Projet génie logiciel

UNIVERSITE DE BOURGOGNE

UFR DES SCIENCES ET TECHNIQUES

MASTER1 Informatique

Année universitaire 2023-2024

Seydina DIA

Mélissandre GALLOIS

Hugo GUYOT

Nathan JACQUINET

Julie LAVAUZELLE

Constant OUEDRAOGO

TABLE DES MATIERES

TABLE DES ILLUSTRATIONS

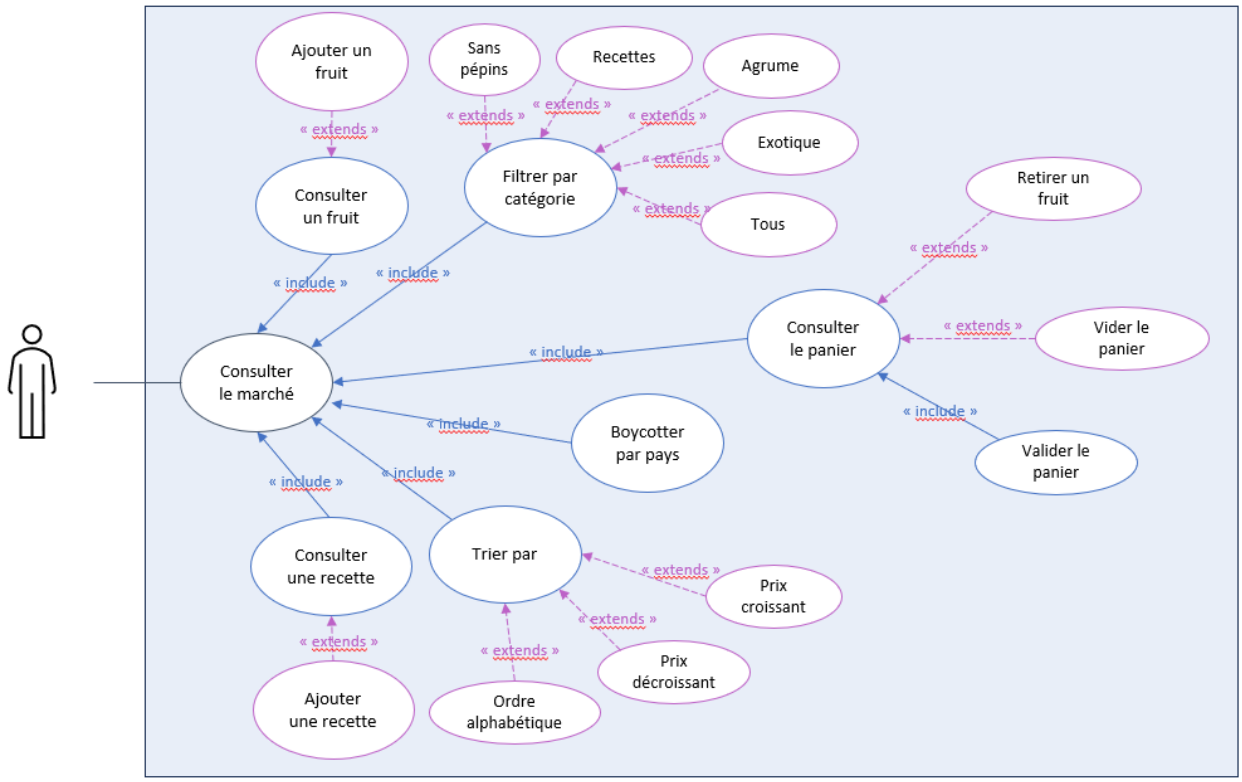
I. INTRODUCTION

Lien Github : https://github.com/Hephaist19/TPGL

Nous sommes six étudiants en première année de Master Informatique à l'UFR Sciences et Techniques de Dijon. Pour notre module de Génie Logiciel, nous devons réaliser une application de panier de fruits. Afin de pouvoir mettre en place cette application, nous sommes passés par différentes étapes d'élaboration des méthodes AGILE qui ont mené à notre produit final.

Le sujet consiste en une application de marché dans lequel on peut acheter des fruits, avec la possibilité de consulter des recettes afin de s'aider dans les achats des fruits. Notre application offre également la possibilité de trier les fruits pour plus de facilité lors de la sélection. De plus, nous pouvons consulter le panier avec les fruits ajouter, et en supprimer directement depuis celui-ci.

