RAPPORT DE PROJET :

GESTIONNAIRE DE BUDGETS

BAUDON Nicolas

Juillet 2020

SOMMAIRE:

I/ Problématique :

Comment toujours savoir où nous dépensons notre argent pour ne jamais dépasser notre budget ?

Il est facile de dépenser plus d'argent que l'on ne le pense dans certains types de produits et de services sans forcément se rendre compte des montants totaux.

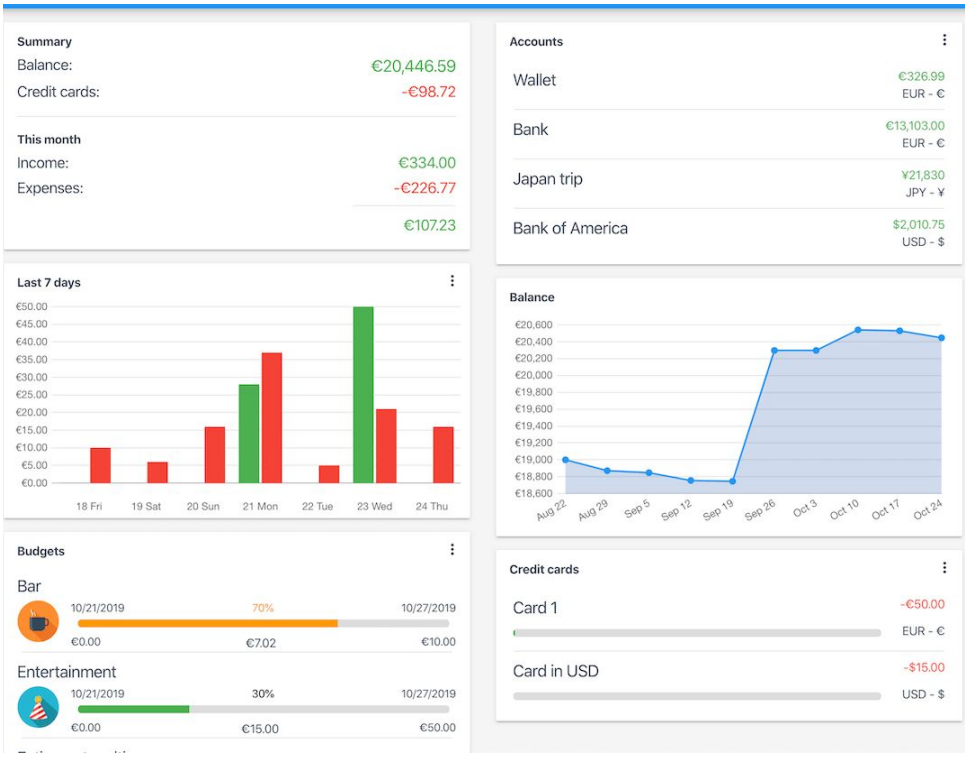
Mon but est de fournir à l'utilisateur un outil lui permettant de toujours **avoir une vue globale sur ses dépenses**. Ainsi il peut savoir dans quels domaines il débourse son argent et s'il le souhaite changer ses secteurs de dépenses pour faire des économies ou pour utiliser son argent à meilleur escient.

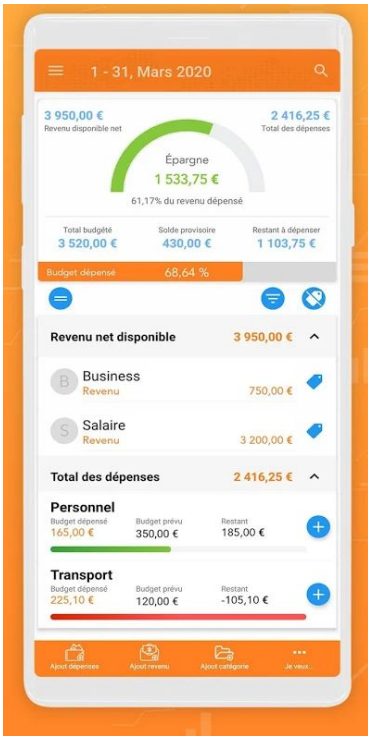
Un **gestionnaire de Budget** semble donc une solution adéquate pour répondre à cette problématique, en organisant les Dépenses par Budget avec un montant maximal pour chaque Budget. Cela permettrait de pouvoir suivre les secteurs de Dépenses au cours du temps pour toujours savoir où va notre argent.

Vous pouvez accéder au projet ici.

II/ Etat de l'art :

Il exist à l'heure actuelle un certain nombre d'outils pour suivre ses dépenses, sous forme d'**application mobile et web**.





On peut en retirer quelques **fonctionnalités** communes comme :

- **L'utilisation de Budget:**
  - Soit représentant le **Budget mensuel total** .
  - Ou un **secteur / type de dépenses** avec un certain montant qui est renseigné par l'utilisateur (alimentation, loisirs, ...).

- La possibilité d'**ajouter à ces Budgets des Dépenses** caractérisées par un type et un montant.
  - La possibilité de définir une date de répétition d'une Dépense pouvant être utile pour un paiement mensuel d'un loyer ou de toute autre charge fixe arrivant de manière Récurrente.

- Un **affichage donnant une vue d'ensemble des dépenses** réalisées, permettant une compréhension rapide de l'état de notre budget mensuel.

- **Mise en parallèle des dépenses actuelles avec celles des mois précédents**, aidant à voir l'évolution des finances de l'utilisateur.

