

Épreuve E6.2
Rapport de projet

Suivi d'une production de jus de pommes artisanal

La Cidrerie de la Baleine



Corentin RIBEZZO

Lycée Vauban – Brest (29)
BTS Systèmes numériques, Informatique et réseaux

Sommaire

1. Introduction.....	5
1.1. Analyse de l'existant.....	5
1.2. Analyse du besoin.....	7
1.3. Problématique.....	7
2. Exigences du système embarqué.....	9
2.1. Répartition sommaire des tâches.....	9
2.1. Cahier des charges du code source.....	9
3. Conception générale du système.....	11
3.1. Structure globale du système.....	11
3.2. Synoptique d'utilisation de l'interface utilisateur.....	12
3.3. Étude préliminaire des interfaces utilisateur.....	13
3.3.1. Modélisation comportementale de l'interface principale.....	13
3.3.2. Modélisation comportementale de l'interface de récolte.....	14
3.3.3. Modélisation comportementale de l'interface de remplissage.....	15
3.3.4. Modélisation comportementale de l'interface de pasteurisation.....	16
3.4. Justification des technologies utilisées.....	17
4. Étude détaillée de l'interface homme-machine.....	19
4.1. Introduction aux contrôles d'interfaces utilisateur.....	19
4.1.1. Conception détaillée du bouton standard.....	19
4.1.1. Conception détaillée du bouton de liste déroulante.....	20
4.1.1. Conception détaillée du bouton de sélection.....	20
4.1.1. Conception détaillée de la zone de saisie.....	21
4.2. Introduction aux formes non-interactives de l'interface utilisateur.....	22
4.2.1. Conception détaillée de l'afficheur à sept segments.....	22
4.1.1. Conception détaillée du cadran progressif.....	24
5. Réalisation du code applicatif.....	25
5.1. Implémentation de matrices de pixels dans le code source..	25
5.2. Utilisation de la fonction de rappel.....	26
5.3. Utilisation de la fonction de mise à jour asynchrone.....	27
6. Tests unitaires.....	30
6.1. Vérification des retours console des contrôles d'interface utilisateur.....	30
6.1. Vérification de la réception des données de production sur sortie standard.....	31
7. Intégration de la base de données à l'interface homme-machine.....	33
8. Conclusion.....	35
8.1. Bilan.....	35
6.1. Perspectives d'améliorations.....	35
Annexes.....	36

A.1. Maquettes prototypes de l'interface principale.....	36
A.2. Maquette prototype de l'interface de récolte.....	37
A.3. Maquettes prototypes de l'interface de production.....	38
A.3. Maquette prototype de l'interface d'administration.....	39
A.1. Contenu du fichier pixmap « bag.xpm ».....	40
A.2. Code source de l'interface principale.....	42
A.3. Code source de l'interface de récolte.....	43
A.4. Code source de la bibliothèque « display.h ».....	45
A.4. Code source de la fonction de rappel de l'afficheur à 7 segments.....	45
A.5. Code source de l'interface de remplissage.....	45
A.6. Code source de la fonction de rappel des cadrans.....	46
A.7. Code source de l'interface de pasteurisation.....	46

1. Introduction

De son nom « E6.2 », cette épreuve orale ponctuelle prend appui sur ce dossier de projet. Celui-ci contient les chapitres et sections décrivant les étapes contribuant au développement d'un système embarqué. Le projet ne repose pas seulement sur un travail individuel car il rassemble électronique et informatique - une collaboration essentielle afin de mener à bien ce projet.

1.1. Analyse de l'existant

Ces travaux de recherches et de conceptions ont pour but d'améliorer les conditions de production de jus de pomme pour l'association la cidrerie de la Baleine, où travaillent des bénévoles engagés dans une production dont le travail manuel et épuisant suggère une automatisation de l'organisation et des mécanismes de production au service de cette association.

Au début des années 2000, Jean-Yves Breton et un groupe d'amis fondent cette association. Avec les années, la structure a gagné en confiance et en clients. Aujourd'hui, le fondateur et les membres de cette association récoltent des surplus de pommes ramassés dans les vergés environnants de Plabennec. En moyenne, ils parviennent à produire 7300 litres de jus de pommes par an via l'opération « Pompompidou ». Grâce à la la vente du jus de pommes, la Cidrerie de la Baleine vient en aide à d'autres associations.



Fig. 1 : Récolte de dons de pommes à l'association

La récolte est généralement effectuée par les membres de l'association. La figure 1 ci-dessus montre que l'aide d'extérieurs est la bienvenue.

Actuellement, la Cidrerie de la Baleine propose une production de jus de pommes artisanal avec des moyens primaires que nous proposons d'améliorer.

Les étapes permettant la production et la vente de jus de pommes sont les suivantes :

1. Récolte et ensachage par 20 kg de pommes
2. Triage, rinçage et broyage des pommes
3. Ensachage et pressage du broya à la presse hydraulique puis extraction et filtrage du jus
4. Pompage et stockage du jus dans deux cuves de 1000 litres puis décantation pendant 18h
5. Remplissage à la main de bouteilles consignées et lavées, bouchées puis placées en casiers de capacité de 12 bouteilles
6. Pasteurisation des bouteilles dans 5 cuves à bain-marie, d'une capacité de 33 bouteilles, à 76°C pendant 10 minutes
7. Mise en casier des bouteilles et vérification du serrage des bouchons
8. Rinçage des bouteilles permettant de nettoyer et de s'assurer de l'étanchéité des bouchons
9. Entreposage des bouteilles avant la vente



Fig. 2 : Tri et nettoyage de la récolte

Le rinçage des pommes présenté en amont dans la figure 2 précède l'étape du broyage des pommes et permet d'éliminer les impuretés visibles à l'œil nu.

1.2. Analyse du besoin

En nous appuyant sur les besoins de cette associations, nous devons imaginer des moyens à appliquer à certaines étapes de gestion des campagnes, de production et de vente.

☑ Ce que nous savons

Les ateliers de l'association disposent initialement de moyens simples pour la gestion et le recensement de récoltants et des campagnes, à savoir :

- Un **tableau** comptabilisant l'effectif de récoltants et de campagnes en cours ainsi que la date et l'heure

Lors de la pasteurisation du jus, la surveillance des températures des cuves à bain-marie se fait à l'œil nu et à la main sur :

- De simples **thermomètres**, ce qui inclut des risques de brûlures et un manque de visibilité à cause de la trop petite taille des nombres gradués

De plus, le remplissage des bouteilles se fait **à la main**. La quantité dans chaque bouteille n'est donc pas toujours équitable.

☑ Ce que nous proposons

Après constatation des conditions de gestion et de production de cette association, notre équipe propose de travailler sur différentes améliorations qui pourraient permettre de réduire la pénibilité des conditions de travail des bénévoles :

Logiciel

- Facilité de gestion des campagnes avec un accès direct à une **IHM** (Interface Homme-Machine) associée à une base de données, sur écran tactile
- Accessibilité de la l'état de pasteurisation du jus via une **IHM** dynamique actualisée toutes les 30 secondes
- Un coup d'œil rapide sur le nombre de bouteilles remplies, en vue de potentielles statistiques sur **IHM**

Matériel

- Simplicité d'utilisation avec un **écran tactile** connecté à une **Raspberry Pi** (petit ordinateur) pour héberger l'ensemble des IHM ainsi que la base de données
- Exactitude des températures prélevées lors de la pasteurisation grâce à des **sondes** plongées dans les bain-maries
- Précision de remplissage avec au moyen de **pistolets** à pression semi-automatique, idéal pour une production dynamique et agréable

1.3. Problématique

La production de jus de pommes se fait dans des conditions parfois difficiles sans compter l'approximation des procédés de production. Outre certains risques sur la santé, ces méthodes de production ne sont pas efficaces et nous invitent à réfléchir sur une automatisation de ces tâches manuelles et épuisantes. Ainsi, pour parvenir

aux améliorations prévus, nous devons nous pencher sur les besoins de cette association afin de mettre en place un cahier des charges adapté.

2. Exigences du système embarqué

Afin de répondre aux objectifs introduits précédemment, nous devons d'abord répartir les tâches de chacun. Le cahier des charges présenté en section 2 de ce chapitre est cependant personnel.

2.1. Répartition sommaire des tâches

Le diagramme de structure représenté ci-dessous présente le rôle de chaque étudiant du groupe. Les élèves dits « IR » sont en charge de la partie dite « logiciel » et les élèves dits « EC » sont eux en charge de la partie dite « électronique » et « communication » à mettre en place dans la structure globale de notre système.

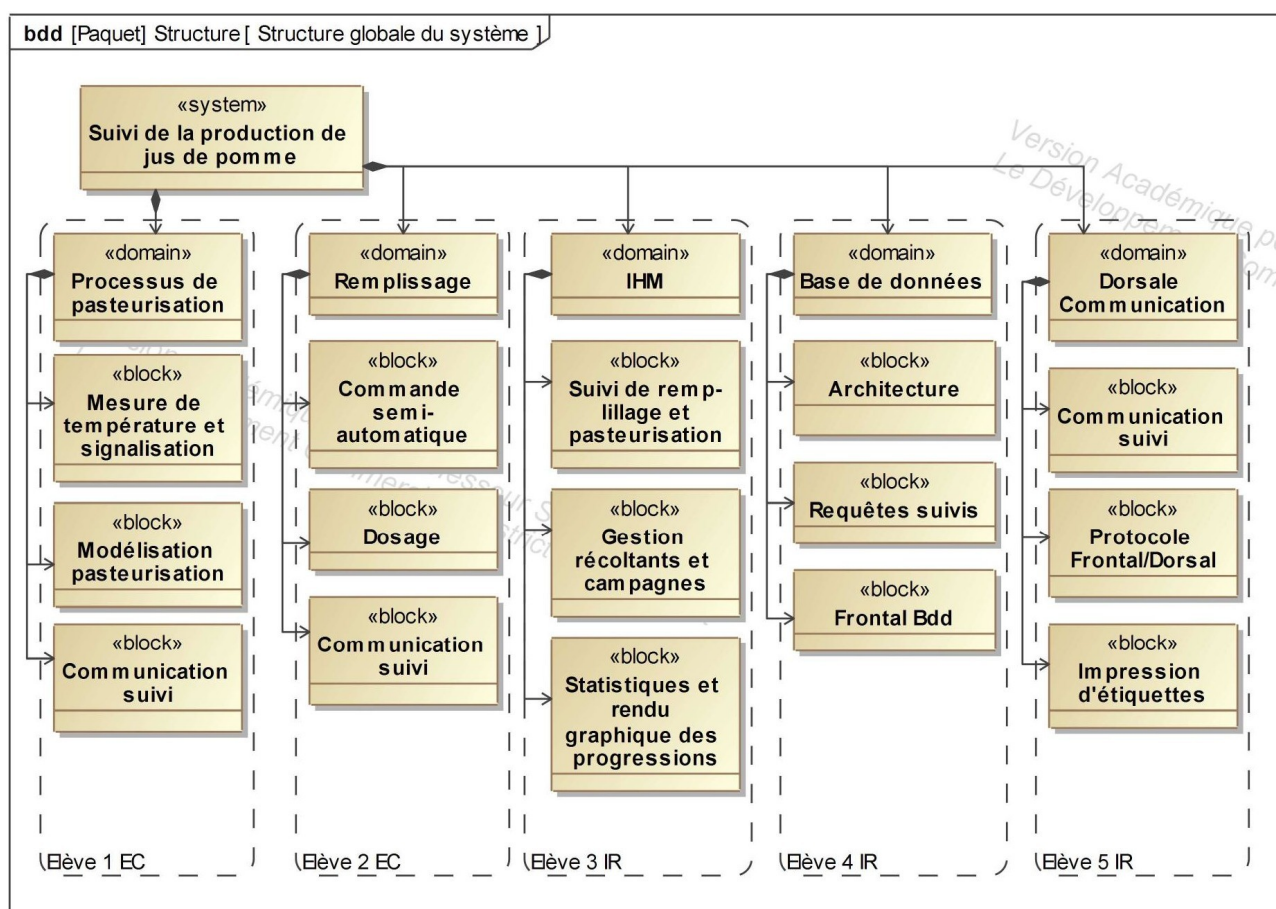


Fig. 3 : Diagramme de la structure globale du système

Pour ma part, les missions auxquelles je suis rattaché consistent en l'implémentation du fichier de données des interfaces principale, secondaires, et sous-interfaces ainsi que celle du code source récupérant le nombre de bouteilles remplies ainsi que l'état de pasteurisation des bouteilles.

Le détail de mes objectifs est défini en section suivante dans les tableaux 1 et 2.

2.1. Cahier des charges du code source

Ce cahier des charges ne s'applique donc qu'à l'étudiant 3. Mon objectif est d'implémenter l'interface utilisateur et le fichier de données permettant la réception de la trame de données du remplissage de bouteilles ainsi que celle de l'état de la pasteurisation de bouteilles, de sorte qu'elles soient lisibles pour un utilisateur

lambda. Les trames sont transmises de la carte électronique vers l'ordinateur via liaison RS-422.

Le tableau 1 ensuite présenté définit les exigences du cahier des charges concernant les interfaces utilisateur de l'IHM.

Nature	Interface	Fonctions	Exigences fonctionnelles	Exigences techniques	Environnement d'exécution	Support matériel
Interfaces	Récolte	Interface utilisateur dédiée à la récolte	- Identité - Nombre de sacs - Variété - Impression d'étiquettes	- Xforms Toolkit for X (Language C)	- Linux Debian 12.4.0	- Raspberry Pi 3b+ dotée d'un écran tactile
	Gestionnaire	Interface dédiée à la gestion des récoltants et des campagnes	- Ajouter une identité - Supprimer identité - Nouvelle variété - Créer une nouvelle campagne - Statistique des campagnes effectuées			
	Remplissage	Interface dédiée au remplissage des bouteilles	- Nombre de bouteilles remplies			
	Pasteurisation	Interface dédiée à la pasteurisation sur 5 cuves à bain-marie	- Statut de pasteurisation			

Tab. 1 : Cahier des charges des interfaces utilisateur de l'IHM

Le tableau 2 ci-dessous traite du cahier des charges du fichier de données.

Nature	Fichiers de données	Fonctions	Exigences fonctionnelles	Exigences techniques	Environnement d'exécution	Support matériel
Code source	Remplissage	Fonctions permettant d'identifier le nombre total de bouteilles remplies et de produire un rendu graphique des progressions à l'écran	Identifier : - numéros de pistolet - température enregistrée par chaque capteur Extraire : - informations identifiées Afficher : - nombre de bouteilles remplies	Language C	- Linux Debian 12.4.0	- Raspberry Pi 3b+ dotée d'un écran tactile
	Pasteurisation	Fonctions permettant d'identifier chaque valeur de capteurs, d'extraire	Identifier : - données de températures Extraire : - informations identifiées Convertir : - données de températures en ° Afficher : - progression de température de chaque cuve à bain-marie			

Tab. 2 : Cahier des charges du fichier de données

3. Conception générale du système

Ce chapitre aborde la conception générale des interfaces utilisateurs de l'IHM ainsi que celle des fonctions relevant les données de production. L'utilisation d'algorigrammes est un bon moyen de décrire les processus permettant à l'utilisateur d'accéder aux services que nous proposons.

3.1. Structure globale du système

En guise de conception générale du système, nous avons imaginé sa structure puis réalisé le diagramme UML correspondant aux composantes nécessaires.

La figure 4 ci-dessous constitue le diagramme de déploiement du système, représentant chaque aspect du projet à apporter.

