RAPPORT DE STAGE

DAUMARIE Emeric

GINKOIA

Maître de stage : Ludovic GENEST

Période : 10/01/2022 au 25/02/2022

Diplôme : BTS SIO deuxième année

Remerciements

Tous mes remerciements s'adressent à :

Morgane DUCREY et Ludovic GENEST, pour m'avoir ouvert les portes de Ginkoia puis offert un merveilleux stage.

Sylvain CORBELETTA qui m'a dirigé durant ce stage, m'a partagé ses connaissances et ses précieux conseils.

Carole-Anne VIFFRAY avec qui j'ai pu discuter tout au long des nombreux co-voiturages effectués ensemble et pour son expérience partagée.

Toute l'équipe CASH pour son accueil, son professionnalisme et sa bonne ambiance au quotidien.

Ma lectrice et correctrice qui a contribué, grâce à ses conseils, recommandations, à l'élaboration et au bon déroulé de mon rapport de stage.

Table des matières

Introduction

Durant cette seconde année de mon BTS SIO (Services Informatiques aux Organisations), un stage d'une durée de sept semaines doit être réalisé dans le thème de l'option choisie en première année. Pour ma part, ayant choisi l'option SLAM (Solutions Logicielles et Applications Métiers), mes recherches de stage ont porté sur des entreprises proposant des services de développement logiciels et/ou de sites web.

Après de nombreuses demandes et une recherche quotidienne, c'est finalement l'entreprise GINKOIA, basée à Passy en Haute-Savoie, qui m'a ouvert ses portes. Une présentation globale de l'entreprise est énoncée en première partie de ce rapport de stage, toutefois, pour contextualiser en quelques mots, GINKOIA conçoit des logiciels répondant aux besoins des commerçants du sport.

Pendant cette période, j'ai découvert et ai pu m'initier au fonctionnement du logiciel GINKOIA. Puis, un projet m'a été confié : celui de développer le Dashbord de l'écran d'accueil du logiciel. Ceci s'est alors fait en plusieurs étapes, progressivement et de façon structurée, en respectant un cahier des charges.

Ce rapport de stage s'organise alors en trois grandes parties. D'abord, la présentation générale et globale de l'entreprise pour mieux la comprendre, la visualiser. Ensuite, mon parcours de découverte du logiciel GINKOIA. Enfin, la représentation fonctionnelle et technique du projet confié.

1 Présentation de l'entreprise

1.1 Qui est Ginkoia ?

Ginkoia, entreprise basée à Passy en Haute-Savoie, propose une solution progicielle complète pour piloter un magasin ou un réseau de points de vente. Leader auprès des enseignes de sport et avec plus de 1500 magasins qui utilisent quotidiennement ses solutions, Ginkoia est aussi un progiciel de référence sur les marchés de la mode, de la chaussure, du cycle et de la location.

Plug & Sell est la signature des solutions Ginkoia qui se veulent rapides et simples à mettre en œuvre.

En quelques chiffres, Ginkoia est le n°1 des éditeurs pour la distribution de sport, 85 collaborateurs, 9,9 millions de chiffre d'affaires en 2021 ainsi que 1700 points de vente clients.

1.1.1 Histoire de la société

En 1989, création de la société Algol à Chamonix en Haute-Savoie. A l'époque, Algol développe des logiciels de gestion pour les magasins de disques (logiciel Saxo) et la location de skis dans la vallée de Chamonix sous les noms de Husky (gestion vente) et Pulka (gestion location)

C'est en 2000 que l'entreprise réalise le déploiement des logiciels Ginkoia au sein des réseaux Sport 2000 puis Twinner et Skimium.

Dès 2008, Ginkoia rejoint le Groupe DL Software. DL Software est le spécialiste français des ERP métiers, un groupe d'éditeurs d'ERP chacun leaders sur des marchés spécifiques qui lui donnera les moyens de nourrir encore sa croissance.

Puis c'est en 2011 que l'entreprise va signer une collaboration avec Intersport France et deviendra ainsi le leader français du logiciel magasin pour la distribution sport en plaine et en montagne. Ginkoia développe alors les notions d'intégration avec le système d'information en place au sein de l'enseigne pour une chaîne sans coupure et écosystème connecté entre l'enseigne et le réseau.

2015, mise en place de ses solutions de tablette vendeur & skiman mobile. L'encaissement mobile est en pilote. De plus, en 2017, sera lancé le service de location out-store, pour un service de livraison VIP digitalisé, au sein des chalets et hôtels, ou en drive.

Enfin, en 2021, l'année est marquée par le déploiement au sein de son parc d'utilisateurs de la nouvelle version majeure de son logicielle de caisse, avec une interface utilisateur totalement redesignée.

1.1.2 Activité et marché

Avec un ensemble très large de secteurs d'activités, tels que la location, la mode, le cycle ou encore le Sport Shop, Ginkoia détient la position de leader sur son marché avec une présence au sein de 3 des 4 plus grandes enseignes d'articles de sport en France et plus de 1500 magasins clients à ce jour.

Sous-segment de marché, l'activité du commerce de location en montagne est un marché historique. Ginkoia a su y développer une présence forte avec plus de 350 magasins clients.

Ginkoia propose des services autour de l'informatisation d'un magasin pour le réseau d'une enseigne, un groupe de magasins ou un indépendant mono/magasin :

- Intégration des solutions Ginkoia
- Installation du matériel en magasin ou à distance
- Vente et location de matériels
- Formation sur site et en téléformation (organisme référencé Datadock)
- Support client logiciel et matériel 6j/7 en été, 7j/7 en hiver
- Financement de projets matériels (via son partenaire DL Lease)

1.2 Organisation

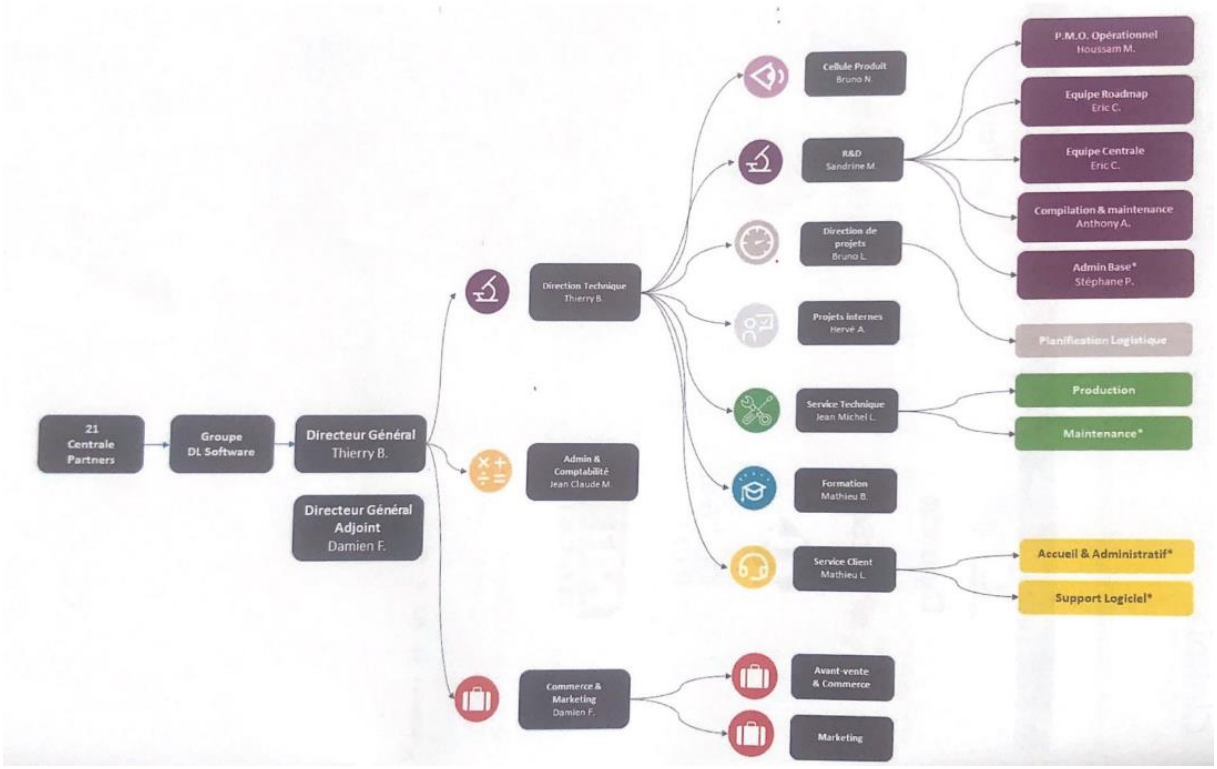

Figure 1.1 – Organigramme de la société

2 Découverte de l'environnement de travail

Pendant ce stage, j'ai été affecté à l'équipe CASH en charge du développement de la nouvelle caisse pour les enseignes du sport dont Sport 2000, Intersport ou encore Chullanka.

