1 : insertion de données par script dans la BDD | OK |
| Test 2 : génération de la BDD | OK |

## Problèmes rencontrés et résolution

[10.05.2021] Connexion à la base de données

Suite à l’expérience retirée du projet de pré-TPI, pour diminuer le nombres d’ouverture et fermeture à la base de données, j’ai décidé d’ouvrir la connexion une fois au lancement de l’application et de fermer cette connexion à la fermeture de l’application.

Le souci qui s’est présenté est le suivant : la vue principale (MainPage.cs) ne reconnaît pas la classe DBConnection, comme s’il manquait un référencement quelque part.

Afin de résoudre ce problème, je vais demander de l’aide pour obtenir une piste de résolution, afin de ne pas perdre trois heures à régler un petit problème, qui a des conséquences importantes sur le projet. L’aide sera demandée à F.Andolfatto.

L’aide apportée par F.Andolfatto le 11.05.21 a permis de corriger le problème : la référence manquante devait se faire dans le projet *WineManager\_View*. Un clic droit sur Références -> Ajouter une référence (voir image ci-contre), avec une coche pour le projet *WineManager\_Library* est l’élément qui manquait pour faire fonctionner le tout.

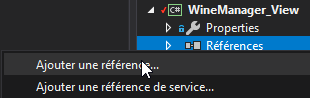


Figure 16 : problème rencontré -> référence de projet

[11.05.21] Récupération d’une liste dans une liste par requête SQL (classe « Bottles »)

Pour pouvoir afficher les différents cépages d’un vin, il m’est nécessaire de récupérer une liste de cépages, dans la liste représentant un vin spécifique.

Hors, il n’est pas possible de récupérer une liste de données d’une table intermédiaire si d’autres paramètres sont également nécessaires.

Ainsi, pour pouvoir insérer une liste dans la liste, il est nécessaire d’appeler une autre requête qui crée une liste. Celle-ci sera ensuite parcourue, pour intégrer élément par élément à la liste globale.

Cette méthode a été suggérée par X.Carrel, qui a pu m’expliquer le manquement dans ma logique de résolution et apporter des éléments théoriques complémentaires.

[12.05.2021] Bouton retour à l’accueil non fonctionnel

Le bouton permettant de retourner de la page sélectionnée à la page d’accueil ne fonctionne pour le moment pas comme il se doit.

La fenêtre active se ferme bien, mais la page d’accueil reste grisée et inactive, impossible d’interagir avec.

Plusieurs éléments ont été testés, des recherches supplémentaires sont encore nécessaires pour régler le souci.

Il s’agit d’un élément complémentaire pour fluidifier l’utilisation de l’application, ce n’est donc pas prioritaire pour le moment.

**EDIT 25.05.2021 :** une solution a été trouvée finalement. L’utilisation de la méthode « ShowDialog() » plutôt que « Show() » sur les formulaires permet d’obliger l’utilisateur à fermer le formulaire avant de pouvoir à nouveau interagir avec la page précédente.

[12.05.2021] récupération de la liste dans la liste (cf problème du 11 mai)

La requête SQL permettant de récupérer les données est désormais fonctionnelle. Cependant, seul le premier élément de la liste est récupéré, les suivants ne le sont pas. Du travail est encore nécessaire pour afficher la totalité des données.

**EDIT 17.05.2021 :** en recherchant la manière de mettre en place des listes déroulantes, la syntaxe nécessaire est apparue sur une des pages (voir source : récupération de données pour une liste déroulante depuis MySQL). En mettant en place les requêtes d’ajout de bouteilles à la BDD, les modifications ont été faites et cela a corrigé le problème rencontré.

[26.05.2021] pop-ups de confirmation

Lors de la conception, les scénarios intégraient des pop-ups de confirmation avant de valider la réalisation d’une action.

Au cours de la réalisation, j’ai pu m’apercevoir que mettre en place ces pop-up étaient tout sauf simples. De plus, cela avait pour conséquence de surcharger la page.

Le fait de valider les modifications de données par une MessageBox a donc été supprimé et les scénarios ont été modifiés en conséquence.

## Erreurs restantes

Recherche par mot-clé :

La fonctionnalité « Rechercher par mot-clé » ne fonctionne pas. Lorsque le bouton « Rechercher » est cliqué sur l’application, le programme plante et affiche un message d’erreur.