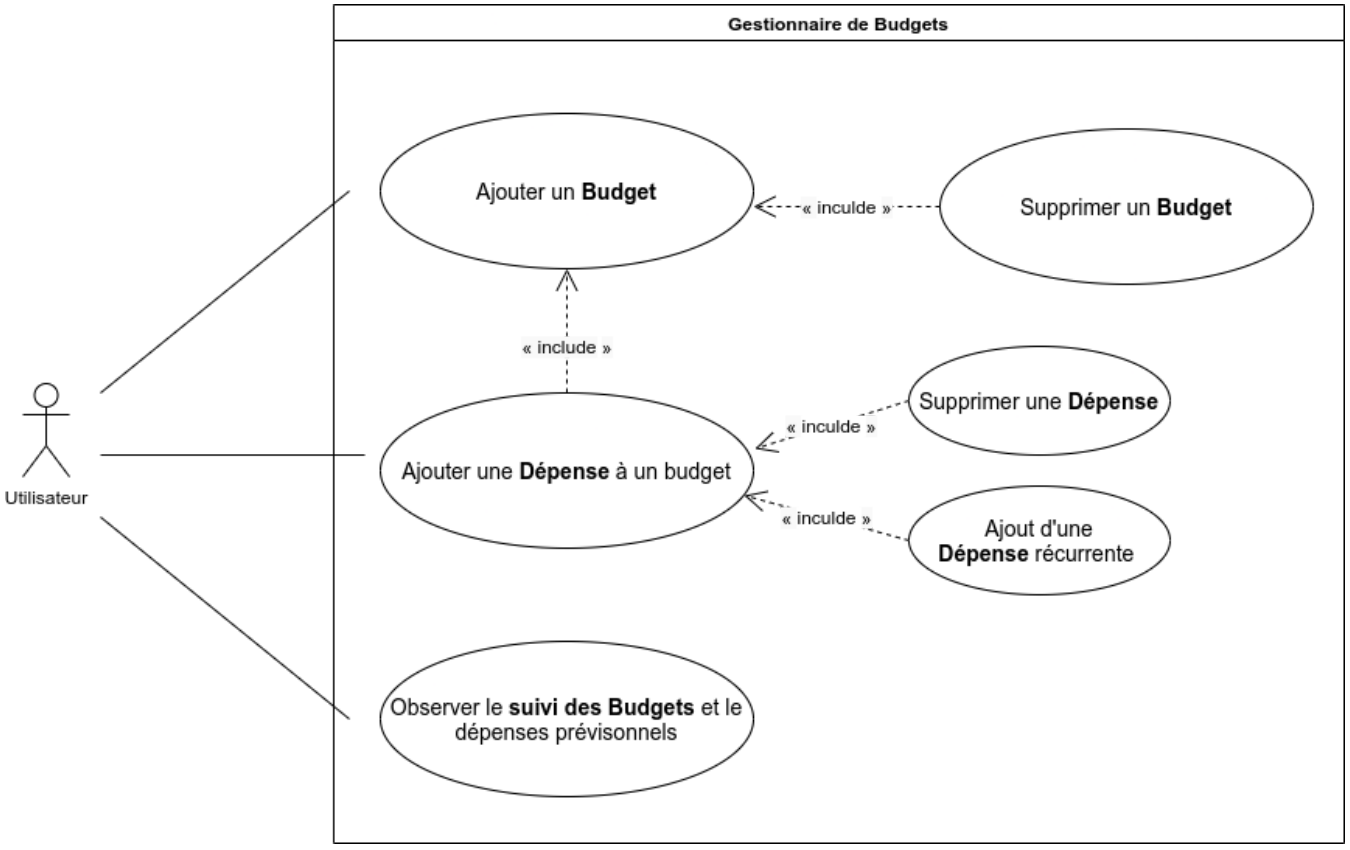
III/ Fonctionnalités :

Après étude des différents outils existant **une sélection de fonctionnalités indispensables pour tous gestionnaires de budget ressort** ainsi que d'autres plus spécifique à la problématique présentée précédemment.

- **Création de Budget** avec un type et un montant
- **Ajout de Dépenses** avec un type, un montant et une date (automatique)
- **Ajout de Dépenses Récurrentes** mensuelles avec un type, un montant.
- Différentes i**nterfaces de Visualisation et de Suivi** :
    - Progression de chaque budget
    - Progression du budget total
    - Graphique détaillant pour chaque mois les dépenses pour chaque budget
    - Prévision des dépenses récurrentes pour les mois à venir

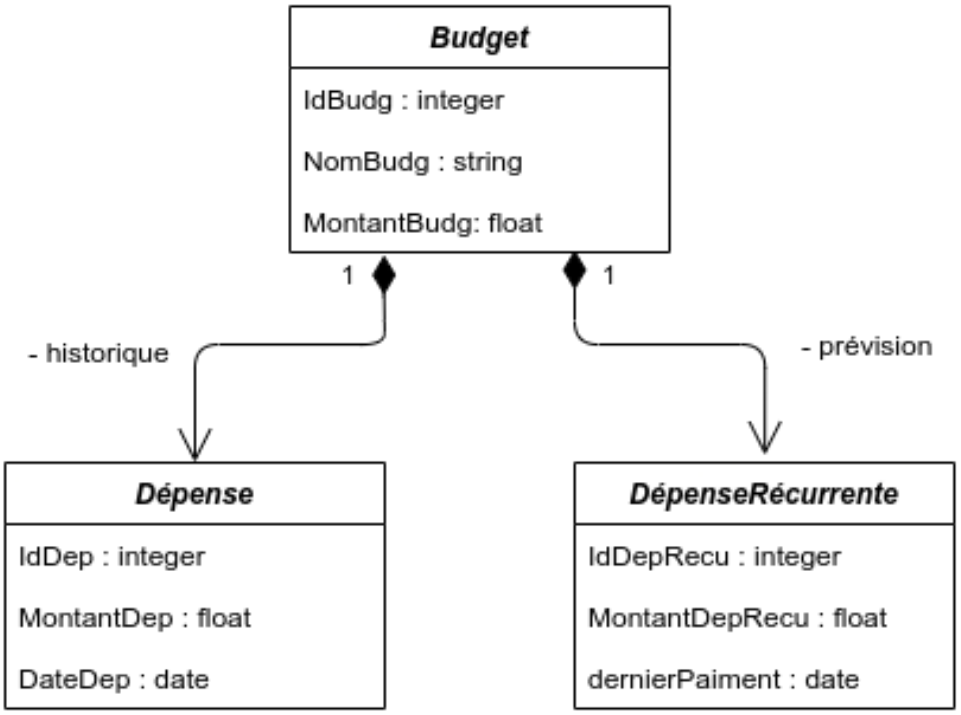
1/ Cas d'utilisations :

On peut alors imaginer des Cas d'Utilisations correspondant à ces fonctionnalités qui pourraient être pertinent au projet. En ajoutant la possibilité de supprimer des Budgets et des Dépenses par exemple.



2/ Structure de base de données :

On peut alors déterminer une structure de base de données permettant de réaliser les différentes fonctionnalités.



Budgets(**idBudg**, montantBudg, nomBudg)
Depenses(**idDep**, montantDep, **#idBudg,** dateDep)
DepensesRecurrentes(**idDepRecu**, montantDepRecu, **#idBudg,** dernierPaiement)

Nous avons donc trois tables dans notre base : **Budget**, **Depense**, et **DepenseRecurrentes**, on peut remarquer que les deux tables des dépenses ont une clé étrangère correspondant à l'id du Budget qui leur est associé.

```
sqlite> select * from DEPENSES;
3|10|2|1595596818
5|10|2|1595596825
11|25|4|1595597250
12|100|4|1595597404
13|75|5|1595597413
14|300|6|1595597961
```

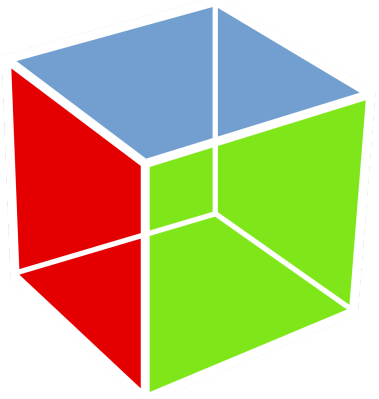
```
sqlite> select * from BUDGETS ;
1|250|Alimentation
2|100|Loisirs
3|250|Loyer
4|50|Economies Vacances
5|50|Transport
6|400|Dépenses Vacances
```

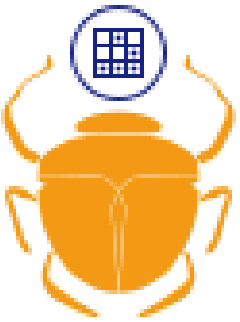
```
sqlite> select * from DEPENSESRECURRENTE;
1|100|1|07
2|250|2|07
3|25|3|07
```

3/ Outils et Technologies utilisés :

La totalité du développement de ce logiciel à été effectué en entièreté sous **Linux.** La contrainte du système d'exploitation et des différentes fonctionnalités envisagées a orienté le choix de la bibliothèque dédiée à l'interface graphique. Il en est sorti que **GTK** serait le meilleur choix, GTK n'est pas seulement une bibliothèque mais un ensemble de bibliothèques pour faire une interface, comme par exemple **Cairo** une bibliothèque permettant de faire des dessins pour créer des graphiques par exemple.