2.1 CASH

2.1.1 Présentation du logiciel

CASH est le logiciel de caisse présent sur tous les postes de caisse. Le logiciel peut prendre en charge les périphériques comme TPE, imprimante ticket, afficheur, tiroir-caisse et douchette. De plus, CASH à la possibilité d'être utilisé avec le tactile.

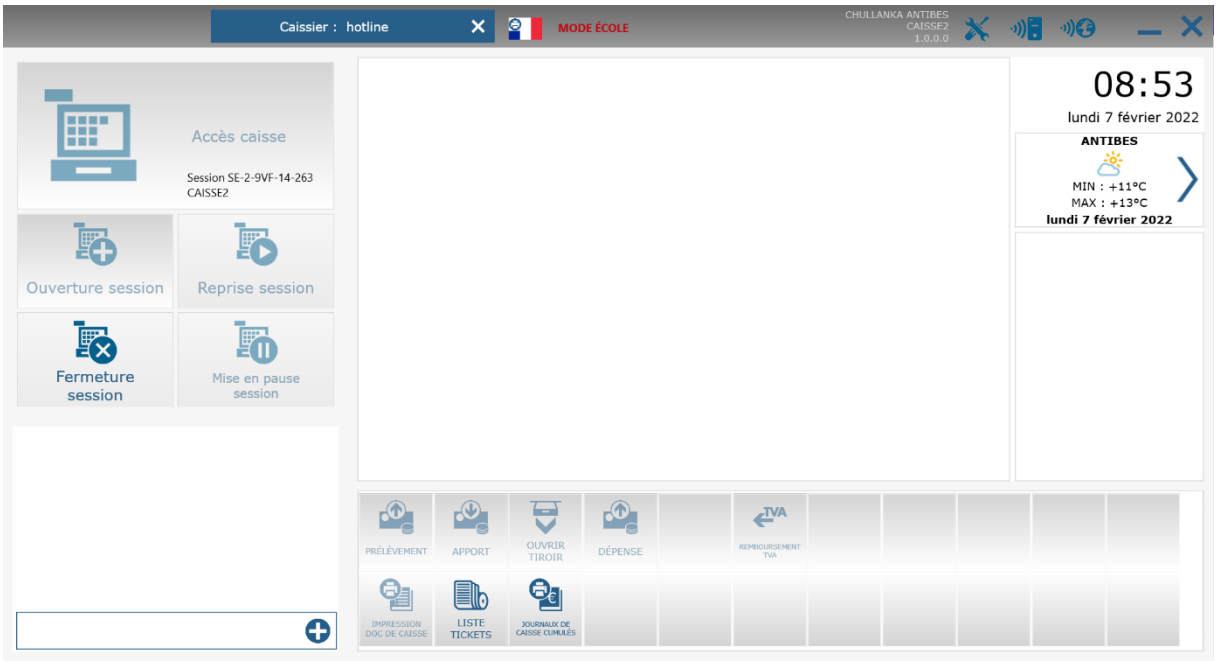

Figure 2.1 – Accueil CASH

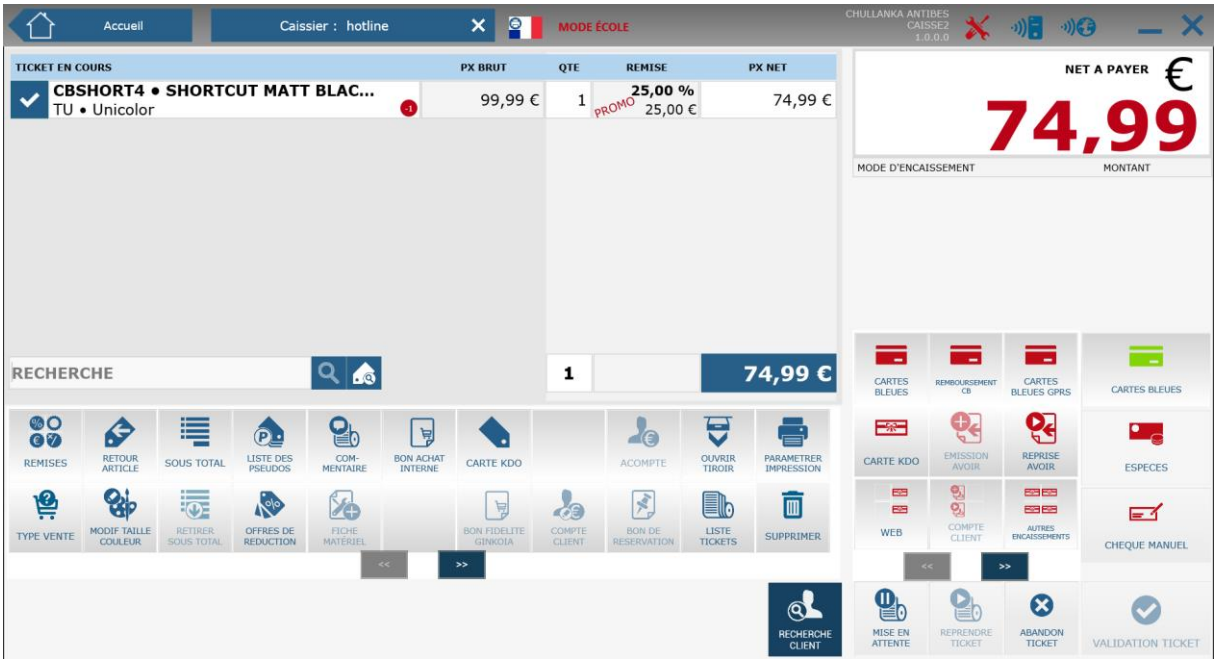

Figure 2.2 – Caisse CASH

CASH communique avec les APIs du Sabo pour récupérer n'importe quelle donnée.

2.1.2 Un travail en préface

Pour m'habituer à ce logiciel, j'ai été chargé dans un premier temps du thème de l'internationalisation. Ce sujet consistait à parcourir le logiciel et y chercher des chaînes de caractères, visibles à l'écran, pour les remplacer par leur traduction.

Pour ce faire, je devais utiliser TI18nUtils, classe créée par Ginkoia, permettant de traduire la chaîne de caractère. Il suffisait de la remplacer par un appel à la méthode 'Context' de la classe 'TI18nUtils' et lui passer comme paramètre une « clé » faisant référence à la traduction de cette chaîne. Toutefois, il fallait également que je me rende dans le fichier de traduction du logiciel pour y créer une clé avec cette même valeur.

Cet exercice m'a permis de prendre connaissance du fonctionnement du logiciel CASH, des différentes Frames (c'est-à-dire les fenêtres affichables à l'écran), mais aussi et principalement du langage et de l'environnement de travail utilisé par l'entreprise.

Néanmoins, certaines traductions pouvaient provenir du Sabo ce qui me limita dans mon travail.

2.2 SABO

2.2.1 Présentation du logiciel

Le Sabo, logiciel développé par Ginkoia, correspond à Serveur d'Application BackOffice et contient le fonctionnel.

Figure 2.3 – Interface Sabo

C'est lui qui fait le lien entre CASH et la base de données. Il permet d'agir sur celles-ci, mais également de générer des APIs pouvant être appelées directement par la caisse.

2.2.2 Le parcours des données

Comprendre le fonctionnement de ce logiciel s'est avéré être une nécessité pour poursuivre mon travail sur le sujet de l'internationalisation, mais en plus pour celui qui allait m'être confié pour la suite du stage.

J'ai donc bénéficié d'une formation sur son mode opératoire pour me repérer dans cette interface, comprendre comment les données sont transmises du SABO à la caisse CASH, de quelle façon s'effectue la communication entre eux.

Depuis la caisse, le point de départ correspond la plupart du temps à un appel d'API. Il fallait donc que je commence par trouver cet appel, puis que je remonte en suivant les méthodes utilisées.