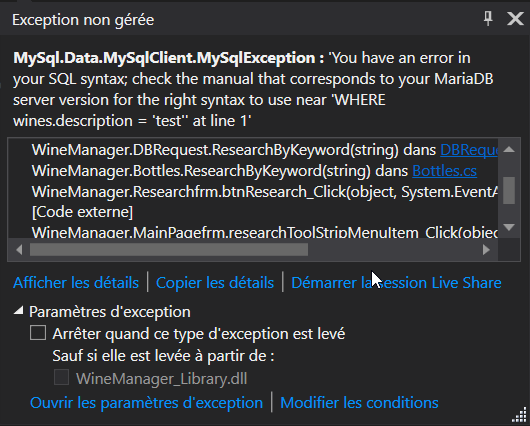


Figure 17 : erreur -> recherche par mot-clé

L’erreur restante à ce niveau-là est une erreur de syntaxe dans la requête SQL. Cette erreur se trouve dans la méthode « ResearchByKeyword ».

L’impact de cette erreur est minime. Seule la fonctionnalité de recherche par mot-clé est inutilisable, tout le reste est fonctionnel et n’est pas impacté.

Afin de corriger cette erreur, il s’agit uniquement de consacrer du temps à trouver l’erreur de syntaxe SQL dans le code.

Tests unitaires :

Lors de l’exécution des tests unitaires, seuls 7 des 19 tests créés sont validés (36.8%). Cela provient d’une erreur de développement des tests et non pas de l’application, car les tests fonctionnels sont validés (voir [État des tests](#_État_des_tests) pour plus de détails).

La raison pour laquelle les tests échouent semblent être une génération incorrecte de ma part des listes de contrôle de données. Les tests contiennent en effet un résultat fixe (resExpected) qui est comparée aux valeurs obtenus par l’application en exécutant la méthode à tester (resCalculated).

Dans le cas où le résultat fixe est mal généré, le test ne peut pas être validé car sa base de comparaison est incorrecte.

Techniquement parlant, les compétences nécessaires pour corriger ces erreurs dans les tests sont présentes. L’élément manquant dans ce projet est le temps.

Du côté fonctionnel, le fait que les tests échouent n’impactent en rien le fonctionnement de l’application. Tout fonctionne sans être dépendant de la réussite des tests.

## Comparaison des délais entre la planification et la réalisation (TODO)

Chaque semaine – entre planif déf et réalité – avec image (gant) – retards ? avance ? -> raison

[05.05.2021] Retard de train

-> début du travail 25 minutes plus tard

[12.05.2021] Retard de train

-> début du travail 30 minutes plus tard

[17.05.2021] mise en place de l’ajout de bouteilles

-> beaucoup plus long que prévu : entre les tests de types, les requêtes et tout, cela a pris beaucoup plus de temps que prévu

[17.05.2021] implémentation des recherches

-> le temps envisagé pour la mise en place des différentes recherches est beaucoup plus élevé que le temps réel : la solution finale est beaucoup plus courte à mettre en place => équilibre avec l’ajout de bouteilles

[25.05.2021] Retard de train

-> début du travail 30 minutes plus tard

[26.05.2021] Débug des tests unitaires

3h de temps pour essayer de débugger les tests unitaires -> abandonné, car l’application est fonctionnelle derrière

[31.05.2021] retard de train

->début à 9h50, prochaine période

# Conclusions

## Atteinte des objectifs

Comme spécifié au début du projet, dans [Objectifs](#_Objectifs) et [Fonctionnalités](#_Fonctionnalités), de nombreux objectifs ont été énoncés. Tous n’ont pas pu être complétés, mais la majorité ont été réalisés. Le tableau des tests fonctionnels au point [État des tests](#_État_des_tests) donne un très bon aperçu de l’avancée de l’application.

Ainsi, dans les fonctionnalités annoncées au début du projet, seule la recherche par mot-clé ne fonctionne pas du tout. Le reste des options existent et sont utilisables. Concernant les objectifs plus larges, l’application me semble instinctive d’utilisation et donc tout à fait utilisable par un novice en informatique.

De plus, le manuel d’installation permet à n’importe qui de mettre en place l’application, à condition d’avoir l’archive .zip associée.

Globalement, l’application est fonctionnelle et la majorité des objectifs annoncés ont été atteints.