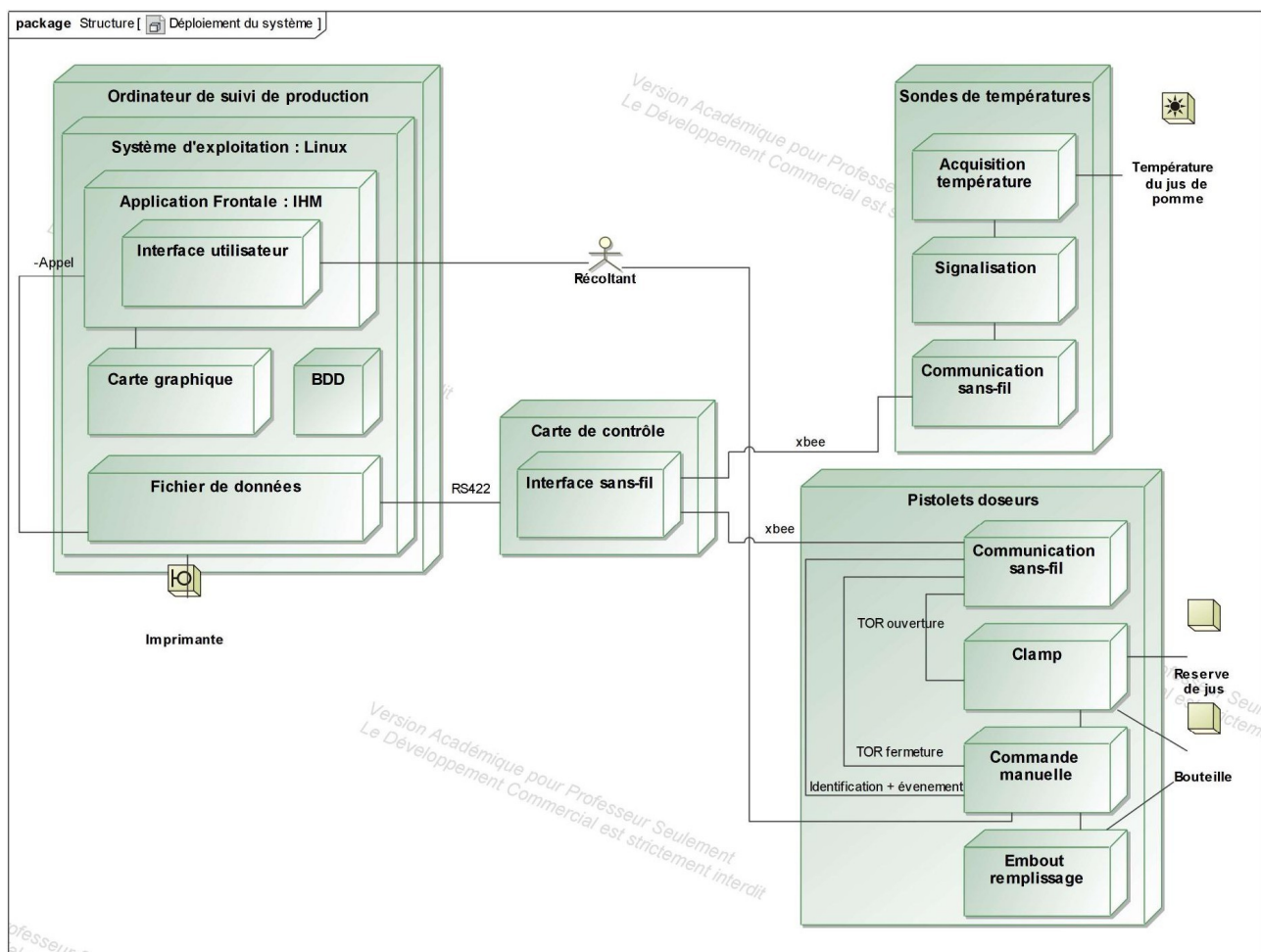


Fig. 4 : Diagramme de déploiement du système

Ce diagramme illustre la configuration matérielle, les connexions entre les éléments de ce système ainsi que leur déploiement physique. Outre l'aspect matériel du système, ce diagramme permet de situer l'utilisateur ayant à la fois accès au système embarqué permettant de suivre la production et aux composantes physiques telles que celles utilisées pour le remplissage des bouteilles et celles permettant de surveiller la pasteurisation. Chaque nœud ou « boîte » constitue un élément du système dont l'utilisation est essentielle afin que le projet respecte les exigences.

L'« application frontale » ou IHM s'exécutant sur un système d'exploitation Linux Debian 12, installé sur la Raspberry Pi, est étudiée en conception générale dans le chapitre suivant.

3.2. Synoptique d'utilisation de l'interface utilisateur

Une interface homme-machine laisse à l'utilisateur l'évidente possibilité d'interagir avec le contenu graphique qui lui est proposé. La figure 5 ci-dessous présente globalement l'utilisation du système et assure au client le bon fonctionnement de l'ensemble des services proposés.

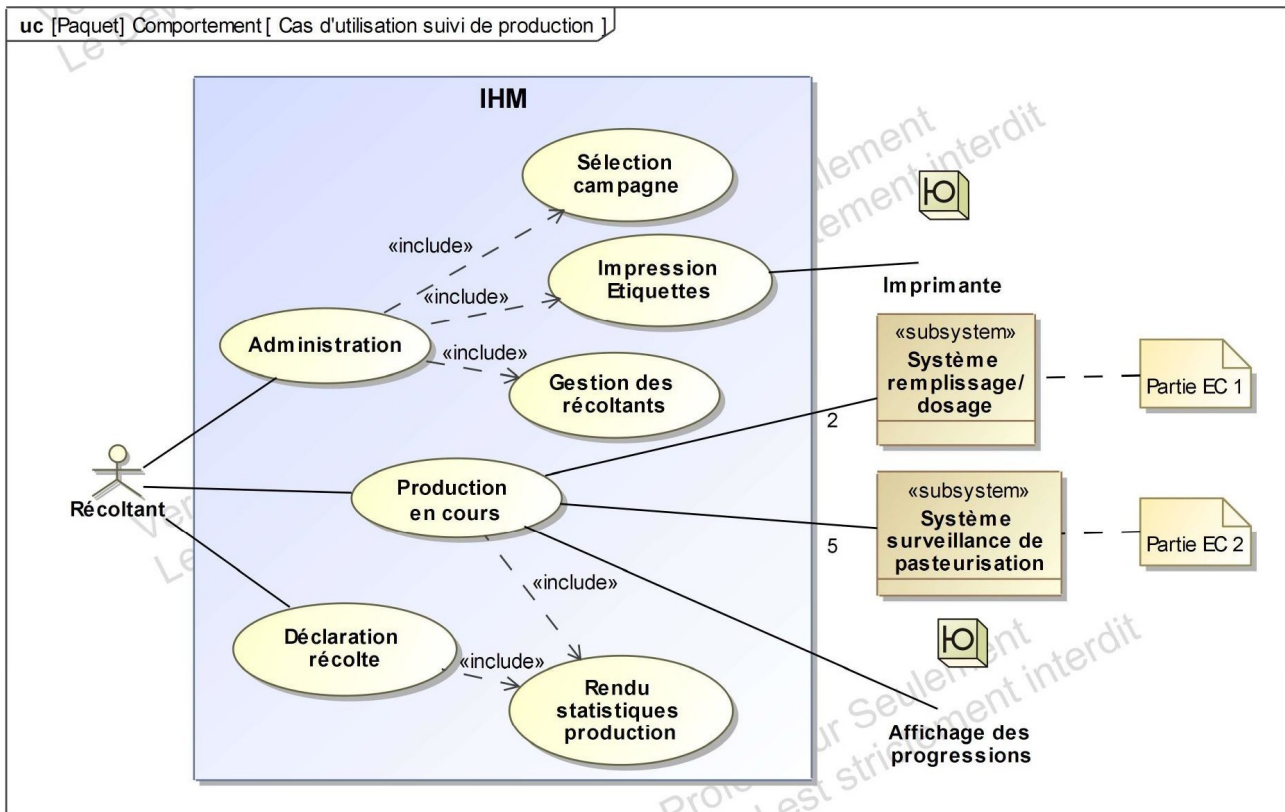


Fig. 5 : Diagramme de cas d'utilisation de l'IHM

Le diagramme ci-dessus présente les interactions entre l'utilisateur ici identifiable par sa fonction de « Récoltant » et le système représenté par « IHM ». Ce diagramme met en lumière les relations entre chaque cas d'utilisation.

L'acteur principal de ce système interagit avec plusieurs cas d'utilisation pour accomplir diverses tâches essentielles à la gestion de la production. Le cas d'utilisation « Administration » joue un rôle central en intégrant plusieurs sous-fonctions indispensables. Pour administrer efficacement les opérations, le récoltant doit sélectionner une campagne de production via le cas d'utilisation « Sélection campagne ». Une fois la campagne choisie, il est possible d'imprimer des étiquettes à disposer sur les bouteilles, géré par « Impression Étiquettes ». De plus, l'administration inclut la « Gestion des récoltants », permettant de gérer les informations des différents récoltants impliqués dans la campagne.

La « Déclaration récolte » est un autre cas d'utilisation clé, permettant au récoltant de déclarer les récoltes effectuées. Ainsi, pour déclarer une récolte, il est indispensable de produire des données statistiques, soulignant la relation d'inclusion entre ces deux cas d'utilisation.

Par ailleurs, le cas d'utilisation « Production en cours » permet au récoltant de suivre et de gérer les différentes étapes de la production en temps réel. Ce processus est soutenu par deux sous-systèmes critiques. Le « Système de remplissage/dosage » assure le bon remplissage des produits, et le « Système de surveillance de pasteurisation » surveille le processus de pasteurisation pour garantir la qualité du produit final. Bien que ces interactions ne soient pas explicitement des relations d'inclusion, elles sont essentielles au bon déroulement de la production.

La section 3.3. de ce chapitre comprend le résumé des fonctionnalités de chaque interface.

3.3. Étude préliminaire des interfaces utilisateur

L'interface homme-machine contient une interface principale, quatre interfaces secondaire dont l'enchaînement des processus peut être décrit avec l'utilisation d'algorigrammes.

3.3.1. Modélisation comportementale de l'interface principale

Cette interface présente simplement les principales actions possibles pour l'utilisateur, à savoir :

- ➔ La déclaration d'une récolte, possible grâce à un bouton simple pour y accéder
- ➔ L'administration des récoltants ainsi que des campagnes en cours ou à démarrer, enregistrés en base de données par le biais d'un bouton simple
- ➔ Les données de remplissage affichées sont accessible par un bouton simple à presser
- ➔ L'état de la pasteurisation est accessible au moyen d'un autre bouton simple

L'enchaînement des processus des interactions de l'utilisateur avec cette interface est représenté dans l'algorigramme présenté ci-dessous.

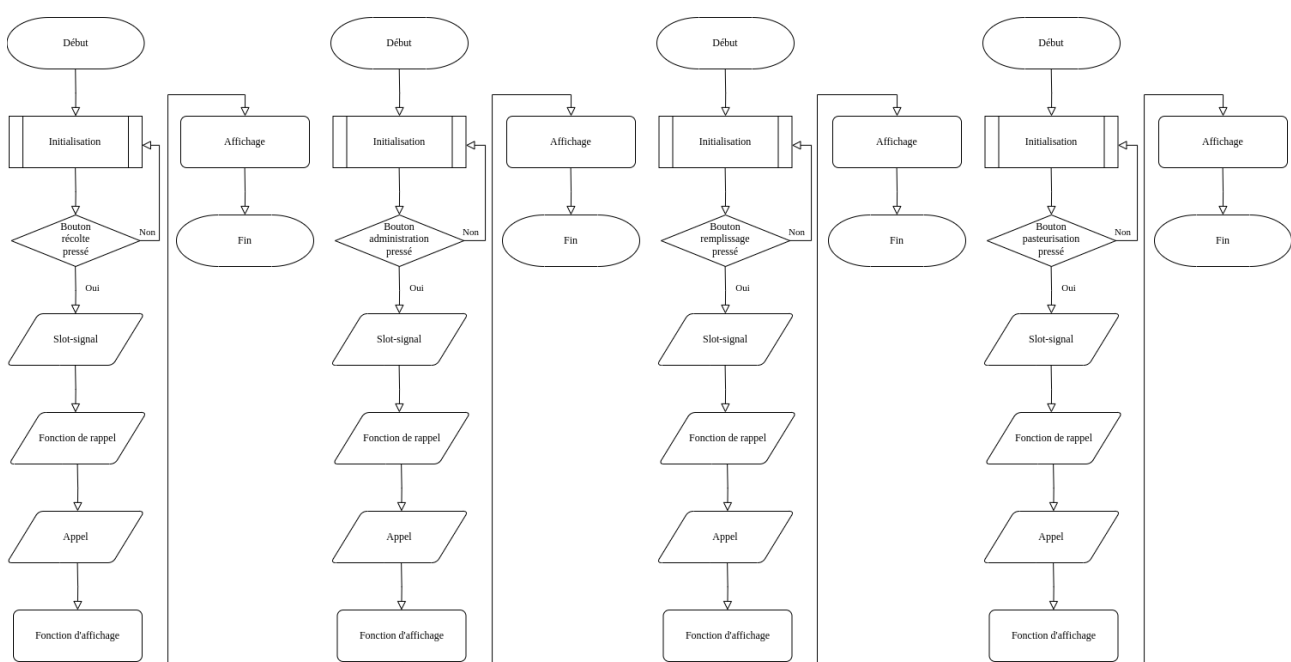


Fig. 6: Algorigrammes de l'interface principale

Grâce à des boutons présents sur ce menu principal, l'utilisateur peut accéder à des interfaces secondaires ayant pour gestion la base de données ou encore affichage les données de production.

3.3.2. Modélisation comportementale de l'interface de récolte

Afin de déclarer en base de données les apports de sacs, le récoltant peut utiliser les boutons disposés sur cette interface. Ainsi, cette interface propose différentes options de manipulation :

- Sélection de l'identité du récoltant à partir d'une [liste déroulante](#)
- Sélection du nombre de sacs apporté en utilisant un [compteur de quantité](#)
- Choix de la variété possible avec une [liste déroulante](#)
- Impression de l'étiquette grâce à [bouton simple](#)
- Validation de la déclaration de la récolte après clic sur un [bouton simple](#)

L'algorithme ci-présent illustre le comportement de cette interface en fonction des décisions prises par l'utilisateur.

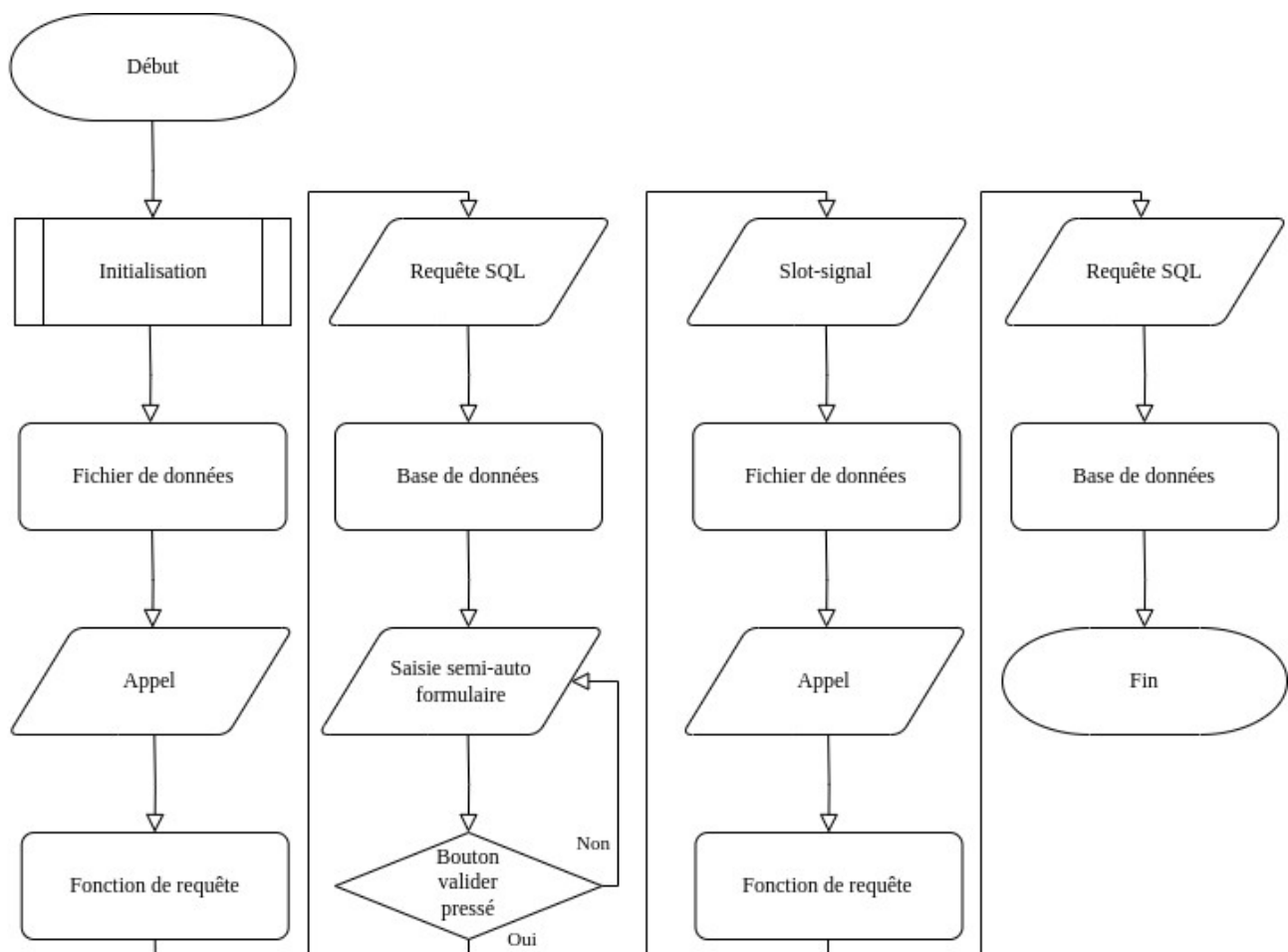


Fig. 7 : Algorithme de l'interface de récolte

Le code source présente une fonction de requête qui produit une requête vers la base de données pour récupérer les informations d'identités existantes. Le résultat de cette requête est renvoyé au bouton, permettant ainsi à l'utilisateur de sélectionner dans la liste déroulante une identité. Le principe pour la sélection de la variété est le même. La sélection du nombre de sacs apportés repose sur une fonction du fichier de données en incrémentant un à chaque fois que la quantité désirée est supérieure et décrémente de un à chaque fois que la quantité désirée est inférieure. La sélection du nombre de sac se fait par clic sur les deux boutons du compteur de quantité. L'ensemble de ces actions est symbolisé par un bloc « entrée/sortie » représenté ci-dessus par un parallélépipède rectangle dont l'intitulé est « saisie semi-auto formulaire ».