Ce travail m'a permis de mieux m'orienter lorsque je suis à la recherche d'une donnée, d'un objet ou autre et en plus d'améliorer ma prise en main des logiciels utilisés, des environnements de travail, du langage utilisé par l'entreprise, des conventions de nommages.

3 Widgets Dashbord

Il faut savoir qu'une première mouture de ce projet a été publiée, mais qu'après mûre réflexion, c'est vers quelque chose de plus moderne que s'est tournée sa réalisation.

Dans cette partie, je présente davantage la seconde version que j'ai réalisé qui est plus opérationnelle que la première.

3.1 Objet

L'objectif de ce projet était de remplir le Dashboard de l'écran d'accueil de la caisse CASH avec des widgets.

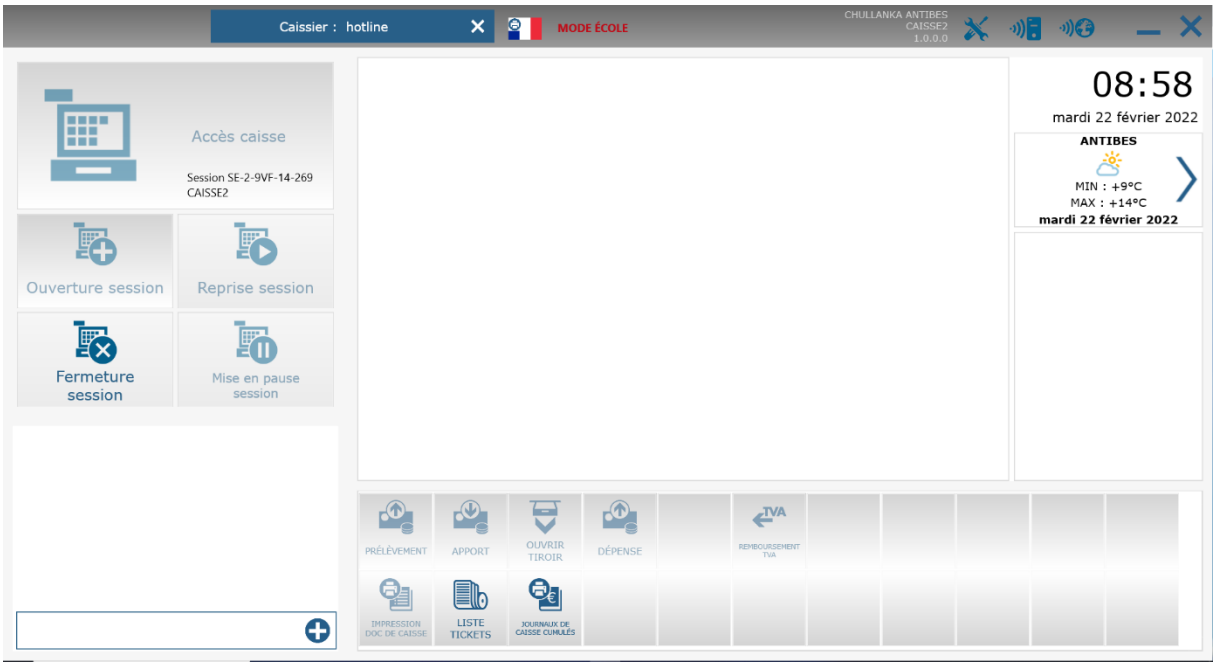

Figure 3.1 – Ancien Dashboard

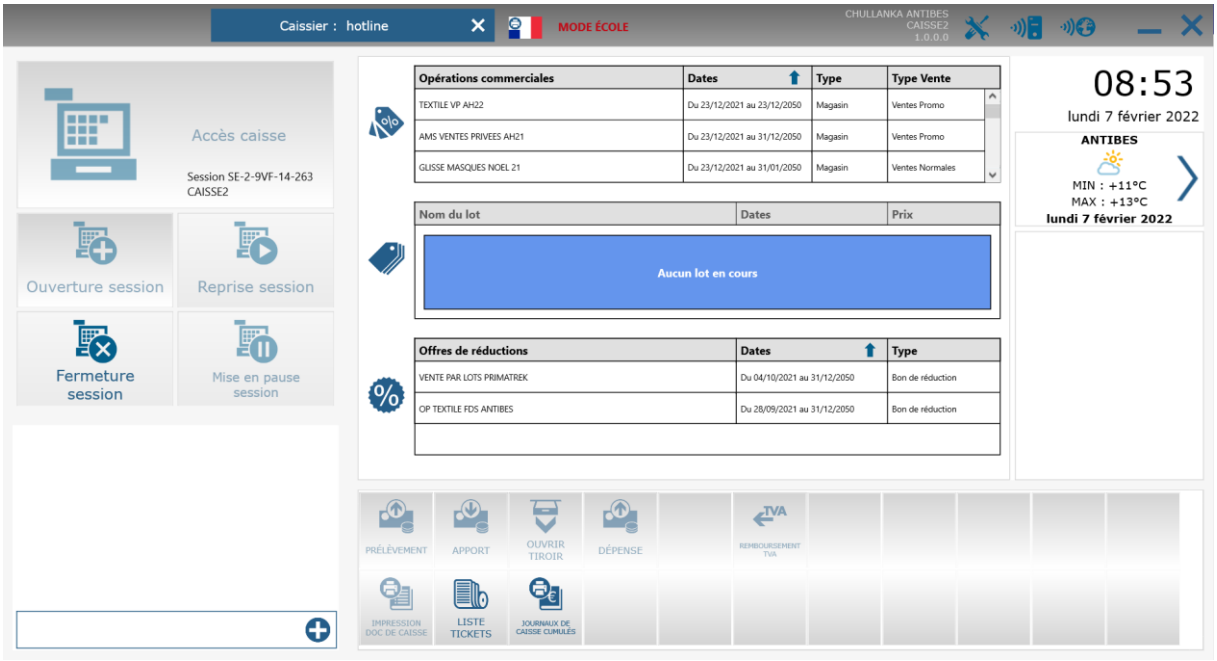

Figure 3.2 – Première version

Ci-dessus un aperçu de ma première version avec le développement des widgets. Ils sont sous forme de tableaux et peuvent être uniquement triés.

3.2 Fonctionnel

3.2.1 Démarrage

Lors du démarrage de la caisse, trois widgets sont affichés sur le Dashboard. Ces widgets représentent les remises automatiques du magasin :

