Réfrigerateur connecte

Dossier de conception

IUT LYON 1

LANDRY Florian | Parseihian-hoppenot tom | tavares alexis | willemy malik

S2G3.1

Table des matières

[I) Introduction 2](#_Toc10644720)

[II) Les interfaces 2](#_Toc10644721)

[III) Diagramme de classes 2](#_Toc10644722)

[IV) Scénarios et diagrammes de séquence 3](#_Toc10644723)

[V) Conclusion 4](#_Toc10644730)

# Introduction

# Les interfaces

Liste\_de\_Courses ->

* int addAliment(Aliment a)

    Ajout de a dans ListeCourse

        return 1 si ajouté

        return 0 si problème d’ajout

**addAliment**

L’utilisateur insère un Aliment qu’il veut ajouter à sa Liste de Courses. Il précise la liste dans laquelle il souhaite insérer l'aliment choisi. Si cet Aliment n’existe pas, la fonction retourne 0. Si elle réussit à l’ajouter elle retourne 1.

* int deleteAliment(Aliment a)

Effacement de a dans ListeCourse

    return 1 si a trouvé et supprimé

return 0 sinon

**deleteAliment**

L’utilisateur sélectionne une Liste de course. Cette liste est affichée avec son contenu. L’utilisateur peut ensuite sélectionner un Aliment pour le supprimer, celui-ci est retiré de celle-ci.

* double prixTotal()

Soit *prixT* un double qui additionne les *Prix \* Quantité* contenus dans ListeCourse

return 0.0 si pas d’aliments

Sinon return *prixT*

**prixTotal**

Parcours la Liste De Courses et renvoie la somme des éléments présents dans la liste, multipliés par leur quantité respective.

* void displayListe()

Parcours ListeCourse et affiche les Aliments

        Affiche les informations de chaque aliments

**displayListe**

Affiche la liste des Aliments présents dans la Liste De Courses ainsi que le *prix total* quand l’utilisateur le demande.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Assistant\_Vocal ->

* String ecouter()

    return la requête orale

**ecouter**

Enregistre les mots prononcés par l’utilisateur et les renvoie sous forme de chaîne de caractères.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Recettes ->

* string displayRecette(string nom)

Envoie une requête par HTTP sur Marmiton le nom

Si la requête est valide alors

return le texte associé à la recette

return “Aucune recette correspondante” sinon

**displayRecette**

Nous tapons là le nom de la recette voulue. On demande à Marmiton si une telle recette existe et récupérons le texte associé. Sinon, on affiche un message d’erreur.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Aliment->

* void setPrix(double p)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Frigo->

* void initialisation()

Affiche l’interface de base:

    Affichage de l’heure système

    Affichage de la température en appelant la méthode **getTemperature()**

    Affichage des différentes icônes

Lancement de l’assistant vocal par la fonction Assistant\_Vocal.**ecouter()**

Mise en place du système de routine, permettant d’appeler à l’intervalle régulier **getTemperature** et Capteur.**detectAliment()**

**Initialisation**

C’est la fonction essentielle afin d’avoir une interaction utilisateur/machine, qui permet d’avoir une interface facile d’utilisation. Elle permet d’accéder à des logiciels par les icônes.

* void addNote(Note n);

    Ajoute n dans ListeNote

**addNote**

Ajoute une Note à Liste de Note

* void displayNote()

    Parcours ListeNote et affiche les Notes

        Affiche le contenu de chaque Note

**displayNote**

Affiche les Notes présent dans la liste de Notes sur l’écran

* boolean modifyNote(Note n1, string text)

    Compare n1 à chaque éléments de ListeNote

   Lorsque n1 = n voulu de ListeNote

        Modification du texte présent dans n par text

return true

Sinon return false

**modifyNote**

La méthode reçoit une Note qu’il compare aux Notes existantes et un texte. Une fois trouvée, on change le texte présent dans celle-ci par le texte reçu par la méthode

* void deleteNote(Note n)

Effacement de n dans ListeNote

    return 1 si n trouvé et supprimé

return 0 sinon

**deleteNote**

    Supprime la Note de la Liste de Notes correspondante à l'élément sélectionné

* Double getTemperature()

    Si le capteur de température fonctionne

        return la valeur perçu par le capteur de température

    return -1.0 sinon

**getTemperature**

    Le contrôle de la température d’un réfrigérateur est essentiel pour la bonne conservation des aliments. Il est donc important de connaître cette température régulièrement et facilement.