II. ANALYSE DES BESOINS



III. SPECIFICATIONS ET TESTS

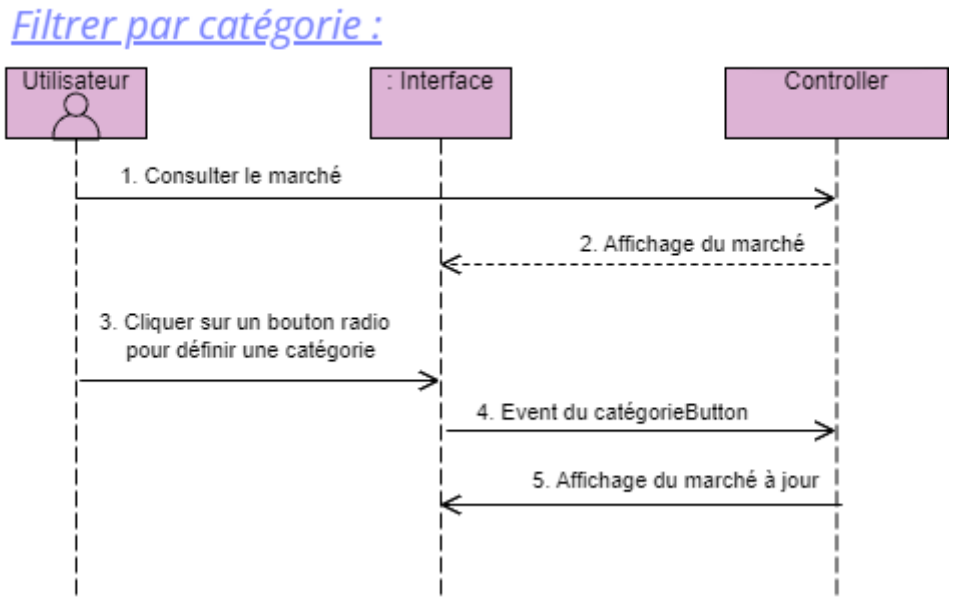


Image 1 : Diagramme de séquence Filtrer par catégorie

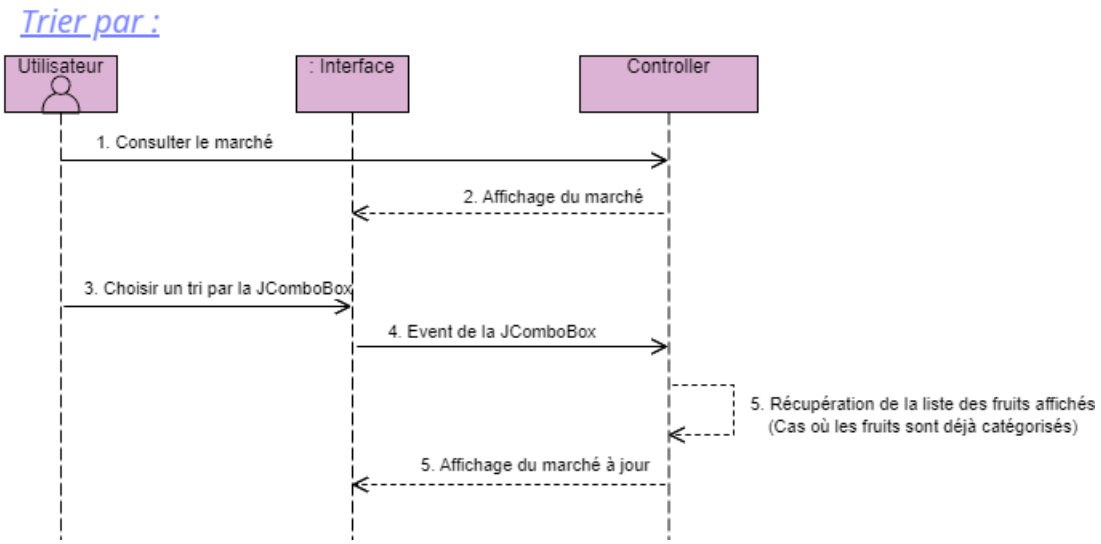


Image 2 : Diagramme de séquence Trier par

Boycott :

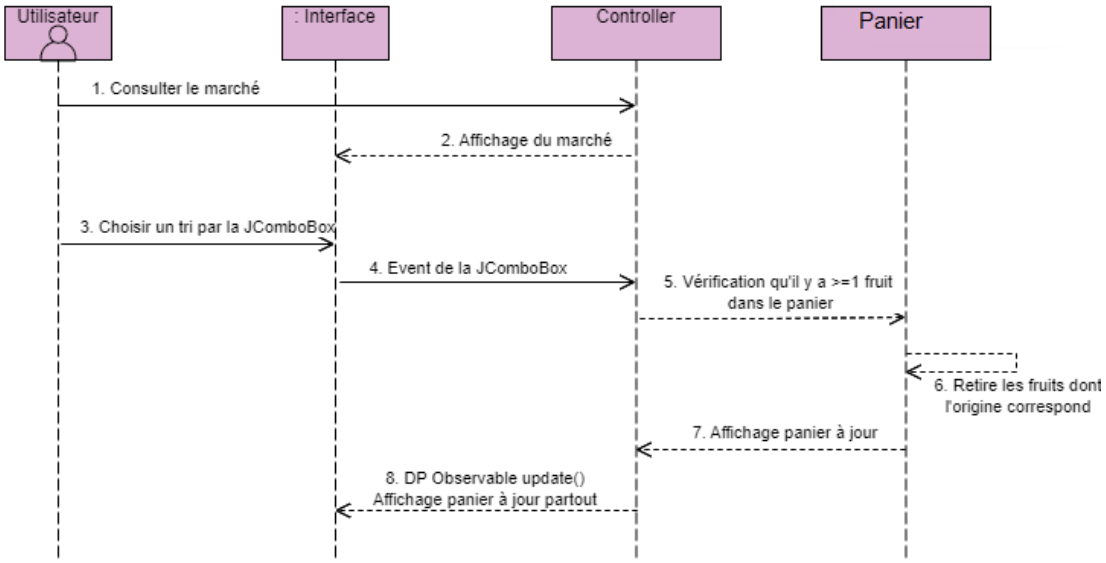


Image 3 : Diagramme de séquence Boycott

Ajouter fruit :

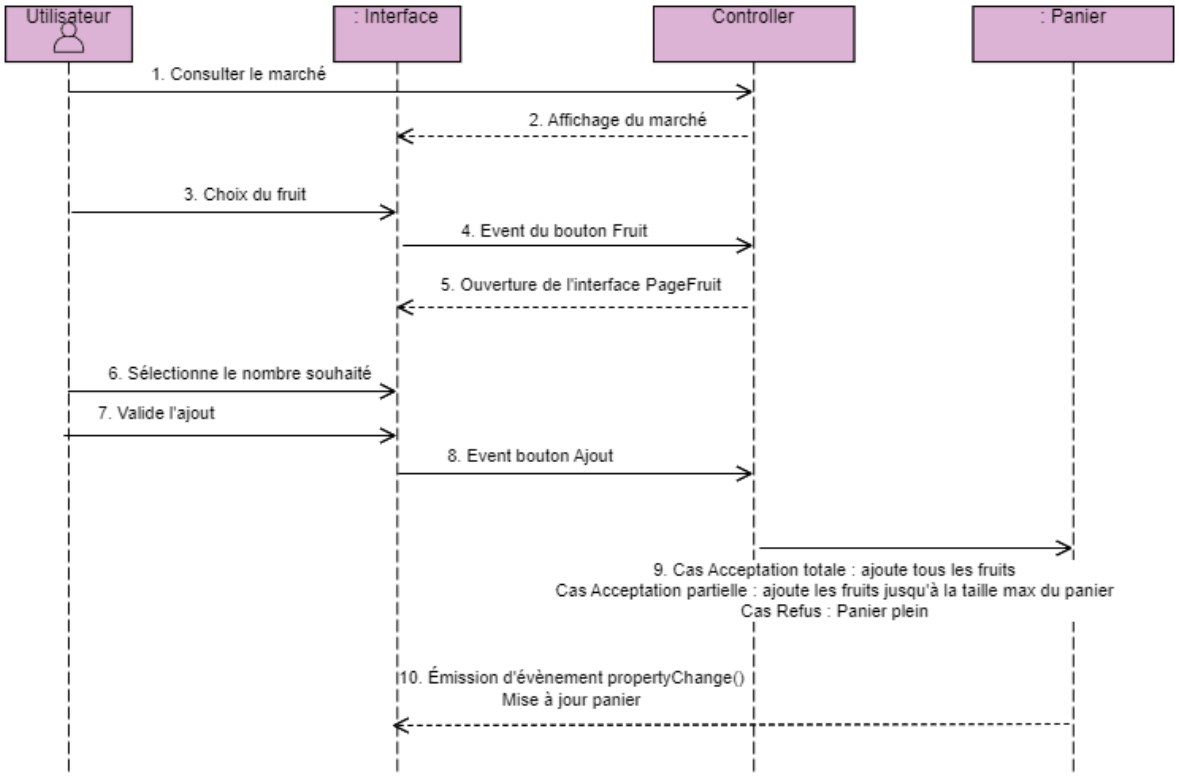


Image 4 : Diagramme de séquence Ajout d'un fruit

Vider le panier :

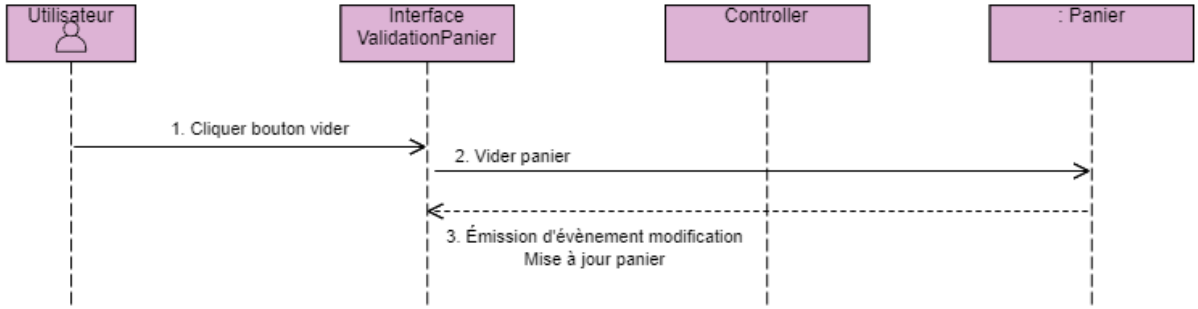


Image 5 : Diagramme de séquence Vider le panier

Retirer un fruit :