3.3.3. Modélisation comportementale de l'interface de remplissage

Lors du remplissage des bouteilles, cette interface dédiée informe le bénévole du nombre de bouteilles qu'il remplit par :

→ Affichage des données de remplissage sur un [afficheur à sept segments](#)

Cette interface tient au courant l'utilisateur du nombre de bouteilles remplies actualisé toutes les trente secondes par le biais d'un affichage à sept segments dont voici le procédé d'affichage.

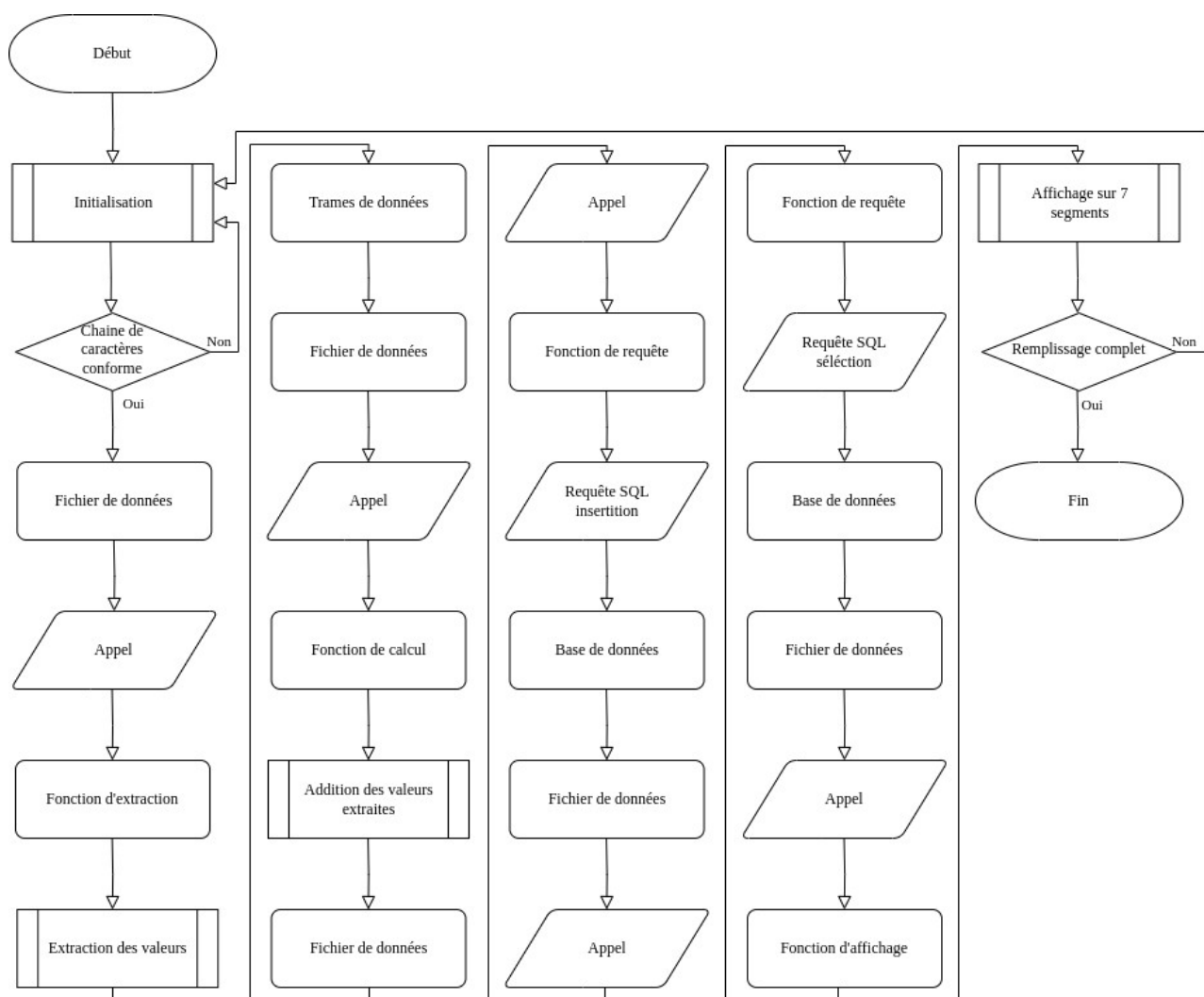


Fig. 8 : Algorithme de l'interface dédiée au remplissage

L'ensemble des sous-processus à l'initialisation du programme concernant la lecture de la trame sur le port série. Toutes les trente secondes, le contenu de la trame envoyé est analysé par une fonction afin d'en déterminer la conformité puis son contenu est extrait. Étant donné que le remplissage se fait avec deux pistolets, les trames lues en port série sont doubles. La somme des bouteilles remplies par chaque pistolet est additionnée puis intégrée en base de données. L'affichage est ensuite reproduit sur un afficheur numérique à 7 segments.

3.3.4. Modélisation comportementale de l'interface de pasteurisation

Concernant l'interface présentant l'avancée de la pasteurisation, la forme appropriée pour relever clairement la progression de l'état de pasteurisation des cinq bain-maries devrait se présenter par :

→ Affichage de la progression de la pasteurisation sur cinq **cadrons** progressifs

Le procédé d'affichage et de mise en marche de ces cadrons est décrit ci-dessous.

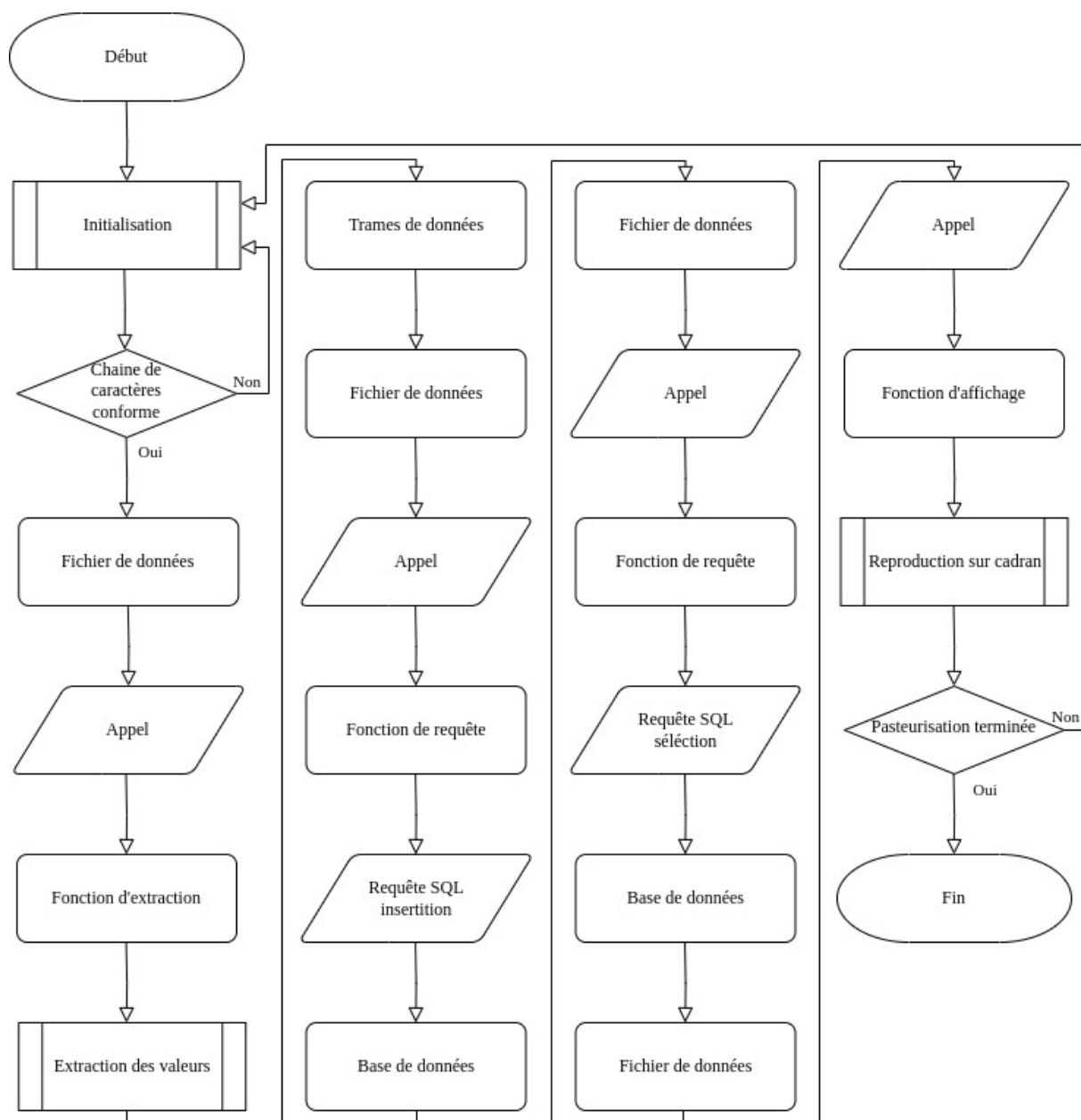


Fig. 9 : Algorithme de l'interface « pasteurisation »

L'algorithme ci-dessus est quasiment identique au précédent. En effet, le programme d'extraction des valeurs est le même, les caractères extraits de la trames sont également envoyés en base de données puis sélectionnés pour être affichés. Néanmoins, l'addition des valeurs n'est d'aucune utilité. Au contraire, cette opération pourrait fausser l'état affiché pour la pasteurisation !

Grâce à cette représentation, nous pouvons maintenant comprendre l'enchaînement des actions effectuées par le programme associé. C'est aussi grâce à la diversité des formes incluses dans la bibliothèque XForms que j'ai pu inclure des formes représentatives d'une pasteurisation progressive.

3.4. Justification des technologies utilisées

Pour bien comprendre le choix des technologies utilisées, il faut se pencher sur les besoins de cette association. En effet, les bénévoles n'ont pas besoin d'environnement informatique très complexe. Nous avons adaptés en conséquence le choix du langage utilisé pour l'implémentation de l'IHM ou encore le système d'exploitation sur lequel repose la bibliothèque du langage utilisé.

Le tout premier cahier des charges impliquait l'utilisation de technologies web telles que HTML, CSS ou encore JS. L'ensemble du système devait donc être ce que l'on qualifie de « client léger », une application interagissant avec un serveur ici implanté en local. Ne nécessitant peu de ressources, la majeure partie des processus devaient être gérées par le serveur, plus précisément par le fichier situé en back-end.

Après mûre réflexion, seule suffit une interface utilisateur en local, que l'on qualifie de « client lourd ». Le système embarqué comprenant une interface utilisateur riche en options et ne comprend plus qu'un fichier de données pleinement traité par l'ordinateur local.

Avec les étudiants du groupe « IR » nous avons donc décidé d'appliquer ces changements au cahier des charges car l'utilisation d'un « client lourd » plutôt qu'un « client léger » constitue un gain en terme d'étalement de l'utilisation des ressources mémoires.

En ce qui concerne le langage choisi, le plus simple a été d'utiliser un toolkit utilisant le langage C, c'est-à-dire la bibliothèque XForms, idéale pour de petites interfaces simples d'utilisation.

4. Étude détaillée de l'interface homme-machine

Ce chapitre a pour objectif d'introduire les éléments de l'IHM permettant à l'utilisateur d'interagir avec les différentes interfaces qui se trouvent à disposition. En première section, nous verrons l'ensemble des éléments qui permettent à l'utilisateur d'interagir avec des éléments interactifs puis en seconde section nous verrons les éléments non-interactifs de l'interface et seulement informatifs pour lesquels l'utilisateur ne peut modifier le comportement.

4.1. Introduction aux contrôles d'interfaces utilisateur

Certains éléments de l'IHM permettent effectivement de naviguer entre interfaces comme ceux utilisés sur l'interface principale. D'autres sont utiles aux services proposés. Dans cette section, nous allons voir les principaux types de formes employés sur cette interface.

4.1.1. Conception détaillée du bouton standard

Le premier élément de contrôle d'interface à noter correspond au bouton « normal ». Ici, l'interface en comprend quatre.

La maquette du type de bouton utilisé est présentée ci-dessous.

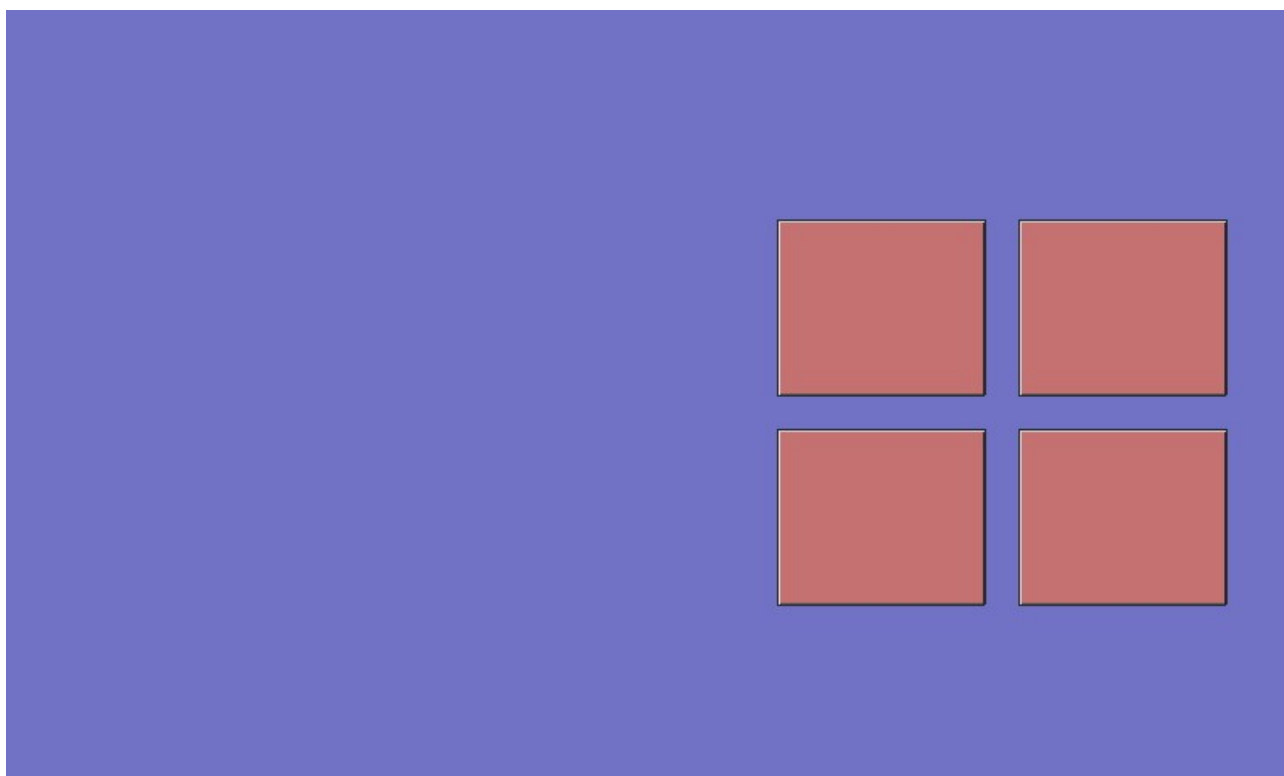


Fig. 10 : Maquette des boutons « simples »

Ces quatre boutons simples appelés `FL_NORMAL_BUTTON` dans la bibliothèque XForms permettent, selon l'action prédéfinie, d'interagir avec l'IHM.