- Opérations commerciales

- Lots

- Offres de réduction

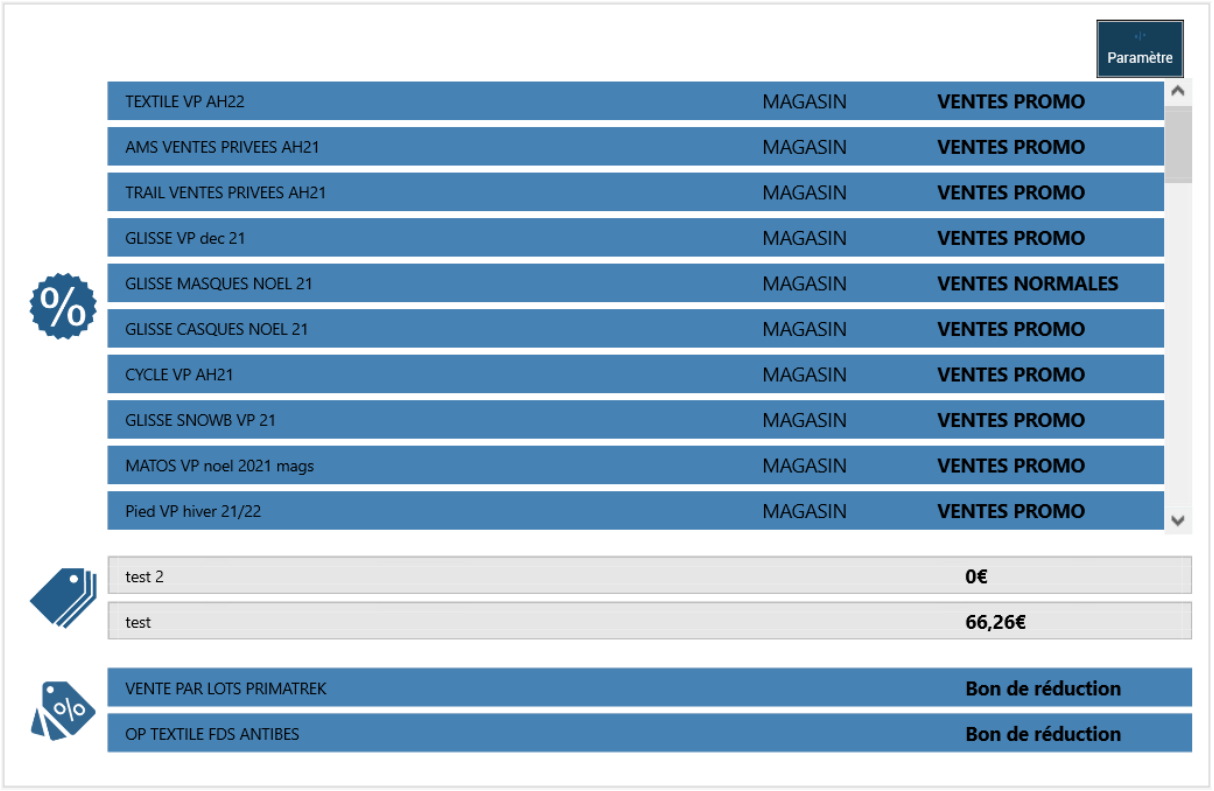

Figure 3.3 – Dashboard actuel

A noter : pour qu'un widget s'affiche, il faut au minimum une remise automatique en cours.
Lors d'une première utilisation, l'affichage de ces widgets se fait avec des paramètres par défaut.

3.2.2 Comprendre l'affichage

Au niveau visuel, si un widget s'avère non visible (choix de l'utilisateur ou du fait qu'aucune remise soit en cours), les widgets affichés s'ajustent dans l'espace disponible. De même si le volet de droite (affichant la météo) doit prendre plus de place, alors les widgets s'adaptent.
Les barres de progression sont la représentation d'une durée en jours. C'est grâce à cela que l'on peut savoir approximativement le nombre de jours restants pour une remise automatique.

Certaines informations sont communes aux trois widgets :

- Le nom de la remise
- Un code couleur :
    - Vert = remises venant de débuter
    - Rouge = remises allant se terminer
    - Gris = remises permanentes
    - Bleu = autres remises

D'autres informations seront spécifiques à chaque widget :

- Opération commerciale
    - Le type d'opération (magasin, client, vente privée)
    - Le type de remise (promo, solde, normale, …)
- Lots
    - Le prix du lot
- Offre de réduction
    - Le type (opération commerciale, bon de réduction)

Si ces informations ne suffisent pas, l'utilisateur a la possibilité de cliquer sur une remise pour des éléments complémentaires.

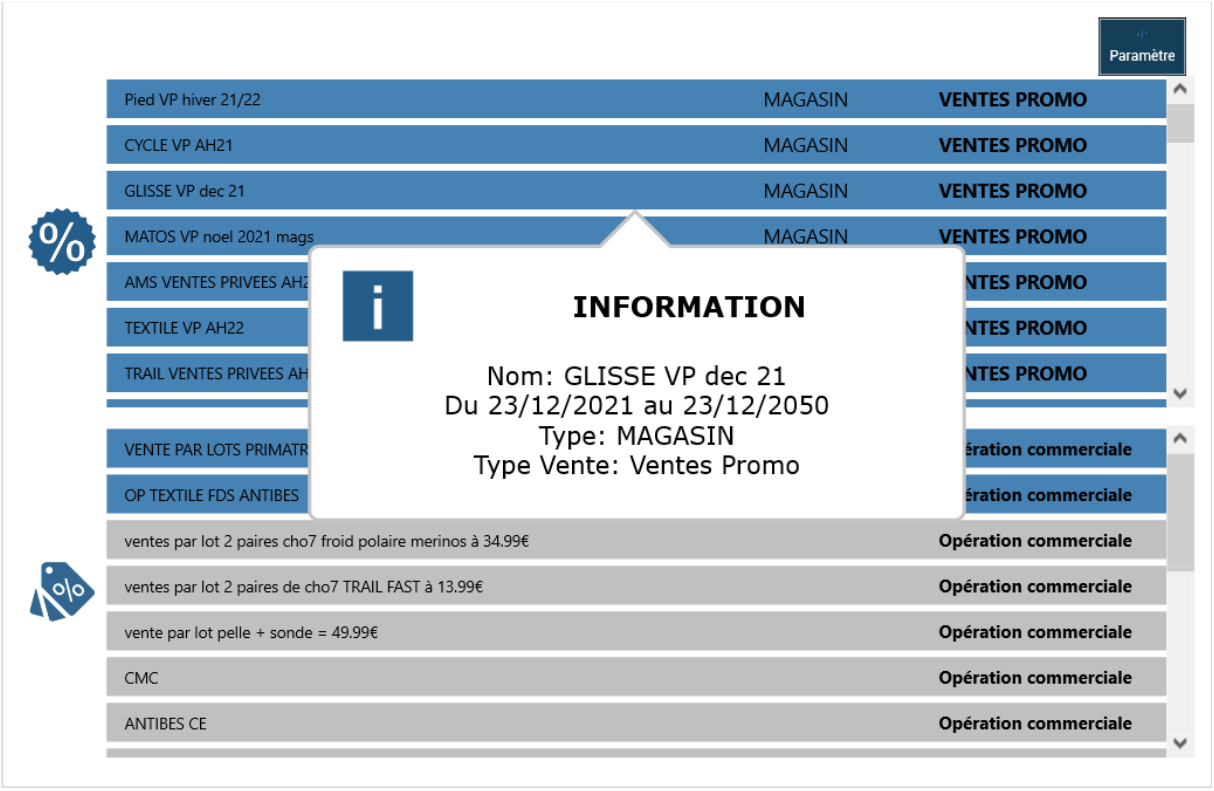

Figure 3.4 – Informations complémentaires d'une remise

3.2.3 Paramétrage

Tout d'abord, l'accès à la fenêtre de paramétrage des widgets est soumise à un droit. Uniquement les utilisateurs 'Patron' ont se droit.

L'utilisateur a donc la possibilité grâce à cette fenêtre de déterminer ses propres paramètres :

- Rendre visible ou non le widget
- Rendre visible ou non les remises permanentes
- Trier les remises selon le nom ou la date de début, de façon croissante ou décroissante
- Déterminer la durée des flags New et Last
- Selon le widget, choisir le type de remises à afficher
- Déterminer le nombre de jours que la barre de progression doit représenter