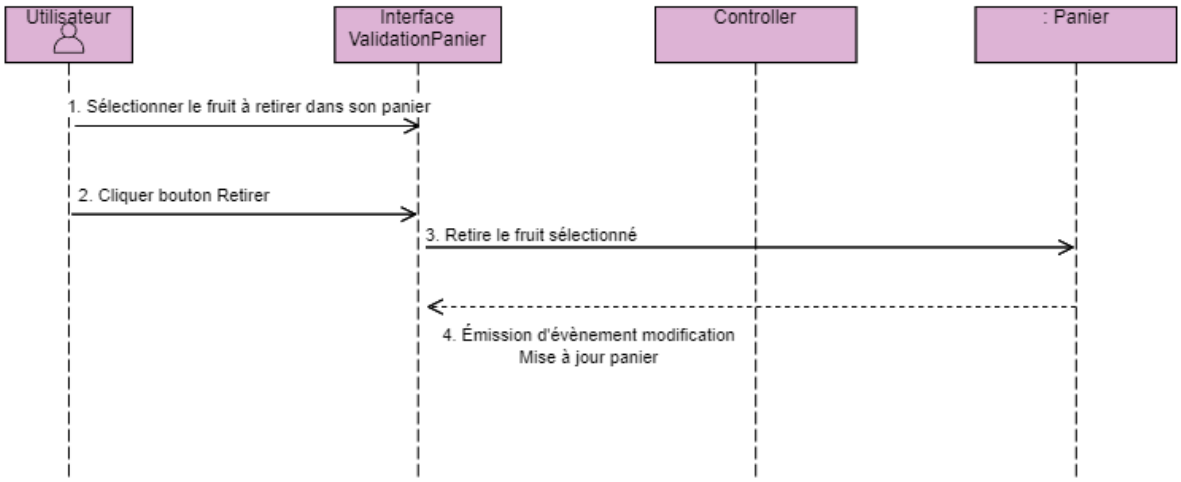


Image 6 : Diagramme de séquence Retirer un fruit

Valider le panier :

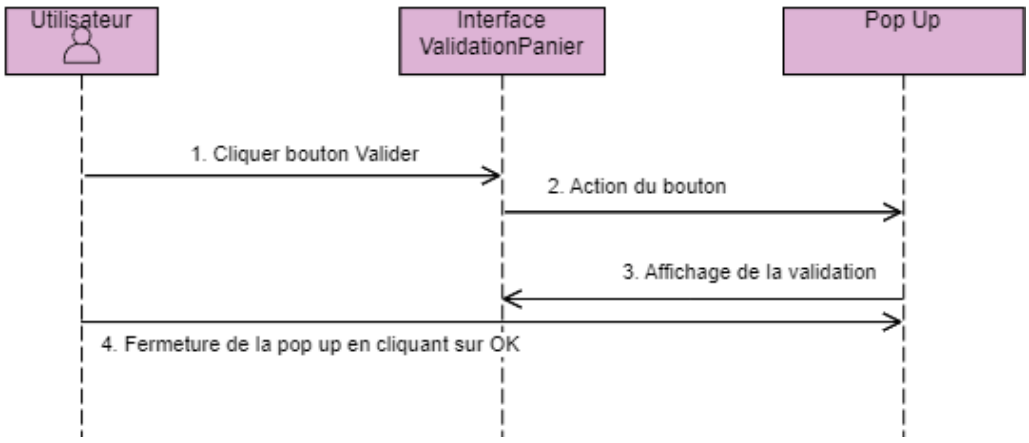


Image 7 : Diagramme de séquence Valider le panier

Pour les tests des modèles, nous avons définis des tests unitaires classique.

Pour tester que les classes implémentant les interfaces comme Fruit, nous avons fait une classe abstraite de test, avec des tests unitaires testant de manière opaque l'instance de Fruit en question.

Ainsi chaque classe implémentant l'interface Fruit, dérive de cette classe abstraite de teste.

Et elle devra alors instancier le fruit en question.

Cela nous permet donc de ne pas refaire pour chaque classe de Fruit différentes, des tests identiques.

Par exemple, un Fruit doit absolument renvoyer un prix positif, donc dans la classe abstraite de teste, nous testons cette obligation, indépendamment de l'instance de fruit testé.

Les tests de Panier et Recettes suivent un chemin de test unitaire plus simplifiés.

Pour les tests des usines, nous nous assurons que pour chaque valeur possible de l'énumération, l'usine renvoie une instance valide (non null)

Pour le constructeur de liste de fruit (FruitFilter), nous avons fait le choix d'utiliser un jeu de test fixe, et de contrôler le résultat attendu avec des listes types.

Pour les tests d'interface, l'utilisation de mocks était impossible, (par exemple les JDialog avaient absolument besoin d'une instance de fenêtre parent valide). Nous instancions les interfaces, mais ne les affichons pas.

Cependant, lors des tests, des JDialog d'erreur/informations propre à l'applications s'ouvrent, il faut alors les fermer. Elles reflètent le comportement attendu.

Nous avons testé les comportements des boutons, avec des jeux de test, et comparé à des jeux types, en simulant un clic sur les boutons à l'aide de la methode doClick().

IV. CONCEPTION

Nous avons basé notre architecture sur un Model View Controller.

Les models regroupe des classes simples :

Les fruits, les recettes et les paniers. Ils héritent d'interfaces, comme par exemple pour chaque fruit implémente Fruit.

Ainsi, le panier ne sait travailler qu'avec des instances implémentant l'interface Fruit, ce qui permet une opacité totale sur le type d'instance du côté du panier.

De manière totale, les Controllers, les views travaillent uniquement avec de Fruit, et jamais avec des instances spécifiques de Fruit.

Le panier peut émettre des évènements, qui peuvent être capturés par des instances implémentant PropertyChangeListener. Par exemple la JDialog ValidationPanier, ou la fenêtre principale peuvent capturer ces évènements et ainsi réagir le cas échéant.

Ensemble, ils forment un design Observer moderne.

Dans les controllers, nous avons une Usine par modèle :

Une usine pour les fruits, une pour les paniers, et une pour les recettes.

Nous avons aussi un petit Builder (FruitFilter) qui permet de construire une liste de Fruit spécifique respectant une série de filtres et des tris.