De plus GTK donne la possibilité de générer une interface via un fichier XML ce qui facilite sa création et l'arrangement des éléments graphiques.

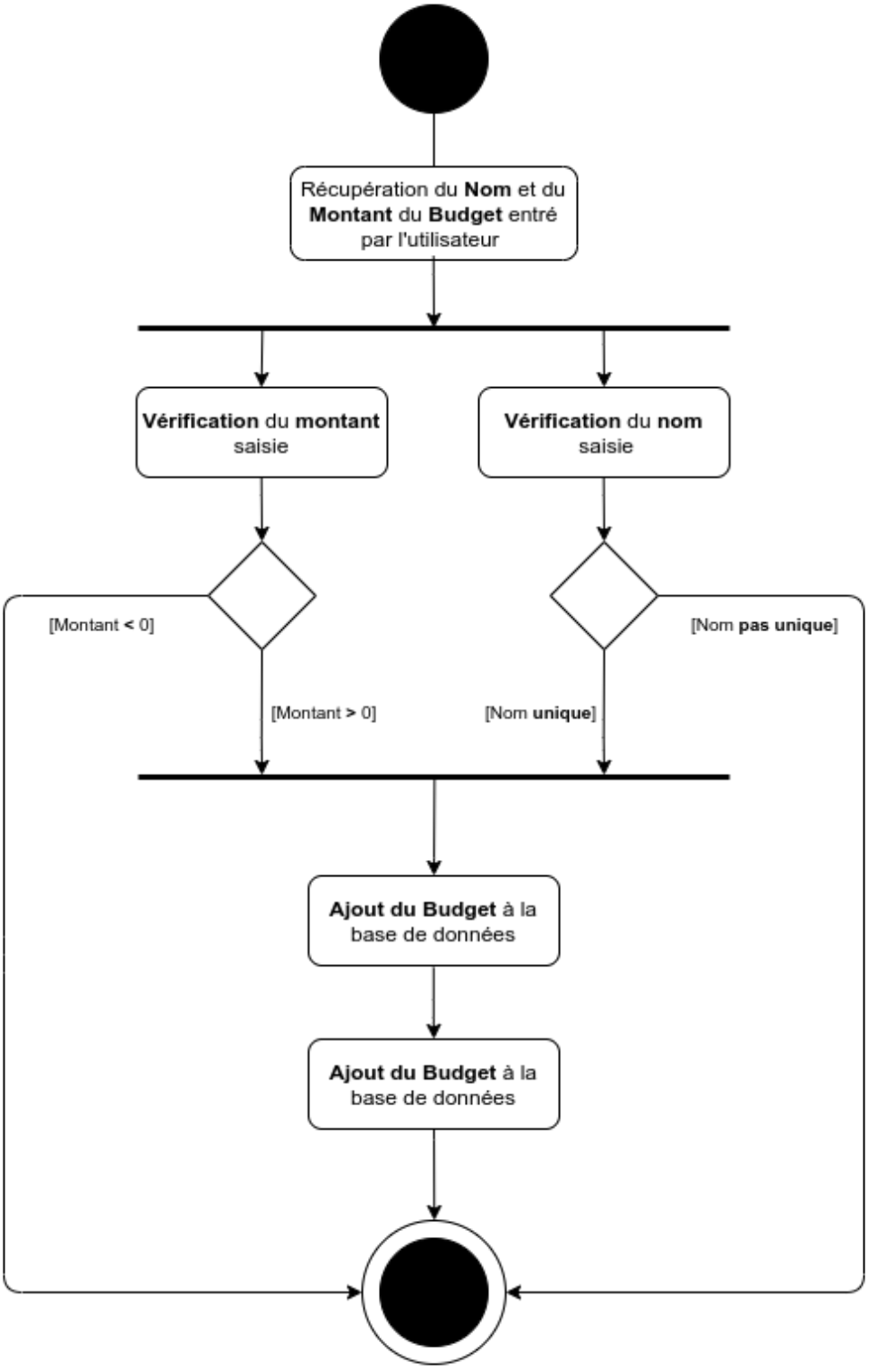




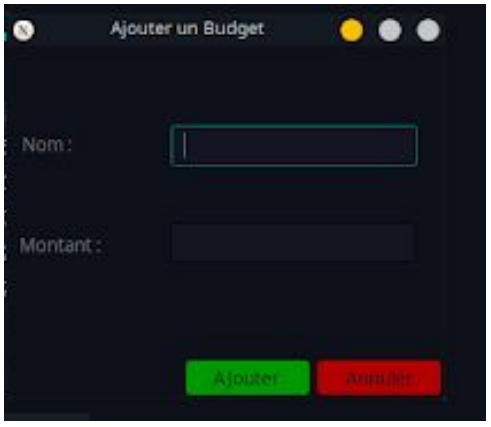
Pour le choix de la bibliothèque chargée de la **base de données** j'ai opté pour **SQLite** qui est une solution légère et simple d'utilisation. Permettant de créer une base de données facilement sans serveur, elle est simplement stockée dans un fichier texte.

4/ Description des Fonctionnalitées :

- a - Création de Budget :



L'ajout d'un Budget commence une fois que l'utilisateur clique sur le bouton d'ajout d'un Budget, ou une fenêtre de saisie s'ouvre pour pouvoir rentrer le nom et le montant voulu pour le nouveau Budget. La fonction d'insertion s'exécute une fois le formulaire validé.
En cas d'annulation la fenêtre se ferme et l'ajout est annulé.



Une fois le Budget validé par l'utilisateur, la fonction "**budgUnique**" vérifie si le nom d'un budget est unique, cela permet de ne pas avoir plusieurs budgets avec le même nom.
Cette fonction retourne 0 si elle trouve un budget dans la base avec le même nom et -1 si elle ne trouve pas un Budget existant avec le même nom dans la base de données.

Fonction **budgUnique**

```c
int budgUnique(char *type){
 sqlite3_stmt *stmt;
 char request[60] = "select count(*) from budgets where typeBudg = '";
 strcat(request, type);
 strcat(request, "'");

 if (sqlite3_prepare_v2(db, request, -1, &stmt, NULL)) {
   printf("ERROR TO SELECT DATA : getBudget\n");
   exit(-1);
 }
 sqlite3_step(stmt);
 return sqlite3_column_int(stmt, 0) - 1;
}
```

Nous utilisons donc pour **i**nsérer un nouveau budget dans la base de données la bibliothèque SQLite mise à disposition pour C.
Après avoir créé la requête avec le nom et le montant saisies par l'utilisateur nous utilisons la fonction "**sqlite3_exec()**" qui permet d'exécuter une requête SQL en un ligne.

Insertion d'un **Budget**

```c
char *request = "insert into BUDGETS (montantBudg,typeBudg)values(";
replacechar(montant, ',', '.');
strcat(request, montant);
strcat(request,", '");
strcat(request, nom);
strcat(request,"')"); // Creation de la requete
if(sqlite3_exec(db, request, NULL, NULL, NULL)){ // Insertion
    printf("ERROR IN INSERTION : BUDGET\n"); // Erreur si insertion impossible
}else{
    printf("INSERT BUDGET : %s : %s\n", montant, nom);
    vueBudgets();//Mise à jour de la vue des Budgets
}
```

Une fois le Budget inséré dans la base on actualise l'affichage avec la fonction "**vueBudget**" pour le rendre visible sur l'interface.