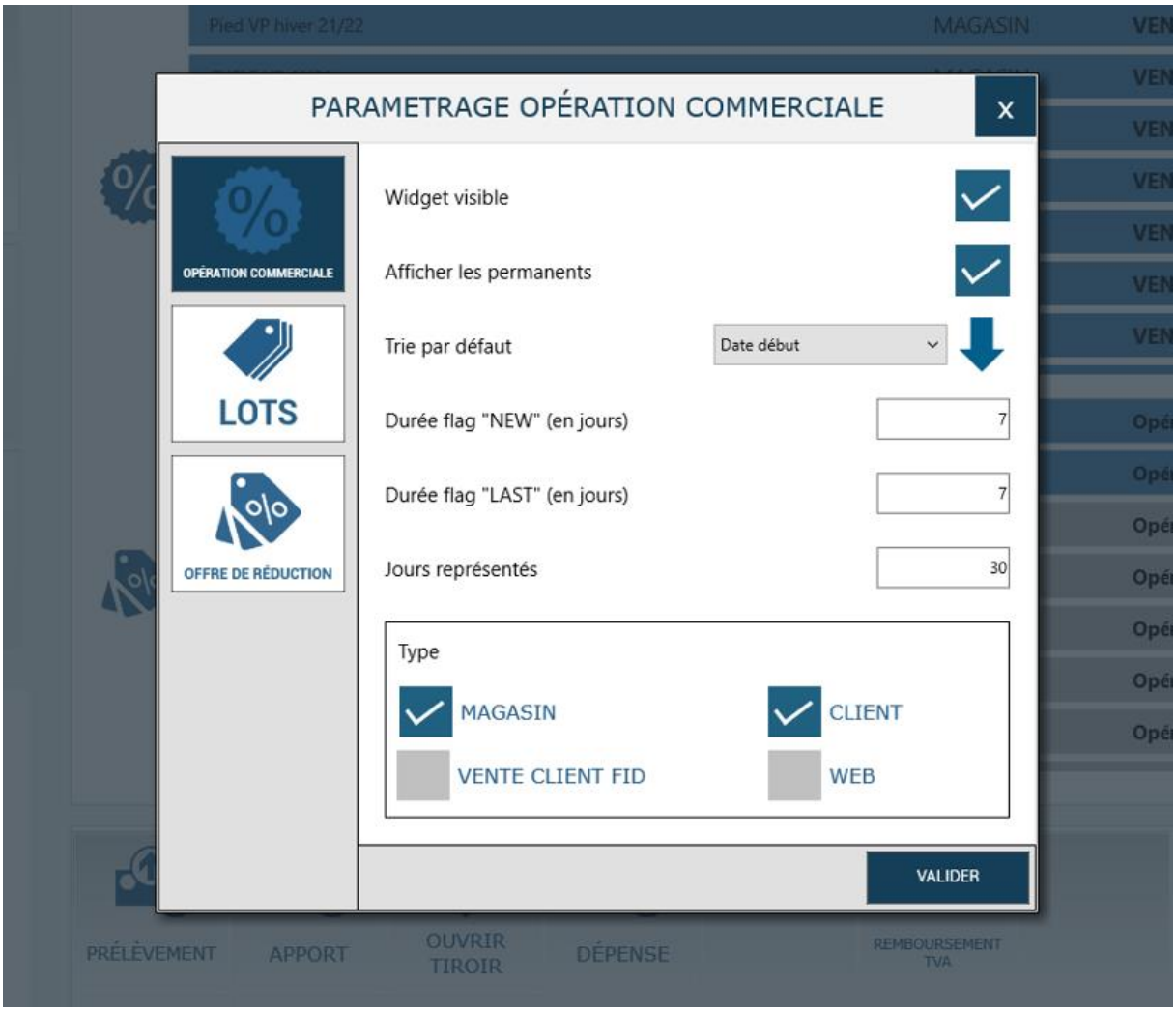

Figure 3.5 – Fenêtre de paramétrage

Par défaut et commun à chaque widget :

- Widget visible

- Affichage des remises automatiques permanentes
- Remises automatiques triées sur la date de début de façon décroissante
- Flags New et Last de 7 jours
- 30 jours représentés

Le widget des opérations commerciales et celui des offres de réduction possède un paramètre en plus qui représente le choix du type de remise.
Les opérations commerciales dont le type est Magasin ou Client sont affichées par défaut. Il est également possible de visualiser celles de type Web.
Par défaut pour les offres de réduction, toutes sont affichées (bons de réduction et opérations commerciales).

3.3 Technique

Dans cette partie, je présente uniquement le travail technique d'un des trois widgets. En effet la réalisation de chacun des widgets étant très similaire.

Ceci est variable pour le paramétrage. La façon dont je récupère les paramètres se fait de la même manière que pour les widgets. Le seul point différent est la sauvegarde de ceux-ci, dont je parlerai après (voir 3.3.4.2).

3.3.1 Patchs

Pour accéder au paramétrage des widgets, il est nécessaire d'ajouter un premier patch pour obtenir un nouveau droit.

```
EXECUTE PROCEDURE BN_UIL('CASH : PARAMETRAGE - PARAM_DASHBOARD', 'PATRON', 'CAISSE CASH');

UPDATE UILPERMISSIONS SET PER_LIBELLE='Parametrer le dashboard de la caisse CASH' WHERE PER_PERMISSION=' CASH : PARAMETRAGE - PARAM_DASHBOARD ';
```

Lors du clic sur le bouton de paramétrage, un contrôle est fait pour vérifier si oui ou non l'utilisateur possède le droit d'y accéder.

Puis d'autres patchs pour de nouveaux paramètres des widgets en base.

Exemple :

```
EXECUTE PROCEDURE SM_CREER_PARAM (666, 50, 'Widget opération commerciales - visibles','Lire PRM_INTEGER : 0 non, 1 oui', 1, 0, -1, 0);
```

3.3.2 Classes

Travailler avec des objets, classes, fut d'abord une partie intégrante dans la réalisation du projet mais aussi dans l'amélioration de mes compétences. En effet j'ai dû, pour chacun des widgets présentés ci-dessus et pour le paramétrage, développer des classes me permettant alors de travailler avec des objets pour pouvoir traiter correctement les données.

3.3.2.1 Classe Widget

```
TSelfApplicableDiscount = class(TGinClassAncestor)
  private
    FDisplayName: string;
    FStartDate: TDateTime;
    FEndDate: TDateTime;
    FPermanent: boolean;
    FCategory: integer;
    FDiscountType: string;
  public
    procedure Assign(Source: TPersistent); override;
  published
    property DisplayName: string read FDisplayName write FDisplayName;
    property StartDate: TDateTime read FStartDate write FStartDate;
    property EndDate: TDateTime read FEndDate write FEndDate;
    property Permanent: boolean read FPermanent write FPermanent;
    property Category: integer read FCategory write FCategory;
    property DiscountType: string read FDiscountType write FDiscountType;
end;

TSelfApplicableDiscounts = class(TGinCollection<TSelfApplicableDiscount>);
```

Figure 3.6 – Classe Widgets Opérations commerciales

Le code ci-dessus représente une classe assez simple que j'ai développé pour le widget des opérations commerciales.

La seconde classe déclarée en bas de l'image se différencie de la première par le fait qu'elle en dérive et qu'elle est une liste.

3.3.2.2 Classe paramétrage

```
TEnumTriWidget = (NomAsc=1, NomDesc=2, DateDebutAsc=3, DateDebDesc=4);

TDashBoardParameters = class(TGinClassAncestor)
private
  FWidgetOpeComVisible: integer;
  FWidgetOpeComPerm: integer;
  FWidgetOpeComTrie: Integer;
  FWidgetOpeComNewFlag: Integer;
  FWidgetOpeComLastFlag: Integer;
  FWidgetOpeComTypeMag: Boolean;
  FWidgetOpeComTypeCli: Boolean;
  FWidgetOpeComTypeWeb: Boolean;
  FWidgetOpeComTypeCliFid: Boolean;
  FWidgetOpeComProgressBar: integer;

  FWidgetLotsVisible: integer;
  FWidgetLotsPerm: integer;
  FWidgetLotsTrie: Integer;
  FWidgetLotsNewFlag: Integer;
  FWidgetLotsLastFlag: Integer;
  FWidgetLotsProgressBar: integer;

  FWidgetOffreReducVisible: integer;
  FWidgetOffreReducPerm: integer;
  FWidgetOffreReducTrie: Integer;
  FWidgetOffreReducNewFlag: Integer;
  FWidgetOffreReducLastFlag: Integer;
  FWidgetOffreReducTypeOP: boolean;
  FWidgetOffreReducTypeReduc: boolean;
  FWidgetOffreReducProgressBar: integer;
public
  constructor Create; overload;
  constructor Create(sWidgetParameters: TDashBoardParameters); overload;
  procedure Assign(AObject: TDashBoardParameters); reintroduce;

  property WidgetOpeComVisible: integer read FWidgetOpeComVisible write FWidgetOpeComVisible;
  property WidgetOpeComPerm: integer read FWidgetOpeComPerm write FWidgetOpeComPerm;
  property WidgetOpeComTrie: Integer read FWidgetOpeComTrie write FWidgetOpeComTrie;
  property WidgetOpeComNewFlag: Integer read FWidgetOpeComNewFlag write FWidgetOpeComNewFlag;
  property WidgetOpeComLastFlag: Integer read FWidgetOpeComLastFlag write FWidgetOpeComLastFlag;
  property WidgetOpeComTypeMag: boolean read FWidgetOpeComTypeMag write FWidgetOpeComTypeMag;
  property WidgetOpeComTypeCli: boolean read FWidgetOpeComTypeCli write FWidgetOpeComTypeCli;
  property WidgetOpeComTypeWeb: boolean read FWidgetOpeComTypeWeb write FWidgetOpeComTypeWeb;
  property WidgetOpeComTypeCliFid: boolean read FWidgetOpeComTypeCliFid write FWidgetOpeComTypeCliFid;
  property WidgetOpeComProgressBar: integer read FWidgetOpeComProgressBar write FWidgetOpeComProgressBar;
  procedure InitOpeComTypeValue(vObj: TDashBoardParameters; aStr: string);
  function SetOpeComTypeValue(withTypeMag, withTypeCli, withTypeWeb, withTypeCliFid: boolean): string;

  property WidgetLotsVisible: integer read FWidgetLotsVisible write FWidgetLotsVisible;
  property WidgetLotsPerm: integer read FWidgetLotsPerm write FWidgetLotsPerm;
  property WidgetLotsTrie: Integer read FWidgetLotsTrie write FWidgetLotsTrie;
  property WidgetLotsNewFlag: Integer read FWidgetLotsNewFlag write FWidgetLotsNewFlag;
  property WidgetLotsLastFlag: Integer read FWidgetLotsLastFlag write FWidgetLotsLastFlag;
  property WidgetLotsProgressBar: integer read FWidgetLotsProgressBar write FWidgetLotsProgressBar;

  property WidgetOffreReducVisible: integer read FWidgetOffreReducVisible write FWidgetOffreReducVisible;
  property WidgetOffreReducPerm: integer read FWidgetOffreReducPerm write FWidgetOffreReducPerm;
  property WidgetOffreReducTrie: Integer read FWidgetOffreReducTrie write FWidgetOffreReducTrie;
  property WidgetOffreReducNewFlag: Integer read FWidgetOffreReducNewFlag write FWidgetOffreReducNewFlag;
  property WidgetOffreReducLastFlag: Integer read FWidgetOffreReducLastFlag write FWidgetOffreReducLastFlag;
  property WidgetOffreReducTypeOP: Boolean read FWidgetOffreReducTypeOP write FWidgetOffreReducTypeOP;
  property WidgetOffreReducTypeReduc: Boolean read FWidgetOffreReducTypeReduc write FWidgetOffreReducTypeReduc;
  property WidgetOffreReducProgressBar: integer read FWidgetOffreReducProgressBar write FWidgetOffreReducProgressBar;
  procedure InitOffreReducTypeValue(vObj: TDashBoardParameters; aInt: integer);
  function SetOffreReducTypeValue(withTypeOP, withTypeReduc: boolean): integer;
end;
```