Le clic est détecté et appelle une fonction de rappel déclenchant l'affichage d'une nouvelle fenêtre. La fonction de rappel est étudiée dans le chapitre suivant.

D'autres boutons plus complexes permettent des actions plus approfondies.

4.1.1. Conception détaillée du bouton de liste déroulante

La bibliothèque XForms inclut des boutons plus adaptés pour sélectionner des informations. Le type de bouton dont nous allons parler présente un menu déroulant lorsqu'il est cliqué et offre plusieurs avantages.

Les boutons de menu déroulants sont présentés dans la maquette en figure 11.

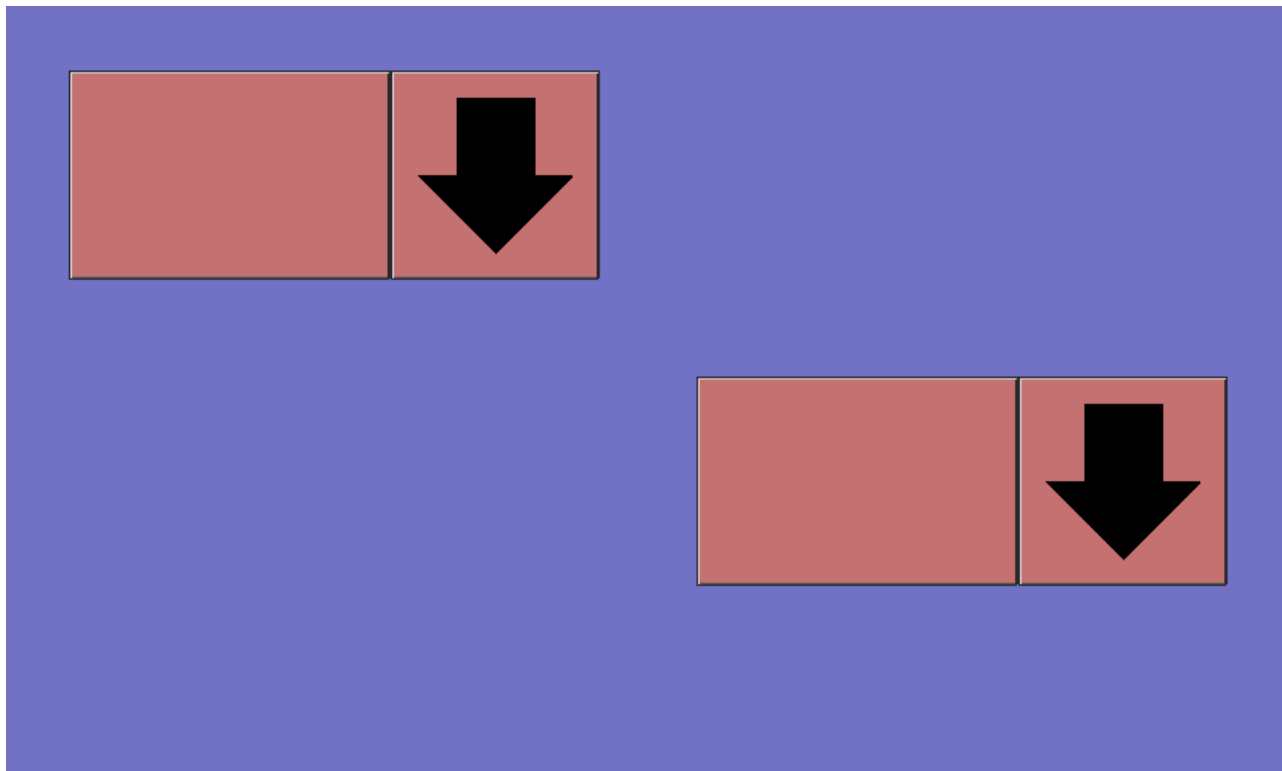


Fig. 11 : Maquette des boutons de menu déroulant

Non seulement le bouton FL_DROP_LIST_CHOICE permet d'économiser de l'espace en ne montrant les options possibles à la verticale mais il rend aussi la navigation plus fluide et plus rapide. Les options sont organisées et encombrement donc moins l'interface, ce qui constitue un choix de bouton idéal dans le cas de dépôts de récoltes rapides et « sans prise de tête » pour l'utilisateur. D'autres boutons plus simples nécessiteraient plus d'espace.

4.1.1. Conception détaillée du bouton de sélection

L'interface est également accompagnée d'un bouton pour l'inventaire du nombre de sacs apporté après la récolte.

La maquette correspondante au type de bouton envisagé est présentée ci-dessous.

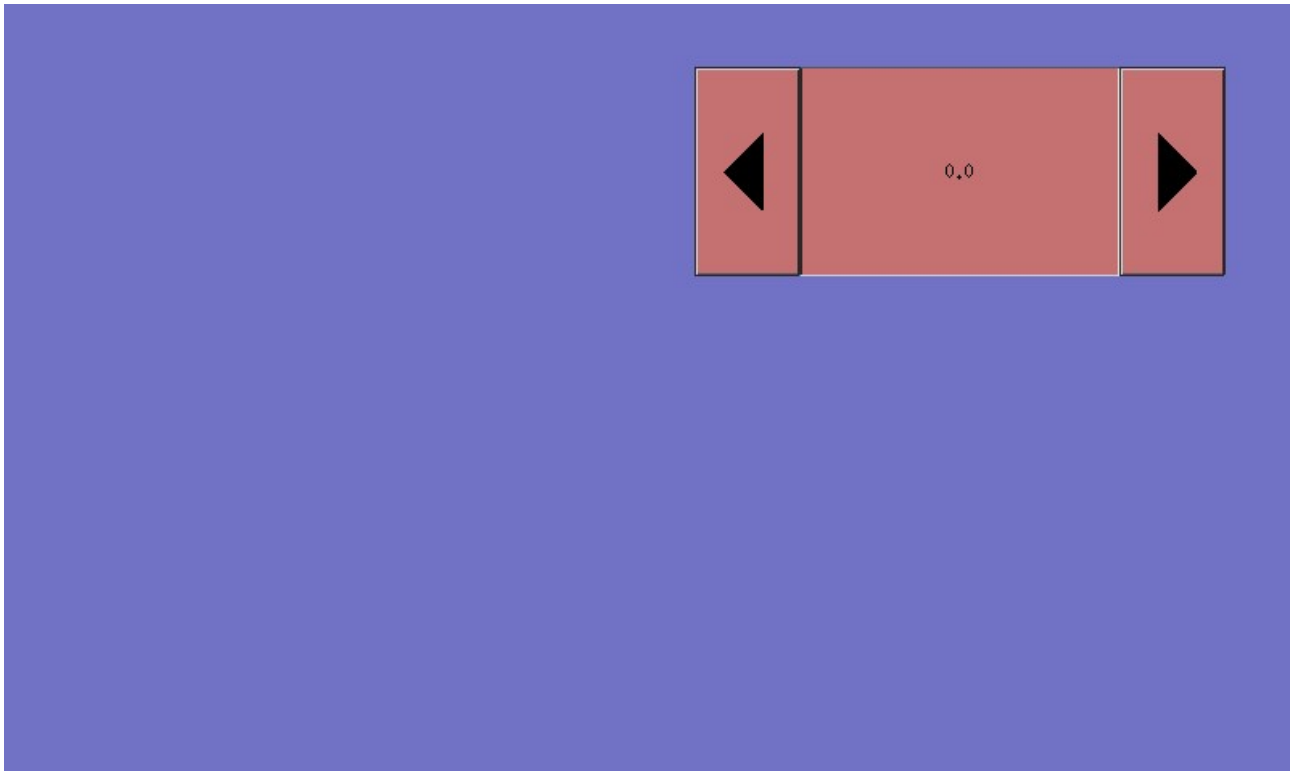


Fig. 12 : *Maquette du bouton de sélection*

Avec sa forme ergonomique, le bouton FL_SIMPLE_COUNTER est un choix tout à fait adapté dans de telles circonstances. En utilisant les boutons de ce compteur, l'utilisateur peut ajuster la quantité de sacs nécessaire tout en gardant un œil sur la quantité enregistrée, ce qui constitue un gain de temps pour le récoltant.

Néanmoins, d'autres éléments nécessaires sont à insérer dans l'interface afin que le récoltant puisse ajouter un nouvel utilisateur.

4.1.1. Conception détaillée de la zone de saisie

La manipulation de la base de données rendue possible grâce à l'élément de zone de saisie demeure être la seule option pour que l'utilisateur insère directement un texte personnalisé.

La maquette de cette zone de saisie est présentée en figure 13 ci-après.

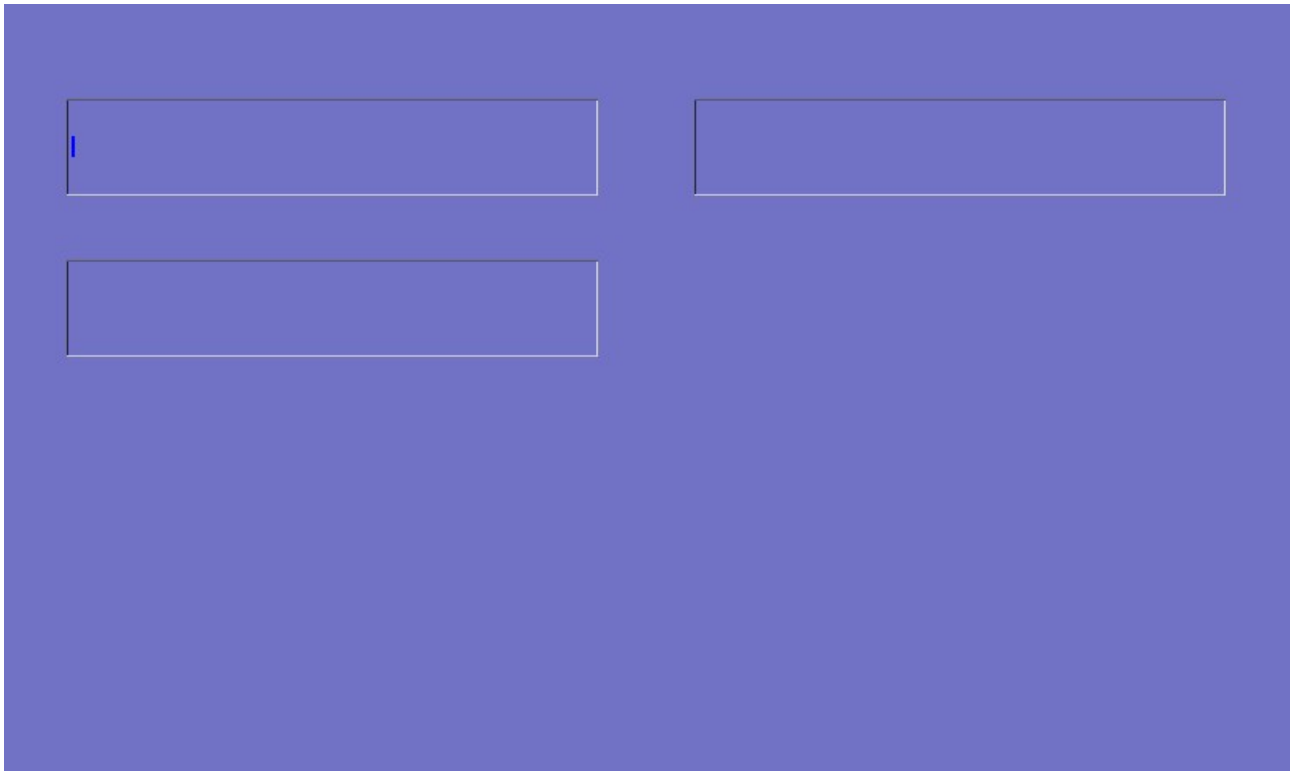


Fig. 13 : *Maquette des formes « zones de saisies »*

Bien que l'interaction avec ces FL_NORMAL_INPUT plus communément appelés « zones de saisie » ne soit pas « confortable » pour l'utilisateur, elle reste une option intéressante quant à l'envoi d'éléments personnalisés en base de donnée. Cette méthode dite « plus naturelle » pour l'utilisateur lorsque celui-ci utilise un ordinateur reste accessible car l'ajout d'un plug-in pour clavier virtuel est envisageable. Ainsi, l'utilisation d'un clavier physique est à dépourvoir car l'entièreté de ces interactions doit rester interne à l'IHM.

L'envoi de données choisies par l'utilisateur n'est pas le seul élément personnalisable. La bibliothèque XForms offre par défaut de nombreuses formes mais il n'est pas pour autant interdit d'en innover d'autres.

4.2. Introduction aux formes non-interactives de l'interface utilisateur

Bien que la bibliothèque de XForms détienne une multitude de formes à portée de l'utilisateur, certaines sont pour le moins singulières. Notre but en tant qu'étudiant est, complémentirement, de vendre l'interface produite. Nous avons dû faire appel au Dr. Plassart, développeur-auteur d'une forme d'afficheur 7 segments numérique.

4.2.1. Conception détaillée de l'afficheur à sept segments

Concernant l'interface indiquant le nombre de bouteilles remplies, j'aurai pu me contenter d'un simple affichage en relief.

Par exemple, j'aurai pu utiliser une forme FL_UP_BOX, encadrant un nombre qui évoluerait toutes les trente secondes comme présenté ci-dessous.



Fig. 14 : *Maquette de l’affichage en relief*

Esthétiquement, ce type de forme quant à l’affichage du nombre de bouteilles remplies est rudimentaire. Une alternative est donc plus appropriée dans ce contexte. Ainsi, le choix d’utilisation d’un affichage plus sophistiqué s’est révélé plus intéressant.



Fig. 15 : *Maquette de l’afficheur 7 segments*

Ce type de forme n'est pas issu directement de la bibliothèque XForms. Il est cependant le fruit d'un ensemble de fonction de cette bibliothèque donnant lieu à une forme appelée FL_NORMAL_SEGDISPLAY. La clarté de cette forme permet de mettre en avant le remplissage. Aussi, la taille de l'affichage de chaque ensemble de segments constituant un chiffre entre 0 et 9 s'adapte à la distance qui sépare l'utilisateur de l'écran.

4.1.1. Conception détaillée du cadran progressif

Le cadran progressif s'est avéré être une solution adéquate dans le cas d'utilisation de voyants pour la pasteurisation.

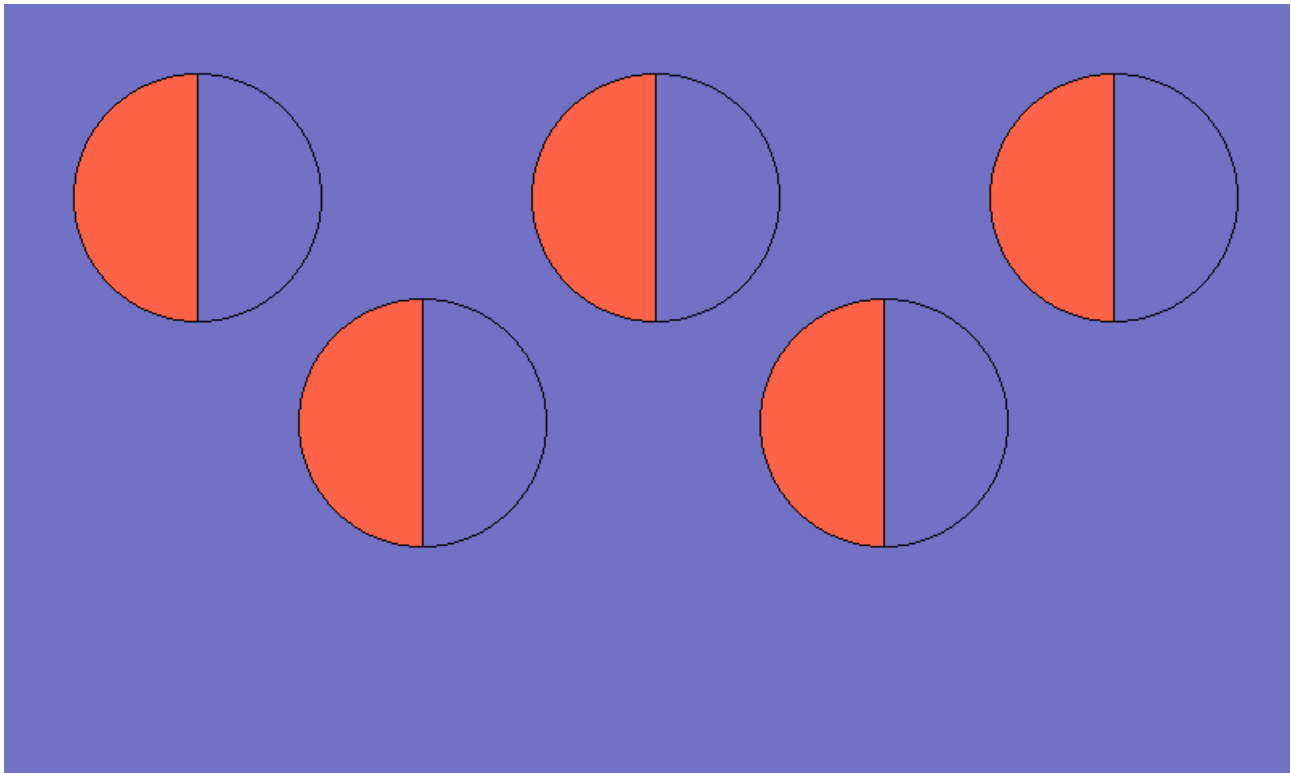


Fig.16 : Maquette des cadrans progressifs

Bien que la bibliothèque XForms offre diverses formes graphiques, elle n'assure cependant pas de multiples formes dont l'utilisation est similaire aux cadrans disposées ci-dessus.