## Maintien des délais (TODO)

Comparaison de gant (définitif et final) – délai maintenu ? pourquoi ? - raison des retards / avances sur tâches individuelles

Ajout de bouteilles -> perte de temps : beaucoup de vérification de données à mettre en place, première méthode : temps important nécessaire

Recherche /tri -> temps très courts par rapport à l’estimation

Génération des pdfs -> débug à cause d’une erreur d’index : perte de temps à ce niveau-là

Mise en place des alertes -> relativement compliqué : perte de temps associée

## Points positifs et négatifs (TODO)

Bien passé ? – mal passé ? – raison – impact – connaissances acquises (comment, où) – réflexion critique sur approche + résultats obtenus – comparaison des variantes de solutions ou explications sur pourquoi il n’y en a pas – bilan personnel

++ beaucoup appris (débug, gestion de fenêtres, …)

++ envie de continuer à bosser dessus

-- sentiment de travail inachevé -> améliorations nécessaires

-- frustration de ne pas savoir résoudre les erreurs

-- code mal géré -> fonctionnel, mais retravaillable pour être plus léger et moins brouillon (passer des éléments en paramètres pour diminuer le nombre de méthodes …)

GLOBAL : bien passé, arrivé au bout, satisfaite, mais déçue d’un certain côté de la façon dont c’est codé. Une certitude : je ne suis pas faite pour développer 100% de ma journée et ça, je le sais désormais. Par contre, rédiger de la doc, c’est pas la même :3

## Difficultés particulières

Comme tout projet, il arrive que des difficultés se présentent. Lors de celui-ci, certains points spécifiques ont été problématiques. Le point qui a posé le plus de problèmes a été l’affichage de listes dans une liste, récupérées depuis une base de données (voir [Problèmes rencontrés et résolution](#_Problèmes_rencontrés_et)). Comme cet affichage est nécessaire sur la majorité des fenêtres de l’application, cela a bloqué le projet pendant un bon moment.

Un autre élément qui a embêté lors du projet a été la gestion des fenêtres de l’application, afin de pouvoir griser l’ancienne quand une nouvelle fenêtre est ouverte. La solution a finalement été trouvée au moment de devoir ajouter des pop-up de confirmation.

De manière générale, le débug est un outil que je connais mal. Ce manque de connaissance a causé quelques retards, car des bugs rapides à résoudre ont finalement demandé du temps à cause du manque de maîtrise.

Malgré cela, les délais, même s’ils ont été dépassés sur certaines tâches de manière significative, ont été presque tenus. D’autres délais étaient plus longs que nécessaire, ce qui crée au final un équilibre permettant de rendre le projet terminé dans les délais.

## Évolutions et améliorations

Au vu de l’avancée actuelle du projet, il reste peu à faire pour qu’il soit entièrement fonctionnel. Les premières améliorations à faire resteraient quand même les quelques erreurs présentes dans [Erreurs restantes](#_Erreurs_restantes), afin de conclure définitivement le but premier du projet.

Une fois cela fait, de nombreux changements peuvent être effectués, dans le but d’améliorer l’application. En voici quelques-unes ci-dessous, la liste est cependant loin d’être exhaustive, de nombreuses autres possibilités existent :

* Ajouter à l’historique les actions effectuées concernant les casiers et les alertes
* Permettre l’export en PDF ou l’impression de la page d’historique
* Gérer l’ajout d’un certain nombre d’un même exemplaire de bouteilles à plusieurs casiers différents
* Autoriser la suppression d’alertes et de casiers en les sélectionnant dans le tableau récapitulatif puis en cliquant sur un bouton
* Ajouter un fichier de configuration pour permettre la modification facile de certains paramètres (nom de la base de données, nom de l’imprimante à utiliser, nom d’utilisateur et mot de passe pour la base de données …)
* Optimiser le code
* Optimiser l’affichage des listes de données dans les tableaux
* Ajouter un logo à l’application (page d’accueil, symbole de la barre de la fenêtre, logo de l’application …)
* Ajouter des rôles administrateurs et utilisateurs, ainsi qu’une authentification

Concernant ce qui est déjà en place, peu de modifications seraient à apporter selon moi. Un des éléments qui mériteraient d’être modifié, surtout au niveau du code, est l’affichage des listes sous une forme de chaîne de caractère. Cette solution a été utilisée pour simplifier l’envoi de données d’une méthode à une autre, mais n’est de loin pas la plus élégante.

# Annexes - sommaire

* A1 : résumé du rapport de TPI
* A2 : Glossaire
* A3 : Sources et bibliographie
* A4 : Protocoles de discussion
* A5 : Planification initiale
* A6 : Planification définitive
* A7 : Planification finale
* A8 : Journal de travail
* A9 : Manuel d’installation