Figure 3.7 – Classe paramétrage Widgets

Voici un visuel de la classe développée pour permettre le paramétrage des widgets. Chaque attribut exprime un paramètre, c'est pour cela qu'il y en a autant.

Cette dernière comporte également deux constructeurs ('Create'). En fait, ceci est une sorte de sécurité apportée dans le cas où les données en base seraient inaccessibles. Le premier constructeur contient les paramètres par défaut des widgets.

```
constructor TDashBoardParameters.Create;
begin
  inherited;

  FWidgetOpeComVisible          := 1;
  FWidgetOpeComPerm             := 1;
  FWidgetOpeComTrie             := 4;
  FWidgetOpeComNewFlag          := 7;
  FWidgetOpeComLastFlag         := 7;
  FWidgetOpeComTypeMag          := True;
  FWidgetOpeComTypeCli          := True;
  FWidgetOpeComTypeWeb          := False;
  FWidgetOpeComTypeCliFid       := True;
  FWidgetOpeComProgressBar      := 30;

  FWidgetLotsVisible            := 1;
  FWidgetLotsPerm               := 1;
  FWidgetLotsTrie               := 4;
  FWidgetLotsNewFlag            := 7;
  FWidgetLotsLastFlag           := 7;
  FWidgetLotsProgressBar        := 30;

  FWidgetOffreReducVisible      := 1;
  FWidgetOffreReducPerm         := 1;
  FWidgetOffreReducTrie         := 4;
  FWidgetOffreReducNewFlag      := 7;
  FWidgetOffreReducLastFlag     := 7;
  FWidgetOffreReducTypeOP       := True;
  FWidgetOffreReducTypeReduc    := True;
  FWidgetOffreReducProgressBar  := 30;
end;
```

Figure 3.8 – Constructeur paramétrage par défaut

Pour pouvoir récupérer et sauvegarder les paramètres utilisateurs, j'ai surchargé le constructeur.

```
constructor TDashBoardParameters.Create(sWidgetParameters: TDashBoardParameters);
begin
  inherited Create;

  FWidgetOpeComVisible          := sWidgetParameters.WidgetOpeComVisible;
  FWidgetOpeComPerm             := sWidgetParameters.WidgetOpeComPerm;
  FWidgetOpeComTrie             := sWidgetParameters.WidgetOpeComTrie;
  FWidgetOpeComNewFlag          := sWidgetParameters.WidgetOpeComNewFlag;
  FWidgetOpeComLastFlag         := sWidgetParameters.WidgetOpeComLastFlag;
  FWidgetOpeComTypeMag          := sWidgetParameters.WidgetOpeComTypeMag;
  FWidgetOpeComTypeCli          := sWidgetParameters.WidgetOpeComTypeCli;
  FWidgetOpeComTypeWeb          := sWidgetParameters.WidgetOpeComTypeWeb;
  FWidgetOpeComTypeCliFid       := sWidgetParameters.WidgetOpeComTypeCliFid;
  FWidgetOpeComProgressBar      := sWidgetParameters.WidgetOpeComProgressBar;

  FWidgetLotsVisible            := sWidgetParameters.WidgetLotsVisible;
  FWidgetLotsPerm               := sWidgetParameters.WidgetLotsPerm;
  FWidgetLotsTrie               := sWidgetParameters.WidgetLotsTrie;
  FWidgetLotsNewFlag            := sWidgetParameters.WidgetLotsNewFlag;
  FWidgetLotsLastFlag           := sWidgetParameters.WidgetLotsLastFlag;
  FWidgetLotsProgressBar        := sWidgetParameters.WidgetLotsProgressBar;

  FWidgetOffreReducVisible      := sWidgetParameters.WidgetOffreReducVisible;
  FWidgetOffreReducPerm         := sWidgetParameters.WidgetOffreReducPerm;
  FWidgetOffreReducTrie         := sWidgetParameters.WidgetOffreReducTrie;
  FWidgetOffreReducNewFlag      := sWidgetParameters.WidgetOffreReducNewFlag;
  FWidgetOffreReducLastFlag     := sWidgetParameters.WidgetOffreReducLastFlag;
  FWidgetOffreReducTypeOP       := sWidgetParameters.WidgetOffreReducTypeOP;
  FWidgetOffreReducTypeReduc    := sWidgetParameters.WidgetOffreReducTypeReduc;
  FWidgetOffreReducProgressBar := sWidgetParameters.WidgetOffreReducProgressBar;
end;
```

Figure 3.9 – Constructeur paramétrage utilisateur

Pour finir, on remarque que cette classe comprend les méthodes ‘InitOpeComTypeValue’ et ‘SetOpeComTypeValue’ (de même pour les offres de réduction). L’explication vient du fait que, dans la fenêtre de paramétrage, l’utilsateur à la possibilité de choisir le type de remise (Magasin, Client, Web, etc) en cochant (ce qui se traduit par True ou False).

Cependant, en base, le type des remises est sauvegardé sous la forme ‘000’ en chaîne de caractère. J’ai de ce fait dû développer des méthodes permettant de gérer ceci.

```
procedure TDashBoardParameters.InitOpeComTypeValue(vObj: TDashBoardParameters; aStr: string);
begin
  vObj.FWidgetOpeComTypeMag    := aStr.Substring(0,1) = '1';
  vObj.FWidgetOpeComTypeCli    := aStr.Substring(1,1) = '1';
  vObj.FWidgetOpeComTypeWeb    := aStr.Substring(2,1) = '1';
  vObj.FWidgetOpeComTypeCliFid := aStr.Substring(3,1) = '1';
end;

function TDashBoardParameters.SetOpeComTypeValue(withTypeMag, withTypeCli, withTypeWeb, withTypeCliFid: boolean): string;
begin
  if withTypeMag then Result    := Result + '1' else Result := Result + '0';
  if withTypeCli then Result    := Result + '1' else Result := Result + '0';
  if withTypeWeb then Result    := Result + '1' else Result := Result + '0';
  if withTypeCliFid then Result := Result + '1' else Result := Result + '0';
end;
```

Figure 3.10 – Méthodes gérants le type de remise

La première méthode, pour la récupération des données en base, va initialiser les attributs gérants le type de remise. Comme exprimé ci-dessus, ces différentes remises sont de type chaîne de caractère et concaténés (‘000’ par exemple). Il me suffisait alors de récupérer chaque caractère et de vérifier si celui-ci était égal à 1, ce qui signifie alors True pour l’attribut.