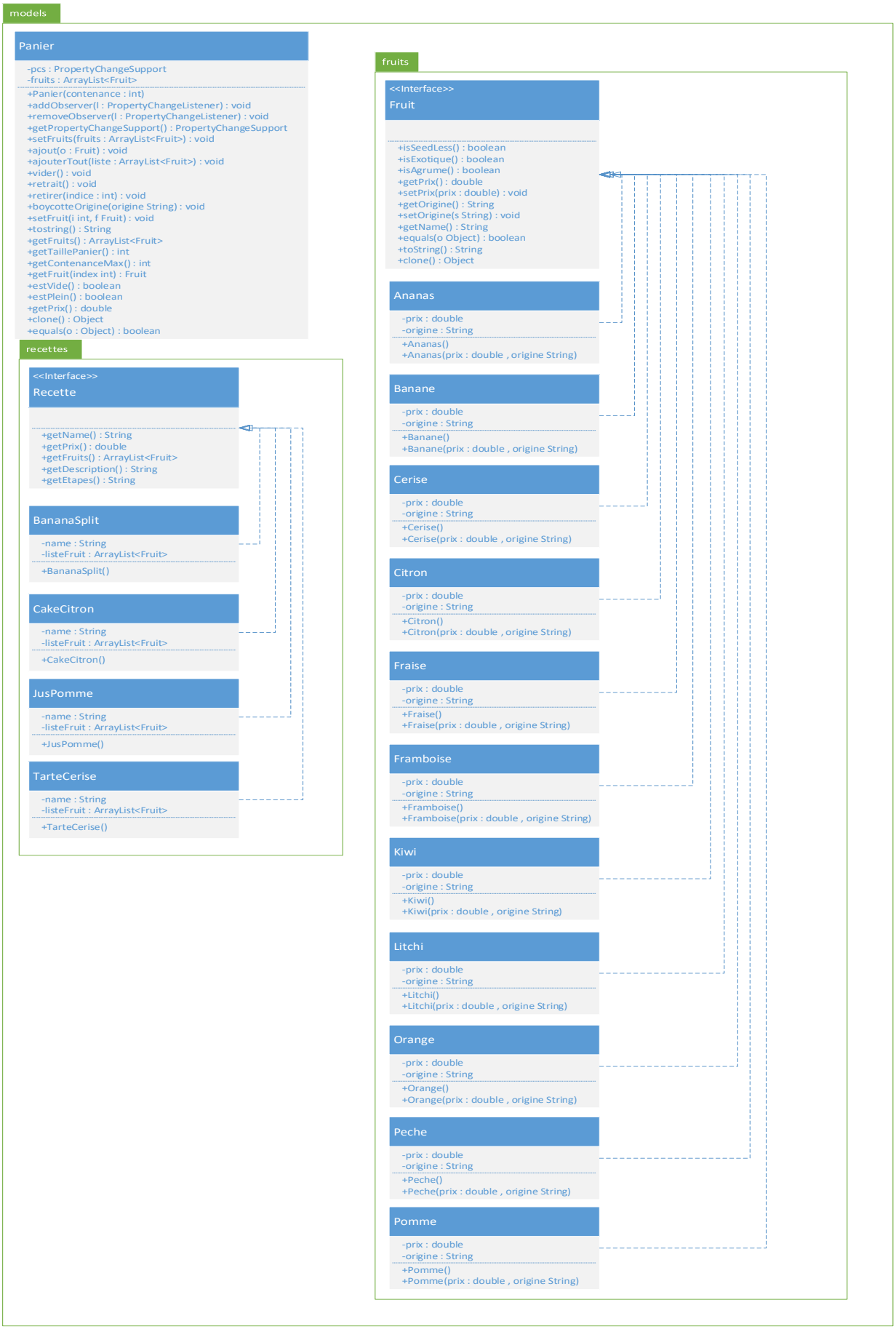
Pour rendre plus lisible et facilement interprétable l'utilisation des Usine, ou des Filtres/Tris, nous avons décidé d'utiliser des énumérations (dans le package utils)

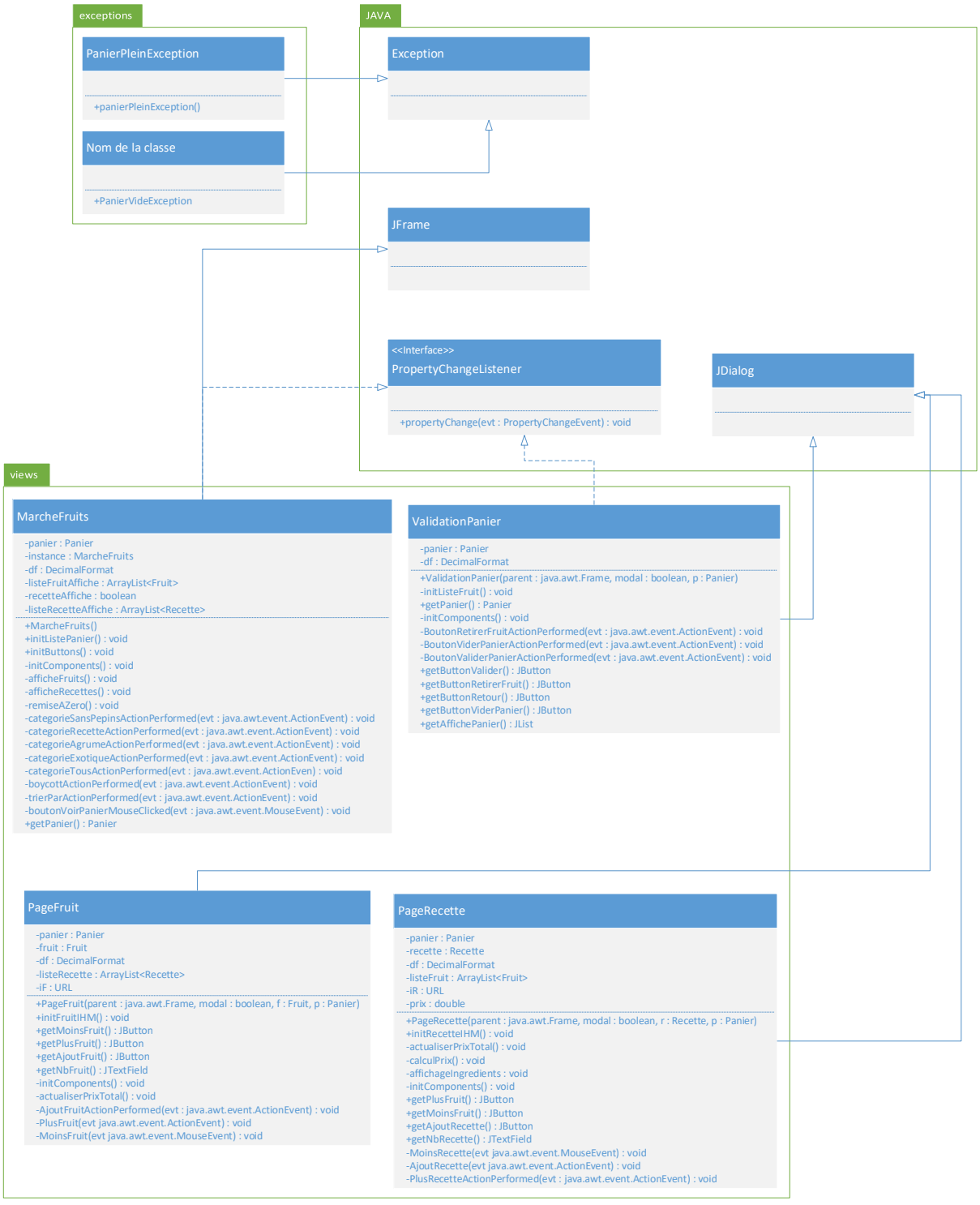
Cela nous permet une évolutivité, tout en adaptant bien évidemment les usines dès lors nous ajouterons des fruits/filtre/tris etc…

Les view possèdent un comportement dynamique, primordiale pour une évolutivité facile.