- b - Ajout d'une Dépense à un Budget :

L'ajout d'une Dépense à une Budget est en grande partie identique à l'ajout d'un Budget excepté le fait que l'utilisateur doit sélectionner un Budget associé à sa Dépense dans la liste de ceux qu'il a créés.

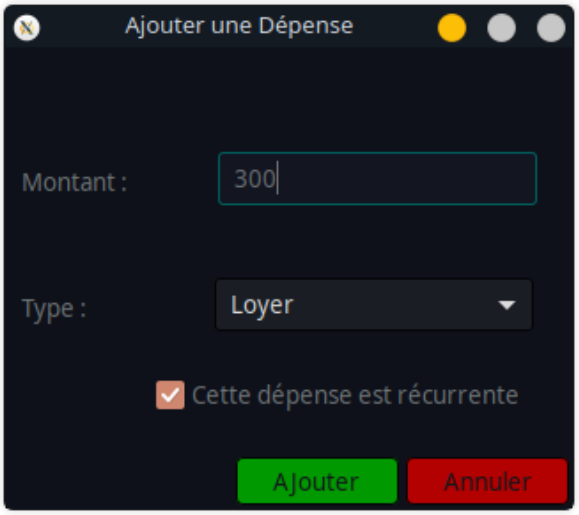
Comme pour le Budget une fois les vérifications de la validité du montant (> 0) et de l'existence du Budget sélectionné, une requête SQL est générée et la nouvelle dépense est insérée dans la base de données. On actualise alors les vues des Budgets, pour afficher le nouveau montant restant, et les Dépenses pour afficher la Dépense ajoutée à l'instant.

Vérification et Insertion d'une **Dépense**

```
//Si le montant est > 0 et type != null
 if (dep.montant > 0 && strcmp(dep.type, "(null)") != 0) {
   char request[80] = "INSERT into DEPENSES (montantDep, idTYpe) values(";
   replacechar(m, ',', '.');
   strcat(request, m);
   strcat(request,", '");
   snprintf(buffId, sizeof(buffId), "%d", idB);
   strcat(request, buffId);
   strcat(request,"')");
   printf("REQUEST: %s", request);
   if(sqlite3_exec(db, request, NULL, NULL, NULL)){
     printf("ERROR IN INSERTION : DEPENSE\n"); // Erreur d'insertion
   }else{
     printf("INSERT : DEPENSE\n");
     vueBudgets(); // actualisation de la vue des Budgets
     vueDepenses(); // actualisation de la vue des Dépenses
   }
 }
```

- c - Ajout d'une Dépense Récurrente à un Budget :

Une Dépense Récurrente se répétera chaque mois et ajoutera une dépense classique d'un montant donné au budget auquelle elle est associée. Lors de la création d'une nouvelle dépense l'utilisateur peut sélectionner une "**CheckBox**" pour que la dépense soit récurrente.



La distinction entre Dépense classique et Dépense Récurrente s'effectue après la validation du formulaire grâce à la fonction "**gtk_toggle_button_get_active**" qui retourne TRUE si l'élément "**CheckBox**" est sélectionné.

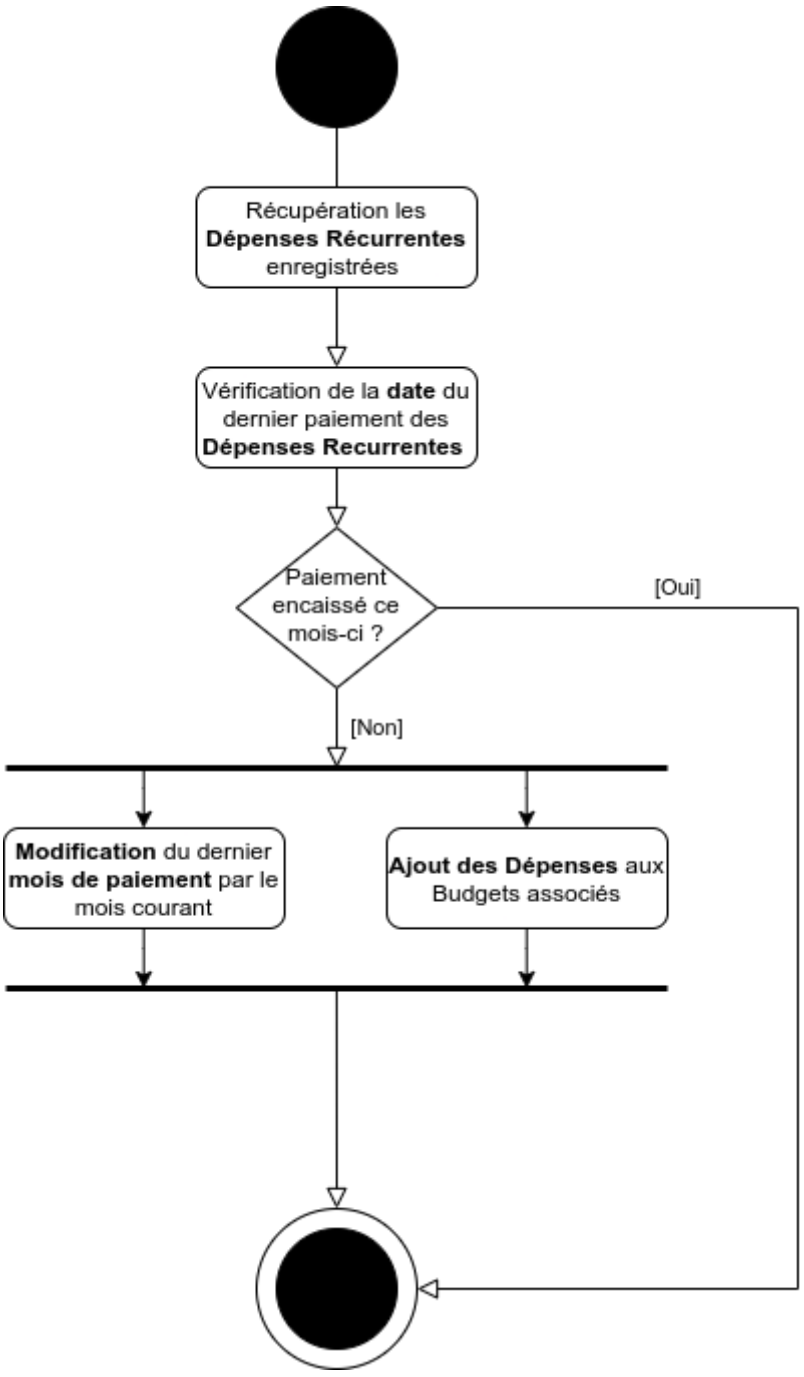
Nous pouvons donc appeler la fonction correspondant au type de Dépense soir "**insertDepense**" ou "**insertDepenseRecu**" en leur passant en paramètre le montant entré par l'utilisateur ainsi que l'id associé au Budget sélectionné.