La seconde méthode, pour la sauvegarde des données en base, est l’inverse de la première. Si un type est True, alors on retourne ‘1’. Ainsi de suite jusqu’à obtenir une chaîne de caractère sous forme ‘110’ par exemple.

3.3.3 Module

Les modules sont un point important dans ce projet. En effet, concernant le widget des offres de réduction, son affichage dépendra du magasin. Pour pouvoir le visualiser, il faut que le magasin en question possède le module ‘OFFRE DE REDUCTION’. Cette vérification se fait lors de l’ouverture de la caisse.
Dans le code, ceci se résume par une vérification de ce module dans le magasin.

```
constructor TFrameSelfApplicableDiscountMainFrame.Create(AOwner: Tcomponent; AHolder: TFmxObject);
begin
  inherited Create(AOwner);
  Parent := AHolder;

  FSelfApplicableDiscountSettingsFrame := TFrameSelfApplicableDiscountSettings.Create(Self);
  FSelfApplicableDiscount := TFrameSelfApplicableDiscount.Create(Self, GridPanelLayout1);
  FBatchesDiscount := TFrameBatchesDiscount.Create(Self, GridPanelLayout1);
  isModuleOffreReducActif := dm_Main.FonctionnaliteActive('OFFRE DE REDUCTION');
  if isModuleOffreReducActif then
  begin
    FOffersDiscount :=  TFrameOffersDiscount.Create(Self, GridPanelLayout1);
    FSelfApplicableDiscountSettingsFrame.GinBtnOffreReduc.Visible := True;
  end;

  TDm_Main.SetBitmap(GinBtnSettings.Icon, 'IMG_PARAMETRES');

  GetWidgetParam;

  InitWidgets;

  FTotalWidgetsRows := FSelfApplicableDiscount.CountRows + FBatchesDiscount.CountRows + FOffersDiscount.CountRows;
  //GestionAffichageWidget;

  Traduction;
end;
```

Figure 3.11 – Vérification module

Voici une illustration du contrôle du module, fait lors du constructeur de la frame dans laquelle sont appelés les widgets.
On remarque en effet un appel à une méthode ‘FonctionnaliteActive’ qui prend comme paramètre le nom du module souhaité. Cette méthode renvoie alors True si le module est actif et False dans le cas contraire. Si le module est actif, l’affichage du widget sera effectué.

3.3.4 Les données

3.3.4.1 Récupération des données

Les paramètres sont récupérés lors de l'ouverture de la caisse, appliqués dans la fenêtre de paramétrage et pris en compte lors de l'affichage des widgets.

Toutefois, si la récupération des paramètres échoue (suite à un problème quelconque), ceux par défaut seront de nouveau utilisés pour tout de même permettre d'avoir un visuel des remises automatiques en cours dans le magasin.

Premièrement : dans le Sabo, il m'a fallu récupérer les données grâce à une requête Linq.

```
function TSearchOffers.GetSelfApplicableDiscount(MagID: integer): TList<TOctete>;
var
  query: ILinqQueryable;
  join : ILinqJoin;
  currentDate: TDateTime;
begin
  join := From(FContext.Octete, FContext.Octete.OctId, FContext)
    .Join(FContext.Ocmag).&On(FContext.Ocmag.OcmOctid = FContext.Octete.OctId)
    .Join(FContext.Gentypcdv).&On(FContext.Octete.OctTypid = FContext.Gentypcdv.TypId);

  join := JoinK(join, FContext.Ocmag, FContext.Ocmag.OcmId, FContext);
  currentDate := Now;
  query := join
    .Where((FContext.Ocmag.OcmMagid = MagID)
      and (FContext.Octete.OctArchive<>1)
      and (FContext.Octete.OctDebut<currentDate)
      and (FContext.Octete.OctFin>currentDate))
    .Select([
      FContext.Octete.OctId,
      FContext.Octete.OctNom,
      FContext.Octete.OctComment,
      FContext.Octete.OctDebut,
      FContext.Octete.OctFin,
      FContext.Octete.OctTypid,
      FContext.Octete.OctWeb,
      FContext.Octete.OctCentrale,
      FContext.Octete.OctCode,
      FContext.Octete.OctTypeprix,
      FContext.Octete.OctArchive,
      FContext.Octete.OctIdexterne,
      FContext.Octete.OctSource,
      FContext.Octete.OctDemarque2,
      FContext.Octete.OctDebut2,
      FContext.Octete.OctFin2,
      FContext.Octete.OctDemarque3,
      FContext.Octete.OctDebut3,
      FContext.Octete.OctFin3
      ]).Distinct;

  Result := FContext.GetEntities<TOctete>(query).toList;
end;
```

Figure 3.12 – Requête Linq pour les opérations commerciales

Petite précision : rien n'est supprimé en base. C'est pour cela que je fais une jointure avec une table 'K' pour vérifier que la donnée que je souhaite récupérer n'est pas archivée mais bien toujours active.

'FContext' qui correspond au 'Context' de l'ORM (Object Relational Mapping), est le point d'entrée à la base de données. Il permet d'instancier des objets à partir d'enregistrements de la base : « Result := FContext.GetEntities<TOctete>(query).toList ; ».

Deuxièmement : une fois les données récupérées, je devais convertir les objets fonctionnels du Sabo (objet métier) en objets visuels.

```
class function TConvert.Convert_SelfApplicableDiscountList(aList: TList<TOctete>; AFreeAfterConvert: boolean): TSelfApplicableDiscount
var
  i: integer;
begin
  Result := TSelfApplicableDiscounts.Create();

  for i := 0 to aList.Count - 1 do
  begin
    Result.add(TConvert.Convert_SelfApplicableDiscount(aList.items[i]));
  end;

  if AFreeAfterConvert then
    FreeAndNil(aList);
end;

class function TConvert.Convert_SelfApplicableDiscount(aObj: TOctete): TSelfApplicableDiscount;
var
  StartDateStr: string;
  EndDateStr: string;
begin
  Result := TSelfApplicableDiscount.Create;
  if Assigned(aObj) then
  begin
    Result.DisplayName := aObj.OctNom;
    Result.StartDate := aObj.OctDebut.Value;
    Result.EndDate := aObj.OctFin.Value;
    Result.Category := aObj.OctWeb.Value;
    Result.DiscountType := aObj.Gentypcdv.TypLib;

    StartDateStr := DateToStr(Result.StartDate);
    EndDateStr := DateToStr(Result.EndDate);

    if (StartDateStr = '01/01/1950') and (EndDateStr = '01/01/2050') then
    begin
      Result.Permanent := True;
    end;
  end;
end;
```

Figure 3.13 – Conversion en objet visuel

Enfin : j'ai dû générer un appel d'API pour pouvoir récupérer les données du côté de la caisse CASH.

```
procedure TFrameSelfApplicableDiscount.GetOffreCom;
begin
  dm_Main.SetImage(ImgOpe, 'IMG_REMISE_POURCENT');
  TDSActionSecurisee<TSelfApplicableDiscounts>.ExecuterProcedure(Self, TGetSelfApplicableDiscountAction.Create()
  ).ExecuterSiFinie(
    procedure (const aResponse: TDSTypeResponse<TSelfApplicableDiscounts>)
    begin
      vSelfApplicableDiscount := aResponse.Result;
    end).ExecuterSiErreur(
    procedure(AException: Exception)
    begin
      dm_Main.GestionException(AException,
        procedure(AException: Exception)
        begin
          AddLog('Le chargement du Widget a échoué !' + sLineBreak + AException.Message, logError, UnitName, GetProcName(ClassName, Pro
        end);
    end).ExecuterEtAttendreLaFin(False);
end;
```

Figure 3.14 – Appel d'API

Cet appel me retourne donc un objet que je peux utiliser pour l'affichage des données qu'il contient.

Ici, j'ai expliqué comment j'ai obtenu les données d'un widget uniquement. Je rappelle que pour les deux autres widgets ainsi que la récupération des paramètres, j'ai effectué le même cheminement.

3.3.4.2 Sauvegarde des données

Pour ce qui est de la sauvegarde des paramètres, elle a lieu lorsque l'utilisateur décide de valider ses choix. J'ai, dans un premier temps, dû récolté les choix de l'utilisateur, les affecter aux attributs de l'objet contenant tous les paramètres, puis envoyer l'objet au Sabo.