La clarté visuelle de ces formes permet à l'utilisateur d'interpréter l'avancée du processus de pasteurisation pour chaque cuve à bain-marie. Accessoirement, l'espace offert par l'interface n'est pas surchargé ce qui permet d'identifier clairement à distance l'état de pasteurisation de chaque cuve.

5. Réalisation du code applicatif

Ce chapitre souligne les parties importantes relevées dans l'interface utilisateur ainsi que celles du script incluant des fonctions spécifiques que nous allons étudier au travers des sections ci-après.

5.1. Implémentation de matrices de pixels dans le code source

L'utilisation d'image sur certain boutons de l'interface présente plusieurs avantages pour l'utilisateur. Ces images permettent une reconnaissance visuelle rapide des fonctionnalités proposées. Aussi, le fait d'utiliser une image plutôt qu'un texte est une façon d'inscrire un langage universel dans l'interface en la rendant plus attrayante.

Les images utilisées sont des fichiers pixmap. Ce type de fichier contient un tableau que l'on appelle « matrice » dans lequel chaque pixel contient des informations de couleurs.

Dans ce fichier, la variable « pommes » est pointée par un tableau char contenant des chaînes de caractères, désignant les informations de l'image et de type static car sa durée de vie est indéterminée. La variable est donc active pendant toute la durée de l'exécution de l'image.

Après la définition du format de l'image, la ligne "450 374 256 2 " permet de définir la largeur, la hauteur, le nombre de couleurs utilisées ainsi que la profondeur de l'image.

Un extrait de ce fichier pixmap est disponible en annexe.

La figure 17 ci-dessous constitue la version finale de l'interface principale de l'IHM.

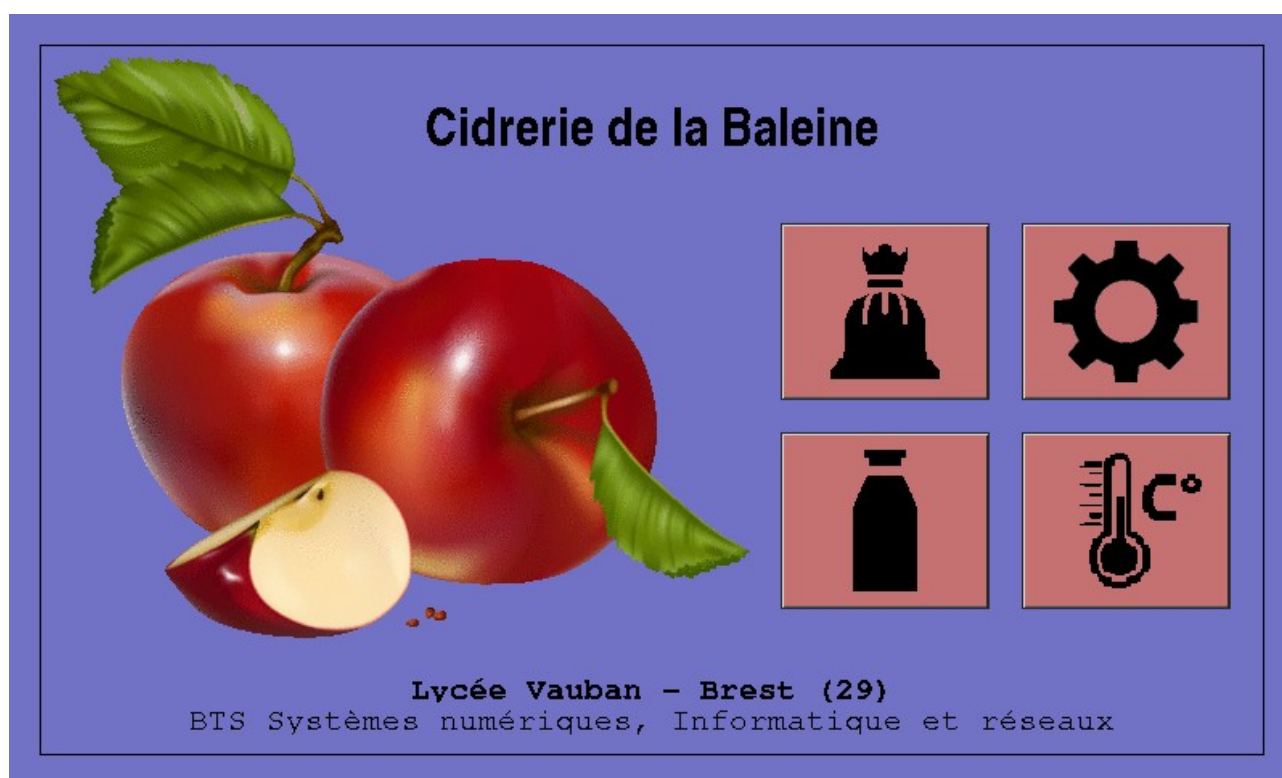


Fig.17 : Maquette de l'interface principale

Afin que la fonction `fl_set_pixmap_data` puisse utiliser puis afficher les données de l'image pixmap, il faut déclarer l'image pixmap dans l'en-tête du code source de l'interface comme suit : `#include "pommes.xpm"`.

La variable `obj` contenant la référence de l'objet pixmap créer un `FL_NORMAL_BITMAP` aux coordonnées cartésiennes de 500 pixels sur l'axe x et de 140 pixels sur l'axe y. Les dimensions de cette objets sont initialisées à une largeur de 90 pixels et une hauteur de 90 pixels ce qui correspond finalement à la taille de l'image. Le pointeur `NULL` est initialisé à 0 et pointe vers les données du pixmap bag en indiquant que celles-ci sont définies ultérieurement dans la fonction `fl_set_pixmap_data()`.

Les trois autres boutons de l'interface principale ainsi que l'image de pommes disposent également d'images pixmap permettant d'accéder aux différentes interfaces grâce à un type de fonction que je vous propose d'étudier dans la section suivante.

5.2. Utilisation de la fonction de rappel

La fonction de rappel ou *callback function* est une fonction passée en argument d'une autre fonction. La fonction de rappel est « appelée » et permet d'exécuter une autre partie du programme dans un enchaînement défini.

Par exemple, lorsque l'utilisateur interagit avec le premier `FL_NORMAL_BUTTON`, la fonction `fl_set_object_callback()` détecte le clique et déclenche l'initialisation de la fonction de rappel `cb_btn_recolte()` qu'elle prend en argument.

Le résultat de l'exécution de cette fonction de rappel est présenté ci-dessous.

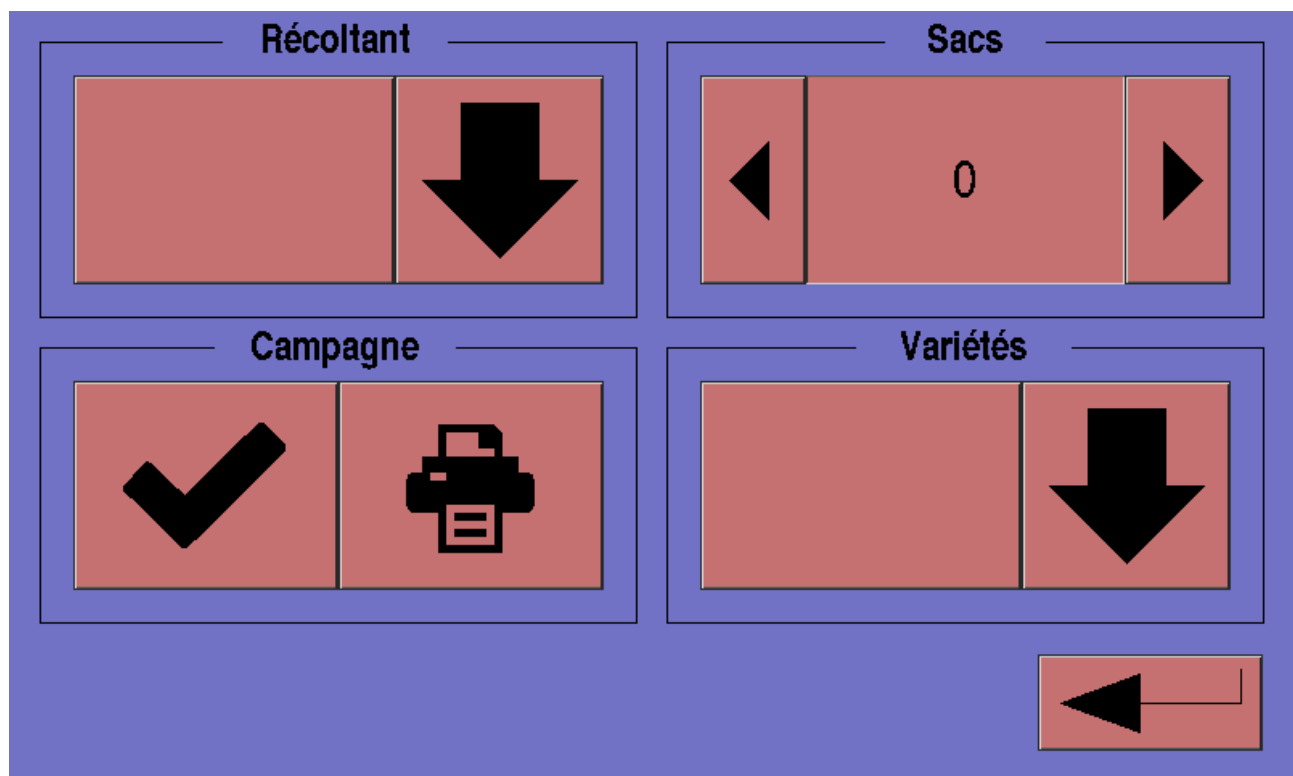


Fig.18 : Maquette de l'interface de récolte

Une fois appelée, la fonction de rappel `cb_btn_recolte()` créer une nouvelle fenêtre avec la fonction `fl_bgn_form()` dans laquelle une subform prend le type de forme

d'une FL_FLAT_BOX sur 800 pixels de longueur et 480 pixels de hauteur. Une fonction d'initialisation de la couleur de fond applique ensuite une couleur FL_SLATEBLUE prédéfinie dans la bibliothèque forms.h en guise d'arrière plan. L'implémentation des autres formes de cette interface est présentée dans le code source en annexe.

5.3. Utilisation de la fonction de mise à jour asynchrone

L'interface de remplissage est également générée par une fonction de rappel, ici nommée `cb_display_timer`. En effet, la fonction `getNombreTotalBouteilles()` récupère en amont le nombre total de bouteille qu'elle stocke dans une variable `nombre`. C'est ensuite que la fonction de rappel `cb_display_timer()` entre en jeu. Celle-ci récupère la valeur stockée dans la variable `nombre` et met à jour l'afficheur à 7 segments. Cette fonction de rappel utilise une boucle `for` dans laquelle, lors de chaque itération, elle extrait le chiffre des unités du nombre total de bouteilles en utilisant l'opérateur de reste de division euclidienne « % » puis la fonction `fl_set_segdisplay` de la bibliothèque `display.h` présentée en annexe, adapte la valeur de la variable `nombre` pour chaque tableau `display[i]` où chaque $i^{\text{ème}}$ de 7 segments prend la forme d'un chiffre.

À la fin de la boucle, la valeur stockée dans la variable `nombre` est divisée par 10 puis réattribuée à elle-même en réduisant sa valeur. Au sortir de la boucle `for`, on observe un décalage progressif de groupe de 7 segments en groupe de 7 segment vers la gauche. Ce type d'opération est appelé opération d'affectation combinée.

Ainsi, on obtient l'affichage présenté en figure 19.

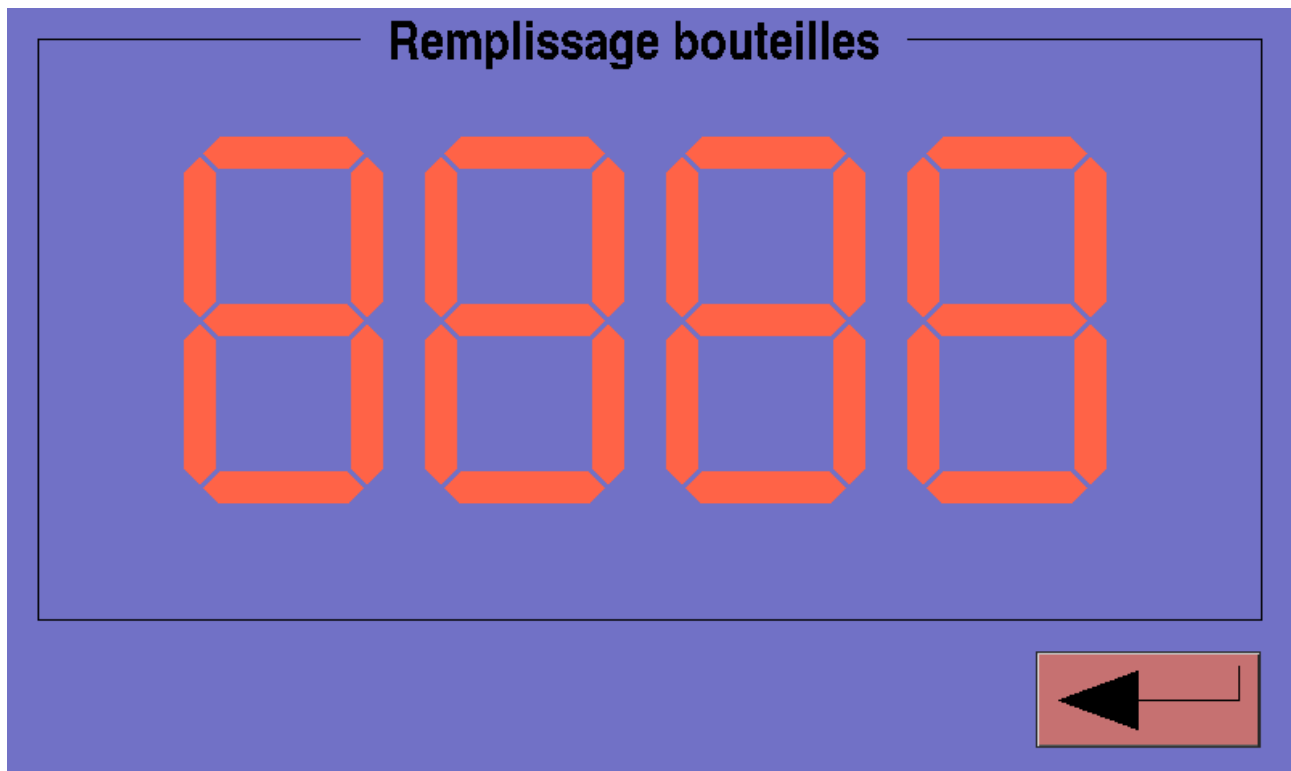


Fig.19 : Maquette de l'interface de remplissage

Après extraction des valeurs de température dans la chaîne de caractère puis conversion en flottant, les valeurs sont assignées au tableau `sensorValues[]`. La fonction `processTrame()` appelle la fonction de rappel `cb_dial_timer()` en fin d'exécution afin de mettre à jour la progression des cadrans.