* void modifyListeCourses(ListeCourse ldlc)

**modifyListeCourses**

    Affiche la liste de course sélectionnée sur l’écran du réfrigérateur. Indique la quantité de chaque aliment a droite de son nom. La quantité sera ajustable via des boutons “+” et “-” affiché de part et d’autre de la quantité. Lorsque l’utilisateur appuie sur un aliment dans la liste affichée, un menu contextuel composé de deux boutons, un bouton “stylo” et un bouton “poubelle”, apparaît. Le bouton “stylo”, il permet de choisir un nouvel aliment qui remplacera l’aliment précédent. Le bouton “poubelle”, il permet de supprimer l’aliment de la liste de course sélectionnée. Un bouton “+” est affiché à côté du nom de la liste de course, il permet d’ajouter un aliment en faisant appel à la fonction ListeCourse.addAliment. Une flèche de retour est affichée sur l’écran afin de quitter le menu de modification de course. Lorsque l’utilisateur quitte l’interface de modification de liste, on vérifie la quantité de chaque aliment présent, si cette quantité vaut 0, l’aliment sera supprimé de la liste de course.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Capteurs->

* Aliment detectAliment()

Vérifie l’état du capteur

Si = 0 alors

Return Aliment

Sinon

Return null

**detectAliment**

Régulièrement, le système vérifie l’état des capteurs. Si l’un d’eux est relâché, alors il appelle la fonction **modifyListeCourses** afin de supprimer l’Aliment en question dans ListeAliment.

* void detecteOuverture()

Soit un entier *compteur*

La lampe s’allume

Si le capteur associé à la porte = 0

Incrémente le *compteur* toutes les secondes

Si ouvert depuis 30 secondes

        → Sonnerie

Si frigo fermé

        compteur = 0

        La lampe s’éteint

**detecteOuverture**

Si le Réfrigérateur est ouvert, alors la lumière s’allume. Si la porte est ouverte depuis 30 secondes, la méthode déclenche une alarme sonore pour rappeler les utilisateurs dans les alentours à la refermer.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Drive->

* boolean listeCommandable(Liste\_de\_Courses Course L)