Pour ce faire, j'ai préparé un appel d'API dans lequel j'ai passé l'objet en paramètre. De fait, du côté du Sabo, j'ai pu récupérer l'objet puis mettre à jour les données en base.

3.3.4.3 Affichage des données à l'écran

L'affichage se divise en trois Frames distinctes.

La première est la Frame principale. C'est dans celle-ci que les widgets sont appelés lors de leur instanciation (voir Figure 3.11).

La seconde va afficher le widget dans son intégralité. Elle est étroitement liée à la troisième Frame puisqu'elle appelle une property pour permettre l'affichage de chaque remise automatique.

La troisième correspond à l'affichage d'une seule remise. La précédente l'appelle pour chaque remise à afficher. Cette dernière possède donc une property qui va lire l'objet contenant la remise et l'afficher grâce aux attributs de l'objet.

```
procedure TFrameSelfApplicableDiscount.FillTab;
var
  I: integer;
  FrameCommercialOfferDisplayLine: TFrameCommercialOfferDisplayLine;
begin
  CountRows := 0;
  ScrollBox1.BeginUpdate;
  Try
    I := 0;
    while I < vSelfApplicableDiscount.Count do
    begin
      if CanFillTab(I) then
      begin
        FrameCommercialOfferDisplayLine := TFrameCommercialOfferDisplayLine.Create(self);
        FrameCommercialOfferDisplayLine.Parent := ScrollBox1;
        FrameCommercialOfferDisplayLine.Align := TAlignLayout.Top;
        FrameCommercialOfferDisplayLine.Position.Y := -1;
        FrameCommercialOfferDisplayLine.Name := 'TFrameCommercialOfferDisplayLine' + IntToStr(I);
        FrameCommercialOfferDisplayLine.SelfApplicableDiscount := vSelfApplicableDiscount.Items[I];
        CountRows := CountRows+1;
      end;
      inc(I);
    end;
    Frm_SelfApplicableDiscountMainFrame.NoCategoryLine(CountRows, ScrollBox1, 'OpeCom');
  Finally
    ScrollBox1.EndUpdate;
  End;
end;
```

Figure 3.15 – Appel de la troisième Frame, fait par la seconde Frame

```
procedure TFrameCommercialOfferDisplayLine.SetSelfApplicableDiscount(const Value: TSelfApplicableDiscount);
begin
  FMessage := TCallOutMessage.Create(Frm_main);
  FSelfApplicableDiscount := Value;

  lblDisplayName.Text := FSelfApplicableDiscount.DisplayName;

  lblType.Text := CategoryCase;

  lblTypeVente.Text := AnsiUpperCase(FSelfApplicableDiscount.DiscountType);
end;

function TFrameCommercialOfferDisplayLine.CategoryCase: string;
begin
  case FSelfApplicableDiscount.Category of
    0 : Result := UpperCase(TI18nUtils.Translate('FrameCommercialOfferDisplayLine_lblTypeMag'));
    1 : Result := UpperCase(TI18nUtils.Translate('FrameCommercialOfferDisplayLine_lblTypeWeb'));
    2 : Result := UpperCase(TI18nUtils.Translate('FrameCommercialOfferDisplayLine_lblTypeCli'));
  else
    Result := TI18nUtils.Translate('FrameCommercialOfferDisplayLine_lblTypeError');
  end;
end;
```

Figure 3.16 – Affichage de la remise

3.3.5 Ajustement automatique Widgets

Une des difficultés que j'ai rencontré lors de la réalisation de mon projet est l'ajustement automatique des widgets. Car les remises à afficher à l'écran sont variables, on ne peut donc pas déterminer la taille de chaque widget.

Il faut savoir que le composant utilisé dans la Frame principale qui affiche les widgets proposait trois possibilités : un affichage automatique, en pourcent ou en pixels. La première possibilité fut non

appropriée dans mon cas puisqu'elle reprenait la taille des Frames affichant les widgets (donc des tailles statiques). Pour la seconde possibilité il fallait obligatoirement une valeur totale de 100%. Donc je devais déterminer les pixels à attribuer à chaque widget pour qu'ils soient affichés convenablement. J'ai alors écrit un algorithme pour apporter une solution.

J'ai remarqué qu'une ligne de remise représentait environ 32 pixels à l'affichage. De ce fait, pour afficher entièrement un widget avec x lignes (x remises automatiques), il fallait x*32 pixels.

Cependant, il fallait de la place pour les trois widgets. Alors, un tri entre les trois widgets était nécessaire pour déterminer un classement, dans l'ordre croissant, en fonction du nombre de lignes de chacun.

```
{Tri du tableau}
for i := 0 to Length(TabWidgetsRowsValue)-1 do
begin
  for j := 0 to Length(TabWidgetsRowsValue)-1 do
    begin
      if j<Length(TabWidgetsRowsValue)-1 then
      begin
        if TabWidgetsRowsValue[j] > TabWidgetsRowsValue[j+1] then
        begin
          temp := TabWidgetsRowsValue[j];
          TabWidgetsRowsValue[j] := TabWidgetsRowsValue[j+1];
          TabWidgetsRowsValue[j+1] := temp;
        end;
      end;
    end;
end;
```

Figure 3.17 – Algorithme d'ajustement (Tri)

Ceci permet d'éviter qu'un widget reçoive trop de pixels et qu'il n'en reste plus assez pour les autres.

```
{Distribution}
for i := 0 to Length(TabWidgetsRowsValue)-1 do
begin
  for j := v to TabWidgetsRowsValue[i]-1 do
  begin
    for k := i to Length(TabWidgetsRowsValue)-1 do
    begin
      WidgetRowHeight := 32;
      while WidgetRowHeight>0 do
      begin
        for s := k to Length(TabWidgetsRowsValue)-1 do
        begin
          TabWidgetsFinalValues[s] := TabWidgetsFinalValues[s]+1;
          ParentHeight := ParentHeight-1;
          if ParentHeight<=0 then break;
        end;
        if ParentHeight<=0 then break;
        WidgetRowHeight := WidgetRowHeight -1;
      end;
      if ParentHeight<=0 then break;
    end;
    if ParentHeight<=0 then break;
  end;
  v := TabWidgetsRowsValue[i];
end;
```

Figure 3.18 – Algorithme d'ajustement (Distribution des pixels)

On commence une boucle sur le nombre de widgets :

- On récupère le nombre de lignes du widget en cours
- Pour chaque ligne de ce widget, on distribue 32 pixels à chaque widget

Lorsque ceci est terminé, on passe au nombre de lignes du second widget, mais en attribuant que (32*le nombre de lignes restantes) pour arriver au nombre de lignes du widget.

Par exemple : le premier widget possède 2 lignes, on distribue 32*2 à chacun; (la boucle recommence en se plaçant sur le widget suivant car le premier aura son nombre de pixels requis pour être affiché correctement) le second possède 3 lignes, on ne distribue que 32*(3-2) car il n'a pas besoin de plus pour être affiché correctement. On vérifie que la hauteur du rectangle parent n'a pas été atteinte, si oui alors on break jusqu'à être sorti entièrement de l'algorithme.

Ainsi de suite jusqu'à avoir atteint la hauteur du rectangle dans lequel ils sont affichés.

L'algorithme distribue en réalité 1 pixel par 1 pour une répartition plus précise.

Par exemple : en distribuant 32 pixels par 32 il y a un risque de dépasser la taille du composant, dans lequel les widgets sont affichés, de minimum 32 pixels.

En distribuant 1 pixel par 1 on risque de dépasser au maximum de 2 pixels.

Conclusion

Pour conclure, ce stage m'a permis de découvrir le travail de développeur en entreprise, le fonctionnement, le développement et la gestion d'un logiciel professionnel. Ainsi que le travail d'équipe, la communication, le partage d'informations pour la réussite des missions.

J'ai également appris la notion de code propre, comprendre pourquoi il est important de bien nommer ses variables, méthodes, etc et aussi de mutualiser mon code.

J'ai découvert les revus de code, ce qui m'a vraiment fait prendre conscience des notions citées précédemment.

Enfin, travailler avec des développeurs expérimentés au quotidien, discuter avec eux, écouter leurs conseils, m'a été bénéfique dans mes connaissances et mon apprentissage en tant qu'étudiant.