Le fenêtre principale, MarcheFruit, peut venir à ouvrir des JDialog pouvant modifier le contenu de l'instance du panier, mais grâce à l'utilisation de listeners, chaque modification est répercutée entre toutes les instances abonnées à ce panier, de manière totalement opaque.


Main
+main(String[] args) : void
controllers
FruitsFilter
+FruitsFilter()
+FruitsFilter(Arraylis<Fruits>)
+getResult() : ArrayList<Fruit>
-deepCopie(toCopie : ArrayList<Fruit>) : Fruit
+sort(type : SortType) : FruitsFilter
+filter(type : FilterType) : FruitsFilter
Nom du package
FruitsFactory
+createFruit(type : FruitType) : Fruit
+createFruit(type : FruitType, prix : double, origine : String) : Fruit
+createAll : ArrayList<Fruit>
+createAllOf(all : FruitType) : ArrayList<Fruit>
PanierFactory
+createPanier(contenance : int) : Panier
+createPanier(type : PanierType) : Panier
RecettesFactory
+createRecette(type : RecetteType) : Recette
+createAll() : ArrayList<Recette>
+createAllOf(all : RecetteType) : ArrayList<Recette>
utils
<<Énumeration>>
FilterType
PEPINS
NPEPINS
EXOTIQUE
NEXOTIQUE
AGRUME
NAGRUME
<<Énumeration>>
PanierType
EXOTIQUE
AGRUME
SANSPEPINS
AVECPEPINS
NULL
<<Énumeration>>
SortType
ALPHABETIQUE
ANTIALPHABETIQUE
PRIXCROISSANT
PRIXDECROISSANT
<<Énumeration>>
FruitType
ANANAS
BANANE
CERISE
CITRON
FRAISE
FRAMBOISE
KIWI
LITCHI
ORANGE
PECHE
POMME
<<Énumeration>>
RecetteType
JUSPOMME
BANANASPLIT
CAKECITRON
TARTECERISE


models
Panier
-pcs : PropertyChangeSupport
-fruits : ArrayList<Fruit>
+Panier(contenance : int)
+addObserver(l : PropertyChangeListener) : void
+removeObserver(l : PropertyChangeListener) : void
+getPropertyChangeSupport() : PropertyChangeSupport
+setFruits(fruits : ArrayList<Fruit>) : void
+ajout(o : Fruit) : void
+ajouterTout(liste : ArrayList<Fruit>) : void
+vider() : void
+retrait() : void
+retirer(indice : int) : void
+boycotteOrigine(origine String) : void
+setFruit(i int, f Fruit) : void
+tostring() : String
+getFruits() : ArrayList<Fruit>
+getTaillePanier() : int
+getContenanceMax() : int
+getFruit(index int) : Fruit
+estVide() : boolean
+estPlein() : boolean
+getPrix() : double
+clone() : Object
+equals(o : Object) : boolean
recettes
<<Interface>>
Recette
+getName() : String
+getPrix() : double
+getFruits() : ArrayList<Fruit>
+getDescription() : String
+getEtapes() : String
BananaSplit
-name : String
-listeFruit : ArrayList<Fruit>
+BananaSplit()
CakeCitron
-name : String
-listeFruit : ArrayList<Fruit>
+CakeCitron()
JusPomme
-name : String
-listeFruit : ArrayList<Fruit>
+JusPomme()
TarteCerise
-name : String
-listeFruit : ArrayList<Fruit>
+TarteCerise()
fruits
<<Interface>>
Fruit
+isSeedLess() : boolean
+isExotique() : boolean
+isAgrume() : boolean
+getPrix() : double
+setPrix(prix : double) : void
+getOrigine() : String
+setOrigine(s String) : void
+getName() : String
+equals(o Object) : boolean
+toString() : String
+clone() : Object
Ananas
-prix : double
-origine : String
+Ananas()
+Ananas(prix : double , origine String)
Banane
-prix : double
-origine : String
+Banane()
+Banane(prix : double , origine String)
Cerise
-prix : double
-origine : String
+Cerise()
+Cerise(prix : double , origine String)
Citron
-prix : double
-origine : String
+Citron()
+Citron(prix : double , origine String)
Fraise
-prix : double
-origine : String
+Fraise()
+Fraise(prix : double , origine String)
Framboise
-prix : double
-origine : String
+Framboise()
+Framboise(prix : double , origine String)
Kiwi
-prix : double
-origine : String
+Kiwi()
+Kiwi(prix : double , origine String)
Litchi
-prix : double
-origine : String
+Litchi()
+Litchi(prix : double , origine String)
Orange
-prix : double
-origine : String
+Orange()
+Orange(prix : double , origine String)
Peche
-prix : double
-origine : String
+Peche()
+Peche(prix : double , origine String)
Pomme
-prix : double
-origine : String
+Pomme()
+Pomme(prix : double , origine String)


exceptions
PanierPleinException
+panierPleinException()
Nom de la classe
+PanierVideException
JAVA
Exception
JFrame
<<Interface>>
PropertyChangeListener
+propertyChange(evt : PropertyChangeEvent) : void
JDialog
views
MarcheFruits
-panier : Panier
-instance : MarcheFruits
-df : DecimalFormat
-listeFruitAffiche : ArrayList<Fruit>
-recetteAffiche : boolean
-listeRecetteAffiche : ArrayList<Recette>
+MarcheFruits()
+initListePanier() : void
+initButtons() : void
-initComponents() : void
-afficheFruits() : void
-afficheRecettes() : void
-remiseAZero() : void
-categorieSansPepinsActionPerformed(evt : java.awt.event.ActionEvent) : void
-categorieRecetteActionPerformed(evt : java.awt.event.ActionEvent) : void
-categorieAgrumeActionPerformed(evt : java.awt.event.ActionEvent) : void
-categorieExotiqueActionPerformed(evt : java.awt.event.ActionEvent) : void
-categorieTousActionPerformed(evt : java.awt.event.ActionEven) : void
-boycottActionPerformed(evt : java.awt.event.ActionEvent) : void
-trierParActionPerformed(evt : java.awt.event.ActionEvent) : void
-boutonVoirPanierMouseClicked(evt : java.awt.event.MouseEvent) : void
+getPanier() : Panier
ValidationPanier
-panier : Panier
-df : DecimalFormat
+ValidationPanier(parent : java.awt.Frame, modal : boolean, p : Panier)
-initListeFruit() : void
+getPanier() : Panier
-initComponents() : void
-BoutonRetirerFruitActionPerformed(evt : java.awt.event.ActionEvent) : void
-BoutonViderPanierActionPerformed(evt : java.awt.event.ActionEvent) : void
-BoutonValiderPanierActionPerformed(evt : java.awt.event.ActionEvent) : void
+getButtonValider() : JButton
+getButtonRetirerFruit() : JButton
+getButtonRetour() : JButton
+getButtonViderPanier() : JButton
+getAffichePanier() : JList
PageFruit
-panier : Panier
-fruit : Fruit
-df : DecimalFormat
-listeRecette : ArrayList<Recette>
-iF : URL
+PageFruit(parent : java.awt.Frame, modal : boolean, f : Fruit, p : Panier)
+initFruitIHM() : void
+getMoinsFruit() : JButton
+getPlusFruit() : JButton
+getAjoutFruit() : JButton
+getNbFruit() : JTextField
-initComponents() : void
-actualiserPrixTotal() : void
-AjoutFruitActionPerformed(evt : java.awt.event.ActionEvent) : void
-PlusFruit(evt java.awt.event.ActionEvent) : void
-MoinsFruit(evt java.awt.event.MouseEvent) : void
PageRecette
-panier : Panier
-recette : Recette
-df : DecimalFormat
-listeFruit : ArrayList<Fruit>
-iR : URL
-prix : double
+PageRecette(parent : java.awt.Frame, modal : boolean, r : Recette, p : Panier)
+initRecetteIHM() : void
-actualiserPrixTotal() : void
-calculPrix() : void
-affichageIngredients : void
-initComponents() : void
+getPlusFruit() : JButton
+getMoinsFruit() : JButton
+getAjoutRecette() : JButton
+getNbRecette() : JTextField
-MoinsRecette(evt java.awt.event.MouseEvent) : void
-AjoutRecette(evt java.awt.event.ActionEvent) : void
-PlusRecetteActionPerformed(evt : java.awt.event.ActionEvent) : void