Tout d'abord, la limite de température de pasteurisation est établie à 76°C où la variable `max_temperature` correspond à la température maximale attendue. La valeur maximale des cadrans correspond à une rotation complète de 360°, assignée à la variable `max_dial_value`. Ensuite, chaque valeur de capteur est normalisée par rapport à la variable `max_temperature` puis mise à l'échelle pour correspondre à une plage de 0 à 360° pour chaque cadran.

Ainsi, les tableaux `sensorValues[i]` comprenant les valeurs des capteurs sont divisés par la variable `max_temperature` puis on obtient le ratio. Le ratio est ensuite multiplié par la variable `max_dial_value` afin d'obtenir la valeur correspondante au cadran. Les variables `dial_value1`, `dial_value2`, `dial_value3`, `dial_value4` et `dial_value5` prennent successivement le résultat de ces opérations.

La fonction `fl_set_dial_value()` met à jour les valeurs des cadrans dans l'interface utilisateur dont le résultat d'exécution est présenté dans la figure 19 ci-dessous.

La personnalisation de ces cadrans ainsi que l'implémentation des autres formes et textes de cette interface est notable dans l'annexe correspondante.

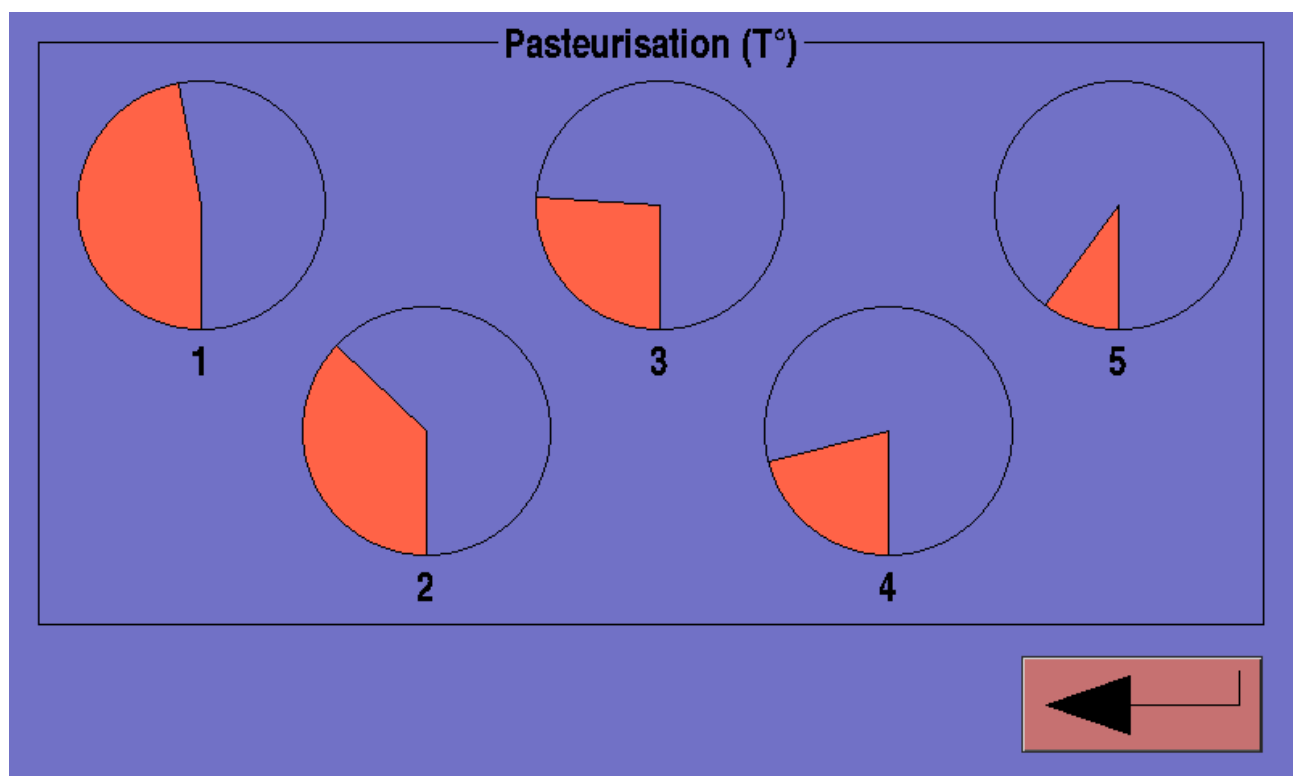


Fig.20 : Maquette de l'interface de pasteurisation

Chaque cadran est construit avec la fonction `fl_add_dial()` et bénéficie de ses propres coordonnées cartésiennes `x` et `y`. Les `dial1`, `dial2`, `dial3`, `dial4` et `dial5` prennent ensuite la forme de plusieurs `FL_FILL_DIAL`.

Néanmoins, la progression des cadrans n'est mise à jour que lors de l'initialisation et la réception des 5 premières valeurs. L'utilisateur est donc contraint, afin d'actualiser la fenêtre, de sortir de cette interface puis d'y revenir.

Ainsi, pour palier au problème de rafraîchissement de cette interface, l'utilisation d'un *timer* s'avère être une solution efficace.

La fonction `fl_add_timer` ajoute dans un premier temps un *timer* de type `FL_HIDDEN_TIMER` puis le résultat de cette dernière est stocké dans la variable `dial_timer`. La fonction de rappel `fl_set_object_callback` associe une *callback* au *timer* qu'elle appelle à chaque fois qu'il expire.

En conséquence, lors de chaque appel de la fonction de rappel `cb_dial_timer`, le *timer* est redémarré puis il expire après un délai `DELAY` de 0.5 secondes défini en en-tête du programme.

Les cadrans sont ajoutés avec la fonction `fl_add_dial()` qui spécifie le type de cadran utilisé : `FL_FILL_DIAL`. Les caractéristiques esthétiques de ces formes ainsi que les coordonnées de dimensions et de position sont détaillées dans le code source en annexe.

6. Tests unitaires

Ce chapitre présente les différentes phases de tests des contrôles d'interface utilisateur tel que les types de boutons abordés en conception détaillée. Ces phases de tests permettent de vérifier le bon fonctionnement de l'association des différents types de boutons aux fonctions de rappel ainsi que la bonne

6.1. Vérification des retours console des contrôles d'interface utilisateur

Afin de vérifier le bon fonctionnement de la fonction de rappel associée au bouton d'administration de la base de donnée, j'ai utilisé la fonction `printf()` pour générer un message de débogage lors du clic sur le bouton.

Le test effectué est montré en figure 21.

```
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$ gcc -Wall test.c -o test -lforms -lX11
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$ ./test
Bouton administration appuyé !
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$
```

Fig.21 : Affichage du message de débogage du bouton simple sur la console

Cet outil de débogage permet de confirmer que la fonction de rappel est exécutée lorsqu'un clic sur le bouton est détecté.

Je précise également le retour console de la chaîne de caractères générée par la fonction de sortie sur le flux standard.

```
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$ gcc -Wall test.c -o test -lforms -lX11
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$ ./test
Quantité sélectionnée : 1
Quantité sélectionnée : 2
Quantité sélectionnée : 3
Quantité sélectionnée : 4
Quantité sélectionnée : 5
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$
```

Fig.22 : Affichage du message de débogage du bouton de sélection sur la console

Le message inscrit sur la console en figure 22 reprend simplement l'incréméntation de 1 en 1 correspondant à une addition du nombre de pression sur le bouton.

La fonction de rappel associée à ce bouton de sélection est rattaché à une fonction de requête programmant l'envoi ainsi que la réception d'informations sql.

```
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$ gcc -Wall test.c -o test -lforms -lX11
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$ ./test
Récoltant sélectionné : Récoltant n°1
Récoltant sélectionné : Récoltant n°2
Récoltant sélectionné : Récoltant n°3
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$
```

Fig.23 : Affichage du message de débogage du bouton de liste déroulante sur la console

Une fois que le bouton est pressé et que la fonction de rappel est déclenchée, la fonction `printf()` renvoie le résultat du clic d'une option de la liste déroulante.

6.1. Vérification de la réception des données de production sur sortie standard

Aussi, afin de vérifier que les données de remplissage sont bien enregistrées dans le tableau `pistolValues[]`, j'effectue un test avec la fonction `printf()` ci-dessous.

```
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$ ./test
Nombre de bouteilles remplies par le pistolet 1 : 9
Nombre de bouteilles remplies par le pistolet 2 : 0
Nombre de bouteilles remplies par le pistolet 1 : 9
Nombre de bouteilles remplies par le pistolet 2 : 0
^C
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$
```

Fig.24 : Affichage du message de débogage de la réception du nombre de bouteilles remplies pour chaque pistolet sur la console

Ce test montre que les données de remplissage sont bien enregistrées. Elles sont ensuite stockées dans un tableau `pistolValues[]` que je renvoie sur la console.

J'effectue des tests de bonne réception des valeurs de pasteurisation sur le port série. Ces données sont également affichées sur le flux de sortie en figure 25 ci-dessous.

```
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$ ./test
Valeurs des capteurs stockées dans sensorValues[i] : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs converties en degrés pour les cadrans : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs des capteurs stockées dans sensorValues[i] : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs converties en degrés pour les cadrans : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs des capteurs stockées dans sensorValues[i] : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs converties en degrés pour les cadrans : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs des capteurs stockées dans sensorValues[i] : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs converties en degrés pour les cadrans : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs des capteurs stockées dans sensorValues[i] : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs converties en degrés pour les cadrans : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs des capteurs stockées dans sensorValues[i] : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs converties en degrés pour les cadrans : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs des capteurs stockées dans sensorValues[i] : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs converties en degrés pour les cadrans : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs des capteurs stockées dans sensorValues[i] : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs converties en degrés pour les cadrans : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs des capteurs stockées dans sensorValues[i] : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs converties en degrés pour les cadrans : 0.000000, 0.000000, 0.000000, 0.000000, 0.000000
Valeurs des capteurs stockées dans sensorValues[i] : 26.500000, 56.400000, 0.000000, 0.000000, 0.000000
Valeurs converties en degrés pour les cadrans : 125.526316, 267.157895, 0.000000, 0.000000, 0.000000
Valeurs des capteurs stockées dans sensorValues[i] : 26.500000, 56.400000, 0.000000, 0.000000, 0.000000
Valeurs converties en degrés pour les cadrans : 125.526316, 267.157895, 0.000000, 0.000000, 0.000000
^C
c32jp@c32jp-NBR-WAX9:~/Bureau/e6_2/tests_unitaires$
```

Fig.25 : Affichage du message de débogage de la réception de la température stockées chaque pistolet sur la console

On constate que la fonction `printf()` affiche correctement les valeurs des capteurs stockées dans le tableau `sensorValues()`. On constate également que ces valeurs de températures ont été correctement converties en degrés afin de correspondre aux positions de progression par rapport aux cadrans de 360°.

7. Intégration de la base de données à l'interface homme-machine

Ce chapitre présente les interactions possibles entre l'interface de récolte et la base de données ainsi que les 2 interfaces concernées par le remplissage et la pasteurisation avec l'utilisation de requêtes SQL.

L'interface de déclaration de récolte comprend une zone de saisie. Lorsqu'une chaîne de caractère est saisie en entrée d'une `FL_NORMAL_INPUT` et que celle-ci est validée, une *callback function* est déclenchée au clic pour ainsi initialiser la construction d'une requête SQL d'insertion en base de données.

Une requête d'insertion est également construite lors de la réception des trames de données de remplissage et de pasteurisation. En effet, la trame est d'abord traitée en application dorsale gérée par un autre étudiant du groupe « IR » puis capturée par une fonction qui la stocke dans une variable. Cette variable est ensuite exploitée par une fonction de requête SQL qui insère ces données de production en base de données.

L'interface d'administration de la base de données et plus particulièrement l'interface permettant d'afficher les statistiques de déclaration de récoltes de pommes initialise également une fonction d'appel à une *callback function*. Cette dernière génère une requête de sélection SQL dans la base de données afin d'afficher toutes les informations concernant un récoltant, à savoir : la date de déclaration de dépôt de pommes, le nombre de sacs apporté ainsi que la variété.

8. Conclusion

Ce dernier chapitre clôture le développement de ma partie du projet m'étant attribuée et permet de conclure sur la fiabilité de l'IHM et des fonctions du code source concernant les interfaces de production. Des perspectives d'améliorations sont précisées en seconde section.

8.1. Bilan

Le cahier des charges spécifie la mise en place d'une IHM implémentée en langage C que nous avons étudié au travers de maquettes et dont nous avons étudié les principales fonctions.

L'ensemble des interfaces est maintenant intégré dans l'environnement cible et est à disposition de la base de données ainsi que de l'application dorsale chargée de traiter les données transmises par trames.

Les spécifications du système, fixées en amont, permettent dorénavant de répondre aux besoins des utilisateurs tout en respectant les contraintes techniques et contextuelles annoncées en problématique.

6.1. Perspectives d'améliorations

Plusieurs améliorations sont notables concernant l'interface utilisateur d'administration de la base de données. En effet, celle-ci peut être mieux agencée afin de mieux guider l'utilisateur dans sa navigation.

Actuellement, cette interface présente un espace d'ajout de nouveau récoltant et de variété qui n'est pas sur la même page de désactivation de récoltants.

Aussi, l'interface de statistiques peut être implémentée indépendamment de l'interface de création de campagne sans devoir faire appel à une fonction de rappel.

Des images de types pixmap peuvent également être appliquées sur les boutons correspondants aux actions de gestion des récoltants, de création de campagne et d'affichage de statistiques.