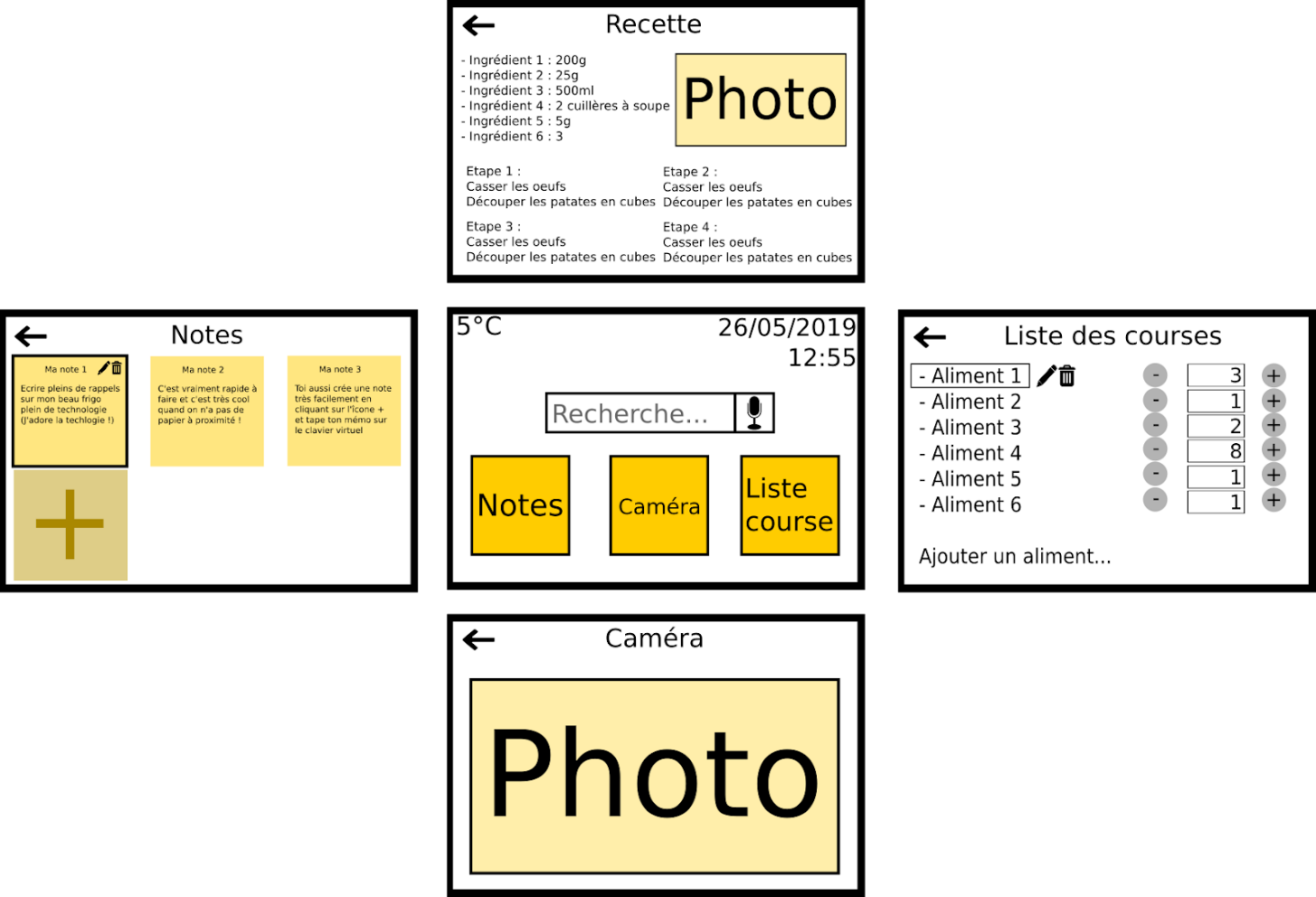
Envoi une requête HTTP auprès du Drive sélectionné en comparant chaque Aliments de la ListeCourse à leur Base de données

Si Aliment != à la requête

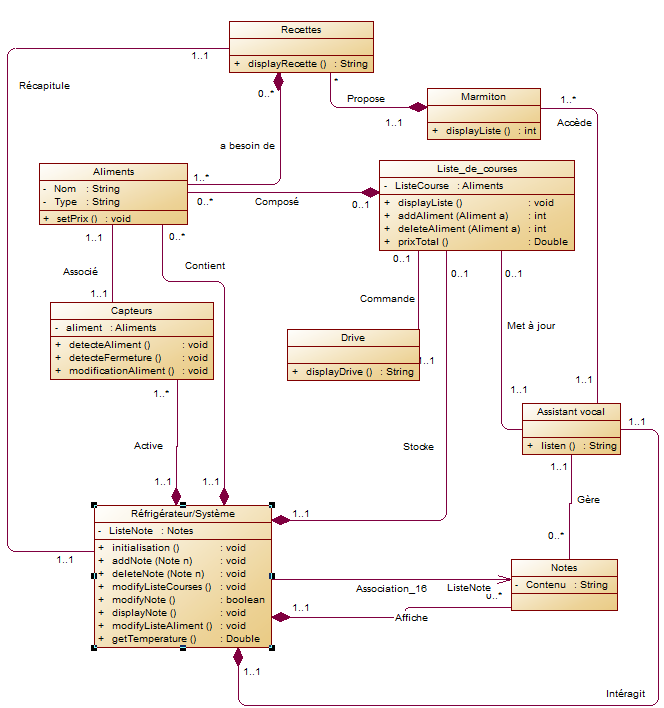
    return false

return true sinon

# Maquette

[](https://online.visual-paradigm.com/diagrams.jsp#diagram:proj=0&type=SequenceDiagram&gallery=/repository/46f00715-b2be-42be-9d40-30c371a0001b.xml&name=Object%20Creation%20and%20Deletion)