V. IHM ET MAQUETTES

1. MAQUETTES

Ci-dessous les maquettes sur lesquelles nous nous sommes basées, nous avions prévu quatre interfaces au départ. L'interface principale affichant le panier de l'utilisateur, une seconde interface où l'utilisateur a accès aux fruits en cliquant sur 'Ajouter un fruit', une interface pour afficher un fruit lorsqu'on clique sur un fruit du marché et une dernière interface visuelle pour confirmer la validation du panier en simulant un achat.

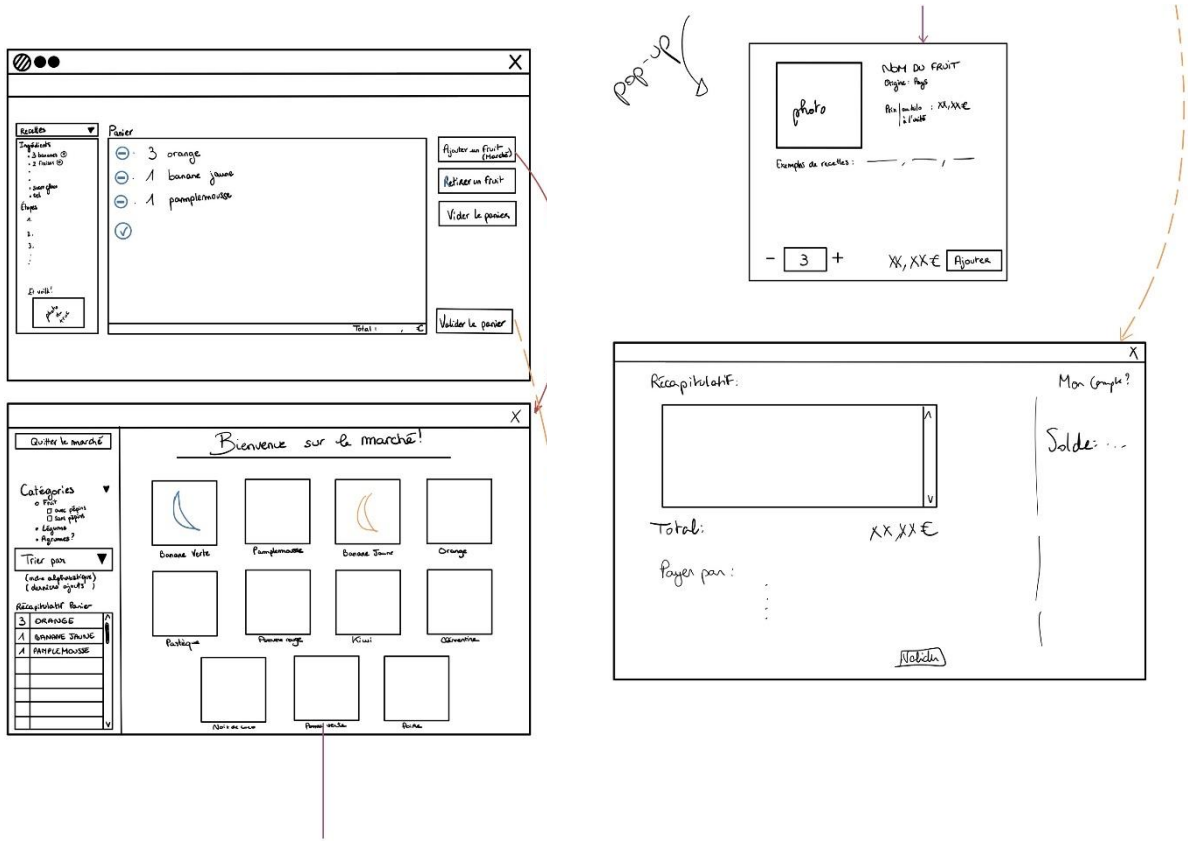


Image 8 : Maquettes de l'application

2. DESIGN DE L'APPLICATION

Au cours du temps, l'IHM a évolué et s'est vu retirer voire ajouter des affichages visuels pour certaines fonctionnalités. Nous avons mis à jour l'interface jusqu'au bout pour une meilleure visibilité pour le client.