Dépense classique ou **Récurrente**

```
if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(check_box_recu))){
   printf("DEPENSE RECURRENTE\n");
  // Insertion d'une Dépense Récurrente avec son montant et l'id du Budget
   insertDepenseRecu(m, (int)sqlite3_column_int(stmt, 0));
 }else{
   printf("DEPENSE\n");
  // Insertion d'une Dépense classique avec son montant et l'id du Budget
   insertDepense(m, (int)sqlite3_column_int(stmt, 0));
 }
```

- d- Paiement des Dépenses Récurrentes :

Comment expliqué précédemment les Dépenses Récurrentes sont payées tous les mois.
À chaque lancement du logiciel une vérification est effectuée pour voir si toutes les Dépenses Récurrentes enregistrées ont été encaissées ce mois si. Si cela n'a pas été fait alors la Dépense est encaissée et le mois du dernier paiement de la Dépense est actualisé.

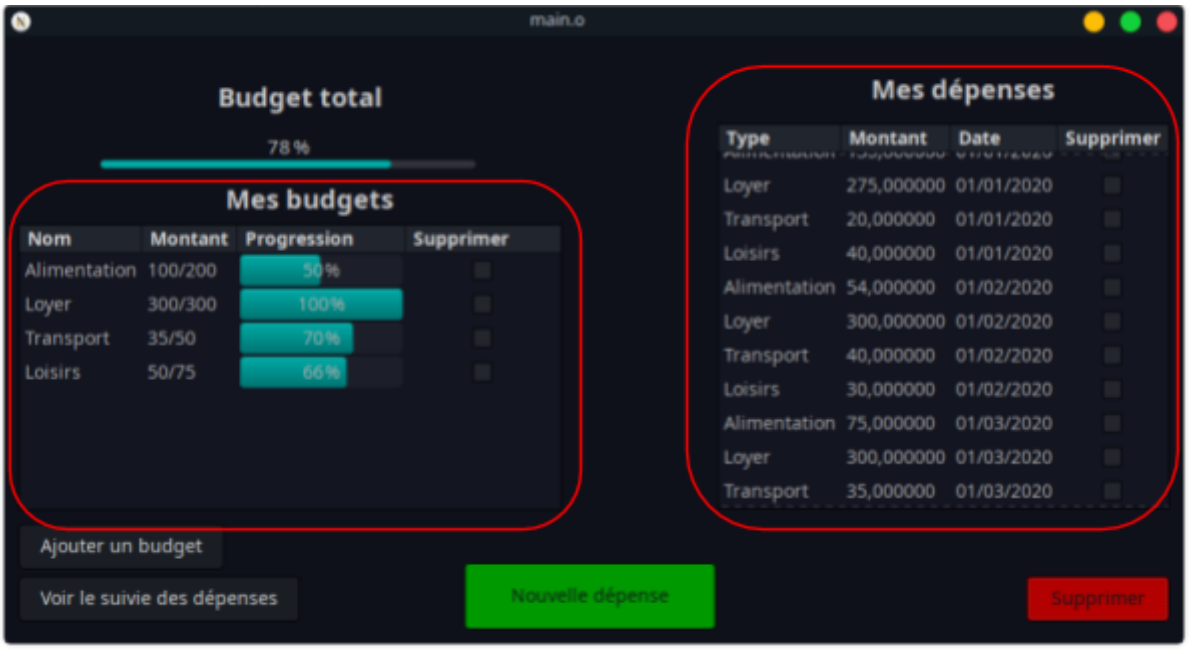


- e - Affichage des Budget et de l'historique des Dépenses :

L'affichage de la liste des Budgets créés et de l'historique des Dépenses se fait grâce à un élément graphique appelé "**TreeView**" qui est majoritairement utilisé avec une structure de données appelée "**TreeModel**". Elle permet alors d'afficher des arborescences d'éléments comme par exemple des fichiers et des dossiers.

Dans notre cas nous l'utiliserons avec une structure appelée "**ListStore**" puisque nous n'avons pas besoin d'afficher d'arborescence mais simplement des listes. Cet élément graphique rend l'affichage de liste plus simple et plus ergonomique en effet un certain nombre de fonctions y sont associées pour faciliter l'ajout de ligne, leur lecture et leur suppression par exemple.

Nous avons donc deux "**TreeView**" associées chaqu'unes à une "**ListStore**", une pour les Budgets et l'autre pour les Dépenses.



Ces deux éléments sont créés à partir d'un fichier XML décrivant à la fois les "**TreeView**" et les "**ListStore**". Ce fichier XML décrit plus généralement l'intégralité de l'interface du logiciel mais nous aborderons ce sujet dans une autre partie.

La gestion de ces "**TreeView**" dans le programme est fait grâce à deux fonctions : "**vueBudget**" et "**vueDepense**". Ces deux fonctions permettent à chaque Budget et à chaque Dépense de la base de données d'être affiché sur leur "**TreeView**" respectif.

Les fonctionnalités de cet élément nous permettent d'ajouter dans les lignes d'autres éléments graphiques comment des barres de progressions montrant le pourcentage de dépense de chaque Budget et des "**CheckBox**" pour choisir les lignes que l'on pourrait vouloir supprimer.

Dépense classique ou **Récurrente**

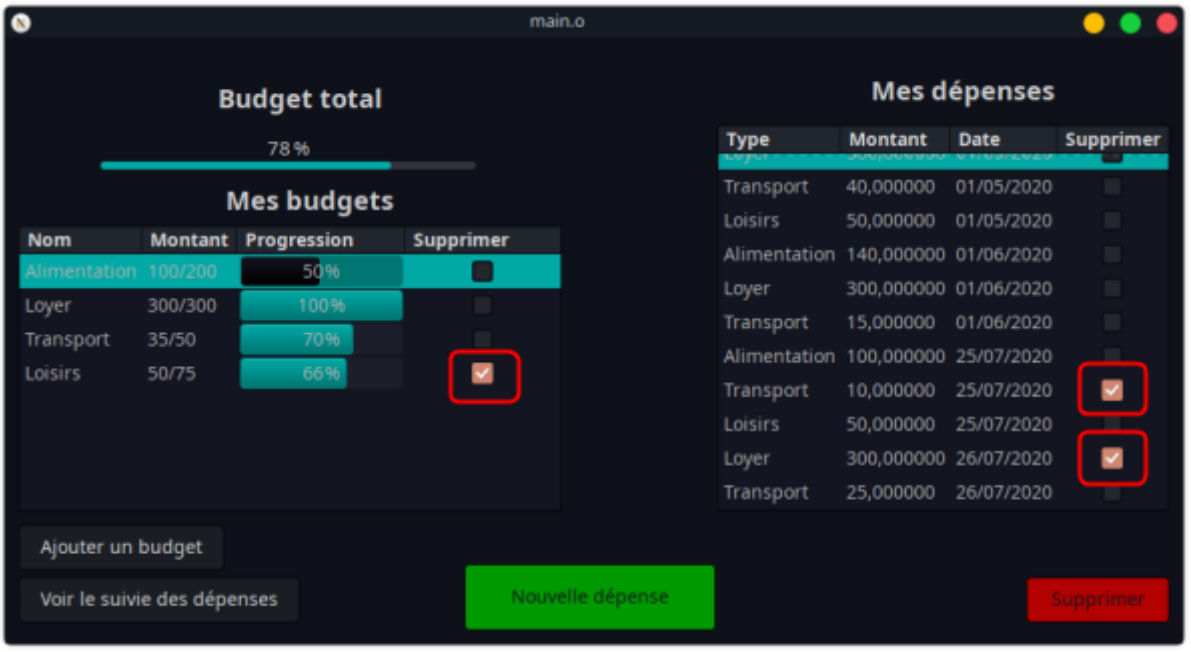
```
void vueDepenses(){
    [ … ]
    char request[80] = "select * from DEPENSES INNER JOIN BUDGETS on idType = idBudg
    order by dateDep";

    gtk_list_store_clear(list_store_dep);// Suppression du contenu de la ListStore

    while (sqlite3_step(stmt) == SQLITE_ROW) {
                [ … ]
        gtk_list_store_append(list_store_dep, &iter);
        // Création d'une nouvelle ligne dans la ListStore "list_store_dep"

        gtk_list_store_set(list_store_dep, &iter, 0, type, 1, montant, 2, date, 3,
        FALSE, 4, id, -1); // Ajout dans la nouvelle ligne
    }
}
```