# Diagramme de classes



# Scénarios et diagrammes de séquence

Scénario 1

* Un premier capteur détecte l’ouverture de la porte du frigo
* Un autre capteur détecte la bouteille de lait que l’utilisateur enlève
* Il envoie la donnée à notre programme
* Notre programme met à jour la liste des aliments disponibles dans le frigo en supprimant la bouteille de lait

Scénario 2

* L’utilisateur tape sur l’icône “notes”
* Le programme reçoit l’objet qui a été tapé
* Il lance la fonction **displayNotes();**
* L’utilisateur tape l’icône “petit stylo” de la note 2
* La fonction modify() est appelée
* L’utilisateur tape le nouveau texte qu’il souhaite à l’aide du clavier virtuel, il est stocké sous le type String
* L’utilisateur tape le bouton “Valider”
* Le nouveau String remplace l’ancien dan sle stockage
* L’application relance la fonction **displayNotes();**

Scénario 3

* L’utilisateur tape l’icône “Liste de courses”
* Le programme reçoit l’objet qu’il a tapé
* Il lance la fonction displayListeCourses();
* L’utilisateur veut supprimer l’aliment dernièrement ajouté
* Il fait glisser son doigt vers le haut pour que l’affichage des notes aille vers le bas
* Une fois arrivé en bas, il tape le dernier aliment de la liste
* Ce dernier devient sélectionné
* Un mini menu contextuel s’affiche à la droite de l’aliment selectionné montrant 4 options : +, -, “petit stylo”, “poubelle”
* L'utilisateur tape la poubelle
* Le programme récupère cette action et supprime l’aliment de la liste
* L’utilisateur tape ensuite la touche “Ajouter” de la fenêtre, située en haut à droite
* Le programme ouvre une boîte de dialogue avec deux *TextField*
* L’utilisateur, à l’aide du clavier virtuel, entre le nom de l’aliment dans le premier, puis la quantité dans le second, il tape ensuite “valider”
* Le premier String est sauvegardé comme tel, le second est converti en *int* avant d’être enregistré
* La fonction **displayListeCourse()** est ensuite appelée, l’aliment étant bien affiché

Scénario 4

* L’utilisateur commence à modifier une note
* Il appuie sur le “Ajouter” de la fenêtre pour ajouter une note en même temps
* Hélas, cela ne fonctionne pas. Le programme empêche la création d’une nouvelle note lorsque l’on est déjà en train d’en modifier une
* Le programme affiche donc une fenêtre pop-up composée d’un *JLabel* où il est écrit “Veuillez valider ou annuler avant de créer une nouvelle note” et d’un *JButton* où il est écrit “Compris !”
* L’utilisateur tape le “Compris !” et termine la modification de sa note pour en créer une nouvelle

Scénario 5

* L’utilisateur recherche la recette de l’omelette
* Il n’est pas connecté au réseau *Wi-Fi* !
* Il est redirigé sur la page de connexion *Wi-Fi*
* L’utilisateur rentre le login et le mot de passe
* La machine se connecte
* La page se ferme, laissant apparaître la recette de l’omelette

Scénario 6

* L’utilisateur ferme la porte du frigo
* Le capteur de détection de fermeture ne marche plus
* Le *buzzer* sonne au bout de 30 secondes
* L’utilisateur ne peut pas l’arrêter
* Il redémarre le système
* Tous les capteurs sont donc aussi réinitialisés
* Le capteur de détection de fermeture fonctionne à nouveau

# Conclusion