Image 9 : Interface principale de l'application / Interface du marché

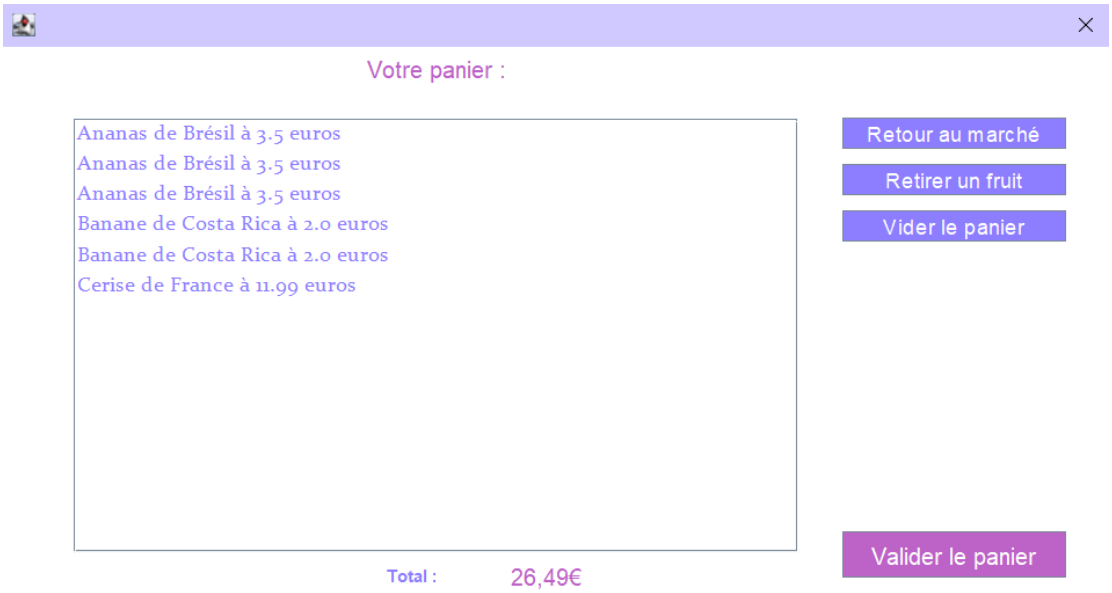


Image 10 : Interface du panier

Image 11 : Interface de présentation d'un fruit

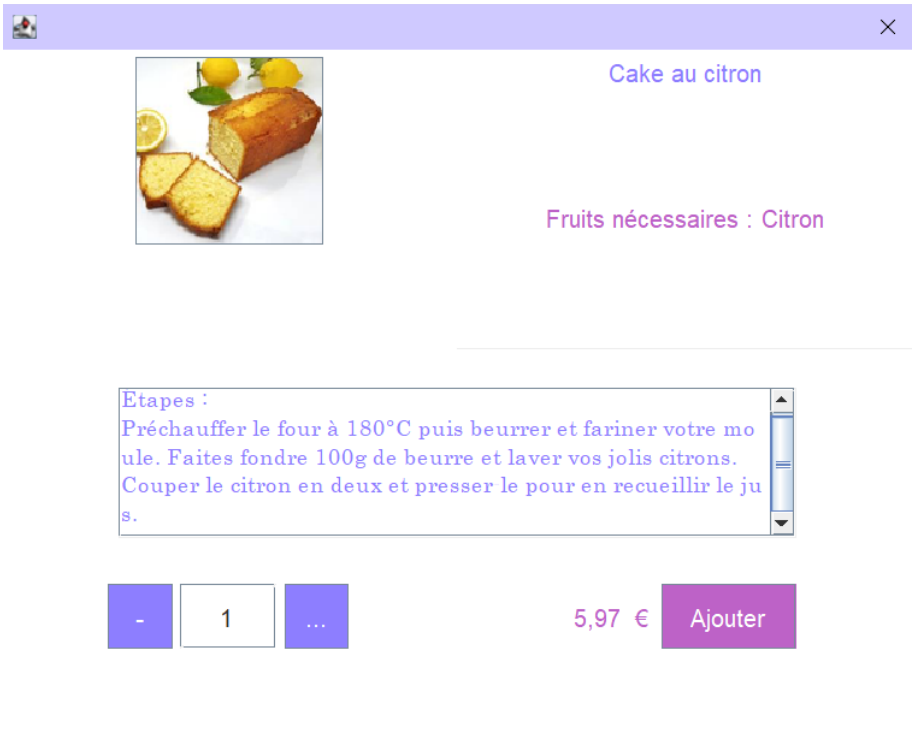
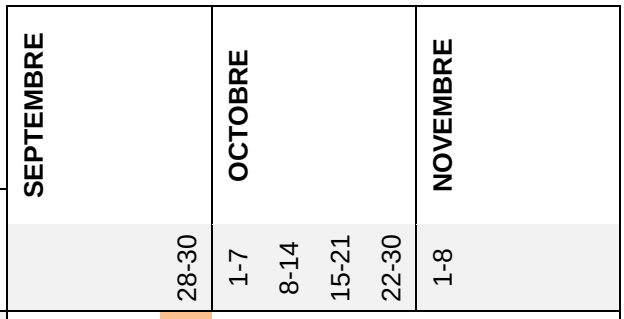