A.1. Maquettes prototypes de l'interface principale

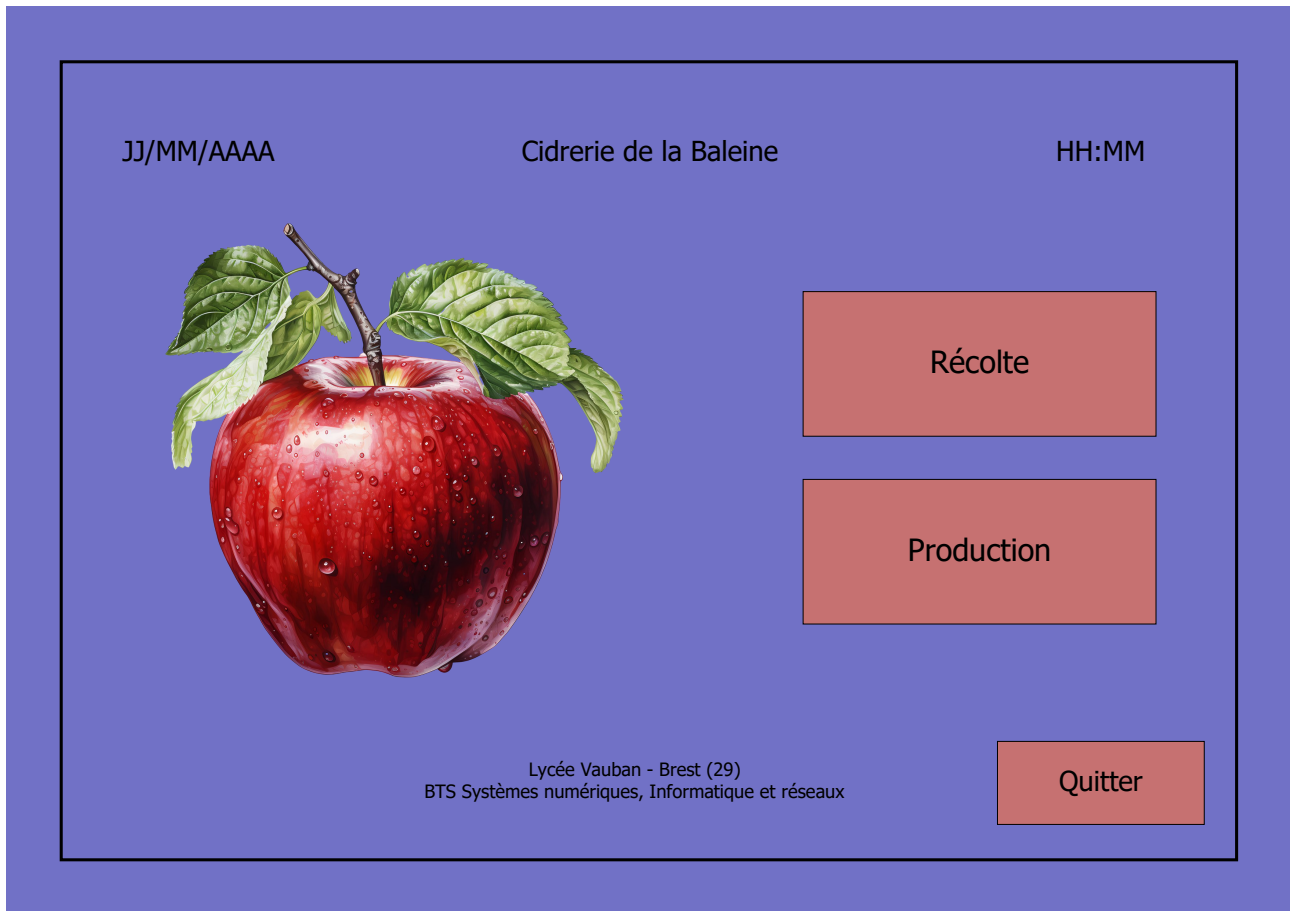


Fig.26 : Maquette du prototype #1 de l'interface principale

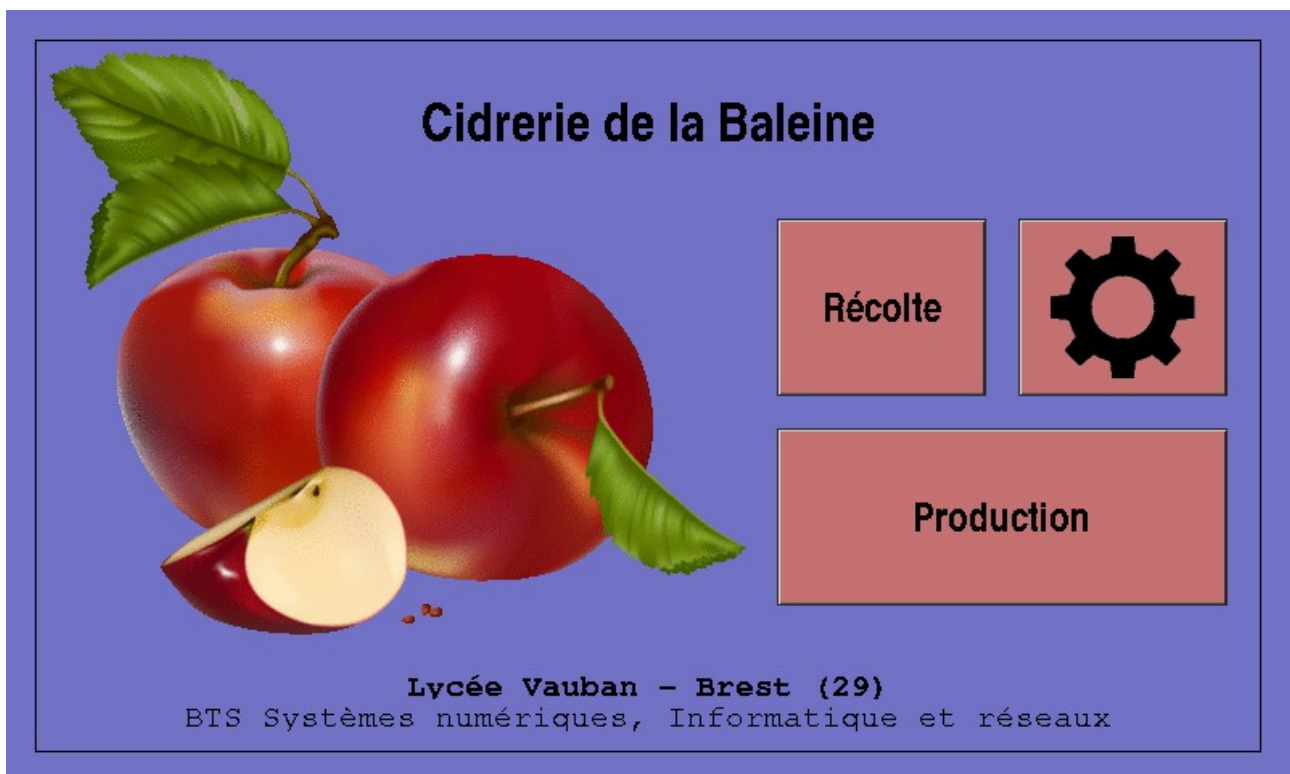



Fig.27 : Maquette du prototype #2 de l'interface principale

A.2. Maquette prototype de l'interface de récolte

JJ/MM/AAAA

Récoltes


HH:MM



Nom :

Prénom :

Lieu de la récolte :

Date de la récolte : 

Nombre de sac(s) :

Imprimer des étiquettes

Valider le formulaire




Fig.28 : Maquette du prototype #1 de l'interface de récolte

A.3. Maquettes prototypes de l'interface de production

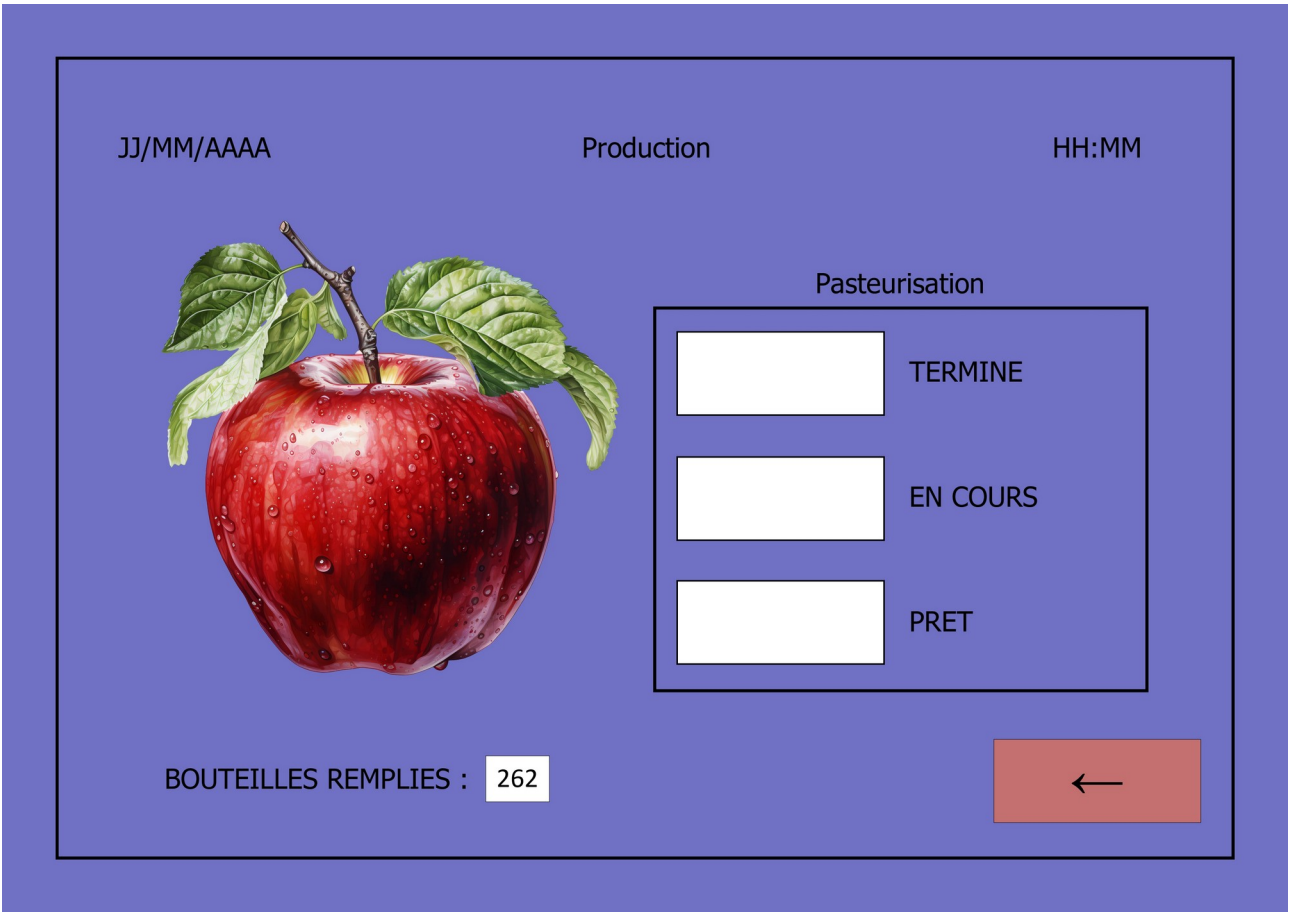


Fig.29 : Maquette du prototype #1 de l'interface de production

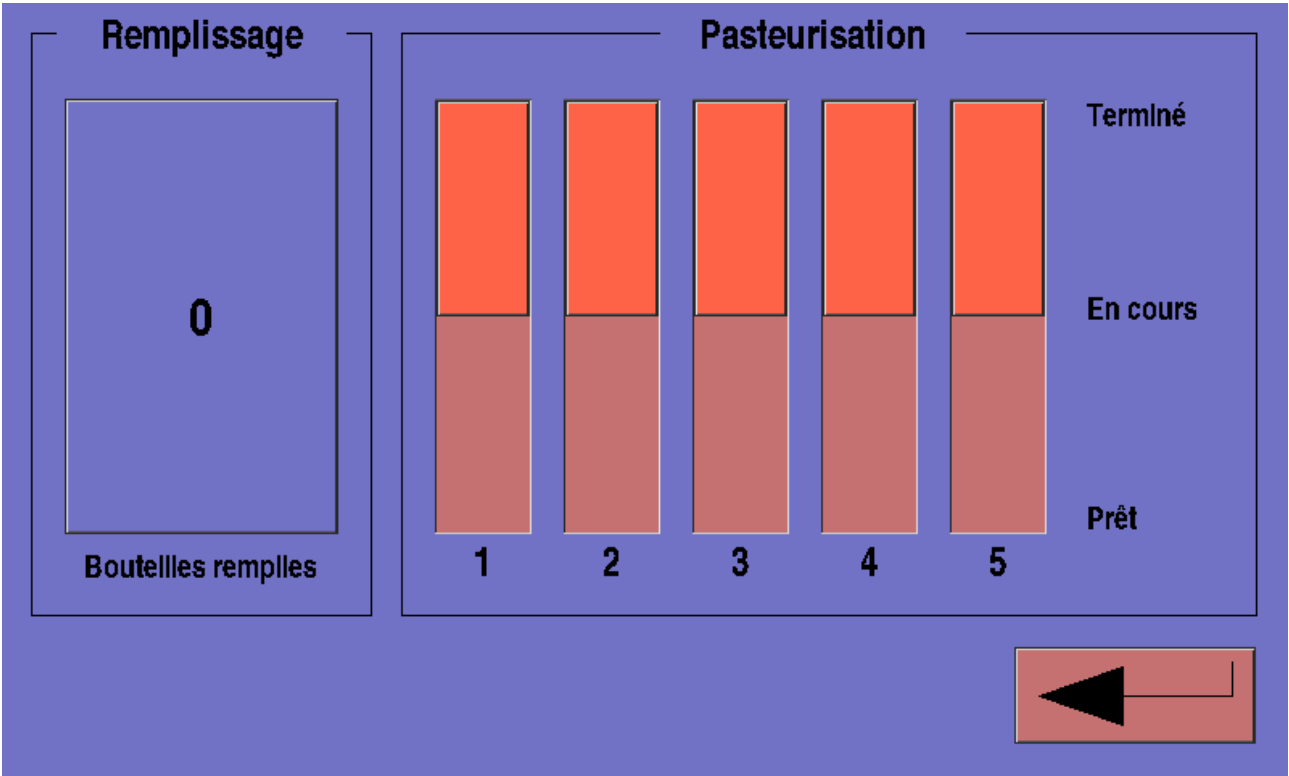
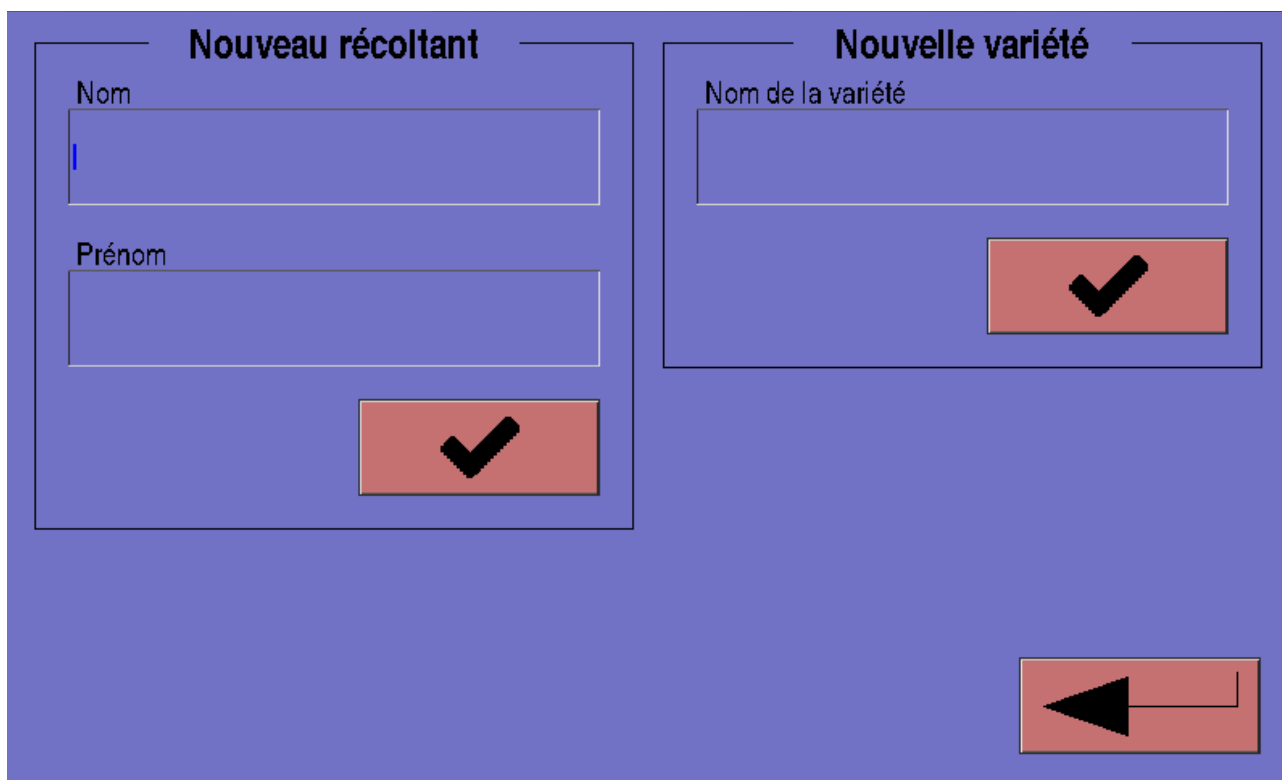


Fig.30 : Maquette du prototype #2 de l'interface de production

A.3. Maquette prototype de l'interface d'administration



The image shows a wireframe of an administrative interface with a blue background. It features two main panels: 'Nouveau récoltant' on the left and 'Nouvelle variété' on the right. The 'Nouveau récoltant' panel contains two text input fields labeled 'Nom' and 'Prénom', and a red button with a black checkmark. The 'Nouvelle variété' panel contains one text input field labeled 'Nom de la variété' and a red button with a black checkmark. A red button with a black left-pointing arrow is located at the bottom right of the interface.

Nouveau récoltant

Nom

Prénom

Nouvelle variété

Nom de la variété

Fig.31 : Maquette du prototype #1 de l'interface d'administration

A.1. Contenu du fichier pixmap « bag.xpm »

[illegible]

.....

.....

.....

.....