À chaque appel des fonctions "**vueDepense**" ou "**vueBudget**" le contenu des deux "**ListStore**" est supprimé puis réajouté à la liste en parcourant les Tables Dépenses et Budgets. Ces deux fonctions sont appelées au démarrage du logiciel puis à chaque fois qu'une modification (ajout ou suppression) est faite à une des deux tables Dépenses ou Budget

Pour la fonction "**vueBudget**" les seules différences sont la récupération du montant total des Dépense pour chacun des Budget avec la fonction "**getDepensesSumByType"**, ainsi que l'appel à la fonction "**gtk_progress_bar_set_fraction**" qui permet de changer le niveau d'avancement des barres de progressions.

- f - Suppression de Budget et Dépense :

Comme vu précédemment il est possible de supprimer des Budgets ou des Dépenses en sélectionnant le “**CheckBox**” associé à ligne.



Une fois les lignes voulues sélectionnées l’utilisateur doit confirmer la suppression en cliquant sur le bouton “**Supprimer**”.

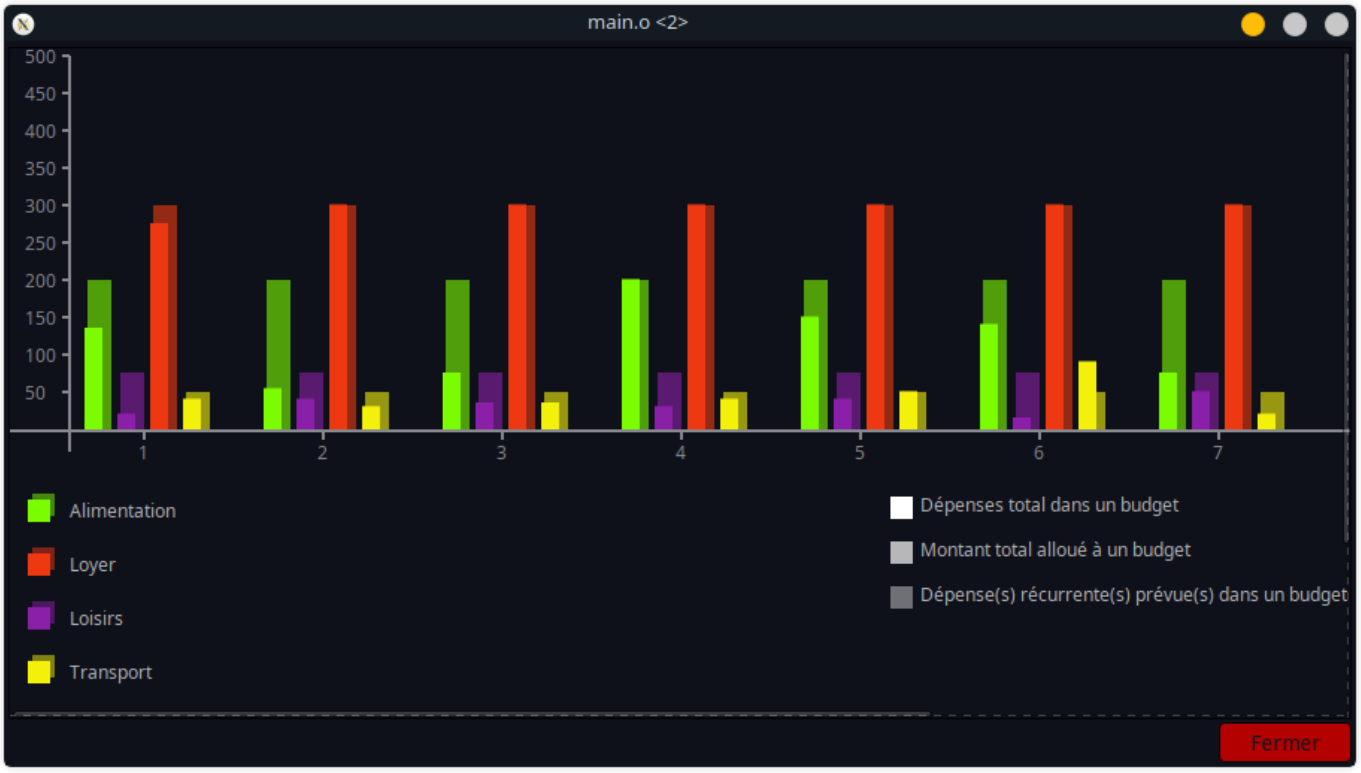
À chaque fois que l’utilisateur clique sur le bouton **Supprimer,** les fonctions “**supprRowBudg**” et “**supprRowDep**” parcourent les lignes de Budget et de Dépense et pour chaque ligne sélectionnée les fonctions “**deleteBudg**” ou “**deleteDep”** sont appelées pour supprimer le Budget ou la Dépense correspondant dans la base de données.

En sachant que, lorsqu’un Budget est supprimé toutes les Dépenses classiques et Récurrentes associées le sont aussi pour éviter tout problème.

Une fois tous les éléments supprimés de la base on peut actualiser l’affichage des Budgets et des Dépenses grâce aux fonctions “**vueBudg**” et “**vueDep**”.

- g - Suivi des Dépenses et des Budget:

Comme mentionné en introduction, le but de ce logiciel est de fournir un suivi des Dépenses en fonction de leur type pour savoir où nous dépensons notre argent. L'implémentation d'un graphique pour suivre l'évolution de nos Dépenses semble donc parfaitement adapté pour répondre à cette problématique.