Image 12 : Interface de présentation d'une recette

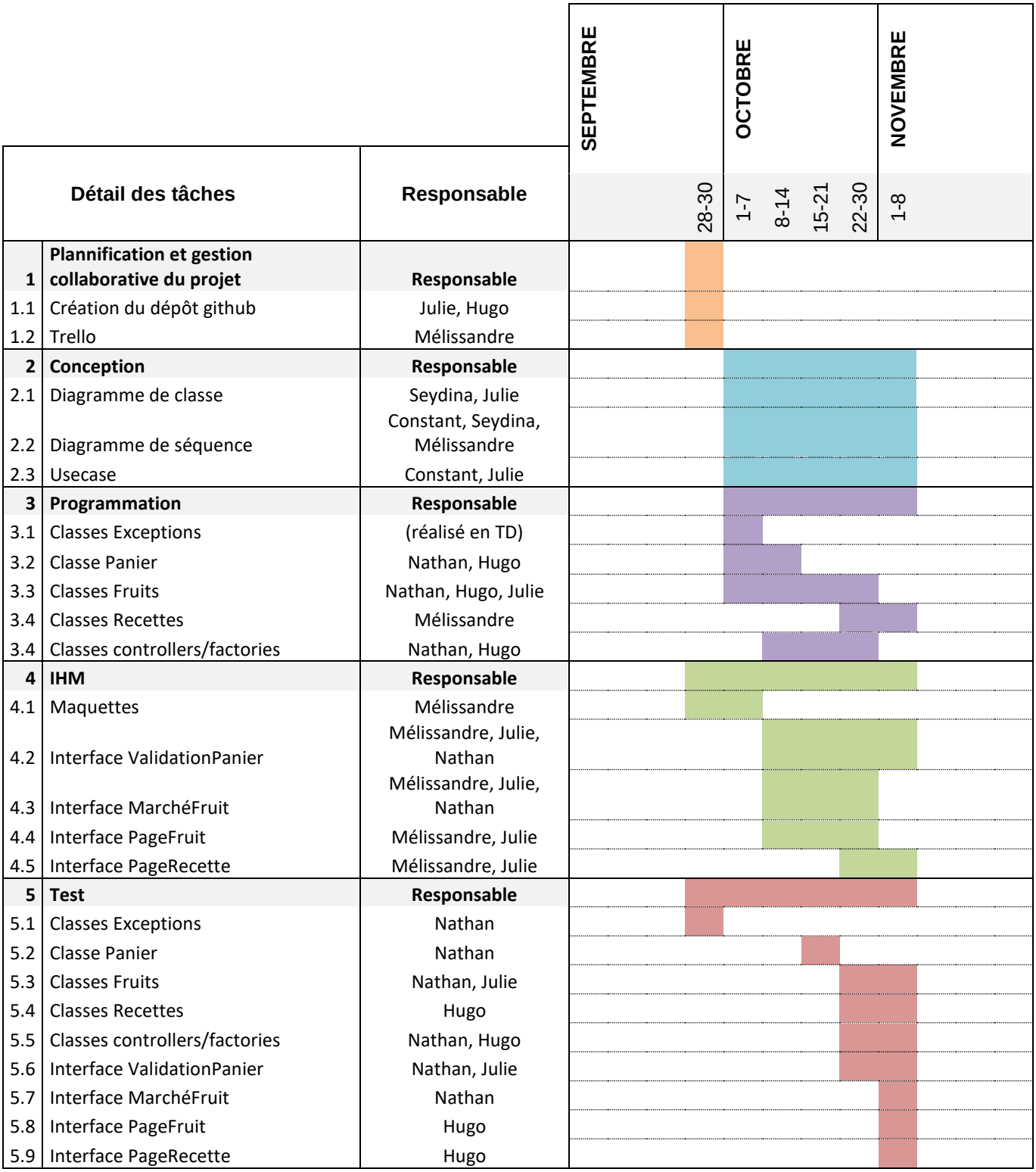
VI. PLANNING ET REALISATION DES TACHES

1. DIAGRAMME DE GANTT PREVU

	Détail des tâches	Responsable
1	**Plannification et gestion collaborative du projet**	**Responsable**
1.1	Création du dépôt github	Julie, Hugo
1.2	Trello	Mélissandre
2	**Conception**	**Responsable**
2.1	Diagramme de classe	Constant, Seydina
2.2	Diagramme de séquence	Constant, Seydina
2.3	Usecase	Constant, Seydina
3	**Programmation**	**Responsable**
3.1	Classes Exceptions	(réalisé en TD)
3.2	Classe Panier	Nathan, Hugo
3.3	Classes Fruits	Nathan, Hugo, Julie
3.4	Classes Recettes	Mélissandre
3.4	Classes controllers/factories	Nathan, Hugo
4	**IHM**	**Responsable**
4.1	Maquettes	Mélissandre
4.2	Interface ValidationPanier	Mélissandre, Julie, Nathan
4.3	Interface MarchéFruit	Mélissandre, Julie, Nathan
4.4	Interface PageFruit	Mélissandre, Julie
4.5	Interface PageRecette	Mélissandre, Julie
5	**Test**	**Responsable**
5.1	Classes Exceptions	Nathan
5.2	Classe Panier	Nathan
5.3	Classes Fruits	Nathan, Julie
5.4	Classes Recettes	Hugo
5.5	Classes controllers/factories	Nathan, Hugo
5.6	Interface ValidationPanier	Nathan, Julie
5.7	Interface MarchéFruit	Nathan
5.8	Interface PageFruit	Hugo
5.9	Interface PageRecette	Hugo



2. DIAGRAMME DE GANTT REEL



	Détail des tâches	Responsable
1	**Plannification et gestion collaborative du projet**	**Responsable**
1.1	Création du dépôt github	Julie, Hugo
1.2	Trello	Mélissandre
2	**Conception**	**Responsable**
2.1	Diagramme de classe	Seydina, Julie
2.2	Diagramme de séquence	Constant, Seydina, Mélissandre
2.3	Usecase	Constant, Julie
3	**Programmation**	**Responsable**
3.1	Classes Exceptions	(réalisé en TD)
3.2	Classe Panier	Nathan, Hugo
3.3	Classes Fruits	Nathan, Hugo, Julie
3.4	Classes Recettes	Mélissandre
3.4	Classes controllers/factories	Nathan, Hugo
4	**IHM**	**Responsable**
4.1	Maquettes	Mélissandre
4.2	Interface ValidationPanier	Mélissandre, Julie, Nathan
4.3	Interface MarchéFruit	Mélissandre, Julie, Nathan
4.4	Interface PageFruit	Mélissandre, Julie
4.5	Interface PageRecette	Mélissandre, Julie
5	**Test**	**Responsable**
5.1	Classes Exceptions	Nathan
5.2	Classe Panier	Nathan
5.3	Classes Fruits	Nathan, Julie
5.4	Classes Recettes	Hugo
5.5	Classes controllers/factories	Nathan, Hugo
5.6	Interface ValidationPanier	Nathan, Julie
5.7	Interface MarchéFruit	Nathan
5.8	Interface PageFruit	Hugo
5.9	Interface PageRecette	Hugo

En comparaison des deux diagrammes on peut voir que toutes les tâches n'ont pas été réalisé suivant le planning. Du retard dans certaines tâches à entrainer le retard d'autres. Cependant, nous avons tout de même réussi à finir les tâches prévues dans les temps.

VII. OUTILS DE COLLABORATION

3. GITHUB

Nous avons utilisé GitHub comme dépôt distant pour notre projet. De cette façon, cela permet à l'équipe de pouvoir récupérer le projet et de travailler en même temps via l'utilisation du système de branche de GitHub.

De plus, nous avons utilisé les issues proposées par GitHub afin de créer des tâches, et de les attribuer à différents membres de l'équipe.

4. DISCORD

Nous avons utilisé Discord pour la partie communication. Avec Discord nous avons pu créer un salon de discussion avec tous les membres du groupe afin de pouvoir s'envoyer des messages, discuter du projet ou poser des questions. L'aspect où tout le monde est sur le salon permet que chaque membre de l'équipe puisse suivre l'avancer de chacun sur le projet.

De plus, nous avons utilisé Discord pour nos réunions en distanciel lorsque des membres de l'équipe ne pouvaient pas se déplacer.

Pour les réunions en présentiel, nous nous réunissions dans une salle de la fac qui étaient disponible.

5. TRELLO

Trello est un outil qui nous a permis de détailler et afficher nos tâches. De cette façon, chaque membre de l'équipe pouvait voir les tâches à faire, et de les attribuer. Trello permet également de notifier l'avancer d'une tâche : pas commencer, en cours, ou terminer. Cette fonctionnalité nous a été utile afin de bien répartir le travail et nous permettre de savoir où chaque membre de notre équipe en était dans leur tâche, et donc l'avancée du projet.

VIII. REUNIONS

20 octobre, réunion en distanciel review et retrospective sur ce qui a été fait et d'une discussion sur les diagrammes. Nous avons essayé d'avoir une meilleure vue sur les IHM, afin de mettre en place pour la prochaine réunion prévue le lundi 23 octobre en présentiel

Lundi 23 octobre, nous avons fait une réunion en présentiel durant laquelle nous avons parlé des diagrammes de classes et de la mise en place des design pattern. Avec cela, nous avons donc fais un point sur ce qui avait été fait et ce qu'il restait à faire. Nous avons réfléchi à de nouvelles fonctionnalités et aux ajouts des classes associés.

Mardi 31 octobre réunion récapitulative avant la fin du projet, résumé des taches effectué et attributions des tâches qui restaient à faire.

IX. RAPPORT DE COUVERTURE

Nous avons exigé une couverture maximale pour les models, et les controllers.

Les fonctionnalités des vues ont été testées, par exemple les comportements des boutons.

ProjetGL

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
fr.ufrsciencestech.views		95 %		90 %	25	138	82	1 015	21	107	1	27
fr.ufrsciencestech		86 %		n/a	1	4	2	7	1	4	0	2
fr.ufrsciencestech.models.fruits		100 %		100 %	0	209	0	308	0	143	0	11
fr.ufrsciencestech.models		100 %		100 %	0	44	0	105	0	22	0	1
fr.ufrsciencestech.models.recettes		100 %		100 %	0	32	0	81	0	24	0	4
fr.ufrsciencestech.controllers.factories		100 %		100 %	0	43	0	74	0	12	0	3
fr.ufrsciencestech.controllers		100 %		100 %	0	33	0	62	0	6	0	1
fr.ufrsciencestech.utils		100 %		n/a	0	5	0	35	0	5	0	5
fr.ufrsciencestech.exceptions		100 %		n/a	0	2	0	4	0	2	0	2
Total	279 of 9 451	97 %	6 of 341	98 %	26	510	84	1 691	22	325	1	56

X. CONCLUSION

L'avancée constante du projet grâce à la méthode agile a été des plus favorables. Nous avons ainsi réussi à recentrer l'évolution des besoins de l'utilisateur en fonction de l'avancée de chacun des membres du groupe.