• • • •

• • • • • • • • • • • •

A.2. Code source de l'interface principale

```
int main(int argc, char **argv)
{
    fl_initialize(&argc, argv, 0, 0, 0);
    mainform = fl_bgn_form(FL_FLAT_BOX, 800, 480);
    fl_set_form_background_color(mainform, FL_SLATEBLUE);

    fl_add_labelframe(FL_BORDER_FRAME, 20, 20, 760, 440, NULL);
```

```

obj = fl_add_text(FL_NORMAL_TEXT, 100, 20, 600, 100, "Cidrerie de la Baleine");
fl_set_object_color(obj, FL_SLATEBLUE, 0);
fl_set_object_lsize(obj, FL_HUGE_SIZE);
fl_set_object_lstyle(obj, FL_BOLD_STYLE);
fl_set_object_lalign(obj, FL_ALIGN_CENTER);

obj = fl_add_text(FL_NORMAL_TEXT, 50, 410, 700, 20, "Lyc\xE9""e Vauban - Brest (29)");
fl_set_object_color(obj, FL_SLATEBLUE, 0);
fl_set_object_lsize(obj, FL_MEDIUM_SIZE);
fl_set_object_lstyle(obj, FL_FIXEDBOLD_STYLE);
fl_set_object_lalign(obj, FL_ALIGN_CENTER);

obj = fl_add_text(FL_NORMAL_TEXT, 50, 430, 700, 20, "BTS Syst\xE8""mes num\xE9""riques,
Informatique et r\xE9""seaux");
fl_set_object_color(obj, FL_SLATEBLUE, 0);
fl_set_object_lsize(obj, FL_MEDIUM_SIZE);
fl_set_object_lstyle(obj, FL_FIXED_STYLE);
fl_set_object_lalign(obj, FL_ALIGN_CENTER);

obj = fl_add_button(FL_NORMAL_BUTTON, 480, 130, 130, 110, NULL);
fl_set_object_lsize(obj, FL_LARGE_SIZE);
fl_set_object_lstyle(obj, FL_BOLD_STYLE);
fl_set_object_color(obj, FL_INDIANRED, FL_TOMATO);
fl_set_object_callback(obj, cb_btn_recolte, 0);

obj = fl_add_pixmap(FL_NORMAL_BITMAP, 500, 140, 90, 90, NULL);
fl_set_pixmap_data(obj, bag);

obj = fl_add_button(FL_NORMAL_BUTTON, 630, 130, 130, 110, NULL);
fl_set_object_lsize(obj, FL_LARGE_SIZE);
fl_set_object_lstyle(obj, FL_BOLD_STYLE);
fl_set_object_color(obj, FL_INDIANRED, FL_TOMATO);
fl_set_object_callback(obj, cb_btn_roue_crantee, 0);

/* Disposition du pixmap roue crantée au dessus du bouton */
obj = fl_add_pixmap(FL_NORMAL_BITMAP, 640, 140, 110, 90, NULL);
fl_set_pixmap_data(obj, toothed_wheel);

obj = fl_add_button(FL_NORMAL_BUTTON, 630, 260, 130, 110, NULL);
fl_set_object_lsize(obj, FL_LARGE_SIZE);
fl_set_object_lstyle(obj, FL_BOLD_STYLE);
fl_set_object_color(obj, FL_INDIANRED, FL_TOMATO);
fl_set_object_callback(obj, cb_btn_pasteurisation, 0);

obj = fl_add_pixmap(FL_NORMAL_BITMAP, 640, 270, 110, 90, NULL);
fl_set_pixmap_data(obj, thermometer);

obj = fl_add_button(FL_NORMAL_BUTTON, 480, 260, 130, 110, NULL);
fl_set_object_lsize(obj, FL_LARGE_SIZE);
fl_set_object_lstyle(obj, FL_BOLD_STYLE);
fl_set_object_color(obj, FL_INDIANRED, FL_TOMATO);
fl_set_object_callback(obj, cb_btn_remplissage, 0);

obj = fl_add_pixmap(FL_NORMAL_BITMAP, 500, 270, 90, 90, NULL);
fl_set_pixmap_data(obj, bottle);

obj = fl_add_pixmap(FL_NORMAL_BITMAP, 20, 20, 450, 374, NULL);
fl_set_pixmap_data(obj, pommes);

pthread_t thread;
pthread_create(&thread, NULL, communicationThread, NULL);

fl_end_form();
fl_show_form(mainform, FL_PLACE_MOUSE, FL_NOBORDER, NULL); /* FL_NOBORDER */
fl_do_forms();
fl_finish();

return 0;
}

```

A.3. Code source de l'interface de récolte

```

void cb_btn_recolte(FL_OBJECT *obj, long data)
{
    FL_OBJECT *obj_ok3;
    printf("Bouton 'R\xE9""colte' appuy\xE9""e !\n");
    subform = fl_bgn_form(FL_FLAT_BOX, 800, 480);

```

```

fl_set_form_background_color(subform, FL_SLATEBLUE);

fl_add_labelframe(FL_BORDER_FRAME, 20, 20, 370, 170, NULL);

FL_OBJECT *text_recoltant = fl_add_text(FL_NORMAL_TEXT, 133, 0, 140, 34, "R\xE9"coltant");
fl_set_object_color(text_recoltant, FL_SLATEBLUE, 0);
fl_set_object_lsize(text_recoltant, FL_LARGE_SIZE);
fl_set_object_lstyle(text_recoltant, FL_BOLD_STYLE);
fl_set_object_lalign(text_recoltant, FL_ALIGN_CENTER);

droplist_recoltants = fl_add_select(FL_DROPLIST_SELECT, 40, 40, 330, 130, NULL);
fl_set_object_color(droplist_recoltants, FL_INDIANRED, FL_TOMATO);
fl_set_object_callback(droplist_recoltants, cb_focus_recoltant, 0);
remplir_droplist_recoltants(droplist_recoltants);

fl_add_labelframe(FL_BORDER_FRAME, 410, 20, 370, 170, NULL);

obj = fl_add_text(FL_NORMAL_TEXT, 545, 0, 100, 34, "Sacs");
fl_set_object_color(obj, FL_SLATEBLUE, 0);
fl_set_object_lsize(obj, FL_LARGE_SIZE);
fl_set_object_lstyle(obj, FL_BOLD_STYLE);
fl_set_object_lalign(obj, FL_ALIGN_CENTER);

quantity_counter = fl_add_counter(FL_SIMPLE_COUNTER, 430, 40, 330, 130, NULL);
fl_set_object_color(quantity_counter, FL_INDIANRED, FL_BLACK);
fl_set_object_lalign(quantity_counter, FL_ALIGN_LEFT);
fl_set_object_lsize(quantity_counter, FL_HUGE_SIZE);
fl_set_counter_bounds(quantity_counter, 0, 999999);
fl_set_counter_step(quantity_counter, 1, 1);

fl_set_counter_precision(quantity_counter, 0);

fl_set_object_callback(quantity_counter, cb_btn_quantite, 0);

fl_add_labelframe(FL_BORDER_FRAME, 410, 210, 370, 170, NULL);

droplist_varietes = fl_add_select(FL_DROPLIST_CHOICE, 430, 230, 330, 130, NULL);
fl_set_object_color(droplist_varietes, FL_INDIANRED, FL_TOMATO);
fl_set_object_callback(droplist_varietes, cb_focus_varietes, 0);
remplir_droplist_varietes(droplist_varietes);

obj = fl_add_text(FL_NORMAL_TEXT, 527, 191, 134, 34, "Vari\xE9"t\xE9"s");
fl_set_object_color(obj, FL_SLATEBLUE, 0);
fl_set_object_lsize(obj, FL_LARGE_SIZE);
fl_set_object_lstyle(obj, FL_BOLD_STYLE);
fl_set_object_lalign(obj, FL_ALIGN_CENTER);

fl_add_labelframe(FL_BORDER_FRAME, 20, 210, 370, 170, NULL);

obj_ok3 = fl_add_button(FL_NORMAL_BUTTON, 40, 230, 165, 130, NULL);
fl_set_object_lsize(obj_ok3, FL_HUGE_SIZE);
fl_set_object_lstyle(obj_ok3, FL_BOLD_STYLE);
fl_set_object_color(obj_ok3, FL_INDIANRED, FL_TOMATO);
fl_set_object_callback(obj_ok3, cb_btn_ok_depot, 0);

obj_ok3 = fl_add_pixmap(FL_NORMAL_BITMAP, 70, 255, 103, 80, NULL);
fl_set_pixmap_data(obj_ok3, check_recolte);

obj = fl_add_button(FL_NORMAL_BUTTON, 205, 230, 165, 130, NULL);
fl_set_object_lsize(obj, FL_LARGE_SIZE);
fl_set_object_color(obj, FL_INDIANRED, FL_TOMATO);

obj = fl_add_pixmap(FL_NORMAL_BITMAP, 202, 212, 170, 170, NULL);
fl_set_pixmap_data(obj, printer);

obj = fl_add_text(FL_NORMAL_TEXT, 128, 191, 150, 34, "Campagne");
fl_set_object_color(obj, FL_SLATEBLUE, 0);
fl_set_object_lsize(obj, FL_LARGE_SIZE);
fl_set_object_lstyle(obj, FL_BOLD_STYLE);
fl_set_object_lalign(obj, FL_ALIGN_CENTER);

obj = fl_add_button(FL_NORMAL_BUTTON, 640, 400, 140, 60, "@returnarrow");
fl_set_object_lsize(obj, FL_LARGE_SIZE);
fl_set_object_color(obj, FL_INDIANRED, FL_TOMATO);
fl_set_object_callback(obj, cb_btn_retour, 0);

fl_deactivate_form(mainform);
fl_hide_form(mainform);

fl_end_form();
fl_show_form(subform, FL_PLACE_MOUSE, FL_NOBORDER, NULL);

```

```
}
```

A.4. Code source de la bibliothèque « display.h »

```
#ifndef __DISPLAY_H__
#define __DISPLAY_H__

#include <math.h>
#include <stdlib.h>
#include <forms.h>

#define FL_SEGDISPLAY (FL_USER_CLASS_START + 1) /* Class number must be between
FL_USER_CLASS_START and FL_USER_CLASS_END */
#define FL_NORMAL_SEGDISPLAY 0 /* The only type */
#define FL_HORIZONTAL 0
#define FL_VERTICAL 1
#define FL_SHIFT 2

#ifndef M_PI
#define M_PI 3.14159265359
#endif

FL_OBJECT *fl_create_segdisplay(int, FL_Coord, FL_Coord, FL_Coord, FL_COLOR, FL_COLOR);
FL_OBJECT *fl_add_segdisplay(int, FL_Coord, FL_Coord, FL_Coord, FL_COLOR, FL_COLOR);
int fl_get_segdisplay(FL_OBJECT *);
int fl_set_segdisplay(FL_OBJECT *, int);

typedef struct
{
    int value; /* Digit to display */
} SEGDISPLAY_SPEC;

#endif /* __DISPLAY_H__ */
```

A.4. Code source de la fonction de rappel de l'afficheur à 7 segments

```
void cb_timer(FL_OBJECT *obj, long data)
{
    int nombre = getNombreTotalBouteilles();

    for (int i = 3; i >= 0; i--)
    {
        int chiffre = nombre % 10;
        fl_set_segdisplay(display[i], chiffre);
        nombre /= 10;
    }

    fl_set_timer(obj, DELAY);
}
```

A.5. Code source de l'interface de remplissage

```
void cb_btn_remplissage(FL_OBJECT *obj, long data)
{
    printf("Bouton 'Remplissage' appuyé !\n");
    subform = fl_bgn_form(FL_FLAT_BOX, 800, 480);
    fl_set_form_background_color(subform, FL_SLATEBLUE);

    fl_add_labelframe(FL_BORDER_FRAME, 20, 20, 760, 360, NULL);

    obj = fl_add_text(FL_NORMAL_TEXT, 220, 0, 340, 40, "Remplissage bouteilles");
    fl_set_object_lsize(obj, FL_HUGE_SIZE);
    fl_set_object_color(obj, FL_SLATEBLUE, 0);
    fl_set_object_lstyle(obj, FL_BOLD_STYLE);
    fl_set_object_lalign(obj, FL_ALIGN_CENTER);

    display[0] = fl_add_segdisplay(FL_FLAT_BOX, 110, 80, 100, FL_RED, FL_SLATEBLUE);
    fl_set_segdisplay(display[0], 10);
    display[1] = fl_add_segdisplay(FL_FLAT_BOX, 260, 80, 100, FL_RED, FL_SLATEBLUE);
    fl_set_segdisplay(display[1], 10);
    display[2] = fl_add_segdisplay(FL_FLAT_BOX, 410, 80, 100, FL_RED, FL_SLATEBLUE);
    fl_set_segdisplay(display[2], 10);
}
```

```

display[3] = fl_add_segdisplay(FL_FLAT_BOX, 560, 80, 100, FL_RED, FL_SLATEBLUE);
fl_set_segdisplay(display[3], 10);

obj = fl_add_timer(FL_HIDDEN_TIMER, 10, 10, 10, 10, NULL);
fl_set_object_callback(obj, cb_timer, 0);
fl_set_timer(obj, DELAY);

obj = fl_add_button(FL_NORMAL_BUTTON, 640, 400, 140, 60, "@returnarrow");
fl_set_object_lsize(obj, FL_LARGE_SIZE);
fl_set_object_color(obj, FL_INDIANRED, FL_TOMATO);
fl_set_object_callback(obj, cb_btn_retour, 0);

fl_end_form();
fl_show_form(subform, FL_PLACE_MOUSE, FL_NOBORDER, NULL);

fl_deactivate_form(mainform);
fl_hide_form(mainform);
}

```

A.6. Code source de la fonction de rappel des cadrans

```

void cb_dial_timer() {
    double max_temperature = 76.0;
    double max_dial_value = 360.0;
    double dial_value1 = (sensorValues[0] / max_temperature) * max_dial_value;
    double dial_value2 = (sensorValues[1] / max_temperature) * max_dial_value;
    double dial_value3 = (sensorValues[2] / max_temperature) * max_dial_value;
    double dial_value4 = (sensorValues[3] / max_temperature) * max_dial_value;
    double dial_value5 = (sensorValues[4] / max_temperature) * max_dial_value;

    if (dial1) fl_set_dial_value(dial1, dial_value1);
    if (dial2) fl_set_dial_value(dial2, dial_value2);
    if (dial3) fl_set_dial_value(dial3, dial_value3);
    if (dial4) fl_set_dial_value(dial4, dial_value4);
    if (dial5) fl_set_dial_value(dial5, dial_value5);
}

```

A.7. Code source de l'interface de pasteurisation

```

void cb_btn_pasteurisation(FL_OBJECT *obj, long data)
{
    printf("Bouton 'Pasteurisation' appuyé !\n");
    subform = fl_bgn_form(FL_FLAT_BOX, 800, 480);
    fl_set_form_background_color(subform, FL_SLATEBLUE);

    fl_add_labelframe(FL_BORDER_FRAME, 20, 20, 760, 360, NULL);

    obj = fl_add_text(FL_NORMAL_TEXT, 300, 0, 190, 40, "Pasteurisation");
    fl_set_object_lsize(obj, FL_LARGE_SIZE);
    fl_set_object_color(obj, FL_SLATEBLUE, 0);
    fl_set_object_lstyle(obj, FL_BOLD_STYLE);
    fl_set_object_lalign(obj, FL_ALIGN_CENTER);

    dial1 = fl_add_dial(FL_FILL_DIAL, 40, 40, 160, 160, "1");
    fl_set_object_color(dial1, FL_SLATEBLUE, FL_TOMATO);
    fl_set_object_lsize(dial1, FL_LARGE_SIZE);
    fl_set_object_lstyle(dial1, FL_BOLD_STYLE);

    dial2 = fl_add_dial(FL_FILL_DIAL, 180, 180, 160, 160, "2");
    fl_set_object_color(dial2, FL_SLATEBLUE, FL_TOMATO);
    fl_set_object_lsize(dial2, FL_LARGE_SIZE);
    fl_set_object_lstyle(dial2, FL_BOLD_STYLE);

    dial3 = fl_add_dial(FL_FILL_DIAL, 325, 40, 160, 160, "3");
    fl_set_object_color(dial3, FL_SLATEBLUE, FL_TOMATO);
    fl_set_object_lsize(dial3, FL_LARGE_SIZE);
    fl_set_object_lstyle(dial3, FL_BOLD_STYLE);

    dial4 = fl_add_dial(FL_FILL_DIAL, 467, 180, 160, 160, "4");
    fl_set_object_color(dial4, FL_SLATEBLUE, FL_TOMATO);
    fl_set_object_lsize(dial4, FL_LARGE_SIZE);
    fl_set_object_lstyle(dial4, FL_BOLD_STYLE);

    dial5 = fl_add_dial(FL_FILL_DIAL, 610, 40, 160, 160, "5");
    fl_set_object_color(dial5, FL_SLATEBLUE, FL_TOMATO);
    fl_set_object_lsize(dial5, FL_LARGE_SIZE);
}

```

```
fl_set_object_lstyle(dial5, FL_BOLD_STYLE);

obj = fl_add_button(FL_NORMAL_BUTTON, 630, 400, 150, 60, "@returnarrow");
fl_set_object_lsize(obj, FL_LARGE_SIZE);
fl_set_object_color(obj, FL_INDIANRED, FL_TOMATO);
fl_set_object_callback(obj, cb_btn_retour, 0);

fl_deactivate_form(mainform);
fl_hide_form(mainform);

fl_end_form();
fl_show_form(subform, FL_PLACE_MOUSE, FL_NOBORDER, NULL);
}
```