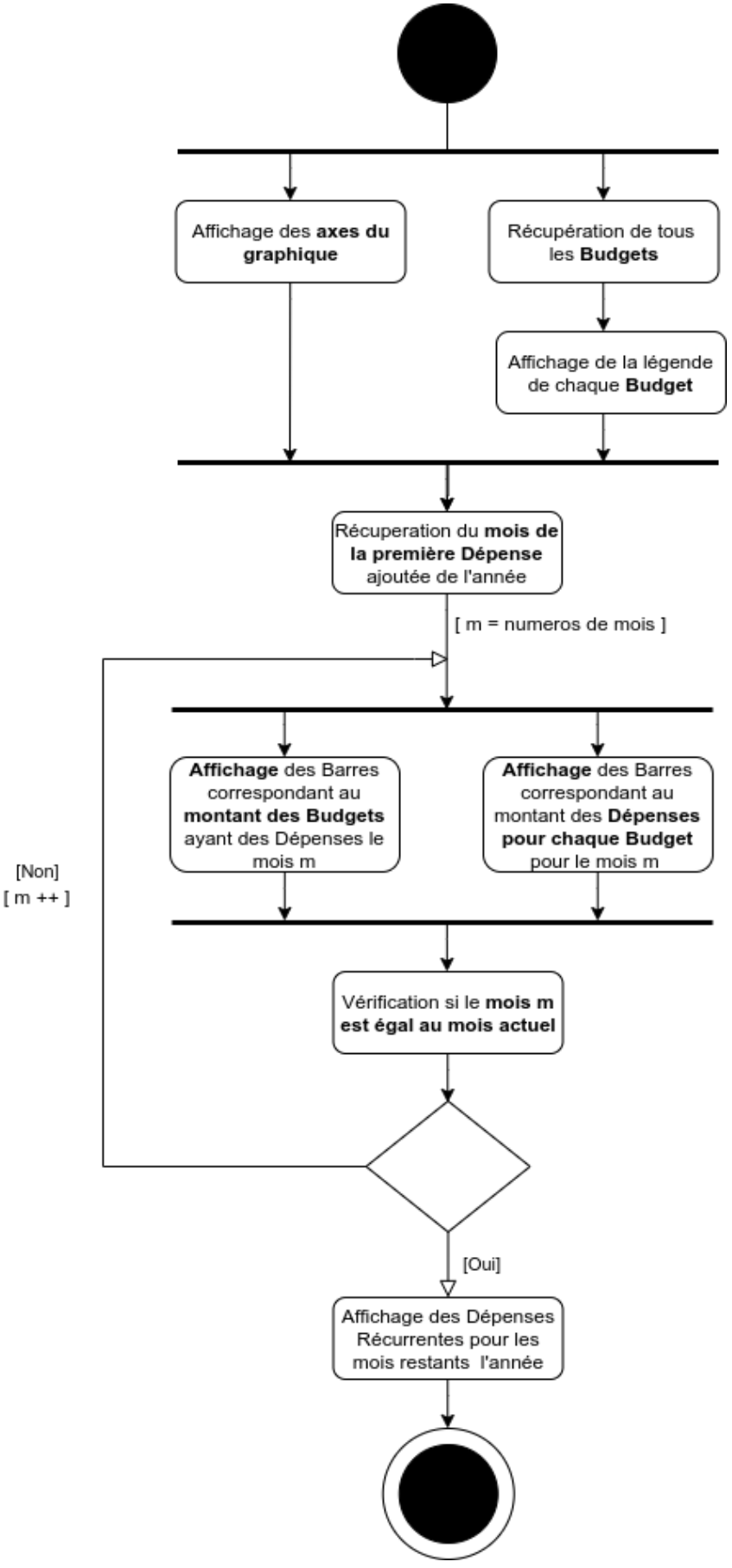


Comment on peut le voir le graphique de suivi est composé de plusieurs éléments.
Chaque mois est représenté par une graduation sur l'axe des abscisses, on retrouve les Budgets présentés sous forme de barre de couleur.

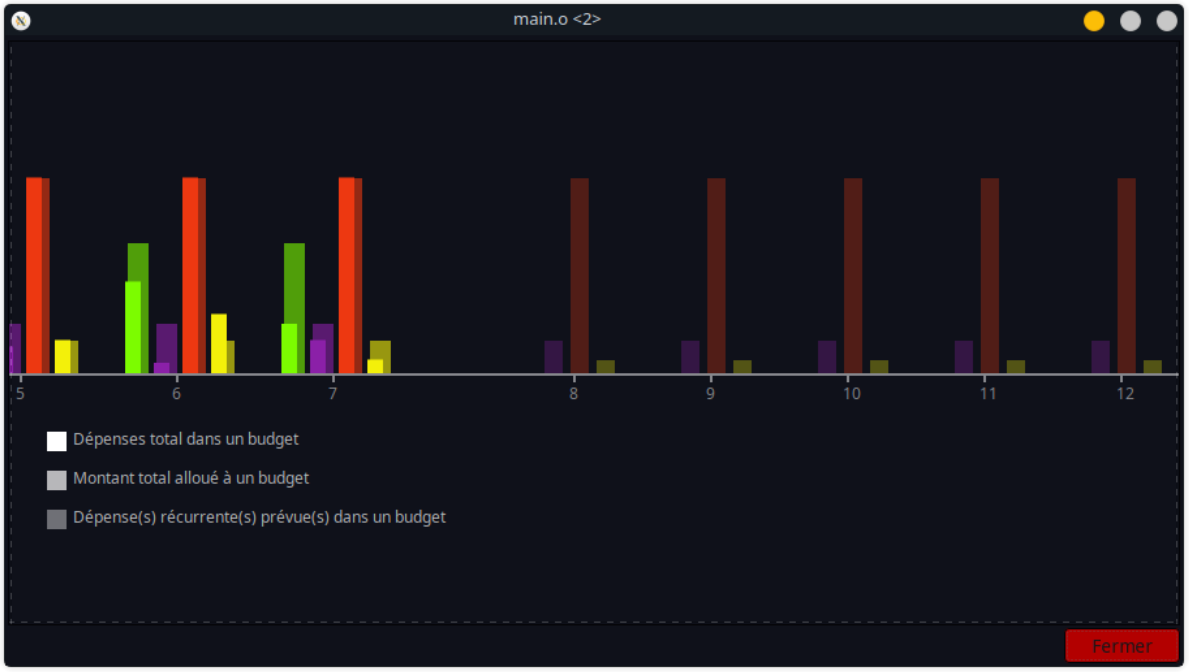
La barre au premier plan de couleur plus foncé représente le montant des Dépenses pour le mois dans le Budget et la barre en arrière plan le montant total associé au Budget. Cela permet de savoir s' il nous reste encore de l'argent dans ce Budget.

Le graphique commence avec le premier mois de l'année ayant des Dépenses enregistrées, ce qui permet d'avoir un suivi des Dépenses sur l'année en cours.

Ce **Diagramme d'Activité** décrit la création de l'affichage du suivi des Dépenses:



L’autre fonctionnalité de ce graphique est d’afficher une prévision des Dépenses à venir. Ces Dépenses à venir sont les Dépenses Récurrentes enregistrées par l'utilisateur. Elles sont payées chaque mois et peuvent donc être prévues. Elles sont affichées avec une couleur correspondant à leur Budget associé mais en plus claire pour pouvoir les différencier.

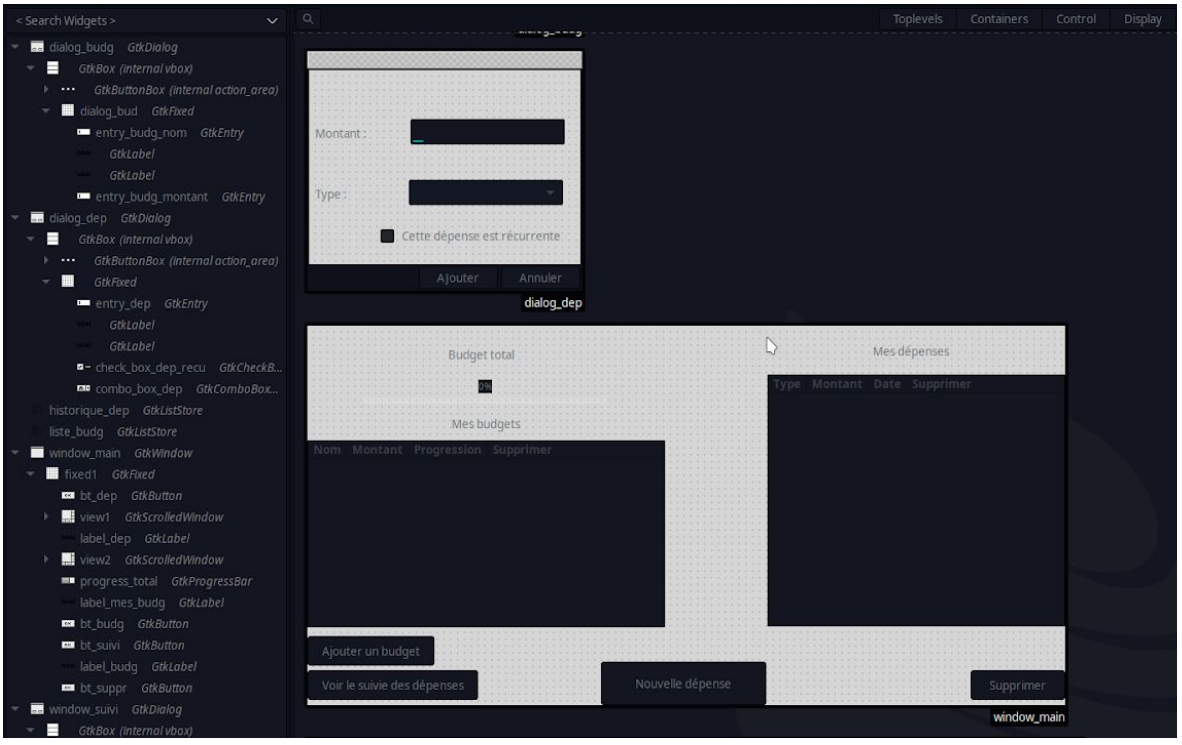


IV/ Problèmes Rencontrés :

- **La première difficulté de ce projet fut le langage de programmation choisi : "C"**. En effet bien qu'ayant déjà utilisé ce langage pendant la première année d'IUT ce n'était que pour faire des programmes basiques pour apprendre les bases du langage. Avant de commencer ce projet j'ai dû faire de nombreuse recherches, relire mes anciens cours et faire de nombreux tests pour me remémorer le fonctionnement de C, comme avec les pointeurs et le fonctionnement de programmes avec plusieurs fichiers.

  De plus je n'avais jamais fait de programme en C utilisant une interface graphique ou une base de données. J'ai dû apprendre tout cela par moi même en lisant beaucoup de documentation et en regardant un certain nombre de tutoriel en ligne pour apprendre ces nouvelles choses.

- Un des autre problème rencontré fut la **difficulté de créer une interface ergonomique**. Bien que grâce à GTK nous pouvons générer une interface avec un fichier XML cela reste compliqué et difficile d'ajouter et personnaliser des éléments graphiques.
  Après des recherches, il se trouve qu'il existe un logiciel de création d'interface appelé Glade, il permet de générer un fichier XML à partir d'une interface créée. Cela rend le processus de création et de modification de l'interface plus simple et plus rapide.



- La partie la plus complexe de ce projet fut sûrement l**a création du graphique de suivi des Dépenses**. En effet j'ai du faire de nombreuses recherches pour choisir la meilleure solution. Après avoir lu la documentation de GTK au sujet de l'élément graphique "**GtkdrawingArea**" qui permet de créer des éléments graphiques personnalisés j'ai choisi cet élément pour contenir le graphique de suivi de dépense. Grâce à la bibliothèque Cairo intégré à GTK nous pouvons dessiner ce que nous voulons dans cet élément "**GtkDrawingArea**"

  Après avoir trouvé cette technique pour créer le graphique il faut le dessiner. Cela est long à faire, vu le nombre d'éléments à dessiner. Il faut pour chaque mois dessiner toutes les barres, soit deux par budget, une pour le montant du budget et une pour le montant des dépenses dans ce Budget, ainsi que les axes et leur graduation et les montants des Dépenses à venir.

Affichage des barres de **Budget / Dépenses**

```
gdk_cairo_set_source_rgba (cr, &color);
cairo_fill (cr);
cairo_rectangle (cr, (nbBudg - 1) * 22 + 52 + (moisCourant * 40) +
(nbBudgAfficherTotal * 20), 255, 16,  -((budgMontant + 1) / 2));
gdk_rgba_parse (&color, colorsTab[sqlite3_column_int(stmt2 , 1) - 1]);
color.alpha = 0.6;
gdk_cairo_set_source_rgba (cr, &color);
cairo_fill (cr);
```

Le plus difficile fut que pour chaque mois il faut prendre en compte le nombre de mois précédent avec le nombre de Budget précédemment affiché pour ne pas avoir d'erreur d'affichage comme la superposition de plusieurs barres de Budget.
C'est grâce au trois variables, nbBudg, moisCourant et nbBudgAfficherTotal que l'on peut éviter la superposition des barres et les affichers à la bonne position.

La première variable nbBudg contient le nombre de Budget déjà affiché ce mois si, pour permettre de séparer les barres de budget d'un même mois de 22 pixels pour plus de visibilité.

La seconde variable moisCourant à la valeur du nombre de mois déjà affiché, pour permettre elle de séparer chaque Mois les un des autre de 40 pixels.

La dernière variable nbBudgAfficherTotal et celle qui fut le plus difficile à définir. Elle contient le nombre de Budget total déjà affiché sur le graphique, ce qui est essentiel pour avoir un bon affiche sans superposition.

V/ Conclusions :

Pour conclure la problématique annoncée au début à été en grande partie répondue par le logiciel et ses fonctionnalités.

Nous avons un logiciel capable d'**enregistrer des Budgets** créés par l'utilisateur pour tout type de Dépense qu'il souhaite enregistrer. Il peut y **ajouter des Dépenses** et ainsi voir combien il dépense dans chaque secteur sur le mois. De plus, le **graphique de suivi des Budgets** permet d'avoir une vue plus globale sur ses Dépenses au cours du temps et ainsi savoir où il débourse son argent. Pour finir les Dépenses Récurrentes permettent d'avoir un logiciel plus intéressant permettant de prévoir les futures dépenses qui reviennent tous les mois et de les anticiper.

Néanmoins je pense que **d'autres fonctionnalités** peuvent encors être ajoutées :

- Comme la possibilité de rentrer une date de paiement d'une Dépense. Cela permettrait d'ajouter des Dépenses à venir qui ne serait comptabilisées qu'
- à la date donnée ou de rajouter des Dépenses passées à la bonne date.
- L'ajout d'un graphique supplémentaire à la place de la barre de progression du budget total, affichant la part de chaque budget dans le budget total avec leur montant de Dépense à chaqu'un pourrait être un plus pour un meilleur affichage.

Vous pouvez accéder au projet ici.

Remerciements :

Pour finir je souhaiterais remercier Mme.Illina, M.Mangeol, ainsi que toutes les personnes qui m'ont permis de réaliser ce travail de remplacement et qui m'ont aidé au cours de ce projet pour le mener à bien.